University of New Haven

University of New Haven
Digital Commons @ New Haven

Electrical & Computer Engineering and Computer Science Faculty Publications

Electrical & Computer Engineering and Computer Science

9-30-2020

Memory FORESHADOW: Memory FOREnSics of HArDware CryptOcurrency Wallets – A Tool and Visualization Framework

Tyler Thomas
University of New Haven

Mathew Piscitelli
University of New Haven

Ilya Shavrov
University of New Haven

Ibrahim Baggili
University of New Haven, ibaggili@newhaven.edu

Follow this and additional works at: https://digitalcommons.newhaven.edu/electricalcomputerengineering-facpubs

 Part of the Computer Engineering Commons, Electrical and Computer Engineering Commons, Forensic Science and Technology Commons, and the Information Security Commons

Publisher Citation

Tyler Thomas, Mathew Piscitelli, Ilya Shavrov, Ibrahim Baggili, Memory FORESHADOW: Memory FOREnSics of HArDware CryptOcurrency wallets – A Tool and Visualization Framework, Forensic Science International: Digital Investigation, Volume 33, Supplement, 2020, 301002, ISSN 2666-2817, https://doi.org/10.1016/j.fsidi.2020.301002. (https://www.sciencedirect.com/science/article/pii/S2666281720302511)

Comments

Article published in, "*Forensic Science International: Digital Investigation*," Volume 33, Supplement, July 2020.

Forensic Science International: Digital Investigation 33 (2020) 301002

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 USA — Proceedings of the Twentieth Annual DFRWS USA

Memory FORESHADOW: Memory FOREnSics of HArDware CryptOcurrency wallets – A Tool and Visualization Framework

Tyler Thomas [a, b, *], Mathew Piscitelli [a, b], Ilya Shavrov [a, b], Ibrahim Baggili [a, b]

[a] University of New Haven Cyber Forensics Research and Education Group (UNHcFREG), Samuel S. Bergami Jr. Cybersecurity Center, USA
[b] Connecticut Institute of Technology at the University of New Haven, USA

ARTICLE INFO

Article history:

Keywords:
Memory Forensics
Data recovery
Hardware wallet
Cryptocurrency
Bitcoin
Ethereum
Ledger
Trezor

ABSTRACT

We present Memory FORESHADOW: Memory FOREnSics of HArDware cryptOcurrency Wallets. To the best of our knowledge, this is the primary account of cryptocurrency hardware wallet client memory forensics. Our exploratory analysis revealed forensically relevant data in memory including transaction history, extended public keys, passphrases, and unique device identifiers. Data extracted with FORESHADOW can be used to associate a hardware wallet with a computer and allow an observer to deanonymize all past and future transactions due to hierarchical deterministic wallet address derivation. Additionally, our novel visualization framework enabled us to measure both the persistence and integrity of artifacts produced by the Ledger and Trezor hardware wallet clients. The framework can be generalized for use in future memory forensics work.

© 2020 The Author(s). Published by Elsevier Ltd on behalf of DFRWS. All rights reserved. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Nefarious actors are attracted to cryptocurrency because of its privacy focused properties. Anyone may view the full ledger of transactions due to the blockchain being public. However, linking a series of transactions to a given computer or individual can prove difficult with just this information. Cryptocurrency is regularly used to anonymize these transactions. Over one billion dollars worth of illegal cryptocurrency exchanges took place in 2018 (Chainalysis, 2019). These transactions could involve the purchase of narcotics, firearms, or other services provided on the darknet. In addition to black market trading, cryptocurrency is regularly used as the preferred payment method in ransomware attacks.

Anyone with access to the private keys of a cryptocurrency wallet may sign transactions and send funds from the wallet. Because of this, the security of the private keys is of great concern to those who have amassed a large amount of crytpocurrency wealth. A solution thought to be one of the most secure forms of private key storage is the hardware wallet, which although is more secure, will still have processes that run in memory.

Volatile memory is becoming increasingly important in not only forensics, but every aspect of cybersecurity. Modern malware attempts to limit its footprint on the filesystem as much as possible due to the advances made in real-time antivirus monitoring. This makes the collection and analysis of memory critical during incident response. Likewise, the abundance of forensically relevant data in memory can provide investigators with information that would not be recoverable with traditional techniques. The widespread adoption of encryption has also severely limited the power of filesystem forensics alone. Therefore, it is critical for the forensic community to understand what significant forensic artifacts can be recovered from memory for any given application, as well as how long these artifacts persist after the relevant activity has ceased.

In the case of cryptocurrency hardware wallet clients, it is not necessary to save a large amount of data to the filesystem because much of it can be generated during run-time or retrieved by querying a blockchain Application Program Interface (API).

Typically, these clients allow a user to set a passphrase to encrypt some of this cached data and deny access to the device from unauthorized users. In addition, some of these clients run in web-browsers permitting the use of private browsing mode, further limiting the forensic footprint on the filesystem. Conversely, a remote attacker with the ability to read memory may use this information to identify individuals with large amounts of cryptocurrency as targets for blackmail, ransomware, or kidnapping (Popper, 2019; Schlesinger and Day, 2018). This makes the

* Corresponding author. University of New Haven Cyber Forensics Research and Education Group (UNHcFREG), Samuel S. Bergami Jr. Cybersecurity Center, USA
E-mail addresses: tthom10@unh.newhaven.edu (T. Thomas), mpisc1@unh.newhaven.edu (M. Piscitelli), ishav1@unh.newhaven.edu (I. Shavrov), ibaggili@newhaven.edu (I. Baggili).

https://doi.org/10.1016/j.fsidi.2020.301002
2666-2817/© 2020 The Author(s). Published by Elsevier Ltd on behalf of DFRWS. All rights reserved. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

persistence of forensically relevant data in volatile memory invaluable to answering three questions related to cryptocurrency activity: (1) Can the activity of a hardware wallet be detected? (2) Can the transactions be reconstructed and linked back to a given hardware wallet? and (3) Can these transactions be associated with a given computer?

Our work focused on two hardware wallets and their respective software clients, Ledger Nano X and Trezor One. These two models dominate the hardware wallet market with over 1.4 million and 800 thousand units sold respectively (BitcoinNews, 2018; Ledger, 2019). The software architecture of these devices also makes them ideal candidates for analysis due to the widespread adoption of their development frameworks. In this work, we observe memory during run-time to identify and extract data structures containing forensically relevant information and conduct differential memory analysis to determine how persistent these structures are over time. Our work makes the following contributions:

1. To the best of our knowledge, this is the primary account for memory analysis of cryptocurrency hardware wallet clients.
2. We publicly share identified artifacts with the Artifact Genome Project (AGP) (Grajeda et al., 2018).
3. We present an open source tool FORESHADOW in the form of a plugin for the Volatility Framework. FORESHADOW may be used in the analysis of memory dumps from Windows systems for both the Ledger Live and Trezor Wallet hardware wallet clients.[1]
4. We present a novel open source visualization framework and algorithm for exploring the memory persistence and integrity of artifacts.

In Section 2, we review related work and introduce hardware wallets and the forensic data associated with their use. Our apparatus and used applications are detailed in Section 3. Section 4 serves as an overview of the methodology used to create FORESHADOW and our memory visualization framework. Section 5 presents our findings. Section 6 and 7 discuss our results and possible paths forward, respectively. Finally, we make closing remarks in Section 8.

2. Background information and related work

2.1. Background information

Cryptocurrencies such as Bitcoin and Ethereum have seen significant adoption over the last ten years. Recent privacy-focused improvements to cryptocurrency include hardware wallets and Hierarchical Deterministic (HD) wallets.

2.1.1. Cryptocurrency and privacy

Both Bitcoin and Ethereum operate on a decentralized blockchain ledger system which builds upon itself with a series of digital signatures. The self-referencing nature of the blockchain provides non-repudiation without the need for a trusted central authority. In order to send cryptocurrency from one user to another, the recipient shares their public wallet address with the sender. The sender then cryptographically signs the transaction using their private key and sends the transaction request to miners. After the transaction is approved and verified by miners solving a computational problem, it is added to the public ledger in the next block. The recipient can be assured of the validity of the transaction as the transaction persists after new blocks are added.

While the ledger of transactions may be public, there is no trivial way to associate any of these transactions with an individual. The wallet addresses that identify senders and receivers appear as pseudo-random bytes. A recurring theme in the literature of both cryptocurrency enthusiasts and developers is that of privacy (Conti et al., 2018; Zyskind et al., 2015). Privacy in the context of cryptocurrency implies that a third party can neither identify the sender or receiver of any given transaction, nor can they link multiple transactions to a single point of origin.

2.1.2. Hardware wallets

A hardware wallet is a dedicated private key storage device. Typically, these devices go through great efforts to cryptographically isolate the private keys from any part of the device that is externally accessible. This is represented by the secure element in Fig. 1. A software client is used to interact with the device via a paired phone or computer over Bluetooth or Universal Serial Bus (USB). To make a transaction, the transaction is prepared using the software client and sent to the device to be signed. The device then performs a series of authentication steps. Finally, the signed transaction is returned to the software client without the private keys leaving the dedicated device. Two widely used hardware wallet clients are the Ledger Live and Trezor Wallet.

Ledger Live is a software client used to interact with the Ledger Nano X and Ledger Nano S hardware wallets. The application runs in an Electron instance local to the user's machine. Electron is a software development framework that allows developers to encapsulate JavaScript inside of an independent Chromium instance for portability.

Trezor Wallet is a web application used to control the Trezor One and Trezor Model T hardware wallets. The web application communicates with the USB device through a locally installed program called the Trezor Bridge. Trezor Wallet supports both Firefox and Chrome.

2.1.3. Hierarchical deterministic wallets

The Bitcoin development community uses Bitcoin Improvement Proposals (BIP), to introduce new features and functionality for the currency that miners and node operators adopt over time. Occasionally, a BIP will provide enough value that it will be adopted by other cryptocurrencies. Three such BIPs are BIP32, BIP39, and BIP44 (Palatinus and Rusnak, 2014; Palatinus et al., 2013; Wuille, 2012). These proposals outline a framework and methodology for hierarchically deriving public and private key pairs from a single master key pair and seed that can be used for a variety of cryptocurrencies. Hardware wallet manufacturers such as Ledger and Trezor have adopted this implementation, and a rudimentary understanding of this derivation scheme is critical to identifying and understanding the significance of the forensic artifacts that can be extracted from memory.

Wallet clients that implement these three BIPs are referred to as HD wallets. HD wallets enable users to generate a unique address

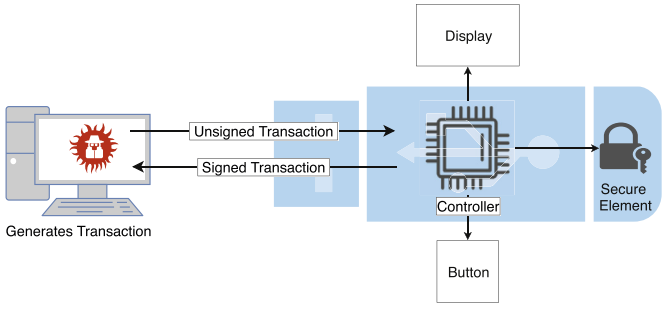

Fig. 1. Hardware wallet security model.

[1] Source code available at https://github.com/UNHcFREG/FORESHADOW.

for every transaction while ensuring that if the user retains their master key pair, they can derive all necessary child keys.

For the purposes of this work, it is only necessary to understand that multiple iterations of this algorithm produce a tree of keys, shown in Fig. 2. It is also important to note that the privacy of this implementation relies upon an observer not being able to move leftward on the key-tree. If an observer is able to move left one level and obtain an extended parent public key, they will be able to deterministically derive all addresses belonging to that parent key. This observer will now be able to de-anonymize both past and future transactions made by the owner of this public key. In this work, our goal is to move leftward on this tree to link several transactions under the same parent key back to a single device by extracting forensic artifacts from memory.

2.2. Past digital forensics of cryptocurrency clients

Previous digital forensic research on cryptocurrency clients by Van Der Horst et al. (2017) focused on Bitcoin Core and Electrum. This work conducted filesystem triage and searched volatile memory for the presence of string literals with known values. However, no attempt was made to generalize these searches and extract data from a system where these values are not known at the time of acquisition. Continuing this work, Zollner et al. (2019) created a tool to automate retrieval of Bitcoin Core and Electrum specific files. Haigh et al. (2019) identified filesystem artifacts left behind by a wide variety of Android cryptocurrency clients and evaluated the security of the applications against tampering and reverse engineering. Gurkok (2015) created a plugin for the Volatility Framework that performed regular expression searches for Bitcoin keys and wallet addresses on memory dumps of the Multibit Bitcoin client on Mac OS X systems.

Most notably, Ali et al. (2018) analyzed the forensic memory artifacts created by Bitcoin and Monero software clients and extracted Bitcoin and CryptoNote network protocol messages from volatile memory. Their findings and tool are the primary account of memory forensic investigation of cryptocurrency network protocols. A limitation to their approach is that artifacts are only present on clients that operate as a full blockchain node connected directly to the network. Many other software clients, including Trezor Wallet and Ledger Live, do not operate as a full node and do not have the network artifacts generated by the blockchain protocol itself.

2.3. Memory forensics

The first formal tool for memory forensics was developed and presented in 2004 (Ford, 2004). Further memory forensics tools came as a result of the Digital Forensics Research Workshop (DFRWS) 2005 Forensics Challenge (DFRWS, 2005). More powerful open source tools such as Volatility, a modular post-mortem memory forensics framework, quickly followed in 2007 (Volatility Foundation, 2018). Memory forensics has since rapidly grown as an area of interest in both incident response and academia due to the quality of tools and research developed by the forensic community.

Most research has focused on memory forensics of the Windows operating system (Schatz, 2007; Schuster, 2006; Zhang et al., 2009). Windows techniques proven to be particularly influential include those that target the allocation pool and registry (Dolan-Gavitt, 2008; Schuster, 2008; Sylve et al., 2016).

While Windows has seen the most attention in the field, some advances have been made in memory forensics for Linux based systems as well (Block and Dewald, 2017; Case et al., 2010; Sttgen and Cohen, 2014). However, memory forensics for mobile devices has received limited attention, due to the challenges posed by acquisition (Sun et al., 2014; Sylve et al., 2012; Thing et al., 2010; Yang et al., 2017).

Other work has focused on application specific memory forensics (Case and Richard, 2016; Casey et al., 2019; Ghafarian and Wood, 2019). As forensic techniques advanced, so did adversarial countermeasures. Palutke and Freiling (2018); Lee et al. (2016); Zheng et al. (2017) presented sophisticated anti-forensics techniques that combat robust acquisition methods.

Lastly, memory visualization and differential analysis have shown promise in advancing the state of the art and the domain (Baum, 2014; Case and Richard, 2017; Garfinkel et al., 2012; Inoue et al., 2011).

Despite these advances, our work is the primary account of combining memory forensics, hardware crypto wallets and a novel visualization framework that employs differential analysis for exploring the memory persistence and integrity of artifacts.

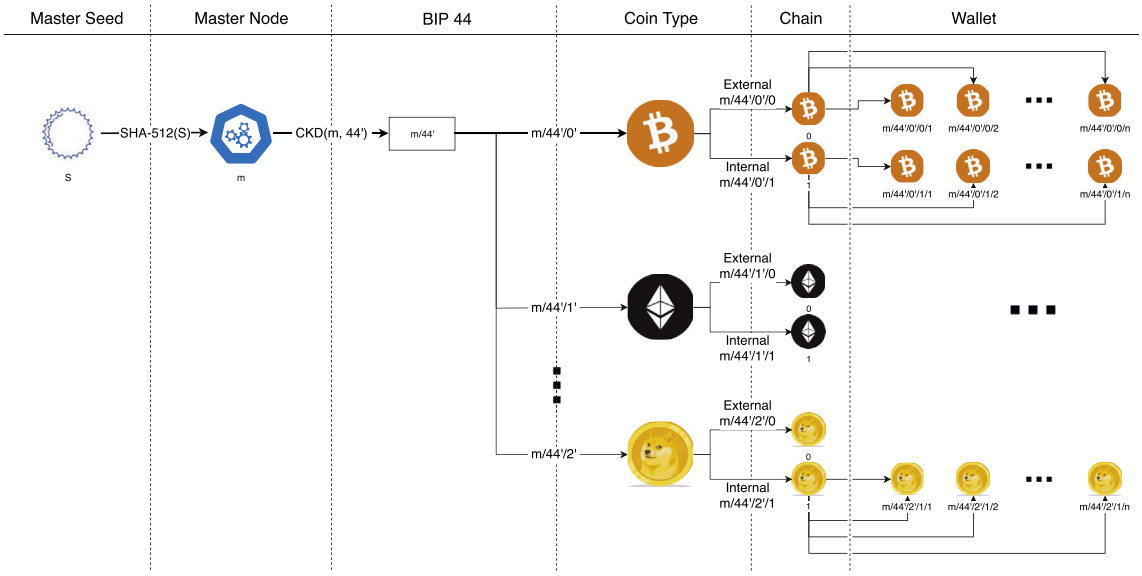

Fig. 2. BIP 32 address derivation scheme.

Table 1
Workstation details.

System Details		Application Details	
Device	Details	Application Name	Version
Processor	Intel Core i7-8750H	Cheat Engine	7.0
Operating System	Windows 7 Professional SP1 7601	VMWare Workstation Pro	15.5.1
System Type	64-bit OS, x64 processor	Mozilla Firefox	71.0
Virtual Memory (VRAM)	2.00 GB	Google Chrome	79.0.3945.88
Ledger Nano X	Firwmare 1.2.4−1	Ledger Live	v1.18.2
Trezor One	Firmware 1.8.3	Trezor Wallet	1.8.3
		Trezor Bridge	2.0.27
		Volatility	2.6

3. Apparatus

All memory analysis took place on a Windows 7 Service Pack 1 64-bit Virtual Machine (VM). Volatility plugin development and testing was conducted using the Win7SP1x64 profile. However, attempts were made to ensure that the plugin was written in such a way that the code would remain profile independent for Windows systems. A detailed account of all software and versions used in this work is presented in Table 1.

4. Methodology

A general overview of the employed methodology can be broken into the following phases:

Scenario creation: The devices were connected to the system and stimulated with normal user behavior.

Identification of data structures: Process memory was observed during runtime to identify candidates for extraction.

Plugin development: FORESHADOW was created to extract observed structures and parse forensically relevant data.

Visualizing physical memory: Statistics about artifacts in the physical memory space were compiled and a tool was created to visualize how these artifacts change over time.

4.1. Scenario creation

The Ledger Live desktop client was downloaded and installed from the official Ledger website without changing the default USB drivers installed by Windows. Although Trezor Wallet is a web application, it still requires users to install the Trezor Bridge software to allow the web browser to abstract away interaction with the USB drivers. As such, Trezor Bridge was downloaded and installed from the official Trezor website, also without changing the default Windows 7 USB drivers. All analysis of Trezor Wallet occurred in private browsing mode.

Although Cheat Engine was used to identify and detect forensically relevant data structures and verify results during testing, all memory dumps were created after a fresh reboot to ensure that any artifacts left behind by the Cheat Engine activity did not taint the acquired dumps. The dumps themselves were collected while performing normal user activity such as logging into the client, sending and receiving funds, and viewing past transactions and account details while the respective client was the only open application other than standard system processes.

4.2. Identification of data structures

In order to explore how forensically relevant data is stored in memory, Cheat Engine was used to observe Ledger Live and Trezor Wallet processes during run-time. Cheat Engine is a real-time memory analysis and debugging tool used by video game hackers. After every action inside the wallet software, a series of string and byte array searches were made using keywords and literals related to the activity. Once a data structure containing relevant information was identified, it was observed over time to determine its suitability as a candidate for extraction. Due to both applications being written in JavaScript, many of these structures were in JSON format. It was immediately clear that both the frequency of occurrence and persistence of the structures varied wildly, especially in Ledger Live. Therefore, a tool for extracting Ledger Live transaction histories must employ a variety of searching strategies because the existence of a given structure cannot be guaranteed. However, in both Firefox and Chrome, Trezor Wallet's memory proved to be more stable.

A second category of structures unique to Ledger Live was also found to contain a large quantity of evidentiary data. After reviewing the Ledger Live source code, it became clear that they were Electron Inter Process Communication (IPC) messages (LedgerHQ, 2019). Electron applications typically run in several processes and use the built-in Electron ipcMain and ipcRenderer JavaScript libraries for messaging.

The third category of data structure is the API request. Because the wallets are not running a full blockchain node, the clients are constantly querying blockchain APIs to monitor price changes and new transactions involving the accounts on the device. These API requests are analogous to network protocol messages in a full node.

```
1: y ← yara.compile(rules)
2: s ← newlist                                  ▷ Serialized structures
3: for task ∈ processList do
4:     if task = LedgerLive.exe
               or task = Chrome.exe
               or task = Firefox.exe then
5:         for vad ∈ getProcessVad do
6:             m ← y.match(vad)                 ▷ Address of match
7:             if m = JSON_match then
8:                 e ← end.match(vad) + offset
9:                 data ← json.serialize(vad[m : e])
10:                s.insert(repair(data))
11:            end if
12:            if m = IPC_match then
13:                data ← vad[m : m + read_size]
14:                data ← data.asciiOnly.split()
15:                s.insert(data)
16:            end if
17:            if m = URL_match then
18:                data ← vad[m :].readUrlParams()
19:                s.insert(data)
20:            end if
21:            if m = XPUB_match then
22:                s.insert(vad[m : xpubSize])
23:            end if
24:        end for
25:    end if
26: end for
```

Algorithm 1. FORESHADOW: Extracting Relevant Structures.

4.3. Plugin development

After the structures were identified with Cheat Engine, the VM was restarted and the same scenario creation process described in Section 4.1 was conducted. Memory dumps were taken by pausing the VM and copying the vmem file. After acquisition, a simple Volatility plugin was made to wrap Yet Another Recursive Acronym (YARA) scans to allow for rapid debugging and testing of different signatures. Once the YARA rules were stable and able to consistently identify the relevant structures, it was then possible to begin extracting forensically relevant data. The accuracy of all data extracted using FORESHADOW was cross-referenced with the current state of the blockchain, as reported by Blockchain Explorer (2020).

Since Volatility plugins are developed in a Python environment, extracting the structures means serializing the raw bytes read from the memory dump into meaningful Python objects. In the case of the JSON structures, this was straightforward. An arbitrarily large number of bytes were read and a second YARA scan was conducted to find where the JSON structure terminated. During testing, it seemed as if 10240 bytes was the optimal read size for extracting the entire structure. In cases of very large transaction histories, this value can be overridden with an optional command line argument. Then, the complete JSON structure was read into a string and serialized with the Python JSON library (Algorithm 1, Line 9). After testing, it became clear that certain edge cases in Ledger Live process memory resulted in a smaller JSON structure being written on top of one being used for extraction. However, this smaller structure was constant in both content and size so it was possible to detect when this happened and repair it by replacing the smaller structure with the delimiters it overwrote.

Extracting Electron IPC messages from Ledger Live proved to be significantly less straightforward because payloads of these messages were JavaScript objects. The JavaScript objects contained data in a structure similar to JSON, but lacked delimiters. A series of transformations was required to meaningfully parse the data. Starting with a single large string with ASCII data separated by null-bytes and non-printable characters, the transformations serialized the data into a list of strings in Python (Algorithm 1, Line 14). Dictionary-like behavior could then be emulated by searching for the desired key in the list and returning the next sequential element in the list as the corresponding value. This method was found to successfully reconstruct the structures regularly. However, the downside to such an approach is that unlike the raw JSON structures, there is no way to know if the data structure has been damaged by overwrites or frees. Instead, sanity checks have be put in place to ensure the desired keys exist and that the data being read is in the expected format. The lack of meaningful delimiters between keys and fields also complicates the repair process. In the event that the plugin detects a damaged IPC structure, it discards it and continues, with a command line option to print discarded structures so that they may be manually reviewed by the user of the tool.

Serializing API requests was trivial as the forensically relevant data was passed as URL parameters (Algorithm 1,18). Because there were also a large number of occurrences of such Uniform Resource Locators (URL) in memory, the information was consistently recoverable without the need to reconstruct damaged or overwritten structures.

Finally, a large number of extended public keys were observed distributed across process memory. In an attempt to collect all extended public keys, even those that are not in the previously defined structures, a final YARA regular expression scan was used to extract extraneous keys.

4.4. Tool use

FORESHADOW is a Volatility plugin for extracting forensically relevant data from the aforementioned structures. The same plugin is used for dumps containing both LedgerLive and Trezor Wallet activity. When ran without command line arguments, FORESHADOW will parse the provided memory dump to produce a report similar to the one shown in Fig. 3.

It may be the case that a data structure is recognized by YARA scans but unable to be serialized by FORESHADOW. In this case, we allow the user to specify a command line option to dump these damaged structures to a target directory for manual review.

4.5. Visualization framework

In order to determine how consistent this methodology would be for extracting transaction data from memory, it was necessary to observe the frequency of occurrence of these structures as the user performs different actions, as well as how long they persist before and after the process is killed. To accomplish this, we constructed a general framework for visualizing and analyzing small contiguous data structures in physical memory.

An overview of the process is shown in Fig. 4. The first of which generates memory dumps by pausing and unpausing the VM and copying the virtual memory file on a fixed time interval of 60 s. Each memory dump is then passed through a custom Volatility plugin that scans the physical memory space to create a Comma Separated Value (CSV) file with each row containing the structure type, location, size, and the base64 encoded data of each found structure. Once a CSV has been created for every memory dump, the files are passed to the second component of the visualization. Here, the CSVs were utilized to generate the illustrations shown in Figs. 5–7.

The visualization component served to accomplish three tasks. First, to illustrate the quantity of artifacts in memory which FORESHADOW was able to detect at each point in time. Second, to present the level of corruption present in the memory artifacts that were detected, or what percentage of the artifact was not yet overwritten by other structures. Third, to spatially visualize individual artifacts in memory and show their change over time.

To determine the number of artifacts present in each memory dump, each resultant CSV file was loaded into a pandas DataFrame, grouped based on the labeled artifact types, and queried to determine unique occurrences. This produced counts of each artifact type for each memory dump, which were plotted using the Matplotlib and seaborn Python libraries, as shown in Fig. 5.

The second component of the visualization required calculating the degree to which each memory artifact had been overwritten. Each artifact was compared to its counterpart in the preceding memory dump to create a vector which represented its integrity. This process is documented in Algorithm 2, and an example is shown in Table 2.

The first appearance of an artifact served as a baseline for its integrity. In Table 2, this would be occurrence 1. Note that both the change vectors and integrity vectors for this occurrence are all ones.

The next time the artifact appeared, a bytewise *and* operation was performed on the current and previous occurrences. This determined which bytes had changed between memory dumps. The result of this operation was a change vector, represented as an array of bits, one for each byte of the artifact's data. A one represented an unchanged bit and a zero represented a changed bit. For example, in occurrences 2 and 3 in Table 2, "$d\ a\ t\ \not{a}$" and "$d\ \not{a}\ t\ \not{a}$" were compared (a strikethrough represents overwritten data). This resulted in a change vector of $[1,\ 0,\ 1,\ 1,]$, due to the difference in the second character. This is shown in line 13 of Algorithm 2.

```
volatility --plugins=/usr/share/volatility/contrib/plugins -f testimage.vmem --profile=Win7SP1x64 foreshadow
Volatility Foundation Volatility Framework 2.6
Scanning Ledger Live.ex pid: 144
Scanning Ledger Live.ex pid: 2852
Scanning Ledger Live.ex pid: 2424
**************************************************
modelId: nanoX
language: en
systemLanguage: en
region: US
appVersion: 1.18.2
path: \\?\hid#vid_2c97&pid_0004&mi_00#9&ab9b3a8&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
osType: Windows_NT
osVersion: 6.1.7601
**************************************************
Found 3 public keys
.....Public key:
        libcore:1:bitcoin:xpub6DJBUQWdgF8c2afh1gY8T1679maU8nRxMQHKeoTxgkWfdWGnfpDnFzLR
        dWc5NghHk2VjvLTYts4Wb9PBP9m6t8LmkrdMn8rfD5L5n6iocK5
.....Public key:
        libcore:1:bitcoin:xpub6CUQpry1t11Dn1Q9D4HwCzjpgMdQzwR7MzvVe6kvvMFAj93RiAonDaFkYvE
        UNJppmG9dLqGQFWWzpVg9u4RdZMr9vCBcrAb3KLuVGKLQ73k
.....Public key:
        libcore:1:ethereum:xpub6BemYiVNp19ZzZoFuD8wsVuMyZD7tBPYuJFAcNZbKyJ49aHSGAHmSs
        D47ZzKyXF6SC91qaVxM4KxXYHVmDd5nyzadCpVW3a42r7tR1YqC4f
**************************************************
Account:
        libcore:1:bitcoin:xpub6DJBUQWdgF8c2afh1gY8T1679maU8nRxMQHKeoTxgkWfdWGnfpDnFzLR
        dWc5NghHk2VjvLTYts4Wb9PBP9m6t8LmkrdMn8rfD5L5n6iocK5
currencyId: bitcoin
satoshis: 1000
operations:
----------
hash: 2584924cf9f29f558966086da52d803d33e3b058cf7bf9d57b678c2b93855fff
.....date: 2019-12-02T18:47:14.000Z
.....satoshis: 1000
.....type: IN
.....senders: bc1qem70k77694v6grx8fe4ama9ju6xe8x0ylcnz7e
.....recipients: 345eNsfVgzvsK9HXd9ZHcky6WWZnYNHd4s bc1q3wxl6kq7vnn30e7ns8qr897ldhssky42s0vml6
```

Fig. 3. FORESHADOW example output.

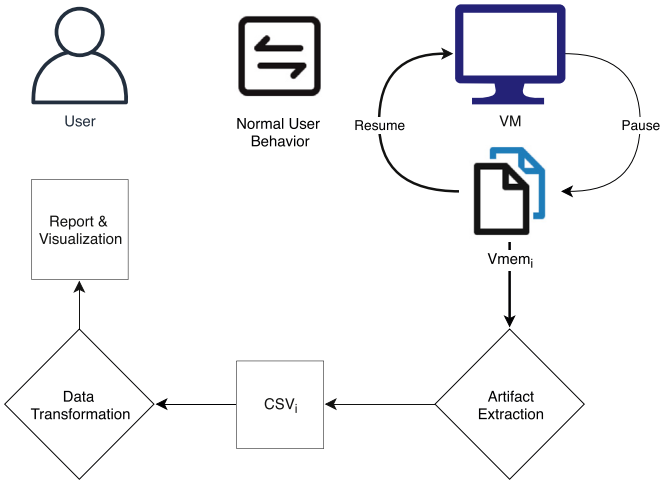

Fig. 4. Data collection and visualization process.

Because this operation only took into account changes from the immediately preceding memory dump, it was necessary to store a vector which represented the integrity of the artifact based on its original occurrence. A bitwise *and* operation was performed on the change vector of the current occurrence and the integrity vector of the previous occurrence. The result of this operation was another bit array again containing one bit for each byte of the artifact's data, however with a one representing a byte which has not changed since the original occurrence, and a zero representing a byte which has changed. This is also shown on line 14 of Algorithm 2. For example, in occurrences 2 and 3 in Table 2, the integrity vector of occurrence 2 is $[1, 1, 1, 0]$, and the change vector of occurrence 3 is $[1, 0, 1, 1]$. A logical *and* of those two vectors produced the integrity vector of occurrence 3, $[1, 0, 1, 0]$. NumPy's vectorization methods were used to increase the efficiency of these operations. While it is a limitation that this algorithm does not take into account the case where a byte may have been overwritten with its original value, this does not affect the extractability of the data.

With the integrity of each artifact calculated, a line plot was generated to visualize the average corruption of each artifact. In Fig. 6, a y-axis value of 1.0 represents an unchanged artifact, while 0.5 represents an artifact where half of its bytes have been overwritten.

The visualization framework also served to spatially display artifacts at their location in memory. Fig. 7 shows these visualizations for two memory dumps captured while the Ledger application was running. This component serves as a direct view into the presence and integrity of the artifacts, contrasting the statistical representations in the prior two components. Areas of memory are labeled to illustrate examples of artifacts which have been partially corrupted or entirely overwritten.

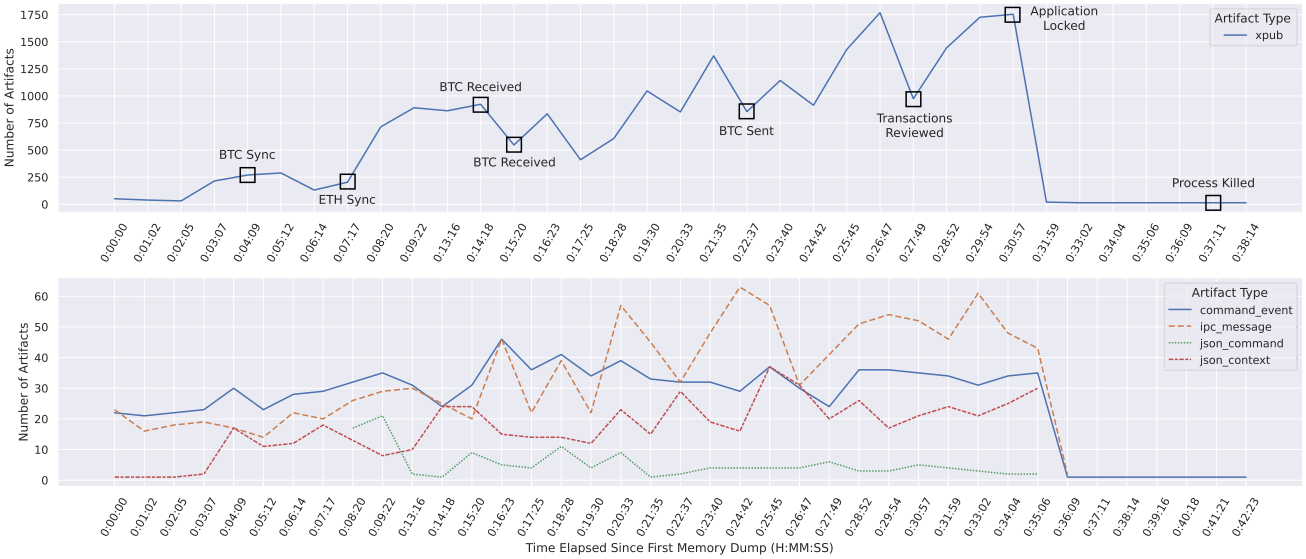

Fig. 5. Ledger: Quantity of artifacts in memory over time.

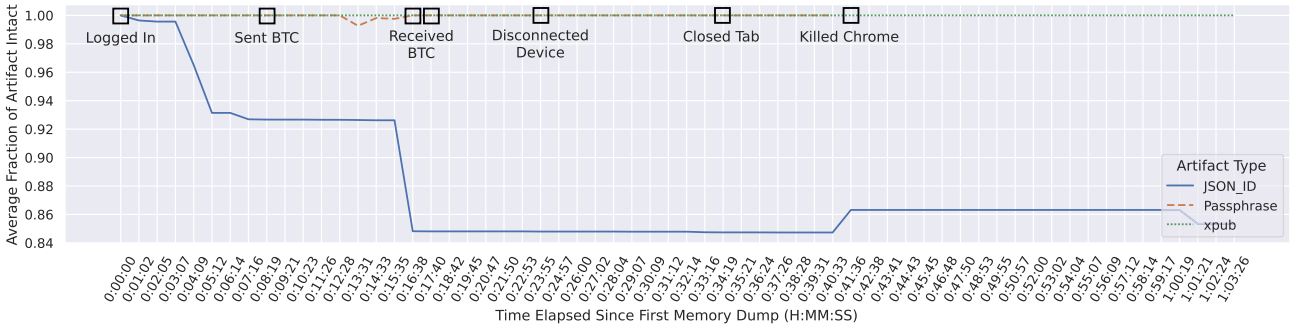

Fig. 6. Trezor (chrome): Integrity of artifacts in memory over time.

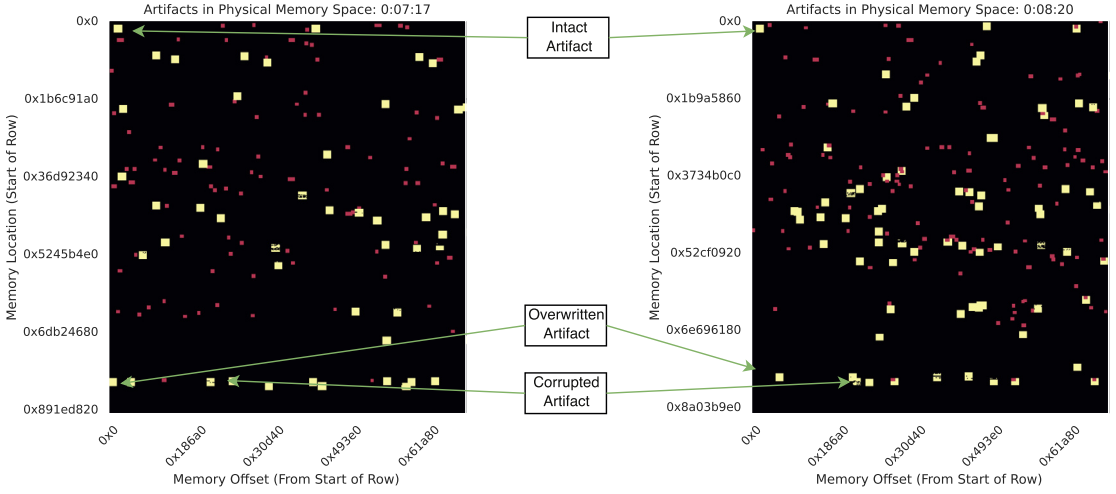

Fig. 7. Ledger: Visualization of artifacts in physical memory space.

```
function CalcMemDumpIntegrity(currData, prevData)
    for currArtifact ∈ currData do                ▷ Each artifact occurrence in memory dump
        currArtifact.integrity ← CalcArtifactIntegrity(currArtifact, prevData)
    end for
end function

function CalcArtifactIntegrity(currArtifact, prevData)
    if containsArtifact(currArtifact, prevData) then
        prevArtifact ← matchArtifact(currArtifact, prevData)        ▷ Has occurred before
    else
        prevArtifact ← currArtifact                                 ▷ First occurrence
    end if
    changeVector ← bytewiseAnd(currArtifact.data, prevArtifact.data)
    integrityVector ← bitwiseAnd(changeVector, prevArtifact.integrity)
    return integrityVector
end function
```

Algorithm 2. IntegrityCalculator: Calculating Corruption of Memory Artifacts.

5. Findings

Table 3 shows the identified data structures and their descriptions. This subsection serves as an overview of the forensic artifacts contained in each of the previously mentioned data structures, as well as the results of the differential analysis and visualization.

5.1. Ledger Live

Ledger Live's process memory consistently contained a large quantity of preference settings and device information such as the model ID, language, region, the application version, USB ID of the device, operating system, and operating system version. This configuration data was regularly able to be extracted with FORESHADOW. The extended public keys currently synced with the device were also regularly found, along with the full transaction history, balance, derivation path, and fresh addresses of every extended public key. Due to the volatility of these structures and how regularly they overwrote each other, it was rarely possible to extract this information for all public keys at once. However, the extended public key is all that is required for a forensic investigator to derive the other information. Some of the data persisted in memory even after the client automatically locked itself with a passphrase. The passphrase was not found in memory.

Analyzing the artifacts present across multiple memory dumps yielded an upward trend, with the number of artifacts increasing over time. Actions carried out in the Ledger client during the experiment, such as the initial synchronization of Bitcoin and Ethereum, caused spikes, as shown in Fig. 5. Upon locking the application, most artifacts were quickly overwritten in memory. Six minutes after locking the application, only fourteen instances of an extended public key and one instance of a command-event artifact were present, compared to peaks in excess of 1700 and 40 respectively. Within 6 min of terminating the Ledger process, all of the remaining artifacts had been overwritten and were undetectable by the Volatility plugin.

Table 2
Artifact Integrity Calculation Example. Note: A strikethrough represents overwritten data.

Occurrence	Data	Change	Integrity
1	d a t a	[1, 1, 1, 1]	[1, 1, 1, 1]
2	d a t ~~a~~	[1, 1, 1, 0]	[1, 1, 1, 0]
3	d ~~a~~ t ~~a~~	[1, 0, 1, 1]	[1, 0, 1, 0]
4	d ~~a~~ t ~~a~~	[1, 1, 0, 1]	[1, 0, 0, 0]

The quantitative representation of the artifacts also demonstrates that numerous copies of each artifact are being stored in memory at any particular point in time. For example, the JSON command artifact containing transaction information is present as many as 20 times in memory at once, as shown in Fig. 5. These large quantities contrast what was seen while viewing process memory in Cheat Engine, where only a few instances of any artifact was present when manually searched for. This may be due to freed artifacts still being present and cached in physical memory (Inoue et al., 2011).

Additionally, the spatial memory diagram shown in Fig. 7 helps illustrate these spikes in artifact quantity. Fig. 7 shows the comparison between the artifacts in memory at time 0:07:17 versus those present at time 0:08:20, when the Ethereum was synchronized. The larger yellow blocks represent command-event, ipc-message, and JSON artifacts, which increased in number at that time. Moreover, this visualization shows which specific areas of memory were overwritten between two times. For example, the annotated area in the lower-left corner of the diagrams in Fig. 7 shows an area of memory which was partially overwritten between two dumps.

5.2. Trezor Wallet

In both Chrome and Firefox, a reliably accessible JSON structure was found to contain forensically relevant information such as the unique device ID, encrypted passphrase, version numbers for both the device firmware and boot loader, boot loader hash, whether or not the device is backed up, model ID, and all extended public keys currently synced with the device. During testing, this data was

Table 3
Data structures.

Data	Contents
LedgerLive	
command-event	Transaction history and public keys
JSON command	Transaction history and public keys
ipc-message	Public keys
JSON context	Device metadata
API Request	Wallet addresses
xpub	Public keys
Trezor Wallet (Firefox)	
JSON Result	Transaction history
Trezor Wallet (Chrome)	
API Request	Wallet addresses
Passphrase	Extractable password
Trezor Wallet (Both)	
JSON id	Device metadata and public keys
xpub	Public keys

consistently extracted with FORESHADOW. A second data structure unique to Firefox also contained the full transaction history in JSON.

Trezor Wallet allows users to set an optional password to add an additional layer of authentication. This password was not found in memory when using Firefox. Unlike Firefox, the cleartext passphrase persisted in predictable locations in Chrome memory even with active use of the application. The passphrase was found to remain uncorrupted in the same location after the client automatically locked itself. As shown in Fig. 6, the average integrity of the passphrase artifacts stayed at 100% at all times except at timestamp 0:13:31 where the integrity dipped to 99%, meaning that most instances of the key would be intact and usable if extracted.

5.3. Trezor Bridge

Not only is the Trezor Bridge responsible for abstracting USB driver interaction from the web browser, but it also logs all USB related events. While Trezor Wallet is running, the Trezor Bridge constantly monitors the list of USB devices currently connected to the system and iterates over them one by one to determine if they are a Trezor device. The logs are located in the user's Application Data folder and contain timestamps of when USB devices were connected to the system, even if they are not Trezor devices. This log file is open while the Trezor Bridge is running so it can be extracted from a memory dump of a running system using the *dumpfiles* Volatility command.

6. Discussion

Returning to the three questions posed in Section 1: Yes, a forensic investigator can use artifacts obtained from volatile memory to identify cryptocurrency hardware wallet use, extract the transaction history, and associate a specific hardware wallet device with a host computer. The techniques presented in this study allow for the deanonymization of transactions and public keys by serializing data structures identified by YARA scans. Although transaction histories can be recovered, their availability in memory cannot always be guaranteed across all platforms. That being said, regular expression based YARA scans were consistently able to find all extended public keys synced with the client, giving a forensic investigator all required information to view all past and future transactions on the blockchain.

Pertaining to Ledger, the persistence of memory artifacts was extremely poor. When the application was locked at the 0:30:57 timestamp in Fig. 5, the majority of memory artifacts were overwritten nearly immediately, and the artifacts which remained in memory were significantly corrupted. Terminating the process at timestamp 37:11 caused the remaining memory artifacts to be overwritten to the point that FORESHADOW could not detect them. This is a potential limiting factor of this tool's effectiveness against the Ledger wallet, as its artifacts would likely not remain in memory at the time when a device was acquired.

The Trezor wallet client produced certain artifacts in memory which persisted for long after its browser tab was killed. As shown at timestamp 0:33:16 in Fig. 6, the extended public keys remained in memory without being corrupted after the termination of the process. Therefore, this data could be feasibly retrieved if a machine was recovered within a reasonable time frame after the Trezor application was closed. The persistence of the Trezor artifacts over the Ledger artifacts is likely due to the former being stored in a memory segment devoted to storage, and the latter being in an area which is frequently used for IPC. The passphrase artifacts were overwritten immediately after the application was terminated.

7. Future work

The data structure extraction methodology presented in this work has several shortcomings. The first of which being that in order to apply it to other applications, it is necessary to go through the same signature discovery process outlined in Section 4.2. The second being that data availability cannot be guaranteed across applications. Future work may focus on creating a runtime environment specific framework for generic data structure extraction. For example, Chrome and Firefox use the V8 and SpiderMonkey JavaScript engines, respectively. A tool that directly interacts with the memory management system of the engine may have generalizable results.

Likewise, since Electron is Chromium at its core, it also leverages the V8 engine. However, the additional layer of abstraction provided by the Electron framework presents an opportunity to extract data unique to Electron applications. For example, robust extraction techniques for the IPC messages identified in this work may be a powerful forensic tool for other popular Electron applications such as Skype, Slack, and Mailspring. Because Electron uses these IPC messages to communicate with a renderer process, such a method may be able to reconstruct the state of the GUI at the time of the dump similar to Ligh (2012) or Saltaformaggio et al. (2015).

While the visualization framework was produced specifically for analysis of the wallet clients studied in this work, it could be used to analyze any application for which artifacts are extracted using our data collection method. To study the effectiveness of both the statistics and the memory space visualization, the framework could be tested across scenarios for multiple different applications. For example, this framework could be tested against both browser-based applications and standalone applications in order to analyze how each application type's artifacts persist in memory, and how the locations of their artifacts differ.

Both FORESHADOW and the visualization framework should be tested against multiple operating systems to prove its effectiveness when the host is not running the tested version of Windows 7. Additionally, this would serve to extract information about potential differences in artifact persistence based on how the operating systems allocate memory.

8. Conclusion

Cryptocurrency will likely remain the payment method of choice for illicit online payment in the foreseeable future and hardware wallets are seeing significant adoption. As such, practitioners must be equipped to collect forensic evidence from the clients used to interact with these devices. FORESHADOW enables the extraction of such evidence from Windows systems.

Additionally, application specific memory forensics relies heavily upon scanning and signature based searches. Our visualization framework serves to determine the availability and integrity of the target artifacts, enabling expedited development of future tools using similar techniques.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1921813. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Ali, S.S., ElAshmawy, A., Shosha, A.F., 2018. Memory forensics methodology for investigating cryptocurrency protocols. In: 'Proceedings of the International

Conference on Security and Management (SAM)', the Steering Committee of the World Congress in Computer Science. Computer, pp. 153–159.
Baum, J.B., 2014. Windows memory forensic data visualization. https://scholar.afit.edu/etd/515/.
BitcoinNews, 2018. Hardware wallet sales booming as nano s tops 1.3 million units. https://bitcoinnews.com/hardware-wallet-sales-booming-as-nano-s-tops-1-3-million-units/.
Block, F., Dewald, A., 2017. Linux memory forensics: dissecting the user space process heap. Digit. Invest. 22, S66–S75. https://doi.org/10.1016/j.diin.2017.06.002.
Blockchain Explorer, 2020. https://www.blockchain.com/explorer.
Case, A., Marziale, L., Richard, G.G., 2010. 'Dynamic recreation of kernel data structures for live forensics', *Digital Investigation* **7**, S32 – S40. In: The Proceedings of the Tenth Annual DFRWS Conference. https://doi.org/10.1016/j.diin.2010.05.005.
Case, A., Richard, G.G., 2016. 'Detecting objective-c malware through memory forensics', *Digital Investigation* **18**, S3 – S10. https://doi.org/10.1016/j.diin.2016.04.017.
Case, A., Richard, G.G., 2017. Memory forensics: the path forward. Digit. Invest. 20, 23–33 (Special Issue on Volatile Memory Analysis).
Casey, P., Lindsay-Decusati, R., Baggili, I., Breitinger, F., 2019. Inception: Virtual Space in Memory Space in Real Space.
Chainalysis, 2019. Crypto Crime Report: Decoding increasingly sophisticated hacks, darknet markets, and scams, Chainalysis. https://uploads-ssl.webflow.com/5a9360f88433cb00018 022c2/5c4f67ee7deb5948e2941fda_Chainalysis%20January%2020 19%20Crypto%20Crime%20Report.pdf.
Conti, M., Sandeep Kumar, E., Lal, C., Ruj, S., 2018. A survey on security and privacy issues of bitcoin. IEEE Commun. Surv. Tutorials 20 (4), 3416–3452.
DFRWS, 2005. Digital Forensics Research Workshop: Forensics Challenge 2005. http://old.dfrws.org/2005/challenge.
Dolan-Gavitt, B., 2008. 'Forensic analysis of the windows registry in memory', *Digital Investigation* **5**, S26 – S32. In: The Proceedings of the Eighth Annual DFRWS Conference.
Ford, M., 2004. Linux memory forensics. https://www.drdobbs.com/linux-memory-forensics/199101801.
Garfinkel, S., Nelson, A.J., Young, J., 2012. 'A general strategy for differential forensic analysis', *digital investigation* **9**, S50 – S59. In: The proceedings of the twelfth annual DFRWS conference. https://doi.org/10.1016/j.diin.2012.05.003.
Ghafarian, A., Wood, C., 2019. Forensics data recovery of skype communication from physical memory. In: Arai, K., Kapoor, S., Bhatia, R. (Eds.), Intelligent Computing. Springer International Publishing, Cham, pp. 995–1009.
Grajeda, C., Sanchez, L., Baggili, I., Clark, D., Breitinger, F., 2018. 'Experience constructing the artifact genome project (agp): managing the domain's knowledge one artifact at a time'. Digit. Invest. 26 https://doi.org/10.1016/j.diin.2018.04.021. S47 – S58.
Gurkok, C., 2015. 'bitcoin.py'. https://github.com/volatilityfoundation/community/blob/b4d65bd01870299c8c08e1e11b7b70dfe96cd1dd/CemGurkok/bitcoi n.py.
Haigh, T., Breitinger, F., Baggili, I., 2019. If I Had a Million Cryptos: Cryptowallet Application Analysis and a Trojan Proof-Of-Concept, pp. 45–65.
Inoue, H., Adelstein, F., Joyce, R.A., 2011. 'Visualization in testing a volatile memory forensic tool', *digital investigation* **8**, S42 – S51. The proceedings of the eleventh annual DFRWS conference. https://doi.org/10.1016/j.diin.2011.05.006.
Ledger, 2019. 'Celebrating 5 years at ledger then & now', *Ledger*. https://www.ledger.com/celebrating-5-years-at-ledger-then-now/.
LedgerHQ, 2019. ledger-live-desktop. https://github.com/LedgerHQ/ledger-live-desktop.
Lee, K., Hwang, H., Kim, K., Noh, B., 2016. Robust bootstrapping memory analysis against anti-forensics. Digit. Invest. 18 https://doi.org/10.1016/j.diin.2016.04.009. S23 – S32.
Ligh, M.H., 2012. Movp 4.2 taking screenshots from memory dumps. https://volatility-labs.blogspot.com/2012/10/movp-43-taking-screenshots-from-memory.html.
Palatinus, M., Rusnak, P., 2014. 'Bip44'. https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki.
Palatinus, M., Rusnak, P., Voisine, A., Bowe, S., 2013. 'Bip39'. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.
Palutke, R., Freiling, F., 2018. 'Styx: countering robust memory acquisition', *digital investigation* **24**, S18 – S28. https://doi.org/10.1016/j.diin.2018.01.004.
Popper, N., 2019. Bitcoin thieves threaten real violence for virtual currencies. https://www.nytimes.com/2018/02/18/technology/virtual-currency-extortion.html.
Saltaformaggio, B., Bhatia, R., Gu, Z., Zhang, X., Xu, D., 2015. Guitar: piecing together android app guis from memory images. In: 'Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security', CCS 15. Association for Computing Machinery, New York, NY, USA, p. 120132. https://doi.org/10.1145/2810103.2813650.
Schatz, B., 2007. Bodysnatcher: towards reliable volatile memory acquisition by software. Digit. Invest. 4, 126–134. https://doi.org/10.1016/j.diin.2007.06.009.
Schlesinger, J., Day, A., 2018. Growing blackmail scam demands payment in bitcoin. https://www.cnbc.com/2018/01/22/growing-blackmail-scam-demands-payment-in-bitcoin.html.
Schuster, A., 2006. 'Searching for processes and threads in microsoft windows memory dumps', *Digital Investigation* **3**, 10 – 16. In: The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06). https://doi.org/10.1016/j.diin.2006.06.010.
Schuster, A., 2008. 'The impact of microsoft windows pool allocation strategies on memory forensics', *Digital Investigation* **5**, S58 – S64. The Proceedings of the Eighth Annual DFRWS Conference. https://doi.org/10.1016/j.diin.2008.05.007.
Sttgen, J., Cohen, M., 2014. 'Robust linux memory acquisition with minimal target impact', *Digital Investigation* **11**, S112 – S119. In: Proceedings of the First Annual DFRWS Europe. https://doi.org/10.1016/j.diin.2014.03.014.
Sun, H., Sun, K., Wang, Y., Jing, J., Jajodia, S., 2014. Trustdump: reliable memory acquisition on smartphones. In: Kutyłowski, M., Vaidya, J. (Eds.), Computer Security - ESORICS 2014. Springer International Publishing, Cham, pp. 202–218.
Sylve, J., Case, A., Marziale, L., Richard, G.G., 2012. Acquisition and analysis of volatile memory from android devices. Digit. Invest. 8 (3), 175–184. https://doi.org/10.1016/j.diin.2011.10.003.
Sylve, J.T., Marziale, V., Richard, G.G., 2016. 'Pool tag quick scanning for windows memory analysis', *Digital Investigation* **16**, S25 – S32. DFRWS 2016 Europe. https://doi.org/10.1016/j.diin.2016.01.005.
Thing, V.L., Ng, K.-Y., Chang, E.-C., 2010. 'Live memory forensics of mobile phones', *Digital Investigation* **7**, S74 – S82. In: The Proceedings of the Tenth Annual DFRWS Conference. https://doi.org/10.1016/j.diin.2010.05.010.
Van Der Horst, L., Choo, K.-K.R., Le-Khac, N.-A., 2017. Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core.
Volatility Foundation, 2018. Volatility. https://github.com/volatilityfoundation/volatility.
Wuille, P., 2012. 'Bip32'. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.
Yang, S.J., Choi, J.H., Kim, K.B., Bhatia, R., Saltaformaggio, B., Xu, D., 2017. Live acquisition of main memory data from android smartphones and smartwatches. Digit. Invest. 23, 50–62.
Zhang, R., Wang, L., Zhang, S., 2009. Windows memory analysis based on kpcr. In: 'Information Assurance and Security, 2009. IAS'09, vol. 2. Fifth International Conference on', pp. 677–680. IEEE.
Zheng, J., Tan, Y., Zhang, X., Liang, C., Zhang, C., Zheng, J., 2017. An anti-forensics method against memory acquiring for android devices. In: IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)', vol. 1, pp. 214–218. https://doi.org/10.1109/CSE-EUC.2017.45.
Zollner, S., Choo, K.-K.R., Le-Khac, N.-A., 2019. An automated live forensic and postmortem analysis tool for bitcoin on windows systems. IEEE Access 7, 158250–158263.
Zyskind, G., Nathan, O., Pentland, A., 2015. Decentralizing privacy: using blockchain to protect personal data. In: IEEE Security and Privacy Workshops', pp. 180–184.