

THE VIABILITY AND POTENTIAL CONSEQUENCES OF IOT-BASED RANSOMWARE

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF PHD

By
Calvin Brierley
April 2022

All work, figures, and images featured within this thesis were created or photographed by myself, unless otherwise stated or cited.

Abstract

With the increased threat of ransomware and the substantial growth of the Internet of Things (IoT) market, there is significant motivation for attackers to carry out IoT-based ransomware campaigns. In this thesis, the viability of such malware is tested.

As part of this work, various techniques that could be used by ransomware developers to attack commercial IoT devices were explored. First, methods that attackers could use to communicate with the victim were examined, such that a ransom note was able to be reliably sent to a victim. Next, the viability of using “bricking” as a method of ransom was evaluated, such that devices could be remotely disabled unless the victim makes a payment to the attacker. Research was then performed to ascertain whether it was possible to remotely gain persistence on IoT devices, which would improve the efficacy of existing ransomware methods, and provide opportunities for more advanced ransomware to be created. Finally, after successfully identifying a number of persistence techniques, the viability of privacy-invasion based ransomware was analysed.

For each assessed technique, proofs of concept were developed. A range of devices – with various intended purposes, such as routers, cameras and phones – were used to test the viability of these proofs of concept. To test communication hijacking, devices’ “channels of communication” – such as web services and embedded screens – were identified, then hijacked to display custom ransom notes. During the analysis of bricking-based ransomware, a working proof of concept was created, which was then able to remotely brick five IoT devices. After analysing the storage design of an assortment of IoT devices, six different persistence techniques were identified, which were then successfully tested on four devices, such that malicious filesystem modifications would be retained after the device was

rebooted. When researching privacy-invasion based ransomware, several methods were created to extract information from data sources that can be commonly found on IoT devices, such as nearby WiFi signals, images from cameras, or audio from microphones. These were successfully implemented in a test environment such that ransomable data could be extracted, processed, and stored for later use to blackmail the victim.

Overall, IoT-based ransomware has not only been shown to be viable but also highly damaging to both IoT devices and their users. While the use of IoT-ransomware is still very uncommon “in the wild”, the techniques demonstrated within this work highlight an urgent need to improve the security of IoT devices to avoid the risk of IoT-based ransomware causing havoc in our society. Finally, during the development of these proofs of concept, a number of potential countermeasures were identified, which can be used to limit the effectiveness of the attacking techniques discovered in this PhD research.

Acknowledgements

I would like to thank my supervisors Budi Arief, Julio Hernandez-Castro, and David Barnes for providing me with ample advice, direction, and feedback throughout my PhD. All have supplied me with generous insight into the world of security research and have helped me grow as an academic.

I would also like to thank my parents for inspiring me to push for further education and supporting me through difficult times. As promised, I have now cited you in an academic work¹.

A thank you to Jamie and Matt, who made my experience at Kent much more enjoyable, and a big thank you to Caio, who provided plentiful support during the writing of this thesis.

A quick shout out to the various teammates of security *capture the flag competitions* that I took part in during this PhD, who led me to push harder at learning new security tricks that I would not have considered otherwise.

Finally, I would like to thank the University of Kent, at which I have studied for 8 amazing years, and the School of Computing, which provided resources, support, and IoT devices for this research.

¹“Val was here.” - Valerie Brierley, 2022 [45]
“What?” - Tony Brierley, 2022 [44]

Contents

Abstract	iii
Acknowledgements	v
Contents	vi
List of Figures	xiii
List of Tables	xv
List of Listings	xvi
List of Publications	xvii
1 Introduction	5
1.1 Research Questions and Objectives	5
1.2 Research Contributions	6
1.3 Thesis Outline	7
2 Literature Review	9
2.1 Introduction	9
2.2 The Internet of Things	9
2.2.1 IoT Device Design and Use	10
2.2.2 IoT Security	11
2.2.3 IoT Malware	13
2.3 Ransomware	16
2.3.1 Types of Ransomware	16
2.3.2 Post-Infection	18

2.3.3	Large Scale Attacks	20
2.4	IoT-Based Ransomware	23
2.4.1	Targets	23
2.5	Conclusions	30
3	Methodology	31
3.1	Introduction	31
3.2	Research Challenges	32
3.2.1	IoT Device Variance	32
3.2.2	Availability of Malware Samples	32
3.2.3	Summary	33
3.3	Threat Model	33
3.3.1	Ransomware Targets	33
3.3.2	Attacker Capabilities	34
3.3.3	Attacker Goals	34
3.3.4	Research Scope	34
3.4	Research Plan	35
3.5	Overall Process	36
3.6	Conclusions	36
4	Challenges in Delivering IoT Ransom Notes	38
4.1	Introduction	38
4.2	Desktop-Based Ransom Notes	39
4.3	IoT-Based Ransom Notes	40
4.3.1	Communication Hijacking	41
4.3.2	Common Communication Channels	42
4.4	Proofs of Concept	45
4.4.1	Device Analysis	45
4.4.2	Malware Loader	47
4.4.3	Hijack Modules	50
4.4.4	Device investigation	52
4.5	Conclusions	62
5	Bricking Ransomware	64
5.1	Introduction	64

5.2	Locking Ransomware	66
5.2.1	Design	66
5.2.2	Limitations of Locking Ransomware	66
5.3	Related IoT Malware	68
5.3.1	Mirai	68
5.3.2	Brickerbot	69
5.3.3	Silex	70
5.4	Limitations of IoT Ransomware	70
5.4.1	Asset Value	71
5.4.2	Persistence	71
5.4.3	Device Variation	72
5.5	Known IoT Ransomware Implementations	73
5.5.1	Proofs of Concept	73
5.5.2	In the Wild	75
5.5.3	Limitations	78
5.5.4	Summary	79
5.6	PaperW8	80
5.6.1	Structure	81
5.6.2	Exploitation and Infection	81
5.6.3	Permanent Denial of Service	82
5.6.4	Communication Hijacking	84
5.6.5	Recovery	84
5.6.6	Categorising PaperW8	87
5.7	Tested Devices and Results	87
5.7.1	HG532 TalkTalk Router	88
5.7.2	R6250 Netgear router	89
5.7.3	TV-7104HE MVPower DVR	90
5.7.4	WiPG-1000 Presenter	91
5.7.5	5020L and 932L D-Link Cameras	92
5.7.6	Summary	93
5.8	Limitations	94
5.8.1	Device Cost and Ransom Pricing	94
5.8.2	Premature Rebooting	94
5.9	Conclusions	95

6	Persistence in Linux-Based IoT Malware	96
6.1	Introduction	96
6.2	Background	97
6.2.1	Non-Persistent Malware	97
6.2.2	Persistent IoT Malware	98
6.2.3	Challenges with Gaining Persistence	99
6.2.4	Previous Persistent IoT Malware	102
6.3	Filesystems	104
6.3.1	Utility Filesystems	104
6.3.2	Storage Filesystems	105
6.4	Device Enumeration	107
6.4.1	/proc/mtd	108
6.4.2	/proc/mounts	108
6.4.3	Binwalk	108
6.4.4	Startup Scripts	108
6.5	Methods of Gaining Persistence	109
6.5.1	Modifying Writable Filesystems	110
6.5.2	Recreating Read-Only Filesystems	111
6.5.3	Initrd and Initramfs Modification	113
6.5.4	“Set Writeable Flag” Kernel Module	114
6.5.5	Update Process Exploitation	116
6.5.6	Ubootkit	117
6.6	Experimental Proof of Concepts and Results	119
6.6.1	Netgear R6250 Router	119
6.6.2	D-Link DCS-932L	121
6.6.3	Yealink SIP-T38G	125
6.6.4	WiPG-1000	125
6.7	Conclusions	128
7	Privacy-Invasion Based IoT Ransomware	129
7.1	Introduction	129
7.2	Background	130
7.2.1	IoT Ransom Methods	130
7.2.2	Privacy Invasion	131

7.3	Data Sources	132
7.3.1	In-Built Sensors	132
7.3.2	Network Data	132
7.3.3	Local Storage and Configuration Settings	133
7.4	Identifying Private Data	133
7.4.1	Malicious Use of Machine Learning	134
7.4.2	Network-Based Privacy Invasion	136
7.4.3	Data Processing	140
7.5	Data Management	142
7.5.1	Ransom Note Generation	142
7.5.2	Publishing Private Information	143
7.5.3	Scale of operations	144
7.6	Proofs of Concept	144
7.6.1	Data Collation	144
7.6.2	Netgear R6250 Router	147
7.6.3	Yealink SIP-T38g Phone	149
7.6.4	DCS-932L Camera	153
7.6.5	Summary	155
7.7	Categorising Privacy-Based IoT Ransomware	156
7.7.1	Spyware	156
7.7.2	Extortion	157
7.8	Future Privacy-Based IoT Ransomware	158
7.8.1	Native Malicious Machine Learning	158
7.8.2	False Data	158
7.9	Conclusions	159
8	Countermeasures	160
8.1	Introduction	160
8.2	Desktop-Based Countermeasures	161
8.2.1	Best Practices	161
8.2.2	PayBreak	161
8.2.3	CryptoDrop	163
8.2.4	Unveil	164
8.2.5	ShieldFS	164

8.2.6	Redemption	165
8.2.7	Compatibility with IoT Devices	166
8.3	IoT-Based Countermeasures	166
8.3.1	General IoT Security	167
8.4	Preventing Communication Hijacking	171
8.5	Preventing Malicious Storage Manipulation	172
8.5.1	Effective Factory Reset Processes	172
8.5.2	Read-Only Partitions	173
8.5.3	Support for Direct Storage Access	174
8.5.4	Data Signing	175
8.6	Privacy Invasion Protection	175
8.6.1	Preventing Domain Extraction	176
8.6.2	Malicious Activity Detection in Cloud Services	177
8.6.3	Data Devaluation	177
8.7	Conclusions	178
9	Conclusions and Further Work	180
9.1	Introduction	180
9.2	Methodology Review	180
9.3	Objective Review	182
9.4	Contribution Review	184
9.5	Limitations	186
9.5.1	Lack of IoT Developer Perspective	186
9.5.2	Development of Countermeasure Tools	187
9.5.3	Commercial Device Focus	187
9.5.4	Alternative Operating Systems and Design Architectures	187
9.6	Further Work	189
9.6.1	Bricking-Based Ransomware	189
9.6.2	Persistence Techniques	190
9.6.3	Privacy-Invasion Based Ransomware	191
9.6.4	Countermeasures	193
9.7	Final Remarks	194
A	D-Link 932L Exploitation	195

B CVE Patches	197
Bibliography	198

List of Figures

3.1	Research Process Graph	37
4.1	WannaCry Ransom Note	39
4.2	HTTP Hijacking Ransom Note	43
4.3	HG532 Router	53
4.4	Netgear R6250 Router	54
4.5	TV-7104HE MVPower DVR	55
4.6	Hijacking the Framebuffer of the TV-7104HE MVPower DVR . .	58
4.7	WiPG-1000 Presenter	59
4.8	D-Link camera models 5020L and 932L	60
4.9	CVE-2019-10999 Exploit Structure	61
5.1	“Smart Thermostat” ransom note	73
5.2	“Smarter Coffee Maker” infected with ransomware	74
5.3	LG SmartTV locked displaying a ransom note	75
5.4	ChastityLock Ransomware Overview Graph	77
5.5	Binwalk analysis of smart thermometer’s firmware	79
5.6	PaperW8 Structure Graph	80
5.7	Manually Programming a MX29LV320ETTI-70G Flash Chip	85
5.8	8-Pin SOIC Test Clip	86
5.9	MTD partitions in HG532’s Flash Memory	88
5.10	MTD partitions in R6250’s Flash Memory	89
5.11	MTD partitions in TV-7104HE’s Flash Memory	90
5.12	MTD partitions in WiPG-1000’s Flash Memory	92
5.13	MTD partitions in 932L’s Flash Memory	93
6.1	Process graph for modifying a compressed filesystem	112
6.2	DCS-932L Filesystem Extraction Stages	122
6.3	4 pin UART header exposed on the 932L’s PCB. (Highlighted in red)	124

6.4	Process to select optimal persistence method	127
7.1	Various Google-based cloud vision system examples	135
7.2	Basic SSLStrip attack structure	138
7.3	Data collator structure graph	145
7.4	IoT Collator summarising information collected from an R6250 router	146
7.5	Configuration data extracted from the R6250 router	148
7.6	Example privacy-based ransom note with “proof of compromise” .	150
7.7	IBM speech-to-text demo recognising selected keywords	151
7.8	Hijacking the screen of a Yealink SIP-T38G	153
7.9	Labelling images extracted from an infected DCS-932L Camera .	154
9.1	Effects of an ARP poisoning attack	192

List of Tables

4.1	Hijacking experiments performed on various IoT devices.	52
6.1	IoT Persistence Methods	109
7.1	API endpoints hosted by the IoT Collator	145
7.2	Privacy invasion methods used for each device	156
8.1	Applicability of our suggested countermeasures for Linux-based ransomware	179
B.1	Patches and mitigations provided by the device manufacturers. . .	197

List of Listings

4.1	Example TFTP command	47
4.2	Example File Redirection	48
4.3	Recreating a small text file using echo commands	49
4.4	R6250 Command Injection	54
4.5	File Descriptor Closing Script ²	57
5.1	Shell commands run by Brickerbot	69
5.2	HG532 /proc/mtd file	88
5.3	R6250 /proc/mtd file	89
5.4	TV-7104HE /proc/mtd file	91
5.5	WiPG-1000 /proc/mtd file	92
5.6	932L Camera /proc/mtd file	93
6.1	R6250 /proc/mounts file	120

List of Publications

1. **Brierley, C.**, Arief, B., Barnes, D., Hernandez-Castro, J. (2021). Industrialising Blackmail: Privacy Invasion Based IoT Ransomware. In Secure IT Systems: 26th Nordic Conference, NordSec 2021, Virtual Event, November 29–30, 2021, Proceedings, Springer Nature, pp. 72-92
2. **Brierley, C.**, Pont, J., Arief, B., Barnes, D., Hernandez-Castro, J. (2020). Persistence in Linux-Based IoT Malware. In Secure IT Systems: 25th Nordic Conference, NordSec 2020, Virtual Event, November 23-24, 2020, Proceedings, Springer Nature, pp. 3-19
3. **Brierley, C.**, Pont, J., Arief, B., Barnes, D., Hernandez-Castro, J. (2020). PaperW8: an IoT Bricking Ransomware Proof of Concept. In Proceedings of the 15th International Conference on Availability, Reliability and Security, pp. 1–10
4. Pont, J., Abu Oun, O., **Brierley, C.**, Arief, B. Hernandez-Castro, J. (2019). A Roadmap for Improving the Impact of Anti-ransomware Research. In Nordic Conference on Secure IT Systems, Springer, pp. 137–154
5. Kocaogullar Y., Cetin O., Arief, B., **Brierley C.**, Pont, J., and Hernandez-Castro, J. (2022). Hunting High or Low: Evaluating the Effectiveness of High and Low Interaction Honeypots. 12th International Workshop on Socio-Technical Aspects in Security (STAST 2022), pp. 15-31, 29 September 2022

Associated Talks

1. Industrialising Blackmail: Privacy Invasion Based IoT Ransomware
(https://www.youtube.com/watch?v=g5X_ghS0-jA)
2. Persistence in Linux-Based IoT Malware
(<https://www.youtube.com/watch?v=tE19yWXXIw0>)
3. PaperW8: an IoT Bricking Ransomware Proof of Concept
(https://www.youtube.com/watch?v=q_tBbZsuJ7I)

Abbreviations

APK Android Application Package

ARP Address Resolution Protocol

ASCII American Standard Code for Information Interchange

ASLR Address Space Layout Randomisation

CAN Controller Area Network

CNC/C&C Command and Control Server

CPIO Copy In Copy Out

CVE Common Vulnerabilities and Exposures

DDoS Distributed Denial of Service

DEP Data Execution Protection

DNS Domain Name Server

DVR Digital Video Recorder

ECH Encrypted Client Hello

ECU Electronic Control Unit

eMMC Embedded Multi Media Card

ESNI Encrypted Server Name Indication

FBI Federal Bureau of Investigation

GBP Great British Pound

Gbps Gigabits Per Second

GPL GNU Public License

GPS Global Positioning System

GUI Graphical User Interface

HSTS HTTP Strict Transport Security

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICS Industrial Control System

IIoT Industrial Internet Of Things

IoT Internet Of Things

IRC Internet Relay Chat

IV Initialisation Vector

IVI In-Vehicle Infotainment

JFFS Journaling Flash File System

JTAG Joint Test Action Group

kHz Kilohertz

LKM Loadable Kernel Module

MAC Media/Mandatory Access Control

MBR Master Boot Record

MitM Man in the Middle

MTD Memory Technology Device

NAO National Audit Office

NAS Network Attached Storage

NFS Network File Share/Network File System

NHS National Health Service

NSA National Security Agency

PLC Programmable Logic Controller

PoC Proof of Concept

RC Run Commands

RELRO Relocation Read-Only

ROP Return Oriented Programming

RSA Rivest-Shamir-Adleman

RTOS Real Time Operating System

SCADA Supervisory Control And Data Acquisition

SEPA Scottish Environmental Protection Agency

SNI Server Name Identification

SOIC Small Outline Integrated Circuits

SSID Service Set Identifier

STT Speech-to-Text

TFTP Trivial File Transfer Protocol

TLS Transport Layer Security

TPMS Tire Pressure Management System

TTL Time To Live

UART Universal Asynchronous Receiver-Transmitter

USD United States Dollar

VoIP Voice over Internet Protocol

WPS WiFi Positioning Systems

YAFFS Yet Another Flash File System

Chapter 1

Introduction

The Internet of Things (IoT) is becoming more pervasive in society, with millions of devices present in homes, businesses and institutions worldwide. While this technology is in its early stages, new interconnected devices are rapidly being developed. Unfortunately, malware has also been observed affecting IoT devices, which have then been used to perform large scale attacks.

Thankfully, a particularly destructive form of malware known as “Ransomware”, which has drastically impacted desktops and businesses by holding essential devices and data to ransom [184; 117; 96], has not yet targeted the majority of IoT devices.

However, should it be implemented by attackers, IoT-based ransomware could cause significant damage to IoT devices, the reputation of devices developers, their users, and even trust in IoT devices as a whole. The research presented in this thesis investigates the viability of IoT-based ransomware and its possible effects by creating proofs of concept of ransomware techniques that can be applied to IoT devices. This assists in predicting the behaviour of future IoT-based ransomware and the preemptive implementation of countermeasures.

1.1 Research Questions and Objectives

Three research questions were chosen as the basis of the work performed in this thesis, as shown below.

Rq-1 At what scale can IoT-based ransomware be implemented?

Rq-2 What threats could IoT-based ransomware pose to users of IoT devices?

Rq-3 What IoT specific countermeasures can be implemented against IoT-based ransomware attacks?

To answer these questions, six objectives that could be plausibly achieved during this research were established.

Obj-1 Determine the methods that may be used to infect IoT devices with ransomware.

Obj-2 Identify issues attackers may encounter when attempting to deploy IoT-based ransomware.

Obj-3 Identify IoT devices that are likely to be targeted by IoT-based ransomware.

Obj-4 Explore the possible impact IoT-ransomware may have on victims.

Obj-5 Develop proof of concept attacks to demonstrate the feasibility of IoT-based ransomware threats.

Obj-6 If possible, identify potential countermeasures that could be used by developers and end users.

1.2 Research Contributions

Below, the key contributions of this research are summarised.

First, the limitations of IoT devices that could influence the implementation of ransomware were identified. During this work, two major obstructions that would limit the effectiveness of IoT-based ransomware were encountered. To determine whether these issues would prevent IoT-based ransomware from being implemented, methods that attackers could use to circumvent these limitations were investigated. One of these obstacles was the lack of a “standardised communication method” for IoT devices. This prevented ransom notes from being easily delivered to victims of IoT-based ransomware. To overcome this issue, methods to transmit ransom notes using a variety of hijacked IoT communication channels were explored. The lack of “persistence” in IoT-based malware also presented a problem. As ransomware relies upon retaining control over user-owned assets its

effectiveness may be limited if it is unable to persist on a device after a reboot. After investigating this issue, six methods were identified that could be used by an attacker to retain persistent control of a device after being restarted.

Second, techniques that attackers could use to ransom victims through IoT devices were evaluated. During this work, two viable ransom techniques were identified that could be used in a real-world setting. The first technique, “bricking”, would allow an attacker to perform persistent damaging changes to IoT devices, such that infected devices would no longer function unless the victim made a payment to the attacker to reverse the modifications. The second identified technique would allow an attacker to access the sensors of infected IoT devices to invade a victim’s privacy. During this work, multiple methods to extract and interpret private data from IoT devices were discovered. IoT sensors such as microphones, cameras, internet traffic, and nearby WiFi signals were all found to be potential data sources that could be used by attackers to extract private data about infected victims for the purpose of performing ransomware attacks.

Third, for each of the above methods and techniques, proofs of concept were created to demonstrate their viability. These developments were created with generalisability in mind, such that they could be applied to multiple devices with different uses, such as cameras, routers, and IP phones.

Fourth, the impact of these implementations were assessed for their possible effect on victims and the IoT industry in general. The scalability of the proposed solutions was also evaluated, including how a ransomware campaign could be designed to feasibly manage devices, data, and victims.

Fifth, countermeasures for each of the discovered techniques were identified, including existing IoT anti-ransomware tools, general security policies, and mitigation techniques for specific ransom methods.

1.3 Thesis Outline

This work is organised into a number of chapters.

Chapter 2 provides the reader with a literature review, covering previous relevant research on IoT, ransomware, and IoT-based ransomware.

Chapter 3 contains the methodology, which summarises and justifies the approaches taken during this work.

Chapter 4 outlines the challenges prospective attackers may have in delivering ransom notes via infected IoT devices and possible methods of communication, such as channel hijacking.

Chapter 5 explores the viability of bricking-based ransomware on IoT devices using methods such as bootloader encryption.

Chapter 6 discusses persistence (the ability to retain control of a device after a reboot) and its relevance for IoT-ransomware, including a summary of persistence methods that could potentially be used by attackers.

Chapter 7 investigates the possible use of IoT devices to violate the privacy of users for the purposes of ransomware, such as by surreptitiously extracting private images, audio, or browsing activity via infected devices.

Chapter 8 examines possible countermeasures that could be used by both developers and users to reduce the effectiveness of IoT-based ransomware.

Chapter 9 discusses the results and contributions of this work. It then examines its limitations and any potential future work.

Chapter 2

Literature Review

2.1 Introduction

In this section, a literature review is provided, covering the topics discussed within this thesis. This will be separated into three sections. First, in Section 2.2, literature examining IoT is discussed. Topics include the various uses of IoT and studies of previous IoT malware. Next, in Section 2.3, papers covering desktop-based ransomware are reviewed. Topics include examinations of previous notable families of ransomware, ransomware’s targets and effects, and techniques that can be used to reduce the effectiveness of ransomware. Finally, building upon the previous sections, Section 2.4 will introduce IoT-based ransomware. Topics include IoT-based ransomware’s possible impact, previously discovered instances of IoT-based ransomware, and device types that may be targeted for future attacks.

2.2 The Internet of Things

As technology becomes cheaper and more readily available to developers, smart devices used for various purposes are becoming more ubiquitous in day to day life. The emphasis on introducing connectivity has led to the creation of devices exhibiting networking capabilities. Such devices have been grouped under the banner of “The Internet of Things”.

When investigating the viability of ransomware for IoT devices, it becomes necessary to have a clear definition as to what can be classified as an “IoT Device”,

which is often up for debate [276].

While it could be argued that “any device that is capable of connecting to the Internet” could fit under the term, these types of definitions [200] may be too general, as it would not only include common devices that most would expect to be included, such as virtual assistants or cameras, but desktops and phones as well. A scope of this scale would be too large to effectively cover and could lead to the results being too general to be applied in any meaningful way, or lacking focus. Conversely, if the definition of IoT devices is too restrictive, the results are likely to be inaccurate and heavily skewed by bias. For the purpose of this research, a more targeted definition of IoT is employed instead, to strike a balance between these two outcomes.

For the work contained within this thesis, an IoT device is considered to be “*an Internet-capable device that has been designed for a single purpose*”. For example, the single purpose could be to provide a video stream for surveillance purposes, monitor a user’s vital statistics, or remotely manage the temperature of the user’s home. This allows for various types of Internet-connected devices, while excluding the more general-purpose counterparts, such as smartphones and desktop PCs.

2.2.1 IoT Device Design and Use

The pervasiveness of IoT devices has only grown over time, with approximately 10 billion IoT devices currently believed to be connected to the Internet, with an increase to 25.4 billion predicted by the year 2030 [255]. This increase in adoption can be partially attributed to the many possible applications of IoT. While IoT devices may typically be associated with “personal use” by consumers, businesses and industries such as healthcare and manufacturing [74; 218] have also implemented IoT devices as part of their everyday operation.

Due in part to their use in many different contexts, IoT devices also implement differing software, hardware and architecture to fit their requirements. Given the sheer number of devices that are in use, it can be challenging to track or measure such features of IoT devices “in the wild”, however, attempts have been made by both researchers [65] and developers [82] to identify attributes of popular IoT platforms.

In 2014, researchers analysed over 32 thousand firmware images in an attempt

to identify common security vulnerabilities in IoT applications [65]. Of the collected firmware, 86% were found to use Linux as the main operating system, with 112 distinct kernel versions.

Various architectures were also observed to be in use by developers, 63% of the collected firmware images targeting ARM platforms, followed by MIPS at around 7%¹.

Variation of this kind can be inconvenient for both malicious actors attempting to attack devices, and for developers who aim to implement effective defences, as more consideration must be taken to account for the differing architectures, operating systems and software.

2.2.2 IoT Security

IoT devices are generally considered to be less “valuable” than personal computers, as they are less likely to contain unique user files, and are typically much less expensive. Unlike personal computers, however, many IoT devices are produced at scale and are explicitly designed to provide services that are remotely accessible over the Internet. Such services are likely to function identically on devices of the same model and version, and therefore, if they are found to be vulnerable, attackers can attempt to exploit such services *en-masse*. As such, developing secure IoT software is very important to limit the impact of attacks.

In 2010, researchers at Columbia University performed scans over “large portions of the Internet”, to discover “trivially vulnerable embedded devices” [70]. During their research, 3.9 million devices in 144 countries were scanned for weak authentication vulnerabilities. Of the scanned devices, 540,000 (approximately 14% of the discovered total) were found to be using a factory default root password. Additionally, during subsequent scans, 96.75% of the discovered devices remained vulnerable after a period of 4 months.

Further analysis of the scanned devices allowed the researchers to identify the intended usage of the discovered devices. Interestingly, some IoT device types exhibited surprisingly high vulnerability rates when compared to others. Video conferencing devices, for example, were found to be vulnerable over 55% of the

¹While this research does indicate some of the common attributes of IoT devices, it should be noted that these statistics were produced by examining the contents of discovered firmware images. Therefore, this may not accurately reflect the attributes of the total “in use” devices.

time.

It should be noted that as this paper was written in 2010, and only studied the effect of weak credentials for 73 device types, it is somewhat limited in scope when compared to the current scale of IoT. The authors also highlighted that the vulnerable population could be significantly increased by “slightly escalating the level of sophistication” of the attacker.

As an example of more complex attacks, actionable vulnerabilities such as leaked private RSA keys, backdoors or hard-coded credentials have also been discovered by researchers performing mass firmware scans. In 2014, researchers performed vulnerability scans on 32,000 firmware images, leading to the discovery of vulnerabilities in over 123 different products and 38 new CVE definitions [65]. At the time this research was performed, over 140,000 devices that implemented these vulnerable firmware images were found to be publicly accessible from the Internet. This automated approach to discovering and reporting embedded vulnerabilities may allow developers to more easily manage the security of embedded devices en-masse and catch simple vulnerabilities before the device is put into production.

Rather than using password brute-forcing to exploit IoT devices’ weak authentication practices, more advanced exploitation techniques, such as buffer overflows, can also allow attackers to obtain remote access to vulnerable devices. Multiple binary hardening protections have been created to reduce the impact of these types of vulnerabilities, such as non-executable stacks [177; 213], address space layout randomisation (ASLR) [199], and stack guards/canaries [48]. These are often implemented by desktop applications but are not as commonly used on IoT devices.

In 2019, a study of over three million IoT binaries was performed to identify binary protections used by different IoT vendors [71]. The results showed that in 2012, most vendors implemented the bare minimum of protections, with some forgoing them altogether. When assessing later images produced in 2018, the use of non-executable stack protections exhibited a marked increase for most vendors. Other protection types (such as RELRO) showed similar increases, but only by some developers, rather than universally.

While there is still plenty of room to improve, and the use of effective binary protections is far from universal, this positive trend indicates that developers are

aware of these issues and are making attempts to secure IoT devices during the development stage.

The studies covered in this section highlight the extensive insecurity in IoT. As more devices are produced and purchased, additional opportunities to exploit and hijack these devices will be provided to attackers.

2.2.3 IoT Malware

Over time, IoT devices have been increasingly targeted by malware, with devices' insecurity and availability leading to large scale compromise via malicious Internet scanners. Here, literature examining previous IoT malware activity will be covered, including case studies that discuss the more notable examples in further detail.

In 2016, researchers developed a honeypot to capture and analyse IoT-based malware, named "IoTPOT" [204]. IoTPot acted as a honeypot of emulated IoT devices by running instances of OpenWRT² using QEMU³ with weak authentication.

This setup allowed the researchers to study IoT-based malware attacks on IoT devices with weak telnet [209] credentials. Around 16,900 IP addresses connected to the honeypot over 39 days, and approximately 76,700 attempts were made to download and execute malware.

Further analysis allowed the researchers to dissect the stages of the telnet based infection process, such as unique bruteforce password lists, device reconnaissance steps, and the structures of the associated command and control servers. This work provided an early insight into fledgling IoT malware behaviour.

The researchers later manually downloaded 43 unique malware binaries for further analysis, 39 of which had not been seen by the VirusTotal⁴ database. The authors also discovered that some families were found to target up to 9 different CPU architectures, highlighting the attackers' need to adapt to account for IoT devices' variations in design.

²OpenWRT: <https://openwrt.org/>

³QEMU: <https://www.qemu.org/>

⁴VirusTotal is an online service which allows users to analyse files and URLs for malicious content. Files that are uploaded are scanned by over 40 different anti-virus tools.

A later study performed in 2019 categorised features used in IoT-based malware over the previous 10 years. This included the exploitation methods used, employed DDoS attack techniques, and the means of monetisation [266]. This provided a much more in-depth insight as to the *evolution* of IoT-based malware, summarising the lineage of various features as they were “transferred” from one malware family to the next. Overall, this paper classified and tracked 15 features across 16 different families of IoT malware.

Most previous IoT malware was found to focus almost exclusively on performing DDoS attacks. However, some families exhibited alternative attack methods, such as man-in-the-middle packet sniffing, or alternative forms of monetisation, such as cryptomining. Some families did not even intend to turn a profit for the author. Brickerbot, for example, simply aimed to disable any device that it infected. Below, some case studies of the more notable families of IoT-based malware are examined.

2.2.3.1 IoT Malware Family - Mirai

One of the more successful families of malware, Mirai, was put under extensive study by multiple researchers, producing several papers. One such paper, “Understanding the Mirai Botnet”, went into great detail as to its method of operation [12]. Mirai was especially interesting in part due to its fast spread. Its initial method of exploitation and infection was relatively basic, only attacking weak SSH or telnet authentication with a list of hardcoded passwords, before later adding the ability to exploit services known to be served on TCP ports 7547 and 5555. Despite this, Mirai was able to infect hundreds of thousands of devices of different types and architectures during its lifetime.

Mirai was used to perform over 15,000 DDoS attacks, some of which targeted high profile services such as Dyn, a DNS provider for “...high-traffic sites such as Amazon, Github, Netflix, PayPal, Reddit, and Twitter” [12], causing large scale outages. In 2016, Mirai was also used to DDoS a network provider in Liberia, “Lonestar Cell”, significantly reducing the Internet service quality in the country. Some attacks reached 800 Gigabits per Second (Gbps) which, at the time, was the largest of its kind.

In the same year, the source code of Mirai was publicly released on the “hackforums” website [11]. This led to new variations being created and released by

other malicious authors, often with additional features, such as support for further devices and new methods of exploitation.

The authors of the paper state that the success of Mirai demonstrated the relative ease at which hundreds of thousands of devices could be exploited and infected by simple brute force attacks. They recommended some possible countermeasures to limit the spread of such malware, such as the implementation of automatic updates and binary hardening mechanisms (such as ASLR). It was also suggested that IoT devices should be more easily identifiable by network administrators, such that the device could be replaced or removed from the network if found to be vulnerable.

2.2.3.2 IoT Malware Family - VPNFilter

VPNFilter is a family of advanced IoT malware that targets various routers and Network Attached Storage (NAS) devices. While most current IoT malware typically monetises infected devices by performing DDoS attacks or mining cryptocurrency, VPNFilter does not act with the intention of generating income for the attacker. Researchers at Talos Intelligence likened VPNFilter to an “intelligence-collection platform”, which after infecting a targeted device, could remotely execute commands, extract local information, or intercept traffic on the local network.

In a three-part analysis of the VPNFilter malware [239; 241; 240], Talos Intelligence estimated that at least 500,000 devices had been infected. Unlike previously observed IoT malware, VPNFilter has a much more flexible design that supports the use of downloadable modules to implement additional features.

With the use of modules, VPNFilter can selectively add various capabilities, including network mapping, destruction of the infected device, and even interception of Modbus traffic, which is used to communicate with Programmable Logic Controllers (PLCs) that are often used in Industrial Control Systems (ICS). This malware highlights some of the more creative approaches that attackers could employ, given the flexibility of IoT devices.

2.3 Ransomware

Ransomware is a type of malware that is designed to extort victims through the restriction or exposure of valuable data [98]. For the effects of a ransomware infection to be reversed, the victim must make a “ransom payment” to the attacker, normally through the use of cryptocurrency (such as Bitcoin or Monero) within a specified time frame [123; 165]. Here, various aspects of ransomware are examined, including their typical method of operation, large scale attacks of notable ransomware families, and countermeasures that could be implemented to reduce their effectiveness.

2.3.1 Types of Ransomware

Ransomware can use a combination of different methods to extort victims. Previous work has attempted to classify these methods into various categories [7], which are described below.

2.3.1.1 Locker-Ransomware

Locker-ransomware limits the functionality of the devices that it infects, preventing victims from being able to use their devices effectively unless a ransom is paid. One downside of this approach is that if locker-ransomware is successfully cleared from an infected device, its functionality will likely be returned to the victim, removing the motive for making a payment to the attacker.

However, researchers at Symantec have highlighted that locker-ransomware may be particularly effective on IoT devices, as users typically have fewer methods of interaction when compared to desktops, which may hinder recovery efforts [220].

2.3.1.2 Crypto-Ransomware

Crypto-ransomware is a particularly popular extortion method, whereby an attacker encrypts files that have value to the victim with a secret key. Victims can regain access to these files by making a payment to the attacker, who will typically provide them with a decryption key or recovery tool. Crypto-ransomware can be further categorised by its encryption methods: symmetric, asymmetric and hybrid, each with its own benefits and drawbacks [28].

- **Symmetric.** Symmetric encryption uses one key for both encryption and decryption and is typically much faster to perform than asymmetric. However, this could allow the victim to intercept the key being used during a ransomware attack, which may lead to them being able to recover encrypted assets without paying the ransom. Examples of ransomware that uses this type of encryption includes **UIWIX** [257] and **Bucbi** [118].
- **Asymmetric.** Asymmetric encryption uses a “key pair”, requiring one key for the encryption stage, and one for the decryption stage. While this would prevent the decryption key from being intercepted by the victim, it is generally much slower and may result in larger file sizes.
- **Hybrid.** Hybrid ransomware uses both symmetric and asymmetric encryption as part of an attack. Individual assets are encrypted quickly using symmetric encryption, then the symmetric keys that were used are saved and encrypted using an asymmetric key pair. If the victim elects to pay the ransom, the attacker can then provide the private key, which is then used to decrypt the symmetric keys and regain access to any encrypted assets. Examples of ransomware that uses this type of encryption includes **WannaCry** [26] and **CryptoLocker** [205].

2.3.1.3 Information Leaking

In addition to the previous methods, more recent ransomware has begun to steal potentially valuable information from targets that they exploit. If the stolen information is particularly sensitive, the attacker can further pressure the victim by threatening to publicly release the stolen data unless a payment is made. This approach can be particularly effective when used against targets that handle private customer or employee information, such as personal details or medical documentation.

In one notable example, a Finnish mental health institution, “Vastaamo”, was attacked with ransomware in 2020. During this attack, many patients were sent ransom notes threatening to publicly release notes taken during their therapy sessions unless they paid the attackers €200 in bitcoin [211].

2.3.2 Post-Infection

For ransomware to be profitable, there must be a means to obtain payment from victims. A paper titled “Tracking Ransomware End-to-end” studied the methods used by ransomware to extract ransom payments from its victims [123].

The researchers split the process of ransomware into five distinct stages:

- **Delivery:** How the ransomware authors chose to infect its victims, such as malicious email attachments or exploiting vulnerabilities.
- **Execution:** The method used to ransom a device, such as encrypting files, or locking the devices’ functionality.
- **Payment:** Communicating with the victim such that the attacker can extract payment.
- **Decryption:** Restoring the victim’s access to their assets if a payment has been made.
- **Liquidation:** Transferring monetary gains for use, such as converting cryptocurrency into fiat currency.

This section will focus on the payment and liquidation stages.

2.3.2.1 Payment

To obtain payment from a victim, ransomware must first attempt to communicate with the user. This typically involves displaying a ransom note, which will ideally detail what has occurred, how the ransomed assets may be recovered, and how a payment may be made. Additional features may be utilised by the ransom note to further pressure or “assist” victims making a payment, such as:

- **Timers.** Timers for payment are often used to add additional time pressure to victims. The victim is typically informed that when a timer expires, they will suffer some form of penalty, such as the ransom price being doubled, files being deleted at random [2], or removal of the ability to recover the ransomed files altogether [116].

- **“Demo” Decryption.** Some crypto-ransomware provided services that would allow victims to upload small files for decryption. By providing the original file to the victim, it served as proof that the victim’s files could be recovered once the payment was made [122].
- **Support Lines.** Some strains of ransomware would supply support channels to assist victims, such as telephone helplines or support emails, which may be in response to less technical victims being unable to navigate the purchase and transmission of cryptocurrency. In some cases, victims were able to communicate with the ransomware authors and negotiate time extensions or discounts [86].
- **Payment Guidance.** Ransom notes often provide guides as to how to effectively make payments to the attacker [256; 123]. These typically include how to purchase the necessary currency, where it needs to be sent, and how to recover the ransomed assets once payment has been made. For ransomware payments to succeed, the process must be easy for victims to follow and, in theory, impossible to reverse.

Originally, ransoms could be paid using various different money transfer systems. One such example was pre-paid cards, the value of which could be transferred to another by sending the recipient a unique numerical code. However, this method presented some downsides for ransomware operators, such as the “geographic availability” of the chosen systems, which would prevent victims in certain countries from being able to pay, and the possibility of the owning company reversing transactions if the malicious activity was detected [123].

Now, modern ransomware often uses cryptocurrency to facilitate payment. The use of cryptocurrency provides a number of advantages, such as a lack of regulation or accountability, and universal availability. Additionally, given the decentralised nature of cryptocurrency networks, it is incredibly difficult to reverse any payments made by the victims [123].

However, Huang et al. [123] also highlighted that the public ledger also benefits researchers, as it allows anyone to examine the details of any transaction, including those made to wallets known to be associated with ransomware. To exploit this information leak, the researchers extracted addresses of ransomware related wallets from public posts made by victims, or from ransomware set to run

within a controlled environment. During this research, approximately 16 million dollars worth of bitcoin ransom payments made by 19,750 victims over a period of two years were observed [123]. While some malware families were found to be generating a unique address for each victim, others, such as WannaCry, used the same addresses for multiple victims. Huang et al. also highlighted that if wallet addresses were reused, the attackers had to use alternative methods to determine which victims were making a payment, such as requiring the victim to confirm their “payment transaction hash”, or in WannaCry’s case, not decrypting the victim’s files at all.

By making “micropayments” to the extracted bitcoin addresses, Huang et al. were able to track payments made to other wallets, identify “clusters” of addresses associated with certain ransomware families, and estimate the potential revenue of the various campaigns.

2.3.2.2 Liquidation

After successfully ransoming their victims, attackers are likely to liquidate the obtained cryptocurrency via an exchange, where it can be sold for fiat currency [123].

As the exchange may be compelled to reveal the identity of malicious users, attackers may attempt to obfuscate this process by using a cryptocurrency tumbler [123]. Tumblers aim to anonymise cryptocurrency by “mixing” multiple users’ assets, breaking any connections with the previous addresses, and then returning the assets to the original owners.

While a significant portion of liquidation could not be tracked due to the use of tumblers, it was found that the ransomware families “Locky” and “CryptoDefense” were using an exchange named “BTC-e.com”, with over \$3 million USD in Bitcoin being transferred. The exchange was later seized by US authorities in 2017 [123; 285].

2.3.3 Large Scale Attacks

Not only individuals have been adversely affected by ransomware; companies, cities and organisations have also been subject to attack. Here, some of the notable large scale attacks will be examined, including the methods of infection, recovery, and damages caused.

2.3.3.1 WannaCry - National Health Service

In May 2017, the ransomware family “WannaCry” leveraged a leaked exploit developed by the American National Security Agency (NSA), named EternalBlue, to infect over 200,000 computers in more than 100 countries.

The United Kingdom National Health Service (NHS) was heavily affected by these attacks. In October of the same year, the National Audit Office (NAO) released a report detailing an investigation of the impact on health services caused by the attack [184]. This provided a glimpse into the potential damage that could be caused by ransomware, including the effects beyond monetary loss.

Infections were able to spread throughout the NHS via the Internet and the “N3 Network”, an internal network that connected NHS locations throughout the UK. During the initial stage, WannaCry was able to infect Windows XP and Windows 7 based systems, affecting “at least 81 of 236 trusts”, as well as 603 other related organisations, including 595 GP practices.

Infected trusts were locked out of their devices, preventing staff from being able to effectively access patient information. Connected medical equipment (Such as MRI scanners running Windows XP embedded) were either locked or isolated from the main network to block infection, preventing them from being used.

After the initial attack, 1,200 diagnostic devices (accounting for “1% of all such NHS equipment”) were reported to have been infected. During the resulting disruption, five trusts were forced to “divert emergency ambulance services to other hospitals”.

Patients were heavily impacted, with the NAO estimating that around 19,494 appointments were cancelled overall, impacting at least 139 patients who had “an urgent referral for potential cancer”. However, there were no reported cases of direct harm to patients, or any patient data being modified or stolen.

On the 12th of May, a researcher discovered that after infecting a device, WannaCry would attempt to access a web page at a certain hard-coded domain, which at the time, was not hosting any content. After the researcher purchased the domain and used it to host a web page, it was found that WannaCry would activate a “kill-switch” if the domain was found to be available, preventing it from spreading or locking additional devices. Unfortunately, this did not return access to the devices that had previously been infected.

None of the affected NHS trusts were reported to have paid the ransom. However, there was still a significant financial impact. While an estimate was not included in the initial report, the Department of Health and Social Care reported that the cost of IT support and lost output totalled approximately £92 million [78].

2.3.3.2 Norsk Hydro

In March 2019, Norsk Hydro, an aluminium producer, was attacked with the “LockerGoga” ransomware [46], which had previously been used to attack other engineering and industrial firms. The company had been infected via a phishing email delivered from a trusted customer which, when opened, installed a Trojan. The attackers then used the acquired network access to escalate to administrative privileges, leveraging the company’s domain controllers to distribute the ransomware, infecting 22,000 computers over 170 sites and 140 countries [46; 247].

Despite the damages caused by the ransomware, Norsk Hydro refused to pay the ransom or communicate with the attackers, and efforts were made to continue production where possible. In some cases, assistance was provided by retired workers who were “familiar with the old paper system” [46; 247]. Overall, the financial impact was estimated to reach around \$71 million USD [46].

Approximately two years later, a group of 12 people linked to 1,800 ransomware attacks, including the attack that impacted Norsk Hydro, were arrested as part of an international effort performed by Europol and Eurojust [85; 254].

2.3.3.3 Colonial Pipeline Attacks

In 2021, the Colonial Pipeline was held to ransom. It is considered to be “the largest pipeline system in the US” and can transfer three million barrels of fuel over 5,500 miles per day [157; 151].

The attackers were able to gain access to the company’s internal VPN via a previous employee’s password, which may have been obtained from an online password leak. The attack was attributed to the “DarkSide” group, who left a ransom note demanding payment for the affected systems to be restored. Additionally, if the ransom was not paid, the group threatened to release almost 100 gigabytes of private data that was stolen during the attack [262; 203].

Although it is believed that the attackers were not targeting the “operational”

systems that governed the pipeline’s functionality [203], the pipeline was shut down as a precaution, leading to gas shortages and increased prices across the nation [262].

Colonial opted to pay the ransom, a total of 75 bitcoins, which was worth approximately \$4.4 million USD at the time [262]. After the payment was received, the attackers provided a recovery tool that allowed the company to regain access to their systems. A later update from the department of justice claims that the FBI were able to recover 63.7 bitcoins of the ransom paid to the attackers, although it was not disclosed how this was achieved [79].

2.3.3.4 JBS

In May 2021, shortly after the Colonial Pipeline attack, JBS, the “world’s largest meat processing company”, was infected with ransomware [32], which the FBI attributed to the Russian based hacking group “REvil” [131].

This attack lead to the shutdown of multiple plants in the US and Australia [187]. Although JBS released a statement that the backup servers had not been affected by the attack [132], in the following month, a ransom of around \$11 million USD in Bitcoin was sent to the attackers to restore operations and prevent further disruption.

2.4 IoT-Based Ransomware

With the increasing popularity of IoT, ransomware and IoT malware, it is not surprising that IoT-based ransomware has also received increased interest. As attackers explore new methods to monetise exploited IoT devices, it would be natural for IoT-based malware to progress from more “traditional” methods, such as DDoS attacks or mining botnets, into ransomware, given ransomware’s success when attacking personal computers.

2.4.1 Targets

IoT is a very expansive topic of study, due in part to its variability and applicability to a wide range of tasks and use cases. This has driven research into how IoT-based ransomware could attack various device types used in different industries.

In this section, papers investigating the use of ransomware on various device types will be examined. While this list is in no way exhaustive, it illustrates how ransomware could behave differently depending on what is being targeted.

2.4.1.1 Mobile Phones

Due in part to its prevalent use, Android (an operating system designed primarily for mobile devices) has been a popular target for various malware. In 2018, Gartner stated that Android accounted for 85.9% of the market share for smartphones [95], providing malicious actors with ample targets.

As one might expect, mobile phones are very appealing targets for ransomware authors, as they are often expensive, difficult to replace, and are likely to contain troves of personal or unique information that has value to the victim.

Most Android phones have “Google Play” installed by default, which acts as a central store where users can search for, purchase, and install various applications. While Google does implement an approval process for new applications, it is possible to install applications from other sources by directly installing Android application package (.apk) files.

By convincing victims to install malicious applications from outside of the Google Play Store, attackers can sidestep the approval process. One such example, “ScarePakage” masqueraded as a fake flash player, which upon installation, would lock the device after supposedly discovering “illicit content” [282].

Early ransomware on the Android platform locked infected devices using a variety of methods. Three of the most commonly used methods are shown below [10]:

- Requesting administrative permissions, then calling an Android function to force a “lock screen timeout”, preventing the device from being used.
- Creating “immortal activities” that would fill the screen and disable home button actions, preventing other applications from being accessed.
- Creating “immortal dialog” alerts that are impossible to close.

While it is theoretically possible to recover the use of the device by factory resetting, any personal data belonging to the user would also be lost.

Some families of Android ransomware, such as “Simplocker”, also exhibited the ability to encrypt local files stored on the device or SD card. The early version would use a single “master” key to encrypt files on infected devices, but this design was heavily flawed as the key was hardcoded within the malware and was not unique for each device. Developers at Avast were able to exploit this weakness by extracting the master key and creating an anti-ransomware tool that allowed victims to decrypt ransomed files, thereby neutralising the attack without needing to pay the ransom [55]. However, later versions would evolve to use “per-device” encryption keys, preventing easy recovery.

Others have also been attempting to create countermeasures for such ransomware attacks. One of the notable examples is “Heldroid” [10], which aimed to detect both encrypting and locking ransomware targeting the Android platform. To do so, it attempts to identify features in Android applications that may indicate malicious behaviour. First, it scans strings present in the application and passes them through a “Threatening Text Detector”, which can be used to classify whether threats are being made to the user. As threatening language is likely to be present in any included ransom notes, applications that exceed the acceptable threshold are passed to the next stage. In the next stage of detection, the application is statically analysed to detect whether any “malicious operations” are performed, such as the use of encryption and deletion on local files, or the use of any functions related to maliciously locking the device, as mentioned above.

When tested against a set of 433 Android applications, 375 were correctly labelled as ransomware or scareware and 49 were correctly flagged as benign. Of the remaining 19 samples, 11 used languages that Heldroid was not trained against, and were thus unable to be detected by the language processing, 4 contained no static or generated text, and 4 (while undetected) were unable to perform a ransomware attack due to unavailable Command and Control (C&C) servers.

2.4.1.2 Medical Devices

While medical devices have not been specifically targeted by ransomware, it has been shown that they could potentially be impacted by it. As mentioned in Section 2.3.3.1, various medical equipment was brought offline by the WannaCry attack, despite not being the original targets [184].

Devices of this type are often very expensive to replace and operate in time-sensitive settings. As such, attackers may view them as valuable targets for ransomware.

2.4.1.3 Storage Devices

Crypto-ransomware typically aims to encrypt “valuable” user-generated files in order to ransom victims. While most IoT devices do not contain such files, this is not always the case. NAS devices are one such exception, as their primary use is to store files and allow them to be accessed over a network. Due to the likelihood of such devices containing large amounts of valuable data, they can be quite an appealing target for attackers.

One family of ransomware, named “QLocker”, exploited QNAP brand NAS devices using an undocumented “backdoor” account [97], encrypting victims’ files using “7-zip”⁵, an open source archiving utility.

For the encrypted files to be recovered, victims were required to make a ransom payment of 0.01 bitcoins (approximately \$500 USD) to the attacker. During the campaign’s first month of operation, over \$350,000 USD worth of bitcoin was extracted from victims [5].

2.4.1.4 Industrial IoT

Industrial Control Systems (ICSs) and Programmable Logic Controllers (PLCs) can be used to assist in the automation and monitoring of industrial applications, such as power generation, manufacturing, and infrastructure management.

ICS have been subject to various high profile attacks, such examples include the Stuxnet worm, which targeted uranium enrichment facilities in Iran [160], and the BlackEnergy attacks performed on the Ukrainian power grid in 2015, which prevented approximately 225,000 customers from receiving power for up to 6 hours [81; 286].

The use of Internet-connected devices in these systems is sometimes referred to as the “Industrial Internet of Things”, or IIoT, which can allow companies to remotely monitor, test, control, and track devices or trends within their manufacturing processes. While connecting these types of devices to the Internet can

⁵Available at: <https://www.7-zip.org/>

benefit those managing the development process, it can also open the system up to attack.

As such, research has been performed to investigate the possibility of ransomware being used to attack ICS. One paper presented in 2017 investigated the possible methods ransomware could use to extract a payment from targeted manufacturers [92]. The authors highlighted that ICSs do not typically hold valuable data, which would limit the effectiveness of traditional crypto-ransomware. Instead, they identified three other aspects that could be exploited by the attacker to maximise a possible payout:

- **Downtime.** Unlike commercial IoT devices, failures in ICS can result in significant losses. One such example given by the authors is the car manufacturing industry, in which millions of dollars could be lost for every hour of downtime. As such, causing downtime in critical systems would pressure victims to pay ransoms quickly in order to recover system operation.
- **Equipment Health.** As ICSs often interact with the physical world, it could be possible to cause physical damage to the attached equipment, which may require repairs and lead to further delays.
- **Human Safety.** In addition to potentially impacting attached equipment, unreliable or unpredictable systems may present a danger to staff, which may prevent easy remediation and further encourage payment.

Formby et al. [92] also designed and tested a proof of concept (PoC) PLC-based ransomware named “LogicLocker” targeting three popular PLC models: MicroLogix 1400, Schneider Modicon M221 and Siemen S7-1200. Many of these devices were found to be publicly accessible via the Internet, with around 1429 MicroLogix 1400s being discovered via the Shodan search engine. However, the authors emphasised that some PLCs may not be directly accessible from the Internet, but they could still potentially be accessed if an associated corporate network is compromised, increasing the number of targets into the tens of thousands.

To test the PoC malware, a testbed was created using the aforementioned device models to simulate the “disinfection stage” of a hypothetical water treatment plant. The device was then infected with “LogicLocker”, which would scan the network for other devices, reprogram any that were encountered with a new password, and encrypt any discovered programs, preventing them from being used.

The attacker can then use a separate computer to send a ransom note to the victim via email⁶. In this scenario, the attacker can threaten to add excessive chlorine to the water supply if the ransom is not paid to further pressure the victim.

Upon payment, a program would be provided to the victim, which could be used to restore the original functionality of infected devices. The authors of this malware do highlight that while the “encryption” used to disable the programs on the infected PLCs is by no means secure, it “merely has to slow down recovery enough to make paying the ransom more attractive than a recovery attempt” [92], as downtime can be incredibly expensive in an industrial environment.

Similar work performed by Zhang et al. [289] in 2020 created another ICS-based ransomware PoC named “ICS-BROCK”, which in addition to infecting and locking individual PLCs, was also able to infect “supervisory computers” running Windows, which are typically used to monitor the status of ICSs.

IIoT gateways, which allow legacy ICSs to connect to other services on the Internet, have also been researched as a potential target for ransomware. In 2019, researchers were able to examine existing Linux-based ransomware and develop a simple Python-based crypto-ransomware to attack an IIoT gateway testbed [6]. In addition to encrypting various directories relating to the functionality of the device, the researchers were also able to modify sensor readings, which may be able to cause failures or damage in a “real” setting.

2.4.1.5 Travel

In 2010, research was performed to assess the attack surface of modern “connected” vehicles by analysing a “late-model mass-production sedan” [54]. During this work, multiple methods that could be used to exploit the target vehicle were discovered via various channels of communication, such as:

- Malicious CDs being played via the infotainment system
- Causing a buffer overflow via Bluetooth
- Unauthorised access to “Passthru” devices

⁶The authors indicate that future versions could instead force infected PLCs to send emails to the victim on the attacker’s behalf.

- Malicious phone calls to the car to cause a buffer overflow

Some of the defined attacks could be performed at a significant distance, and subsequently controlled via the Internet. In this work, compromised cars were forced to join an Internet Relay Chat (IRC) channel, which would then act as a method of C&C. The researchers were also able to monitor and control significant portions of the vehicle, such as recording audio from within the car, reporting the current GPS location at a predefined interval, and disable locking/anti-theft features. Various methods that attackers could use to monetise compromised vehicles were then explored, such as via theft or surveillance.

Later work also examined the potential of performing similar attacks on future automated vehicles, which could result in significant threats to users, such as causing incorrect decisions in navigation, forcing sudden braking, or disabling the vehicle [208].

While ransomware was not mentioned within papers [54] and [208], such attacks could be used as the basis of fledgling ransomware to be implemented on vulnerable vehicles remotely.

In 2015, researchers were able to remotely exploit and control a 2014 Jeep Cherokee via the Sprint network [178]. This attack would give an attacker full control of the in-built entertainment system, access to the internal CAN bus, and the ability to influence the brakes and steering of the vehicle. As a result of this research, approximately 1.4 million cars outfitted with the vulnerable entertainment systems had to be recalled. This research called attention to the possible attacks that could be performed against Internet-connected automobiles.

Later research produced PoC ransomware targeting In-Vehicle Infotainment (IVI) Systems [27] and Electronic Control Units (ECU) [274]. After creating a test environment with an IVI running QNX RTOS (an embedded real-time operating system), the researchers attempted to identify possible methods attackers could use to infect a car with ransomware. One such discovered method was the ability to connect to an open “QCONN” port and run unauthenticated applications, which could theoretically be used to run malicious code [27]. Possible attacks could include denial of service, distracting the driver, or encryption of local files.

Bajpai et al. [27] also highlighted several differences between “traditional ransomware” and “vehicular ransomware”, such as the limited resources, differing operating systems, and the lack of “valuable” personal data. This re-enforces the

notion that attackers may need to take novel approaches to ransom users when attacking new “types” of devices.

2.5 Conclusions

In this chapter, a review of existing research was performed, with a focus on general IoT security, IoT-based malware, previous studies of desktop-based ransomware implementations, and how ransomware has been used to target Internet-connected devices in various industries.

When exploring the security of IoT devices, various works highlighted how vulnerabilities that can be encountered on insecure IoT devices can be detected, exploited, and mitigated. Analysis was also performed on existing IoT-based malware, which is currently being used to perform Distributed Denial of Service attacks, with some families managing to infect hundreds of thousands of devices simultaneously.

A different form of malware, ransomware, has been shown to be highly profitable and destructive, being used in multiple large-profile attacks against oil pipelines, medical institutions and manufacturers. However, while IoT-based ransomware has been considered, it is still in the early stages of development. For example, attackers have produced ransomware for Android-based phones, and researchers have created proofs of concept demonstrating implementations of ransomware targeting vehicles and industrial control systems.

Current examples of IoT-based ransomware are designed to be compatible with a certain subset of devices. This is dissimilar to the designs of existing IoT malware, which typically aims to be highly adaptable, such that it can target a wide range of commercial devices. Current research does not adequately cover the possible applications of IoT-based ransomware on “common” commercial devices, which would require a more generalised and scalable design when compared to current proofs of concept.

Following the findings within this literature review, the next Chapter covers the methodology, which describes the design of the research process for this work.

Chapter 3

Methodology

3.1 Introduction

With the rise of ransomware as a cybersecurity threat, there has been a significant effort made by the scientific community to study, analyse and mitigate it. However, despite the rise in the adoption of IoT, and IoT devices being generally considered much less secure than their desktop counterparts [71], there has been limited research regarding IoT's vulnerability to ransomware.

This research focuses on the use of experimentation and generation of proofs of concept, based upon traditional ransomware implementations that have previously been used to target desktops. This approach allowed a more accurate assessment of the viability of IoT-based ransomware to be performed, as empirical evidence was provided to support the proposed ransom techniques. An analytical approach was initially considered, but some key challenges (Discussed below in Section 3.2) would have limited its effectiveness.

In this Chapter, the challenges that are presented when attempting to research IoT-based ransomware will be explored, and the steps taken to perform this work will be justified, followed by an overview of the research process.

3.2 Research Challenges

In this section, the challenges encountered when researching IoT-based ransomware will be covered, including how they influenced the research plan and the steps taken to mitigate their effects.

3.2.1 IoT Device Variance

There is a significant degree of variance between IoT devices, as they are often designed with different purposes, users, and price points in mind. As such, this may prevent certain questions, such as whether IoT-based ransomware is viable, from being definitively answered with a simple “yes” or “no”. Instead, the *extent* of IoT-ransomware’s viability must be determined.

Therefore, in order to provide beneficial conclusions from this research, some conditions must be met.

Experiments must be reproducible in a real-world context. Testing on devices specifically designed to “emulate” a real-world device – such as a raspberry pi with an intentionally vulnerable test application – will likely misrepresent the security of the average IoT device.

Instead, the most logical option would be to perform tests on IoT devices that are designed to be used “in the wild”. Successfully deploying ransomware on such devices would imply that it could also affect real-world users.

Impact must be taken into consideration. Work that would only apply to relatively unknown devices, or a limited number of users is unlikely to be useful. For example, an attack that only affects a specific type or brand of device would limit its potential impact, and may not be considered a viable investment for attackers attempting to deploy ransomware.

Therefore, any techniques that are created should aim to be “generalisable”, such that they are compatible with as many different devices as possible.

3.2.2 Availability of Malware Samples

To analyse the behaviour of certain types of malware, researchers often study existing malware samples. This approach provides a plethora of research avenues, such as how the malware impacts its victims, or how it is able to spread to other

devices. A larger scope of research can be achieved by performing analysis on multiple families of similar malware. If researchers are able to obtain sufficient samples of the malware type in question, this can lead to the discovery of common traits, which can then be leveraged to produce countermeasures and protect end-users, and it may even be possible to test the effectiveness of said countermeasures in a semi-realistic environment [64].

With IoT-based ransomware, however, while there are some proof of concept malware variants built for IoT devices [252] (which is covered in more detail in Section 5.5), they typically target very specific devices and often do not consider “real world” applicability. The specificity and limited availability of this type of malware would reduce the efficacy of an analytical approach.

3.2.3 Summary

Considering these challenges, an analysis of existing IoT-based ransomware was not considered to be feasible, and attempting to predict the actions of “theoretical” IoT-based ransomware may have produced misleading results. Therefore, a more practical approach was chosen, such that the viability of IoT-based ransomware could be empirically proven. This required the exploration and development of techniques that could be used by IoT-based ransomware.

A research plan was then developed with these challenges in mind, which is detailed below. But first, let us examine the threat model associated with IoT ransomware.

3.3 Threat Model

During this work, various possibilities as to how IoT-based ransomware may be implemented were explored. As part of this, a number of assumptions concerning the threats, motivation and attacker capabilities were made, which are detailed below.

3.3.1 Ransomware Targets

The targeted IoT devices are assumed to be accessible by the attacker, such that they can be exploited. Often, attacks of this type would be performed over the

Internet, but they could also be performed by attackers on the same local network, or with physical access to the device.

The device is also assumed to be vulnerable such that arbitrary code execution can be achieved by the attacker. Some examples of such exploits would be buffer overflows or command injection. It should be noted that some vulnerabilities or exploits, such as denial of service attacks, would not be adequate for implementing ransomware on a target IoT device.

It is also assumed that once code execution is achieved, the attacker will have the necessary permissions to perform the actions required by the ransom method, such as access to storage devices, peripherals, and private files on the device. Given that most IoT devices run their services as “root” [39], this is a relatively safe assumption to make.

3.3.2 Attacker Capabilities

Attackers are assumed to have a basic understanding of Linux, embedded systems, and cryptocurrency. They are also assumed to be able to exploit IoT devices and are capable of programming and installing custom software that will be able to run on the targeted device. In addition, for the methods discussed in Chapter 5, the attacker is assumed to have a basic knowledge of partitioning systems and cryptography, and for the methods discussed in Chapter 7, the attacker is assumed to understand the usage of machine learning tools or online cloud services.

3.3.3 Attacker Goals

The assumed goal of the attacker is to infect IoT devices with ransomware, with the intention of economic benefit. By performing a successful ransomware infection, attackers will be able to profit via payments made by victims (usually with cryptocurrency).

3.3.4 Research Scope

Some methods of infection are considered out of scope for this work, such as the implementation of hardware based trojans, or supply chain attacks. The methods defined in this work could also potentially be used for other malicious purposes,

such as mass bricking attacks (Chapter 5) or spyware (Chapter 7). Although these attacks may be possible using these methods, they are considered out of scope, and will not be explored in detail within this thesis.

3.4 Research Plan

As previously mentioned, the limited availability of appropriate samples of IoT-based ransomware may prevent an effective analysis from being performed. Essential components for successful ransomware can be identified by analysing traditional ransomware that targets desktops, but such an analysis would not give much insight as to the components' compatibility with IoT-based platforms.

By attempting to convert these components for use in various IoT device environments, the overall viability of IoT-based ransomware can be assessed. This can also be used to highlight any “roadblocks” that ransomware developers may encounter when confronted with the limitations of IoT devices. Attempting to overcome or circumvent these roadblocks can allow predictions to be made as to the techniques attackers are likely to use in the future. Further, it would be possible to use these attempts to develop proof of concept ransomware for use on various types of IoT device.

Developing a proof of concept would assist in determining the scope of IoT-based ransomware, and provide a more accurate prediction as to its impact if used in the “real world”. Possible measures of an IoT-based ransomware's impact could include the number of viable targets, the monetary value of ransomable assets, or the likelihood of a payment being made by victims.

Acting from the perspective of a malware author in this manner presents a rare opportunity to be “ahead” of attackers in what is normally a defensive field of work. Often, cyber security researchers find themselves in the position of defending against the latest attacks in a very fast-moving industry. Instead, the approach taken here puts us in the position of identifying and preventing dangerous techniques from being used before they have a chance to be implemented. Flaws or potential weaknesses identified when developing the aforementioned ransomware components can contribute to the development of countermeasures, and proofs of concept can be used to create test-beds where such countermeasures can be evaluated.

3.5 Overall Process

To assess the viability of IoT-based ransomware, the “essential” components used within traditional ransomware were identified and compared against the limitations presented by IoT devices. From this, multiple components were identified that, if possible to implement, would increase the viability of IoT-based ransomware.

These components included:

- **Communication:** A method of communication with the user, such that a ransom note could be transferred.
- **Asset control:** A method that would allow an attacker to restrict users’ control of infected IoT devices.
- **Persistence:** The use of techniques that would allow an attacker to retain control of IoT devices if they are rebooted.
- **Privacy Invasion:** An alternative method of ransoming users via IoT devices by invading the privacy of the devices’ owners.

The overall research process is shown in Figure 3.1. One of the primary aims of this research is to assess the viability of ransomware. This can be achieved by creating proofs of concepts for each of the identified components. If a component is successfully implemented, it can then be used to assist in ascertaining potential countermeasures, and act as a measure of its possible impact.

3.6 Conclusions

In this Chapter, the challenges that would be encountered when attempting to research IoT-based ransomware were identified and discussed. A research plan was then produced per the previous constraints. Finally, the overall process was examined and justified.

The following Chapters document the process of attempting to implement the previously defined components, the issues encountered during development, the potential impacts of any findings, and possible remediation strategies.

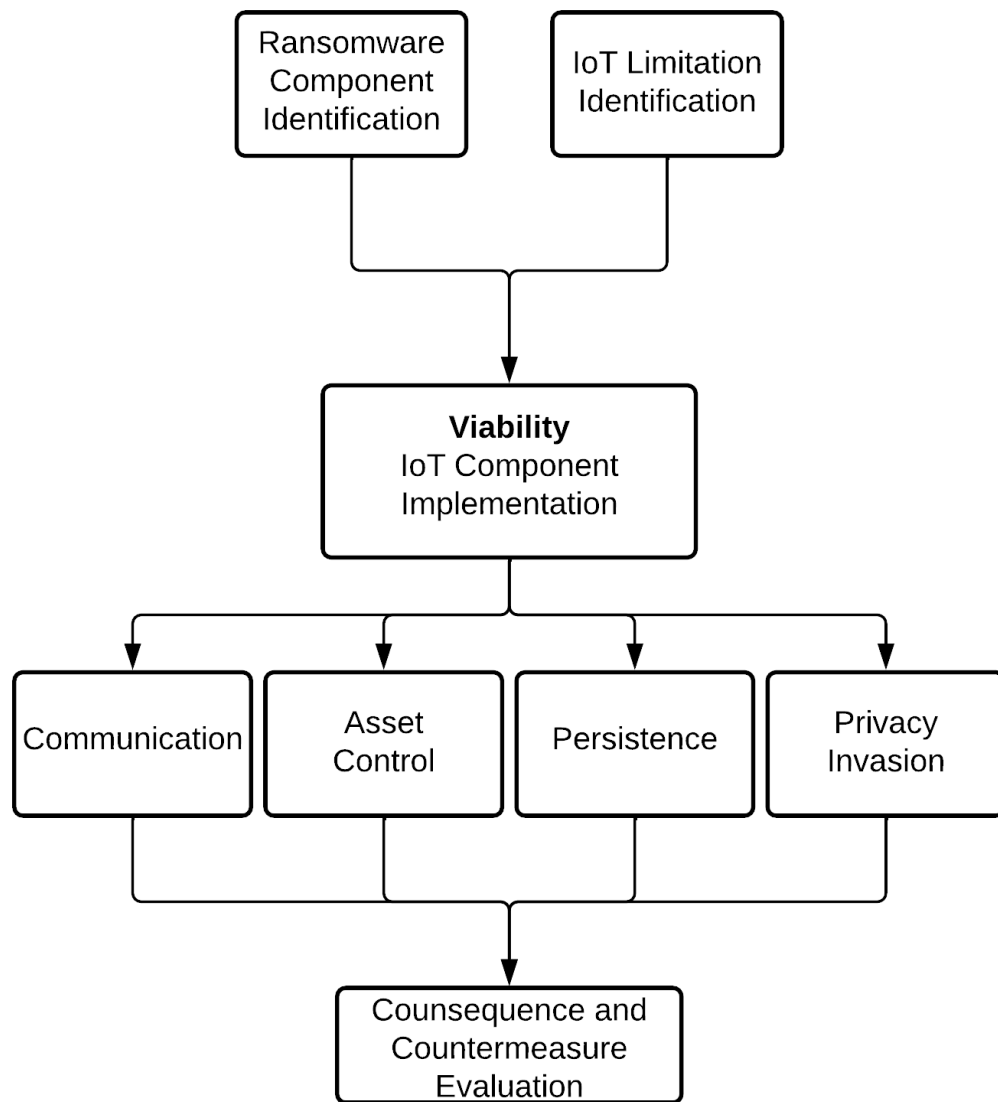


Figure 3.1: Research Process Graph

Chapter 4

Challenges in Delivering IoT Ransom Notes

Based on content of previous publication:

“PaperW8: An IoT Bricking Ransomware Proof of Concept” [41]

4.1 Introduction

It is often in a malware author’s best interest to keep their malware from being detected on the devices that they infect, as it allows them to continue their attacks, retain control, or prevent victims from attempting to remove the infection. For example, malware which directs infected devices to participate in DDoS attacks, and cryptominers that force devices to mine cryptocurrency for the malware author, are most profitable if they can run without interference. However, if the victim notices a significant impact on the device’s performance, it will increase the likelihood of the malware being detected and removed.

Ransomware, on the other hand, only needs to avoid detection during the initial encryption or locking stages. After ransomware gains control of the victim’s assets, it becomes necessary for it to notify the victim of its existence, typically via a ransom note [90]. The purpose of the ransom note is to communicate with the victim, threatening them with the loss of their device or files unless a payment is made. Generally, the note will include information as to what has happened to the victim’s assets, instructions as to how to pay the ransom, and a timer for when the payment must be completed [284]. Without providing a ransom note, the attacker

has no means to extract payment from their victims. Thus, communication with the victim is an essential component of successful ransomware attacks.

In this Chapter, the challenges encountered when attempting to communicate with victims via infected IoT devices are discussed, followed by potential methods that could be used by attackers.

4.2 Desktop-Based Ransom Notes



Figure 4.1: WannaCry Ransom Note¹

In a ransomware attack, ransom notes, such as the one shown above in Figure 4.1, are the main method used by attackers to communicate with their victims. Effective ransom notes typically contain an explanation as to what has been encrypted or locked, in such a manner that any victim, regardless of their technical

¹Image source: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/05/07175133/wannacry_05.png

ability, should be able to understand [256; 261; 68].

An agreement to return the assets that have been ransomed is often included, upon the condition that the victim makes a payment of a pre-determined amount. Many families of ransomware also employ other techniques, such as timers, to further pressurise victims into making a payment [284; 14]. The expiry of these timers could increase the ransom price [248; 233], delete random files [2], or prevent users from recovering their assets entirely [123].

Payments are typically expected to be made in the form of cryptocurrency, but there have been cases of payments being made using gift cards [3] or other payment services such as “MoneyPak” [152].

Displaying ransom notes to victims is relatively simple for traditional ransomware that targets desktop computers or laptops, as there is a standard method of interaction: the Graphical User Interface (GUI) displayed on attached monitors or a built-in screen. As the attacker is likely to know (or obtain knowledge of) the operating system that they are infecting at runtime, it is relatively simple to use an appropriate method to display a message as a window or image on the screen.

Optionally, the attacker may add interactive elements to the note, which the victim can use to gather further information or contact support [86]. Infected IoT devices, however, are unlikely to include such a display, which would prevent a standardised method of communication from being used.

4.3 IoT-Based Ransom Notes

The IoT includes a vast number of devices used for various purposes. As might be expected, devices will be equipped with different methods of communication that are affordable and fit for purpose.

As an example, a Digital Video Recorder (DVR) would be expected to provide the user with a video feed via a connected screen as the main method of communication, whereas a voice-based virtual assistant would be expected to respond to voice commands via a speaker. IoT devices may also be used to simply transfer information rather than interacting with the user directly, such as routers supplying Internet connections to other networked devices. Finally, “out-of-band” communication, such as email alert systems [238; 51], may be utilised by devices to remotely provide updates to their users.

The lack of standardisation between these methods may prove challenging for an attacker, as the methods of communicating with the victim will be almost completely dependent on the device’s output or peripherals. Malware authors may need to be creative in order to communicate, using different communication channels that are available to them in a way that the victim may easily discover and understand what the attacker is attempting to convey. As such, several methods of communication could be used – depending on the target of the ransom – as outlined below.

4.3.1 Communication Hijacking

If an attacker is able to compromise an IoT device and is attempting to send a ransom note to its user, they will have to utilise the information and peripherals made available to them to communicate. As such, the first step is to perform reconnaissance on the device to determine which peripherals the device utilises for its typical operations.

After identifying the methods with which the device normally communicates, the next stage is to determine if they can be used to effectively transfer a ransom note. Peripherals with a low rate of transfer or those that may not be noticed, such as LEDs or number displays, are unlikely to be able to effectively transfer the required information to a victim.

If a channel is identified that can be used to transfer a ransom note effectively, and the ransomware has managed to gain control over assets the device provides, the first step is to kill any processes that are currently using that channel. This will allow the ransom note to be transmitted, while effectively preventing the developer or consumer from using these channels for their intended purpose. Locking avenues of interaction with the device will also reduce the possible attack surface, decreasing the likelihood of the victim being able to regain control of the device.

Once control over the channel has been established, it can be used to display a ransom note to the victim. If the hijacked channel was used as one of the main methods to interact with the device, it can be expected that the victim would encounter the ransom note when they notice that the device is no longer functioning as intended. It should also be noted that the “display” of the ransom note does not have to be confined to a single communication channel. If multiple

channels are hijacked, those that are less “user friendly” can direct victims to the other more accessible means of communication to describe the terms of the ransom note more effectively.

It should further be noted that if a victim’s email address or phone number is stored on the device, out-of-band communication channels could also be used by an attacker to communicate with the victim. For example, a ransom note could be sent “out-of-band” to an IoT device’s owner, after the ransomware takes control of the device.

4.3.2 Common Communication Channels

To test the viability of hijacking communication channels, some common channels used by Linux-based IoT devices to communicate with users were identified. Potential methods to hijack these channels were then explored, with the aim of sending a ransom note to the “victim”.

Linux-based IoT devices were chosen as the testing targets. The reasons for this are outlined below:

- Linux was one of the most widely adopted operating systems for IoT at the time of the research [83].
- Linux-based devices had been previously targeted by many types of IoT-based malware [266].
- The resources required to run an OS of this complexity would imply more expensive IoT devices, which may allow a more costly ransom to be requested.

It should be noted that this approach does not necessarily limit these techniques to only Linux-based devices. Theoretically, if the attacker can gain sufficient control of a device using a different operating system, the attacker may still be able to use the theories outlined in this work to make similar modifications to the communication channels that the target operating system manages.

Below, some of the common communication channels – that were identified during the investigation of various Linux-based IoT devices – are explored.

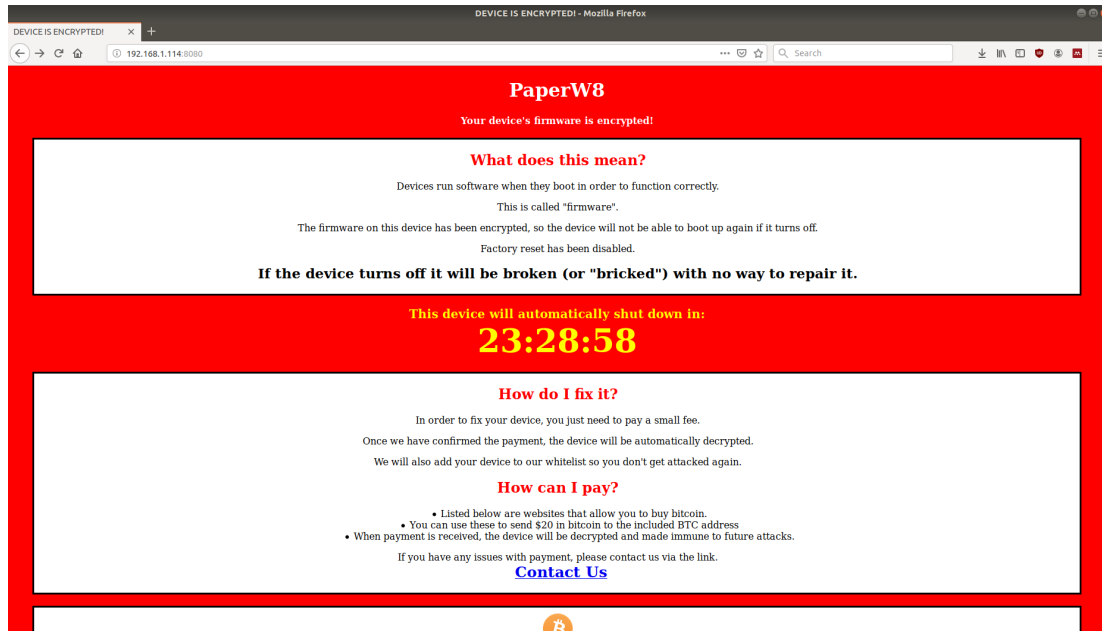


Figure 4.2: HTTP Hijacking Ransom Note

4.3.2.1 HTTP Web Servers

Many devices utilise an HTTP service as a setup and interaction portal, which users can access by visiting the device's IP address via a web browser. From here, users would be able to configure, interact with, or update the device. This method of communication gives a significant degree of freedom as to how information can be presented, and supports the use of interactive elements, if required.

If this communication method were to be hijacked by an attacker, this level of adaptability would allow the attacker to display a ransom note with a significant amount of information. The note could even link to additional resources, such as methods of payment or technical support. An example ransom note is shown in Figure 4.2.

Hijacking this communication channel is likely to also remove a significant amount of control from the device's user, preventing them from performing actions that are potentially detrimental to the attacker, such as updating the device to a more secure version of the firmware.

4.3.2.2 Domain Name Server (DNS)

Some IoT devices, such as routers, provide DNS services to other devices on the network. If such a device is infected, an attacker may modify the DNS server's configuration to maliciously impact the network.

As an example, DNS results could be modified, such that they redirect victims to servers under the attacker's control, creating a malicious captive portal. This could be used to redirect HTTP requests or prevent victims from accessing the Internet entirely. However, this does require a secondary method of communication to be made available, such that there is something to be redirected *to*. This could be a web service hosted by the device itself or provided externally by the attacker.

However, it should be noted that when encountering encrypted or verified protocols, this technique may not be as effective. HTTPS requests, for example, are likely to fail, as when the victims are redirected to the attacker's service, it will not have the expected certificate of the original service to prove its validity. As such, most browsers will present a warning and prevent the website from being opened, which will also prevent the ransom note from being shown to the victim.

4.3.2.3 Graphical User Interfaces

Devices may use a GUI, as this often provides an easy method of quickly conveying information, especially for devices that make use of multimedia components, such as entertainment systems.

This does require a screen for the interface to be displayed on, which can either be internal, such as a touch screen built into the housing of the device, or external, such as with an attached monitor. If an attacker is able to gain control of this channel, they would be able to modify the output to display a ransom note. A previous example of this style of hijacking was performed against a smart thermometer to control an attached screen [252].

The process of hijacking such a communication channel may be complicated by the capabilities of the attached screen, such as the size of the display. While also hijacking the method of interaction may theoretically be possible, it is likely to vary depending on the device type. For example, interaction may be expected to be performed via a touch screen on one device, while a mouse may be expected

on another. As such, including a method of interacting with a GUI-based ransom note is unlikely to be worth an attacker’s time. Instead, if the information that can be conveyed via a ransom note display is limited, attackers will likely attempt to circumvent the issue by directing the victim to access further information via another resource or hijacked channel which has no such limitation.

4.3.2.4 Audio

Some devices, such as virtual assistants, use a speaker to communicate with users. As the audio played is likely to be managed by the IoT device, it may be possible for attackers to play pre-recorded audio, or even speak directly to victims to convey a ransom note.

Vulnerable cameras exposed to the Internet have been known to be exploited by attackers to play various audio clips [258]. In 2019, an attacker was recorded using an Internet-connected camera’s speaker to demand bitcoin from the device’s owners [1; 47] to prevent further harassment. However, as the attacker demanded a ransom that exceeded \$400,000 USD, it can be assumed that this was not a serious ransom attempt.

4.4 Proofs of Concept

The next stage was to attempt to hijack the communication channels of “real” IoT devices. Multiple IoT devices of varying brands and purposes were selected for testing, such that different communication channels would be present, and to prevent the “overfitting” of techniques.

Below, the exploitation stages and implementation of the aforementioned techniques for each of the devices that were selected are shown.

4.4.1 Device Analysis

To infect and subsequently hijack a test device, an “attacker” must first gain control of the device. This can be achieved in many different ways, such as:

- Accessing exposed telnet ports (or similar services) with no authentication or a weak/known password.

- Using backdoors or debug interfaces that were not intended to be discovered or accessed outside of a development environment.
- Exploiting vulnerabilities in services provided by the IoT device.

Ideally, this would allow the attacker to interact with the device via the use of remote shell commands. After gaining access to a device, further analysis can be performed, such as discovering potential communication methods, exploring potentially valuable files, and determining the available storage space.

However, to save space (and therefore limit the cost of production), IoT developers often only install the bare minimum utilities that the device requires to function. Therefore, some utilities that may be required by the attacker (such as `tftp` or `wget` to download the next stage of the attack) may not be available on the device. To solve this issue, it is possible to use a utility called “Busybox” to supplement any missing functionality.

4.4.1.1 BusyBox

“BusyBox” is a tool that provides many common Linux utilities in a single small executable [9]. This is often found on Linux-based IoT devices where these utilities are required, but space is a limited resource.

BusyBox is designed to be very adaptable and can be configured before compilation to suit the IoT developer’s needs. For example, a subset of commands can be selected such that only those that are required by the device are compiled into the resulting binary to save space. As BusyBox is open-source, anyone can compile their own version of BusyBox to fit their needs.

To investigate the plethora of IoT devices more easily, a custom version of BusyBox was required that included useful utilities that IoT developers may not incorporate into production devices, such as `tftp` for file transfers, or `telnetd`, which can be used to start a telnet service. A list of currently available utilities is available as part of the BusyBox documentation [267].

After setting up a new configuration, BusyBox could be cross-compiled² to build a binary for any targeted device architecture.

²Cross-compilation is the act of compiling code for a platform that has a different architecture to the “host”, such as an x86 system compiling a MIPS binary.

4.4.2 Malware Loader

After the initial analysis of the device, an attacker needs to be able to upload any dependencies required to effectively transmit a ransom note to the victim.

As part of this work, a Python based “malware loader” was created that could be used to perform many of the repetitive tasks required for the testing process, such as exploitation of the device and uploading malware resources.

Uploading data to an exploited device is a non-trivial task, as common file transfer utilities that are often present on desktops systems are not guaranteed to be available on the target IoT device. To solve this problem, two techniques that could be used to upload data to devices with limited functionality were implemented into the malware loader, which are discussed below.

4.4.2.1 Trivial File Transfer Protocol (TFTP)

During this work, the `tftp` utility, which allows users to upload or download files from TFTP servers, was found to be pre-installed on many of the test devices. Uploading a file to a target device using this method requires two steps.

First, the loader must start a TFTP server on the “attacking” machine, which will host any files that need to be transferred. Next, the target device is forced to run the `tftp` utility to download the required files into temporary memory. An example command that would be run on a targeted device is shown below.

```
1 # Where -l is local file, and -r is remote file name.  
2 tftp -l /tmp/file.txt -r file.txt -g <tftp server IP>;
```

Listing 4.1: Example TFTP command

If the `tftp` utility is not available on the device, an alternative method called “echoloading” is used.

4.4.2.2 Echoloading

Most Linux shells (such as Bash [93]) allow users to change the destination of a command’s output via “redirection”. For example, the output of a command can be saved to the filesystem by redirecting the output to a file, as shown below in Listing 4.2. This feature can be used by the loader to “re-create” files on remote devices.

```
1 # Putting the output of an "ls" command into a file named  
   "example".  
2 ls > example;
```

Listing 4.2: Example File Redirection

Most devices include a utility program named “**echo**”, which prints any arguments that are provided to it [40]. Using the aforementioned “redirection” feature, the loader can force the device to run multiple **echo** commands and append the output to a file. To transfer files from the “host” to the target device, the loader converts the file for transfer into shell compatible **echo** commands, which can then run on the device to remotely reconstruct the file in temporary memory.

Some files, such as executables, are likely to contain “unprintable” characters that would produce unexpected results if used in a shell command, such as new-lines or null characters. This issue can be circumvented with the use of escape characters, which can be enabled when running the **echo** utility by setting the “-e” flag. This allows the attacker to use hexadecimal notation (`\xnn`), rather than the typical “ASCII-only” input [40]. Using this notation, non-printable characters can be written to the output file without impacting the execution of the command.

During this work, a previous implementation of echoloading which was used to load cryptomining malware to Internet-connected DVRs [143] was discovered. After downloading and studying the source code, a modified version that could be implemented by the malware loader was created. When echoloading is chosen as the method of upload, the loader splits the chosen file into 50-byte³ hexadecimal encoded “chunks” which can then be recreated via remote **echo** calls.

The Listing 4.3 shows an example of the commands that could be run to recreate a small text file using the **echo** utility. At the end of each command, another **echo** call is made, which outputs the word “DONE” to the shell. This allows the uploader to confirm that the **echo** utility supports decoding hexadecimal notation and that the device has completed the previous command and is ready to receive more data.

³Some IoT devices limit the length of commands that can be given via a remote shell, hence the 50-byte limit.

```

1 # A file is created at /tmp/a.txt on the device.
2 # Text file contents: "This is a test file used to
   demonstrate how files can be transferred using
   echoloading."
3 echo -ne '\x54\x68\x69\x73\x20\x69\x73\x20\x61\x20\x74\x
   65\x73\x74\x20\x66\x69\x6c\x65\x20\x75\x73\x65\x64\x
   20\x74\x6f\x20\x64\x65\x6d\x6f\x6e\x73\x74\x72\x61\x
   74\x65\x20\x68\x6f\x77\x20\x66\x69\x6c\x65\x73\x20\x
   63' >> /tmp/a.txt && echo -e '\x64\x6f\x6e\x65'
4 echo -ne '\x61\x6e\x20\x62\x65\x20\x74\x72\x61\x6e\x73\x
   66\x65\x72\x72\x65\x64\x20\x75\x73\x69\x6e\x67\x20\x
   65\x63\x68\x6f\x6c\x6f\x61\x64\x69\x6e\x67\x2e\x0a'
   >> /tmp/a.txt && echo -e '\x64\x6f\x6e\x65'

```

Listing 4.3: Recreating a small text file using echo commands

While this technique could be used to upload the dependencies directly, echoloading is not very efficient, as it may require thousands of commands to fully upload large files. Instead, this technique can be used to “bootstrap” the rest of the upload process by uploading a small, but more efficient, application that can then download other dependencies.

Ideally, the “bootstrapped” application should be as small as possible to maximise efficiency. As an example, the IoT malware family Mirai would upload a small utility approximately one kilobyte in size if no other upload methods were available, which could then be used to continue the infection process [136]. For the purposes of this work, the loader would upload a `tftp` client, which could then be used to download any other dependencies.

4.4.2.3 Alternative Utilities

While additional methods were unnecessary for this work, if other common utilities such as `wget`, `curl` or `scp` are found to be present on a target device, attackers could use these in a similar fashion to the `tftp` utility to download malware from their server, as evidenced by previous existing IoT malware [175].

4.4.3 Hijack Modules

The available communication channels will differ from device to device. As creating bespoke methods for each encountered IoT device is infeasible, in this work, hijacking methods for the most common communication channels were packaged into executable “modules”.

Each module was designed to be generic, such that they could function independently from one another, and run on any device with minimal modification. When necessary, binary portions of the created modules were cross-compiled to be compatible with CPU architectures that were most commonly used by IoT devices, such as ARM [15] and MIPSel [179].

Once a device had been analysed, it was added to the malware loader such that it could be automatically exploited. The appropriate modules could then be uploaded and executed to hijack any relevant communication channels.

The functionality of each of these modules is shown below.

4.4.3.1 HTTP

The most common method of communication encountered on the devices that were tested was the use of an HTTP server. While the method was common, the implementation varied, with some brands, such as D-Link, developing their own custom applications to serve HTTP traffic to the device. Instead of adapting a ransom note to be compatible with each implementation, it was simpler to include a minimal HTTP server in the upload stage instead.

As mentioned in Section 4.4.1.1, “BusyBox” is a tool that provides many common Linux utilities in a single small executable [9]. One such utility is `httpd`, a small HTTP daemon that can be used to host webpages. By cross-compiling a version of BusyBox containing this compatible utility, the target device could be forced to host webpages with custom content.

A webpage hosting a ransom note was created for testing purposes, which included a timer, a fake bitcoin address and further instructions that fit the theme of a typical ransom note. The website and all its resources were packaged into a single archive for easy transmission to any infected devices.

By killing existing HTTP services provided by the device, the attacker can replace the website hosted on the vacant ports. The uploaded `httpd` utility can

be configured to host web pages from a certain directory, such that when the victim then attempts to connect to the device via a browser, instead of being greeted by a configuration or settings page, they are presented with the attacker’s ransom note, as shown in Figure 4.2, for example.

4.4.3.2 DNS

Devices that provide Internet access to other devices on the network, such as routers, are sometimes tasked with providing DNS services to clients that connect via DHCP. During this research, an application called “`dnsmasq`”, was encountered, which can be used to “provide DNS services to a small-scale network” [144]. By running `dnsmasq` with the argument `--address=#[TARGET IP]`, a rule can be created to resolve all DNS requests to a chosen IP address. This could be used by an attacker to perform a DNS poisoning attack to redirect internet traffic to the infected device, preventing users from accessing the Internet. IoT devices that use other applications to implement their DNS services could be similarly impacted if the attacker is able to modify the tool’s configuration or replace the tool with a similar service, such as `dnsmasq`.

4.4.3.3 Framebuffer

Developers may create custom applications to display a GUI on attached screens, which users may be able to interact with. On Linux, attached screens can be accessed via the Framebuffer [263].

The framebuffer is an abstraction layer that can be used to interact with video hardware. Framebuffer devices can be accessed by users and applications via files within the `/dev` directory. For example, the `/dev/fb0` file can be used to access the first framebuffer device. Directly interacting with these devices can be relatively complex, but several libraries have been designed for use on embedded devices, such as LittlevGL [169], an open-source library for building GUIs.

For this hijacking method, an application was created to interact with the framebuffer of infected devices using a modified version of the LittlevGL library. When the application is run, it displays a ransom note, including a simple description and countdown timer, on a screen attached to the device. The ransom note can also direct the victim to another source for further information. For example,

the attacker could implement the HTTP module to display any information that could not be conveyed via the framebuffer due to limited screen space, which can then be accessed via the local IP address of the infected device.

Device Name	Exploitation			Hijacking Method		
	Architecture	Exploit	Number of Exposed Devices ¹	Web-server	DNS	Frame-buffer
HG532 Router	Mipsel	CVE-2017-17215	Unknown	✓	✓	N/A
R6250 Router	Arm	CVE-2016-6277	850 ²	✓	✓	N/A
MVPower DVR	Armv5l	CVE-2016-20016	94,171	✓	✗	✓
WiPG-1000 Presenter	Armv5l	CVE-2019-3929	Unknown	✓	✗	✓
5020L Camera	Mipsel	CVE-2019-10999	73,533 ³	✓	✗	N/A
932L Camera	Mipsel	CVE-2019-10999	90,359	✓	✗	N/A

Table 4.1: Hijacking experiments performed on various IoT devices.

¹ The number of devices that were publicly exposed to the internet, checked via Shodan on the 9th December 2019.

² Checked via Shodan on the 26th March 2020.

³ Checked via Shodan on the 18th February 2020.

4.4.4 Device investigation

Devices that had been previously targeted by IoT malware were prioritised as potential targets for investigation, as they were more likely to have well-understood vulnerabilities, which could be exploited to gain access to the device and perform the required tests. Exploiting previously targeted devices also demonstrates that the suggested methods could be used to attack popular IoT devices in a practical setting.

Below, the processes for exploiting and hijacking the communication methods of various IoT devices are shown. A summary of the devices used in these experiments is shown in Table 4.1. Further details concerning any patches to the highlighted vulnerabilities are available in Appendix B.

4.4.4.1 HG532 TalkTalk Router

The HG532 router is a popular device built by Huawei that is sold in a multiple countries. Testing was performed on a device distributed by the UK based Internet Service Provider, ‘Talk-Talk’.



Figure 4.3: HG532 Router

Exploitation

The HG532 router had previously been found to be vulnerable to a remote code execution attack via its UPnP service on port 37215 [195].

By sending a crafted “upgrade” XML message, a command injection attack could be performed, allowing attackers to remotely run custom shell commands. Code for the exploit was available online [17], which was adapted for use with the malware loader.

Communication Hijacking

This device primarily communicated with users via two methods:

- Providing Internet access to connected devices
- A local HTTP server

To hijack these channels, the various HTTP hijacking dependencies (including an `httpd` binary and an archive containing the website’s source code) were uploaded to the device via the `tftp` utility found natively on the device.

An application running on the router was found to be responsible for serving the original web service (which the user could access to make configuration changes), and the vulnerable UPnP service. The application was killed to shut

down both services, preventing the victim from re-configuring the device, and stopping others from being able to gain shell access via the same exploit. Port 80, the default port used for HTTP traffic, was also freed for use by other processes.

The DNS service was then reconfigured to redirect all requests to the router's IP address, creating a captive portal for devices on the victim's network.

Finally, the previously uploaded archive containing the website based ransom note was extracted into temporary memory, and the uploaded `httpd` binary was run to host it on port 80. From then on, any device connected to the router that attempted to make an HTTP request was redirected to the ransom note hosted on the router.

4.4.4.2 R6250 Netgear Router

The R6250 is a router produced by Netgear that was first made available in 2013.



Figure 4.4: Netgear R6250 Router

Exploitation

The R6250 router, along with several other similar models, was found to be vulnerable to a command injection attack via its web service [194]. Attackers could achieve code execution by including a shell command within a crafted request, as shown below in Listing 4.4.

```
http://192.168.1.1/cgi-bin/;<SHELL_COMMAND>
```

Listing 4.4: R6250 Command Injection [158]

On the 26th of March 2020, Shodan [225], a search engine that indexes Internet-connected devices, was used to determine the number of potentially vulnerable

devices exposed to the Internet. The listing showed 855 accessible R6250 devices, with the total number of vulnerable routers numbering approximately 19,470.

Communication Hijacking

This device used methods similar to the HG532 to communicate. However, as it was built using a different CPU architecture, a different version of the hijacking modules had to be compiled and uploaded. After exploiting the device, it was found that the router used a native version of the `httpd` utility to host its web service.

After killing the original `httpd` instance, no other modifications were required, and the DNS and HTTP services were successfully hijacked without any further issues.

4.4.4.3 TV-7104HE MVPower DVR

Internet-connected DVRs, such as the TV-7104HE DVR built by MVPower, allow users to remotely access and record the video feed of multiple networked cameras. As of the 9th of December 2019, Shodan listed approximately 94,170 MVPower devices as publicly accessible from the Internet [224].



Figure 4.5: TV-7104HE MVPower DVR

Exploitation

These DVRs exhibited an undocumented “feature”, which some described as a

backdoor [57], allowing unauthenticated attackers to access a root shell via the web server. By providing a shell command in the HTTP GET parameter to the `/shell` path on the DVR's web service, attackers could run arbitrary shell commands and gain control of the device [251].

Communication Hijacking

A number of communication methods used by the device were identified, such as:

- A web server hosted on port 80, which allowed the user to make basic configuration changes.
- A graphical user interface displayed via an external screen, such as a monitor. The device provided multiple outputs, such as VGA and composite video ports. During this work, a screen was connected via the VGA port.

The GUI displayed the current state of the device and a live feed of any connected cameras. Users could interact with the GUI by using a USB mouse and keyboard.

- An infrared remote (supplied with the DVR), which could be used to control the device. As this was a one-way communication channel from the user to the device, it was not beneficial to hijack or replace it.

For this device, the web server and framebuffer modules were chosen to transmit the ransom note. As with the previous devices, the required modules were uploaded to the device in preparation for hijacking each of the channels. However, despite killing the original web server, it was not possible to host a new web server on the original port⁴.

As a workaround, the ransom note was initially hosted on port 8080, which is often used as the alternative port for HTTP connections. However, this was not ideal, as alternative port numbers need to be included in the URL in order to be accessed. Unless the victim was redirected to the website-based ransom note by an alternative communication method, it would be incredibly unlikely for the ransom note to be discovered “naturally” by the victim.

⁴It was eventually discovered that due to the nature of the exploit, any processes that are spawned via the command injection would be spawned as a child process of the web service. In Linux, child processes inherit the “file descriptors” of the parent process. As open sockets are also assigned a “file descriptor” [146] when created by a process, port 80 remained “occupied”, preventing its reuse.

Therefore, it was deemed necessary to return to hosting the ransom note on port 80, increasing the likelihood of its discovery. To solve the issue of port 80 being occupied, a small bash script was run to iteratively close file descriptors opened by the current process, which in this case was the exploited web application. Sub-processes would then be able to populate the now vacant ports. The Bash script is included below in Listing 4.5.

```
1 for fd in $(ls /proc/$$/fd); do
2     # Skip standard file descriptors that won't interfere
3     # with port 80.
4     case "$fd" in
5         0|1|2|255)
6             ;;
7             *)
8                 # Close file descriptor
9                 eval "exec $fd>&-"
10                ;;
11        esac
12 done
```

Listing 4.5: File Descriptor Closing Script⁵

The next step was to attempt to hijack the device's GUI display. However, after killing the application responsible for running the DVR's GUI, (`dvr_gui`), it was discovered that a watchdog had been implemented as part of the device's design.

Watchdogs are commonly used to monitor the performance of the devices that they run on. If a watchdog timer is not "refreshed" during a certain period of time, the device may be reset [66]. The watchdog timer is normally managed by the main application of the device, which in this case, was the `dvr_gui` binary. The "resetting" process can vary depending on the device, but for some devices, as with the DVR, the reset process performs a full reboot, which removes any data stored in volatile memory, including any uploaded hijacking modules.

Previous IoT malware families, such as Mirai, endeavoured to disable such

⁵Code source:

<https://unix.stackexchange.com/questions/123413/close-all-file-descriptors-in-bash>

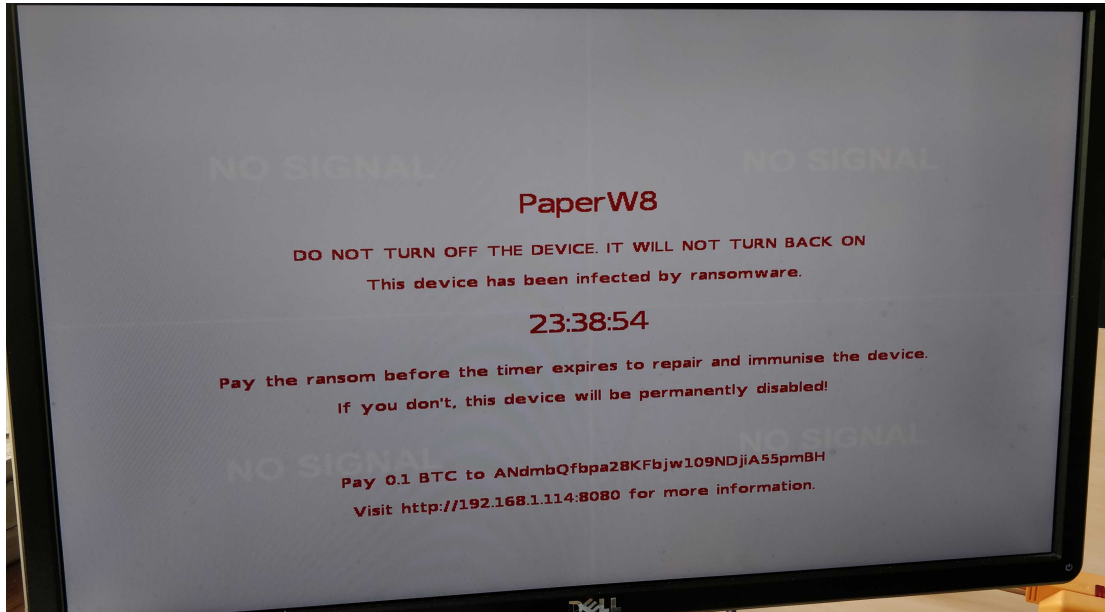


Figure 4.6: Hijacking the Framebuffer of the TV-7104HE MVPower DVR

watchdogs to prevent the device from rebooting [137]. For simplicity, the approach taken during this research was to run an instance of the BusyBox `watchdog` utility instead, which continually resets the watchdog timer. The main application, `dvr_app`, could then be killed, clearing the framebuffer from interference.

The uploaded framebuffer application was then run, which was able to communicate with the Linux framebuffer device at `/dev/fb0` and successfully display a ransom note on the attached screen, as shown in Figure 4.6.

4.4.4.4 WiPG-1000 Presenter

The WiPG-1000 is a network-accessible presenter produced by “WePresent” [30], which allows users to stream presentations over the network to connected screens or projectors. The device is relatively expensive, with some vendors having listed it for prices up to £399 GBP [24]. The 1600W model (which was also found to be vulnerable) had also been previously listed with prices of up to £1029 GBP [18].

Exploitation

The WiPG-1000 hosted a web service that allowed its users to configure the device via any connected browser. Unfortunately, this web service hosted an unused file named “`file_transfer.cgi`”, which was found to be vulnerable to command injection.



Figure 4.7: WiPG-1000 Presenter

This file could be targeted by attackers to gain remote code execution via a crafted POST request [197].

Communication Hijacking

The main methods of communication used by this device were identified as:

- Video output which could connect to external displays, such as projectors or monitors, via either VGA or HDMI. By default, any connected displays showed a “standby screen”, which included the IP address of the device and a quick guide as to how to use the presentation functionality.
- The WiPG-1000 also provided a web server, which the user could use to modify the configuration settings of the device.

Upon attempting to upload hijacking modules to the device, it was discovered that the `tftp` utility was not natively available. Instead, echoloading was used to upload an appropriate version of the utility from the attacking machine, which was then used to upload the HTTP and framebuffer modules.

After the upload was completed, the original services running the web server and GUI were killed, allowing both channels of communication to be hijacked without any further modifications required.

4.4.4.5 5020L and 932L D-Link Cameras

The D-Link 5020L and 932L cameras are produced for general surveillance use both indoors and outdoors. They can be accessed at their IP addresses through a web browser, or anywhere via the “mydlink” platform [73]. A 2016 report

from Shodan lists the 932L model as the most popular publicly accessible D-Link product, with the 5020L listed as the sixth most popular device [223].



Figure 4.8: D-Link camera models 5020L and 932L

Exploitation

To gain access, a vulnerability that was known to affect several D-Link devices – which implemented the “`alphapd`” web server application – was used. The vulnerability had been given a CVE ID of CVE-2019-10999 [196], and source code for an implementation of the exploit was available online [271]. However, the provided exploit was only compatible with certain firmware versions of the 5020L and 930L models. This did provide an easy method of exploiting the 5020L model, but an exploit for the 932L had yet to be developed. As such, a modified version of the exploit that would work with the required model and version was produced as part of this research⁶.

The exploit was a typical buffer overflow that allowed the attacker to overwrite the “return address” of a function called within the binary when a certain URL was

⁶It should be noted that the developers of the original proof of concept later extended their codebase to support some new models, including the 932L model. An overview of the similarities is given in Appendix A.

visited with too large an input. Modifying the return address allows the attacker to define the address of the next instruction to execute when the current function returns. This is still somewhat limiting, as the attacker can only use instructions that are already available as part of the application unless the attacker is able to inject their own instructions using shellcode.

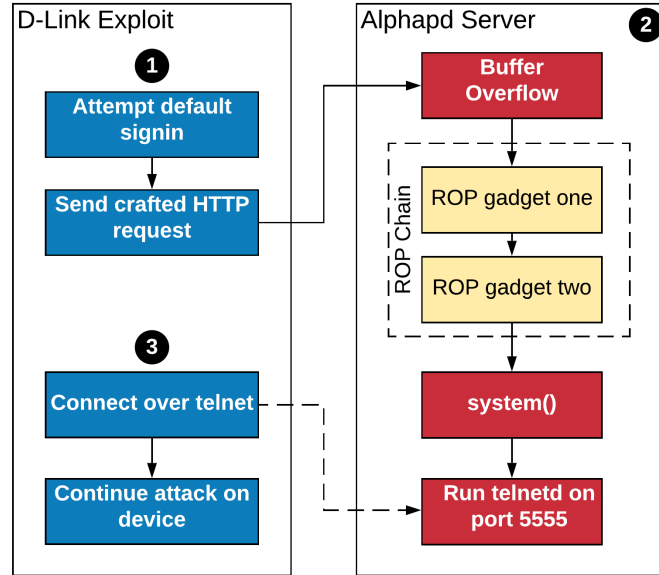


Figure 4.9: CVE-2019-10999 Exploit Structure

To circumvent this limitation, attackers can make use of “Return Oriented Programming” (ROP) [216]. By identifying small sections of existing assembly that end in a return instruction (known as “gadgets”), attackers can perform complex actions by constructing “ROP chains”. By “returning” to each gadget in the chain, attackers can implement complex actions by combining multiple sections of code that were already available in the binary.

This technique was used to exploit the D-Link camera via a buffer overflow that could be triggered with a crafted HTTP request, then jumping to two gadgets discovered within the binary to start a `telnetd` daemon. Remotely accessing the telnet service provided access to a shell running with root privileges. An overview of the exploit process is shown in Figure 4.9.

One limitation of this exploit is that the attacker must have access to an authenticated session on the camera’s web application to access the vulnerable webpage. For the purposes of this work, it was assumed the victim had not

changed the default admin password, which is blank by default. The use of a blank default password can be considered a grave security error, as even without performing this exploit, if the attacker can access this service, they would be able to view the camera output, change various configuration settings, or modify the device’s firmware.

Communication Hijacking

The main methods of communication for this device were identified as:

- A web server on port 80, through which users can view the live camera feed and configure the device.
- The “mydlink” platform and its associated access methods (e.g. web services, mobile apps), which supposedly could be used to access the camera from a distance. However, during this research, the platform did not function correctly and could not be accessed. It is assumed that, while it was not possible to confirm that this communication method could be hijacked, it is very likely that by killing processes on the device, this channel could be blocked, preventing the user from modifying the camera configuration or encouraging an investigation of the camera to be made. While hijacking the channel would have been preferable, creating such a method would require a significant time investment and would likely not be transferable to other devices.

For both devices, it was relatively simple to hijack the web server. As the exploit that was used to gain access to the camera would crash the main application upon completion, the attacker simply needs to upload and run the required HTTP module to hijack the web server and display a ransom note.

4.5 Conclusions

In this Chapter, *communication* as a requirement for successful ransomware attacks was discussed. Ransom notes used in the past by desktop-based ransomware were examined, and the methods used were compared to the limitations of IoT devices. The viability of identifying and hijacking communication channels present on IoT devices was examined, and the most common channels were assessed to determine their effectiveness in transmitting a ransom note.

A malware loader and generalised “modules” were created, which allowed the communication hijacking process to be automated. These techniques were then tested on six devices to study their efficacy in a realistic scenario.

Despite the chosen devices being manufactured under different brands, and designed for a variety of purposes, they were all found to be vulnerable to communication hijacking. It was demonstrated that there were multiple effective methods to communicate with victims via exploited IoT devices, despite the lack of a standardised output.

It was also established that by generalising these hijacking methods into “modules”, it was possible to adapt these methods to be “generic” to maximise compatibility with many different devices, without the need to make significant modifications per device. This could be used by future attackers to circumvent some of the limiting factors of IoT-based ransomware, namely the variability of targeted IoT devices and the time investment required for bespoke implementations, drastically reducing the costs of performing a successful attack.

With all these factors, limitations, and techniques being considered, the next stage of research was focused on putting them into practice through the development of a proof-of-concept bricking-based IoT ransomware, as discussed in the next Chapter.

Chapter 5

Bricking Ransomware

Based on content of previous publication:

“PaperW8: An IoT Bricking Ransomware Proof of Concept” [41]

5.1 Introduction

Users and businesses alike have become increasingly reliant on Internet-connected devices, and the data contained within them. One of the main reasons that ransomware is so successful is its ability to control the availability of such resources. Generally, this type of asset control fits into one of three categories¹:

- **File Encryption.**

By encrypting files that have value to the victim, while retaining control of the decryption keys, a victim can be forced to make a payment to regain access. Files which have been created by the victim are often unique and cannot be easily recreated unless a backup has been made.

- **Locking.**

“Locking” is the act of preventing a victim from accessing the device or its functionality [147]. This is somewhat related to “scareware”, in that scare tactics may be used to further encourage payment [167]. FBI Lock, as an example, is an Android-based ransomware family that masquerades itself as

¹It should be noted that some of these types can be used in conjunction with one another. Notpetya [117], as an example, not only encrypted vital files but also modified the bootloader to prevent the device from being used.

the Federal Bureau of Investigation (FBI) and locks the device under the pretence that the victim had committed a crime, such as piracy [80]. When a payment is received by the attacker, control of the device is then returned to the owner.

- **Private Information.**

Ransomware authors may attempt to extract private information found on the devices that they infect, which they can use as leverage to encourage payment. This has been used in targeted attacks on various companies [96; 4] by threatening to release stolen information if a payment was not made. A proof of concept of this type of ransomware being implemented for IoT is covered in detail in Chapter 7.

While these ransomware categories can be implemented for desktops and laptops, they could also be applied to IoT devices. However, the variation in design may impose limitations for attackers. Therefore, more creative methods may be required.

IoT devices rarely store any valuable private or user-generated information on the device, which lowers the likelihood of successfully ransoming the user by encrypting personal files. Locking, however, is much more applicable, as the *functionality* of *any* device will have value to its user. Assuming an appropriately valued ransom demand, disabling access to this capability should therefore work universally, regardless of the type of device being attacked. The process of disabling a device’s capability in order to render it useless is often called “bricking a device” [212; 244], and it is one of the most destructive threats facing IoT devices.

In this Chapter, the viability of bricking-based ransomware attacks on IoT devices is investigated. The limitation of IoT devices are identified, and possible methods that could be used to circumvent them are explored. Finally, a proof of concept IoT bricking-based ransomware is demonstrated, and the extent of the damage such attacks could cause is evaluated.

5.2 Locking Ransomware

Both Desktop and Android systems have been targeted by locking malware such as WannaCry [116; 184] and FBILOCK [80]. As part of this work, previous malware instances of locker-based ransomware were examined to identify common design choices and potential pitfalls in their implementation. An outline of the observations is shown below.

5.2.1 Design

Locking ransomware typically aims to remove aspects of control from the user, restricting the functionality of the device and the access of the victim. For desktops, locking malware can remove the victim’s ability to run applications or navigate away from the ransom note window [152]. Victims who wish to regain access to their desktops are instructed to make a payment to recover the device.

Malware that impacts mobile operating systems, such as Android, takes a similar approach, locking victims into malicious apps, or using administrative powers to change the accessibility settings and PIN of the device [10].

Some ransomware implements both locking and file encryption in their attacks. Petya, a ransomware family that attacked Windows systems, modifies the master boot record to prevent the machine from booting up normally, then encrypts the master file table, rendering files inaccessible [125].

5.2.2 Limitations of Locking Ransomware

While locking devices and functionality may seem like an effective method to ransom victims, it does have some limitations, as outlined below.

5.2.2.1 Payment

For a ransomware campaign to be successful, the victim must be able to pay the ransom, which could be difficult if they are locked out of the device that they would be most likely to use to make a payment. It would be in the attacker’s best interest to allow their victim to be able to carry out activities on the device which can facilitate payment.

5.2.2.2 Manual Recovery

While locking a device, the attacker aims to prevent the victim from being able to use the device unless a payment is made. However, it may be possible for the victim to recover the device if they can replace or reinstall the necessary components for the device to function.

Unlike user-generated files, files that manage the operation of the device are unlikely to be unique and may be provided by the developer. Therefore, even if the files governing the functionality of the device are encrypted or otherwise modified to prevent it from being used, they could potentially be restored using external copies quite readily. It should be noted that while it may be simple to access the storage of some devices directly, others may require specialised tools, such as flash chip readers and writers.

5.2.2.3 Factory Reset

If no important files are stored on the device, victims of locker-based ransomware may be able to factory reset or reinstall the operating system of the infected device to regain functionality without much loss.

A factory reset typically allows a user to wipe any changes made to the device and restore the essential components, such that it returns to an original “safe” state. For example, Android locker-ware such as “FBILOCK” or “Koler” can be removed by performing a factory reset [87; 80; 202].

5.2.2.4 Bypassing Locking Mechanisms

In some cases, it may possible to fully recover the device without needing to pay the ransom, or performing a factory reset. As no files are encrypted as part of the infection process, if the victim is able to bypass the locking mechanism, the device and any stored files can be considered “recovered”, with no lasting effects.

As mentioned in the previous section, some Android locker-ware can be removed using a factory reset, but this does have the unfortunate side effect of also removing any user-created files on the infected device. However, it is possible to use an alternative Android provided feature to attempt recovery instead: “safe-mode”.

Safe-mode allows the device to boot while disabling most third-party applications. This is traditionally intended to be used as a debugging tool to hunt down misbehaving applications, but it can also be used to bypass locking malware that attempts to run on startup. The victim can then use this access to remove the malware from the device [87; 80; 202].

5.3 Related IoT Malware

During this work, attempts were made to predict the future directions that IoT malware might take. As part of this process, several strains of previously-discovered IoT malware were examined to identify common techniques that could be used to facilitate IoT-based ransomware. The IoT malware families that were found to be relevant are outlined below.

5.3.1 Mirai

At the time of writing, Mirai – which was written by Paras Jha, Josiah White, and Dalton Norman [153; 154] – is arguably the most well-known family of IoT malware. As previously mentioned in Section 2.2.3.1, Mirai attacked Linux based IoT devices and was incredibly successful, infecting approximately 65,000 devices, including cameras, routers and VoIP phones, during its first 20 hours of operation [12].

Initially, Mirai spread by targeting open telnet ports with weak authentication, brute-forcing logins with credentials selected at random from a hard-coded list. Each infected device was then used to further spread the malware, scanning the Internet at random for other vulnerable telnet services. When a new device was successfully detected and exploited, a notification was sent to the “report” server, marking the device for infection.

Mirai was used to perform damaging Distributed Denial of Service (DDoS) attacks on multiple targets, including the DNS provider “Dyn”, rendering many popular websites such as Github and Twitter inaccessible [273], and a 1Tbps DDoS attack against the French web service provider “OVH” [102].

5.3.1.1 Variations

Mirai’s source code [138] was publicly released on HackForums in September 2016 [11]. Several malware variants were quickly developed, with some boasting additional exploits, extra functionality or larger lists of targeted devices.

One such notable variation, dubbed “Echobot”, was found to be using over 50 different exploits in addition to brute-forcing telnet logins [259]. Some variants, such as “OMG”, even added new methods of monetisation, such as re-purposing infected IoT devices to act as paid proxy servers [173].

The open-source nature of Mirai was particularly useful for this work, as it allowed direct analysis to be performed, providing an overview as to how Mirai was developed and the features contributing to its success. During this work, aspects of Mirai inspired how the design and implementation of various proofs of concept and tests were developed.

5.3.2 Brickerbot

Brickerbot is an IoT-based malware that is capable of “bricking” vulnerable IoT devices [103]. In this case, “bricking” is a term that is used to describe “damaging a device in such a way that it no longer functions” [212; 244].

Brickerbot scanned for devices with known vulnerabilities, exploited them, and then ran a list of shell commands that would render them inoperable. This was achieved by writing random data to flash memory, throttling services to a near unusable level, and adding firewall rules that dropped all incoming traffic [246]. Examples of some of the commands run by Brickerbot are shown in Listing 5.1.

```
1 # Overwrite important flash partitions with random data
2 cat /dev/urandom >/dev/mtdblock0 &
3
4 # Delete all iptables rules, then add a new rule to drop
   all incoming traffic.
5 iptables -F;iptables -A INPUT -j DROP
6
7 # Turn off TCP timestamps
8 # May reduce Internet connection reliability
9 sysctl -w net.ipv4.tcp_timestamps=0
```

```
10  
11 # Set the max number of kernel threads used by the kernel  
    to one.  
12 sysctl -w kernel.threads-max=1
```

Listing 5.1: Shell commands run by Brickerbot [130; 210]

Some of Brickerbot’s obfuscated source code responsible for the device bricking was later published online [130], alongside the author’s “manifesto” [58]. Within the manifesto, the author detailed the purpose of the original “project”, dubbed “Internet Chemotherapy”. Brickerbot’s author (who identified themselves as “The Doctor” or “Janit0r”), claimed the purpose of the malware was to prevent vulnerable devices from being exploited for criminal purposes or added to botnets such as Mirai and to raise awareness of the general insecurity of IoT devices.

5.3.3 Silex

Silex is another instance of IoT bricking malware that, similarly to Brickerbot, uses shell commands to brick compromised IoT devices [50]. While Silex was believed to be an independent family of malware, some of the commands that were run on the infected devices had been directly copied from the obfuscated Brickerbot source [130].

According to a researcher at NewSky Security, Silex was originally written “as a joke” by a 14-year-old going by the pseudonym “Light Leafon” [59]. Although not as impactful as Brickerbot, the development of Silex does highlight the danger of possible “copycat” attacks being performed with relative ease.

5.4 Limitations of IoT Ransomware

Previous IoT malware has proven to be very successful and has been used to perform effective attacks on multiple high-profile targets. However, IoT devices have a number of limitations that attackers would need to overcome for IoT-based ransomware to become a viable threat. Below, some of these challenges are discussed.

5.4.1 Asset Value

The success of current ransomware stems from being able to ransom resources that have value to the affected victim, such as private information or irreplaceable documents. While these types of files are commonly found on personal computers, they are not so common on IoT devices, which typically only store binaries responsible for running the system, or files containing users' configuration settings. This data is unlikely to have much personal value to the user and, if damaged, is normally quite easy to replace by simply restarting or factory resetting the device. Additionally, unique data that is not recoverable as part of this process, such as user profiles or configuration settings, will likely be easy for the user to recreate.

While there may be devices that are exceptions to the rule by storing valuable information – such as those that collect photos, health data, or the user's location – the lack of asset value available in *most* IoT devices would drastically limit the malware's scope and possible ransom price. Methods that could be used to ransom such devices more effectively are covered in more detail in Chapter 7.

Alternatively, the functionality of *any* device will have value to its user. Assuming that an appropriate ransom is demanded, disabling access to a device could be used to force the device's owner to pay a ransom, regardless of the type of device being attacked.

5.4.2 Persistence

The majority of IoT infections are not persistent, which is partially due to the nature of IoT storage. For many IoT devices, the partitions used to manage the device, such as the filesystem, kernel, user configuration and bootloader, are stored in flash memory. Therefore, for an instance of malware to be persistent, the data stored in flash memory must be altered to include it.

However, the format, structure and purpose of stored data can vary on a device-to-device basis, as they often implement different bootloaders, Linux kernels and filesystem formats depending on their needs. This diversity complicates the development of a universal and reliable method to modify flash memory without corrupting the existing firmware.

Due to this complexity, IoT malware rarely exhibits any form of persistence, which means that infections can be removed by simply restarting the device [60].

While there have been exceptions to this rule, such as with the IoT malware “torii” [156], or “Hide N’ Seek” [35], the methods used by these families to obtain persistence were not particularly sophisticated and relied on being able to make direct changes to writeable filesystems via shell commands, which are unlikely to work on IoT devices with read-only filesystems.

All of the devices that were used as part of this research into bricking ransomware implemented read-only root filesystems, which made them difficult to modify without completely re-flashing the partition that contained them. While it is technically possible to gain persistence on these types of devices, it can be complex, and the technique required is highly dependent on the device’s design².

Bashlite and Mirai are examples of IoT malware that do not exhibit any forms of persistence. Persistence was not required in these instances, as neither malware family interfered with the victim’s use of the infected device (other than the usual network overheads of receiving commands or performing attacks). Infections were therefore difficult for an average user to detect, which lowered the likelihood of the infected device being rebooted, hence reducing – if not removing altogether – the need for persistence in these cases.

Unlike typical malware, however, ransomware is generally required to inform the victim that they have been infected in order to display the ransom demand [14]. If such a ransom note is encountered on an IoT device, it will most likely result in the victim simply forcing the device to restart in an attempt to regain access.

Surviving or preventing this restart is key for an IoT-based ransomware attack to have any chance of success. Hence, for IoT-based ransomware to work effectively, it will either need to be capable of obtaining persistence or become persistence-independent.

5.4.3 Device Variation

IoT devices can vary heavily. For instance, they may run on differing hardware, architectures and operating systems appropriate for their needs. The availability of installed tools and applications will also change depending on the device, as each IoT developer will likely install only what is strictly necessary for the device

²For the work presented in this Chapter, implementing persistent malware is not required. However, persistence on IoT devices and its potential benefits are covered in more detail in Chapter 6.

to function in order to save valuable storage space.

When creating IoT-based ransomware, attackers will need to account for this variation to ransom multiple devices effectively. If the created ransomware is only compatible with a small number of devices, it could seriously limit the potential scope, and therefore the potential profit.

5.5 Known IoT Ransomware Implementations

Despite these limitations, there have been several previous attempts to implement ransomware on various IoT devices, some of which are detailed below.

5.5.1 Proofs of Concept

Researchers have created basic “proof of concept” ransomware for specific IoT device models to demonstrate methods that attackers might use to ransom a victim.



Figure 5.1: “Smart Thermostat” ransom note³

5.5.1.1 Thermostat

Researchers have previously demonstrated infecting an IoT thermostat by modifying existing firmware to include ransomware. After exploiting the device over the local network using a command injection attack, the new firmware could be

³Image Source: <https://www.mcafee.com/blogs/internet-security/black-hat-danger-drones-thermostat-ransomware/>

installed. When run, the malware displayed a simple ransom note via the attached screen (as shown in Figure 5.1), and was able to change the victim’s PIN or modify the temperature in the victim’s home via a local HTTP based API [252]



Figure 5.2: “Smarter Coffee Maker” infected with ransomware⁴

5.5.1.2 Coffee Machine

Researchers at Avast demonstrated that an attacker could compromise a smart coffee machine, allowing them to display ransomware notes (shown in Figure 5.2) and force the device to beep, boil water, or run the grinder without any user interaction [121].

One limitation of this attack, however, is that the attacker has to be on the same Wi-Fi network as the coffee maker. Although a ransomware attack is still possible, this drastically limits the number of potential targets an attacker could infect.

⁴Image Source: <https://arstechnica.com/information-technology/2020/09/how-a-hacker-turned-a-250-coffee-maker-into-ransom-machine/>

5.5.2 In the Wild

IoT-based ransomware has also been seen “in the wild”. While this does highlight that IoT ransomware is being considered by attackers, it is still a less common occurrence than the traditional botnet-based IoT malware.

5.5.2.1 LG Smart TV

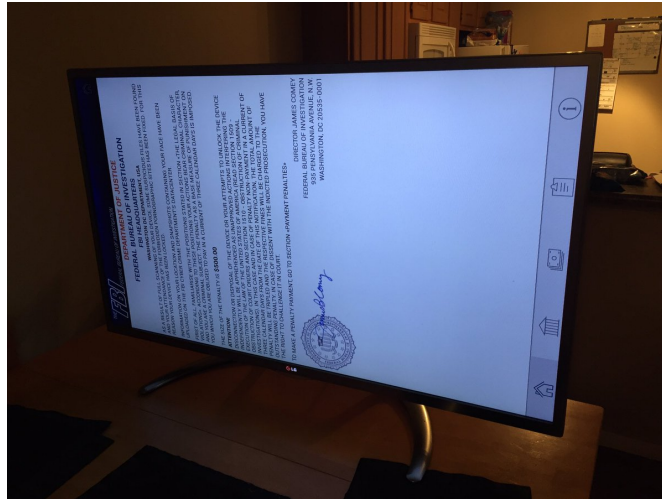


Figure 5.3: LG SmartTV locked displaying a ransom note⁵

In 2016, an LG Smart TV was disabled by the Android-based malware “CYBER.POLICE” [56]. Smart TVs were unlikely to be the intended target, as the malware family typically affected Android phones. The ransom note was also displayed in an orientation that was more apt to a phone screen, as shown in Figure 5.3.

Despite Smart TVs not being the intended target, the ransomware did seem to lock the device successfully, requiring LG support to provide an alternative method to perform a factory reset, with which the victim was able to successfully regain control of the device [61]. While local storage is wiped during a factory reset, it is unlikely to cause much distress to victims as, unlike phones which may contain unique data of value to the victim, Smart TVs are unlikely to store such information.

⁵Image Source: <https://www.techspot.com/news/67573-smart-tvs-arent-immune-ransomware-one-user-recently.html>

5.5.2.2 ChastityLock

In October 2020, researchers at PenTestPartners published a blog post highlighting various flaws in the “Qiui Cellmate” chastity device, such as potentially exposing private information, and unauthorised use of the API [168]. However, despite these issues being highlighted, these flaws were later exploited by a ransomware family known as “ChastityLock” [129].

ChastityLock was one of the earliest instances of ransomware that affected IoT devices with limited processing capabilities. By exploiting vulnerabilities in the device’s API implementation, ChastityLock was able to remotely lock Internet-connected chastity devices and demand 0.02 BTC (Approximately \$270 USD) for recovery [129].

The source code of ChastityLock was later published to GitHub by the research group “VXUnderground” [270], allowing the structure to be analysed. The malware was written in Python, separated into two files. One script handled user enumeration, and the other was used to disable the device and deliver the ransom demand. An overview of the process can be seen in Figure 5.4.

The “Enum.py” script was designed to be run first. It would query the device developer’s API, looping through potential user IDs to identify valid devices, then extract the usernames and “member codes” (which can then be used to make API requests on behalf of the user) to a file.

Once all the user data had been collected, the “Ransom.py” script would be run. The ransom script would loop through the user details obtained by the enumeration script, then after impersonating another random user, it used the API to add the target as a “friend”, transferred control of the device to the impersonated account, then sent a ransom note to the victim via the accompanying app to demand payment. It is unclear whether any payments to the attacker were made, or if there was any process in place to return control of the device to the affected victim.

Although this is not strictly “malware”, as it exploited flaws within the API rather than attacking the device itself, this attack does highlight the creative approaches attackers could take to extract payment from victims with vulnerable IoT devices.

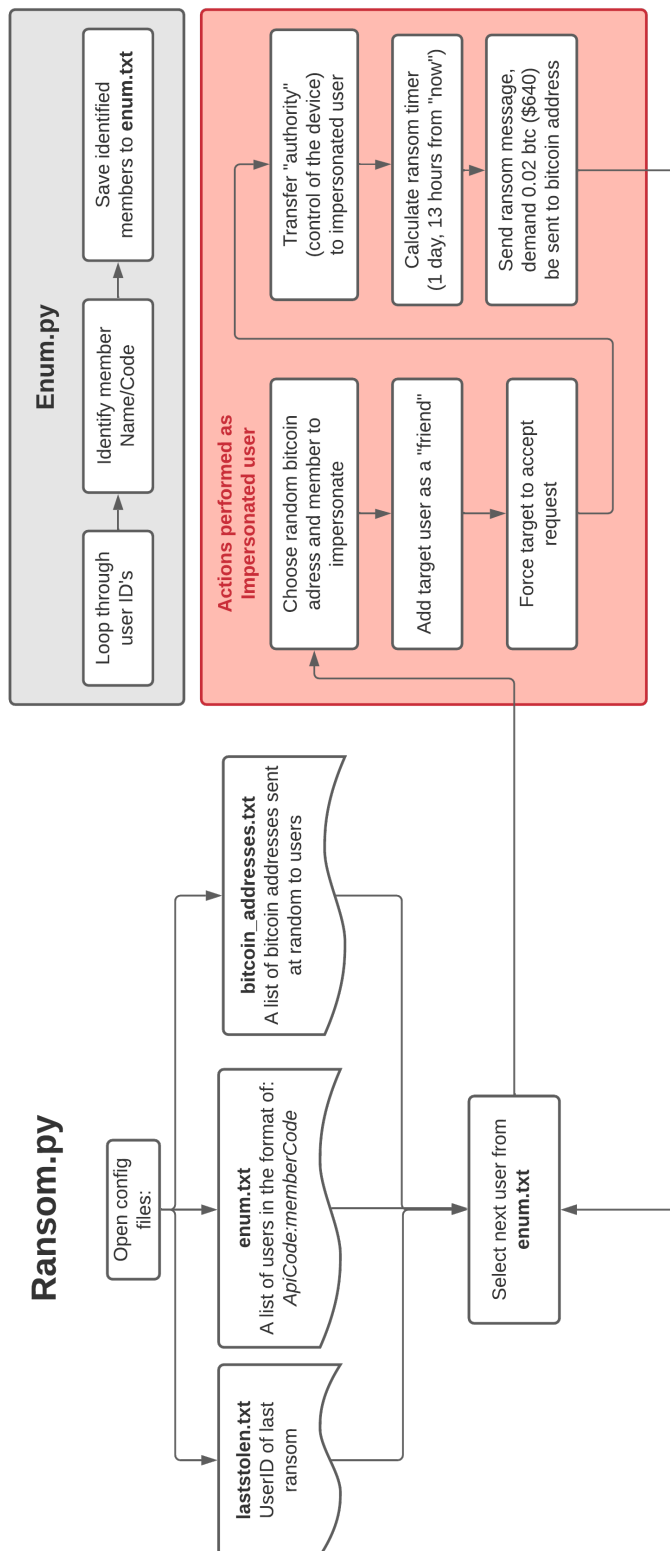


Figure 5.4: ChastityLock Ransomware Overview Graph

5.5.3 Limitations

These examples highlight the possibility for attackers to use creative approaches when implementing ransomware on IoT devices, especially if the accessed devices are able to influence real-world resources. However, while they could potentially be profitable, these approaches have limitations that may prevent attackers from being able to implement them in a practical setting, or a large scale attack. These limitations are outlined below.

5.5.3.1 Scope

The thermostat, coffee machine and chastity ransomware all used creative methods for locking down and ransoming their targeted device. However, this creativity comes with the burden of only working for the targeted model or brand, which greatly limits the potential number or scope of affected devices, and therefore the total likely payout. To work on other devices, the ransomware would likely need to be heavily modified for each new device model.

The “CYBER.POLICE” malware that affected the LG smart TV was an exception to this rule. Despite originally targeting phones, it was able to infect a smart TV due to sharing a common operating system: Android. Rather than encrypting files on the device, it locks the victim in an application that they cannot leave. Unfortunately, this ended up being to the ransomware’s detriment, as without valuable user files being stored on the TV, the victim was able to recover the device by factory resetting.

Therefore, the “scope” of the ransomware must be considered during its design, otherwise, it may either prove unprofitable or ineffective in a large scale campaign.

5.5.3.2 Persistence

While the methods used to lock the IoT thermostat, coffee machine, and LG Smart TV did seem to be effective, they would require the malware to persist through reboots.

Based on an image included in the report [252] (shown in Figure 5.5), the thermostat used a writeable root filesystem which allowed persistent modifications to be made from the shell [252], while the coffeemaker’s firmware had to be reverse-engineered, modified, then replaced [268].

```
andrewtierney@ubuntu:~/vs$ binwalk 4.bin
```

DECIMAL	HEXADECIMAL	DESCRIPTION
22	0x16	uImage header, header size: 64 bytes, header
16 bytes		16 bytes, Data Address: 0x20008000, Entry Point: 0x20008000, data CRC: 0x
		ssion type: none, image name: "Linux-3.15.0"
86	0x56	Linux kernel ARM boot executable zImage (l
17783	0x4577	gzip compressed data, maximum compression,
2001502	0x1E8A5E	JFFS2 filesystem, little endian

Figure 5.5: Binwalk analysis of smart thermometer’s firmware

Additionally, it should be noted that the authors of the thermostat and coffee-based ransomware proofs of concept did not state whether their malware would survive a factory reset. If not, it would allow victims to easily recover their infected devices.

The “ChastityLock” ransomware was a notable exception, as unlike the other examples, the malware was never run or stored directly on the targeted device. Instead, interactions were made externally via a vulnerable API, requiring no persistent modifications of the device [129]. It should be noted that this was an opportunistic attack that only works on this device model, and is unlikely to be recreated.

Gaining persistence can be a complicated process. Moreover, a method that may work on one IoT device is not guaranteed to work on another. Further discussion on persistence is provided in Chapter 6.

5.5.4 Summary

For effective IoT ransomware to be achieved at scale, the limitations described above in Section 5.5.3 must be solved or circumvented. In the next section, a proof of concept IoT-based ransomware, which demonstrates how these challenges can be addressed, is presented. This represents one of the main contributions of this PhD research.

5.6 PaperW8

As part of this research, a proof of concept ransomware named “PaperW8”⁶ was created, to explore how ransomware could realistically attack exploitable Linux-based IoT devices at scale.

This proof of concept can assist in the process of identifying roadblocks attackers may encounter when developing ransomware, and the creation/testing of countermeasures for developers, end-users and security researchers. This section outlines the structure of PaperW8, how it operates and how it overcame the previously described limitations.

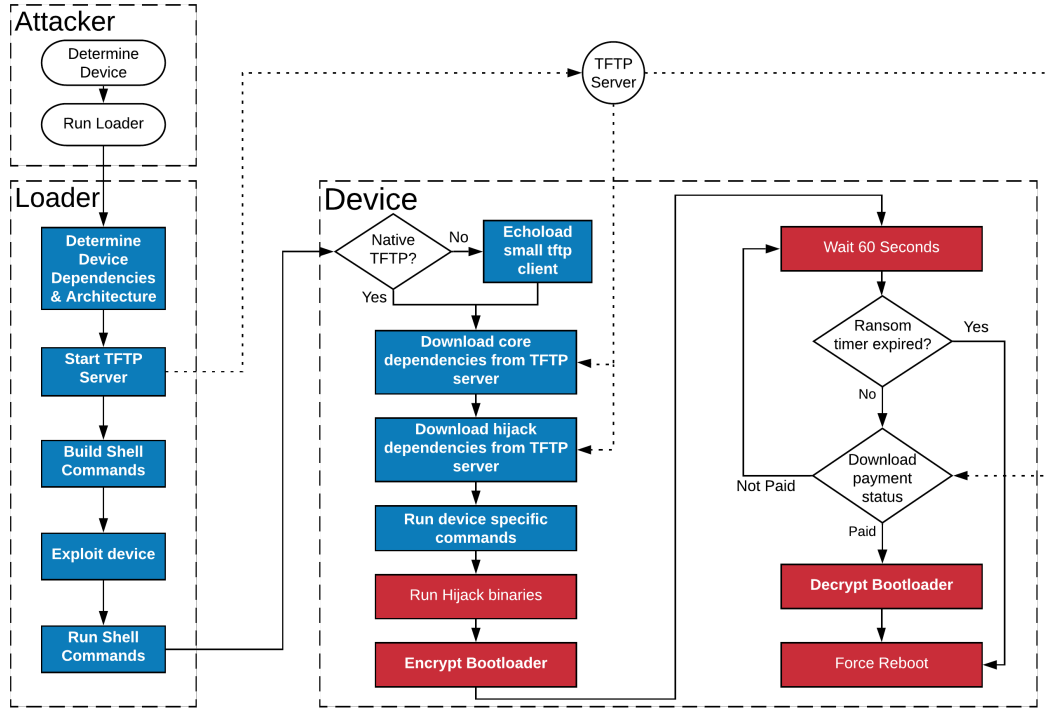


Figure 5.6: PaperW8 Structure Graph

⁶In the simplest form, PaperW8 removes all functionality from the affected device, turning it into a “paperweight”, hence the name.

5.6.1 Structure

PaperW8 was designed with adaptability and generalisation in mind. While previous attempts at IoT-based ransomware target specific models and brands, PaperW8 aimed to work on many different devices, such that as few modifications as possible would be necessary to extend the malware to new devices.

This was achieved by limiting the number of requirements to ransom a device, and building upon the “generic” communication modules developed in Chapter 4, such that PaperW8 could hijack common communication channels and deliver an effective ransomware message.

There are multiple steps required to infect a device with PaperW8, including loading dependencies, hijacking communication channels, and disabling the device, which was partially managed by the malware loader previously described in Section 4.4.2. An overview of how a PaperW8 attack is structured is shown in Figure 5.6. The following sections cover the stages of a successful attack in more detail.

5.6.2 Exploitation and Infection

To infect a device, PaperW8 first attempts to run shell commands on the target device. This can be achieved in many different ways, such as accessing an exposed telnet port with weak authentication, or utilising an existing exploit, such as command injection via an unsanitised input.

Once PaperW8 has gained access, the dependencies relevant to the targeted device are uploaded, such as hijacking and bricking modules. This is achieved via the methods described in Section 4.4.2, using TFTP or echoloading where appropriate.

After all the required dependencies have been uploaded, any services that communicate with the user are killed to prevent the device from being used. Killing the vulnerable services that were used to exploit the device where possible can also prevent the victim or other attackers from using the same vulnerability to regain access. Finally, PaperW8 executes the uploaded binaries, taking full control of the device.

5.6.3 Permanent Denial of Service

To permanently disable infected devices, persistent modifications to the device needed to be made. As such, the storage of infected devices were chosen as a potential target for attacks. Many Linux-based IoT devices implement “flash memory” to store essential components required to run the device effectively, such as bootloaders, filesystems, or user configurations.

To interact with the various forms of flash storage, Linux typically uses the Memory Technology Device (MTD) subsystem, an “abstraction layer for raw flash devices” [21]. MTD can be used to read data, manage partitions, or make modifications to flash storage. This allows Linux users to interact with attached flash devices through device files stored in the `/dev` and `/proc` directory, some of which were used as part of this work. Some of the notable files are described below.

- `/proc/mtd`. The `/proc/mtd` file contains the definition of each MTD partition, including the names set by the developer, which may indicate each partitions purpose.
- `/dev/mtdX`. Individual partitions within the flash device can be accessed via the `/dev/mtdX` character files, where “X” is the index of the partition.
- `/dev/mtdblockX`. The `mtdblockX` files are similar to `mtdX` files, but allow the chosen partition to be accessed as if it were a block device⁷. In this work, modifications to the flash chip are primarily performed through these files.

5.6.3.1 Partition Encryption

To ransom infected devices, PaperW8 uses the MTD subsystem to make modifications to the essential components stored in certain partitions, such that the victim cannot reasonably recover without making a payment to the attacker.

The MTD subsystem is a convenient method of making these malicious changes for a number of reasons:

⁷Block devices organise their data into “fixed-size blocks”. Unlike character devices, block devices must be accessible at random offsets, and communication is performed by sending blocks of data (often via a buffer) [236; 140]

- The MTD subsystem abstraction provides a standardised interface between devices of different brands and models.

Device variation can present significant difficulty when developing malware, as it can require considerably more effort to remain compatible with a large number of devices. Any standardisation that may simplify this process should be utilised.

- The MTD subsystem was present in every device encountered during the testing stage.
- The MTD subsystem has been tried and tested by previous malware such as Brickerbot and Silex (see Section 5.3), which both used shell commands to write random data to flash memory via the MTD subsystem, irrecoverably damaging the infected device.

An attacker can prevent flash partitions from being effectively used by encrypting their contents. While the partitions containing the kernel or filesystem could be encrypted by ransomware, doing so might interfere with the running of the device, inadvertently putting it into an irrecoverable state. Instead, PaperW8 targets the bootloader, which is unlikely to be used after the booting process has been completed.

To ransom infected devices, PaperW8 uploads an application that reads the bootloader into a buffer, encrypts it with AES-256-CBC⁸, then writes it back to the same partition. This mangles the bootloader such that the device will fail to boot if it is restarted⁹.

This approach is somewhat similar to those taken by Windows-based ransomware that modifies the Master Boot Record (MBR) of infected machines, such as *Seftad* [91; 149] and *Petya* [125].

While PaperW8 may not implement persistence, the extortion method used can be considered “persistence-independent”, such that any attempts to recover from an infection by restarting the device will result in the device being bricked.

⁸When performing early tests, a simple XOR cipher was used instead.

⁹During the encryption stage, the Initialisation Vector (IV) and the key are hard-coded for testing purposes, but it is reasonable to expect that these could be generated, stored and managed by a C&C server in a real setting.

At this point, PaperW8 will have taken full control of the device, while restricting the victim's access. As PaperW8 retains the ability to modify the flash memory until the device is reset, if a payment is made by the victim, the bootloader partition can be decrypted by PaperW8 with the appropriate key. Upon rebooting, the device will continue to function as usual, but will still need to be patched to prevent future infections.

The victim, if infected, is unlikely to have an easy method of recovery, thus leaving paying the ransom as the only remaining option.

5.6.4 Communication Hijacking

As part of the ransomware process, the victim must be notified that their device has been infected, what has occurred, what the consequences are, and how a payment must be made. In order to do so, PaperW8 uses the various communication hijacking techniques covered in Chapter 4.

Each communication hijacking module is packaged as a binary compatible with the exploited device and uploaded as a dependency as part of the infection stage. After the exploitation and bricking stages have been completed, the binaries are executed to display a ransom message with further details, using the most appropriate methods for the device.

5.6.5 Recovery

As mentioned previously, PaperW8 shuts down vulnerable services when possible, preventing victims from being able to use the same method to regain control of the device and potentially recover.

As rebooting the device is no longer an option, factory resetting is the obvious next choice. However, as updating the bootloader of devices is generally not advised [227; 277; 265; 226], the partition containing the bootloader is unlikely to be modified or corrupted during normal operation of the device. Therefore, the bootloader partition is often not included in the factory reset process. Consequently, if the victim attempts to perform a factory reset, the bootloader may be left unchanged, and the device would remain bricked.

Therefore, there are two main methods for recovering an infected device after the encryption stage, which are described below.

5.6.5.1 Ransom Payment

After the bootloader partition has been encrypted and the ransom message has been displayed, the victim will have a limited time to make a payment to the attacker. Typically, a C&C server is used by ransomware developers to manage a large number of infected victims. For this proof of concept, a simplified pseudo-C&C server was used to simulate payment and receipt confirmation for testing purposes.

Using a bash script, PaperW8 periodically checks for a predetermined value on the attacking host via TFTP to see if the victim has “paid”. If the expected value is found, PaperW8 will decrypt the bootloader and reboot the device. If, however, the victim does not make a payment before the time expires, PaperW8 can force the device to reboot (without decrypting the bootloader) and — as a consequence — brick it.

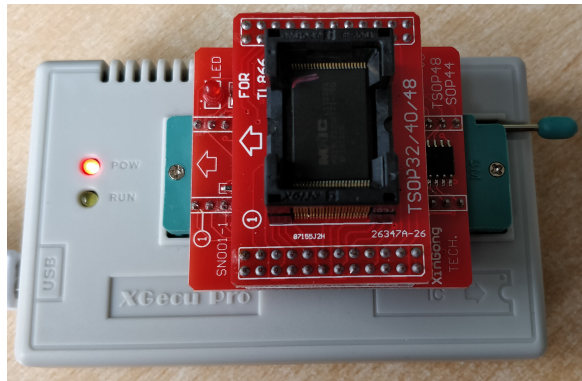
In a real-world setting, a more secure method of key management and confirmation would be used, but the proof of concept presented here sufficed for emulating such a payment process.

5.6.5.2 Manual Recovery

If the victim has access to the required tools, manual recovery can be attempted by reinstalling the original bootloader using an external programmer. Some of the possible techniques that could be used are detailed below.



(a) Flash Chip on Router PCB



(b) Flash Chip Placed in TL866 Programmer

Figure 5.7: Manually Programming a MX29LV320ETTI-70G Flash Chip

- Victims could attempt to identify and interface with an on-board debug interface that can be used to reprogram flash memory (such as JTAG [141]). However, this would require the device to support such interfaces.
- The flash chip could be desoldered from the device for reprogramming, as shown in Figure 5.7 (a) and (b), then re-soldered to the board.
- A test clip, as shown in Figure 5.8, could be used to attach an external programmer to the flash chip on the board. This does, however, run the risk of accidentally powering other components on the board if they share power traces¹⁰, which may result in interference when attempting to read from, or write to, the chip.

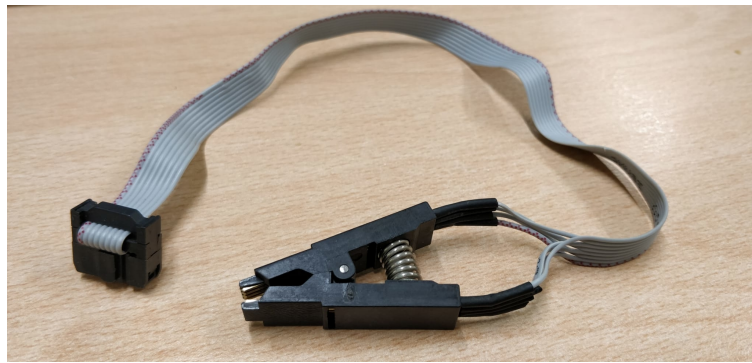


Figure 5.8: 8-Pin SOIC Test Clip

It should be noted that these processes would also require the victim to either obtain or somehow decrypt the original bootloader.

The diversity of IoT devices works against victims in this case, as finding a solution may require them to identify debug interfaces, storage locations or the correct bootloader for each attacked device and model. The level of expertise required for the above techniques is unrealistic to expect from an average user, and in the absence of specialised knowledge or equipment, most victims will likely believe that paying a ransom to the attacker is the only viable recovery option.

¹⁰For efficiency, multiple electrical components in a circuit often share the same power source connected by a shared “trace”. By providing power to this trace, other components may be powered and attempt to communicate with the chip at the same time as the user, which may result in undefined behaviour.

5.6.6 Categorising PaperW8

PaperW8 has a design that is similar to traditional ransomware, but there are a number of key differences that prevent it from being easily categorised using existing terminology (such as those described in Section 2.3.1). For example, while PaperW8 does encrypt files on the devices that it infects, similar to “crypto-ransomware”, it does not aim to encrypt “user” files. Instead, it only targets the bootloader, with the intention of preventing the device from functioning after it is rebooted, which could arguably be used to categorise PaperW8 as “locker-ransomware”. However, traditional “locker” ransomware does not typically use cryptography as a method of “locking” functionality.

While it may be tempting to place PaperW8 into a “hybrid” category (e.g. “crypto-locking” ransomware), this could lead to confusion with crypto-ransomware, or a previously existing ransomware family named **CryptoLocker** [165]. Instead, it may be best to focus on the effect of PaperW8 should the ransom not be paid, such as the device being bricked.

Traditional “brickware”, which has no ransom functionality, does not require the use of cryptography to brick a target device. Instead, previous instances (such as those mentioned in Section 5.3.2 and Section 5.3.3) simply write random data to essential bootloader partitions. The use of cryptography implies the possibility of recovery through decryption, therefore, in order to differentiate from “brickware”, this type of ransomware could be referenced with a new term of “crypto-bricking”.

5.7 Tested Devices and Results

To evaluate the viability of PaperW8, it was tested on a collection of devices with differing purposes and brands, which had been used in previous work shown in Chapter 4.

In this section, an overview of each device’s storage structure will be given, followed by a review of the methods used by PaperW8 to perform the ransomware attack.

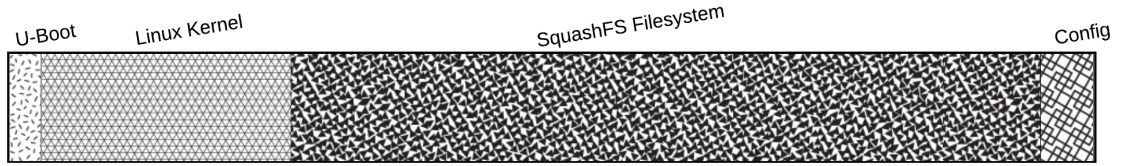


Figure 5.9: MTD partitions in HG532's Flash Memory

5.7.1 HG532 TalkTalk Router

By reading the contents of the router's `/proc/mtd` file (shown below in Listing 5.2), the names and sizes of several partitions set in the device's flash memory could be identified. A visual representation of the layout is shown in Figure 5.9.

1	Part	Size	Erasesize	Name
2	mtd0:	00020000	00010000	"Bootloader"
3	mtd1:	000ed47f	00010000	"Main Kernel"
4	mtd2:	002c2b81	00010000	"Main RootFS"
5	mtd3:	00030000	00010000	"Protect"

Listing 5.2: HG532 `/proc/mtd` file

Fortunately, the partition that contained the bootloader was relatively easy to identify. The “Bootloader” partition, which could be accessed via the `mtdblock0` file, was therefore selected for encryption.

After performing the encryption step, the device was reset, after which it failed to boot. A flashing LED indicated that the device was receiving power, but it was otherwise unresponsive. The factory reset process was then activated, but the functionality of the router was not able to be recovered.

To confirm that the encryption stage had been performed as intended, data was read directly from the device's flash storage chip. The chip¹¹ was desoldered and read with an external flash programmer, as shown in Figure 5.7 (b). The bootloader was found to have been encrypted correctly using the expected scheme, and the original data could be obtained after decrypting the encrypted data with the appropriate key, proving that a device could be “recovered” by PaperW8.

¹¹Manufacturer part ID: MX29LV320ETTI-70G [170]

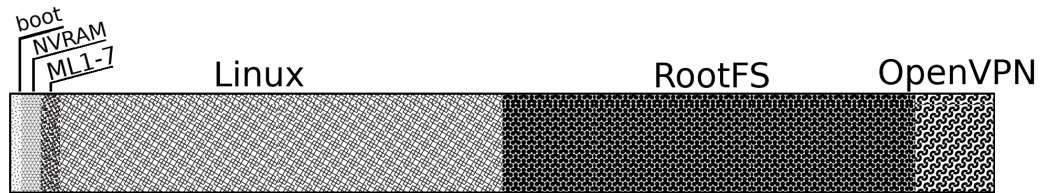


Figure 5.10: MTD partitions in R6250's Flash Memory¹²

5.7.2 R6250 Netgear router

The R6250 router defined 18 flash partitions in its `/proc/mtd` file, shown in Listing 5.3¹³ and Figure 5.10.

```

1 dev:      size      erasesize name
2 mtd0:    00080000  00020000  "boot"
3 mtd1:    00180000  00020000  "nvram"
4 mtd2:    00020000  00020000  "board_data"
5 [...Irrelevant Partitions have been omitted...]
6 mtd14:   01e00000  00020000  "linux"
7 mtd15:   01bc0fc0  00020000  "rootfs"
8 mtd16:   05980000  00020000  "brcmnand"
9 mtd17:   00500000  00020000  "OpenVPN"

```

Listing 5.3: R6250 `/proc/mtd` file

The first partition, `mtd0`, contained a U-Boot bootloader, which was chosen as the initial target. However, the developers had set the partition to be read-only when configuring the MTD subsystem.

Attempts to remove the read-only flag via the shell were unsuccessful, leading to the assumption that such a restriction could not be easily circumvented¹⁴.

Therefore, the “linux” partition in `mtd14`, which contained the Linux Kernel and the root filesystem, was selected as an alternative target¹⁵.

After the “linux” partition was encrypted, the device was rebooted and found to be unresponsive. Similarly to the HG532, some LEDs were lit to indicate that

¹²Due to their relatively small size, partitions `mtd2`-`mtd6` are not shown in Figure 5.10.

¹³Irrelevant partitions have been omitted from the Listing to save space.

¹⁴It was later discovered that there was a method that could be used to modify read-only flags set by the kernel. This method is covered in more detail in Section 6.5.4.

¹⁵It is believed that `mtd15`, labelled as “rootfs”, overlapped with the end of `mtd14`, and acted as a reference point to the beginning of the root filesystem.

the device was receiving power, but no other output was given.

An attempt was made to factory reset the device as described in the user manual, but functionality was not able to be restored [189]. However, an alternative method to recover the device was found on Netgear’s support website [191]. It is believed that as the bootloader had not been encrypted in this case, it was still able to function during boot. Upon failing to boot into Linux, the bootloader would open a tftp server, such that recovery firmware could be uploaded to the device. While this method was not mentioned in the official manual, it could be feasibly achieved by a knowledgeable user without any specialist tools.

5.7.3 TV-7104HE MVPower DVR

Unlike the previous devices, the DVR’s partition list was not particularly informative. Each partition was labelled using a single character, leaving little indication as to their use. Further analysis was performed by extracting the contents of each partition via the associated `mtdblock` devices and analysing each section with Binwalk [215].

Binwalk is a forensics tool used to read and analyse binary files to extract possible meaning. For example, it can be used to identify firmware headers, filesystem types, or system kernels. In this case, it was used to determine the purpose of each partition, as shown below in Listing 5.4. A visual representation of the flash memory’s layout is shown in Figure 5.11.



Figure 5.11: MTD partitions in TV-7104HE’s Flash Memory


```

1 dev:      size    erasesize  name
2 mtd0: 00020000 00010000 "U" # Compressed U-Boot
3 mtd1: 00010000 00010000 "E" # U-Boot configuration
4 mtd2: 00020000 00010000 "L" # Image for startup screen
5 mtd3: 00020000 00010000 "C" # DVR config
6 mtd4: 001b0000 00010000 "K" # Kernel
7 mtd5: 005e0000 00010000 "R" # Root filesystem

```

Listing 5.4: TV-7104HE /proc/mtd file

As the “U” partition was discovered to contain a U-Boot bootloader [76], it was chosen as the target for the encryption stage. After successfully encrypting the partition, the device was rebooted. While the in-built LEDs did show signs of activity, no output was displayed and the device made no attempts to connect to the attached router.

As the factory reset process had to be activated via the device’s GUI, which would no longer be available after the device was infected, it was not possible to activate the factory reset process. Additionally, the factory reset function was described as only resetting the user configuration of the device. As such, even if it were possible to start the factory reset process after the encryption stage, the reset process would not be able to restore the original state of the device.

5.7.4 WiPG-1000 Presenter

The presenter’s flash storage was split into 11 partitions, shown in Listing 5.5 and Figure 5.12. Due to the naming convention used in the partition listing, there were multiple potential targets selected for the encryption stage. After performing a Binwalk analysis of the targeted partitions, “u-boot-SF” was chosen as the final target, as it was found to contain the device’s U-Boot bootloader.

After encrypting the partition, the device was forced to reboot. Upon restarting, the button normally used to activate the device was unresponsive. While a setup screen would typically be displayed during the startup phase, no output to the attached screen was detected.

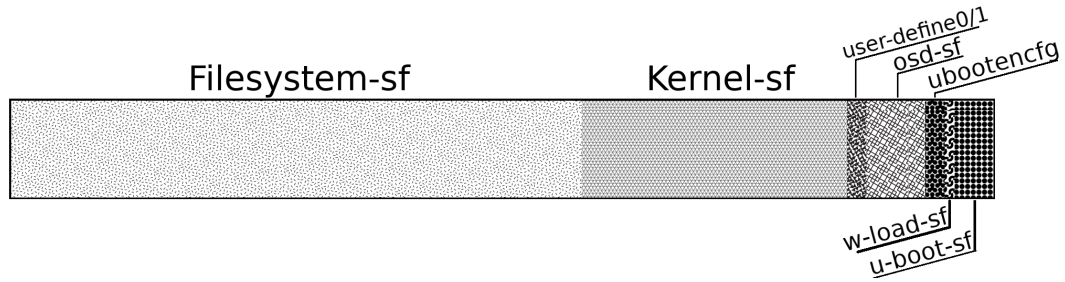


Figure 5.12: MTD partitions in WiPG-1000's Flash Memory

```

1 dev:      size    erasesize  name
2 mtd0: 00480000 00010000 "filesystem-SF"
3 mtd1: 00220000 00010000 "kernel-SF"
4 mtd2: 00010000 00010000 "user-define0"
5 mtd3: 00010000 00010000 "user-define1"
6 mtd4: 00010000 00010000 "user-data-SF"
7 mtd5: 00080000 00010000 "osd-SF"
8 mtd6: 00010000 00010000 "u-boot env. cfg. 1-SF"
9 mtd7: 00010000 00010000 "u-boot env. cfg. 2-SF"
10 mtd8: 00800000 00010000 "full image"
11 mtd9: 00010000 00010000 "w-load-SF"
12 mtd10: 00050000 00010000 "u-boot-SF"

```

Listing 5.5: WiPG-1000 `/proc/mtd` file

Additionally, no attempts to connect to the attached router were made. A factory reset was attempted using the provided reset button, but the device did not respond.

5.7.5 5020L and 932L D-Link Cameras

Both cameras split their storage into five partitions, which were helpfully labelled in their respective `/proc/mtd` files. The partitions used in both the 5020L and 932L models were identically named and structured, but differed slightly in size and content. The partitions of the 932L model are shown in Listing 5.6 and Figure 5.13. For both devices, the “Bootloader” partition was chosen for the encryption stage, as it contained a U-Boot bootloader. After encryption, each camera was rebooted. Upon restarting, the in-built LEDs would no longer light

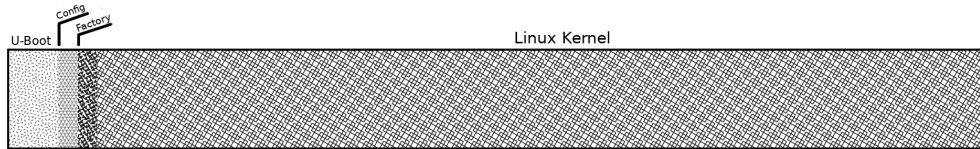


Figure 5.13: MTD partitions in 932L's Flash Memory

during the boot process, and the device's web interface could no longer be accessed. A factory reset was attempted, and the device's LEDs indicated that a reset was successfully performed, but the device remained unresponsive.

```

1 mtd0: 00400000 00010000 "ALL"
2 mtd1: 00030000 00010000 "Bootloader"
3 mtd2: 00010000 00010000 "Config"
4 mtd3: 00010000 00010000 "Factory"
5 mtd4: 003b0000 00010000 "Kernel"

```

Listing 5.6: 932L Camera /proc/mtd file

5.7.6 Summary

Every device that was tested was found to be vulnerable to permanent denial of service attacks. This demonstrates that the method used to encrypt the bootloader is generalisable, and can easily be adapted to target other devices in the future if they are found to be vulnerable. In addition, the majority of factory reset processes provided by the device's vendors proved ineffective when attempting to recover the functionality of infected devices.

When paired with the communication hijacking techniques shown in Chapter 4, PaperW8 demonstrated an effective and destructive form of IoT-based ransomware that could feasibly be used in large scale malware campaigns.

This style of ransomware could also negatively impact the public perception of IoT devices. Unlike most other IoT-Based malware, infections are very obvious and have very tangible consequences for those that are infected.

5.8 Limitations

While PaperW8 successfully demonstrated the feasibility of this approach when creating IoT-based ransomware, some restrictions may limit its overall effectiveness.

5.8.1 Device Cost and Ransom Pricing

In traditional ransomware, the price demanded by the ransom can vary drastically. This can be partly attributed to the worth of the encrypted assets being highly subjective. Price can also be impacted by the victim’s income and willingness to pay [49; 120].

In PaperW8’s case, however, rather than encrypting valuable files, the functionality of the device is disabled. Therefore, the ransom price has an objective upper bound: the retail price of a suitable replacement. If the ransom greatly exceeds the price of a “new” device of the same type and the associated costs, victims may refuse to pay, and instead buy another device like-for-like, or even possibly shift to another brand entirely.

While IoT devices are assumed to be comparatively cheap and replaceable when compared to personal computers, phones or certain valuable user data, this may not always be the case. A smart TV (such as the one mentioned in the previous IoT malware example in Section 5.5.2.1), may be deemed too expensive to replace for some victims. The victim may also be unable to find a replacement, depending on their income or the price and availability of a suitable device.

The decision as to whether to pay may be further influenced by the cost or inconvenience of the disruption caused, such as losses in revenue or production due to a device becoming unavailable, especially if the time needed for a replacement to arrive is also considered. Additionally, if a large number of devices of a particular brand are infected at the same time, the devices’ developer may not have the stock or resources to provide replacements or repairs to affected customers.

5.8.2 Premature Rebooting

In order to effectively ransom the victim, the ransom note must instruct them to not restart the device, as doing so will effectively cause the device to be bricked.

However, victims may choose to ignore these warnings, either due to panic or a lack of trust. In the event of a premature reboot, PaperW8 will not be able to recover the device, and there will no longer be any motivation for the victim to pay the ransom, which may limit the effectiveness of the ransomware.

Future malware could modify the filesystem so only the functionality of the device is disabled when booted. This would allow the victim to reboot, but would require the attacker to gain persistence for each targeted device, and selectively choose which applications should be allowed to run – all these would increase the complexity of development. It may also be easier to recover the device via flash memory editing or a factory reset when this method is used, as while the filesystem may be modified, it will still be readable due to the lack of encryption. This could allow victims aiming to recover the device to make their own modifications without an accompanying decryption key.

5.9 Conclusions

In this Chapter, the viability of implementing bricking-based ransomware on IoT devices was explored, previous instances of IoT-based ransomware were examined, and the limitations presented by IoT devices were investigated. A method to circumvent these limitations was presented in the form of PaperW8, an IoT-based ransomware that uses communication hijacking to communicate with victims, and can brick devices by encrypting flash partitions that store content essential to the continuing functionality of the device.

PaperW8 was then tested against six different IoT devices to demonstrate the viability of the attack. The method used to ransom infected IoT devices was shown to be highly destructive and adaptable, such that it successfully “locked” multiple Linux-based IoT devices of different brands and purposes without significant modifications. Finally, conventional methods of recovery were attempted, such as restarting or performing factory resets on infected devices, which were found to be unsuccessful.

Chapter 6

Persistence in Linux-Based IoT Malware

Based on content of previous publication:

“Persistence in Linux-Based IoT Malware” [42]

6.1 Introduction

Unlike desktop-based malware, most IoT malware families (including the proof of concept malware discussed in Section 5.6), do not exhibit the ability to gain *persistence*. Instead, most IoT malware is wiped from compromised devices when they lose power, as they are often stored within volatile memory.

As such, a standard piece of advice typically given to affected victims for removing malware from their IoT devices is to restart the device in question. This will clear any data in volatile memory, which is often where IoT malware is stored. IoT malware that exhibits persistent capabilities, however, would be much more difficult to remove and could lead to more damaging variants being produced.

This Chapter examines IoT persistence and its role in IoT-based malware, followed by outlining the challenges currently preventing most IoT-based malware from gaining persistence. Methods that could be used by attackers to gain persistence are then investigated, followed by an examination of their limitations, and practical tests on four vulnerable IoT devices of differing designs. Finally, it explores how persistence will change malware authors’ approaches, including how

persistence could be achieved and used to perform new and previously infeasible attacks, such as new forms of ransomware.

6.2 Background

While most forms of IoT malware do not implement persistence, it is not impossible to achieve. In this section, persistent and non-persistent malware will be compared. Finally, the effects persistent malware would have on victims is discussed.

6.2.1 Non-Persistent Malware

Popular botnets such as Bashlite and Mirai have infected hundreds of thousands of devices [12; 232]. Fortunately, this type of IoT malware is relatively simple to remove.

As many IoT devices use read-only root filesystems, “temporary” filesystems located in volatile memory are used as a “working area” for files that need to be regularly modified but do not need to be retained. To maximise simplicity and compatibility, non-persistent malware is often loaded into and run from these temporary filesystems [266]. However, as the memory area is volatile, data is not retained when power is lost. By restarting the device, the malware is unloaded from volatile memory, resulting in a “clean” device when it reboots.

To supplement devices disconnected from the C&C by these restarts, such non-persistent malware often exhibits worm-like behaviour [12], using the device to scan the Internet for further victims to infect.

While victims may restart their devices (either deliberately or coincidentally) and clear the infection, it would not remove the underlying issue. While the volatile memory is cleared, the persistent memory is not. Therefore, any vulnerabilities that were originally exploited by the attacker to install the malware would persist, and the devices could easily be reinfected from an external source, possibly within minutes [60].

In effect, this behaviour has led to competition between botnet authors, as each author aims to maximise their share of the limited number of vulnerable IoT devices. Some malware even implements security features to remove competing

malware from infected devices, or prevent other malware authors from gaining access [12; 269].

Mirai, for example, searches the memory of other processes for known strings present in competing malware [134], then kills any that are discovered. It also closes potentially vulnerable services running on certain ports, preventing competitors from performing further attacks [135]. It should be noted that these defensive techniques are *also* non-persistent, and would be removed should the device be reset, returning it to a vulnerable state.

6.2.2 Persistent IoT Malware

If persistent IoT malware becomes more prevalent, it will likely be much harder to remove from IoT devices. IoT malware capable of making persistent changes to secure its presence would be able to maintain control of devices through reboots, both removing the requirement to reinfect the equipment and helping towards keeping competitors at bay, as measures used to prevent competitors from exploiting the device would also be retained. Therefore, it is increasingly important for IoT developers both to understand their devices' potential vulnerabilities to persistence techniques and to implement preventative measures to inhibit attackers from exploiting them.

Currently, restarting an infected device will remove the majority of IoT malware. To remove persistent malware, however, the victim would have to modify the flash memory of the device to remove the infection. This is usually not easy nor practical for the average user to achieve. Typically, the only reasons users have to modify flash memory are to:

- Update the device to the latest firmware version.
- Factory reset the device.
- Modify user configuration settings, which need to be retained after a reboot.

Unless the persistence method used by the attacker relies on the contents of the user configuration, making modifications to the configuration is very unlikely to allow for the recovery of the device. Updating or factory resetting the device, on the other hand, depending on how the associated processes are implemented

by the developer, may “overwrite” the infected partitions with uninfected content, effectively removing the malware.

However, if the malware can prevent these processes from being used, or is stored in a partition that is not affected by them, specialist equipment or access to debug/programming interfaces may be required to manually clear the infection, such as a UART connection to interact with the shell, or a JTAG interface which may provide access to the flash memory [141]. This would be considered too complicated for most IoT users to perform, which may lead to infected IoT devices being discarded, or worse, knowingly left in an infected state.

Persistence may also lead to improvements in existing malware implementations. In Chapter 5, the limitations of non-persistent IoT malware were discussed in-depth, but many of these limitations could be easily circumvented if persistence is achieved. For example, IoT ransomware may no longer need to rely on partition encryption as its main method of asset control, and instead stop certain applications from booting at startup to prevent the device from being used. Attempts to regain control of an infected device by rebooting could even be punished by the attacker, such as by increasing the cost of the ransom payment.

Certain types of IoT malware that would have previously been considered infeasible may also be explored by attackers. For example, malware could use the long term access to collect private data to facilitate blackmail (which is explored further in Chapter 7), or provide the means for malware operators to install additional malicious features at a later date, such as modules to attack other devices on the infected device’s network.

6.2.3 Challenges with Gaining Persistence

There are two key challenges currently faced when attempting to gain persistence on IoT devices: “Read-only” states, and device variance. These challenges and why they can be difficult to overcome are explained below.

6.2.3.1 Read-Only Flags

IoT devices often have data that is set to be read-only, which can prevent accidental modifications due to programmer or user errors. The decision to make certain data read-only may also unintentionally prevent attackers from making the

necessary modifications to stored data or filesystems in order to gain persistence.

Memory or storage being “Read-Only” is normally determined by three factors: the filesystem, the MTD subsystem, or file permissions.

- **Filesystems**

IoT devices often store their filesystems on flash chips, which are known to be relatively cheap, fast and compact, and therefore ideal for use in embedded systems [164]. These chips allow devices to save files that are essential to the running of the device (such as the bootloader, kernel and filesystem) and any additional configuration information used by any running applications.

To make a change to flash memory, the block the user is attempting to modify must first be wiped before the new data can then be rewritten. This process is often referred to as a “cycle”. One limitation of flash memory is that after a certain amount of cycles, a block of memory can become “worn”, leading to the data stored within it being permanently corrupted and left in an unreliable state [186]. To extend the lifetime of the chip, developers have implemented various strategies to limit the “wear” caused by repeated cycles.

For example, some blocks may contain data that needs to be regularly updated, leading to them being disproportionately worn. To mitigate this issue, wear levelling can be used to spread modifications more evenly across other memory areas on the chip [186].

To further preserve the storage of the device, developers may use compressed “read-only” filesystem formats. Files that are unlikely to be changed (such as utilities, system binaries or device applications) can be kept in these filesystems, while a separate writeable filesystem located in RAM, such as `tmpfs` [217] or `ramfs` [159], can be used to store files that are regularly modified, but do *not* need to be persistent.

Files stored in RAM-based filesystems are expected to be temporary and will be wiped when the device is restarted [230]. This design structure reduces the number of changes that need to be made to the flash chip, preserving it for longer.

The potential benefits and drawbacks of the different filesystem formats are covered in more depth in Section 6.3.

- **MTD Partitions**

As previously mentioned in Section 5.6.3, flash memory for Linux-based devices is normally managed by the MTD subsystem, which can be used as an abstraction layer to transparently interact with different types of flash storage [21]. MTD can also be used by developers to separate the flash chip into several “partitions” based on their purpose.

When defining these partitions, the developer can also set a number of flags to determine how they will behave. One such flag, `MTD_WRITEABLE`, dictates whether the partition is writeable [166]. If it is unset for a partition, any attempted modifications will be prevented by the kernel.

Unlike files or filesystems set to read-only, the state of the partitions’ flags is managed by the kernel, which makes it difficult to modify from userspace. This may prevent attackers from being able to make persistent edits to partitions containing important data.

- **Permissioned Systems**

While not strictly a “Read-Only Flag”, permissioned systems can prevent users from modifying files if they do not have the necessary privileges. This could prevent an attacker from gaining persistence if they do not have permission to create or modify certain files or directories, although they may be able to escalate their privileges to circumvent this restriction.

6.2.3.2 Variance

As previously mentioned, IoT devices are likely to use different hardware, update mechanisms, software, architecture and filesystem types appropriate for their purpose. Fortunately for IoT developers, this variation limits the attackers’ ability to create a universal method for gaining persistence.

While it may be possible for attackers to search for bespoke weaknesses or vulnerabilities in a targeted IoT device that would allow them to gain persistence, the time investment required to discover and develop customised implementations for each targeted device model may dissuade attackers.

However, if a method were developed that would be able to affect a large number of devices with similar implementations, it could reduce the time investment when developing the malware immensely. This would make implementing persistence in IoT malware much more appealing and may lead to persistent IoT malware becoming more common.

6.2.4 Previous Persistent IoT Malware

After identifying an increase in the presence of Linux-based malware, researchers analysed 10,548 samples over the period of a year to gain a better understanding of the techniques used by malware authors [67]¹. The results highlighted the quick development and deployment of insecure IoT devices as a potential motive for attackers to target Linux for malware development.

As part of this analysis, it was found that 21.10% (1,644) of the analysed samples implemented some form of persistence method. Some of the techniques used modified startup service configurations (such as “rc” files², `.bashrc`, or `crontab`), or simply replaced or infected existing files and utilities.

While some of these methods could theoretically be applied to IoT devices, the targeted device must implement a writeable filesystem. As mentioned in Section 6.2.3.1, IoT devices often set certain data as read-only, which would prevent these methods from working.

Despite the challenges presented by IoT devices, previous persistent IoT malware observed in the wild has used such methods. However, they are less common and relied on the availability of writeable filesystems, which may reduce the malware’s applicability. Two examples of persistent IoT malware are examined below.

6.2.4.1 Torii

Torii is a Mirai variant that added several features to the original malware. Most notably, it introduced six techniques that could potentially gain persistence on infected devices [128]. Each technique would modify files on the infected device that managed parts of the boot process, such as [156]:

¹It should be noted that this study did not focus on IoT explicitly, as it included desktop-based Linux malware.

²Also known as “run command” files, which are scripts that run after the system startup completes.

- `.bashrc`, which is executed whenever an interactive bash session is started;
- `inittab`, which is used to determine which processes should be run during the Linux boot process at certain runlevels³.
- `crontab`, which is used to execute files at a certain time or time interval.
- The `/etc/init` directory, which is traditionally used by the “Upstart init” daemon.
- Creating a “System Daemon” service, which is run by `systemd` on startup.
- Creating an SELinux policy to run the malware on startup.

Modifications to these files allow attackers to set particular programs, shell scripts, or commands to be run when the device boots.

6.2.4.2 VPNFilter

VPNFilter is a complex IoT malware that targeted at least 73 router models from 11 different brands [241]. It is believed to have been developed by “Fancy Bear”, a Russian backed hacker group [260].

VPNFilter exhibits a “modular” structure, which allows it to be very adaptable. After fully infecting a device, it can dynamically add new features by utilising downloadable modules. Through these modules, it can extend its original functionality to perform new tasks, such as various man-in-the-middle attacks, network mapping, and even SCADA sniffing⁴ [240; 241; 239].

VPNFilter also includes a section of code that allows it to erase and rewrite connected flash memory via the MTD subsystem, which could be used to brick the device by wiping segments of the device’s storage [239], not unlike the various bricking malware described in Chapter 5.

In order to gain persistence, VPNFilter modifies the `/etc/config/crontab` file, such that the malware (which has presumably already been written to permanent storage) will be run every 5 minutes [239; 231]. If the device is rebooted,

³A “runlevel” is an integer which determines how a system functions on boot, and which applications should be run.

⁴Supervisory control and data acquisition (SCADA) is a protocol that is commonly used to interact with ICSs.

the `cron` daemon will start, read the appropriate crontab file, and set `VPNFilter` to be run.

6.3 Filesystems

Many of the persistence methods that are discussed in this Chapter interact with the filesystems of infected devices. Understanding the intended purpose of each filesystem can aid the process of determining the best persistence method. Below, an itemised list of filesystems that are often used by IoT devices is provided.

6.3.1 Utility Filesystems

There are many filesystem types used by the Linux operating system for various purposes, such as to store the status of connected devices, expose certain kernel information, or act as a temporary filesystem for files that do not need to be permanently saved. Some of these “utility” filesystems were encountered during this research, and are described in more detail below.

6.3.1.1 Temporary Filesystems

As previously mentioned, temporary filesystems are used to store files that do not need to persist through reboots. These filesystems typically reside within volatile memory, such as RAM, and are often writable by default. This filesystem is often mounted as the `/tmp` directory and is frequently used by various families of IoT malware as a reliable storage location during the infection stage.

- **ramfs**: `Ramfs` is a filesystem that is designed to function entirely within RAM. Due to being stored in volatile memory, all files that are stored within a `ramfs` filesystem are wiped when the device is rebooted [159].
- **tmpfs**: `Tmpfs` is another RAM based filesystem which is based upon `ramfs`. While it is very similar in design, `tmpfs` supports setting an upper limit on the filesystem size, and the use of swap spaces [217].
- **rootfs**: The `rootfs` is a special instance of `ramfs` (or `tmpfs`) which cannot be unmounted. In Linux version 2.6 and earlier, a CPIO format archive is

appended to the kernel, which is extracted into the `rootfs` to make up the basis of the root filesystem [159].

6.3.1.2 Pseudo Filesystems

Some filesystems are not intended to store “real” user files. Instead, they provide access to system and kernel information, such as loaded kernel modules, CPU information, or direct access to connected devices. While the attacker may not be able to directly use these filesystems to gain persistence, they may contribute to ascertaining a device’s vulnerability. For example, as these filesystems expose information about the host device, they can be used to identify valid techniques that could be used to gain persistence.

- **proc**: A virtual filesystem which is often mounted at `/proc`. This filesystem provides userspace access to process and system information. In some versions of Linux, it also provides access to kernel status and parameter information [36].
- **sysfs**: A virtual filesystem which is often mounted at `/sysfs`. It provides userspace access to various aspects of the kernel, including information about attached hardware [180].
- **usbdevfs/usbfs**: Filesystems that are primarily used to communicate with USB devices [16].

6.3.2 Storage Filesystems

During this research, multiple filesystems that provided persistent storage were also encountered. Each had its benefits and drawbacks, which are detailed below.

6.3.2.1 Read-Only

Read-only filesystems are perfect for storing files that do not need to be regularly modified. When these types of filesystems are first created, the files they contain are compressed before storage, which reduces the size of the resulting filesystem.

A side-effect of this process is that it becomes very difficult to modify files directly [20]. Instead, the uncompressed version of the original filesystem must be

obtained, which can then be modified and compressed into a new instance. The new instance can then be used to overwrite the original filesystem.

Below are some of the read-only filesystems that were encountered during this work.

- **cramfs**: A compressed read-only filesystem written by Linus Torvalds, which was designed to be used on embedded systems with limited resources.

cramfs produces very small filesystems but has drawbacks that must be considered during the design stage. For example, in **cramfs** filesystems, files must not exceed 16MB in size, the overall filesystem size has a maximum size limit of 256MB, and file timestamps are not supported [20].

- **squashfs**: A compressed read-only filesystem first released in 2002, as a response to **cramfs**. As **squashfs** boasts additional features and drastically increases the max file and filesystem sizes, it is recommended to replace **cramfs** with **squashfs** where possible [23; 198].

6.3.2.2 Log-Structured

Unlike read-only filesystems, log-structured filesystems allow users to update the files relatively easily. To store data, compressed “nodes” are used to define files and directories. A benefit of this method is that new nodes or data defining changes made to files can be appended to the log, allowing modifications to be made without needing to rebuild the filesystem [275; 172].

As previously mentioned in Section 6.2.3.1, “blocks” of flash memory can become worn out and unreliable after a certain amount of use [186]. To combat this, some log-based filesystems attempt to use all areas of the chip evenly to “spread the wear” over a larger area.

Two log-structured filesystems that were encountered during this work are now examined.

- **JFFS2**: The Journalling Flash File System (JFFS) is a log-structured filesystem released in 2001. It was designed for use on various flash storage devices and allows users to directly edit files within the filesystem through the use of a flash abstraction layer [275].

Despite generally being larger than compressed “read-only” filesystems, each node present in the filesystem can be compressed using a variety of algorithms.

- **YAFFS2**: Yet Another Flash File System (`yaffs`) is specifically designed for use on NAND flash devices, and has a similar use case to JFFS2 [8]. However, `yaffs2` does *not* support compression, which may lead to larger filesystems [139].

Both `jffs` and `yaffs` have associated “MTD user modules”, which allow users to mount and modify the filesystem without having to directly interact with the flash chip [176]. Any file modifications made in memory are transparently written to the filesystem stored in flash.

6.3.2.3 Other

- **ext2**: The “second extended file system” (`ext2`) is a filesystem that was previously used by default in older Linux distributions. It has wide support, but is not very space efficient and has no journaling capacity, although this can be remedied by using the related filesystem type: `ext3` [219].

6.4 Device Enumeration

Due to the challenges described in Section 6.2.3, no universal methods to gain persistence on IoT devices have yet been identified.

Instead of attempting to create such a method, this research aimed to investigate the possibility of using a *collection* of methods that can gain persistence on *certain subsets* of IoT devices. As this could allow attackers to easily adapt their approach without needing to create a bespoke solution for each device model.

However, as shown in the previous section, devices can vary significantly in their implementation. While it would be possible to simply attempt each method sequentially until persistence is achieved, this could be potentially damaging to the targeted device. Instead, to ascertain which persistence method should be used, an attacker can perform reconnaissance to gather relevant key information.

Ideally, the attacker should be able to identify the device model and determine the storage capabilities of the targeted device. They must also have a privilege

level such that they can identify, access and modify the resources required for the selected methods to succeed, such as the filesystems and partitions stored in flash memory.

Below are some of the techniques that can be used to gather information about a target device and determine the best method to gain persistence.

6.4.1 /proc/mtd

As previously demonstrated in Chapter 5, the `/proc/mtd` file contains the definition of the various MTD partitions. This includes names set by the developer, which may indicate each partition's purpose.

Partitions can be accessed directly via files presented in the `/dev` directory. Each partition is listed as `mtdX` and `mtdblockX` files, where “X” is the index of the partition.

6.4.2 /proc/mounts

The `/proc/mounts` file provides a list of all of the mounted filesystems. Most importantly, for each file mount, it also provides the mount point, device type, filesystem type, and whether it is read-only or writeable, all of which can be useful information for the attacker.

6.4.3 Binwalk

In the cases where the previous files do not adequately outline the storage configuration of the device, it may be possible to extract and analyse individual partitions with tools such as Binwalk [215].

As mentioned in Section 5.7.3, Binwalk can be used to analyse binary files to identify useful information, such as file structures and image headers. This may allow an attacker to determine the function of a certain partition or extract filesystems for inspection.

6.4.4 Startup Scripts

Before modification of any discovered filesystems can be performed, the attacker must first examine its contents to determine whether any valuable changes can

be made. While the ability to store new files is worthwhile, the final objective is to execute custom code, preferably each time that the system is rebooted, which would allow the installed malware to maintain activity on the device.

Scripts that control the startup process of the targeted device (such as the crontab file, or `init.d`'s `rcS` file) should be the first targets for modification, as after storing the malware on the device, these scripts can be modified to run the malware on boot. If modifying a startup script is not possible, the attacker may be able to replace an existing binary that is run at startup with the required malware, before returning control to the original application. However, this technique was not required during this work.

6.5 Methods of Gaining Persistence

Once the characteristics of the targeted device have been enumerated, the attacker must then choose an appropriate method that is most likely to gain persistence.

Below are several viable methods that could be used by an attacker to gain persistence on a variety of IoT devices. A summary of these methods can be found in Table 6.1, and a detailed overview of each is provided in the following subsections.

ID	Method	Modified Partition	Ease of Use
A	Modifying Writeable Filesystems	Filesystem	Easy
B	Recreating Read-Only Filesystems	Filesystem	Medium
C	Initrd/Initramfs Modification	Kernel	Hard
D	“Set Writeable Flag” Kernel Module	N/A	Hard
E	Update Process Exploitation	Filesystem/Kernel	Device Dependent ⁵
F	UbootKit	Bootloader	Hard

Table 6.1: IoT Persistence Methods

⁵Using this method requires the attacker to exploit the targeted device's update process. As the update process will differ for each device, the complexity of the required exploit will also vary.

Each method includes a description, a list of *requirements* for its applicability, its *feasibility*, and any *potential issues* that may prevent it from working effectively.

The described techniques assume that the attacker has gained access to the shell of the targeted device (such as via a guessable telnet password), and can run arbitrary commands.

6.5.1 Modifying Writable Filesystems

If an IoT device has a writeable root filesystem, and the attacker has the necessary permissions, they will likely be able to directly modify the filesystem via the shell. If it is possible to edit important files that are used as part of the boot process (as discussed in Section 6.4.4), the attacker will be able to force actions to be taken after the device has been rebooted.

6.5.1.1 Requirements

- The device must use a writable filesystem, such as those listed in Section 6.3.2.2 (e.g. `yaffs2/jffs2`).
- The MTD filesystem partition `MTD_WRITEABLE` flag must be set.
- The attacker must be able to modify files that are run or used as part of the boot process.

6.5.1.2 Feasibility

This is a simple method that does not require any additional tools. If the filesystem is writeable by default, the attacker will be able to copy their malware to a known location on the device and modify files involved in the boot process such that the malware is executed when the device is rebooted.

This is similar to the technique used by previous IoT malware such as `Torii` and `VPNFilter`, as described in Section 6.2.4.

6.5.1.3 Potential Issues

- **Filesystem Permissions.** The attacker must be able to obtain write permissions for any files that they are attempting to modify. This is typically

dependent on the privileges held by the compromised account or application exploited by the attacker.

- **On-Boot Execution.** The discovered writable filesystem must contain files that can lead to the execution of arbitrary code during the boot process. Otherwise, while the attacker may be able to persistently store malware on the device, it will not be run after a reboot.
- **Read-Only Mount** While the device may implement a “writeable” filesystem, it may be mounted as read-only. Additional steps may be required to remount it as writable.

6.5.2 Recreating Read-Only Filesystems

Devices may make use of read-only filesystems to preserve storage devices or space. In this case, it will not be possible to modify files contained within these filesystems directly. However, by using specialised tools, attackers could extract and re-create a “new” filesystem of the same format.

6.5.2.1 Requirements

- The device must use a compressed read-only filesystem (e.g. `cramfs/squashfs`).
- The attacker must be able to modify the flash partition which contains the read-only filesystem.
- The attacker must be able to create new filesystems of the same format as the original.

6.5.2.2 Feasibility

While it is not possible to modify files contained *within* compressed read-only filesystems, it is possible to replace the *entire* filesystem with a modified version of the same type.

First, the attacker must obtain the original compressed version of the filesystem, which could potentially be obtained by extracting it from flash memory, or from a previous firmware update provided by the device developer, which is

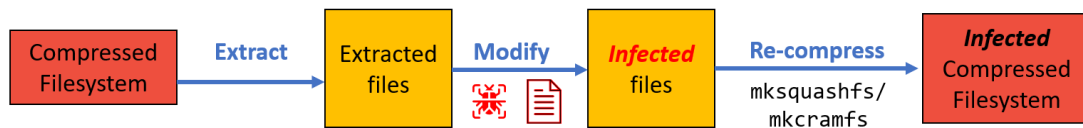


Figure 6.1: Process graph for modifying a compressed filesystem

often made available via online download. Once the attacker has extracted the original filesystem, the uncompressed version can be modified to the attacker’s requirements.

Finally, the attacker can compress the modified filesystem into a new *compressed* filesystem of the same format as the original, using the appropriate tools. For `squashfs` and `cramfs` filesystems, this would require the use of the `mksquashfs` and `mkcramfs` utilities respectively. The partition containing the original filesystem can then be overwritten with the new version.

6.5.2.3 Potential Issues

- **Matching Filesystem Format.** Filesystems can vary significantly, even between versions of the same format. If the replacement filesystem type differs from what is expected by the device, it may not be correctly interpreted, which will lead to failures during the boot process. For this approach to succeed, the attacker must match the expected filesystem as closely as possible.
- **Remote Filesystem Transfer.** Read-only filesystems may prove challenging to modify, as it is unlikely that the tools used to create the filesystem will be included on the targeted device. In the cases where a new filesystem might be required during normal operation, such as a device update, it would be expected that a remote machine would instead be used to generate it.

The attacker can use a similar process by copying the filesystem from the infected device to an external computer, which will be able to install the required tools to make modifications, then uploading the infected filesystem back to the device.

Filesystems are typically much larger than the average malware binary, and

as a copy will need to be provided to each infected device, this approach might not scale well. However, if a “known” compressed filesystem that is identical to one that has been seen in the past is encountered (which can be verified using a checksum, for example), the attacker can instead upload a previously generated infected version without having to download, modify, and generate a new filesystem each time.

Another possible solution is to upload the malware and the required tools to generate the malicious filesystem to the device, such that the new filesystem can be created natively. However, the tools must be compiled to be compatible with the target’s architecture, and the device must have the resources and storage space to locally run the tools. As there are many different architectures and filesystem formats, this may not be easy to manage.

6.5.3 Initrd and Initramfs Modification

As part of its booting process, the Linux kernel may include an appended `initrd` or `initramfs` filesystem [159]. This can be used as the primary root filesystem, or to perform some setup of the device before the “real” filesystem is mounted. The attacker can attempt to modify this appended filesystem so that arbitrary code is executed when it is mounted as part of the boot sequence.

6.5.3.1 Requirements

- The device must use an `initrd` or `initramfs` filesystem.
- The attacker must be able to modify the flash partition that contains the kernel.

6.5.3.2 Feasibility

First, the attacker must identify an MTD partition that contains a Linux kernel. Once a partition has been identified, the attacker must extract its data and locate the offset of any appended filesystems. After carving⁶ out the relevant data, both

⁶“Filesystem carving” is a process that can be used to extract useful information from raw data, with limited knowledge as to its format. In this example, it is assumed that the attacker must identify metadata at an offset within the extracted partition data, such that they can extract the targeted filesystems.

the kernel and filesystem must be saved separately.

The attacker then extracts the original filesystem and adds their chosen malware. The attacker then reverses the extraction process and appends the resulting filesystem to the kernel that was previously saved. The final result can then be used to overwrite the kernel flash partition.

6.5.3.3 Potential Issues

- **Multiple Extraction Steps.** Obtaining the original filesystem may require multiple extraction steps. `Initramfs` filesystems are often contained within a CPIO archive, which is likely to be compressed with a format of the developer's choice.
- **Image Formats.** The kernel may be stored on the flash chip as an image, with headers that instruct the bootloader on how to boot the partition effectively. This may require the attacker to take additional steps to recreate the image format for the modified filesystem to be compatible.
- **Remote Filesystem Transfer.** As mentioned in Method B (Section 6.5.2.3), if filesystem modifications cannot be performed locally, large amounts of data will need to be transferred to and from the device, which is unlikely to scale well.

6.5.4 “Set Writeable Flag” Kernel Module

MTD can be used to manage partitions of flash memory. As mentioned in Section 6.2.3.1, developers may unset the `MTD_WRITEABLE` flag for partitions that do not need to be modified. This may also incidentally prevent attackers from making modifications to partitions that would allow them to gain persistence.

If the requirements are met, this method would allow an attacker to re-enable the `MTD_WRITEABLE` flag from within userspace. While this method does not directly allow an attacker to gain persistence on its own, it may allow other methods to circumvent the read-only protections put in place by the developers.

6.5.4.1 Requirements

- The Linux kernel must support loadable modules.

- The attacker must have permission to load new kernel modules.
- While not a “true” requirement, having access to a device’s kernel header files or source tree will improve the kernel module’s odds of being compatible.

6.5.4.2 Feasibility

The MTD_WRITEABLE partition flags can be difficult to modify from userspace at runtime. However, by using a Loadable Kernel Module (LKM), it is possible to force them to be set from kernel space.

A couple of previous projects have been able to implement this [188; 133], but compiling a kernel module has a few requirements that may make it difficult for an attacker to achieve.

Kernel modules typically need to be compiled against the targeted kernel source to be compatible. This is normally achieved by having access to either the kernel’s headers or source tree [19].

If IoT developers make use of modified software that falls under the GNU Public License (GPL), they may be required to make that corresponding source code available [228]. The attacker can use this code to compile a custom kernel module for the targeted device model.

After compiling and uploading the module to the target device, the attacker can use the `insmod` utility to insert it into the kernel. Once inserted, the module will set the MTD_WRITEABLE flags in each of the MTD partitions, allowing changes to be made. The attacker can then attempt other persistence techniques that previously would have been prevented.

6.5.4.3 Potential Issues

- **Unavailable Source Code.** If the device’s kernel headers and source code are unavailable, it may be difficult to compile the LKM such that it is compatible with the targeted kernel.

It is not *impossible*, however, as a defensive IoT tool “HADES-IoT” demonstrated that loadable kernel modules could be compiled without the support of the original developer [39].

- **Module Verification.** The developer may be able to prevent this method from being used by configuring the Linux kernel to verify the signature of loaded kernel modules [22].

As long as the attacker does not have access to the developer’s cryptographic keys, they will not be able to forge a signature for their malicious modules, preventing the attack from succeeding.

6.5.5 Update Process Exploitation

Most devices are expected to receive updates over their lifetime, either to provide the user with new features or to patch discovered security issues. However, vulnerable update implementations can potentially be attacked to gain persistence. For example, if updates are not signed by the developers, attackers may be able to create “false” updates with malicious content.

6.5.5.1 Requirements

- The device must implement a vulnerable update function, such that the attacker can forge fake updates.
- The attacker must be able to access the vulnerable update function.
- The attacker must be able to generate a false update of the correct format such that it will correctly run on the device.

6.5.5.2 Feasibility

To exploit this vulnerability, an attacker creates a “false” update that contains the chosen malware. The malicious update is uploaded to the vulnerable update process, which is then applied by the targeted device. The device is then rebooted and runs the malware on startup.

However, this does require the attacker to match the format expected by the update process, which in some cases may require the attacker to re-construct metadata headers, such as filesystem checksums.

This method has been previously demonstrated by researchers that discovered vulnerabilities in devices produced by Disney [38], Foscam [37], and Netgear [33],

which allowed them to upload modified firmware and take control of the devices. In one case, a switch developed by Netgear was configured to run a `tftp` server by default, as the server did not require authentication, attackers could easily upload unsigned firmware updates to gain persistence and execute malware [63].

6.5.5.3 Potential Issues

- **Specificity.** The requirements for this method to be effective are quite niche. It not only requires that the attacker has access to the update process (for which they will likely need to be authorised), but the process itself must also be vulnerable in such a way that updates are not adequately verified before being implemented.
- **Update Generation.** As the update process differs from device to device, what may work for one is very unlikely to work on another. The attacker will need to reverse-engineer the required format of the updates for each targeted device’s update process. If the forged update is incorrectly formatted, the update process may be halted, preventing the attacker from gaining persistence.

The attacker could attempt to modify the filesystem of an existing firmware file, but the update process may also need to interpret metadata defined by the developer. As such, the attacker would need to recreate the required metadata, such as filesystem sizes or checksums. Some tools are available that may assist in this process, such as the “Firmware Mod Kit” project, but this will not work for all update formats, especially if the developer has obfuscated, encrypted or signed the firmware that they have made available.

6.5.6 Ubootkit

Das U-Boot (Normally shortened to “U-Boot”) is a universal bootloader, designed to be used with a variety of embedded devices to manage the process of booting into the main operating system [76]. In 2018, researchers at Tencent discovered a method of modifying U-Boot to hijack the booting process and run arbitrary code written by the attacker [279; 280].

6.5.6.1 Requirements

- The device must implement U-Boot as its bootloader.
- The attacker must be able to modify the partition that contains U-Boot.

6.5.6.2 Feasibility

Researchers have produced an attack that demonstrates the creation of a persistent bootkit with root-level access for IoT devices, dubbed “UbootKit” [280; 281].

UbootKit targets the bootloader partition, allowing attackers to corrupt the boot process of the device. First, the attacker must modify the U-Boot partition, such that after the kernel has been loaded into memory, custom assembly will be run.

The new assembly code “Hot Patches” the `Init_post` function within the kernel, before passing control to it. Once the kernel has completed setting up the filesystem, the code injected into `Init_post` appends custom shell commands to the `/etc/init.d/rcS` bash script, which is run as part of the Linux boot process.

Finally, the commands appended to the `rcS` file download and run an application of the attacker’s choosing.

6.5.6.3 Potential Issues

- **Complexity.** The authors of Ubootkit demonstrated their techniques on a MIPS based platform running U-boot version 4.1.2.0 and Linux kernel version 2.6.21. The researchers stated that the technique could be applied to other devices and architectures than those used in the demonstration [280], but that it would require modification.

This technique requires patching the bootloader and kernel of the targeted device with new shellcode at specific offsets. As the bootloader, kernel and architecture will differ slightly on each targeted device model, this attack may be difficult and time-consuming to manually perform. For this method to be effective in large scale attacks targeting multiple device models, some level of automation would be required.

6.6 Experimental Proof of Concepts and Results

As part of this research, a variety of IoT devices were chosen to test the viability of the methods described. For persistence to be considered a realistic attack method, the following two constraints were applied during testing:

- No physical access to the device must be required. Persistence must be achievable remotely, preferably over the Internet.
- The method of persistence must allow the attacker to force the device to run a custom application when the device is rebooted⁷.

6.6.1 Netgear R6250 Router

(Methods B and D in Table 6.1)

The R6250 router is one of many Netgear router models that had a command injection vulnerability present in their web server [194; 158]. This vulnerability was used to gain access to the shell so that reconnaissance could be performed.

6.6.1.1 Information Gathering

As shown in Listing 6.1, the `/proc/mounts` file indicated that the router used `jffs2` and `squashfs` filesystems. Initially, the `jffs2` filesystem seemed to be the best option as, due to it being log-structured, it was more likely to be writeable by default.

⁷During this work, all devices exhibited root filesystems that contained startup scripts and executables that were run on boot. Therefore, being able to modify these filesystems would allow an attacker to gain persistence.

```

1 rootfs / rootfs rw 0 0
2 /dev/root / squashfs ro,relatime 0 0
3 devtmpfs /dev devtmpfs rw,relatime,size=127056k,nr_inodes
  =31764,mode=755 0 0
4 devfs /dev tmpfs rw,relatime 0 0
5 proc /proc proc rw,relatime 0 0
6 sysfs /sys sysfs rw,relatime 0 0
7 ramfs /tmp ramfs rw,relatime 0 0
8 devpts /dev/pts devpts rw,relatime,mode=600 0 0
9 usbdeffs /proc/bus/usb usbfs rw,relatime 0 0
10 /dev/mtdblock17 /tmp/openvpn jffs2 rw,relatime 0 0

```

Listing 6.1: R6250 /proc/mounts file

Unfortunately, the only **jffs2** filesystem instance was mounted at **/tmp/openvpn**, and only contained configuration files. While it was possible to make modifications to this directory, none of the files that it contained would be able to cause any arbitrary execution when the device was rebooted.

Instead, the **squashfs** filesystem was targeted, which was mounted as the root directory. By reading the partition index within the **/proc/mtd** file, an associated flash partition named “rootfs” was identified, which was likely where the root filesystem was stored. After extracting the data from this partition, it was found to contain a **squashfs** version 4.0 filesystem, compressed using the **xz** algorithm.

6.6.1.2 Gaining Persistence

After extracting the **squashfs** data to obtain the original filesystem, a file named “**testfile**” was added in the **/bin** directory. The filesystem was then repackaged into the correct compressed format using the **mksquashfs** utility with the command:

```

1 sudo mksquashfs squashfs-root/editedSquash -comp xz

```

The generated filesystem was then uploaded to the temporary memory of the router. Finally, the existing filesystem partition was overwritten by piping the modified **squashfs** filesystem to the associated **/dev/mtdblock15** file.

After rebooting the device, the **testfile** was present and able to be read,

indicating a successful persistent edit to the root filesystem.

6.6.1.3 Read-Only MTD Partitions

During the exploitation of the device, it was discovered that some of the partitions, notably the bootloader, had been marked as read-only via the MTD subsystem. While not strictly necessary to gain persistence, as the previous method had already succeeded, it was decided that this would be a good opportunity to test the loadable kernel module method (Method D in Table 6.1).

There had been previous attempts at creating a kernel module for this purpose, with source code readily available online [188; 133]. For this device, the “`mtd-rw`” project was chosen for creating the kernel module. However, to compile the project such that it would be compatible with the device, the source code or kernel headers used during the development of the device were required.

As some of the software used by the router was under the GPL license, Netgear was required to make its source code publicly available. This was relatively easy to find, as it was hosted on their main website alongside their documentation [192].

After downloading the provided source code, the `mtd-rw` project was successfully compiled, and the generated module was uploaded to the router. Inserting the module was relatively simple, as the `insmod` utility was already available on the device. After insertion, it was confirmed that all the MTD partition flags had been set to “writable”, which would allow the attacker to make modifications to the previously “locked” partitions.

6.6.2 D-Link DCS-932L

(Method C in Table 6.1)

Similar to previous proofs of concept, a buffer overflow vulnerability in the web service of this web-connected camera was used to run arbitrary commands. With this access, it was possible to investigate how the camera managed its storage and determine the best method to gain persistence.

6.6.2.1 Information Gathering

The `/proc/mounts` file indicated that only temporary and pseudo filesystems were being used. This implied that the device was using `rootfs` as its main filesystem,

and was booting from data appended to the end of the kernel. Therefore, in order to gain persistence, the appended data (an `initramfs` filesystem) needed to be modified.

The `/proc/mtd` file was read to identify the correct flash partition, which was helpfully labelled as “kernel”. After extracting the partition data to the external attacking machine, Binwalk [215] was used to identify the filesystem structure and offsets. After some further experimentation, it was discovered that the original filesystem could be extracted in three stages, as shown in Figure 6.2.

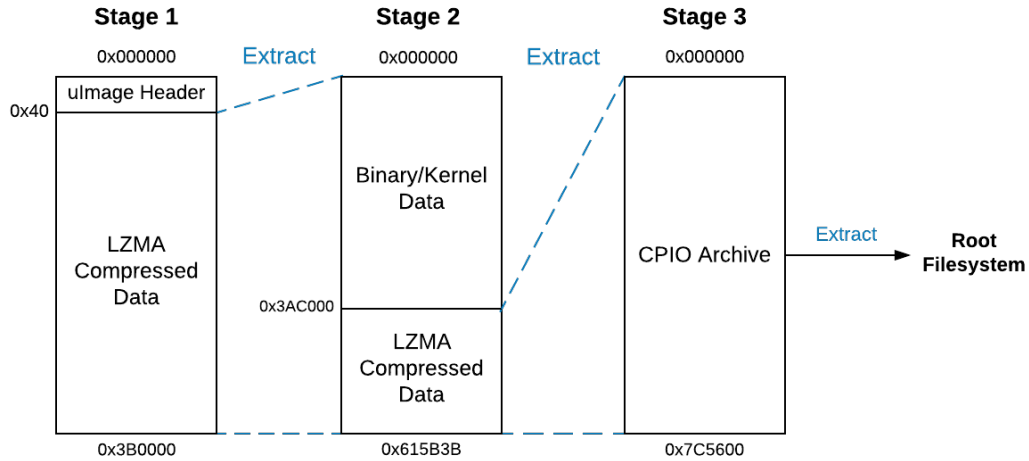


Figure 6.2: DCS-932L Filesystem Extraction Stages

- Stage one represents the raw data extracted from that partition as it was stored in the flash chip. After performing an analysis using Binwalk, it was determined that the file was composed of a 64-bit uImage Header, and LZMA compressed data. The uImage header contained metadata that was used by the U-Boot bootloader to boot the kernel, which was contained in the LZMA payload. Extracting the LZMA compressed data led to stage two.
- Stage two consisted of the kernel data and some further appended LZMA compressed data at offset 0x3AC000. The kernel data and the LZMA data were separated and saved, and the LZMA data was then extracted into stage 3.

- The extracted LZMA data revealed a CPIO archive which, when extracted, provided the root filesystem of the device.

6.6.2.2 Gaining Persistence

To gain persistence on the camera, the kernel partition needed to be altered in such a way that the device would be able to boot and mount a modified version of the appended filesystem. To test this, the filesystem that was extracted from the original partition was modified in a similar manner to the previous example, by adding a `testfile` to the `/bin` directory.

After the modifications had been made, the extraction stages were reversed to re-create the filesystem format required by the device. First, the filesystem was compressed into a CPIO archive with the command:

```
1 sudo pax -w -x sv4cpio -s '^^_filesystem.extracted*^^p'
   _filesystem.extracted > ~/cpio test
```

Next, the archive was compressed using LZMA⁸. The resulting LZMA archive was appended to the original kernel data, and the whole package was compressed into another LZMA archive.

Finally, the uImage header needed to be replaced, such that the U-Boot bootloader would be able to correctly configure the kernel during the booting process. Initially, the original uImage header was used to maintain as much similarity to the original filesystem as possible. However, this caused a boot failure during startup.

Thankfully, an exposed UART port was present on the device, which allowed basic debugging to be performed. UART is a serial connection that allows asynchronous communication to be performed between two devices with as few as two wires. In this case, a 4 pin header was discovered on the main PCB, as shown in Figure 6.3.

After connecting the UART port, it was discovered that the boot process was failing due to a checksum calculation error. uImage headers include two checksums that uBoot uses to check the integrity of attached images: one for the uImage header and one for the kernel and filesystem data [75]. As the filesystem

⁸The compression used by the device was “non-streamed”. To recreate the compression method as accurately as possible, an old version of “LZMA utils” which still supported this option was used. (Available at: <https://tukaani.org/lzma/>)

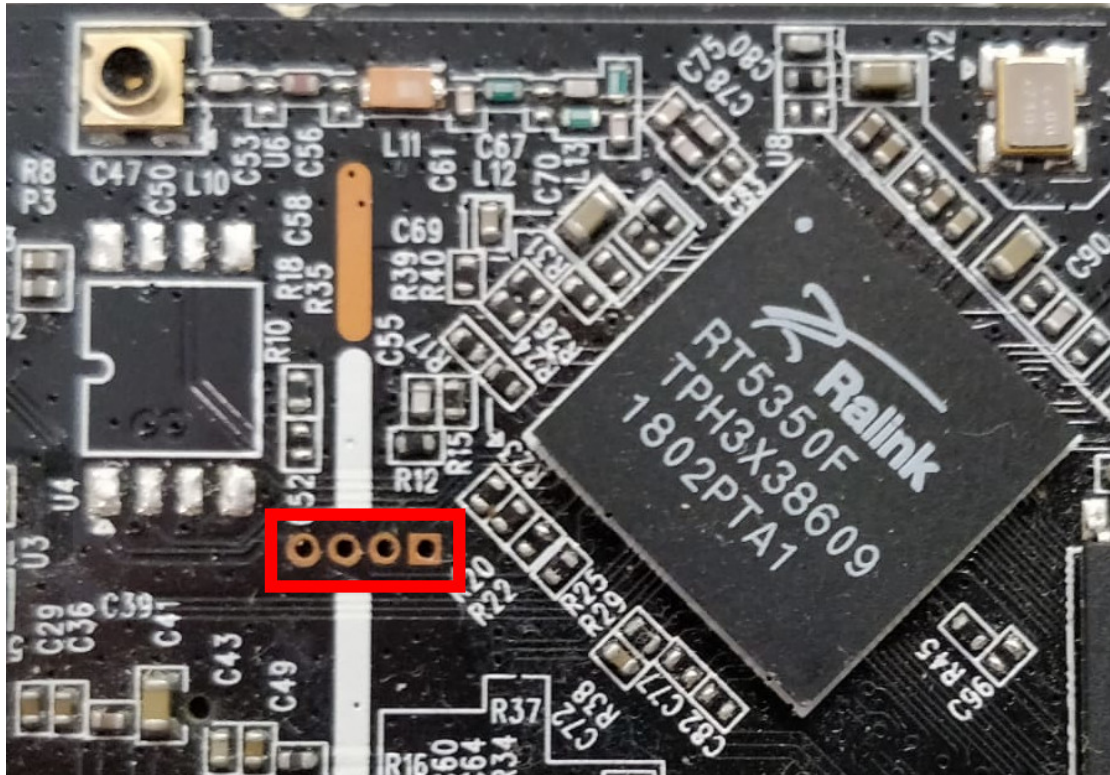


Figure 6.3: 4 pin UART header exposed on the 932L's PCB. (Highlighted in red)

had been modified, the original header could not be used, as the checksums were failing to match the new data.

Therefore, a new header with the correct checksums was required. Using the `mkimage` utility, it was possible to recreate the header, copying the original values (such as the architecture, entry point and compression type) where necessary:

```
1 sudo mkimage -n "Linux Kernel Image" -A MIPS -O linux \  
2 -T kernel -C lzma -a 0x80000000 -e 0x8038B000 \  
3 -d <lzma file> newfirmimage
```

The new header was then prepended to the previously generated LZMA data and uploaded to the device. After restarting the device, the `testfile` was found to be present in the `/bin` directory, indicating a successful persistent modification.

6.6.3 Yealink SIP-T38G

(Method A in Table 6.1)

The SIP-T38G is an Internet-connected VoIP desk phone, allowing users to manage multiple calls and messages. It was possible to gain control of the device by adapting a command injection exploit that affected previous versions of the phone [193].

6.6.3.1 Information Gathering

Reading the `/proc/mounts` file indicated that the device used `yaffs2` filesystems mounted in multiple locations, including the root (`/`), `/boot`, `/phone`, `/data`, `/config` and `/etc` directories. After exploring these directories, it was found that most files could be modified directly from the shell.

6.6.3.2 Gaining Persistence

As `yaffs2` filesystems are typically writeable with MTD user module support, it was possible to write directly to the filesystem via the shell. Additionally, the `/etc` directory held scripts that were run at boot-time, which could be modified to run custom shell commands or applications when the system next booted. After rebooting the device, modifications made to the `/etc` directory remained, demonstrating persistence.

6.6.4 WiPG-1000

(Method A in Table 6.1)

Using a command injection vulnerability present in the web interface [197; 25], a `telnet` daemon was spawned, which could be used to remotely interact with the device. After connecting via telnet, it was possible to gain further information about the device and determine the best method to gain persistence.

6.6.4.1 Information Gathering

Reading the `/proc/mounts` file indicated that the presenter used two types of storage; a flash chip and an Embedded Multi Media Card (eMMC). The eMMC

used an `ext2` filesystem which, as they are typically writeable, was targeted first. However, the filesystem was mounted to the root directory as read-only.

6.6.4.2 Gaining Persistence

While the filesystem was mounted as read-only, it was relatively simple to remount as write enabled by using the natively installed `mount` utility. This was achieved by using the command:

```
1 mount -o remount,rw,noatime /dev/mmc/blk1p1 /
```

After remounting the root directory, it was easy to directly modify the filesystem via shell commands. After restarting the device, it was confirmed that modifications made in this manner persisted through reboots.

6.6.4.3 Results Summary

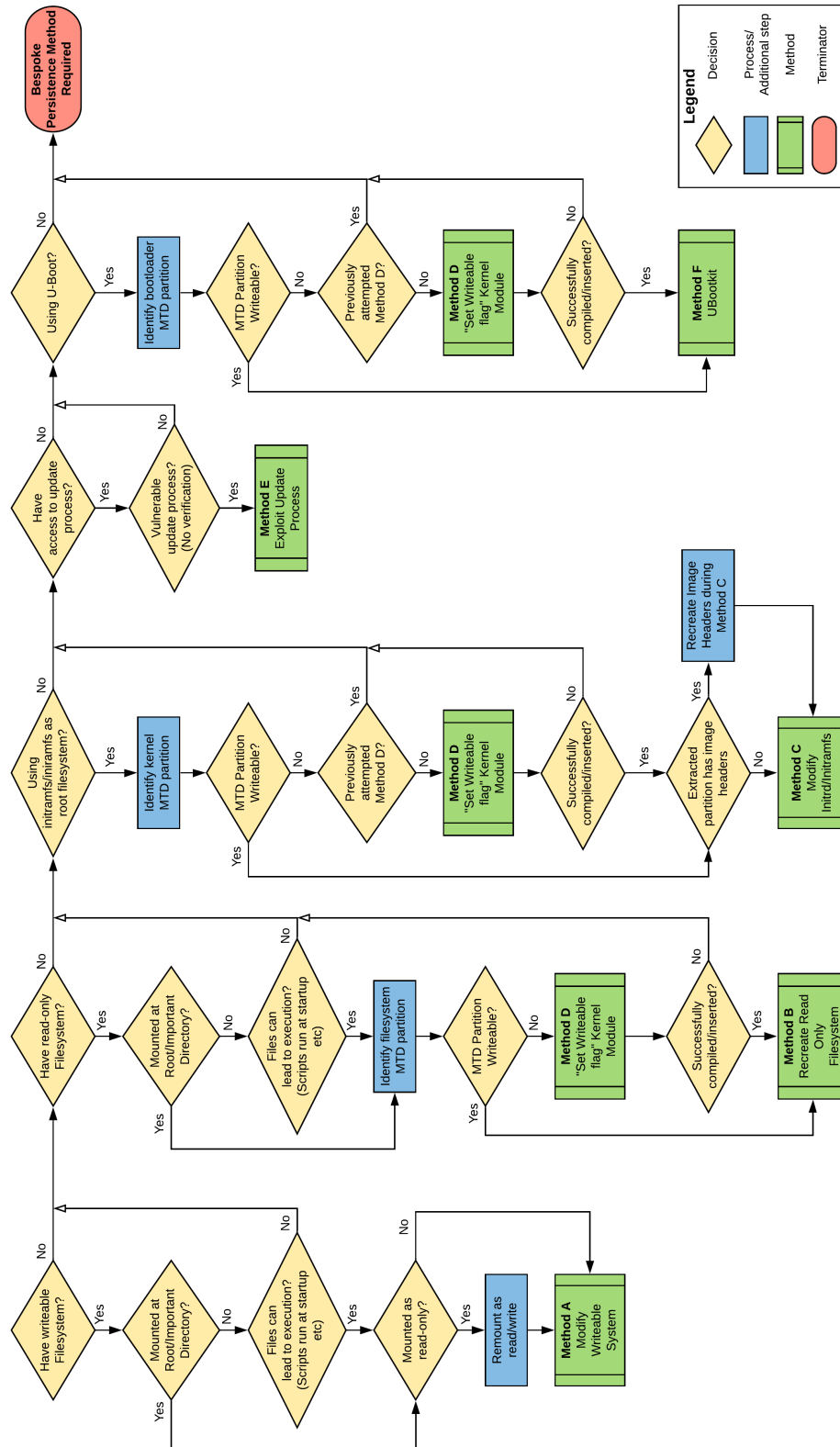
There were significant variations in the structure of the devices used for testing these methods, with different types of storage implementations requiring a variety of methods to be applied. However, it was found that it was possible to gain persistence on every device that was tested by applying the described techniques.

As part of this research, a process to determine the best method for gaining persistence on an arbitrary device was created, such that methods that are less complex to be implemented are prioritised. A graphical representation of this process can be seen in Figure 6.4.

While this work does examine a good number of persistence methods that could be used by attackers, it is by no means exhaustive. As new methods are discovered or implemented, they can easily be inserted into this selection process based on their applicability and ease of use.

During this work, the identification and application of these methods were performed manually. While persistence was achieved on the targeted devices, certain aspects of the process, such as device reconnaissance, were quite time-consuming.

For large scale attacks, automation may need to be performed for the more time-consuming tasks. For example, automatic persistence-method identification could be performed to identify the best persistence methods for new device models. However, false positives must be avoided, as if the incorrect technique is chosen,



or the application of the technique is flawed, it could lead to the device being bricked. Another possible approach would be to remotely fingerprint vulnerable devices and launch the appropriate method for “known” models.

6.7 Conclusions

In this Chapter, the increasing threat of persistence in IoT malware was examined, and the challenges that currently prevent IoT persistence from being easily achieved were outlined. Techniques that attackers could use to gain persistence on IoT devices were investigated, and their requirements, methodology and the potential issues that might be encountered were defined. The techniques were then tested against a wide range of different IoT devices in an effort to achieve true persistence.

While persistence was obtained on all the targeted devices, the variation of devices’ structure and implementation led to a time-consuming process that involved significant manual work. While it was straightforward to gain persistence on some of the devices that were tested, others required more sophisticated methods that were time-consuming to discover and implement. For large scale operations, the discovery and implementation of these more involved methods may need to be automated.

The techniques described in this Chapter present new opportunities that could be exploited by attackers, which could lead to innovations in IoT-based malware. In the following Chapter, a new type of ransomware is explored, which would be much less effective without the ability to utilise persistence.

Chapter 7

Privacy-Invasion Based IoT Ransomware

Based on content of previous publication:

“Industrialising Blackmail: Privacy Invasion Based IoT Ransomware” [43]

7.1 Introduction

Over the last few years, there has been an increase in the popularity of IoT devices and IoT-based attacks. As previously discussed in Chapter 5, early IoT ransomware strains can “lock” infected devices, preventing them from working correctly unless a payment is made. While this method of ransom may be effective, there are several limitations (discussed later in this work) that may dissuade ransomware developers from using it.

Given these limitations, attackers may be more likely to explore other methods of monetising IoT-based ransomware in the future. One such method includes extracting private data via an infected IoT device, which can then be used to extort the victim under threat of public release.

In this chapter, the viability of ransomware attacks leveraging privacy invasion techniques on IoT devices is assessed. First, previous IoT privacy research and privacy-based ransomware attacks are examined. Then, data sources commonly found on IoT devices are identified, and methods that could be used by attackers to access them are discussed. Methods to identify and interpret private information extracted from IoT devices are then explored, as well as potential mechanisms to

collate and manage any collected data. These methods are then tested on various IoT devices with differing sensors and uses to test their viability. Finally, the results and implications of privacy-based IoT ransomware are discussed, followed by potential future developments.

7.2 Background

Various types of ransom techniques have already been discussed in the previous chapters, such as crypto-based and locker-based ransomware. However, as ransomware continues to evolve, new methods are being used to ransom victims more effectively. One of the latest trends is for ransomware operators to steal sensitive data and threaten the owners with its release unless a ransom demand is paid. This method is particularly effective if the stolen data is confidential or embarrassing in nature, as it could be severely damaging if made public.

Multiple companies have already been impacted by this method. In February 2021, CD Projekt Red, a games development company, was subjected to a ransomware attack. As part of the ransom note, the attackers claimed to have stolen source code, employee details and accounting information, which they threatened to release if a payment was not made within 48 hours [52]. After CD Projekt Red refused to pay the ransom, the source code was put up for auction [201]. The attackers claimed that the data was privately sold for an unspecified amount, but some analysts believe that the claims may have been made to “save face”, and that no satisfactory offers were made [201]. It was later revealed that portions of the data were potentially being leaked online [105].

In December 2020, the Scottish Environmental Protection Agency (SEPA) was also subjected to a ransomware attack, with the attackers stealing approximately 1.2GB of files. After refusing to pay the ransom, the attackers publicly released over 4,000 documents on the dark web, including emails and databases used for contracts and commercial services [249; 206].

7.2.1 IoT Ransom Methods

Initial attempts to produce IoT-based ransomware have implemented various “locking” methods to ransom victims, such as the PaperW8 proof of concept in

Chapter 5. While the locking techniques that were previously discussed may work in certain circumstances, consumer IoT devices impose two obvious limitations for successful crypto-based and locker-based ransomware:

- **Replaceability.** Most IoT devices are designed to be relatively “cheap” when compared to traditional desktop targets – as such, victims of locker-based ransomware may instead opt to simply replace the infected device rather than pay a ransom.
- **Lack of Valuable Files.** IoT devices rarely contain files that are essential to the victim, which will reduce the impact of traditional crypto-ransomware.

However, as IoT devices are often designed to have access to data associated with their user’s environment, they thereby may provide a unique opportunity for attackers to obtain that data for malicious use. In the following sections, methods that could be used by attackers to invade victims’ privacy via infected IoT devices will be explored.

7.2.2 Privacy Invasion

IoT devices often have direct access to sensors within users’ homes, which has led to a significant amount of research into the privacy of data that they manage or create [235; 142; 237]. This is especially important as IoT devices are, by design, required to be connected to the Internet [163]. Therefore, if a device is found to be exploitable, this information may be exposed to remote attackers.

Previous research has highlighted the potential of using IoT devices to invade the privacy of users [163]. There have also been investigations as to how attacks on IoT devices may impact victims’ privacy, including case studies that demonstrate the possible methods attackers could use to track user activity [13]. As previously mentioned in Section 4.3.2.4, various attacks have been performed “in the wild”; for instance, there have been numerous instances of attackers accessing network cameras exposed to the Internet, allowing them to view video feeds inside homes and, in some cases, sell discovered “adult content” to others [258]. In one instance, an attacker used a camera’s speaker to threaten victims and demand a ransom of 50 bitcoin [1].

It is therefore straightforward to see that the natural progression of ransomware attack strategy would be to threaten to leak data belonging to victims to encourage payment. It may be possible for attackers to exploit IoT devices' access to sensors – e.g., by monitoring or turning on a microphone or camera without the owner's knowledge – in order to capture personal or potentially embarrassing data. In the next section, possible sources of private information that could be extracted by an attacker are examined.

7.3 Data Sources

Many IoT devices – such as wearables, smart toys, and medical devices – process or generate private data that their legitimate users may not want to be publicly exposed. Below, an exploration is presented as to how various data sources commonly found on such IoT devices could be used by malicious attackers.

7.3.1 In-Built Sensors

IoT devices typically use sensors to measure aspects of their environment in order to function. Some of the most commonly used are:

- **Cameras.** Cameras are often used in Internet-connected security systems for remote surveillance or monitoring.
- **Microphones.** Microphones are sometimes used to communicate with other users or as a method of control for the IoT device. For example, virtual assistants support the use of voice commands via microphones, while VoIP (Voice over Internet Protocol) phones use them to facilitate communication.
- **Geolocation Sensors.** Some IoT devices utilise Global Positioning System (GPS) sensors, which can be used to determine the current location of their users. A fitness tracker, for example, can use collected location data to map its user's walking routes or calculate their running pace.

7.3.2 Network Data

IoT devices often communicate with other devices and their users via the Internet. However, if the device has permission to send, receive or otherwise view any

sensitive data, attackers who successfully exploit the device will gain the same privileges. This can lead to security and privacy issues, such as passive monitoring. If the infected device (such as a router) acts as a gateway to the Internet for other devices, the attacker may be able to “sniff” the packets sent through it, which may contain sensitive information. Similar features have been observed in previous IoT malware, such as VPNFilter, as described in Section 6.2.4.2.

The attacker may also be able to perform internal network scanning of the device’s local network, which could lead to the discovery of additional sources of personal information such as network accessible file storage, or other vulnerable IoT devices.

7.3.3 Local Storage and Configuration Settings

While IoT devices are less likely to contain significant amounts of user-created data, they may still store personal information that is of value. An IoT device may request information from its users during the device’s set-up stage – such as their name, date of birth, or email address – which is often stored within the device’s configuration settings.

If the location of this information is known to the attacker, it could be extracted and used to facilitate communication with, or intimidate, the victim. If the location is *not* known, the attacker could also scan, using regular expressions, the memory of local processes or storage for data with a recognisable structure, such as email addresses or dates.

7.4 Identifying Private Data

For privacy-based ransomware attacks to be successful, the attacker must first be able to extract data from the IoT device. But more importantly, they must be able to identify which data is of high value, such that it can be used to extort their victims.

For large ransomware campaigns, it would be infeasible to manually search through large volumes of data collected from IoT devices to pick out relevant information that could be used as part of a ransom attack. Instead, it would be necessary for attackers to develop methods to categorise and sift through the

valuable data automatically and efficiently. Automated methods that could potentially be implemented by attackers are examined below.

7.4.1 Malicious Use of Machine Learning

IoT devices typically have access to various types of structured data, such as configuration settings, which would be relatively easy for attackers to access and interpret. However, raw data collected from IoT devices' sensors needs to be processed before its "value" can be determined. One approach is to use machine learning tools to automatically classify input data, drastically lowering the amount of manual intervention required by the attacker. This method could be used to exploit two data sources commonly found on IoT devices: cameras and microphones.

7.4.1.1 Identifying Private Images with Image Recognition

Cameras are often considered as a vector that could be used to invade a user's privacy, as if an attacker is able to gain access, they would also be able to extract images from the device without the victim's knowledge or consent. However, the attacker must be able to identify which images are likely to be "valuable".

The process for selecting potentially ransom-able images could be performed manually by the attacker, but it would be a very time-consuming process that would not scale well. Therefore, automating this process would be desirable for the attacker.

Various models that may be used to assist attackers in identifying ransom-able images, such as:

- **Theme/Object Recognition.** If certain themes or objects are detected – such as cars, buildings or crowds – it could indicate that the infected device is stationed outside, and is likely to produce images of "low value". In this case, these devices can either be discarded or re-tasked to run other malware developed by the attacker.

If objects likely to be inside (such as furniture or televisions) or people are detected, the potential value of the images extracted from the device will rise.

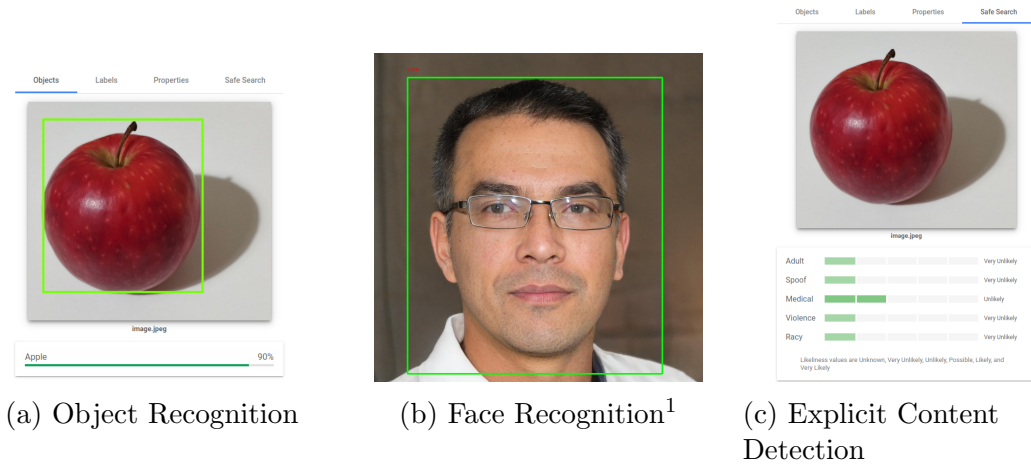


Figure 7.1: Various Google-based cloud vision system examples

- **Face Detection.** Face detection could be used to confirm the presence of victims within obtained images. If a victim is confirmed to be within the image, it could be very valuable when used in a ransom note as proof of exploitation, especially if the victim was caught in a compromising position.
- **Explicit Content Detection.** Some online services offer explicit content detection for uploaded images or videos [107]. A typical use case of this tool is to prevent the upload or transmission of explicit content on “safe-for-work” platforms or services.

An attacker could use this maliciously by scanning images taken without the victim’s knowledge for explicit content, which if detected, could then be used to ransom the victim.

Examples of detection systems using Google-based cloud services [112], demos [111], and code [106] are shown in Figure 7.1.

If the models used by the attacker determine an image to be of “high value”, it can be saved for later use in a ransom attack.

7.4.1.2 Identifying and Transcribing Private Conversations

Microphones are another type of sensor that could be used as a potential vector for privacy invasion. The possibility of eavesdropping via vulnerable IoT devices has

¹Image Source: This-Person-Does-not-Exist.com

been explored in previous research [288; 88], but not in the context of ransomware.

For this method, the attacker aims to intercept private conversations held by the victim, and then use a speech-to-text engine to transcribe the conversation into an easily managed text format. Once the audio has been transcribed, the attacker can then use automated methods to search for valuable keywords, such as those related to personal accounts, or potentially exploitable activity which could be used to ransom the victim.

7.4.2 Network-Based Privacy Invasion

While IoT devices often have access to numerous sensors in order to interact with their surroundings, network activity can also be used to gather information about the user. Multiple techniques can be used by attackers to extract private information from local networks made available via compromised IoT devices. Below, some of these techniques are examined.

7.4.2.1 Intercepting Browsed Domains

If an attacker can intercept a victim’s Internet traffic via an infected device (such as a router), they may be able to extract sensitive information about the victim’s browsing habits.

In this case, the attacker may intercept traffic passing through the device and extract domain names of any websites that the victim visits from various protocols, such as DNS [181], HTTP [89], or HTTPS [53].

The methods used for each protocol are shown below:

- **DNS.** As DNS requests are not typically encrypted, if they are intercepted, it is possible to extract target domains requested by victims [181].

However, if a response to a previous request has been cached locally, a new request might not be made, which would prevent the attacker from detecting re-visited domains until the local cache is cleared, or the time limit on the data expires (expressed as “Time to Live” or “TTL” [181]).

- **HTTP.** When making an HTTP request, clients are required to include a `Host` header, as this allows servers that host multiple websites at the same

IP address to determine which website the client is attempting to connect to [89].

As this header is transferred in plaintext, an attacker would be able to intercept these requests and determine the top-level domain that any connected victims are attempting to visit.

- **HTTPS.** The HTTPS protocol encrypts communication with websites that users visit, including the headers used as part of the request, such as the `Host` header. This prevents attackers from being able to access the websites' content or determining the visited domain using the previous method.

However, servers may still require the ability to host multiple websites at the same IP address. Additionally, as part of the Transport Layer Security (TLS) handshake, each website must provide a certificate to the client. As the certificates for each host will most likely differ from one another, the server has to know which host is being requested *before* the encryption stage, such that the correct certificate is supplied. Otherwise, the connection may fail to be established.

To solve this issue, “Server Name Identification” (SNI) can be used. This requires the client to include the `server_name` TLS extension as part of the initial `CLIENT_HELLO` message, which is used to specify the domain name the user is attempting to access. The server reads this value and then provides the correct certificate pertaining to the request.

As the `server_name` value is sent as part of the initial request before completing the handshake, it is sent in plain text with no encryption, which would allow an attacker to extract the domain before secure communication can be established [53].

Once a visited domain has been extracted, it can then be compared against a list of domains known to be associated with illegal or potentially compromising activities. If a match is found, details can then be logged to a C&C server to extort the victim at a later date.

7.4.2.2 Intercepting Web Content

In addition to being able to extract domains from a victim's browsing activity, it may also be possible to intercept the content of visited web pages. The content of websites with known structures could be read to extract important information, such as video titles, usernames, or personal information.

For HTTP traffic, this is relatively simple, as communication is typically performed in plaintext, allowing attackers to access any transferred content.

Thankfully, web traffic is increasingly using HTTPS, which encrypts the communication between the client and server when transmitting web content [161; 115]. However, it could still be possible to gain access to encrypted content using Man-In-The-Middle (MitM) attacks, such as **SSLStrip**, which would allow attackers to intercept and modify victims' web requests to bypass HTTPS encryption [174].

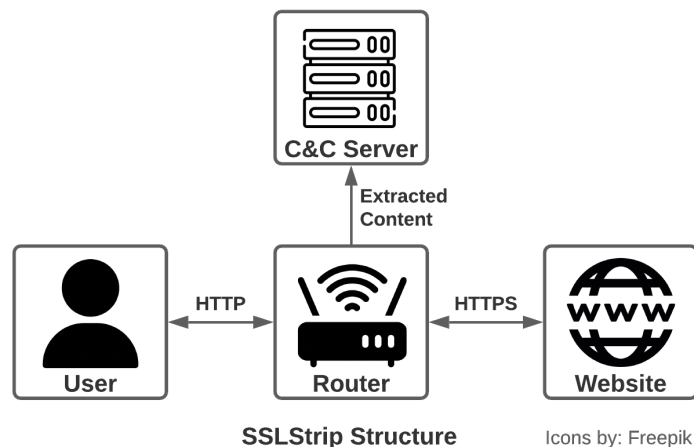


Figure 7.2: Basic SSLStrip attack structure

To perform this attack, the infected device is forced to act as a proxy, examining all the traffic passing between the client and the server. Typically, if a user visits a website via HTTP when HTTPS is available, the server will automatically redirect them to use HTTPS. When SSLStrip is active, these redirects can be prevented, and instead, the device will create its own encrypted connection with the requested server, then act as an intermediary between the client and the server. The device will encrypt and send any requests from the client to the server

while returning any server responses in plaintext back to the client, as shown in Figure 7.2. This allows the attacker to catch inattentive victims unaware and extract plaintext communication that would typically be encrypted.

A similar style of attack has been previously implemented by the IoT malware `VPNFilter` to extract usernames, passwords and logins from victims' traffic [241].

7.4.2.3 Identifying Device Locations via WiFi Positioning

The location of the infected device could be used to determine the address of the victim, which can later be included within a ransom note. However, in order to ascertain the location of the infected device, the attacker must make use of the available data sources.

Some devices need to be aware of their current location to function correctly, such as fitness trackers that track a user's sporting activities and routes. Ideally, this type of information would be acquired using a GPS sensor, which in 2015 was found to be accurate up to 4.9 meters on average [264]. However, most IoT devices are unlikely to implement GPS sensors, especially if they are not designed to be moved often. Instead, a different data source must be used.

Online WiFi Positioning Systems (WPS) allow users to triangulate their current position by comparing a scan of local WiFi signals against a list of known signal locations stored in an online database. The accuracy of this measurement is dependent on various factors, such as the number of detected signals, or matches found within the service provider's database. The predicted accuracy of the estimated location is sometimes provided as part of the result [109], depending on the service provider.

If an infected device has wireless capabilities, attackers may be able to perform a scan to discover the Service Set Identifiers (SSIDs), Media Access Control (MAC) addresses and signal strengths of nearby routers, which can then be sent to the C&C server. The attacker could then upload these details to an online service, such as Mozilla Location Services [185] or the Google cloud platform [109] to obtain an estimate of the device's location.

While it is unlikely that location information alone could be used to ransom a victim, it could be used as an addition to other associated sensitive or personal data as a form of intimidation.

7.4.2.4 Internal Network Structure

Infected devices could provide attackers with access to other devices on the local network, which would be otherwise inaccessible from the Internet. The attackers would then be able to scan or attack them, potentially gaining access to further private data.

7.4.3 Data Processing

Once data has been successfully extracted from a device, it must then be processed to identify any potentially ransomable information. For network data, which is typically well structured, this is a computationally inexpensive process and would require minimal network usage to upload the results to a C&C server.

Less structured data, such as that collected from device sensors, can be much more difficult to interpret. While the use of machine learning can significantly reduce the amount of manual effort required to identify ransomable data, there are some logistical issues that attackers may need to overcome before it can be considered viable.

Many IoT devices are unlikely to have the hardware capable of running the required machine learning models. Additionally, IoT devices' internal memory is often limited to only what is required to run the system during normal operation, which may prevent collected data from being locally stored.

To circumvent these issues, attackers may instead process, classify, and store data collected by infected devices on remote systems. For example, attackers could choose to process collected data on their own recognition server using publicly available models. However, this may not scale well, as a large ransomware campaign may cause immense network strain on the attacker's infrastructure, which could become quite costly to maintain. Therefore, it may become necessary to outsource processing to a third party, such as cloud services.

Cloud services may provide attackers with methods to cheaply analyse large amounts of private information, without having to consider network strain.

7.4.3.1 Cost Saving

While cloud services are likely to be the most cost-effective method of processing data for a large ransomware campaign, attackers may still attempt to limit their

usage to reduce operating costs. Below are some techniques that could be used.

- **Rate-Limiting.** Rate limiting the collection and processing of data from IoT devices' sensors will reduce the overall network traffic required to transport it to the remote service. For example, instead of streaming video feeds from each device to a remote service, taking snapshots at regular intervals will reduce the overall amount of information that needs to be transferred and processed.

- **Pre-Processing.** While many devices will not be able to run full machine learning models, by implementing a simplified “pre-processing” stage on the device, the attacker can prevent “useless” or “repeat” data from being sent for processing.

For example, running a full transcription model might not be possible on the average IoT device, but using a model to detect when specific “hotwords” have been used [287] could highlight when useful information has been collected. This could then be transferred to a remote service to perform a more thorough analysis.

- **Triggers.** Instead of automatically sending data at a certain interval, a “trigger” could be used to start recording when potentially valuable information is likely to be obtained.

For example, the infected device might only record data when a certain audio level is reached, or when motion is detected on a camera feed.

- **Blacklisting.** If a device is found to regularly provide “worthless” data, it could be blacklisted and instead used to facilitate other types of malware, such as cryptojacking.

- **Cloud Service Fraud.** Typically, attackers would need to purchase credits on their chosen cloud service, in the hope that payments made by victims of the ransomware will offset the cost of using cloud processing.

However, some services provide “free tiers”, which offer small amounts of free credit to new users, allowing them to test certain features for free [113; 126]. Attackers may attempt to create or hijack large volumes of these “free tier” accounts, such that they can process collected data without payment.

7.5 Data Management

The privacy invasion methods discussed above present possible avenues for ransomware authors to extract private information from IoT devices. However, using the extracted information to perform a ransomware attack in a large scale campaign presents multiple challenges, such as how to generate an effective ransom note, and how the information could be published should the ransom not be paid.

In this section, how these challenges may be approached by future attackers will be examined.

7.5.1 Ransom Note Generation

Once adequate personal information has been collected from a device, a ransom note demanding payment can be generated and displayed to the victim.

The attacker can then attempt to display the ransom note by hijacking communication methods native to the device, such as the examples explored in Chapter 4. Additionally, if any contact information has been extracted from the device, such as an email address, the ransom note could be sent directly to the victim.

Typically for ransomware attacks, the ransom note would likely contain a description as to what has occurred, a timer, and instructions for paying the ransom at a minimum. However, unlike ransomware that prevents victims from accessing their resources, privacy invasion ransomware threatens to *release* private information unless a ransom is paid before a certain time.

Therefore, including select private information that has been obtained throughout the collection stage in the ransom note may provide sufficient evidence to force the victim into making a payment. By “personalising” ransom notes in this manner, it may lead less technically-aware victims to conclude that the attack was a manual effort made to target them specifically, which may further encourage payment.

Additionally, it is unlikely that the average user would be able to determine when their device was first infected. An attacker could take advantage of this by implying the existence of further collected information that they would not display to the victim. The uncertainty as to what the attacker may have collected could apply additional pressure to victims.

7.5.2 Publishing Private Information

As part of a privacy-based ransomware attack, the victim is threatened with the release of their private information unless a payment is made. Private information could be publicised in a number of ways, varying in complexity. Some of these methods are explored below.

7.5.2.1 Centralised Publication

One method attackers could use to publicise information is to create a centralised “leaking platform” available via a publicly accessible website. Any victims that do not make a payment would have their information published on the website for anyone to view.

As part of the ransom note, victims would be encouraged to visit the website for further information or to facilitate payment, acting as a form of advertisement. Previous victims’ private information would be visible to the “new users”, which would serve as proof that the attacker will follow through with threats to publicise.

If the leaking platform becomes well known, it may attract the attention of various types of users:

- **Curious Users.** Users that have not been infected by any form of ransomware, but are curious viewers of any private information that has been published.
- **Motivated Users.** Users that are looking for private information about a specific victim or group.
- **Malicious Actors.** Independent attackers that use the provided private information for separate attacks. The presence of these attackers may also further encourage payment from victims who wish to avoid being targeted.

7.5.2.2 “Direct” Publication

In addition to making the information publicly available, attackers could use information previously gathered about the victim to determine who would be most impacted by its release, such as friends, family, or co-workers.

If the victim’s address is known, one possible approach could be to threaten to send the blackmail material to neighbours in the area, however, this would

add significant cost and complexity to an attack. Therefore, it is more likely that attackers may instead rely on using online resources. For example, if the attacker identifies the victim's social media accounts (such as Facebook, Instagram or Twitter) during the information gathering stage, they may be able to enumerate people that the victim associates with. Attackers could then attempt to use the observed social media platforms to distribute the victim's private information to those that were identified, such as through the use of automated chatbots. If this technique is used alongside the previous method, these messages could also serve to further advertise the leaking platform's website.

While this approach could drastically increase the impact of publicising information, it may also increase the complexity of the ransomware, as the attacker would need to automate account identification, enumeration and distribution for any supported social media platforms.

7.5.3 Scale of operations

Previously, such malware would require significant manual oversight. The outlined automation steps, such as the use of machine learning and automatic generation of ransom notes, would allow attacks to be performed without needing costly manual labour.

7.6 Proofs of Concept

To ascertain the viability of privacy-based ransomware on IoT devices, several experiments were undertaken to extract private information from a range of device types, and collate it such that it could theoretically be used to ransom a victim².

Below, the process used to extract and collate the data collected for each device is explained.

7.6.1 Data Collation

As previously shown in Section 7.4, there are various methods attackers may use to extract private data from victims' IoT devices. However, the collected data

²For an attack to succeed, it is assumed that the attacker is able to remotely access and exploit the vulnerable service via the Internet.

must be correctly managed for threats of publication to be effective.

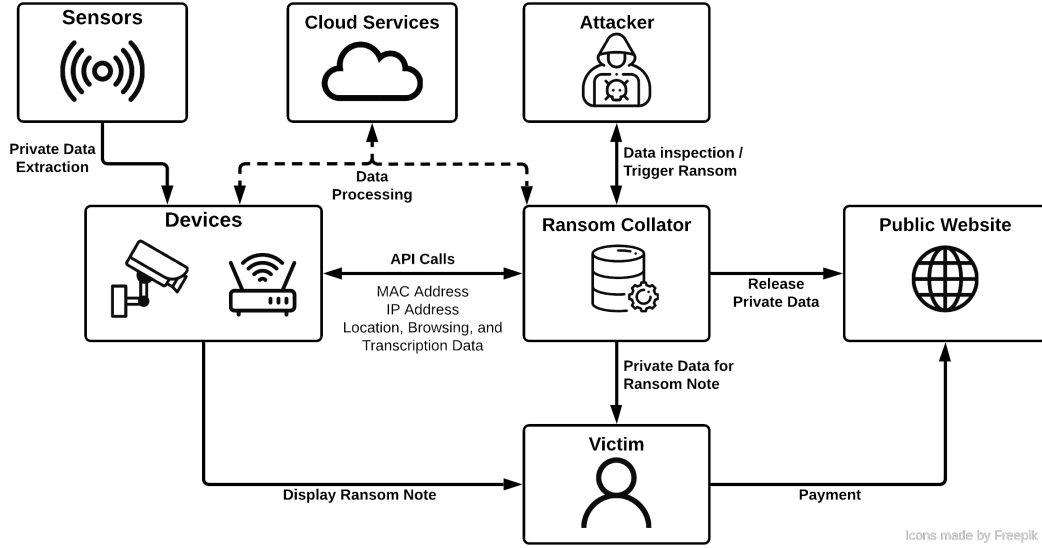


Figure 7.3: Data collator structure graph

As part of this research, a basic proof of concept collator was created that would allow an attacker to manage data collected from various compromised devices. An abstract view of the collator’s operating structure is shown in Figure 7.3.

The server running the collator exposes an API for infected devices to interact with, allowing various types of private data to be uploaded, such as images, audio recordings, or browsing history. A list of the different endpoints is shown in Table 7.1.

Table 7.1: API endpoints hosted by the IoT Collator

Endpoints	Purpose	Arguments
addDevice	Add a new device to the database.	MAC address, Device Model
addWebsite	Add a blacklisted website visited by the victim.	MAC address, Domain
addSpeech	Receive intercepted audio, queue for transcription. (Primarily used for testing)	MAC address, Base64 Audio
addImage	Receive base64 encoded image, decode and store.	MAC address, Base64 Image

Once data is received by the collator, it can be processed using the appropriate method, such as those described in Section 7.4.3. Visited domains can be directly added to the database, while more complex data, such as images or audio, must first be processed and formatted before being stored.

Each data point is then associated with the infected device’s MAC address, as it is an easily available unique identifier that is unlikely to change, even through reboots.

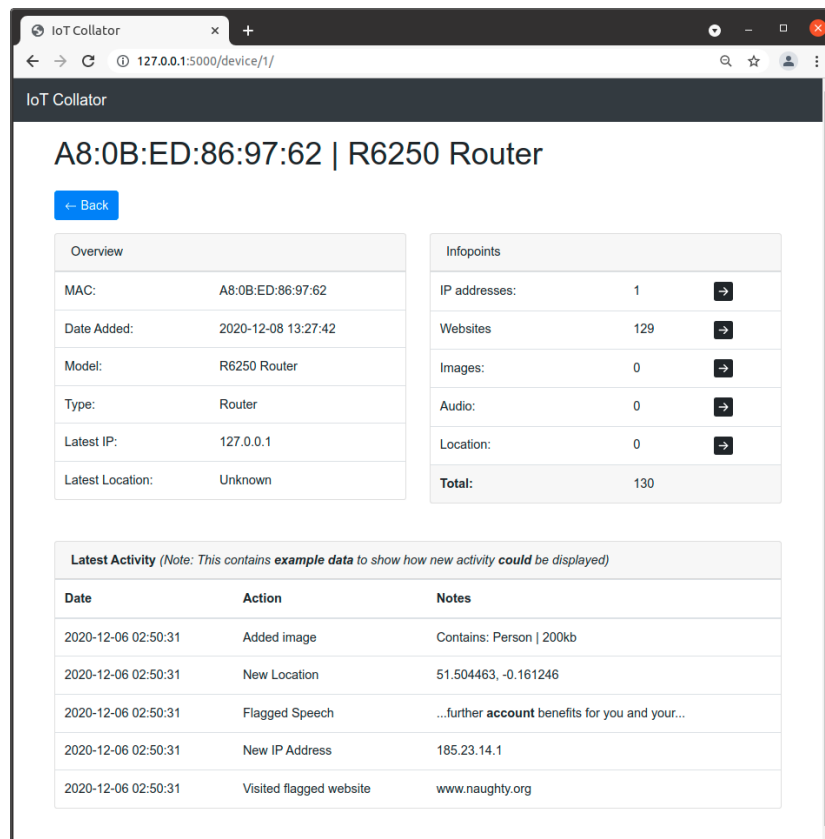


Figure 7.4: IoT Collator summarising information collected from an R6250 router

The attacker can then access the data processed by the collator via a web interface, as shown in Figure 7.4. Additional features, such as highlighting particularly interesting collected information, scoring the “ransomability” of devices, or automatically generating ransom notes could also be implemented by the attacker to improve efficiency.

7.6.2 Netgear R6250 Router

As routers often act as the main gateway for Internet traffic in a network, it was determined that they would be ideal for testing the domain name and network data extraction techniques discussed in Section 7.4.2. The Netgear R6250 router was chosen for testing, as previous work (covered within the earlier Chapters) demonstrated that it could be exploited using a command injection vulnerability [158; 194].

7.6.2.1 Domain Extraction

To test the possibility of extracting information from network activity, a program was created to perform local packet sniffing using the `libpcap` library [245], which was cross-compiled to be compatible with the target router’s architecture.

The program intercepted all packets destined for port 80 or 443 — the default ports used for HTTP and HTTPS traffic — and extracted any visited domain names. The domains were then compared against a hard-coded list, and if a match was found, an API call would be made to the collator, which would record the visited domain, a timestamp of the visit and the calling device’s MAC address.

A network consisting of the R6250 router, a smartphone and a desktop computer was created for testing the effectiveness of this technique. After exploiting the router, the application was uploaded and run on the device.

The connected smartphone and computer then browsed various websites via the router. The application successfully identified and reported any “trigger” domains visited using either HTTP or HTTPS to the collator, which the “attacker” was then able to view.

For this proof of concept, methods to save or interpret intercepted web **content** were not implemented, but could theoretically be implemented by a dedicated attacker in the future.

7.6.2.2 WiFi-Positioning

While the router did exhibit wireless capabilities, it did not seem to be possible to scan for other nearby SSIDs or MAC addresses. This may be due to the limitations imposed by the expected usage of the device.

However, it was possible to view the router’s MAC address and SSID, which could then be used to query a WiFi-Positioning service. While only one “signal” would be available for reference, which may reduce the accuracy of the result, it should still allow an attacker to make an approximate guess as to the victim’s location, as WiFi signals have a limited range within which they can be detected.

Unfortunately, it was not possible to fully test the WiFi-positioning method in this instance, as the router was only powered during the analysis and experimentation stages. It would therefore be very unlikely for the MAC address to be detected or stored by any WiFi-positioning services. As such, attempts to use cloud services would, at best, result in a wildly inaccurate location being provided.

In a realistic scenario, this would likely function as expected, as in a typical use case, the device would be more likely to run for long periods of time in the same location, increasing the chances of it being accurately detected and recorded by location services.

7.6.2.3 Configuration Extraction

As part of the exploitation stage, a telnet service was started such that the device could be investigated further. During this investigation, attempts were made to identify the location where the user’s settings were stored. It was found that settings were being saved to the second partition on the flash chip, which was accessible via the `/dev/mtdblock1` file.

```
# head /dev/mtdblock1 | grep 'wlg_passphrase\|http_username  
\\|http_passwd\|wla_passphrase\|pppoe_username\|pppoe_passwd  
'  
wlg_passphrase=adminlol  
http_username=admin  
ipv6_pppoe_username=  
wla_passphrase_backup=  
http_passwd=adminlol  
wlg_passphrase_2=  
wla_passphrase_2=  
wla_passphrase_3=  
wla_passphrase_4=  
ipv6_pppoe_passwd=  
pppoe_passwd=examplepw123  
wlg_passphrase_backup=  
pppoe_username=examplemail@isp  
wla_passphrase=adminlol
```

Figure 7.5: Configuration data extracted from the R6250 router

By using a `grep` command it was possible to view sensitive configuration data

that was stored in plain text, as shown in Figure 7.5. While this is admittedly a very simplistic example, the relative ease with which it can be applied indicates that attackers can use this method to efficiently extract valuable information.

More complex approaches could be applied by attackers to improve the effectiveness of this technique, especially if the structure or location of the data is known beforehand. Attackers could develop applications to extract usernames, passwords, email addresses or phone numbers from long-term storage or application memory, although the output may need to be validated by the collator to remove duplicates and false positives.

7.6.2.4 Ransom Note

Using the techniques covered in Section 4.4.3, it is possible to redirect DNS requests made to a compromised router. As with previous IoT-based ransomware, the attacker would be able to redirect victims attempting to browse the Internet to a webpage containing a ransom note.

In addition to the traditional ransomware elements, such as timers and demands for payment, as shown in the previous proofs of concept in Section 4.4, the note can also include select personal information collected by the malware to act as “proof of compromise” and apply additional pressure to the victim. An example of how a ransom note could be presented is shown in Figure 7.6.

7.6.3 Yealink SIP-T38g Phone

The SIP-T38g is an Internet-connected IP phone with a built-in LCD screen. As the device is designed for direct communication, it was used to test the audio extraction techniques described in Section 7.4.1.2.

7.6.3.1 Private Conversation Extraction

To extract private conversations from the device, the first step was to obtain access to any audio input and output when a call was made. While it would theoretically be possible to record audio directly from the device’s microphone, the audio can also be extracted from the device’s network activity, which has the added benefit of providing *both* sides of the conversation.

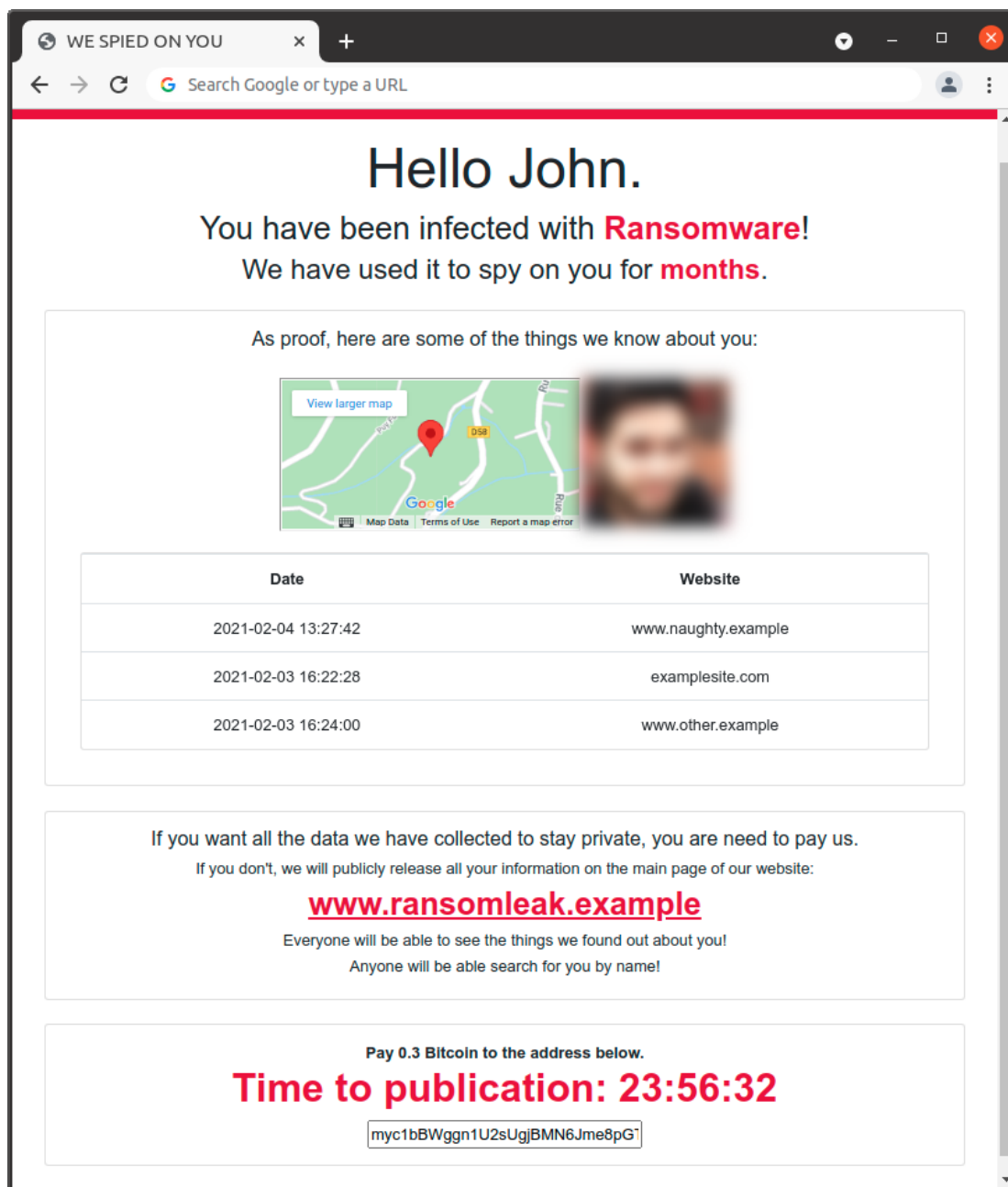


Figure 7.6: Example privacy-based ransom note with “proof of compromise”

An open-source tool named “VoIPong” [84; 29] can be used to intercept, decode and output detected VoIP calls as either `.raw` or `.wav` files. During this research, a custom version of VoIPong was created, configured and cross-compiled such that it would be able to run natively on the phone.

The device was exploited using a command injection vulnerability present in its web interface, allowing the application, which was configured to save any detected calls to a pre-defined folder, to be uploaded and run. Unfortunately, the phone had limited storage capacity, with only a collective 60 megabytes of free space available across all of the available partitions.

To overcome this limitation, a Network File System (NFS) share was hosted on the collator server. The phone could then mount and modify the contents of this filesystem as if it were a local directory. The collator would periodically check for “file close” events within the shared folder, such that when recordings were finalised, the conversations could be queued for processing.

When the audio was ready to be processed, it was passed to a speech-to-text (STT) service for transcription. Initially, attempts were made to use a local instance of Mozilla’s “deepspeech” engine with a pre-trained model and scorer [183]. However, the audio extracted from the intercepted calls was sampled at a rate of 8 Kilohertz (kHz), also known as “narrowband”. Unfortunately, the Mozilla model was designed for an expected sample rate of 16kHz, also known as “wideband”, which led to unsatisfactory performance.

While a new model could be trained to attempt interpretation of the narrow-band audio, it was considered out of scope for this research. Instead, various online services were used to attempt accurate transcription of the calls.

Text	Word Timings and Alternatives	Keywords (6/6)	JSON
<div> ✔ account: Spotted - 11.15-11.5s (100%), 40.08-40.53s (4%) ✔ address: Spotted - 40.53-41.12s (100%) ✔ password: Spotted - 5.91-6.51s (100%), 53.37-53.73s (2%), 70.42-71.08s (100%) ✔ pin: Spotted - 53.37-53.73s (2%) ✔ name: Spotted - 34.81-35.17s (100%) ✔ number: Spotted - 11.5-11.71s (100%) </div>			

Figure 7.7: IBM speech-to-text demo recognising selected keywords

The Google Cloud Services API [110] successfully transcribed conversations

with much higher accuracy. Tests were also performed with an “IBM Watson Speech-to-Text” demo [127], which included support for narrowband audio. IBM Watson was able to extract key components from the conversation, and also provided a “featured keyword” identification tool, which could be used by attackers to listen for subjects of interest, as shown in Figure 7.7.

Finally, calls were converted to a video format and uploaded to YouTube. Approximately ten minutes after the initial upload, captions were automatically added and could be scraped from the source of the video’s webpage. As YouTube provides this feature as a free service, this could potentially be used by attackers to avoid paying for the use of Google’s cloud services.

After the conversations were transcribed, the text and audio files were linked with the source device’s MAC address and inserted into the collator’s database. At this point, the attacker would be able to read the conversations, or search for potential blackmail material by identifying “valuable” words within the text, such as “account”, “password” or “address”. This entire process can be fully automated and performed without giving the victim any indication that they were being monitored until the ransom note is triggered.

7.6.3.2 Ransom Note

The VoIP phone provides multiple avenues for user interaction, such as its web server, microphone, speakers, and in-built LCD screen. As with the R6250 router, an attacker could hijack the device’s web server to display a ransom note, including “proof of compromise”, such as recordings of the victim’s conversations.

However, as the web server was unlikely to be accessed in day to day usage, hijacking the connected screen via the framebuffer technique described in Section 4.4.3 was considered to be the more effective method to communicate with victims, as shown in Figure 7.8.

Other communication methods could potentially be used in the future, such as using the speakers to play back intercepted conversations, but this would likely be unnecessary if the previous communication attempts are successful.

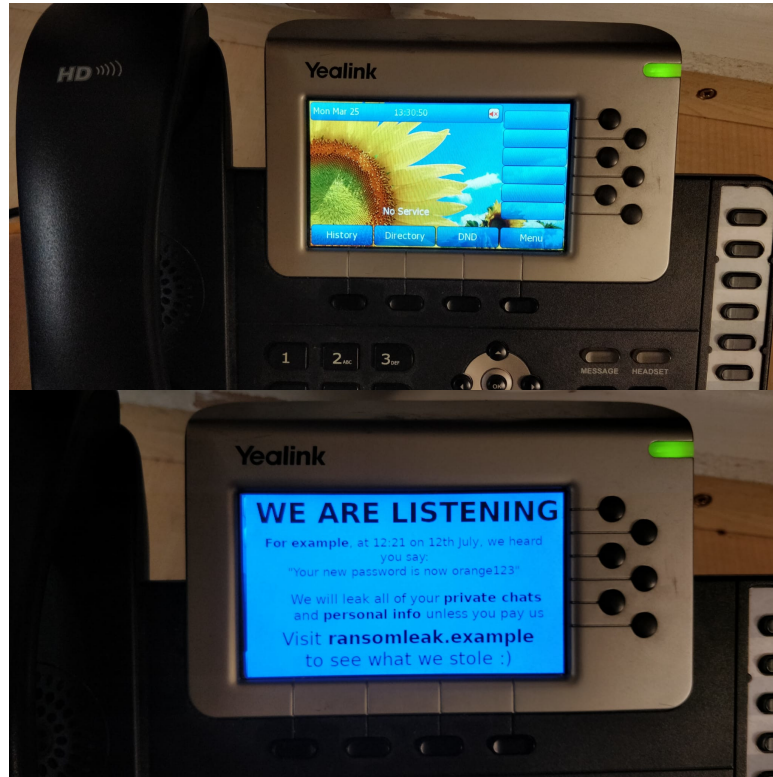


Figure 7.8: Hijacking the screen of a Yealink SIP-T38G

7.6.4 DCS-932L Camera

The DCS-932L is an Internet-connected camera designed by D-Link, and has been used in previous research (such as in Chapter 4, 5, and 6) to test various ransomware components. In our experiments, the device was used to test the WiFi-positioning and image-based privacy invasion techniques.

7.6.4.1 WiFi-Positioning

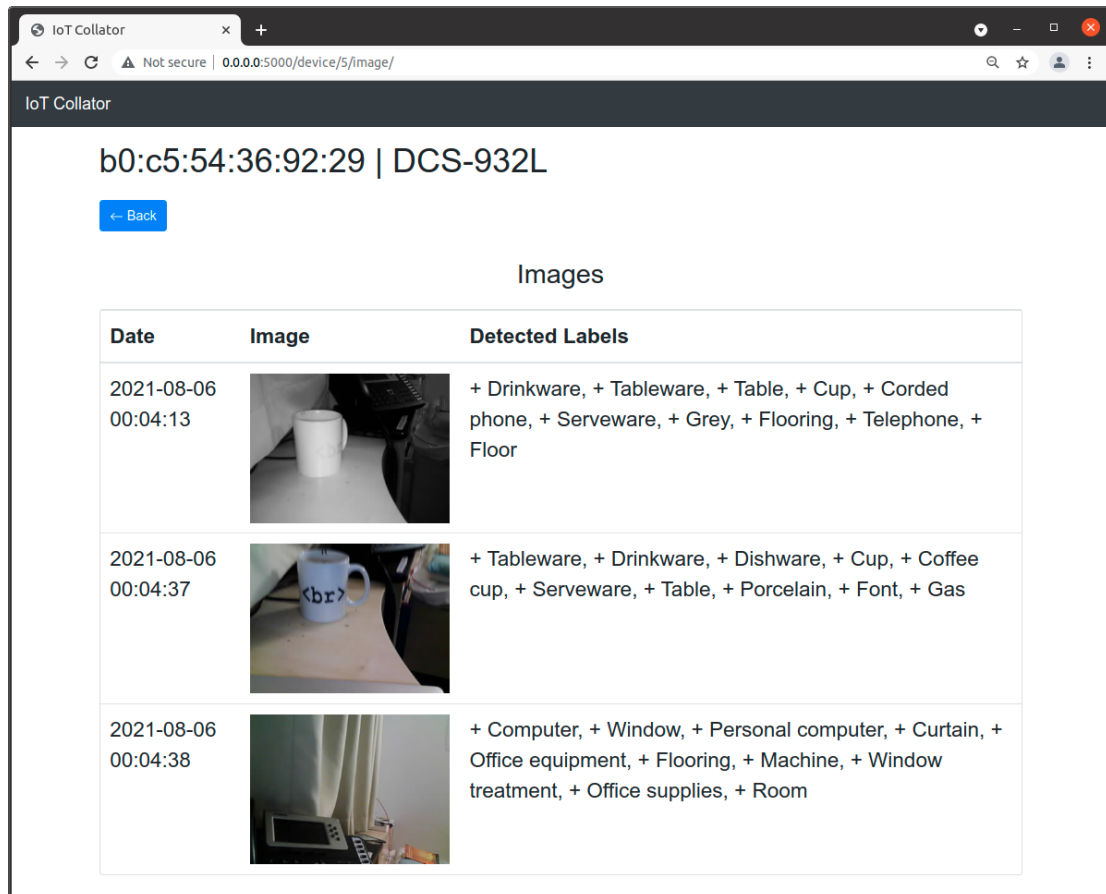
The device was exploited using the same buffer flow vulnerability previously shown in Section 4.4.4.5, allowing further device analysis to be performed. A WiFi scanning application was uploaded to the device, and it was found that when the camera used WiFi to connect to the Internet, it was possible to scan for nearby SSIDs and MAC addresses.

During the testing stage, three nearby access points were detected after performing a series of scans. After uploading the access point information to Google Cloud Services, the location of the device was accurately determined to within 15

meters.

While the accuracy of this reading may vary depending on some factors outside of the attacker's control, this does highlight the potential danger of attackers being able to identify a victim's location after compromising their devices. As highlighted in Section 7.4.2.3, some services (including Google Cloud Services) provide a confidence threshold for any returned location readings. This threshold could be used by attackers to determine whether the location should be considered when performing a ransom attack or included in any generated ransom notes.

7.6.4.2 Image Extraction





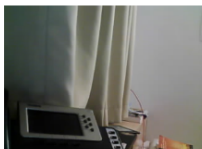
Date	Image	Detected Labels
2021-08-06 00:04:13		+ Drinkware, + Tableware, + Table, + Cup, + Corded phone, + Serveware, + Grey, + Flooring, + Telephone, + Floor
2021-08-06 00:04:37		+ Tableware, + Drinkware, + Dishware, + Cup, + Coffee cup, + Serveware, + Table, + Porcelain, + Font, + Gas
2021-08-06 00:04:38		+ Computer, + Window, + Personal computer, + Curtain, + Office equipment, + Flooring, + Machine, + Window treatment, + Office supplies, + Room

Figure 7.9: Labelling images extracted from an infected DCS-932L Camera

As the camera was designed to be used for surveillance, it was ideal for testing image-based privacy invasion techniques. During the device analysis, it was found

that during normal operation, it would provide a snapshot from the camera to the user when they visited the web server. This snapshot could be retrieved directly by forcing the device to make a local request to the web server at the URL: `/image.jpg`.

An application was created that, when run, would download the image to a temporary directory, base64 encode it, then prepare it for transfer to the collator via the API. The application was cross-compiled to be compatible with the camera and then uploaded for testing.

Once images were received by the collator, they were uploaded to Google Cloud Services [112] to label recognised objects, locations and activities. As shown in Figure 7.9, the platform was able to recognise and correctly label objects within the extracted images. If required, other services, such as face detection [108] or explicit content detection [107], could be applied in the same manner with minimal effort or changes to the code.

7.6.4.3 Ransom Note

The DCS-932L camera did not contain many methods to communicate with the user. As most interaction with the device was performed via the web service, which displayed the current view from the camera, the attacker could use the same method described in Section 7.6.2.4 to hijack the web server to display a ransom note. The extracted image could be included to further pressure the user. If the location is known, a map could also be shown, such as via a Google Maps widget.

7.6.5 Summary

In this section, practical examples of how private information could be extracted and collated from various IoT devices were demonstrated. Private information was then extracted from various data, including Internet traffic, intercepted audio, and captured images.

It was then shown how the collected data could feasibly be analysed, organised and used by an attacker to facilitate privacy-invasion based IoT ransomware.

Table 7.2 provides a summary of the six privacy invasion methods that were

Table 7.2: Privacy invasion methods used for each device

Device	Domain Extraction	Config Extraction	Transcribe Audio	Recognise Images	Identify Location
Netgear R6250	✓	✓	-	-	Partial ³
Yealink SIP-T38g	-	-	✓	-	-
D-Link DCS-932L	-	-	-	✓	✓

described in this work, namely *Domain Extraction*, *Config Extraction*, *Audio Transcription*, *Image Recognition*, and *Location Identification*.

Additionally, Table 7.2 shows how these methods fare when applied against the three IoT devices used in the above proofs of concept.

While using exploited IoT devices to invade the privacy of their users has been theorised in the past, it has rarely been explored as a practical option for the average attacker. Here, an exploration of how such privacy invasion could potentially be monetised using ransomware was performed, including the feasibility of implementing such attacks at scale.

7.7 Categorising Privacy-Based IoT Ransomware

Privacy invasion has been used by previous instances of ransomware in order to encourage payment to be made by victims (previously covered in Section 7.2), and is often referred to as “leakware”. However, the methods defined within this Chapter could allow this implementation of malware to fit into a number of other definitions.

7.7.1 Spyware

As some of the above methods explore the possibility of spying on victims, the malware that implemented them could potentially fit within any of these terms:

- Stalkerware [119; 182]
- Spyware [171; 101]

³As previously mentioned, the WiFi-positioning method was unable to be fully tested for the R6250 router, as it was only powered during analysis and testing, preventing its MAC address from being detected or stored by any WiFi-positioning services.

- Surveillanceware [104]

However, these definitions may be misleading in this context. Malware of this type often aims to either gather information for direct sale online due to its inherent value (such as card numbers or logins to online services), or perform long term spying on an individual for the purpose of “cyberstalking”. Therefore, while some of the techniques highlighted within this chapter could potentially be used in these types of malware, these terms are not appropriate for referring to this particular use case.

7.7.2 Extortion

There are a number of definitions for malware designed for the purpose of extortion, including:

- Leakware [69; 214]
- Extortionware [94; 250]
- Doxware [222; 243]

These definitions are very similar in nature, and seem to be used somewhat interchangeably when referring to privacy-invasion based ransomware. However, there could be a separation between “leakware” and “doxware”. For example, leakware could be used to refer to malware that is designed to steal data from a company or corporation, whereas doxware could instead be used to describe malware that attempts to extort *individuals* with the release of private personal data. The term “doxxing” (from which doxware inherited its name) refers to “reveal[ing] information about somebody on the Internet, usually in order to harm them”, which would not be as applicable to companies [242].

While there is no clear consensus on these definitions, if they can indeed be used interchangeably, for the malware structure described within this Chapter, it may be more accurate to use the term “doxware”, as these techniques are more likely to succeed when being used to attack an individual.

7.8 Future Privacy-Based IoT Ransomware

The methods that have been demonstrated in this work present concerning issues that could become problematic in the future. Here some of the issues that may occur should this style of ransomware become popular will be discussed.

7.8.1 Native Malicious Machine Learning

Currently, the identification and management of data presents a significant hurdle that attackers must overcome in order to create effective privacy-invasion based IoT ransomware.

The infrastructure required to transfer, store, and process any collected data may dissuade malicious actors from attempting to perform these types of attacks. However, as the hardware present in IoT devices continues to improve, and machine learning techniques become increasingly efficient, it may eventually become possible to run certain machine learning tools natively on infected devices, rather than outsourcing the data processing to cloud services or the attackers' C&C servers.

As such, it may be beneficial to monitor the viability of such native tools, as it may heavily reduce the costs of running a large privacy-invasion based ransomware campaign.

7.8.2 False Data

Previous malware has been known to use private data to extort victims. However, this has also led to the creation of scams and “scareware” which capitalises on victims' lack of cyber-security knowledge to extract payment.

One such example is a series of “sextortion” emails that claimed to have collected illicit videos of its victims in a compromising state [155]. While these claims were untrue, the attacker would attempt to convince victims of its legitimacy by including truthful information, such as passwords extracted from unrelated third-party breaches, or by spoofing the sender's email address to make it seem like it was sent from the victims' account.

While many were not convinced by this social engineering attack, over \$100,000 USD was paid to the associated crypto-wallets over five months [100].

If privacy-invasion based ransomware were to become commonplace, private data releases may lend legitimacy to the claims made by such scams, which may increase their likelihood of success.

7.9 Conclusions

In this Chapter, an investigation was performed as to whether IoT devices could be used to facilitate privacy-invasion based ransomware that targets consumers.

First, various data sources commonly found in IoT devices were examined to determine if they could be leveraged by attackers to extract data. Then, methods to identify and process data to extract sensitive user information were proposed. How these methods could be used to perform ransomware attacks was then explored. A system for feasibly managing data collected at scale from IoT devices during a large ransomware campaign was then discussed.

The previously described privacy-invasion techniques were then demonstrated on three IoT devices with differing sensors and data sources. During the demonstrations, various mock “private data” was extracted and sent to a “remote” data collation service, such that an attacker could easily track and process it.

The future developments of privacy-based IoT ransomware were then discussed, before finally identifying the work’s limitations and opportunities for future research.

Chapter 8

Countermeasures

Based on content of previous publications:

“PaperW8: An IoT Bricking Ransomware Proof of Concept” [41],

“Persistence in Linux-Based IoT Malware” [42], and

“Industrialising Blackmail: Privacy Invasion Based IoT Ransomware” [43]

8.1 Introduction

As shown in this work, ransomware is a very destructive form of malware, which typically aims to disable, encrypt or otherwise ransom devices that it infects. By the time a ransom note is displayed, the attacker does not need to maintain the integrity or usability of the device, unless it is in service of forcing the victim to make a payment. As a result, it can have a very direct and damaging effect on its victims, highlighting the need to be proactive with the development and implementation of countermeasures.

In previous chapters, multiple techniques were defined that demonstrated how an attacker could perform effective ransomware attacks on various IoT devices. While the development of countermeasures was not the primary focus of this work, this chapter will cover techniques that could be used to reduce the effectiveness of future ransomware attacks.

First, existing desktop-based ransomware countermeasures will be examined¹,

¹The desktop-based countermeasures section is provided in this chapter to provide context for the following sections that explore IoT-based countermeasures. It should be noted that the desktop-based anti-ransomware tools reviewed in this Chapter were developed by other

and their compatibility with IoT devices will be discussed. Then, “general” IoT-device security techniques will be covered, specifically, methods that prevent the exploitation and installation of malware on IoT devices. Next, methods that could be used to prevent communication-hijacking is explored, followed by anti-bricking techniques that would prevent attackers from weaponising permanent denial of service attacks. Then, techniques to prevent attackers from gaining a persistent foothold in exploited IoT devices will be considered. Finally, methods to prevent attackers from accessing private information stored or collected by the IoT device will be investigated.

8.2 Desktop-Based Countermeasures

Many countermeasures have been suggested by security researchers to mitigate the effects of ransomware. Here, a number of these countermeasures and anti-ransomware tools will be examined.

8.2.1 Best Practices

One of the most effective methods of mitigating the effects of crypto-ransomware is the use of backups. By backing up any important files that have value to the user, infected devices can simply be reset or reinstalled, and any required files can be restored. Ideally, backups should be isolated from the infected device, as otherwise, they run the risk of also being encrypted, and ransomware operators have been known to target backups during an attack [234].

Preventing ransomware from gaining an initial foothold is just as important. Patching vulnerable software, using secure passwords, and avoiding the execution of suspicious files or attachments can prevent ransomware from adversely affecting potentially vulnerable devices.

8.2.2 PayBreak

PayBreak is a tool developed by researchers to combat the use of crypto-ransomware by intercepting and storing keys used to encrypt valuable files [150].

researchers, and for each tool, a citation to the associated paper is provided.

As ransomware has previously been defeated by researchers exploiting weak or broken custom encryption schemes implemented by attackers, ransomware authors may instead use known implementations that have been proven to be secure.

The authors posit that by “hooking” functions that could be used to symmetrically encrypt data, the encryption keys can be intercepted and stored inside of a “key vault”, an append-only file that can only be accessed with administrator privileges. To prevent legitimate keys from being potentially leaked or deleted, any keys that are appended to the vault are encrypted with the user’s public key.

If the user becomes the victim of ransomware, the key vault can then be accessed using the user’s private key. The stored symmetric keys are then iteratively used in attempts to decrypt the encrypted data.

After each attempt, a library named “`libmagic`” is used to identify recognisable file structures within the data. If a suitable file structure is detected, the decryption can continue, and the file is considered to be successfully recovered. Upon any false positives, the user can direct the program to continue attempting decryption with any remaining keys.

The authors created a Windows compatible tool that was tested against 20 active ransomware families. PayBreak was able to successfully recover files from 12 of the 20 tested ransomware families, 9 of which the authors claim had not been previously defeated by other tools. During one round of testing, PayBreak was able to recover 9,821 files encrypted by the “Locky” ransomware family over 6 hours.

In terms of weaknesses, the authors highlight that the method of encryption must be recognised by the system in order to adequately reverse the process. As such, advanced packing² could potentially thwart the system, as statically compiled cryptography functions would not be recognised during the encryption stage. The authors also stated that while PayBreak was not able to intercept keys used by encryption functions implemented by eight of the tested families of ransomware, further work could be performed to support them in the future.

²“Executable packing” is a technique used to compress or otherwise obfuscate code.

8.2.3 CryptoDrop

“CryptoDrop”, another anti-ransomware tool, took a different approach, instead attempting to identify and measure several indicators of ransomware-like behaviour when interacting with the filesystem [221].

These behaviours included:

- **File Type Changes.** As mentioned in Section 8.2.2, files can be identified through the use of “magic numbers”, values at the start of a file that indicate a certain file type. For example, a GIF file may begin with the hex value `0x474946383961` (“GIF89a” in ascii).

If a significant number of files have their file signatures changed, it may indicate malicious activity.

- **Similarity Measurement.** By using “similarity-preserving hash functions”, the authors were able to measure the similarity of a file before and after a modification. If data is being encrypted, it can be assumed that the output of the encryption stage will be dissimilar to the input.
- **Shannon Entropy.** Unlike regular structured files, encrypted and compressed data often exhibit a high level of entropy. If modified files are found to have a marked increase in entropy, it may imply that encryption is being performed.
- **File Deletion.** Some ransomware, rather than encrypting discovered files directly, create a new file in which to write an encrypted version of the target file, before deleting the original. If a great number of user files are being deleted, it could indicate malicious activity.

Using a windows driver, CryptoDrop tracks disk modifications made by processes on the user’s machine. It then uses the previously mentioned indicators to “score” each process for the likelihood of malicious activity. If a process is believed to be malicious, it is prevented from making any further disk alterations unless the user intervenes to give manual approval.

The researchers collected and tested 492 samples of ransomware spanning 14 different families against the CryptoDrop tool, which was able to identify all ransomware with a 100% detection rate.

One downside of this approach is that in order to avoid false positives, a threshold of malicious activity must be met before a process is halted. As such, some files may be encrypted before ransomware is detected and halted. Of the 5,099 test files used for testing the various ransomware, a median of ten files were found to be encrypted before the ransomware was detected.

8.2.4 Unveil

“Unveil” is a tool designed to detect ransomware by monitoring filesystem activity and detecting desktop modifications [147]. The authors determined that ransomware must interact with the device’s filesystem to perform a successful attack. Therefore, Unveil uses the Windows mini-filter driver framework to monitor I/O requests made by various processes. Unveil attempts to identify patterns in file accesses and measures the buffers in read/write requests for large changes in entropy, which can be used to identify crypto-based ransomware.

Additionally, unveil also aims to identify “screen-locker” ransomware. In order to receive payment from victims, ransomware must display a ransom note to the user. Usually, this is achieved by displaying a dialog or image, such as via an application window or desktop background. To detect these messages, Unveil takes a screenshot before and after the tested software is executed. A measure of the “structural similarity” of the images is then taken, with a large difference implying that a ransom note may have been forcibly displayed.

Any displayed text is extracted using an open-source optical character recognition engine called “tesseract” [229], which is then scanned for trigger words, such as “lock” or “encrypt”, which would further indicate ransomware-like behaviour.

Overall, Unveil had a detection rate of 96.3%, with 72.2% reportedly being new detections that were not identified by other anti-virus solutions at the time of research.

8.2.5 ShieldFS

“ShieldFS” is an anti-ransomware tool that focuses on recovering from malicious actions performed by ransomware. It is composed of a collection of mini-filter and kernel drivers, which work in tandem to back up and monitor actions performed on the filesystem by processes [62].

ShieldFS monitors processes for malicious indicators (Such as a high number of files being written, or high entropy write requests). These indicators are used to determine if a process is benign, suspicious, or malicious. When a process is in an “unknown” state, and it opens a file in write or delete mode for the first time, a copy is made in a read-only area. Once ShieldFS is able to classify the process, the file copy is either restored (in the case of malicious activity) or deleted (in the case of benign activity).

If a decision cannot be made, ShieldFS also offers a “system-centric” model, which can be used to monitor the actions of multiple processes, which could potentially be used to identify malicious activity performed by multi-process ransomware.

This tool could allow users to not only prevent ransomware but potentially recover files in real-time after being infected.

8.2.6 Redemption

“Redemption” is a tool that uses process monitoring to detect ransomware-like behaviour in running processes [148]. To do this, the tool is split into two parts, a kernel module and a user-daemon. The kernel module monitors process activity, intercepts file access requests, and stores potential changes to files in a protected area for approval.

The user-daemon, on the other hand, is responsible for assigning malice scores based upon the activity of monitored processes. Events such as overwriting and deleting data, converting files to a specific type, or quickly traversing multiple directories can all lead to a higher score. If the score reaches a certain threshold, the process is marked as malicious, the user is notified, and any changes cached within the protected area can be discarded.

An example implementation of this tool was created for and tested on the Windows operating system. After some initial tests, redemption was able to detect malicious activity performed by all 29 ransomware families that were used for testing. At the optimal malice score threshold, redemption was able to detect ransomware with a 100% detection rate, with 0.8% false positives, with “a median of five exposed files without any data loss” [148].

The authors highlighted Redemption’s similarity to ShieldFS [62], which was

developed “concurrently and independently” [148], as it also attempted to use process behaviour to detect ransomware and prevent damage to users’ filesystems. However, there are some design differences. For example, the authors state that “Unlike ShieldFS, Redemption does not rely on cryptographic primitive identification”.

8.2.7 Compatibility with IoT Devices

Unfortunately, while these tools may function well on desktop machines, extra consideration may need to be taken when applying them to IoT devices.

- **Hardware.** The hardware used by IoT devices is often very limited when compared to the average desktop, which would require any implemented countermeasures to have a very low overhead.
- **Variation.** The variation in IoT devices’ design may also reduce the effectiveness of “universal” or “behaviour-based” anti-ransomware tools.
- **Creative Ransomware.** IoT-based ransomware may take more creative approaches to ransom victims, such as the privacy-invasion techniques shown in Chapter 7, which will bypass anti-ransomware tools that are primarily designed to prevent crypto-based attacks.

8.3 IoT-Based Countermeasures

During this work, obstacles were sometimes encountered that complicated the development and implementation of IoT-based ransomware. In some cases, these complications could be used as the basis of countermeasures to hinder or prevent ransomware from being effectively implemented on IoT devices. While no active “tools” were developed during this work to actively combat ransomware, this section will explore suggestions for countermeasures based on the experience gained when developing ransomware for IoT devices.

8.3.1 General IoT Security

8.3.1.1 Binary Exploit Mitigation Techniques

There are several existing exploit mitigation techniques that are commonly implemented on desktop computers but seem to be significantly rarer on IoT devices [71].

Below, a number of these techniques are examined, and a justification is given for their use. It should be noted that while none of these techniques will guarantee a secure application, they can be difficult to circumvent, increasing the complexity of an attack.

- **Data Execution Prevention (DEP).** A buffer overflow is a common attack used by hackers to write large amounts of data to a buffer in an application that does not have sufficient space to hold it. By writing outside the bounds of a buffer, attackers can overwrite other important data and potentially influence the status of the application [162]. One such example of an important data value is the return address, which dictates where the program will continue execution after the current function has been completed. If the attacker can redirect this address to point to data that they control, such as a buffer that they have written to, they may be able to run their own custom code.

With DEP, certain segments of memory in a process can be marked as non-executable [177; 213], preventing data from being treated as code. Often, this is used to mark the heap and stack, limiting the code that can be run to that which is defined by the developer. This may force attackers to use more complex techniques such as return oriented programming (ROP), which creates payloads by patching together code already present in the binary [216].

- **Address Space Layout Randomisation (ASLR).** For an attacker to “return” to code of their choosing, the address of the code that they wish to run must be known. If an incorrect address is chosen, the application is likely to crash, preventing a successful attack. Without ASLR, the addresses of components within a process will be the same each time it is executed, which would allow an attacker to easily determine the address of a chosen

code snippet to execute. ASLR randomises the location of these components, making it much harder for an attacker to predict the location of any known code, drastically reducing the reliability of an exploit [199].

- **Stack Canaries.** This technique is implemented by generating a random “canary value” when the process starts, then placing it between the local variables and the return address whenever a function is called. If an attacker attempts to modify the return address using a buffer overflow, they will also overwrite the canary value [48]. Before the return instruction is called, the canary value is checked, and if the value has changed, the program is immediately terminated, preventing the exploit from succeeding.

It is unclear why many developers do not implement these techniques [71] when compiling IoT applications. Possible causes could include: significant performance overhead that would limit the effectiveness of the device, the developers simply being unaware of the availability of the techniques, incompatibility with the device hardware, or other unknown reasons. Further research could be performed to determine the source of these issues.

8.3.1.2 Principle of Least Privilege

Linux based IoT devices often run all of their services as root [39], the most privileged account on the Linux operating system. This level of access is likely convenient for developers during the development stage, as it allows them to access any resources required from the device. However, running services as root may increase the severity of any discovered security issues, as when an attacker is able to successfully perform an exploit, they will gain the privileges held by the exploited application.

Therefore, if said application is running as root, the attacker will gain the privileges of the root account, and be granted full access to any other resources available on the device. The attacker could use this access to perform further malicious actions, such as interacting with storage hardware to make persistent modifications, read stored private data, or access attached sensors for the purpose of blackmail.

To prevent attackers from gaining root access, developers can limit the privilege of each service such that they are only given access to the resources they require

to function. In this case, if a service is exploited, the attacker will only gain the *limited* privileges the application holds, reducing the potential damage they could inflict. To perform actions that require root access, the attacker would have to find a method of escalating their privileges, such as via another vulnerability, which would complicate the attack.

8.3.1.3 Device Updates

Once a vulnerability has been discovered for a device, the developers may release an update to prevent it from being exploited “in the wild”, which users can then apply to their devices to prevent attackers from gaining unauthorised access or installing malware. Some of the exploits used within this thesis have been patched by the device’s developers. A summary of the referenced CVEs, and the version in which they were patched, is shown in Appendix B.

The update process will differ from device to device, but various methods can be used to improve users’ experiences and increase the likelihood of updates being applied. First and foremost, developers should endeavour to design the update process such that it is quick and easy to perform, as if the user has been previously inconvenienced, it may discourage them from updating in the future.

Users could also be notified when a new patch is available, warned of the potential dangers, and encouraged to update. Providing users with the option to schedule a more opportune time to update could further improve the likelihood of patches being applied. This notification system could also be used to alert users when support is no longer being provided for a certain device model. Devices that are no longer receiving updates can be very dangerous if new vulnerabilities are discovered. While developers cannot be expected to provide endless support, users should be notified when new vulnerabilities arise. If possible, remediation steps should be provided, such as turning off certain vulnerable features.

Finally, updates could be downloaded and applied automatically by the device, reducing the agency required from a user to limit the spread of malware. However, depending on how the update process is implemented, this may not be feasible. As an example, some devices may need to reboot in order to apply firmware updates, which if performed at an inopportune time, may inconvenience the user. IoT developers may therefore be averse to using this method, as it may result in lower customer satisfaction.

8.3.1.4 Previous Tools

There are a number of tools that have been developed³ that could be used to improve the security of IoT devices and reduce the effectiveness of ransomware. Below, some of these tools are discussed.

- **HADES-IoT.** In 2019, researchers developed a system that provides process whitelisting features to IoT devices, named HADES-IoT [39]. HADES-IoT is designed to only allow programs approved by developers to run on protected IoT devices by intercepting calls to “`execve`”, which is used by the Linux operating system when starting a new process.

First, a “whitelist” of approved programs must be created. HADES-IoT provides a “profiling mode” which can be used to record the device’s behaviour during normal operation. During this mode, when `execve` is called, various parameters (such as the program’s path and binary content) are hashed and added to the whitelist. Once the profiling stage has been completed, the whitelist is saved to the device, and HADES-IoT switches to “enforcing” mode.

When in “enforcing” mode, the parameters in calls to `execve` are hashed and compared to the hashes stored within the whitelist. If the hash is not found, the process is prevented from spawning, which could frustrate attackers attempting to gain persistence and prevent uploaded malware from running. Tests performed during the development of IoT-HADES show that it could be used to prevent previous popular forms of IoT malware (such as Mirai, IoTReaper, and VPNFilter) from spawning new processes and infecting the device [39].

Due to its general low CPU and memory overhead, HADES-IoT is particularly well suited for preventing IoT-based ransomware. The research claims that it can also be adapted for use with many Linux kernels, which would allow it to be used on a large variety of IoT devices. Further details concerning edge cases, proofs of concept and more can be found in the associated paper [39].

³It should be noted that the tools described within this section were created by other researchers. Citations for the associated papers are provided where appropriate.

- **Heldroid.** As mentioned in Section 2.4.1.1, Heldroid is an anti-ransomware tool designed to detect potential ransomware that targets Android-based devices by scanning for “threatening text” (which would imply the use of a ransom note), the use of encryption functions, and common device locking techniques. While many Android ransomware families seem to target mobile devices [167], this does not prevent other device types from being infected [56], which may lead to other Android-based IoT devices being targeted. Therefore, Heldroid could be used to improve the security of potentially vulnerable Android-based IoT devices.

8.4 Preventing Communication Hijacking

In Chapter 4, communication hijacking was highlighted as a method that could be used by attackers to deliver ransom notes to potential victims. This would require the attacker to identify the channels that the device would typically use to communicate with the user during normal operation, then perform a takeover such that a note could be delivered. Developing a countermeasure that would prevent all channels from being hijacked in this manner would be very challenging, however, common channels could potentially be protected through the application of the principle of least privilege (Section 8.3.1.2).

For example, network services are prime candidates for communication hijacking due to their prevalence and adaptability. However, on Linux, ports under 1023 are considered “privileged”, requiring the user to have root access in order to bind them to a service. There are various techniques that could be used by developers to allow certain applications to bind to these privileged ports without root access, such as setting the `CAP_NET_BIND_SERVICE` capability for applications that would run privileged web services [145]. In this case, as long as the attacker is not able to gain root access, they will not be able to hijack services that operate on privileged ports.

However, this would still allow ports above 1023 to be used. In these cases, developers could use filtering services, such as `iptables`, to close ports that are not expected to be used. A more extreme approach would be to set all ports to be marked as privileged when compiling the Linux kernel [34].

A similar approach could be taken for other communication channels, such as

the framebuffer, by restricting device access to only the accounts that require it.

8.5 Preventing Malicious Storage Manipulation

IoT devices need to store important data, such as root filesystems, configuration settings or system kernels, in order to function correctly. If an attacker can access the stored data, they may be able to make malicious modifications to perform attacks that could facilitate ransomware (Chapter 5), or maintain long-term access (Chapter 6).

This section will cover possible countermeasures that could be used to prevent IoT-based malware from performing storage-based attacks.

8.5.1 Effective Factory Reset Processes

For storage, IoT devices often implement embedded flash chips. These can be split into multiple partitions for different purposes, such as separating the device’s operating system from the user’s configuration.

Developers may be inclined to add redundancy to certain partitions that are often modified, as if the data is somehow damaged or corrupted, the device may cease to function. One method that can be used to add redundancy is a “factory reset” process, which can be used to overwrite damaged partitions with “known good” data, reverting any changes and hopefully restoring the device to a stable state.

While factory resets are primarily intended to be used if the device becomes unusable during normal operation, they could also be used to recover a device after it has been infected, as during the recovery process it may overwrite malicious modifications made by an attacker. Victims would then be able to perform other remediation steps, such as performing firmware updates to prevent further exploitation.

However, there is no standardisation as to how factory resets are performed, so the exact process will vary from device to device. As mentioned in Section 5.6.5, developers may create partitions that are not intended to be changed during normal operation of the device, and as this method is mainly used for recovery from benign mistakes, they may not include them in the factory reset process.

For a factory reset to act as an effective countermeasure, the user must be able to reset all partitions that attackers could use to impact the state of the device. Otherwise, attackers may prioritise partitions without redundancy for modification in order to brick, or retain control of, the device. Additionally, the factory reset process itself should be protected from malicious tampering. This could be achieved by making any stored “factory firmware” immutable or implementing the factory reset process separately from the main operating system, such that it cannot be influenced by malware.

8.5.2 Read-Only Partitions

Some data stored on IoT devices, such as the bootloader or kernel, is unlikely to be changed after the device is distributed. Such data can be stored in partitions marked as read-only via the MTD subsystem’s configuration file, which is incorporated into the kernel at compilation time [77]. If a partition is set as read-only, modifications attempted by the user will fail, even when running as root with all the required permissions. Typically, this is used to prevent accidental modifications, such as through programmer error. However, this could also be used to prevent attackers from performing storage-based attacks.

8.5.2.1 Anti-Tampering

This method can be quite difficult to circumvent, as the “MTD_WRITABLE” flag, which is unset to apply the read-only properties during boot, can only be modified by the kernel. However, as shown in Section 6.5.4, it is possible to remove this read-only flag through the use of a malicious kernel module. The simplest method to prevent malicious kernel modules from being used by attackers is to simply disable support for kernel modules entirely when compiling the kernel if it is not required. Alternatively, developers could implement the principle of least privilege and limit the users which can insert modules, or require kernel modules to be signed by the developer to be inserted [99].

8.5.2.2 Adaptable Read-Only Flags

While marking all partitions as read-only to prevent malicious modifications would provide the most protection, some partitions, such as those that hold the filesystem

or configuration settings, will need to be writable for updates to be applied. For partitions that are unlikely to be changed often, developers could set the partition to be read-only flag by default, and then use a kernel module to set the flag when a write needs to occur, such as for an update.

8.5.3 Support for Direct Storage Access

During the creation of the PaperW8, tests were performed to determine if it could be used to effectively ransom real-world devices, such that they could not be recovered by the average victim. Tests that could be performed at runtime via software, such as whether the storage of the device could be modified, were relatively simple. Due to the nature of the filesystem and partitioning system, changes to certain data, such as the bootloader, would only become relevant if the device was rebooted. This allowed small changes to be made, which, if they were deemed damaging, could then be easily reversed via the shell, or by performing a factory reset.

However, as previously mentioned, not all devices have an effective factory reset functionality. If the nature of the test required a damaging modification to not be reversed, such as to test the bricking capability of PaperW8, it was significantly harder to recover the device. To “unbrick” devices in these cases, a test clip was used to directly interact with the flash chip via an external computer, which could be used to overwrite the modified partitions with a backup copy of the “raw” data.

Unfortunately, most victims are unlikely to have the technical ability, nor the hardware, to perform this method of recovery. However, steps could be taken by developers to simplify or standardise the process to make it more accessible to the average user.

Firstly, users could be given easier access to embedded storage, such as via standardised ports, or obvious debug pins for reprogramming. This does have its disadvantages, however, as this may make the device more susceptible to physical attacks, as if an attacker is able to gain physical access, this method could also be used to make malicious modifications.

Secondly, as a backup of the flash chip is often required for recovery, developers could provide a “raw” recovery image that could be written directly to the

flash chip, alongside where firmware updates would normally be provided. Those looking to recover their device should then be able to write the recovery image in its entirety to the chip, which could then be used to bootstrap any further changes.

8.5.4 Data Signing

The use of signatures can allow developers to verify that data contained on a flash chip has not been modified, which can prevent an attacker from gaining persistence. An example implementation of this is uBoot’s “trusted boot” feature, which checks whether an image is correctly signed before continuing the boot process [278].

By cryptographically signing each stage of the booting process – including the bootloader(s), operating system and filesystem – each stage can verify the signature of the next, creating a chain of trust. If a stage has been modified, its signature will not match the expected value, and the device will fail to boot. As an attacker would not have access to the developer’s cryptographic keys, they will be unable to forge a signature for modifications made to any protected stages. It should be noted that immutable memory should be used to bootstrap the process to prevent an attacker from modifying the “root” of the chain of trust, which would allow them to control the rest of the boot process.

This does, however, present the attacker with an opportunity to very easily “brick” the device with minimal changes. While the attacker may not be able to easily gain persistence on the device, they could potentially force the device’s boot process to fail by changing minimal amounts of data, which can result in a failed signature check. This weakness could be mitigated if implemented in tandem with a full “Hard Reset” as described in Section 8.5.1.

8.6 Privacy Invasion Protection

During this work, multiple methods were found to invade the privacy of IoT device users for the purposes of ransomware (Chapter 7). Various countermeasures that could be implemented by device developers, cloud providers or IoT device users to prevent or reduce the impact of these techniques are discussed below.

8.6.1 Preventing Domain Extraction

As shown in Section 7.6.2.1, if a device is positioned in a network such that it can intercept a victim’s traffic, an attacker can extract the domains of websites the victim visits. While users can protect themselves by using privacy tools such as a VPN or Tor [253], it is unrealistic to suggest that every user should use such tools just in case one of their devices is infected with privacy-invasion based ransomware. Alternative methods to secure communication between users and web services must instead be implemented by the website hosts.

As HTTP traffic is designed to be unencrypted by default and requires the domain to be included within the headers, it is very simple to extract information from any traffic generated by the victim. By using HTTPS, the user can limit the information that an attacker can extract through the use of encryption. However, as mentioned in Sections 7.4.2.1 and 7.4.2.2, it is still possible to extract the visited domain or perform downgrade attacks.

Below, some of the methods that can be used to prevent these types of attacks are defined.

8.6.1.1 Encrypted Server Name Indication (ESNI) & Encrypted Client Hello (ECH)

While the contents of HTTPS communication is encrypted, the domain can be extracted from the SNI portion of HTTPS handshake packets. Encrypting this portion of the header using a compatible DNS server would prevent attackers from being able to discern the visited domain [53], while still allowing servers to host multiple web services at the same IP. Encrypted Client Hello (ECH), a more recent protection mechanism that encrypts all metadata contained within “Client Hello” packets, could also be used to prevent domain extraction in the future [207].

8.6.1.2 HTTP Strict Transport Security (HSTS)

In Section 7.4.2.2, HTTPS downgrade attacks were highlighted as a possible method that could be used to intercept the contents of encrypted web service communications. HSTS allows web hosts to force clients to only use HTTPS when visiting their domain, preventing such downgrade attacks from succeeding. Some of the more popular browsers even contain hard-coded lists of HTTPS-only

websites by default [114].

8.6.2 Malicious Activity Detection in Cloud Services

Currently, attackers may find it difficult to natively implement software on infected IoT devices that can locally process data collected from the device’s sensors, such as object recognition models for captured images. While this may change in the future, either through more cost-efficient machine learning algorithms, or greater resources becoming available on the average IoT device, attackers are currently more likely to rely on using external processing methods, such as online cloud services.

As such, attackers may need to use these cloud services at scale to adequately manage the throughput of infected devices. Cloud providers could potentially detect such malicious behaviour through various metrics, such as:

- An account using multiple IP addresses to call the API, which may imply that functions are being called directly from infected IoT devices.
- “Privacy related” functions being called excessively or in certain sequences, such as facial or object recognition followed by nudity detection.
- Whether a trial account is being used, as this may imply that the attacker is aiming to reduce costs by using free processing without payment.

If a cloud service provider identifies a user as malicious, banning or shutting down the associated account may delay the operation of the malware campaign. A more extreme approach may be to prevent accounts from accessing certain functionality commonly associated with privacy-invasion based ransomware until the owner of the account has provided sufficient proof of identity. Ironically, this restriction may infringe on the privacy of those who are legitimately trying to use the service as intended.

8.6.3 Data Devaluation

If a victim has had data stolen during a ransomware attack and is being threatened with its public release, there are very few steps that they can take to remediate the impact, as they will not have any method to delete the stolen data from the

attacker’s storage. However, it may be possible to reduce the trustworthiness of the information attained by the attackers by providing false data to the malware’s data collation server, which will reduce the overall value of any files that are released by the attacker as it becomes more difficult to separate from “real” data.

This may also waste the attacker’s time and resources, as they would have to receive, store and analyse any data sent by the fake “victim”. While it would be possible to blacklist certain IPs or identifiers found to be submitting fake data, this would likely require manual intervention from the malware author, wasting even further time.

8.7 Conclusions

In this Chapter, previous ransomware tools were investigated, and the potential compatibility issues with IoT devices were discussed. General IoT security techniques and tools that could be used to improve the security of Linux-based IoT devices were then explored. The techniques used for the proofs of concept in this work were then examined in more detail, and the possible countermeasures that could be used to reduce their effectiveness were discussed.

From this work, it has been shown that there are steps that can be taken by both developers and consumers to limit the impact of ransomware on IoT devices. However, the variation of IoT devices is likely to lead to similar variations in how ransomware is applied, which may limit the effectiveness of “generic” anti-ransomware approaches. Therefore, IoT developers may need to evaluate various factors, such as the device’s hardware limitations, use case, and peripherals, to determine the appropriate countermeasures to use on a case-by-case basis. An overview of these countermeasures is given below in Table 8.1.

Table 8.1: Applicability of our suggested countermeasures for Linux-based ransomware

Countermeasure	General Device Security	Anti Comm. Hijacking	Anti Bricking	Anti Persistence	Privacy Protection
Binary Exploit Mitigation	✓				
Principle of Least Privilege	✓				
Device Updates	✓				
HADES-IoT Kernel Module	✓				
Privileged Ports		✓			
Effective Factory Reset Process			✓	✓	
Read-Only Partitions			✓	✓	
Direct Storage Access			✓	✓	
Data Signing/ Chain of Trust				✓	
ESNI/ ECH					✓
HTTP Strict Transport Security					✓
Malicious Cloud Activity Detection					✓
Data Devaluation					✓

Chapter 9

Conclusions and Further Work

9.1 Introduction

In this work, the viability of IoT-based ransomware attacks was studied. During this investigation, various ransomware technique proofs of concept were developed, including communication hijacking, bricking-based ransomware, malware persistence, and privacy-invasion based ransomware.

These proofs of concept were able to be implemented on a variety of IoT devices, such as routers, cameras and phones, which all exhibited vulnerability to the suggested techniques. Finally, countermeasures were identified and recommended as mitigations to the discovered techniques, which can be used to reduce the effectiveness of future attacks.

This Chapter provides a discussion and evaluation of the work carried out in this PhD research. This will include a review of the methodology used and the proposed research objectives, followed by the contributions that were made, and an examination of the potential limitations of this work. Finally, the potential future work that could be performed will be explored.

9.2 Methodology Review

Throughout this work, the viability of IoT-based ransomware was evaluated. This did not just involve determining whether ransomware **could** be implemented on an IoT device, but rather to what extent, using what methods, and at what scale.

Economic viability also had to be considered, as without sufficient motivation, attackers would be unlikely to consider IoT devices as valid targets.

During the initial investigation of this topic, a research plan was created with a focus on practical application and testing (shown earlier in Chapter 3). Below, a review of how the proposed plan was implemented in this work is given.

First, attempts were made to investigate the use of crypto-ransomware on IoT. While crypto-ransomware had shown significant success on desktops, servers and laptops, IoT devices presented several limitations, such as the lack of “valuable” files, that would reduce the effectiveness of this approach. However, by implementing communication hijacking (Chapter 4) and bootloader encryption (Chapter 5) as part of the ransomware’s design, it was shown that IoT-based bricking ransomware could be achieved. After creating proofs of concept for multiple devices, it was shown that the methods were generalisable, which would drastically increase the scope of the attack.

While this demonstrated a valid implementation of bricker-based ransomware, the design was adapted from existing malware that did not have IoT devices in mind. Therefore, the resulting ransomware did not fully exploit the features that IoT devices provided to their users. It was theorised that ransomware targeting IoT devices could be more effective if these features were considered during the design stage. For example, the use of sensors natively installed on IoT devices indicated that privacy invasion could be used as a method of ransom. However, upon further consideration, it was established that persistence would most likely be necessary for privacy-invasion based ransomware to be viable, as losing control of an infected device during the information collection stage would likely prevent a ransom attempt from being made. As such, persistence was chosen as the next subject of research.

A study of various storage designs was performed on a number of popular IoT devices. Experimenting with these devices led to the discovery of multiple persistence methods that could feasibly be used by attackers to facilitate malware (Chapter 6). The results of this work also indicated that alternative methods of ransom could potentially be used, and allowed further work to be performed on privacy-invasion based ransomware.

To explore the potential viability of privacy-invasion based IoT ransomware (Chapter 7), a number of IoT devices with different types of sensors were selected

for testing. During this research, data sources were identified and proofs of concept were created to demonstrate the private information that could be extracted from IoT devices by an attacker. Methods to extract speech, images, location and Internet browsing information were all successfully implemented on the test devices.

During this work, “live” experimentation and the development of proofs of concept allowed the viability and limitations of various ransomware types to be assessed, despite the lack of “active” IoT-based ransomware. With this approach, each “Chapter” provided a broader understanding of the risks of IoT-based ransomware, and in some cases produced newly discovered methods that allowed further research to be performed into previously unexplored territory.

9.3 Objective Review

In Chapter 1, various objectives were outlined for this work. These objectives are reviewed below.

Obj-1 Determine the methods that may be used to infect IoT devices with ransomware.

During the experiments performed in this work, multiple methods were used to infect IoT devices. For the initial stages, exploits that could be performed remotely, such as command injections (Section 4.4.4.1, 4.4.4.2, and 4.4.4.4), backdoors (Section 4.4.4.3), and buffer overflows (Section 4.4.4.5), were prioritised, as they could be performed en-masse without requiring physical access, and were the most likely method of entry to be used by future attackers.

Various loading mechanisms were also explored (Section 4.4.2), such as the use of native applications (`tftp`, `wget`), or `echoloding`, depending on the resources made available by the target device.

Obj-2 Identify issues attackers may encounter when attempting to deploy IoT-based ransomware.

When attempting to implement IoT-based ransomware as part of this work, various limitations to the chosen approaches were identified. While some of these

limitations could be overcome, such as discovering methods of communication, or requiring persistence, others were not so easily solved and may dissuade future attackers from attempting to develop ransomware of their own.

For example, bricker-based ransomware was found to be limited by the price of the devices that it infects (Section 5.8), while privacy-based ransomware has to manage the processing and storage of data (Section 7.4.3), which can drastically increase the cost of operating a large-scale ransomware campaign.

Various attacker pitfalls were identified throughout this work. If it was not possible to solve or circumvent these issues, they were considered for potential use as part of a countermeasure or deterrent for future attackers.

Obj-3 Identify IoT devices that are likely to be targeted by IoT-based ransomware.

During the practical experimentation stages of this work, various devices were selected as candidates for testing (Section 4.4, 5.7, 6.6, and 7.6). A number of factors were identified as main contributors for this selection:

- **Vulnerability.** Devices with existing vulnerabilities were likely candidates for future attacks.
- **Popularity.** Relatively unknown devices are unlikely to be targeted by attackers, as it drastically reduces the potential number of victims that could be impacted, and therefore the total likely payout.
- **Available resources.** Some ransomware types required certain resources to be accessible to function correctly. Privacy-invasion based ransomware, for example, required access to devices' sensors in order to extract potentially ransomable information.

Obj-4 Explore the possible impact IoT-ransomware may have on victims.

During this work, it was shown that users of IoT devices could be ransomed using two main methods: device bricking, or privacy invasion. For each of these methods, this work explored the risks both consumers and companies may face should they be infected. Aside from the potential monetary loss should the victim choose to pay the ransom, IoT-based ransomware can cause damage to a victim's property, reputation, and data (Section 5.6.3 and 7.4). If privacy-invasion based

ransomware is used, it may also lead to victims experiencing emotional distress due to the psychological impact of blackmail.

Obj-5 Develop proof of concept attacks to demonstrate the feasibility of IoT-based ransomware threats.

Proofs of concept were created for each proposed ransom method, demonstrating the viability of the approach on a variety of devices (Section 4.4, 5.7, 6.6, and 7.6). During the design stage, the feasibility of large ransomware campaigns was taken into consideration, leading to a focus on generalisation and compatibility. “Modules” were created to specialise for certain tasks, such as extracting audio from phones (Section 7.4.1.2), or Internet traffic from routers (Section 7.4.2.1), indicating that the effectiveness of ransomware may vary, depending on the infected device’s type.

Obj-6 If possible, identify potential countermeasures that could be used by developers and end users.

After proofs of concept were created for each ransom method, the limitations of the approaches were assessed and countermeasures were identified (Chapter 8), such that both users and developers could implement them to mitigate risk.

While this is a good preliminary effort to reduce the impact of IoT-based ransomware, further work could be performed to produce tools designed for this purpose. However, as there are currently limited examples of IoT-based ransomware “in the wild”, there is currently not a high demand for countermeasure tools. The efficacy of such tools would also be difficult to test, due to the lack of “real” examples.

9.4 Contribution Review

In Chapter 1, three research questions were chosen as the focus of this work, and six objectives were outlined as achievable goals. Here, the contributions of this work will be discussed and linked to the previously stated objectives. The progress in answering the research questions will then be evaluated.

Rq-1. At what scale can IoT-based ransomware be implemented?

The scalability of IoT-based ransomware was a constant consideration throughout this research. Bespoke implementations of IoT-based ransomware, while they can be particularly effective or impressive when attacking a targeted device, are less likely to be scalable in a “real” ransomware campaign. The amount of investment required to reverse-engineer the behaviour of individual devices for reliable ransomware would cost significantly more development time, and with limited benefits, when compared to the more “generalised” approaches demonstrated in this work.

The demonstrations in this work emulated behaviour shown by previous IoT malware, such as Mirai, which was able to infect devices numbering in the hundreds of thousands. It was shown that IoT devices could be infected with IoT-based ransomware using various remote exploits (Obj-1), such as command injection or buffer overflows, with no interaction required by the user. This could lead to many IoT devices being remotely infected with ransomware over the Internet without the victims’ knowledge.

Additionally, multiple devices designed for varying purposes were found to be vulnerable to the developed techniques (Obj-3, Obj-5). It should be noted that certain ransomware types will be more effective when installed on devices that provide resources of use to the ransomware. For example, privacy-invasion based ransomware would be more likely to succeed if implemented on devices with multiple sensors from which to extract private data. The number of appropriate devices that can be targeted will influence the scalability of certain ransomware types.

Finally, IoT devices presented limitations that prevented traditional ransomware from being implemented in the same manner as on desktop PCs (Obj-2). These limitations were not insurmountable but could prove costly during a large scale campaign. For example, the limited processing power of the average IoT device required private data to be externally processed when implementing privacy-invasion based ransomware. As such, these limitations, and the cost attackers will need to incur to circumvent them, will influence the scale at which an IoT-based ransomware campaign can operate.

Rq-2. What type of threats does IoT-based ransomware pose to IoT users?

During the development of proofs of concept for the discovered techniques (Obj-5), a number of threats that infected IoT devices can pose were identified. The impacts IoT-based ransomware could have on end-users included the destruction of property, installation of persistent malware, and invasion of the victim’s privacy (Obj-4). The severity of these impacts will depend on the type of devices that are infected (Obj-3) and the scale at which the infections occur. Additionally, large-scale infections of a certain device type or brand may damage the reputation of associated IoT developers, or lower the trust in IoT devices as a whole (Obj-4).

Rq-3. What IoT specific countermeasures can be implemented against IoT-based ransomware attacks?

For each of the discovered ransomware techniques, possible countermeasures were investigated, and appropriate mitigations were suggested (Obj-6). While no “universal” countermeasures to prevent IoT-based ransomware infections were identified, the suggested measures can be used to reduce the impact of such attacks, and frustrate attackers’ attempts to produce new ransomware.

9.5 Limitations

Overall, this work is believed to have sufficiently covered the stated objectives. However, there are some limitations that should be noted, which are discussed below.

9.5.1 Lack of IoT Developer Perspective

Developing proofs of concept for the various ransomware techniques allowed countermeasures to be more easily identified. However, as a significant portion of this work was performed from an “attacker’s” perspective, some nuances may have been overlooked when assessing possible defensive methods.

The perspective of experienced IoT developers may have provided valuable insight during this stage, such as identifying defensive techniques with a high likelihood of success, potential issues in design, or significant software and hardware costs suggested countermeasures may incur.

9.5.2 Development of Countermeasure Tools

The techniques explored within this work have been shown to be highly damaging to IoT devices and their users. While countermeasures that can be used to mitigate the effects of the demonstrated techniques were discussed, automated tools that could be implemented by device owners or manufacturers were not developed. Creating countermeasure tools could provide manufacturers with an easy method of protecting IoT future devices in development.

9.5.3 Commercial Device Focus

The techniques explored in this work were developed with commercial IoT devices as the intended target. While generalisation of the techniques was considered during development, they were not tested on specialised or industrial devices, such as vehicles or ICSs. Given the damage the developed techniques can inflict, testing their potential use in such environments would have been of benefit.

9.5.4 Alternative Operating Systems and Design Architectures

The techniques defined in this work primarily focused on Linux-based IoT devices as, at the time of writing, Linux was one of the most popular operating systems used by IoT developers. Some aspects of this work (such as the methods of gaining persistence on Linux-based IoT systems) are too specific to be transferable, unless the target's operating system is very similar to, or based upon, Linux. However, some of the theory behind the ransomware techniques explored within this thesis could be adapted for use on devices with other operating systems, such as Android or Windows IoT, dependent on a number of factors.

- **Limited Hardware.** Devices that are designed to operate with limited hardware and processing capabilities may be more difficult for attackers to infect, as they will be more constrained by the platform during the design stage of the ransomware. For example, if the attacker needs to limit the use of memory or storage, some features may need to be cut or simplified in order to succeed.

- **Device Cost.** Some operating systems may be designed for use on “low-cost” devices. While ransomware could potentially be implemented on such operating systems, less expensive devices may not be as appealing for attackers, as victims may opt to replace them rather than pay the ransom.
- **Proprietary Technologies.** Some IoT devices may implement proprietary operating systems, applications, or protocols that may make it more difficult for attackers to ransom them. Significant effort may need to be performed by attackers to reverse engineer or exploit such technologies, significantly increasing the malware’s development cost. Additionally, the resulting ransomware may not be re-usable on other devices if the proprietary technology is device- or company specific.
- **Peripheral Support.** Some of the techniques defined within this work rely on access to certain types of peripherals, such as cameras or microphones. If access to these peripherals are not supported by the targeted operating system, such techniques will not be usable.
- **Security Features.** Other operating systems may implement security features that behave differently, or are not commonly encountered on Linux-based IoT systems, such as process sandboxing or mandatory access control. These will need to be considered by the attacker, and if necessary, circumvented or disabled before a ransomware infection can take place.

The devices tested within this thesis performed the majority of processing and user interaction locally. Other devices may use alternative design architectures, such as processing data collected by the device in the cloud. As there is still a requirement for *some* software to be installed to operate the device, ransomware that implements “crypto-bricking” techniques, such as PaperW8, may still be viable for use on these devices. Privacy-invasion type ransomware may also be used if the attacker is able to access the peripherals directly, or intercept sensor traffic being sent between the device and any associated cloud services.

9.6 Further Work

The potential applications of IoT devices are vast, leading to many research opportunities that could be pursued in the future. In this section, further work that could be performed to build upon this research is discussed.

9.6.1 Bricking-Based Ransomware

In Chapter 5, the possibility of bricking-based ransomware was assessed. While bricking-based ransomware was shown to be viable, some areas that could be expanded upon.

9.6.1.1 Monetisation Options

Alternative methods to monetise IoT ransomware could be investigated. For example, attackers could focus on the companies that provide the infected product, pressuring them to pay on behalf of their customers or risk receiving severe backlash from victims. Victims of the malware could even be encouraged by the ransomware to publicly seek assistance from the device’s developing company to further damage the developer’s reputation.

Additionally, if a certain device is targeted as part of a large ransomware campaign, it is within reason to assume that the owning companies may not have readily available stocks to replace those that have been compromised. This would put the manufacturer under extra pressure due to the lack of possible replacements, and customers may be compelled to switch to a competing brand or product to continue operation.

9.6.1.2 Cyberweapon Potential

The potential disruption caused by a very large IoT ransomware-like campaign to critical infrastructures and companies cannot be overstated, particularly if the ransomware implements worm-like capabilities. Techniques not too dissimilar to those highlighted in Chapter 5 could form the basis of cyberweapons acting under the guise of a ransomware attack, with no ulterior economic profitability in mind.

9.6.2 Persistence Techniques

In Chapter 6, the viability of persistent IoT malware was explored. The results of this investigation opened various avenues of research that could be pursued as further work.

9.6.2.1 Further Exploration of Persistence Techniques

In total, six different persistence techniques were explored. While the techniques used were effective in gaining persistence in all of the test devices, this list is by no means exhaustive. Further research could be performed to identify new techniques that may be utilised by attackers.

9.6.2.2 Possible Innovations of Persistent IoT Malware

Being able to achieve persistence may also change how attackers approach the exploitation and infection of IoT devices. While malware authors often compete with others for vulnerable devices (as mentioned in Section 6.2.1), persistence may allow attackers to maintain control of infected devices for longer periods of time. Persistence may also encourage attackers to develop new types of malware, which may not have been previously considered viable without these techniques. This could be explored in further work. Indeed, one such example was demonstrated in Chapter 7.

9.6.2.3 “Single Use” Malware

Unlike stealthier forms of IoT malware, such as those that perform DDoS attacks, IoT-based ransomware must eventually be discovered by design, such that a ransom note can be delivered. Regardless of whether the victim chooses to pay the ransom, given the relatively low cost of many IoT devices, users may be much more inclined to discard them than risk being re-infected.

If this behaviour becomes common, and overall trust in IoT devices is lowered, the pool of potential targets may slowly diminish, leading to reduced effectiveness in any following malware campaigns. Future research could study the effects of IoT malware on device retention and users’ trust in IoT, as it may influence attackers’ future attempts at implementing IoT-based ransomware.

9.6.3 Privacy-Invasion Based Ransomware

In Chapter 7, attempts were made to use IoT devices to facilitate privacy-invasion based ransomware. Below are some subjects that could be pursued to build upon this work.

9.6.3.1 Economic Viability

While it is believed that Chapter 7 sufficiently investigated the technical feasibility of privacy-invasion based IoT ransomware, further research as to the economic viability of this approach may allow researchers to more accurately ascertain the likely direction of IoT-based ransomware.

The cost of obtaining, storing and processing data (either using private servers or cloud services) would need to be considered, alongside the ransom amount set by the attacker, and the likelihood of victims paying the ransom.

9.6.3.2 Psychological Effects

Unlike other malware, which typically aims to restrict access to certain information, privacy-invasion based ransomware instead threatens to expose it, which could potentially be very distressing for victims.

A study of the psychological effects of this style of ransomware could reveal the non-monetary costs of an infection, for example, how public perception of IoT may change, should this malware affect a significant number of devices.

9.6.3.3 ARP Poisoning

In Section 7.4.2.1, a technique to extract private information from intercepted network traffic was discussed. Typically, this method would require the infected device to be positioned such that it could function as a “Man-in-the-Middle” (MitM), with the victim’s network activity passing through it.

Routers, for example, are perfectly positioned for this type of attack, as they act as a gateway to the Internet for other devices. However, devices that do not act as a “gateway” within a network, such as smart cameras, would only be able to analyse their own network activity (i.e., data sent to or from the device in question).

Infected devices that do not act as a gateway may be able to circumvent this restriction through the use of Address Resolution Protocol (ARP) poisoning attacks. Successful attacks would allow attackers to insert infected devices in-between the network gateway and another target device [272].

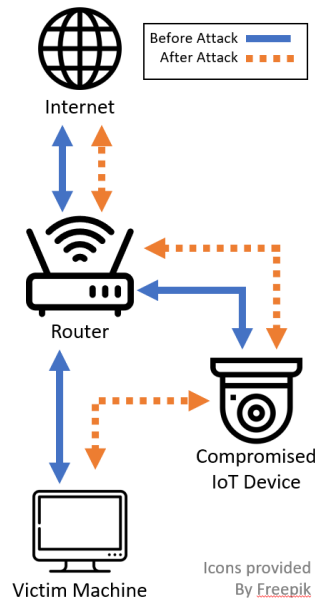


Figure 9.1: Effects of an ARP poisoning attack

Figure 9.1 shows the effects of an ARP poisoning attack on a simplified network. After performing the attack the attacker can redirect Internet traffic through the compromised camera, and extract ransomable information.

While this method could potentially be used to more effectively extract private information, it would likely only be required in certain niche situations and was therefore viewed as out of scope for this work.

9.6.3.4 Linked Accounts

Some IoT devices integrate features provided by external services, such as connections to social media, to seem more appealing to potential customers. However, this may allow attackers that infect such devices to also gain access to these privileged functions.

Research could be performed to assess the security of these connections, and whether they could be abused by IoT malware. For example, privacy-invasion

based ransomware could use linked social media accounts as a source of private information, or as a method of distribution, should a payment not be made.

9.6.3.5 Captive Portal Phishing

In Section 7.4.2.2, the viability of intercepting users' Internet traffic for the purposes of blackmail was discussed. While it was shown that in certain situations it would be possible to intercept the domains victims are browsing, the traffic's contents is likely to be encrypted, unless HTTP is used.

However, as demonstrated in Chapter 4, the attacker may be able to redirect the victim's Internet traffic. By using this technique, a local phishing attack could be performed to weaken the security of victims' encryption while browsing. By creating a malicious captive portal, the attacker would pose as a trustworthy source of information (such as the user's ISP or device manufacturer), and convince the user to install a malicious self-signed root certificate. The victim would then be able to continue browsing, believing their traffic to be encrypted as per usual, but the attacker would then be able to decrypt the traffic's contents and extract private information. Further work could be performed to verify the viability of this attack, and whether IoT devices could feasibly implement it.

9.6.4 Countermeasures

9.6.4.1 Creation of a Countermeasure Framework

In Chapter 8, a number of countermeasures that could be implemented to reduce the effectiveness of the ransomware methods highlighted within this work were discussed. While the countermeasures can be categorised – in a broad sense – based on their effectiveness against certain threats (as shown in Table 8.1), it may be of benefit to develop some form of framework for future countermeasures or mitigations techniques.

The development of such a framework may allow IoT developers to assess their susceptibility to such attacks and more easily identify and implement protections against them, or certify their devices to a certain standard, such that users will be able to identify products which were designed with ransomware-prevention in mind.

In addition, such a framework may provide those attempting to develop new countermeasures with a method of identifying areas which would benefit from further research.

9.7 Final Remarks

Many ransomware techniques that could be implemented for IoT-based ransomware have been demonstrated in this work, such as permanent denial of service, persistence, and privacy invasion, which would inflict significant damage to both IoT devices and their users if implemented “in the wild”.

Currently, IoT-based ransomware is commonly overlooked, but in this work, IoT-based ransomware has been shown to be an outstanding threat, which will only grow in severity as the development of new IoT devices continues to accelerate. To mitigate the future impact of this ransomware, various countermeasures were also defined in this work, which can be implemented by both users and developers to reduce the effectiveness of IoT-based ransomware.

This thesis has explored multiple methods that IoT-based ransomware could use to be implemented on various IoT devices, but given the scale of IoT, it is by no means exhaustive. There is still plenty of further work that can be performed to explore other aspects of IoT-based ransomware, such as for the Industrial Internet of Things, predicting future IoT-based ransomware trends, or the development of countermeasure tools.

Appendix A

D-Link 932L Exploitation

When selecting devices for testing proofs of concept for IoT-based ransomware components, the D-Link 932L was chosen as a potential candidate, as it had been listed as vulnerable to CVE-2019-10999 [196]. This would allow an attacker to perform a remote buffer overflow attack that would result in remote code execution.

An implementation of this exploit was available online [271], however, the 932L was not supported at the time of research. Instead, attempts were made to create a working exploit based on the existing work. First, the firmware was extracted from the device, and the vulnerable application was run in an emulator. `nvramp-faker` was used to simulate the required hardware, and a debugger was attached to identify any crashes.

Using the codebase provided by the previous implementation [271], an exploit for a different version of the camera was run, targeting the virtualised application. The location of the crash was identified, indicating that performing an exploit via this method would still be possible. Using the code written for another device model as a template, support for the 932L device was added. The addresses of an appropriate gadget and the linked `libc` library were identified and written to the exploit, after which it was able to successfully exploit the virtual application. After some further experimentation, the code could be used to exploit a “real” 932L device.

Support for the 932L device was later added to the online implementation by the original developers [271]. Despite being developed separately, the end result was very similar, due in part to the shared codebase. The later version showed

some notable differences, such as support for additional firmware versions (which was deemed unnecessary for this work), and a different gadget being used for the exploitation stage.

Appendix B

CVE Patches

Below, a table is provided detailing the status of patches for the CVEs mentioned within this thesis. This includes the version number, date of production, and references to the patch notes where appropriate.

CVE	Device	Patched	Version	Date	Ref
CVE-2017-17215	HG532 Router	Mitigated ¹	N/A	30/11/17	[124]
CVE-2016-6277	R6250 Router	✓	1.0.4.8	16/01/17	[190]
CVE-2016-20016	TV-7104HE	✗ ²	-	-	-
CVE-2019-3929	WiPG-1000	✓	2.3.2.20	29/04/19	[31]
CVE-2019-10999	5020L Camera	✓	v1.16.01	01/06/17	[72]
CVE-2019-10999	932L Camera	✓	v2.18.01	01/06/17	[72]
CVE-2013-5758	SIP-T38G	Unclear ³	38.70.1.33	28/10/14	[283]

Table B.1: Patches and mitigations provided by the device manufacturers.

¹As Huawei considered this device to be an “End-of-Service Product”, a patch was not developed. Instead, users were provided with steps they could take to mitigate the issue.

²MVPower, the company that manufactured this device, provided no patches or support. PenTestPartners, who discovered this exploit, were also unable to find any updates or further information [251].

³While the patch notes are somewhat vague (in that they do not directly reference CVE-2013-5758), this version is likely to patch the vulnerability, as the patch notes use language that describes effects that are similar to those of the exploit: “Fixed the security vulnerability issue that users could call system commands without restriction” [283].

Bibliography

- [1] abcNEWS (2019). Terrifying video of family’s hacked Ring camera system. <https://abcnews.go.com/GMA/News/video/terrifying-video-familys-hacked-ring-camera-system-67704081/> [Accessed: March 2022].
- [2] Abrams, L. (2016). Jigsaw ransomware decrypted: will delete your files until you pay the ransom. <https://www.bleepingcomputer.com/news/security/jigsaw-ransomware-decrypted-will-delete-your-files-until-you-pay-the-ransom/> [Accessed: March 2022].
- [3] Abrams, L. (2016). TrueCrypter ransomware accepts payment in Bitcoins or Amazon gift card. <https://www.bleepingcomputer.com/news/security/truecrypter-ransomware-accepts-payment-in-bitcoins-or-amazon-gift-card/> [Accessed: March 2022].
- [4] Abrams, L. (2020). BitPyLock ransomware now threatens to publish stolen data. <https://www.bleepingcomputer.com/news/security/bitpylock-ransomware-now-threatens-to-publish-stolen-data/> [Accessed: March 2022].
- [5] Abrams, L. (2021). Qlocker ransomware shuts down after extorting hundreds of QNAP users. <https://www.bleepingcomputer.com/news/security/qlocker-ransomware-shuts-down-after-extorting-hundreds-of-qnap-users/> [Accessed: March 2022].
- [6] Al-Hawawreh, M., den Hartog, F. and Sitnikova, E. (2019). Targeted ransomware: A new cyber threat to edge system of brownfield industrial Internet of Things. *IEEE Internet of Things Journal*, 6(4), pp. 7137–7151.

- [7] Al-rimy, B. A. S., Maarof, M. A. and Shaid, S. Z. M. (2018). Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security*, 74, pp. 144–166.
- [8] Aleph One Ltd (N.D.). Yaffs overview. <https://yaffs.net/yaffs-overview> [Accessed: March 2022].
- [9] Andersen, E. (N.D.). BusyBox: the Swiss army Knife of embedded Linux. <https://busybox.net/about.html> [Accessed: March 2022].
- [10] Andronio, N., Zanero, S. and Maggi, F. (2015). Heldroid: dissecting and detecting mobile ransomware. In *international symposium on recent advances in intrusion detection*, Springer, pp. 382–404.
- [11] Anna-senpai (2017). [FREE] World’s largest net:Mirai botnet, client, echo loader, CNC source code release. <https://hackforums.net/showthread.php?tid=5420472> [Accessed: March 2022].
- [12] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M. et al. (2017). Understanding the Mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*, pp. 1093–1110.
- [13] Arias, O., Wurm, J., Hoang, K. and Jin, Y. (2015). Privacy and security in internet of things and wearable devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2), pp. 99–109.
- [14] Arief, B., Periam, A., Cetin, O. and Hernandez-Castro, J. (2020). Using eyetracker to find ways to mitigate ransomware. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, INSTICC, SciTePress, pp. 448–456.
- [15] ARM (2005). ARMv5 architecture reference manual. <https://developer.arm.com/documentation/ddi0100/i/> [Accessed: March 2022].
- [16] Author Unknown (2010). /proc/bus/usb filesystem output. https://www.kernel.org/doc/Documentation/usb/proc_usb_info.txt [Accessed: March 2022].

- [17] Author Unknown (2017). Huawei router HG532 - arbitrary command execution. <https://www.exploit-db.com/exploits/43414> [Accessed: March 2022].
- [18] Author Unknown (N.D.). Barco wePresent WiPG-1600W wireless presentation system Desktop HDMI + VGA (D-Sub). <https://www.amazon.co.uk/wePresent-WiPG-1600W-wireless-presentation-Desktop/dp/B076T41MVC> [Accessed: April 2020].
- [19] Author Unknown (N.D.). Building external modules. <https://www.kernel.org/doc/html/latest/kbuild/modules.html> [Accessed: March 2022].
- [20] Author Unknown (N.D.). Cramfs - cram a filesystem onto a small ROM. <https://www.kernel.org/doc/Documentation/filesystems/cramfs.txt> [Accessed: March 2022].
- [21] Author Unknown (N.D.). General MTD documentation. <http://www.linux-mtd.infradead.org/doc/general.html> [Accessed: March 2022].
- [22] Author Unknown (N.D.). Kernel module signing facility. <https://www.kernel.org/doc/html/v4.15/admin-guide/module-signing.html> [Accessed: March 2022].
- [23] Author Unknown (N.D.). SquashFS 4.0 filesystem. <https://www.kernel.org/doc/Documentation/filesystems/squashfs.txt> [Accessed: March 2022].
- [24] Author Unknown (N.D.). WePresent WIPG-1000-P. <https://www.ballicom.co.uk/wepresent-r9866100eu-.p1408805.html> [Accessed: April 2020].
- [25] Baines, J. (2019). Crestron AM/Barco wePresent WiPG/Extron Share-Link/Teq AV IT/SHARP PN-L703WA/Optoma WPS-Pro/Blackbox HD WPS/InFocus LiteShow - remote command injection. <https://www.exploit-db.com/exploits/46786> [Accessed: March 2022].
- [26] Bajpai, P. and Enbody, R. (2020). Attacking key management in ransomware. *IT Professional*, 22(2), pp. 21–27.

- [27] Bajpai, P., Enbody, R. and Cheng, B. H. (2020). Ransomware targeting automobiles. In *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*, pp. 23–29.
- [28] Bajpai, P., Sood, A. K. and Enbody, R. (2018). A key-management-based taxonomy for ransomware. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, IEEE, pp. 1–12.
- [29] Balaban, M. (2005). VoIPong user’s manual. <https://web.archive.org/web/20210507215221/www.enderunix.org/voipong/manual/> [Accessed: March 2022] (Original URL: <http://www.enderunix.org/voipong/manual/>).
- [30] Barco (2018). wePresent - wireless presentation system for classrooms, boardrooms, and meeting spaces. - Barco. <https://www.barco.com/en/page/wepresent> [Accessed: March 2022].
- [31] Barco (2022). Firmware for the wePresent WiPG-1000P. <https://www.barco.com/en/support/software/R33050103#version-history> [Accessed: November 2022].
- [32] BBC News (2021). Meat giant JBS pays \$11m in ransom to resolve cyber-attack. <https://www.bbc.co.uk/news/business-57423008> [Accessed: March 2022].
- [33] Birngruber, S., Hehenberger, F., Gründlinger, P., Zeilinger, M. and Vymazal, D. (2017). Netgear Nighthawk firmware update vulnerability. https://iot-lab-fh-ooe.github.io/netgear_update_vulnerability/ [Accessed: March 2022].
- [34] Biro, R., Kempen, F. N. v., Minyard, C. and La Roche, F. (1993). sock.h. <https://elixir.bootlin.com/linux/latest/source/include/net/sock.h#L1447> [Accessed: March 2022].
- [35] Botezatu, B. (2018). Hide and Seek IoT botnet resurfaces with new tricks, persistence. <https://labs.bitdefender.com/2018/05/hide-and-seek-iot-botnet-resurfaces-with-new-tricks-persistence/> [Accessed: March 2022].

- [36] Bowden, T., Bauer, B., Nerin, J., Feng, S. and Seibold, S. (2009). The /proc filesystem. <https://www.kernel.org/doc/Documentation/filesystems/proc.txt> [Accessed: March 2022].
- [37] Bozzato, C. and Cisco Talos (2018). Vulnerability spotlight: Foscam IP video camera firmware recovery unsigned image vulnerability. <https://blog.talosintelligence.com/2018/04/foscam-unsigned-image-vuln.html> [Accessed: March 2022].
- [38] Bozzato, C. and Wyatt, L. (2017). Circle with Disney firmware update signature check bypass vulnerability. https://talosintelligence.com/vulnerability_reports/TALOS-2017-0405 [Accessed: March 2022].
- [39] Breitenbacher, D., Homoliak, I., Aung, Y. L., Tippenhauer, N. O. and Elovici, Y. (2019). HADES-IoT: a practical host-based anomaly detection system for IoT devices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp. 479–484.
- [40] Brian Fox, C. R. (2010). echo(1) - Linux man page. <https://linux.die.net/man/1/echo> [Accessed: March 2022].
- [41] Brierley, C., Pont, J., Arief, B., Barnes, D. J. and Hernandez-Castro, J. (2020). PaperW8: An IoT Bricking Ransomware Proof of Concept. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pp. 1–10.
- [42] Brierley, C., Pont, J., Arief, B., Barnes, D. J. and Hernandez-Castro, J. (2020). Persistence in Linux-Based IoT Malware. In *Secure IT Systems: 25th Nordic Conference, NordSec 2020, Virtual Event, November 23–24, 2020, Proceedings*, Springer Nature, pp. 3–19.
- [43] Brierley, C., Arief, B., Barnes, D. and Hernandez-Castro, J. (2021). Industrialising blackmail: Privacy invasion based iot ransomware. In *Nordic Conference on Secure IT Systems*, Springer, pp. 72–92.
- [44] Brierley, T. (2022). Quote from Tony Brierley, 2022.
- [45] Brierley, V. (2022). Quote from Valerie Brierley, 2022.

- [46] Briggs, B. (2019). Hackers hit Norsk Hydro with ransomware. The company responded with transparency. <https://news.microsoft.com/transform/hackers-hit-norsk-hydro-ransomware-company-responded-transparency/> [Accessed: March 2022].
- [47] Brown, J. (2019). Ring user blocks \$400K Bitcoin extortion attempt by taking out the batteries. <https://gizmodo.com/ring-user-blocks-400k-bitcoin-extortion-attempt-by-tak-1840388093> [Accessed: March 2022].
- [48] Bulba and Kil3r (2000). Bypassing Stackguard and Stackshield. <http://phrack.org/issues/56/5.html> [Accessed: March 2022].
- [49] Cartwright, E., Hernandez Castro, J. and Cartwright, A. (2019). To pay or not: game theoretic models of ransomware. *Journal of Cybersecurity*, 5(1), p. tyz009.
- [50] Cashdollar, L. (2019). SIRT advisory: Silexbot bricking systems with known default login credentials. <https://blogs.akamai.com/sitr/2019/06/sirt-advisory-silexbot-bricking-systems-with-known-default-login-credentials.html> [Accessed: March 2022].
- [51] CCTV Security Pros (2020). Setting up email alerts for CCTV security pros products. <https://www.cctvsecuritypros.com/content/pdfs/e-mail-alert-setup.pdf> [Accessed: March 2022].
- [52] @CDPROJEKTRED (2021). Important update. <https://twitter.com/CDPROJEKTRED/status/1359048125403590660> [Accessed: March 2022].
- [53] Chai, Z., Ghafari, A. and Houmansadr, A. (2019). On the importance of encrypted-SNI ($\{ESNI\}$) to censorship circumvention. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19)*, p. 8.
- [54] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F. and Kohno, T. (2011). Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*.

- [55] Chrysaidos, N. (2015). Mobile crypto-ransomware Simplocker now on steroids. <https://blog.avast.com/2015/02/10/mobile-crypto-ransomware-simplocker-now-on-steroids/> [Accessed: March 2022].
- [56] Cimpanu, C. (2016). Android ransomware infects LG smart TV. <https://www.bleepingcomputer.com/news/security/android-ransomware-infects-lg-smart-tv/> [Accessed: March 2022].
- [57] Cimpanu, C. (2016). Backdoor in MVPower DVR firmware sends CCTV stills to an email address in China. <https://news.softpedia.com/news/backdoor-in-mvpower-dvr-firmware-sends-cctv-stills-to-an-email-address-in-china-500502.shtml> [Accessed: March 2022].
- [58] Cimpanu, C. (2017). BrickerBot author retires claiming to have bricked over 10 million IoT devices. <https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/> [Accessed: March 2022].
- [59] Cimpanu, C. (2019). New Silex malware is bricking IoT devices, has scary plans. <https://www.zdnet.com/article/new-silex-malware-is-bricking-iot-devices-has-scary-plans/> [Accessed: March 2022].
- [60] CloudFlare (N.D.). What is the Mirai botnet? <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/> [Accessed: March 2022].
- [61] Constantin, L. (2017). Ransomware on smart TVs is here and removing it can be a pain. <https://www.pcworld.com/article/3154226/ransomware-on-smart-tvs-is-here-and-removing-it-can-be-a-pain.html> [Accessed: March 2022].
- [62] Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barengi, A., Zanero, S. and Maggi, F. (2016). ShieldFS: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 336–347.
- [63] Corfield, G. (2021). This Netgear SOHO switch has 15 – count ‘em! – vulns, which means you need to upgrade the firmware... now. <https://www.bleepingcomputer.com/news/security/netgear-soho-switch-has-15-vulnerabilities/>

theregister.com/2021/03/11/netgear_jgs516pe_switch_15_vulns/
[Accessed: March 2022].

- [64] Costin, A. and Zaddach, J. (2018). IoT malware: comprehensive survey, analysis framework and case studies. *BlackHat USA*.
- [65] Costin, A., Zaddach, J., Francillon, A. and Balzarotti, D. (2014). A large-scale analysis of the security of embedded firmwares. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 95–110.
- [66] Cox, A., Meskes, M., Ingram, J. and Cinege, D. (2021). watchdog(8) - Linux man page. <https://man7.org/linux/man-pages/man2/socket.2.html> [Accessed: March 2022].
- [67] Cozzi, E., Graziano, M., Fratantonio, Y. and Balzarotti, D. (2018). Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*, IEEE, pp. 161–175.
- [68] CrowdStrike (2021). Ransomware examples: 15 recent ransomware attacks. <https://www.crowdstrike.com/cybersecurity-101/ransomware/ransomware-examples/> [Accessed: March 2022].
- [69] CrowdStrike (2022). Most Common Types of Ransomware. <https://www.crowdstrike.com/cybersecurity-101/ransomware/types-of-ransomware/> [Accessed: November 2022].
- [70] Cui, A. and Stolfo, S. J. (2010). A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pp. 97–106.
- [71] Cyber ITL (2019). Binary hardening in IoT products. <https://cyber-itl.org/2019/08/26/iot-data-writeup.html> [Accessed: May 2021].
- [72] D-Link (2019). DCS-930L/DCS-931L/DCS-932L/DCS-933L/DCS-934L/DCS-5009L/DCS-5010L/DCS-5020L/DCS-5025L/DCS-5030L :: CVE-2019-10999 :: Stack Buffer Overflow. <https://supportannouncement.us.dlink.com/announcement/publication.aspx?name=SAP10131> [Accessed: November 2022].

- [73] D-Link (N.D.). mydlink. <https://www.mydlink.com/> [Accessed: March 2022].
- [74] De Michele, R. and Furini, M. (2019). IoT healthcare: benefits, issues and challenges. In *Proceedings of the 5th EAI international conference on smart objects and technologies for social good*, pp. 160–164.
- [75] Denk, W. (2020). u-boot/image.h. <https://github.com/u-boot/u-boot/blob/master/include/image.h> [Accessed: March 2022].
- [76] DENX Software Engineering (2021). Das u-boot – the universal boot loader. <https://www.denx.de/wiki/U-Boot> [Accessed: March 2022].
- [77] DENX Software Engineering (N.D.). 8.1.1. Memory technology devices. <https://www.denx.de/wiki/publish/DULG/to-delete/FlashFilesystemsMTD.html> [Accessed: March 2022].
- [78] Department of Health & Social Care (2018). Securing cyber resilience in health and care: Progress update October 2018. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/747464/securing-cyber-resilience-in-health-and-care-september-2018-update.pdf [Accessed: March 2022].
- [79] Department of Justice (2021). Department of Justice Seizes \$2.3 Million in cryptocurrency paid to the ransomware extortionists Darkside. <https://www.justice.gov/opa/pr/departments-justice-seizes-23-million-cryptocurrency-paid-ransomware-extortionists-darkside> [Accessed: March 2022].
- [80] Ducklin, P. (2014). Android “FBI lock” malware – how to avoid paying the ransom. <https://nakedsecurity.sophos.com/2014/07/25/android-fbi-lock-malware-how-to-avoid-paying-the-ransom/> [Accessed: March 2022].
- [81] E-ISAC (2016). Analysis of the cyber attack on the Ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 388.

- [82] Eclipse (2015-2020). IoT surveys. <https://iot.eclipse.org/community/resources/iot-surveys/> [Accessed: March 2022].
- [83] Eclipse (2020). 2020 IoT developer survey key findings. <https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2020.pdf> [Accessed: July 2021].
- [84] EnderUNIX (2011). VoIPong. <https://github.com/EnderUNIX/VoIPong> [Accessed: March 2022].
- [85] Europol (2021). 12 targeted for involvement in ransomware attacks against critical infrastructure. <https://www.europol.europa.eu/media-press/newsroom/news/12-targeted-for-involvement-in-ransomware-attacks-against-critical-infrastructure> [Accessed: March 2022].
- [86] F-Secure (2016). Evaluating the customer journey of crypto ransomware. https://f-secure.bg/wp-content/uploads/2016/08/customer_journey_of_crypto-ransomware_f-secure.pdf [Accessed: March 2022].
- [87] F-Secure (N.D.). Trojan:W32/Reveton. https://www.f-secure.com/v-descs/trojan_w32_reveton.shtml [Accessed: March 2022].
- [88] Fabian Bräunlein, L. F. (2019). Smart spies: Alexa and Google Home expose users to vishing and eavesdropping. <https://www.srlabs.de/bites/smart-spies> [Accessed: March 2022].
- [89] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999). RFC2616: Hypertext Transfer Protocol–HTTP/1.1.
- [90] Filiz, B., Arief, B., Cetin, O. and Hernandez-Castro, J. (2021). On the effectiveness of ransomware decryption tools. *Computers & Security*, 111, p. 102469.
- [91] Fisher, D. (2010). New Seftad ransomware attacks master boot record. <https://threatpost.com/new-seftad-ransomware-attacks-master-boot-record-113010/74714/> [Accessed: March 2022].

- [92] Formby, D., Durbha, S. and Beyah, R. (2017). Out of control: ransomware for industrial control systems. In *RSA conference*, vol. 4, p. 8.
- [93] Free Software Foundation (2020). Redirections. https://www.gnu.org/software/bash/manual/html_node/Redirections.html [Accessed: March 2022].
- [94] Froehlich, A. (2021). What Is Leakware? Here's What You Need to Know. <https://www.techtarget.com/searchsecurity/answer/Whats-the-difference-between-extortionware-and-ransomware> [Accessed: November 2022].
- [95] Gartner (2018). Gartner says worldwide sales of smartphones returned to growth in first quarter of 2018. <https://www.gartner.com/en/newsroom/press-releases/2018-05-29-gartner-says-worldwide-sales-of-smartphones-returned-to-growth-in-first-quarter-of-2018> [Accessed: March 2022].
- [96] Gatlan, S. (2019). Maze ransomware demands \$6 million ransom from Southwire. <https://www.bleepingcomputer.com/news/security/maze-ransomware-demands-6-million-ransom-from-southwire/> [Accessed: March 2022].
- [97] Gatlan, S. (2021). QNAP confirms Qlocker ransomware used HBS backdoor account. <https://www.bleepingcomputer.com/news/security/qnap-confirms-qlocker-ransomware-used-hbs-backdoor-account/> [Accessed: March 2022].
- [98] Gazet, A. (2010). Comparative analysis of various ransomware virii. *Journal in computer virology*, 6(1), pp. 77–90.
- [99] Gentoo Foundation (2021). Signed kernel module support. https://wiki.gentoo.org/wiki/Signed_kernel_module_support [Accessed: March 2022].
- [100] Gil Mansharov, A. B. (2019). In the footsteps of a sextortion campaign. <https://research.checkpoint.com/2019/in-the-footsteps-of-a-sextortion-campaign/> [Accessed: March 2022].

- [101] Gillis, A. S. (2021). What is spyware? <https://www.techtarget.com/searchsecurity/definition/spyware> [Accessed: November 2022].
- [102] Goodin, D. (2016). Record-breaking DDoS reportedly delivered by >145k hacked cameras. <https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/> [Accessed: March 2022].
- [103] Goodin, D. (2017). BrickerBot, the permanent denial-of-service botnet, is back with a vengeance. <https://arstechnica.com/information-technology/2017/04/brickerbot-the-permanent-denial-of-service-botnet-is-back-with-a-vengeance/> [Accessed: March 2022].
- [104] Goodin, D. (2020). Android surveillanceware operators jump on the coronavirus fear bandwagon. <https://arstechnica.com/information-technology/2020/03/android-surveillanceware-operators-jump-on-the-coronavirus-fear-bandwagon/> [Accessed: November 2022].
- [105] Goodin, D. (2021). CD Projekt Red does an about-face, says ransomware crooks are leaking data. <https://arstechnica.com/gadgets/2021/06/cd-projekt-red-says-its-data-is-likely-circulating-online-after-ransom-attack/> [Accessed: March 2022].
- [106] Google (2020). faces.py. https://github.com/googleapis/python-vision/blob/main/samples/snippets/face_detection/faces.py [Accessed: March 2022].
- [107] Google (2021). Detect explicit content (safesearch). <https://cloud.google.com/vision/docs/detecting-safe-search> [Accessed: March 2022].
- [108] Google (2021). Detect faces. <https://cloud.google.com/vision/docs/detecting-faces> [Accessed: March 2022].
- [109] Google (2021). Geolocation API. <https://developers.google.com/maps/documentation/geolocation/overview> [Accessed: March 2022].

- [110] Google (2021). Method: `speech.recognize`. <https://cloud.google.com/speech-to-text/docs/reference/rest/v1/speech/recognize> [Accessed: March 2022].
- [111] Google (2022). Cloud Vision API - Try it! <https://cloud.google.com/vision/docs/drag-and-drop> [Accessed: March 2022].
- [112] Google (N.D.). Cloud computing services — Google Cloud. <https://cloud.google.com/> [Accessed: March 2022].
- [113] Google (N.D.). Free trial and free tier — Google Cloud. <https://cloud.google.com/free> [Accessed: March 2022].
- [114] Google (N.D.). HTTP strict transport security. <https://www.chromium.org/hsts/> [Accessed: March 2022].
- [115] Google (N.D.). HTTPS encryption on the web. <https://transparencyreport.google.com/https/overview> [Accessed: March 2022].
- [116] GReAT - Kaspersky Lab (2017). WannaCry ransomware used in widespread attacks all over the world. <https://securelist.com/wannacry-ransomware-used-in-widespread-attacks-all-over-the-world/78351/> [Accessed: March 2022].
- [117] Greenberg, A. (2018). The untold Story of NotPetya, the most devastating cyberattack in history. <https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/> [Accessed: March 2022].
- [118] Grunzweig, J. and Johnston, M. (2016). Bucbi ransomware is back with a Ukrainian makeover. <https://unit42.paloaltonetworks.com/unit42-bucbi-ransomware-is-back-with-a-ukrainian-makeover/> [Accessed: March 2022].
- [119] Grustniy, L. (2019). What’s wrong with “legal” commercial spyware. <https://www.kaspersky.com/blog/stalkerware-spouseware/26292/> [Accessed: November 2022].

- [120] Hernandez-Castro, J., Cartwright, E. and Stepanova, A. (2017). Economic analysis of ransomware. *Available at SSRN 2937641*.
- [121] Hron, M. (2019). The Internet of Thing: how a single coffee maker’s vulnerabilities symbolize a world of IoT risks. <https://blog.avast.com/avast-hacked-a-smart-coffee-maker> [Accessed: March 2022].
- [122] Hsiao, S.-C. and Kao, D.-Y. (2018). The static analysis of WannaCry ransomware. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, IEEE, pp. 153–158.
- [123] Huang, D. Y., Aliapoulios, M. M., Li, V. G., Invernizzi, L., Bursztein, E., McRoberts, K., Levin, J., Levchenko, K., Snoeren, A. C. and McCoy, D. (2018). Tracking ransomware end-to-end. In *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, pp. 618–631.
- [124] Huawei (2021). Security Notice - Statement on Remote Code Execution Vulnerability in Huawei HG532 Product. <https://www.huawei.com/en/psirt/security-notice/huawei-sn-20171130-01-hg532-en> [Accessed: November 2022].
- [125] Hung, G. and Joven, M. (2019). Petya’s master boot record infection. <https://www.fortinet.com/blog/threat-research/petya-s-master-boot-record-infection.html> [Accessed: March 2022].
- [126] IBM (N.D.). IBM Cloud free tier. <https://www.ibm.com/uk-en/cloud/free> [Accessed: March 2022].
- [127] IBM (N.D.). Speech to text demo. <https://speech-to-text-demo.ng.bluemix.net/> [Accessed: March 2022].
- [128] Ilascu, I. (2018). New IoT botnet Torii uses six methods for persistence, has no clear purpose. <https://www.bleepingcomputer.com/news/security/new-iot-botnet-torii-uses-six-methods-for-persistence-has-no-clear-purpose/> [Accessed: March 2022].
- [129] Ilascu, I. (2021). Hacker used ransomware to lock victims in their IoT chastity belt. <https://www.bleepingcomputer.com/news/security/>

- hacker-used-ransomware-to-lock-victims-in-their-iot-chastity-belt/ [Accessed: March 2022].
- [130] Janit0r (2017). BrickerBot source. https://github.com/JeremyNGalloway/mod_plaintext.py/blob/master/mod_plaintext.py [Accessed: March 2022].
 - [131] Janofsky, A. (2021). FBI: JBS ransomware attack was carried out by REvil. <https://therecord.media/fbi-jbs-ransomware-attack-was-carried-out-by-revil/> [Accessed: March 2022].
 - [132] JBS USA (2021). Media statement: JBS USA cybersecurity attack. <https://www.globenewswire.com/news-release/2021/05/31/2239049/17532/en/Media-Statement-JBS-USA-Cybersecurity-Attack.html> [Accessed: March 2022].
 - [133] jclehner (2016). mtd-rw: write-enabler for MTD partitions. <https://github.com/jclehner/mtd-rw> [Accessed: March 2022].
 - [134] jgamblin (2016). Mirai-Source-Code/killer.c (killing competing malware) [accessed: March 2022]. <https://github.com/jgamblin/Mirai-Source-Code/blob/3273043e1ef9c0bb41bd9fcdc5317f7b797a2a94/mirai/bot/killer.c#L519>.
 - [135] jgamblin (2016). Mirai-Source-Code/killer.c (killing services). <https://github.com/jgamblin/Mirai-Source-Code/blob/3273043e1ef9c0bb41bd9fcdc5317f7b797a2a94/mirai/bot/killer.c#L39> [Accessed: March 2022].
 - [136] Jha, P. (2016). Mirai-Source-Code/ForumPost.txt. <https://github.com/jgamblin/Mirai-Source-Code/blob/3273043e1ef9c0bb41bd9fcdc5317f7b797a2a94/ForumPost.txt> [Accessed: March 2022].
 - [137] Jha, P. (2016). Mirai-Source-Code/main.c. <https://github.com/jgamblin/Mirai-Source-Code/blob/3273043e1ef9c0bb41bd9fcdc5317f7b797a2a94/mirai/bot/main.c#L70> [Accessed: March 2022].

- [138] Jha, P. (2017). Mirai source. <https://github.com/jgamblin/Mirai-Source-Code> [Accessed: March 2022].
- [139] Johnson, C. F. (2007). Comparison between Yaffs (Yaffs2) and JFFS2. <https://yaffs.net/comparison-between-yaffs-yaffs2-and-jffs2> [Accessed: March 2022].
- [140] Johnson, M. (1996). Device Driver Basics. <https://tldp.org/LDP/khg/HyperNews/get/devices/basics.html> [Accessed: March 2022].
- [141] JTAG Technologies (2020). Device programming. <https://www.jtag.com/device-programming/> [Accessed: March 2022].
- [142] Kalbo, N., Mirsky, Y., Shabtai, A. and Elovici, Y. (2020). The security of ip-based video surveillance systems. *Sensors*, 20(17), p. 4806.
- [143] kalocpzoep (2014). Twitter Post - @kalocpzoep. <https://twitter.com/a3019397/status/463349646073794561> [Accessed: March 2022].
- [144] Kelley, S. (2017). Dnsmasq. <https://thekelleys.org.uk/dnsmasq/doc.html> [Accessed: March 2022].
- [145] Kerrisk, M. (2021). capabilities(7) — Linux manual page. <https://man7.org/linux/man-pages/man7/capabilities.7.html> [Accessed: March 2022].
- [146] Kerrisk, M. (2021). socket(2) — Linux manual page. <https://man7.org/linux/man-pages/man2/socket.2.html> [Accessed: March 2022].
- [147] Kharaz, A., Arshad, S., Mulliner, C., Robertson, W. and Kirda, E. (2016). {UNVEIL}: A large-scale, automated approach to detecting ransomware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 757–772.
- [148] Kharraz, A. and Kirda, E. (2017). Redemption: Real-time protection against ransomware at end-hosts. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, pp. 98–119.

- [149] Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L. and Kirda, E. (2015). Cutting the gordian knot: a look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, pp. 3–24.
- [150] Kolodenker, E., Koch, W., Stringhini, G. and Egele, M. (2017). Paybreak: defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 599–611.
- [151] Krauss, C. (2021). How the Colonial Pipeline became a vital artery for fuel. <https://www.nytimes.com/2021/05/10/business/colonial-pipeline-ransomware.html> [Accessed: March 2022].
- [152] Krebs, B. (2012). Inside a ‘Reveton’ ransomware operation. <https://krebsonsecurity.com/2012/08/inside-a-reveton-ransomware-operation/> [Accessed: March 2022].
- [153] Krebs, B. (2017). Who is Anna-Senpai, the Mirai worm author? <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/> [Accessed: March 2022].
- [154] Krebs, B. (2018). Mirai co-author gets 6 months confinement, \$8.6M in fines for Rutgers attacks. <https://krebsonsecurity.com/2018/10/mirai-co-author-gets-6-months-confinement-8-6m-in-fines-for-rutgers-attacks/> [Accessed: March 2022].
- [155] Krebs, B. (2018). Sextortion scam uses recipient’s hacked passwords. <https://krebsonsecurity.com/2018/07/sextortion-scam-uses-recipients-hacked-passwords/> [Accessed: March 2022].
- [156] Kroustek, J., Iliushin, V., Shirokova, A., Neduchal, J. and Hron, M. (2018). Torii botnet - not another Mirai variant. <https://blog.avast.com/new-torii-botnet-threat-research> [Accessed: March 2022].
- [157] Kumar, D. K. (2016). Colonial may open key U.S. gasoline line by Saturday after fatal blast. <https://www.reuters.com/article/us-pipeline-blast-alabama-idUSKBN12V2FC> [Accessed: March 2022].

- [158] Land, J. (2016). Multiple Netgear routers are vulnerable to arbitrary command injection. <https://www.kb.cert.org/vuls/id/582384/> [Accessed: March 2022].
- [159] Landley, R. (2005). Ramfs, rootfs and initramfs. <https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt> [Accessed: March 2022].
- [160] Langner, R. (2011). Stuxnet: dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3), pp. 49–51.
- [161] Let’s Encrypt (N.D.). Let’s Encrypt stats. <https://letsencrypt.org/stats/> [Accessed: March 2022].
- [162] Levy, E. (1996). Smashing the stack for fun and profit. <http://phrack.org/issues/49/14.html> [Accessed: March 2022].
- [163] Li, S., Tryfonas, T. and Li, H. (2016). The internet of things: a security point of view. *Internet Research*.
- [164] Li, Y. and Quader, K. N. (2013). NAND flash memory: challenges and opportunities. *Computer*, 46(8), pp. 23–29.
- [165] Liao, K., Zhao, Z., Doupé, A. and Ahn, G.-J. (2016). Behind closed doors: measurement and analysis of CryptoLocker ransoms in Bitcoin. In *2016 APWG symposium on electronic crime research (eCrime)*, IEEE, pp. 1–13.
- [166] linux-mtd (2018). sysfs-class-mtd - the Linux kernel archives. <https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-class-mtd> [Accessed: March 2022].
- [167] Lipovský, R., Štefanko, L. and Braniša, G. (2016). The rise of android ransomware. https://www.welivesecurity.com/wp-content/uploads/2016/02/Rise_of_Android_Ransomware.pdf [Accessed: March 2022].
- [168] Lomas, A. (2020). Smart male chastity lock cock-up. <https://www.pentestpartners.com/security-blog/smart-male-chastity-lock-cock-up/> [Accessed: March 2022].

- [169] LVGL LLC (2021). LittlevGL. <https://lvgl.io/> [Accessed: March 2022].
- [170] Macronix (2013). MX29LV320E T/B datasheet. <https://www.macronix.com/Lists/Datasheet/Attachments/7647/MX29LV320E%20T-B,%203V,%2032Mb,%20v1.3.pdf> [Accessed: March 2022].
- [171] MalwareBytes (2022). All about spyware. <https://www.malwarebytes.com/spyware> [Accessed: November 2022].
- [172] Manning, C. (2010). How YAFFS works. *Retrieved April, 6, p. 2011*.
- [173] Manuel, J., Joven, R. and Durando, D. (2018). OMG: Mirai-based bot turns IoT devices into proxy servers. <https://www.fortinet.com/blog/threat-research/omg--mirai-based-bot-turns-iot-devices-into-proxy-servers> [Accessed: March 2022].
- [174] Marlinspike, M. (2009). New tricks for defeating SSL in practice. *Black Hat DC*, 2.
- [175] Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M. H., Cunha, Í., Guedes, D. and Meira, W. (2018). The evolution of Bashlite and Mirai IoT botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, pp. 00813–00818.
- [176] Micron Technology Inc. (2011). Enabling a flash memory device into the Linux MTD. https://media-www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn0025_enabling_flash_in_linux_mtd.pdf [Accessed: March 2022].
- [177] Microsoft (2021). Data execution prevention. <https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention> [Accessed: March 2022].
- [178] Miller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91).
- [179] MIPS (2014). MIPS® architecture for programmers volume I-A: introduction to the MIPS32® architecture. <https://s3-eu-west->

- 1.amazonaws.com/downloads-mips/documents/MD00082-2B-MIPS32INT-AFP-06.01.pdf [Accessed: March 2022].
- [180] Mochel, P. and Murphy, M. (2011). Sysfs - the filesystem for exporting kernel objects. <https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt> [Accessed: March 2022].
- [181] Mockapetris, P. (1987). Domain names - concepts and facilities. <https://datatracker.ietf.org/doc/html/rfc1034#section-5.3.2> [Accessed: March 2022].
- [182] Mondragon, L. (2021). What’s wrong with “legal” commercial spyware. <https://blog.f-secure.com/what-is-stalkerware/> [Accessed: November 2022].
- [183] Morais, R. (2020). DeepSpeech 0.9.3. <https://github.com/mozilla/DeepSpeech/releases/tag/v0.9.3> [Accessed: March 2022].
- [184] Morse, A. (2018). Investigation: WannaCry cyber attack and the NHS. *Report by the National Audit Office Accessed*, 1.
- [185] Mozilla (2020). Geolocate. <https://ichnaea.readthedocs.io/en/latest/api/geolocate.html> [Accessed: March 2022].
- [186] Murugan, M. and Du, D. H. (2011). Rejuvenator: a static wear leveling algorithm for NAND flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, pp. 1–12.
- [187] Musil, S. (2021). JBS paid \$11M in Bitcoin to resolve ransomware attack. <https://www.cnet.com/tech/services-and-software/jbs-paid-11m-in-bitcoin-to-resolve-ransomware-cyberattack/> [Accessed: March 2022].
- [188] mwarning (2019). mtdRW. <https://github.com/mwarning/mtdRW> [Accessed: March 2022].
- [189] Netgear (2015). R6250 smart WiFi router user manual. http://www.downloads.netgear.com/files/GDC/R6250/R6250_UM_13Apr2015.pdf [Accessed: March 2022].

- [190] Netgear (2017). Security Advisory for CVE-2016-6277, PSV-2016-0245. <https://kb.netgear.com/000036386/CVE-2016-582384> [Accessed: November 2022].
- [191] Netgear (2018). How to upload firmware to a Netgear router using Windows TFTP. <https://kb.netgear.com/000059634/How-to-upload-firmware-to-a-NETGEAR-router-using-Windows-TFTP> [Accessed: March 2022].
- [192] Netgear (2021). Netgear open source code for programmers (GPL). <https://kb.netgear.com/2649/NETGEAR-Open-Source-Code-for-Programmers-GPL> [Accessed: March 2022].
- [193] NIST (2014). Yealink VoIP phone SIP-T38G - remote command execution. <https://nvd.nist.gov/vuln/detail/CVE-2013-5758> [Accessed: March 2022].
- [194] NIST (2017). CVE-2016-6277 detail. <https://nvd.nist.gov/vuln/detail/CVE-2016-6277> [Accessed: March 2022].
- [195] NIST (2018). CVE-2017-17215 detail. <https://nvd.nist.gov/vuln/detail/CVE-2017-17215> [Accessed: March 2022].
- [196] NIST (2019). CVE-2019-10999 detail. <https://nvd.nist.gov/vuln/detail/CVE-2019-10999> [Accessed: March 2022].
- [197] NIST (2019). CVE-2019-3929 detail. <https://nvd.nist.gov/vuln/detail/CVE-2019-3929> [Accessed: March 2022].
- [198] Opdenacker, M. (2013). Cramfs: mark as obsolete. <https://github.com/torvalds/linux/commit/54886a7153353ea2bf21ebfc1b8e030e71d151d7> [Accessed: March 2022].
- [199] Oracle (2019). Address space layout randomization. https://docs.oracle.com/cd/E37670_01/E36387/html/ol_aslr_sec.html [Accessed: March 2022].
- [200] Oracle (2020). What is IoT? <https://www.oracle.com/uk/internet-of-things/what-is-iot/> [Accessed: March 2022].

- [201] Orland, K. (2021). CD Projekt Red source code reportedly sells for millions in dark Web auction [Updated]. <https://arstechnica.com/gaming/2021/02/cd-projekt-red-source-code-reportedly-sells-for-millions-in-dark-web-auction/> [Accessed: March 2022].
- [202] Orozco, A. (2014). How to remove Koler ransomware from Android. <https://blog.malwarebytes.com/cybercrime/2014/05/difficulty-removing-koler-trojan-or-other-ransomware-on-android/> [Accessed: March 2022].
- [203] Osborne, C. (2021). Colonial Pipeline attack: everything you need to know. <https://www.zdnet.com/article/colonial-pipeline-ransomware-attack-everything-you-need-to-know/> [Accessed: March 2022].
- [204] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T. and Rossow, C. (2016). IoTPOT: A novel honeypot for revealing current IoT threats. *Journal of Information Processing*, 24(3), pp. 522–533.
- [205] Palisse, A., Boudier, H. L., Lanet, J.-L., Guernic, C. L. and Legay, A. (2016). Ransomware and the legacy crypto API. In *International Conference on Risks and Security of Internet and Systems*, Springer, pp. 11–28.
- [206] Palmer, D. (2021). Hackers publish thousands of files after government agency refuses to pay ransom. <https://www.zdnet.com/article/hackers-publish-thousands-of-files-after-government-agency-refuses-to-pay-ransom/> [Accessed: March 2022].
- [207] Patton, C. (2020). Good-bye ESNI, hello ECH! <https://blog.cloudflare.com/encrypted-client-hello/> [Accessed: March 2022].
- [208] Petit, J. and Shladover, S. E. (2014). Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent transportation systems*, 16(2), pp. 546–556.
- [209] Postel, J. and Reynolds, J. (1983). Telnet protocol specification. <https://datatracker.ietf.org/doc/html/rfc854> [Accessed: March 2022].

- [210] radware (2017). “BrickerBot” results in permanent denial-of-service. <https://www.radware.com/security/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service/> [Accessed: March 2022].
- [211] Ralson, W. (2021). They told their therapists everything. Hackers leaked it all. <https://www.wired.com/story/vastaamo-psychotherapy-patients-hack-data-breach/> [Accessed: March 2022].
- [212] Raymond, E. (2004). The jargon file - brick. <http://www.catb.org/jargon/html/B/brick.html> [Accessed: March 2022].
- [213] Redhat (2020). What is NX/XD feature? <https://access.redhat.com/solutions/2936741> [Accessed: March 2022].
- [214] Rees, K. (2022). What Is Leakware? Here’s What You Need to Know. <https://www.makeuseof.com/what-is-leakware/> [Accessed: November 2022].
- [215] ReFirmLabs (2019). Binwalk. <https://github.com/ReFirmLabs/binwalk> [Accessed: March 2022].
- [216] Roemer, R., Buchanan, E., Shacham, H. and Savage, S. (2012). Return-oriented programming: systems, languages, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 15(1), pp. 1–34.
- [217] Rohland, C., Dickins, H. and Motohiro, K. (2010). Tmpfs.txt. <https://www.kernel.org/doc/Documentation/filesystems/tmpfs.txt> [Accessed: March 2022].
- [218] Sadeghi, A.-R., Wachsmann, C. and Waidner, M. (2015). Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6.
- [219] Sally, G. (N.D.). Survey of filesystems for embedded Linux. <https://elinux.org/images/b/b1/Filesystems-for-embedded-linux.pdf> [Accessed: March 2022].
- [220] Savage, K., Coogan, P. and Lau, H. (2015). The evolution of ransomware. *Symantec, Mountain View*.

- [221] Scaife, N., Carter, H., Traynor, P. and Butler, K. R. (2016). Cryptolock (and drop it): stopping ransomware attacks on user data. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 303–312.
- [222] Shinder, D. L. (2017). The evolution of extortionware. <https://techtalk.gfi.com/the-evolution-of-extortionware/> [Accessed: November 2022].
- [223] Shodan (2016). D-Link Internet report. <https://dlink-report.shodan.io/> [Accessed: March 2022].
- [224] Shodan (2019). Shodan MVPower DVR search. <https://www.shodan.io/search?query=JAWS%2F1.0> [Accessed: December 2019].
- [225] Shodan (N.D.). Shodan - search engine for the Internet of everything. <https://www.shodan.io/> [Accessed: March 2022].
- [226] Simmonds, C. (2016). Software update for IoT the current state of play. https://elinux.org/images/f/f5/Embedded_Systems_Software_Update_for_IoT.pdf [Accessed: March 2022].
- [227] Simmonds, C. (2016). Software updates for embedded Linux: requirement and reality. <https://mender.io/user/pages/05.resources/04.whitepapers/embedded-linux/mender-whitepaper-software-updates-for-embedded-linux.pdf> [Accessed: March 2022].
- [228] Smith, B. (2007). A quick guide to GPLv3. *Free Software Foundation, Inc Online*: <http://www.gnu.org/licenses/quick-guide-gplv3.html> Referred, 4, p. 2008.
- [229] Smith, R. (2022). Tesseract OCR. <https://github.com/tesseract-ocr/tesseract> [Accessed: March 2022].
- [230] Snyder, P. (1990). tmpfs: a virtual memory file system. In *Proceedings of the autumn 1990 EUUG Conference*, pp. 241–248.
- [231] Sophos (2018). VPNFilter botnet. <https://news.sophos.com/en-us/2018/05/24/vpnfilter-botnet-a-sophoslabs-analysis/> [Accessed: March 2022].

- [232] Spring, T. (2016). Bashlite family of malware infects 1 million IoT devices. <https://threatpost.com/bashlite-family-of-malware-infects-1-million-iot-devices/120230/> [Accessed: March 2022].
- [233] SpywareRemove.com (2017). Wana Decrypt0r 3.0 ransomware. <https://www.spywareremove.com/removewanadecrypt0r30ransomware.html> [Accessed: March 2022].
- [234] Strom, D. (2020). DoppelPaymer: The latest ransomware innovation is all about distribution. <https://blog.avast.com/doppelpaymer-ransomware-resurgence-avast> [Accessed: March 2022].
- [235] Sun, K., Chen, C. and Zhang, X. (2020). "Alexa, stop spying on me!" speech privacy protection against voice assistants. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pp. 298–311.
- [236] Sun Microsystems (2004). Block and Byte Devices. <https://docs.oracle.com/cd/E19455-01/806-1379-10/blockdev.html> [Accessed: March 2022].
- [237] Surbatovich, M., Aljuraidan, J., Bauer, L., Das, A. and Jia, L. (2017). Some recipes can do more than spoil your appetite: analyzing the security and privacy risks of IFTTT recipes. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 1501–1510.
- [238] Swann (2020). Email alerts: how to set up from the DVR per se (DVRx 1590/1600/4480/4575/4780/4980 and NVRx-7450/8580). <https://support.swann.com/s/article/1LdkoAqzpW> [Accessed: March 2022].
- [239] Talos Intelligence (2018). New VPNFilter malware targets at least 500K networking devices worldwide. <https://blog.talosintelligence.com/2018/05/VPNFilter.html> [Accessed: March 2022].
- [240] Talos Intelligence (2018). VPNFilter III: more tools for the Swiss army knife of malware. <https://blog.talosintelligence.com/2018/09/vpnfilter-part-3.html> [Accessed: March 2022].
- [241] Talos Intelligence (2018). VPNFilter Update - VPNFilter exploits endpoints, targets new devices. <https://blog.talosintelligence.com/2018/06/vpnfilter-update.html> [Accessed: March 2022].

- [242] techopedia (2017). dox - verb. <https://www.oxfordlearnersdictionaries.com/definition/english/dox> [Accessed: November 2022].
- [243] techopedia (2017). Doxware. <https://www.techopedia.com/definition/32411/doxware> [Accessed: November 2022].
- [244] TechTerms (2020). Bricking definition. <https://techterms.com/definition/bricking> [Accessed: March 2022].
- [245] The Tcpdump Group (2021). TcpDump/Libcap public repository. <https://www.tcpdump.org/> [Accessed: March 2022].
- [246] Thomson, I. (2017). Forget Mirai – Brickerbot malware will kill your crap IoT devices. https://www.theregister.co.uk/2017/04/08/brickerbot_malware_kills_iot_devices/ [Accessed: March 2022].
- [247] Tidy, J. (2019). How a ransomware attack cost one firm £45m. <https://www.bbc.co.uk/news/business-48661152> [Accessed: March 2022].
- [248] Tidy, J. (2020). How hackers extorted \$1.14m from University of California, San Francisco. <https://www.bbc.co.uk/news/technology-53214783> [Accessed: March 2022].
- [249] Tidy, J. (2021). Cyber criminals publish more than 4,000 stolen Sepa files. <https://www.bbc.co.uk/news/uk-scotland-55757884> [Accessed: March 2022].
- [250] Tidy, J. (2021). “We have your porn collection”: The rise of extortionware. <https://www.bbc.co.uk/news/technology-56570862> [Accessed: November 2022].
- [251] Tierney, A. (2016). New Mirai variant uses multiple exploits to target Routers and other devices. <https://www.pentestpartners.com/security-blog/pwning-cctv-cameras/> [Accessed: March 2022].
- [252] Tierney, A. (2016). Thermostat ransomware: a lesson in IoT security. <https://www.pentestpartners.com/security-blog/thermostat-ransomware-a-lesson-in-iot-security/> [Accessed: March 2022].

- [253] Tor (N.D.). Tor Project — Anonymity online. www.torproject.org/ [Accessed: March 2022].
- [254] Toulas, B. (2021). Police arrest hackers behind over 1,800 ransomware attacks. <https://www.bleepingcomputer.com/news/security/police-arrest-hackers-behind-over-1-800-ransomware-attacks/> [Accessed: March 2022].
- [255] Transforma Insights (2020). Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030, by vertical (in millions). <https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/> [Accessed: March 2022].
- [256] Trend Micro (2016). Ransom notes: know what ransomware hit you. <https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/ransom-notes-know-what-ransomware-hit-you> [Accessed: March 2022].
- [257] Trend Micro (2017). After WannaCry, UIWIX ransomware follows suit. https://www.trendmicro.com/en_gb/research/17/e/wannacry-uiwix-ransomware-monero-mining-malware-follow-suit.html [Accessed: March 2022].
- [258] Trend Micro (2018). Exposed video streams: how hackers abuse surveillance cameras. <https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/exposed-video-streams-how-hackers-abuse-surveillance-cameras> [Accessed: March 2022].
- [259] Trend Micro (2019). Mirai spawn Echobot found using over 50 different exploits. <https://www.trendmicro.com/vinfo/nl/security/news/internet-of-things/mirai-spawn-echobot-found-using-over-50-different-exploits> [Accessed: March 2022].
- [260] Tung, L. (2018). FBI to all router users: reboot now to neuter Russia's VPN-Filter malware. <https://www.zdnet.com/article/fbi-to-all-router-users-reboot-now-to-neuter-russias-vpnfilter-malware/> [Accessed: March 2022].

- [261] Tunggal, A. T. (2021). 17 ransomware examples. <https://www.upguard.com/blog/ransomware-examples> [Accessed: March 2022].
- [262] Turton, W. and Mehrotra, K. (2021). Hackers breached Colonial Pipeline using compromised password. <https://www.bloomberg.com/news/articles/2021-06-04/hackers-breached-colonial-pipeline-using-compromised-password> [Accessed: March 2022].
- [263] Uytterhoeven, G., Hodek, R., Schaller, M. and Neumann, F. (2001). The frame buffer device. <https://www.kernel.org/doc/html/latest/fb/framebuffer.html> [Accessed: March 2022].
- [264] Van Diggelen, F. and Enge, P. (2015). The world’s first GPS MOOC and worldwide laboratory using smartphones. In *Proceedings of the 28th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2015)*, pp. 361–369.
- [265] Vandecappelle, A. (2012). Upgrade without bricking. https://elinux.org/images/6/61/Upgrading_Without_Bricking.pdf [Accessed: March 2022].
- [266] Vignau, B., Khoury, R. and Hallé, S. (2019). 10 years of IoT malware: A feature-based taxonomy. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, pp. 458–465.
- [267] Vlasenko, D., Aina, E., Andersen, E., Anderson, L., Angielski, J., Betts, E., Beppu, J., Candler, B., Chung, R., Cinege, D., Crouse, J., Damm, M., Doolittle, L., Engel, G., Feldman, G., Hegbloom, K., Jacobowitz, D., Kraai, M., Linz, S., Lombardo, J., McGrath, G., Novoa, M., Oleynik, V., Perens, B., Riker, T., Robotti, K., Rosenthal, C., Roskin, P., Sam, G., Torvalds, L., Whitley, M., Wright, C., Zanardi, E. and Ragusa, T. (2021). BusyBox - the Swiss army knife of embedded Linux - commands. <https://www.busybox.net/downloads/BusyBox.html> [Accessed: March 2022].
- [268] Vojtko, M. (2020). Java ransomware (literally): not even your coffee maker is safe. <https://securityboulevard.com/2020/10/java-ransomware->

- literally-not-even-your-coffee-maker-is-safe/ [Accessed: March 2022].
- [269] Voolf, D., Boddy, S. and Cohen, R. (2019). Gafgyt targeting Huawei and Asus routers and killing off rival IoT botnets. <https://www.f5.com/labs/articles/threat-intelligence/gafgyt-targeting-huawei-and-asus-routers-and-killing-off-rival-iot-botnets> [Accessed: March 2022].
 - [270] vxunderground (2020). MalwareSourceCode - ChastityLock. <https://github.com/vxunderground/MalwareSourceCode/blob/main/Python/Trojan-Ransom.Python.ChastityLock.zip> [Accessed: March 2022].
 - [271] Walls, E. (2019). GitHub - fuzzywalls/CVE-2019-10999. <https://github.com/fuzzywalls/CVE-2019-10999> [Accessed: March 2022].
 - [272] Whalen, S., Engle, S. and Romeo, D. (2001). An introduction to ARP spoofing. *Node99 [Online Document]*, https://www.cavalcantetreinamentos.com.br/blog/material-sala-de-aula/SegurancaemRedes/Outros/arp_spoofing_slides.pdf [Accessed: March 2022].
 - [273] Williams, C. (2016). Today the web was broken by countless hacked devices - your 60-second summary. https://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained/ [Accessed: March 2022].
 - [274] Wolf, M., Lambert, R., Schmidt, A.-D. and Enderle, T. (2017). WannaDrive? Feasible attack paths and effective protection against ransomware in modern vehicles. *ESCAR EUROPE*.
 - [275] Woodhouse, D. (2001). JFFS: The journalling flash file system. In *Ottawa linux symposium*, vol. 2001, p. 12.
 - [276] Wortmann, F. and Flüchter, K. (2015). Internet of Things. *Business & Information Systems Engineering*, 57(3), pp. 221–224.
 - [277] Yagh, K., Masters, J., Ben-Yossef, G. and Gerum, P. (2008). *Building embedded Linux systems*, O'Reilly, chap. 8. pp. 219, 236, 297–298.

- [278] Yamada, M. (2017). verified-boot.txt. <https://github.com/u-boot/u-boot/blob/master/doc/uImage.FIT/verified-boot.txt> [Accessed: March 2022].
- [279] Yang, J. and Geng, C. (2018). UbootKit: a worm attack for the bootloader of IoT devices - presentation. <https://i.blackhat.com/briefings/asia/2018/asia-18-Yang-UbootKit-A-Worm-Attack-for-the-Bootloader-of-IoT-Devices.pdf> [Accessed: March 2022].
- [280] Yang, J., Geng, C., Wang, B., Liu, Z., Li, C., Gau, J., Liu, G., Ma, J. and YANG, W. (2019). UbootKit: a worm attack for the bootloader of IoT devices. *BlackHat Asia*.
- [281] Yang, J., Geng, C., Wang, B., Liu, Z., Li, C., Gau, J., Liu, G., Ma, J. and YANG, W. (2020). UbootKit: a worm attack for the bootloader of IoT devices - Video Presentation. <https://www.youtube.com/watch?v=PNnlzP0fPyA> [Accessed: March 2022].
- [282] Yang, T., Yang, Y., Qian, K., Lo, D. C.-T., Qian, Y. and Tao, L. (2015). Automated detection and analysis for android ransomware. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, IEEE, pp. 1338–1343.
- [283] Yealink (2015). Yealink T3X-V70 Release Notes.pdf. <https://support.yealink.com/en/portal/docDetail?documentCode=86c8e32956206b60> [Accessed: November 2022].
- [284] Yilmaz, Y., Cetin, O., Arief, B. and Hernandez-Castro, J. (2021). Investigating the impact of ransomware splash screens. *Journal of Information Security and Applications*, 61, p. 102934.
- [285] Zakharov, A. (2019). Hunting the missing millions from collapsed cryptocurrency. <https://www.bbc.co.uk/news/world-europe-50821547> [Accessed: March 2022].

- [286] Zetter, K. (2016). Inside the cunning, unprecedented hack of Ukraine’s power grid. <https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/> [Accessed: March 2022].
- [287] Zhang, L., Pathak, P. H., Wu, M., Zhao, Y. and Mohapatra, P. (2015). Accelword: energy efficient hotword detection through accelerometer. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 301–315.
- [288] Zhang, N., Mi, X., Feng, X., Wang, X., Tian, Y. and Qian, F. (2018). Understanding and mitigating the security risks of voice-controlled third-party skills on Amazon Alexa and Google Home. *arXiv preprint arXiv:180501525*.
- [289] Zhang, Y., Sun, Z., Yang, L., Li, Z., Zeng, Q., He, Y. and Zhang, X. (2020). A11 Your PLCs Belong to me: ICS Ransomware is realistic. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, pp. 502–509.