

UNIVERSITÀ DI PISA
Facoltà di Ingegneria

Corso di Laurea Specialistica in
INGEGNERIA INFORMATICA

**Inner-Eye:
Appearance-based Detection
of Computer Scams**

Tesi di Laurea di
Jacopo Corbetta

Relatori:

Prof. Beatrice Lazzerini

Prof. Giovanni Frosini

Prof. Giovanni Vigna

Prof. Christopher Kruegel

Sessione 19 Dicembre 2011

Abstract

Readily-available Internet access has become a defining trait of the present years. As more and more inexperienced users gain computer and Internet access, malware writers, criminals and fraudsters are attempting to annoy or take advantage of them in various ways.

In the past decades, increasingly sophisticated methods have been invented to stealthily gain access to computer systems and equally complex tools have been developed to combat them. In recent years, a new trend has emerged. Instead of using complex penetration techniques, fraudsters can simply exploit the naivety of users to trick them into installing malware, disclosing their personal data, paying for services of dubious or inexistent utility, etcetera. Relatively simple techniques, when applied on a large scale, have the potential to yield significant revenue without requiring technological sophistication.

Ironically, modern antiviruses and security systems are often caught unprepared by these simple confidence tricks and are unable to detect them. Their sophisticated heuristics can detect complex exploitation attempts, but are powerless against code that simply presents messages to the user.

Techniques employed by computer scammers and some common frauds will be presented and analyzed in this work to define and clarify the problem and its significance. A survey of possible approaches to automate the detection these threats will be presented. In particular, I will explore the use of imaging and text-matching techniques to detect typical computer scams such as pharmacy and rogue antivirus frauds.

The prototypical *Inner-Eye* system implements the chosen approach in a scalable and efficient manner through the use of virtualization. *Inner-Eye* is able to perform complex analyses on a large volume of possibly malicious websites and executables. Through the use of Optical Character Recognition and an instrumented web browser, pharmacy scams can be detected by an SVM classifier with high reliability.

Sommario

Negli ultimi anni, la sempre più ampia disponibilità dell'accesso a Internet ha portato molti nuovi utenti a contatto con questa nuova realtà. Per criminali e truffatori, l'inesperienza di chi per la prima volta si affaccia al mondo on-line è un'opportunità da sfruttare a proprio vantaggio.

Per molto tempo, il crimine informatico si è basato sulla ricerca di metodi sempre più complessi e raffinati per ottenere accesso non autorizzato ai sistemi di elaborazione; per combattere questa tendenza, la ricerca sulla sicurezza informatica si è indirizzata sullo sviluppo di sistemi altrettanto sofisticati. Recentemente, tuttavia, è emersa una tendenza completamente nuova. Anziché sfruttare complesse vulnerabilità nel software, è possibile semplicemente convincere persone inesperte a volontariamente fornire i propri dati personali, installare malware, pagare per ricevere servizi di dubbia utilità, eccetera. Anche tecniche molto semplici, se applicate su larga scala, possono fornire un profitto notevole.

Paradossalmente, sono proprio le frodi più semplici a trovare impreparati i moderni antivirus e sistemi di sicurezza. Le raffinate euristiche che impiegano, infatti, sono in grado di rilevare attacchi tecnologicamente molto sofisticati ma sono impotenti contro codice che non fa altro che presentare messaggi all'utente.

In questo lavoro, viene innanzitutto presentato uno studio delle tecniche usate da truffatori e ciarlatani informatici per raggiungere le loro vittime e ottenere profitto. Sono poi proposti alcuni metodi per proteggere gli utenti tramite il rilevamento automatico di queste frodi. In particolare, verrà esplorato l'uso di tecniche basate sull'elaborazione di immagini e testo per affrontare il riconoscimento di due tipi di frode molto comuni: la vendita illegale di farmaci e i finti antivirus.

L'approccio scelto è stato implementato in modo scalabile per costruire un sistema di analisi, *Inner-Eye*. Questo prototipo sfrutta tecniche di virtualizzazione per analizzare pagine web o eseguibili (anche maligni) in modo sicuro ed efficiente, sfruttando un motore di riconoscimento ottico dei caratteri (OCR), un browser appositamente modificato e un classificatore basato su una Support Vector Machine. *Inner-Eye* è stato testato sul riconoscimento delle frodi farmaceutiche e ha confermato la propria affidabilità.

Contents

1	Introduction and motivations	1
1.1	Extremes meet: crime and simplicity	5
1.1.1	Spamming	5
1.1.2	Scams	6
1.1.3	Scareware	8
1.2	How to protect users	11
2	Malware and computer scams	13
2.1	Propagation methods	14
2.1.1	Spam	14
2.1.1.1	Collecting spam messages	19
2.1.2	Browser and OS vulnerabilities	20
2.1.3	Search engine result poisoning	21
2.1.4	Malvertisement	22
2.2	Payloads	23
2.2.1	Fake antivirus programs	27
2.2.2	Other scareware frauds	34
2.2.3	Common scams and frauds	34
3	Detecting frauds	39
3.1	Network-based detection	40
3.1.1	Exploiting previously known information	40
3.1.2	Network properties	42
3.2	Visual similarity	43
3.3	Image analysis	43
3.4	Sensitive image elements	44
3.5	Text-based detection	45
3.6	Sensitive words	46
3.6.1	Possible evasion techniques	46
3.6.2	Choosing the words	48

4	The Inner-Eye system	51
4.1	Overview	51
4.2	Automated submission and training	53
4.2.1	The benign dataset	54
4.2.1.1	Alexa topsites	54
4.2.2	The spam dataset	55
4.2.3	The Wepawet dataset	56
4.3	The capture system	59
4.3.1	Capturing webpages	60
4.3.1.1	Robust handling of character data	63
4.3.2	Capturing Windows executables	65
4.3.2.1	Simulating user interaction	66
4.4	Processing phase	69
4.4.1	Optical Character Recognition	69
4.4.1.1	Mitigating OCR errors	70
4.4.2	Features	71
4.4.2.1	Extracting words	71
4.4.3	Detecting obfuscation	72
4.4.4	Classification	75
4.4.4.1	Support Vector Machines	77
5	Building a scalable system	81
5.1	Latency	82
5.2	Parallelism	82
5.3	Virtualization	83
5.4	Exploiting virtualization	84
5.5	Scalability	86
5.6	Cloud-based workers	86
6	Conclusions	89
6.1	Future work	90
6.1.1	Detecting fake AV webpages	90
6.1.2	Detecting fake AV binaries	90
6.1.3	Detecting graphical elements	91
6.1.4	Improvements to the current heuristic	91
6.1.5	Other improvements	92
	Bibliography	93

List of Figures

1.1	Malware propagation methods as observed by Microsoft	2
1.2	A black market rogue av campaign website	4
1.3	A black market pharmacy campaign website	4
1.4	Percentage of e-mail messages detected as spam by Microsoft Forefront	6
1.5	Spam e-mail example	7
1.6	Example of a website that purports to sell prescription drugs .	9
1.7	Example of rogue antivirus software	10
2.1	Software threat categories according to Microsoft	14
2.2	Cost of offer placement for common advertisement approaches	15
2.3	Type of spam e-mails blocked by Microsoft products over time	16
2.4	E-mail selling e-mail addresses	19
2.5	Email spam types as detected by Microsoft products	26
2.6	Installation procedure for the Braviarx rogue AV family	27
2.7	A rogue antivirus webpage	28
2.8	An original fake antivirus interface	29
2.9	Messages shown in the system tray by Braviarx Fake AVs	29
2.10	Result window shown by Braviarx Fake AVs	29
2.11	Comparison of the interfaces shown by the TotalProtect fake AV upon installation and after registration.	30
2.12	Alternate versions of Zentom System Guard	31
2.13	Dropper page using a fake codec error message.	32
2.14	Source code for the fake codec dropper page	32
2.15	Interface shown by the fake codec binary	33
2.16	FakeHDD and Tritax Privacy Center scareware windows	35
2.17	Website selling counterfeit goods	38
3.1	Message displayed by Firefox for blacklisted sites	41
3.2	Graphical elements common to many rogue antiviruses.	44
3.3	Sensitive word list	49

3.4	Discarded automatically constructed word list	50
4.1	Appearance of the Alexa toolbar	55
4.2	URL obfuscation example in spam messages	56
4.3	The publicly accessible Wepawet submission point.	58
4.4	Webdriver capture example	61
4.5	Unicode encoding examples	64
4.6	Interaction with a fake antivirus setup	68
4.7	Exact word occurrences	73
4.8	N-gram occurrences	74
4.9	Feature categories	76
4.10	Sparse representation of a sample	77
4.11	Grid search for the SVM parameters	79
4.12	Classifier accuracy	80

Chapter 1

Introduction and motivations

At the beginning of its history, automatic computing was a kind of arcane wizardry performed in secluded research laboratories. Over the years, computers have found their way into universities, factories and eventually even private homes.

In the present decade, computing has reached an astonishing degree of pervasiveness; it can be said without hyperbole that virtually all working people in the developed world have access to computing equipment (PCs, laptops, smartphones, embedded devices, etc.). The Internet has provided a medium for interconnecting all these devices, creating a global network that spans through countless language, knowledge and cultural barriers.

In a similar fashion, malicious software (often shortened to *malware*) has constantly been evolving. While the first *viruses* could be likened to electronic vandalism, malware writers have since engaged security researchers in an increasingly sophisticated arms race.

Originally, most computer users were knowledgeable individuals, often programmers themselves. Therefore, significant efforts were made by virus authors to ensure that their creations would remain undetected. New and complicated ways were discovered to infect computers by exploiting Operating System and application bugs and weaknesses.

In the late nineties, a different approach became popular. Instead of attempting to remain invisible and gain privileges exclusively through software

vulnerabilities, malware like Melissa actively tried to get the users' attention and enlist their help against their own systems; tricking the victim into running a malicious attachment to an email message was a popular technique. As the number of inexperienced users with Internet connections rose, this approach became increasingly powerful and successful, leading to very large scale infections and substantial economic damage. Presently, misguided user interaction is by far the leading vector for malware propagation.

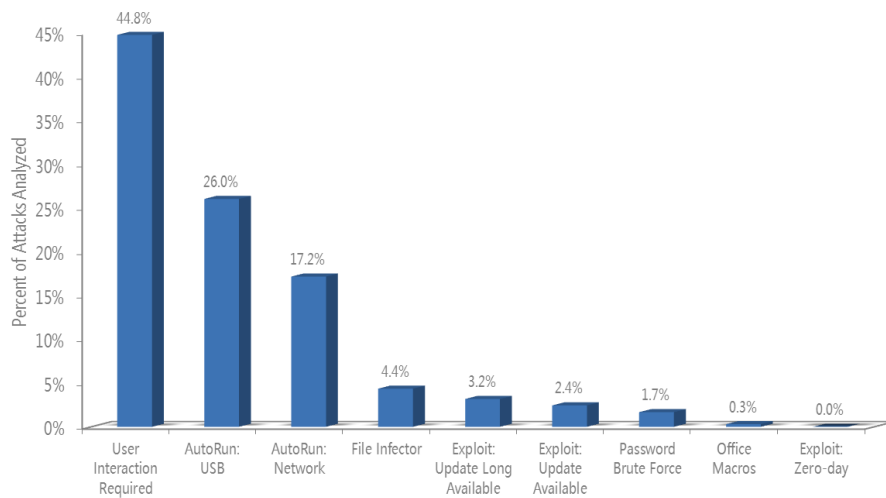


Figure 1.1: *Malware propagation methods as observed by Microsoft in the first quarter of 2011. Notice how intentional user action accounts for about 45% of the infections; user negligence (e.g. unknowingly spreading malware via USB keys, neglecting to update their systems, etc.) contributes another 30%. Microsoft reports that previously unseen (zero-day) exploits accounted for less than 1% of the infections and that none of the top malware families relied on them. [1]*

The objectives of malware writers have also changed significantly. Initially, many of the people researching vulnerabilities and writing exploits were motivated by the desire to master complex techniques and achieve recognition by finding novel and original exploit avenues. People sought the thrill and intellectual challenge involved in overcoming protections and understanding a complex and powerful system, more than the ability to cause damage.

This attitude was especially prevalent among the so-called *phone phreaks*, arguably the grandfathers of computer hackers. Some of them would even put themselves at risk and contact AT&T to notify the company of problems

they had discovered while exploring the phone network. Understandably, not everybody had the same *forma mentis*.

Criminals noticed that computer crime offered the opportunity to generate revenue through what at the time was essentially a “clean” and untraceable channel. In the present day, large numbers of infected computers (so called *botnets*) are used to conduct lucrative spam and extortion campaigns.

Intermediates have also appeared to facilitate computer crime and exploitation: user traffic, exploit techniques, botnet access, “bulletproof” hosting¹ and other illegal services can all be bought and sold on a very active black market.

See Figures 1.3 and 1.2 on the following page for an example of the level of sophistication reached by fraud campaigns. The people who run the campaign provide their affiliates with the tools to monetize page views: rogue antivirus software in the first figure (see Section 1.1.3), prescription drugs sales infrastructure in the second one (see Section 1.1.2). Through various techniques, the affiliates will drive user traffic to the content provided by the campaign masters and will receive a share of the profits made. At any time, affiliates can check their current account balance, request technical support through instant messaging, monitor their progress over time, etcetera.

¹Hosting services specifically designed to host malware or illegal services. Their owners purport to be able to resist law enforcement demands to remove the content or identify the authors.

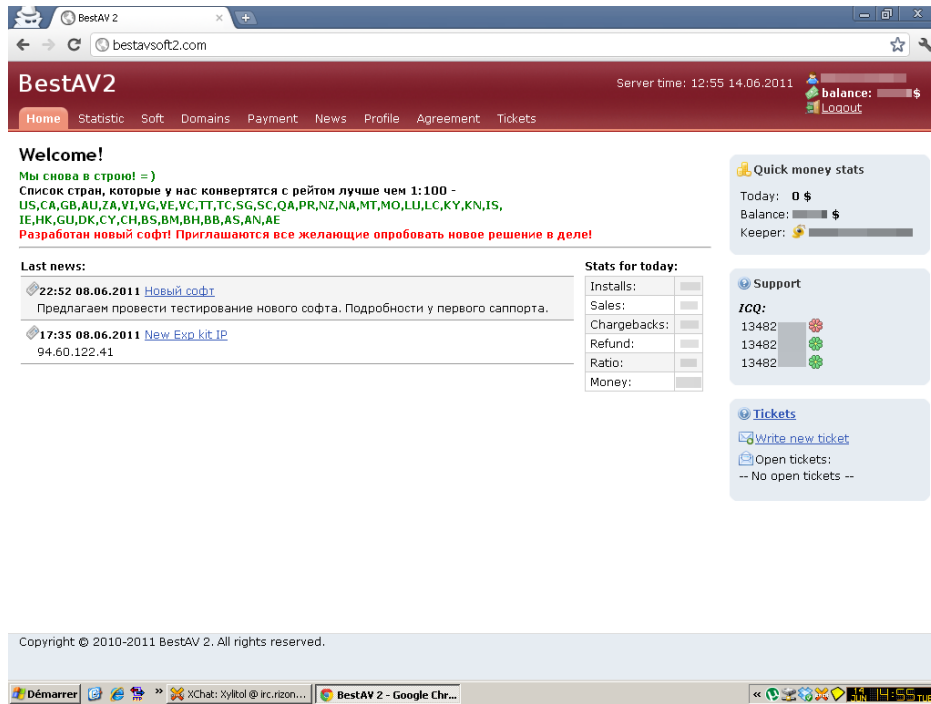


Figure 1.2: *Affiliate control panel for a rogue antivirus software campaign (BestAV). [34]*

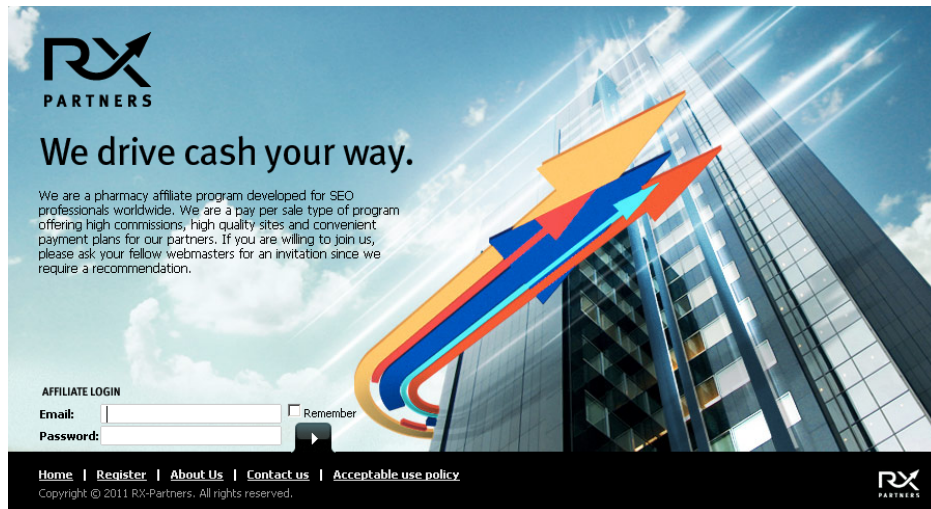


Figure 1.3: *Home page and advertisement for affiliates for a pharmacy campaign. To give an appearance of legitimacy to the website, standard-looking “About us”, “Contact” and “Acceptable use policy” pages are provided. [34]*

1.1 Extremes meet: crime and simplicity

Over the last few years, an interesting development has been observed, in many ways a natural consequence of the trends we discussed in the previous paragraphs. With billions of unsuspecting users potentially available for exploitation, the simplest approaches are often adequate to secure revenue.

Criminals and fraudsters have little interest for the novelty of a technique or for the level of sophistication needed to carry it out; relatively simple methods such as spamvertising (advertising products, often counterfeit, using spam) or harvesting credit card data through keyloggers are profitable enough for them.

In the next few paragraphs we will introduce some of the techniques employed by fraudsters and explain how they rely on user error to be effective despite their simplicity.

1.1.1 Spamming

The best example of how simple techniques can net a profit to fraudsters has actually been around for decades: e-mail *spam*. Spam e-mail messages are a very cheap way to contact millions of unsuspecting users. Of course most of the intended recipients will not interact with the content (some will recognize the message as untrustworthy, for example, and many won't even see the message because of automated spam filtering), but sending the messages is so cheap that even a small percentage of responses will allow the campaign to break even.

As we have seen, the sheer number of reached users can be the decisive factor for the success of a campaign. Increasing the number of e-mails sent is the easiest solution to this problem and to this day the majority of e-mail messages carried over the Internet is spam (Figure 1.4).

Of course the general technique is not limited to e-mail messages. Every communication medium where a low-cost action allows reaching a good number of users will find itself targeted by spammers.

Indeed, e-mail spam might actually be considered inefficient in this respect because each recipient has to be listed explicitly: a single modification

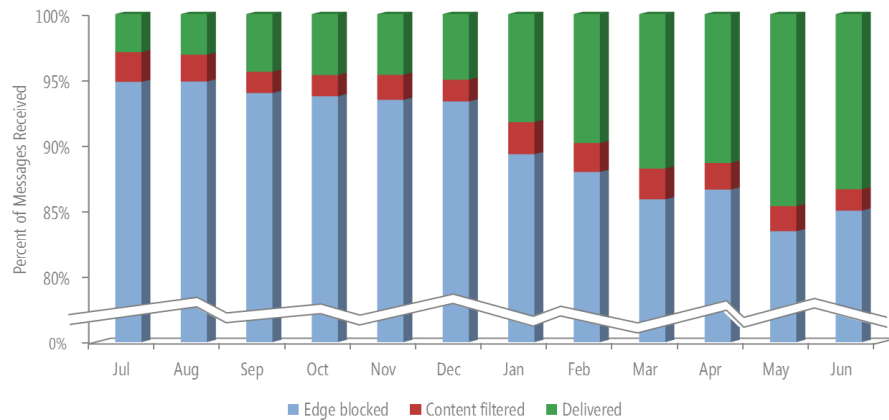


Figure 1.4: *Percentage of e-mail messages blocked by Microsoft’s antispam solution for corporate Microsoft Exchange installations (Forefront Online Protection for Exchange). The decline in blocked messages is due to recent takedowns against two major botnets (namely Cutwail and Rustock). [1]*

to a publicly accessible web page can allow the spammer to reach thousands of users. User-editable pages like discussion forums, wikis and blog comments are targeted by spammers for this reason.

1.1.2 Scams

In itself, spamming is just a technique used to reach people: for the campaign to be profitable, at least a percentage of the messages sent has to generate revenue.

A possible way to monetize spam is to use it to spread malware, e.g. keyloggers. Data collected by the keylogger (credit card numbers, bank account information, ...) can in turn generate a profit. Chapter 2 will present other examples of how spam can indirectly generate revenue, but there are also more direct ways.

For example, spam messages can simply ask the user for money. Various confidence tricks have been perfected over the years to lure the user into giving money to the scammer, ranging from the promise of future revenue to simple pleas for charitable donations.

Offering to sell goods to the user is another very popular way to monetize

```
X-Envelope-From: <arlynelonnie@hofstra.edu>
Delivered-To: mhvhsvph@dauber.sonic.net
Delivered-To: reyesj@dauber.sonic.net
Delivered-To: izfxuoly@dauber.sonic.net
Delivered-To: ueqjlef0@dauber.sonic.net
Delivered-To: holtn@dauber.sonic.net
To: <holtn@dauber.sonic.net>
Subject: FREE bonus pills, FREE shipping on
        orders over $200, wide range of packages.
        Buy CIALIS Now!
From: "DianaLia" <arlynelonnie@hofstra.edu>

Buy CIALIS Now From $1.53 & Get 12 bonus pills FREE!
Purchase CIALIS from $1.53 per pill from any of the
listed reputable online pharmacies. FREE bonus pills,
FREE shipping on orders over $200,
wide range of packages ...
http://rxdrugsion.ru
```

Figure 1.5: *Example of a spam e-mail message, including some of the message headers. Notice the automatically generated sender and list of recipients. The website advertised here is shown in Figure 1.6 on page 9.*

spam. The spam message will simply contain advertisement and a link to a legitimate-looking e-commerce website where the user can voluntarily shop for goods at very low prices. The advertised merchandise often comprises privacy-sensitive or not widely available articles such as prescription drugs.

Section 2.1.1 will present the phenomenon in detail, but it is worth mentioning that these scams do not fundamentally depend on computer technology. The spam message and the linked websites do not generally contain malicious elements: the user naivety and confidence are being exploited, not their computers.

1.1.3 Scareware

Another method employed by criminals is to create simple web pages and programs that trick the users into believing their computer has a problem (ironically, often a malware infection).

The user may be presented with windows designed to look like those of a legitimate antivirus scanning the files on the victim's PC. The scan results window will list several security or reliability problems and will attempt to convince the user that her data is in great danger. To further the effect, the program may pop up fake alert dialogs every few minutes or, in more sophisticated programs, prevent the user from starting any application until action is taken.

The fraudsters then demand payment from the user to solve the (non-existing) issue. This category of malware is generally referred to as *scareware* and has proven to be extremely lucrative for criminals: it is estimated that three large scareware campaigns have generated revenue for over \$ 130 million dollars. [35]

Once again, while many scareware families still rely on “traditional” exploits to infect the user machine, others avoid these altogether. As security software gains more and more detection abilities, it may very well be the case that successful malware will have to be essentially harmless and rely entirely on psychology to generate revenue.

In Section 2.2.1 we will review in detail some of the common scareware families, how they present themselves to the user and how fraudsters attempt to monetize them.

#1 Online Pharmacy
Brand & Generic Medications

Welcome To Our On-line Drug Store
Save Up To 90% OFF Retail Prices
Your Health Is Our Main Concern

✓ FDA Approved Medications
✓ Lowest Prices In The Market
✓ Discreet Shipping
✓ Guaranteed Anonymity

✓ High Quality Medications
✓ 100% Guarantee On Delivery
✓ Guaranteed Satisfaction
✓ 24/7 Customer Support

Customer Support
ONLINE
Feel Free To Contact Us!

Bestsellers All products How to order Shipping policy Order Status Refer a friend Testimonials F.A.Q. About Us

Search products by name
SWITCH CURRENCY: USD / EUR
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Your cart: \$0.00 (0 items)
Proceed to Checkout >

ERECTION PACKS
 + VP Pack
 + Viagra Pack
 + Cialis Pack
 + Professional Pack
 + Super Active Pack
 + Soft Pack
 View all products

MEN'S HEALTH
 + Viagra+Cialis
 + ED Trial Pack
 + Viagra
 + Cialis
 + Levitra
 + Viagra Super Active
 + Cialis Super Active
 + Viagra Soft Tabs
 + Cialis Soft Tabs
 + Viagra professional
 + Cialis Professional
 + Super Viagra
 View all products

WOMEN'S HEALTH
 + Women Viagra
 + Diflucan
 View all products

ANTIBIOTICS
 + Zithromax
 + Amoxil
 + Cipro (Ciprofloxacin)
 + Tetracycline
 View all products

ANTI-ANXIETY
 + Atarax
 + Buspar
 + Cymbalta
 View all products

ANTI-DEPRESSANT
 + Trazodone
 + Lexapro
 View all products

ANTI-DIABETIC
 + Glucophage
 + Actos
 View all products

ALLERGY / ASTHMA
 + Phenergan
 + Allegra
 + Claritin
 View all products

BLOOD PRESSURE
 + Hyzaar
 + Inderal
 + Lopressor
 View all products

GASTROINTESTINAL
 + Nexium
 + Prevacid
 View all products

HEART & CHOLESTEROL
 + Lipitor
 + Norvasc
 + Plavix (Clopidogrel)
 View all products

PAIN RELIEF
 + Prednisolone
 + Indocin
 + Motrin
 + Celebrex
 View all products

WEIGHT LOSS
 + Xenical
 View all products

VIAGRA + Cialis
 10 pills x 100 mg 10 pills x 20 mg
 \$64.95
 +4 Free Viagra

Today's Bestsellers

Viagra Price: \$0.90 Viagra (Sildenafil) is an oral medicine used for treating male impotence (e.g., erectile dysfunction). Add to cart	Cialis Price: \$1.40 Cialis (Tadalafil) is an oral drug, used for treating male impotence, also known as erectile men's erectile dysfunction. Add to cart
Viagra professional Price: \$3.00 Viagra Professional is a new generation extra-strength prescription medicine that is taken orally for the treatment of erectile dysfunction only in men. Add to cart	Cialis Professional Price: \$3.40 Cialis Professional is newly formulated and chemically improved prescription medicine has proved to be useful for multipled orgasm and increased penis size. Add to cart
Viagra Super Active Price: \$2.15 Viagra Super Active is a new and unique formulation of a well known medicine for treatment of ED. It starts acting faster and the effect lasts upto 50 hours. Add to cart	Cialis Super Active Price: \$2.70 Cialis Super Active is a new and unique formulation of a well known medicine for treatment of ED. It starts acting faster and the effect lasts upto 50 hours. Add to cart
Viagra Soft Tabs Price: \$1.55 Viagra Soft Tabs are used to treat impotence in men by increasing the body's ability to achieve and maintain an erection during sexual stimulation. Add to cart	Cialis Soft Tabs Price: \$2.15 Cialis Soft Tabs are quick-dissolving tabs, used to treat male impotence. Add to cart
Amoxil Price: \$0.50 Brand Amoxil (Amoxicillin) is used to treat infections. Add to cart	Zithromax Price: \$0.95 Zithromax (Azithromycin) is used for treating mild to moderate infections caused by certain bacteria. Add to cart
Viagra+Cialis Price: \$2.75 Viagra is an oral medicine used for treating male impotence. Cialis is an oral drug, used for treating male impotence. Add to cart	ED Trial Pack Price: \$2.50 ED Trial Pack includes pills of Viagra, Cialis, Levitra, Viagra Soft Tabs or Cialis Soft Tabs. These drugs are used for treatment of ED in men. Add to cart
Levitra Price: \$2.40 Levitra (Vardenafil) is an oral therapy for the treatment of male impotence. Having the long-lasting effect of 4 hours, and the start time of 15 min. Add to cart	Women Viagra Price: \$1.40 Female Viagra (Sildenafil) is scientifically formulated to provide intense sexual satisfaction for women seeking ultimate pleasure. Add to cart
Atarax Price: \$0.38 Atarax depresses activity in the central nervous system (brain and spinal cord), which causes relaxation and relief from anxiety. Add to cart	Cipro (Ciprofloxacin) Price: \$0.40 Cipro (ciprofloxacin) is fluoroquinolone antibiotic used to treat bacterial infections. Add to cart
Propecia Price: \$0.75 Propecia (Finasteride) is used to treat male pattern hair loss. Add to cart	Nexium Price: \$0.65 Esomeprazole (Nexium) is used for treating gastroesophageal reflux disease (GERD) in patients with a history of irritation and swelling of the esophagus when medicine cannot be taken by mouth. Add to cart
Glucophage Price: \$0.30 Glucophage (Metformin) is used for treating type 2 diabetes. Add to cart	Xenical Price: \$2.10 Xenical blocks absorption of dietary fat into the bloodstream, thereby reducing the number of calories you get from a meal. Add to cart
Lexapro Price: \$0.85 Lexapro (Escitalopram) is used for treating depression and anxiety. Add to cart	Lipitor Price: \$0.40 Lipitor is a cholesterol-lowering drug. Add to cart

WE ACCEPT: VISA, MasterCard, American Express, Discover, E-MONEY

1024 BIT SECURE PAYMENT PROCESSING
Online Pharm Stock LTD, Belize City, Belize
Certified Pharmacy Supplier Since 09/01/2007
100% HACKER-FREE SITE
1024 BIT Encryption

© Copyright rxdrgslon.ru, 2007-2011. All Rights Reserved.
About us :: Refer a friend :: Shipping policy :: Order status :: Privacy policy :: Unsubscribe :: Report spam :: Contact us

Figure 1.6: The website advertised by the spam message in figure 1.5 on page 7. The site purports to sell drugs without requiring a prescription from a doctor.

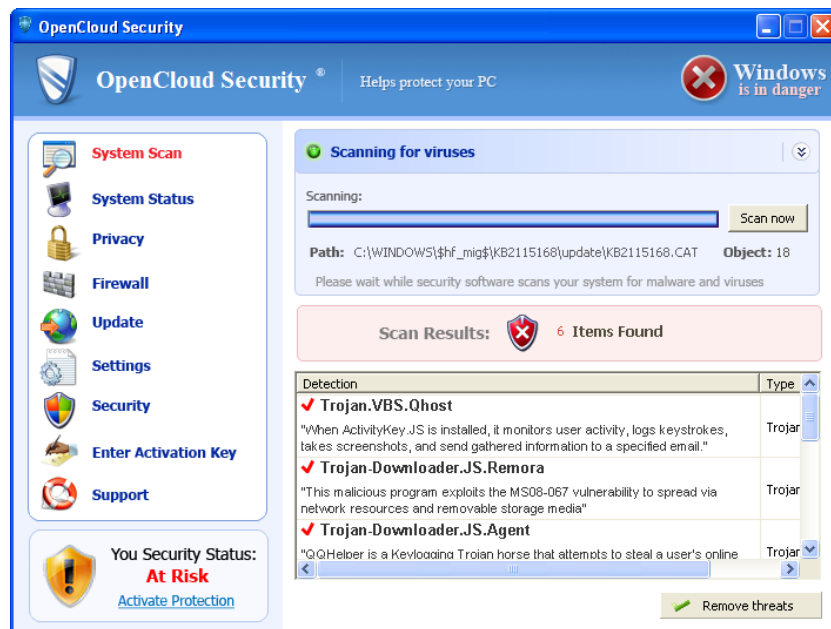


Figure 1.7: *The scan dialog show to the user by the OpenCloud fake antivirus scam.*

1.2 How to protect users

As malware evolved into more and more complicated forms, antivirus and security tools have followed suit by evolving the capability to detect sophisticated attacks on system and application integrity. Today, antivirus products include complex heuristics that detect unusual patterns of interaction with the operating system, attempt to defeat obfuscation via dynamic binary analysis, install filesystem hooks to monitor all files read or written, etc.

However, as we have seen in the previous paragraphs, scams can be very simple and yet maintain effectiveness. Modern security tools are often powerless against confidence tricks, since the infection and stealthiness mechanics they so cleverly detect are simply not present.

Indeed, the traditional approach to computer security is fundamentally inadequate when dealing with the type of scams we have described. Traditional software vulnerabilities are caused by programming mistakes or misguided architectural decisions. Scholars and practitioners have focused for years on developing methods to automatically detect vulnerable code, programming techniques that would reduce the potential to introduce security-sensitive bugs and systems to detect and prevent exploitation.

For the type of frauds discussed here to work, no exploits are needed and no software security patch can correct the problem. A completely new approach is needed to detect them and protect users. Since they work by exploiting the user's confidence, an automatic detection system needs to "see" the content with the same eyes as a user and possibly interact with it in the same way. This will be the subject of this work. In particular, it will focus on *rogue antivirus* and *pharmacy* scams, two very common and representative examples.

Chapter 2 further defines the problem that is being tackled by reviewing common scams, their propagation methods and how they generate revenue while Chapter 3 will present the challenges that arise when attempting to automatically detect them and possible solutions. Chapter 4 describes the ideas behind the *Inner-Eye* system, while Chapter 5 will detail how the use of virtualization techniques allows building a scalable analysis system.

Chapter 2

Malware and computer scams

Computer programs come in many different forms and have many different purposes; naturally, malicious or annoying programs exhibit the same diversity [13]. Nowadays, *malware* is used as an encompassing term for abusive software, while terms like *virus*, *worm* and *trojan* indicate categories of malware.

Many security software vendors distinguish malware and “potentially unwanted software,” with the latter category usually comprising adware, hacking tools, remote surveillance software, etc. (these are applications that may see some legitimate use, but are more often than not undesirable on a machine).

While malicious software is a powerful tool in the hands of criminals, less sophisticated threats also pose significant dangers for users. Scammers use a variety of methods to confuse, defraud or harass victims and make a profit using confidence tricks and exploiting lack of knowledge or familiarity with computers.

In this work, both “pure”, non-software, scams and more traditional malware will be analyzed together in an effort to develop a more comprehensive view of the problem. The term “scam” will be used in an inclusive way to encompass all kinds of frauds, whether they involve binary executables, web pages or non-software elements such as text e-mail messages. Occasionally, the term “malware” will also be used to indicate all kinds of abusive content.

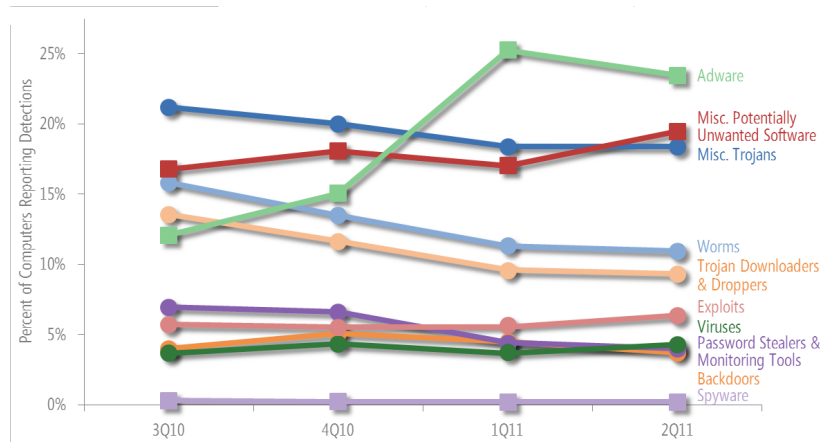


Figure 2.1: Software threat categories according to Microsoft, with recent history (round markers indicate malware categories; square markers indicate potentially unwanted software categories). [1]

As much as possible, this chapter will present generic *payloads* (the component that generates a profit for the criminal) delivered through equally generic *propagation methods*. While not all combinations are possible, this approach eliminates repetition and focuses on the fundamental nature of each problem (mitigating the impact of a certain payload, for example, or the economic efficiency of a propagation method) without drawing attention to irrelevant details.

2.1 Propagation methods

In this chapter we will first review how cyber-criminals reach their victims. By focusing on how abusive content is delivered to users it is possible to develop methods that stop malware before it has any chance to interact with people. This is particularly important in the case of confidence scams, since they are specifically developed to deceive and confuse users.

2.1.1 Spam

Spam is probably the most visible and well-known propagation vector for all sorts of scams and malware. The traditional definition of spamming, devel-

oped in the context of dealing with e-mail spam, is the *sending of unsolicited bulk messages*.

Fundamentally, spamming is an economic problem. Whenever a low-cost action allows delivering a payload to a good number of users, we can be sure that spammers will try to take advantage of the system. All systems that have this property are subject to spam. E-mail messaging is a prominent example: sending e-mails has negligible cost; moreover, messages can have multiple recipients, therefore the spammer can easily craft a single e-mail message with a thousand recipients and let the mailing server do the replication for him.

	Total cost	Recipients	Cost per recipient
Direct mail	\$ 9,700	7,000	\$1.39
Telemarketing	\$ 160	240	\$0.66
Print - targeted	\$ 7,500	100,000	\$0.075
Print - general	\$ 30,000	442,000	\$0.067
Fax	\$ 30	600	\$0.05
Online ads	\$ 35	1,000	\$0.035
Spam	\$ 250	500,000	\$0.0005

Figure 2.2: *Typical cost of offer placement for common advertisement approaches. Source: [10].*

Payloads carried by spam vary: some are malicious webpages or programs, some are simply links to questionable websites (or even to relatively benign websites which try to over-aggressively promote themselves), some are directly attached to the message, some appear in the form of links, etc.

E-mail spam is perhaps the most studied phenomenon; system administrators and common users have been fighting it for decades. Network-based countermeasures include:

Reverse DNS checks

Most SMTP servers will refuse to relay e-mails from a server that doesn't have a valid reverse DNS record. It is assumed that a legitimate mail server will always have one, therefore false positives are not an issue. Most ISPs, however, do assign automatically generated

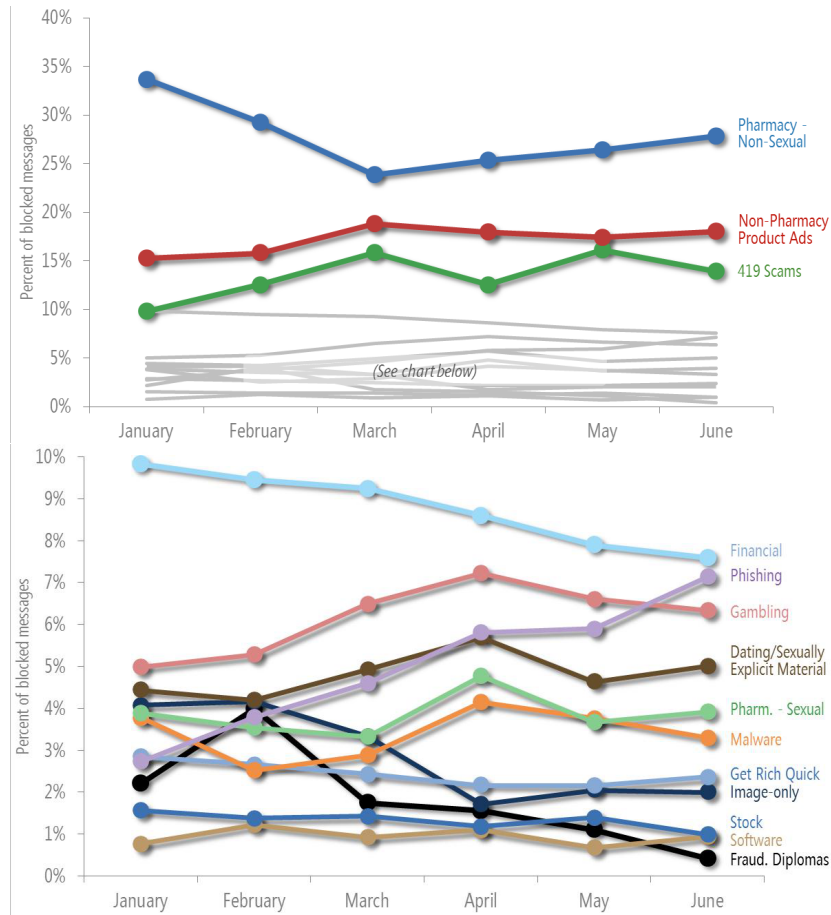


Figure 2.3: *Type of spam e-mails blocked by Microsoft products over time. [1] Notice how a massive but short-lived spam campaign advertising fake diplomas was launched in February causing a visible spike in detections; Microsoft reports that such spikes were often seen in the past but have been rare in 2001, as current spam campaigns tend to run over a longer period of time and show up as more gradual increases and decreases in detection.*

reverse DNS names to all their customers' IPs, so this countermeasure is less effective than it could be.

Graylisting

The practice of always refusing the first delivery attempt for an e-mail message, with the assumption that legitimate protocol-abiding clients will retry the delivery, while spamming bots will just give up.

DNS-based blacklists

Organizations such as SpamHaus maintain extensive blacklists of IP addresses and domains known to be involved in spam distribution. They can usually be queried over DNS and benefit from this protocol's distributed caching. Blacklists are effective in containing spam campaigns run from a handful of IP addresses.

Blacklists were one of the factors that pushed spammers to build (or purchase) extensive *botnets*: networks of compromised computers that span numerous networks and countries. Nowadays botnets often host both spam sending bots and the payload for the campaign. Since security practitioners are extending blacklisting to the so-called *Command & Control* servers that coordinate the campaigns, botnets can contain highly volatile proxy nodes that are used to contact the C&C servers indirectly. [37, 17]

Reputation-based approaches

Mail servers can keep track of the ongoing reputation and habits of known mail servers and quickly exclude those that appear malicious or compromised. This practice can be ineffective if spam is sent directly from the relatively “clean” IP addresses of newly compromised computers, but is still helpful to ensure that vulnerable SMTP servers cannot do much harm.

An interesting variation of this technique is used by Google's email provider Gmail. A rich web interface is exposed to users, who can manually mark (or unmark) email messages as spam. The system automatically learns from the users what messages constitute spam and which are legitimate.

SMTP blocking

Many residential Internet Service Providers block incoming connections to port 25 by default. This makes it somewhat harder to run a mail relay from a botnet. This measure cannot restrict sending e-mail, though,

and since customers legitimately use a variety of SMTP servers to handle their e-mails, filtering outgoing connections is usually infeasible.

These countermeasures attempt to stop the spam e-mails before they are even sent. In Chapter 3 we will see some techniques that examine the content of the message and try to detect spam *a posteriori*.

Obtaining lists of valid e-mail addresses is relatively cheap, as shown in Figure 2.4. Alternatively, spammers can automatically generate addresses or crawl the web to look for addresses embedded in pages. In response, web-page authors tend to obfuscate addresses in various ways; techniques include simple human-readable instructions to amend the wrongly written address, CAPTCHA-based¹ solutions and “spider traps” that try to lure automatic crawlers into following a series of meaningless automatically-generated links and pages (in the hope that the bot will give up crawling the site before getting to the sensitive content).

E-mail spam is only one manifestation of the general phenomenon. Traditional mail spamming and telephone spamming have existed for a very long time, although necessarily on a smaller scale due to the intrinsically higher cost of contacting people through these methods. Legislation has been generally slow to pick up the problem and generally only mandates opt-out procedures.

Other common targets for spam are user-editable websites such as wikis, forums and blogs. Spammers register users and attempt to show links in every possible way: creating new pages full of fake content (and links), populating the user list with dormant users (that will show a link to the website in their profile), adding posts to existing discussions or blog articles (sometimes repeating part of the content in order to make the comment seem legitimate), etcetera. Even Wikipedia, which has a very active anti-vandalism team, occasionally sees link spamming. [41]

Social networks are also emerging as a spam vector. The very personal nature of the exchanges on those websites makes them very attractive to

¹CAPTCHAs are distorted images containing text. In well-constructed CAPTCHAs, the text is easily readable for humans but not for computer programs.

```
ISA - Email Database of EUROPE
TOTAL = 3.038.732 E-MAILS
Spain (1.207.266)
France (211.914)
Italy (1.556.313)
Portugal (63.239)

EUR 120
CLICK HERE TO KNOW HOT TO PURCHASE
We accept Paypal, MoneyGram and Western Union

If you have received this email by mistake
or you don't want to receive anymore newsletters from us,
please click here
```

Figure 2.4: *This e-mail purports to sell lists of e-mail addresses; recipients are filtered for nationality. The spammer uses a handful of different domains and apparently lists his own real name and address in Brazil in order to receive money orders. Interestingly, he advertises his e-mail lists through spam itself.*

spam bots, who can pose as real people and entice users to click on malicious or questionable links with high success. The success rate is even higher when spammers are able to compromise legitimate user accounts: the messages will appear to come from trusted friends of the victim. While social networking websites like Twitter and Facebook are starting to implement link filtering countermeasures, large-scale infections that spread through social networks have already been observed (the *Koobface* botnet is one of the best examples).

2.1.1.1 Collecting spam messages

As we have seen, spam messages may be hard to filter but, obviously, are easy to collect.

A good way to do so is to create a fake mailbox with a guessable name, preferably at an established email provider domain. Every email received by

the fake user can be considered spam with a high probability. Such techniques (setting up apparently innocuous infrastructure that would only catch spam and malicious activity) are employed very often by security researchers and are referred to as *honeypots*.

Alternatively, the address of the honeypot can be distributed on-line, either by putting it on pages reachable by crawlers (but that would not be considered by a human being) or by adding it to email distribution services known to resell their lists to spammers.

The advantage of honeypots is that they will catch all types of spam, including types that have not yet been seen by researchers. They are therefore very useful for research purposes. If volume is more important than novelty, it could also be possible to tap into the filtering systems employed by major email providers: only messages that are detected by existing rules will be detected, but very large quantities can be received.

2.1.2 Browser and OS vulnerabilities

Many traditional computer viruses and worms focus on self-replication [13]. From a single infected computer, they try to infect as many others as possible, and devote a significant part of their code to do so. Methods vary from infecting removable media to remotely exploiting vulnerabilities over the network. Modern malware like *Stuxnet*² combines many different approaches to maximize its chances of success.

In the trend towards simplicity and cost-efficiency, replication capabilities have been neglected by fraudsters. Rogue antiviruses rarely include self-replication capabilities [31]. Centralized distribution through browser vulnerabilities is far more common.

Modern browsers are complex environments: they include JavaScript Just-In-Time compilers, interpreters, many different plugins, ActiveX con-

²A very complex worm that uses multiple infection vectors (including several previously unknown vulnerabilities) and sits dormant on the user's machine. The payload activates only on computers connected to certain Siemens industrial control systems; once it reaches them, it uses rootkit techniques to stealthily infect and reprogram the controllers. The malware is supposedly an attempt to sabotage the Iranian nuclear program by other nation states.

trols, etcetera. As the attack surface is very wide, vulnerabilities have been commonplace. Notably, many attacks do not target the browser itself (which may have sandboxing capabilities), but common plugins such as Java and Adobe Flash: these plugins allow (by design) wider access to the user's computer than the browser normally does (access to the local filesystem, for instance) and can be harder to sandbox effectively. Exploits against plugins also have a good chance to work in many different browsers.

Drive-by downloads and *drive-by installs* are two common results of these vulnerabilities: as soon as the user visits an infected page, an executable file is automatically downloaded to her computer. In some cases, it is possible to automatically launch the downloaded files, making the exploitation process completely automatic; otherwise, the malware author has to rely on social engineering techniques to convince the user to run the binary.

Exploiting browser vulnerabilities, however, always requires the victim to visit a webpage (partly) under the control of the attacker. Spamming (especially on social networks) can expose the link to many users and net a certain number of visits. Compromising legitimate websites to inject the malicious content is also possible. Other techniques commonly used to promote scams are described in the following paragraphs.

2.1.3 Search engine result poisoning

Like everything on the Internet, malicious content is heavily promoted through search engines. While legitimate content can aim to establish an on-line presence and organically receive links over time, pages set up by fraudsters have no useful content and cannot hope to get legitimate back links. In order to evade blacklisting, malware authors frequently change their domain names, use compromised hosts and generally attempt to remain a moving target. These practices all go against the basic principles of legitimate Search Engine Optimization (SEO).

The term *Search Engine Optimization* refers to a variety of techniques used to improve the ranking of websites on popular search engines such as Google and Bing. While many are absolutely legitimate (and often benefi-

cial even to human users), “blackhat” SEO practitioners aim to artificially inflate the ranking of websites. Their techniques include building networks of interconnected sites in order to provide backlinks (the number incoming links is one of the most important factors considered by Google’s *PageRank* algorithm), stuffing pages with popular keywords in the hope to appear in more search result pages, including text invisible to the user (but visible to automatic crawlers), . . .

A recent phenomenon is the exploitation of trending topics. As soon as certain words or phrases are sensed as having importance (e.g. because they are being used a lot on Twitter), scammers will create dozens of websites that purport to be about the currently trending topic. Search engines keep track of which searches are becoming more common and, to provide users with relevant content as soon as possible, quickly add pages to their index in prominent positions. The intrinsic volatility of trending words is a perfect match with the desire of scammers to quickly create and destroy domains and websites. Fake AV authors are reportedly very effective in exploiting trending topics to spread their content. [31]

Malware distributed through fraudulent Google search results has access to some specific obfuscation opportunities: for instance, webpages can check the `Referer` header to confirm that the user arrived on the malicious page by clicking on a Google result page³. The web server can also serve innocuous content to the Google bot, while serving the malicious payload only to users, using IP address, browser version and installed plugins as discriminating factors. [35]

2.1.4 Malvertisement

Compromising a popular website would be a perfect way to spread malware through drive-by install vulnerabilities: website owners know that, and successful sites are generally secure and constantly monitored.

³This particular technique can be expected to lose applicability as Google is currently switching to serve result pages over HTTPS. Browsers generally don’t set the `Referer` header when navigating from a secure page to a non-secure one.

However, a secure website may include content from dozens of other providers: advertisement networks, visitor tracking services, content delivery networks, user-generated content on other websites such as Youtube, social networking features, etcetera. Advertising providers are a particularly attractive target for criminals, since their content runs on multiple websites and they routinely operate on content provided by others.

Reputable advertising networks perform some vetting of the content provided to them (which is often in the form of Adobe Flash banners), but checks have often proven insufficient. Since the flash file is delivered directly to the user's browser, it can be possible for malware authors to write malicious code that will appear innocuous when run inside the ad network's analyzer and trigger only when run on the victim's browser.

The malicious ad can directly use a Flash vulnerability to cause a drive-by install or perform fingerprinting of the browser and redirect the user to a page that will exploit a specific vulnerability (or appear innocuous in case no known vulnerability is noticed: since a vulnerable environment is needed to see the malicious code, analyzing these threats is very complex).

Fake antiviruses are often spread through these ads, and the practice has been christened *malvertisement* (or *malvertizing*). Even popular websites such as the music streaming service Spotify have been used to spread malware with this trick. [4]

2.2 Payloads

As we have seen in the previous section, fraudsters and criminals use a variety of methods to reach users, obfuscate their intent and leverage compromised or ill-designed third-party systems to better propagate their content. Whether the message is targeted to users or to other automated systems, part of it will be designed to carry out the sender's intent: this part is referred to as *payload*.

The payload can take many forms: if the message that carries it was an e-mail message, it will usually be an attachment or a link in the message

body; in a binary, it will be the code section that carries out the malicious action.

The following sections will describe the type of payloads that are targeted in this work. Other common payloads include:

Phishing attempts

This type of payload is often seen in the form of email messages or websites. The scammer purports to be a trusted entity (e.g. the user's bank or credit card provider) and attempts to get the user's credentials for the real entity. Users are presented with fake login pages designed to look like the real website⁴. Obtained credentials can then be sold to other criminals or used to withdraw funds from the compromised account.

Defenses include the blacklisting of known phishing domains (as implemented, for example, by the Google Safebrowsing API [22] included in the Mozilla Firefox and Google Chrome browsers), the personalization of login pages to include user-specified elements (which would make it harder for the scammers to create believable login pages) and educating the users on the risks of entering confidential information on pages visited from untrusted links in emails.

Unsolicited financial advice

Perhaps one of the most interesting applications of e-mail spam is the diffusion of so-called *stock spam*. Speculators write messages that purport to have insider information or make credible market forecasts for certain companies. Messages can advise recipients to buy stocks of a certain company; stock spam has been observed to have a real impact on financial markets. [10]

Malicious binaries

These payloads are often delivered through browser vulnerabilities which

⁴Sometimes the fake website directs the browser to load images and other page assets directly from the real website. Banks have started to implement HTTP Referer checking to detect usage of their images on phishing websites.

allow both downloading and automatically executing the binary. It is not uncommon, however, for webpages to simply ask the user to download and execute the binary: scareware authors sometimes use this method. Attachments to e-mails were once a common way for worms to spread (Loveletter, for example, would send a copy of itself to people in the victim's address book), but nowadays most mail servers scan emails for known viruses or plainly reject executable attachments⁵.

Countless examples of abusive programs exist. Several examples have been mentioned in this work. Notable threats include programs designed to steal sensitive information from the user (e.g. by logging keystrokes), disruptive viruses that corrupt or delete data, remote control software that allows criminals to use the compromised devices for malicious operations, . . .

Links to artificially inflate website ranking

Spam on publicly visible webpages (such as forums and wikis) may not be targeted to users. It may instead be aimed at the spiders employed by search engines. Search engines often use the number of links pointing to a website as a signal of its importance and trustworthiness: by artificially inflating the number of links, blackhat SEO (Search Engine Optimization) practitioners believe they can improve the ranking of their clients' websites.

Several defenses have been developed: for example, wikis and forum can require new users to solve a CAPTCHA before they can post links. Search engines also allow marking certain links as "untrusted" by adding the `rel=nofollow` attribute to the anchor tag. All external links on Wikipedia, for example, are generated with this tag in order to make the encyclopedia a less attractive target for spammers.

Links to questionable websites

In some cases, all the payload author wants to do is to get visibility for his (or his client's) website. As we have seen in the previous section,

⁵Gmail, Google's mail service, is a well-known example.

spam is a relatively efficient way to market goods by exploiting the “long tail” of demand: it is impossible to advertise efficiently for some goods (it might even be illegal to do so, in the case of pharmacy and counterfeit goods), but the low cost of on-line retailing and advertising makes their sale economically viable.

In some cases the websites advertised through questionable means are completely bogus and will take the victim’s money without providing any service. “Get rich quick” schemes and “advance fee” scams fall into this category. In other cases, the website does provide a service: illegal on-line pharmacies, for example, may actually deliver products ordered. The quality and legality of the goods received by the user is obviously not guaranteed, nor does the buyer have the protection of anti-fraud laws.

Figure 2.5 presents common email spam payloads as categorized by Microsoft.

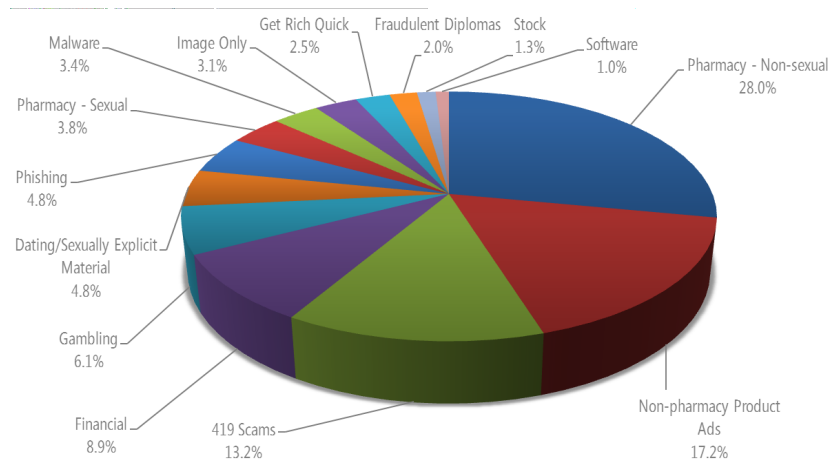


Figure 2.5: *Email spam types as detected by Microsoft products in the first quarter of 2011. [1] Notice how pharmacy-related spam accounts for the majority of messages. The 419 scams category includes all kinds of advance-fee scams and is named after the article that deals with fraud in the Nigerian Criminal Code. [30]*

2.2.1 Fake antivirus programs

Fake (or *rogue*) antiviruses are binaries (or webpages) that fraudulently attempt to convince the user that his or her computer has one or more malware infections. They are perhaps the most common representatives of the *scareware* family of payloads. Google estimates that fake antiviruses make up 15% of the threats detected by their extensive crawling and malware detection infrastructure. [31].

The details of how the fake antivirus presents itself to the users vary. Some include legitimate-looking setup procedures (Figure 2.6 shows the initial installation screens). Others are launched immediately after the user visits an infected webpage (see Figure 2.7 on the next page for an example); the page may in turn try to convince the user to download an executable to “clean” the (bogus) threats detected.

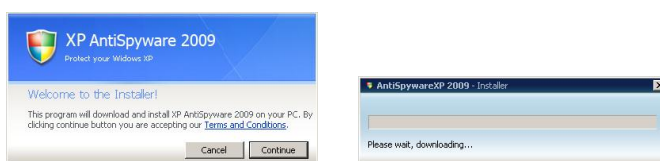


Figure 2.6: Initial installation procedure for the FakeRean/Braviax rogue antivirus family.

Whether or not a setup procedure is provided, fake antiviruses usually center around showing a fake scanning interface, modeled after the system scan dialog of real antivirus products. Some fake AVs attempt to mimic the windows of legitimate antiviruses with varying degrees of accuracy; others use completely made-up interfaces.

The defining trait of this type of malware is that, no matter how clean the user machine can be, the rogue scanner will always report the presence of infections. To enhance their credibility, some samples show the user believable names from known malware families.

While webpages can only show their content within the canvas of the browser, binaries can be much more obnoxious. The tricks employed range from confusing messages shown in the system tray (Fig. 2.9) to replacing the system association for executable binaries so that the user cannot run

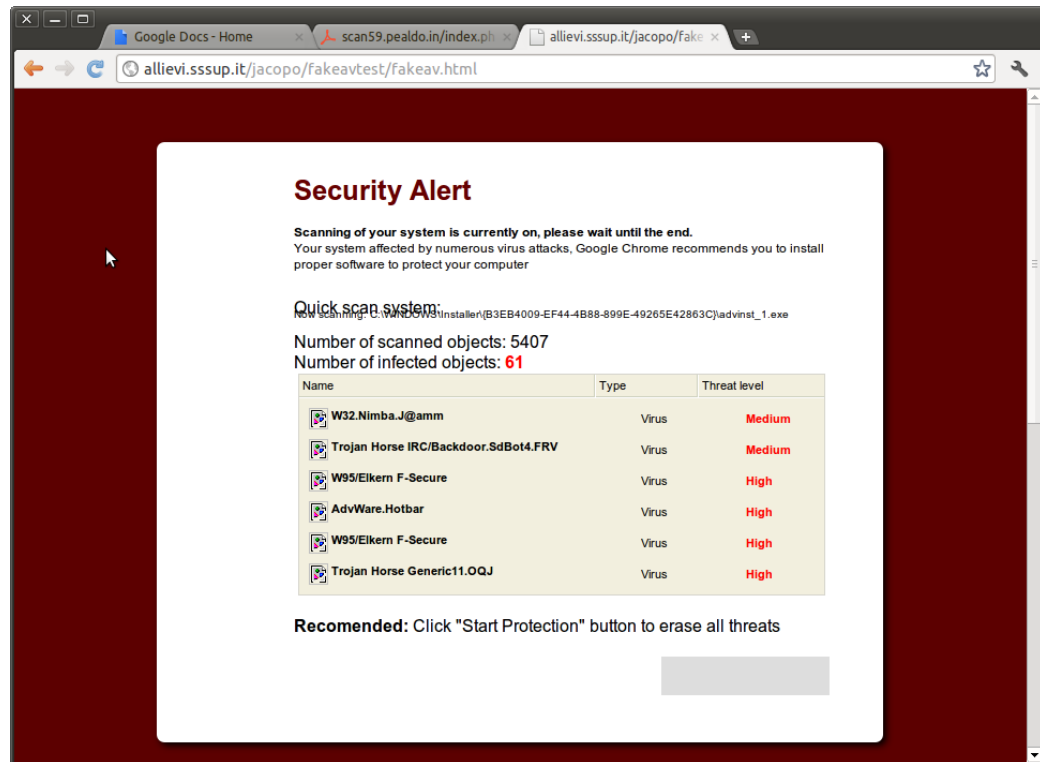


Figure 2.7: A rogue antivirus webpage. The original code is heavily obfuscated; note that the content of the webpage is OS-independent, in this case typical Windows file paths are being shown even if the browser is actually running on a Linux platform.

applications anymore (and would get a “system infected” window every time she tries to launch one). The malware can also claim that malicious or illegal activities (such as sending spam e-mails) are being conducted on the user’s computer.

Invariably, the user is presented with discomfoting scan results (Fig. 2.10). The only way to rectify the situation, according to the malicious program, is to purchase the “full version” of the rogue antivirus product, which will get rid of the (non-existing) infections.

The victim is then redirected to a page where credit card details can be entered; common pricing schemes include 6-months licenses available for about 50 \$, 1-year licenses for about 60 \$ and life-time licenses for about 90 \$ [35]. Once the user buys the registered version of the fake antivirus,



Figure 2.8: An original fake antivirus interface. Notice the invented names for the threats found.

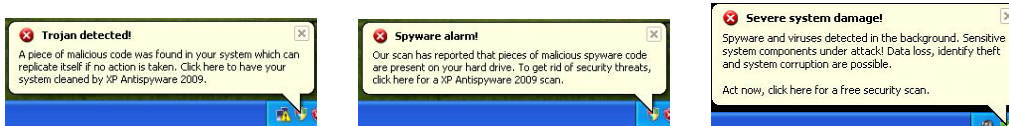


Figure 2.9: Messages shown in the system tray by Braviax Fake AVs



Figure 2.10: Result window shown by Braviax Fake AVs

the program will claim to have cleaned all infections (see Figure 2.11 on the following page).

Rogue antiviruses tend to be distributed in large campaigns with several independent affiliates. Typically, the rogue binary is distributed to the af-

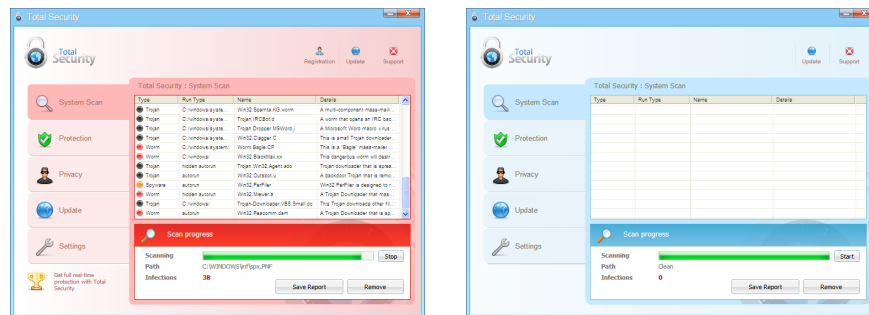


Figure 2.11: Comparison of the interfaces shown by the TotalProtect fake AV upon installation and after registration.

filiates ready for use as-is (the criminals who run the campaign handle all payments and disburse a quota of the profits to the participants; large *partnerka* operations in Eastern Europe may even gift luxury items to reward successful affiliates [35]).

The same binary will likely be used over and over by affiliates who may not have the technical prowess to modify it. This presents some challenges to the fake antivirus authors, if they want to keep their malware profitable for some time. After a while, in fact, the name of the rogue becomes well known; there are blogs (e.g. [34]) that specifically keep track of rogue antiviruses and offer removal instructions. A simple Google search, presumably within the technical means of even the most clueless user, could be enough to defeat the rogue. As a result, fake antiviruses change their names very often. This issue is somewhat unique to this category of malware: since they need to convince user to buy the product, a somewhat believable “brand identity” is necessary, yet it has to change frequently. [29]

Some families, like the OpenCloud rogues, reuse names of existing software products. Others, like those belonging to the FakeRean/Braviac family, will check the version of Windows they are running on and present themselves with names such as “Vista Internet Security” or “Win7 Antivirus Pro”.

A notable trait of all rogue antivirus families is the tendency to reuse interface elements taken from the security dialogs of the operating system. The standard “shield” icon that identifies security critical operations on recent versions of the Windows operating system, the red and white cross icon

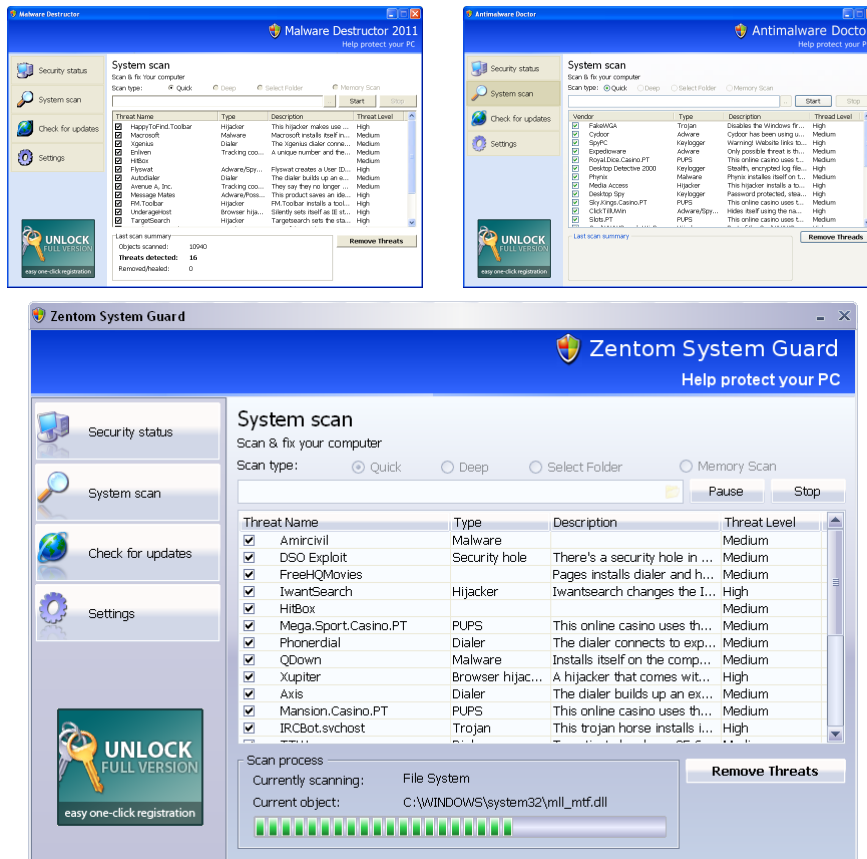


Figure 2.12: Alternate versions of the Zentom System Guard family of Fake AVs

used to signal errors, warning icons (in some cases even the general background images) and other standard elements are used to communicate a sense of risk and danger to the user.

Rogue antiviruses are commonly distributed in the form of Windows executables. Drive-by download vulnerabilities are used, especially when compromised ads are the infection vector. In many other cases, the user is simply invited to run the executable in order to perform some action.

A common way to defraud users is to lure them on a fake video-player page. When the user attempts to play the video, a fake error message is displayed that invites her to install additional video codecs or an updated version of the Flash plugin. Handily, the malicious webpage will provide the required executable, which in turn contains the rogue antivirus. In an

interesting variation of this approach, scammers will try to sell the (freely available) plugin to the user.

See Figure 2.13 for a typical example. Notice the simplicity of the code (Fig. 2.14) and the lack of any obfuscation. Since the download is initiated only after the user takes action, even this simple page will evade detection by analyzers like Wepawet⁶.

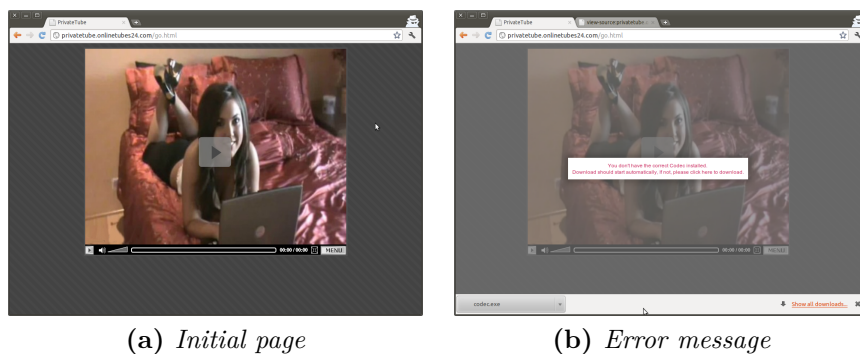


Figure 2.13: Dropper page using a fake codec error message.

```

$(document).ready(function() {
  $("a#inline").fancybox({
    'modal' : true,
    onComplete: function() {
      setTimeout(function () {
        window.location =
          'http://privatetube.onlinetubes24.com/codec.exe';
      }, 1000)
    }
  });
});

```

Figure 2.14: The script that triggers the download for the page in Figure 2.13.

⁶see Section 4.2.3 for details on Wepawet

In an interesting twist, the binary used by this particular fraud (`codec.exe`) relies on another webpage to get the content and essentially acts as a mini-browser. Presumably, this is done in order to dynamically update the campaign: in the current version, the payload is a fake antivirus page (see Figure 2.15). It is also possible that this is done simply to reuse existing assets as a cost-saving measure. As we have said numerous times, these payloads generate money through the gullibility of the user: writing a sophisticated interface could be an overkill.

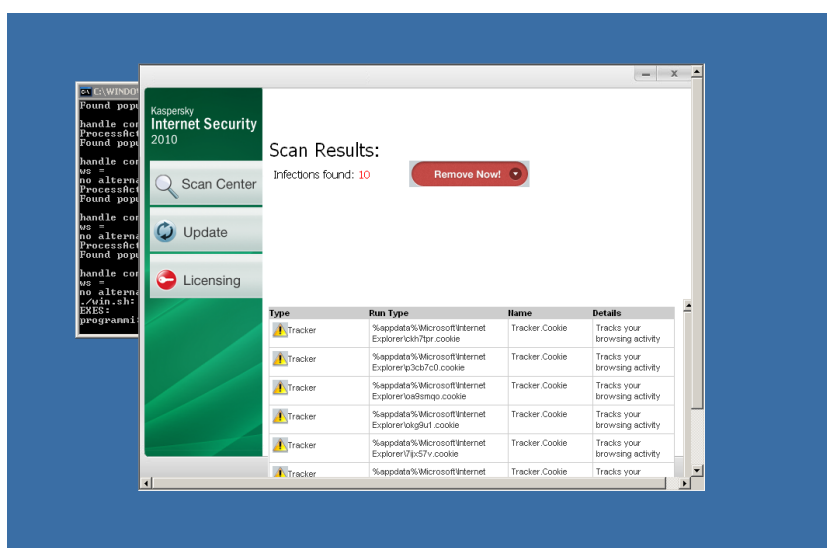


Figure 2.15: Interface shown by the fake codec binary. Even the window controls are fake and contained in the internal payload webpage. The code for the fake antivirus webpage is slightly more complicated because it has to show several interfaces but still does not use any notable obfuscation features besides generating the page structure on the fly and disabling right-clicking on the page.

The amount of obfuscation and complexity of the code varies widely. Some rogue antivirus binaries, like the one presented above, are very simple and have little more than the code needed to display the interface. Others have been reported to detect sandbox environments to make analysis more difficult, block access to security-related websites (including those of legitimate antivirus vendors), use rootkit techniques to make removal more difficult, etcetera. [29, 32]

All the propagation methods discussed in the previous section are used

to spread rogue antivirus software; enterprising fake AV fraudsters have also successfully compromised the official BitTorrent site to replace the legitimate installation program with one that bundles their software. [21]

A study has been performed on the underground economy surrounding rogue antivirus campaigns. Interestingly, it has been found that scammers actually issue some refunds to customers who realize the worthless nature of the software they purchased. Presumably, they do so to attract less attention from credit card companies by keeping the chargeback rate low. Payment intermediaries with lax anti-fraud policies like the Russian website Chronopay can also act as a shield from the stricter rules enforced by banks and major credit card companies. [35]

2.2.2 Other scareware frauds

While fake antiviruses are the most common and lucrative type of scareware, other categories exist. They all simulate a problem on the user's PC: the claims range from hard drive failures to threats to the victim's privacy. To simulate computer instability, malware can generate fake "blue screens of death" and system error messages, replace the desktop wallpaper, inject error messages in webpages, . . . [27]

Direct sales are not the only way scareware can be monetized. Sometimes scareware is bundled with more traditional malware like password stealing trojans and spamming bots. The additional malware may persist even if the user pays to remove the scareware. [29, 31]

Some scareware families even offer "comprehensive" PC health solutions. See Figure 2.16 for examples.

2.2.3 Common scams and frauds

While scareware is a relatively recent invention, other frauds have been perpetrated on-line for decades. Since on-line retailing offers a degree of privacy to the customer, it becomes possible to market "embarrassing" or illegal merchandise that people would be reluctant to buy in the open.

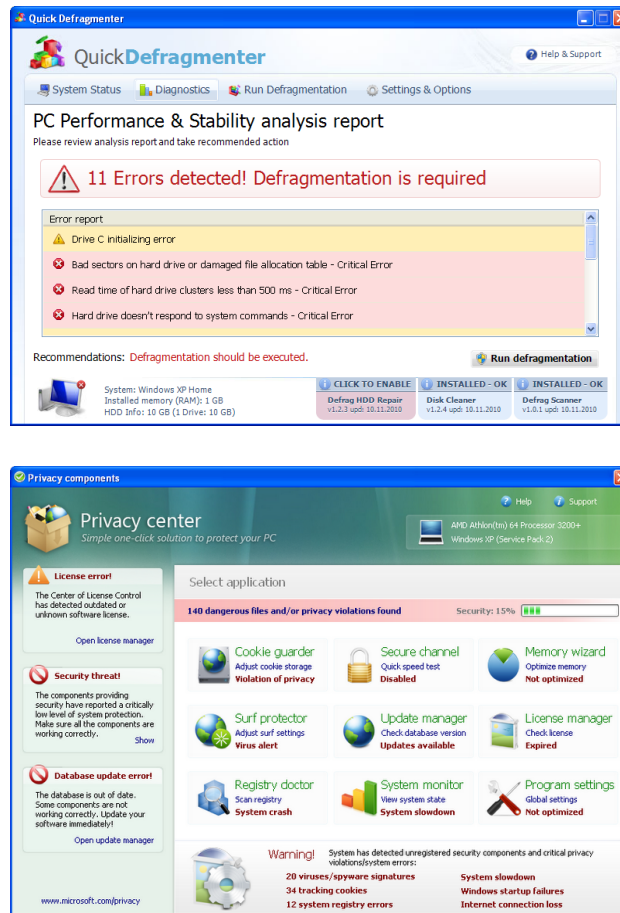


Figure 2.16: A fake hard-drive defragmenter and a “Privacy Center” window shown by the Tritax family of scareware.

Goods that are advertised through spam and other questionable means include:

Prescription drugs

Prescription drugs (especially those related to sexual health) are commonly offered through spam and other distribution channels. These websites offer illegal (or semi-legal) ways to bypass physician prescription requirements and offer relatively anonymous delivery. While most of these on-line pharmacies do actually deliver goods paid for by users, the customer does not have any of the guarantees that legal drug sales carry (both in terms of quality and safety).

Counterfeit items

Replica watches are also very popular with spammers. These counterfeit watches cost a fraction of the real brand-name product. Producers claim the quality and design of their replicas are indistinguishable from the original; obviously, the customer has no guarantees. Such sales may also be in violation of trademark laws and can potentially expose the buyer to legal repercussions should the merchandise be inspected by customs officers. Besides watches, other luxury items are also commonly advertised in spam emails.

Recreational drugs

Websites exist that claim to sell illegal drugs for recreational use. Some sell semi-legal “smart drugs” or try to exploit loopholes in drug control laws by selling intermediate products (or seeds, etcetera).

Unregulated gambling

Many states have tight regulations on gambling businesses and tax them at high rates. Through spam and advertisement, entrepreneurs publicize access to on-line casinos that are typically located in jurisdictions that do not restrict gambling and have lax taxation. Accessing those websites may be illegal from some states. As usual, the customer will probably have no legal protection from abuses from the website owners.

Fake diplomas and other documents

These goods are commonly offered through regular on-line advertisement. In some cases the spammers are technically selling access to an on-line university that will award a degree upon completion of some perfunctory exams. Such “universities” obviously lack any academic credibility and the titles they bestow generally have no value, but the procedure may be legal in jurisdictions that do not regulate academic degrees. Other services claim to help people obtain citizenship from the United States or from European Union countries. The degree of legitimacy of these services varies.

Heavily discounted software

Websites and spam emails may offer software titles at very low prices. While some sellers may be exploiting legal loopholes to resell volume licenses to individual customers, many others are reselling pirated copies. Legality considerations aside, such pirated software may also bundle malware and put the user at risk.

Other scams are not based on selling goods or services to users, but attempt to gather money from them just by offering “get rich quick” scams, requiring advance fees for large money transfers (supposedly as part of money laundering operations, which may or may not be real), or merely asking for a charitable donation to vaguely characterized causes and organizations. Citizens of Nigeria have earned a reputation for carrying out a large amount of these scams and Nigerian laws seem to be ineffective as a deterrent.

While fake antiviruses still make use of some obfuscation features, fraudsters have little need for those. No binaries are needed; no intrinsically malicious operations are performed on the user’s computer. Therefore, the detection for these scams must be based exclusively on their content. Image-based and JavaScript-based obfuscation is sometimes employed to make keyword-based detection ineffective, but no other techniques have been observed.



Figure 2.17: Chinese website claiming to sell counterfeit luxury goods and replicas.

Chapter 3

Detecting frauds

Traditional malware detection approaches are ill-suited to combat fake antivirus frauds and other scams. The fundamental problem is that excellent tools exist to analyze code, disassemble binaries, monitor modifications to host environments, etc. Unfortunately, these tools are close to useless when the user is the weak link.

As we have seen in the previous chapter, many successful scams implement at most a thin layer of obfuscation. In this work, we will present defense approaches that rely on the intrinsic characteristics of the scams and try to detect the very elements that attempt to fool the user: the assumption is that those elements depend on typical human behavior, which is likely to remain constant over time, while implementation details can be changed at will. Of course, cultural differences may undermine the effectiveness of certain detection methods (a trivial example is that a scanner that looks for English text will fail to recognize scams written in Russian or Italian), but the basic approach will hopefully need only small adaptations.

Reliable detection of all scams may never be completely feasible. In some cases, even a human researcher can have trouble in deciding if a given on-line store is going to defraud its customers or not. Detection can rely on indirect signals: legitimate on-line stores do not advertise themselves through spam, for instance. By combining knowledge about the propagation methods, the payloads and the habits of computer fraudsters, automated detection systems

can still significantly raise the bar for scammers. One of the reasons scams are popular with criminals is their low cost, both in economic and technological terms. Any system that increases the costs that fraudsters have to face may deter a good amount of them from attempting the fraud in the first place.

In this chapter some possible detection approaches will be reviewed, together with their strengths and weaknesses. Section 3.6, in particular, presents the approach currently implemented in the *Inner-Eye* system, which will be described in detail in the next chapter. Section 3.4 will describe another promising approach that has been considered during this work and has not been implemented due to time and complexity constraints.

3.1 Network-based detection

A tempting approach to combat scams (and malware in general) is to stop them before they can even interact with the victim system. As a thought experiment, consider a browser that can predict whether a webpage is going to be malicious or not without looking at its content; that browser can remain absolutely secure just by refusing to load abusive pages. All exploitation attempts, whether they are aimed at software vulnerabilities or at the user's naivety, need to interact with the environment in some way: if malicious pages are never loaded, perfect safety can be attained.

Obviously, implementing a system that can actually determine the malicious nature of a web page without loading it is not trivial.

3.1.1 Exploiting previously known information

If we have an external (and trusted) oracle that can determine the malicious nature of a page for us, all we have to do is ask for the information. One might wonder if this doesn't simply move the detection problem from the user's browser to the oracle system, but the problem may be much simpler to approach on the oracle's side: for instance, it can have more computing power than can be available on the user's computer. Moreover, the oracle can collect a huge amount of data from multiple users and analyze it off-line.

Such systems do exist in practice: the Google Safebrowsing API [22] and the equivalent service built into the Microsoft Internet Explorer browser are prominent examples. These services aggregate data from many sources (including, it can be assumed, human input and monitoring) and provide an API that allows checking whether a given URL is known to be malicious or not. Since it is impractical to query the oracle every time the user requests a page, data from the oracle is periodically packed and downloaded by the browser in the form of a *blacklist*.

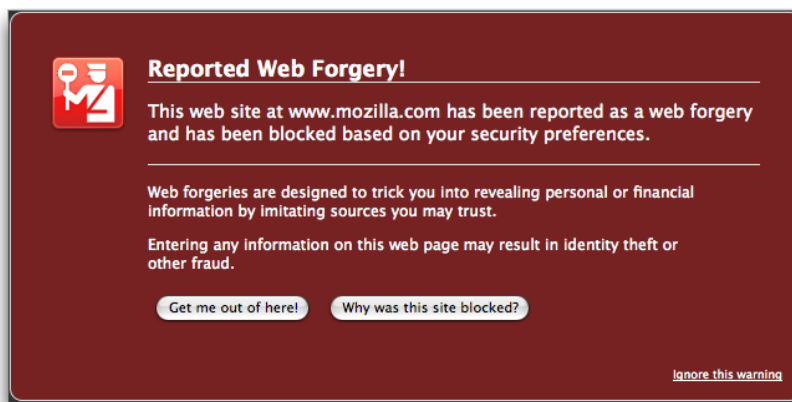


Figure 3.1: Message displayed by the Mozilla Firefox web browser when the user attempts to visit a URL blacklisted by the Google Safebrowsing service.

To prevent people from misusing the content of the blacklist (which basically amounts to a collection of vulnerable compromised hosts), only a hash of the URL is put into the list. This complicates granular blacklisting, as the client has to manually check all possible URL and path combinations (Google Safebrowsing has a granularity of four subdomains and five path components).

Other “oracle” blacklist services exist for known spam senders, vulnerable hosts, suspected C&C servers, ...

As criminals using botnets can use more and more new and “clean” IPs, IP-based blacklists are largely ineffective against modern campaigns: instead of depending on fixed IP addresses, malware can simply use domain names. Since the creators of the program (or the webpage) control the domain, they can modify the records to point to new IPs whenever they want. Naturally,

this simply moves the blacklisting problem from the IP-address space to the domain name space.

DNS-based blacklists can be evaded through *fast-flux* techniques: the domains used (for C&C servers, URLs embedded in spam e-mails, etc.) are constantly changed based on a pre-determined domain name generation algorithm. The malware author registers in advance the domains that will be used for a certain period of time. If the command and control servers of a botnet are contacted through this method, security researchers can reverse-engineer the name generation algorithm and start a registration race with the botnet owners. Successful takeovers have been performed with this technique [36]. Current name generation algorithms can incorporate unpredictable data (for example the Twitter trending topics) in an attempt to make it harder (or at least more expensive) to race with the botnet owners.

3.1.2 Network properties

Blacklists evasion needs force scammers to use highly volatile IP addresses and domain names. By exploiting this fact, systems have been developed that estimate the maliciousness of a web page simply by looking at readily (and safely) retrievable factors such as the Autonomous System number, the registration date for the domain and IP addresses involved, the number of redirects present, . . .

The main advantage of these methods is that they can be applied directly by the client with low overhead. If one is willing to accept the increased cost of using external centralized services, reputation data can be maintained for each network feature detected by the client. For example, a service could keep track of Internet Service Providers commonly used by criminals and respond to queries based on the AS number of the destination.

This approach has been applied to the detection of phishing pages in [24] and [14] and has been used to survey existing fake antiviruses in [16].

3.2 Visual similarity

An interesting way to detect phishing websites is to visually compare pages with known phishing targets such as bank websites. [40]

Detecting image similarity is not simple. If one were to simply render pages and compare, pixel by pixel, the resulting images, it would be possible to escape detection simply by changing some random pixels.

More sophisticated approaches use *shingling* techniques: the image is broken down into blocks, and single blocks are compared. If blocks are computed starting at every possible coordinate in the image, the detection can be robust even with respect to translations of elements within the page (assuming, of course, that the blocks are large enough to capture interesting elements). Scale invariance, however, cannot be obtained with this method.

Moreover, to make the comparison faster, the block content is typically hashed and put into a hash table. The choice of the hashing function is critical. Perceptual hashing algorithms must be used, otherwise even single-pixel changes will allow phishers to evade detection.

Since a collection of target images is required, this technique can't be easily adapted to detect scams other than phishing.

3.3 Image analysis

A different approach, developed to detect image-based spam emails, is to perform analyses on the entire image. [2]

These methods use simple image processing techniques and look for features that are known to be present in the images used by spammers. Text embedded in the image, for instance, is considered suspicious because it is assumed that legitimate emails will use images sparingly (because of size limits) and avoid their use when information can be conveyed textually. This limits the approach to email analysis, since images are overall much more common on the web.

Other features that can be used for detection include the overall color scheme, the saturation and heterogeneity of the colors, etcetera. These meth-

ods are also unlikely to give meaningful results for regular webpages, which naturally exhibit a huge variety of colors and characteristics. These methods are chiefly attractive due to their speed, and are usually proposed for real-time detection of spam e-mails. [9]

3.4 Sensitive image elements

As we have seen in Section 2.2.1, fake antivirus programs and webpages often mimic operating system or browser error messages. In all cases, they have to convey a sense of danger to the user, or confuse her with fake error messages. Relatively consistent graphical elements are employed to remain coherent with the typical user interfaces seen on the operating system. As we have seen, this is seen as an important goal by the scammers, to the extent that some rogue antiviruses detect the Windows version they are running on and change their appearance to better match its graphical appearance.



Figure 3.2: *Graphical elements common to many rogue antiviruses.*

This approach can be a good solution for the fake antivirus detection problem and was considered for implementation in *Inner-Eye*. Robust recognition of these elements has however proven tricky. First of all, fake antivirus interfaces vary widely in size: a scale-invariant algorithm has to be employed. Moreover, while the basic elements tend to be present with great consistency, the precise bitmap used varies among different fake AV families.

Object recognition techniques have been in development for some time. Promising approaches include finding scale-invariant “interesting” features in the image. SIFT [26] and the faster SURF [5] features are good candidates, possibly combined with the Harris corner detection [3]. It should be noted that these approaches also provide rotation invariance, which does not appear to be an interesting property (based on the survey of existing rogue antivirus performed during this work) and can be a source of unneeded complexity.

Simpler recognition techniques also exist, aimed at solving simpler problems such as tracking the ball in soccer game matches. Other algorithms rely on pre-built pictorial structures and try to find a match for them in the image.

Image analysis research, however, has mostly focused on high-resolution images or real-world videos. Due to the relatively low resolution of computer monitors, traditional object recognition techniques do not work well on screenshots. Finding meaningful SIFT features in the graphical elements pictured in Figure 3.2 on the preceding page has proven arduous, and image matching attempts (using both small elements and large image blocks extracted from common fake AV interfaces) have given mixed results.

Creating a detector based on these features remains as an open problem. Some approaches currently being considered for future work on the *Inner-Eye* system are presented in the final chapter.

3.5 Text-based detection

The traditional approach for spam detection is based on the exam of the body of the message. The assumption is that scammers need to use certain words to get the attention of victim: a random message is unlikely to get the user to click on a provided link, while the promise of certain goods will probably be more successful.

The most common algorithm used to detect e-mail spam uses a *bag of words* approach. It does not consider the order of the words as they appear in the text, nor does it attempt to reconstruct the meaning of the text. Conceptually, the system maintains the probability distribution of a bag of “good” words (likely to be found in legitimate e-mails) and of a bag of “bad” words (likely to be used by spammers). Considering the message as having been generated by choosing words randomly from one of the two bags, the system decides which of the two bags was more likely to be the source of the words found in the message.

More sophisticated approaches attempt to find the key words for the text by measuring their relative frequency. The *tf-idf* weight, for instance,

is obtained by dividing the term frequency in the considered document by a measure of the general importance of the word (the inverse document frequency, derived from the number of occurrences of the word in a corpus of documents).

3.6 Sensitive words

In the current implementation of *Inner-Eye*, the main heuristics employed revolve around the presence of certain sensitive words in the captured text.

Manual examination of scam pages (and rogue antivirus software) has revealed that certain words are almost always present in the messages shown to the user. This is because they are necessary to convey the desired message to the user (e.g. “infected”, “warning”, . . . in the case of fake antiviruses) or to advertise the goods that attract the user to the scam (e.g. “rolex”, “viagra”, “pharmacy”, . . . for common replica watches and prescription drugs frauds).

Of course a scammer may attempt to evade detection simply by not including any of the sensitive words in the page. If the word list is chosen correctly, however, this will severely gimp his ability to deceive users or to lure them into visiting the scam page.

3.6.1 Possible evasion techniques

A more realistic threat is that scammers can attempt to obfuscate the presence of sensitive words. Since these detection methods are easily deployed and are (supposedly) in use by search engines, miscreants appear to use obfuscation quite often.

Possible obfuscation techniques can be:

JavaScript-based

JavaScript can be used to dynamically alter the content of a web page and add information that was not originally present. Content can be decrypted from strings already present in the raw source code, obtained through network requests, . . .

These types of obfuscation aim to confuse static analyzers that simply look for sensitive words in the source without executing the code. Additional layers of obfuscation can be added by manually calling the JavaScript interpreter (e.g. using the `eval` function) to create new function bodies, new variables, etc. Simple dynamic analysis systems may be fooled by these tricks.

The emergence of JavaScript techniques such as asynchronous loading¹ and the implementation of entire web applications as scripts included in a single page² has made extensive dynamic modifications to the DOM tree quite common.

Image-based

Scams are aimed at humans, not computers: therefore, any text that the user will be able to read can be used to convey deceiving information. Sensitive text can simply be displayed using images.

This technique has been used for years to evade spam filtering systems and still sees some use in the field (although filtering systems now often consider a large image-to-text ratio as suspicious). Microsoft reports that currently about 3% of the blocked spam messages attempted to use this trick, down from about 9% in 2010. [1]

Once again, it should be noted that image-based text is often used by web designers simply for aesthetic purposes and cannot in itself be used as a reliable sign of obfuscation.

The widespread use of obfuscation can be a double-edged sword for fraudsters. Section 4.4.2.1 will show how *Inner-Eye* deals with these issues and how it can often turn obfuscation attempts into additional clues for the detection of malicious intent.

¹Modern webpages can use dozens of scripts. The traditional model of synchronous loading of scripts at the point of inclusion would lead to unresponsive loading for the pages or force authors to put all scripts at the bottom of the page, which is inefficient and can lead to interaction problems. To simulate asynchronous loading in browsers which do not natively support the `async` attribute, authors write small trampoline scripts that dynamically add code to the page.

²Twitter is a prominent example

3.6.2 Choosing the words

Appropriate choice of the words to include in the list is obviously crucial. Ideally, the list should cover the most common scams while generating as few false positives as possible.

In the current implementation, the word list has been constructed manually by observing collected scam pages and binaries. Refer to Figure 3.3 on the facing page for the words chosen.

Automated approaches to the construction of the word list have been considered. Words that frequently appear in fake antivirus scams, for example, could also frequently appear in legitimate antivirus websites. Using data available from Yahoo³, we tried to construct a list of words (or couples of words) that frequently appear in websites related to security but not as frequently in the general population of websites. The top results are shown in Figure 3.4 on page 50. Given the high number of unrelated words present (which can partially be explained by the abundance of marketing pages on the websites of security vendors), the manual approach was preferred.

It should be mentioned that if a “smart” classifier is employed the list does not need to be very precise or completely exclude false positives. Given the list mentioned above, for example, it can be guessed that generic words like “health” and “protection” are too common to be good signals of a malicious page by themselves. A good classifier, however, can use that information only in conjunction with others, assign it a lower weight, or disregard it completely.

³Yahoo! Term Extractor API, <http://developer.yahoo.com/search/content/V1/termExtraction.html>

Word	Exact occurrences	All occurrences
health	5641	7489
security	5606	5941
safety	3300	3573
protection	3163	3384
pharmacy	2237	2274
viagra	1793	1812
pill	1715	3569
cialis	1575	3216
levitra	1539	1549
propecia	1517	1517
kamagra	1163	1213
prescription	746	810
virus	505	1162
shield	338	554
scan	315	2029
threat	286	852
rogue	165	242
rolex	142	168
spyware	140	250
trojan	96	205
replica	86	335
infected	81	95
worm	52	281
adware	40	77
backdoor	22	37
scar	21	5125
protector	1	4

Figure 3.3: *The list of chosen words. An occurrence is counted whenever the text recovered from a web page contains the word (multiple matches from the single page still count as a single occurrence for the purpose of this table). Exact occurrences require the word to appear between spaces and punctuation. n-gram occurrences (in which the word is simply treated as a sequence of characters and matched in the text independently of context) are also recorded to compensate for OCR defects). About 20 000 of the 50 000 URLs scanned contained occurrences. Refer to section 4.4.2.1 for details.*

solutions industry
threat intelligence
strategic security
distributor partner
z services
portal partners
global alliances
business technology solutions
library products
solution services
database security
mobile security
security education
security risk
health check
english india
partner portal
anti virus
english usa
business operators
usa english
global english
language australia
protection internet free online tools
virus toolkit
management profile
security internet
vision management
customer resource
usa india

Figure 3.4: *Top results of the (discarded) automatic word list construction attempt.*

Chapter 4

The Inner-Eye system

This chapter describes the prototypical *Inner-Eye* system. While not all the techniques described so far are implemented by the system, *Inner-Eye* serves as a proof of concept that implements most of the functionality necessary for the detection of rogue antivirus and pharmacy scams.

Besides a high-level description of the system, the modular implementation and the internal API will be presented, as they have been instrumental in rapidly adapting from targeting exclusively fake antiviruses to a general-purpose scam detection system. The next chapter will show how this modularity can be exploited to create a scalable system by distributing the workload over multiple machines.

4.1 Overview

The final goal in the construction of the *Inner-Eye* system is the capability to analyze arbitrary website or binary samples provided by the user and automatically assign a classification label (e.g. as *benign*, *pharmacy scam*, *fake antivirus*, ...) to them.

The system is organized as a series of modules; each of them implements a specific phase of the analysis. An auxiliary module eases development by summarizing the results and providing programmatic access to the data and state of the entire system.

1. Submission system

In order to collect sufficient data for training, an automated submission system was implemented. For example, the *Wepawet*¹ web page analysis system provided a steady flow of both suspicious and innocuous URLs. Spam e-mail messages provided by a U.S. Internet Service Provider allowed the extraction of a large number of scam URLs.

Of course, manual submission of arbitrary URLs is also supported.

2. Capture environment

Each (possibly malicious) sample is loaded and allowed to execute in a controlled and instrumented environment.

There is a risk that the code being executed will detect the presence of the analysis system and hide its malicious nature: therefore, the capture environment has to be as realistic as possible. For this reason, *Inner-Eye* uses a real browser (to test websites) or a Windows virtual machine (where binaries can be given administrative privileges). Of course the system must also be able to reasonably contain the sample so that it cannot infect the entire system.

The capture system observes the behavior of the sample (possibly interacting with it to simulate user action) while monitoring its execution by taking screenshots, analyzing the elements created in the virtual environment and storing the data for future processing.

3. Processing

The screenshots and text retrieved from the previous phase are analyzed outside the capture environment. Optical character recognition (OCR) is performed on the screenshots. The text and images are analyzed and compared with each other: the result of this process is a set of features (or “clues”) that can be used by a classifier to recognize scam pages and executables.

4. Internal API and monitoring

In order to make the three aforementioned modules work together, a

¹<http://wepawet.cs.ucsb.edu>

certain amount of *glue* code and infrastructure needs to exist. While this task may seem straightforward, it can also become very time-consuming if done entirely by hand; this layer also needs to easily adapt to changes in the other modules. The *Django*² web framework was leveraged to easily model the application, create its persistent storage and provide a rich internal API used both by the other modules and for quick development and prototyping.

The next sections detail the design goals of each of these modules and their implementation in *Inner-Eye*.

The *Inner-Eye* system was originally conceived as a system to analyze websites and detect fake antivirus pages. It was then realized that its analysis capabilities are well suited to tackle the general problem of detecting computer-based confidence frauds. Besides adding detection capabilities for pharmacy and other web-based scams, a capture module for Windows executables was also built. The module has not yet been integrated with the processing and classification modules, but already has functionality to simulate user interaction and retrieve the necessary data; Section 4.3.2 will describe the module in detail.

4.2 Automated submission and training

The purpose of the system is to analyze arbitrary samples provided by the user. In order to fulfill its goal, however, the classifier must first receive training, and the *training set* must contain an adequate number of samples whose nature is known.

Three main sources of data provided samples to the *Inner-Eye* system:

- The top 10 000 most visited websites according to the Alexa ranking.
- About 5 000 URLs automatically extracted from spam emails.
- About 10 000 URLs submitted by users or feeds to the *Wepawet* system.

²<https://www.djangoproject.com>

4.2.1 The benign dataset

An easy way to get a good number of benign URLs is to simply get a list of the most popular websites. It is safe to assume that a scam website cannot get very far in popularity before being noticed and eventually taken down. Scammers are also known [31] to rapidly change their domains in an effort to evade DNS-based blacklisting attempt, which prevents them to acquire any significant popularity.

It could still be possible for a top-ranked website to be compromised and used to host malware. Direct defacements are rare, due to the sheer amount of visitors who can report problems combined with the fact that these sites are generally run by companies with significant financial resources or technical experience.

However, modern websites often use resources hosted on other domains to provide utility scripts (jQuery and other popular Javascript frameworks are available from many content distribution networks, for example), user tracking and statistics (Google Analytics and Webtrends are well-known examples), and advertisement. As we have seen in Section 2.1.4, ad networks have proven to be vulnerable to exploitation and have been used as a vector for the distribution of malware and running scam campaigns.

As a precaution against these risk and to ensure the completely benign nature of the dataset, only the home page of the website was captured. It is safe to assume that the insertion of malicious content on the homepage of a top website would be noticed in a very short time by security researchers; in fact, when Spotify (a popular music streaming website) recently had its ad network hijacked to redirect users to malicious websites, news of the mishap spread very quickly. [4]

4.2.1.1 Alexa topsites

In order to capture the “most popular” websites we need to define a popularity ranking. A well-known list of the top one million websites is provided by Alexa Internet, an Amazon subsidiary, free of charge and updated monthly³.

³<http://s3.amazonaws.com/alexa-static/top-1m.csv.zip>

The company aggregates data collected directly from user PCs through the use of a browser toolbar. The toolbar can be downloaded from the company website and is also bundled with third-party applications. Every time the user visits a page, the toolbar tracks the visit and displays information about the website to the users.



Figure 4.1: *Appearance of the Alexa toolbar as installed on a user PC.*

Privacy concerns aside, it should be noted that anyone can install the toolbar without providing any personal information. The “Alexa toolbar users” sample may therefore not be representative of the population of Internet users, especially for websites that are not very popular. For our purposes, however, we just need a sample of benign sites: the top 10 000 sites in the Alexa ranking are undoubtedly suitable.

4.2.2 The spam dataset

As we have observed in Section 2.1.1.1, spam messages may be hard to filter but are, almost by definition, very easy to collect.

In our case, a US-based Internet Service Provider provided a steady stream of millions of spam messages per day, as collected by a honeypot. This ensures that no false positives can be in the dataset: if a URL appears in the message, we can be sure that it is related to a spam campaign. Advance-fee, pharmacy and replica watches scams were found to be especially common in the dataset.

A system was then built to extract URLs of scam pages from the spam messages. Using the standard python email parser library, all text components were extracted from the message. Most URLs appeared in the clear and included the `http:\\` prefix so that Mail User Agents would recognize them and automatically provide a link to the website. These can be easily extracted by looking for the `http` prefix and using surrounding punctuation and spaces as delimiters for the URL string.

A small number of messages, however, was found to omit the protocol prefix or to have spaces between the components of the domain, presumably in an effort to confuse URL harvesting scripts and filters. As a robustness measure, strings that contain dots are also matched against the list of known top-level domains (TLDs) and an attempt is made to generate a valid URL.

```
Visit us at www .example .com
```

Figure 4.2: *A weak obfuscation technique sometimes employed in spam messages.*

The system analyzed seven millions of spam e-mail messages and extracted about 5 000 unique URLs (some campaigns appeared to consistently re-use the same URLs, while others used strings with random components). Manual inspection of some of the e-mail messages that did not yield URLs confirmed that a majority of the messages did indeed contain duplicate URLs or did not contain any link.

4.2.3 The Wepawet dataset

In order to gauge the performance of the system on miscellaneous URLs, a sample of about 10 000 URLs was collected from the ones submitted to the public Wepawet interface at <http://wepawet.cs.ucsb.edu>.

Wepawet [15] is a webpage analyzer that defeats common obfuscation attempts and detects malicious webpages. It is publicly available and anyone can submit URLs to be tested. The service also retrieves URLs from certain feeds made available by security researchers and that are known to carry a good percentage of malicious webpages. In fact, as part of this work, a publicly available feed known to regularly fake antivirus scam URLs (the *malware domain list*⁴) was added to the system: it should however be noted that time passes between the initial discovery of the malicious website and the instant when the feed becomes available and Wepawet has the chance to

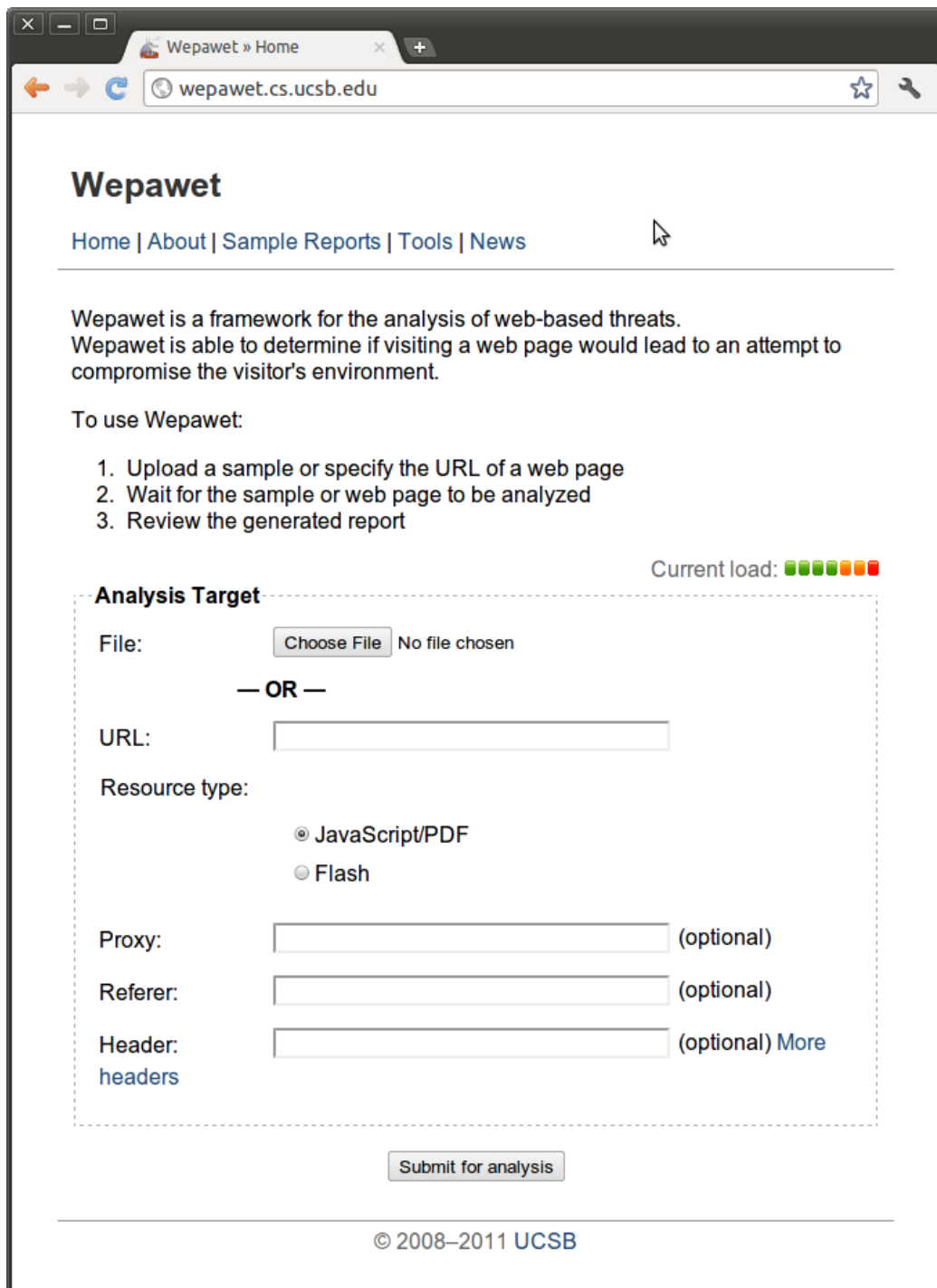
⁴<http://www.malwaredomainlist.com/>

scan the page. Malicious pages (or some their components) are often taken down before Wepawet can collect the full content.

Part of the interest in using the Wepawet dataset comes from the fact that it is known to contain a percentage of malicious pages that might actively try to infect the capture system: it also serves as a test of the robustness of the sandbox.

The dataset includes a large number of benign URLs submitted through random web crawling. Unfortunately, very few fake antivirus webpages were identified through Wepawet⁵. The dataset contains a good percentage of pharmacy and replica watch scams and was therefore still useful in evaluating the performance of the system.

⁵Wepawet does have some simple heuristics that attempt to identify common fake antivirus campaigns by checking the (de-obfuscated) text for some key words; however, this approach yields a large number of false positives and is easily defeated by image-based obfuscation. The data from those heuristics was not passed to *Inner-Eye*.



The screenshot shows a web browser window with the address bar displaying `wepawet.cs.ucsb.edu`. The page title is "Wepawet" and the navigation menu includes "Home | About | Sample Reports | Tools | News". The main content area describes Wepawet as a framework for analyzing web-based threats and provides instructions on how to use it. A "Current load" indicator is shown as a row of seven colored squares (green, yellow, orange, red). The "Analysis Target" section is enclosed in a dashed box and contains the following form elements:

- File:** A "Choose File" button and the text "No file chosen".
- OR —**
- URL:** A text input field.
- Resource type:** Radio buttons for "JavaScript/PDF" (selected) and "Flash".
- Proxy:** A text input field with "(optional)" to its right.
- Referer:** A text input field with "(optional)" to its right.
- Header:** A text input field with "(optional) More headers" to its right.

A "Submit for analysis" button is located below the dashed box. The footer of the page reads "© 2008–2011 UCSB".

Figure 4.3: *The publicly accessible Wepawet submission point. Notice how it is possible to manually specify the value of several HTTP headers: this is necessary to circumvent some evasion techniques employed by malicious pages (see page 22).*

4.3 The capture system

In this work, the *capture* phase is the one that collects information from the sample. The main goals of the capture are the same whether binaries or websites are involved:

Realism

It is important to present to the sample code an environment that is as similar as possible to the one that it expects. This is important both for the accuracy of the analysis and to exclude the possibility that the sample code can detect and evade the analysis system.

Instrumentation

The capture system must be able to collect a significant amount of data. Ideally, it should be able to examine both the inner workings of the sample code and the appearance of what a user would see on her monitor.

Sandboxing

The analysis system must be able to operate on arbitrary samples; many samples will contain questionable content or be outright malicious and attempt to take control of the capture environment. The system must be able to contain the infection while still fulfilling the main goal of gathering data in an unobtrusive way.

These goals may appear to be somewhat mutually excluding. A perfect sandbox, for example, would severely contain the sample code and limit its ability to interact with the network. This behavior, however, would make the analysis too unrealistic (for example, malware samples sometimes load additional components from the network).

Escaping detection and achieving accuracy are strongly related objectives. To gather as much data as possible, one might envision building a custom emulated environment. For example, one might implement a special browser that keeps track of all loaded files and of all the code that is

executed. Indeed, the Java HtmlUnit⁶ package is an example of such a system. When confronted with real-world pages, these systems will invariably lose accuracy. Web designers (even when they have no malicious intent) are known to rely on obscure quirks in existing browsers in order to activate different code paths in specific browser versions, for example. Exploits and obfuscation techniques often depend on platform-specific tricks or complex interactions with plugins and page resources. One would be confronted with the Sisyphean task of adding more and more functionality to the analysis system to achieve better coverage, only to uncover more bugs and quirks (and the necessity to emulate them!) while probably being outpaced by the constant flow of releases by browser vendors⁷.

The advantages of building an analysis system over an existing, real, environment should be evident. Sandboxing and instrumentation can become tricky, however, especially when the original platform does not natively offer those capabilities.

In the next two sections we will show how judicious use of software and hardware *virtualization* can alleviate many of the issues presented here, both for website and binary analysis. As a bonus, virtualization-based solutions are easy to integrate in a scalable architecture, as we will see in Chapter 5.

4.3.1 Capturing webpages

Inner-Eye captures webpages through the popular Mozilla Firefox web browser, instrumented to allow scripting and remote control.

The instrumentation is achieved through the *Selenium* extension⁸. Originally intended as a testing framework for web applications, Selenium enables control of browsers in a platform-independent and vendor-independent way. This will make it easy, should the need arise, to modify *Inner-Eye* to operate

⁶<http://htmlunit.sourceforge.net>

⁷New versions of the Google Chrome browser, for instance, are released with almost monthly cadence and see very rapid adoption due to the silent and unobtrusive upgrading mechanism. Mozilla Firefox developers have recently adopted a similar release strategy.

⁸<http://seleniumhq.org/>

with different browsers (several versions of Internet Explorer and Webkit-based browsers are supported by Selenium).

In particular, the Selenium server component is used. This component consists of a daemon that accepts remote control commands over the network and is able to spawn clean browser instances at will. Since Selenium was conceived as a testing framework, it takes special care to ensure that the state of the browser is consistent at the start of every test case; through the use of temporary profiles, it still allows the website code to run in an environment that maintains state during the analysis session.

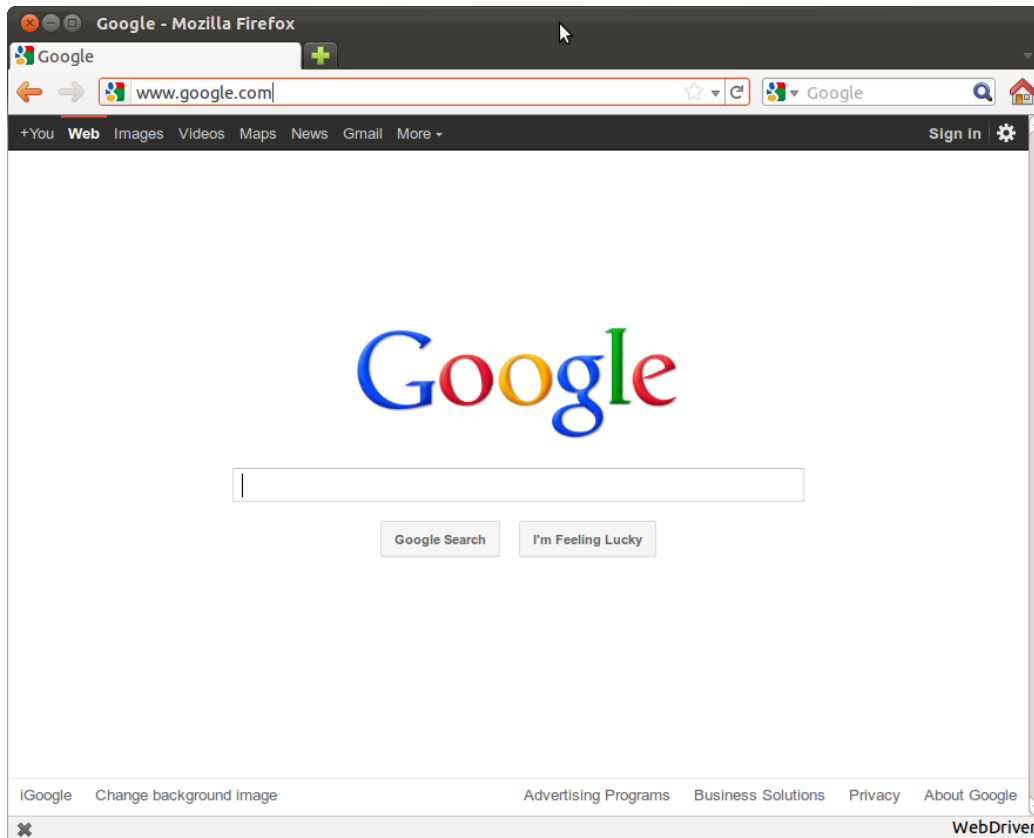


Figure 4.4: *Google's homepage as loaded by the instrumented browser in a clean instance (in the real capture environment all output is redirected to a memory framebuffer, but for testing purposes regular screen output can be enabled).*

In the current system, the following analysis steps are performed:

0. Before any analysis is conducted, the domain part of the URL is checked to see if it resolves to an IP address. This avoids committing resources to the analysis of clearly mis-extracted URLs (the system that extracts URLs from spam e-mail is prone to make mistakes, for example). It was observed that (even in the Alexa 10k dataset) websites often do not set an A DNS records for the base domain, but only for the `www` subdomain. This situation is detected and automatically corrected.
1. The raw source code of the page is fetched. This is done through a version of the open-source *curl* library⁹ configured to send requests that resemble those sent by a real browser (by modifying HTTP headers such as the *User-Agent* value).
2. An instrumented browser session is created and directed to visit the sample URL.
3. After the browser signals that it has finished loading the page, a screenshot of the page as the user would see it is taken. The screenshot is taken through native functionality implemented by the browser and includes the entirety of the webpage, as if it was shown on a screen large and tall enough to contain all of it.
4. Through the standard DOM¹⁰ introspection API, all text nodes in the DOM are retrieved. This includes any node that might have been dynamically created by the page.
5. The browser session is closed and resources freed. The system is now ready to capture another page.

In the current implementation the page content is fetched twice, once by *curl* to retrieve the raw source and once by the instrumented browser.

⁹<http://curl.haxx.se/libcurl/>

¹⁰The Document Object Model is a standard model of interaction with documents created from markup languages such as HTML and XML. It presents the document as a tree of elements and provides methods to traverse it and retrieve information about the nodes.

(Current browsers do not retain the original, unmodified, source of the page.) This can be a problem, as malicious websites will sometimes serve the malicious content only once per IP, as an annoyance and weak evasion attempt against analyzers. Given the nature of the datasets used in this work (well-known websites and URLs that have already been fetched and analyzed by an analysis system), the evasion issue was considered immaterial. Furthermore, it should be noted that a system is being developed for Wepawet that is able to act as an HTTP proxy, intercept the requests, and serve already retrieved resources through a cache; should the need arise, the system could be interposed between *Inner-Eye* and the network.

4.3.1.1 Robust handling of character data

All the data retrieved in the capture phase is stored for later processing. A somewhat interesting and unanticipated problem that has been encountered is the robust handling of possibly corrupted or malicious data.

Data retrieved from the network can be in any format, be malformed, corrupted, malicious, etc. As the ultimate goal of the system is to allow submissions from the public, we must also consider a scenario where an attacker is actively trying to compromise the analysis system by submitting specially crafted webpages for analysis. Robustness is the key here.

Since *Inner-Eye* works mostly on text (screenshots are retrieved only through a safe API exposed by the instrumented browser), storage and parsing might seem a trivial problem. However, even text can present unexpected behavior when one takes into account the Unicode system.

Unicode is an international standard for the encoding and handling of text, defined as a sequence of *code points*. Code points abstract the notion of character and allow encoding messages in any language; each code point has been assigned a number and over one million code points exist. When text is to be stored or transmitted, its code points have to be encoded as a sequence of bytes according to an *encoding* scheme. Legacy encodings used exclusively one byte per character and could represent only a limited number of code points (typically those used in a language or a group of languages),

while variable-length encodings such as UTF-8 and UTF-16 can represent any code point.

Letter	ISO-8859-1	UTF-8	UTF-16	UTF-32
è	E8	C3 A8	E8 00	E8 00 00 00

Figure 4.5: Example Unicode encodings for the letter “è” (code point $U+00E8$). UTF-16 and UTF-32 are shown in little-endian byte ordering.

Due to the existence and continued usage of various encodings (more than ten have been observed in the datasets used), it is impossible to meaningfully parse text without knowing its encoding. Moreover, in the presence of malformed data, the decoding procedure can be destructive. Some websites in our datasets send wrong or malformed encoding information (and sometimes even declare different encodings in the HTTP headers and in the document header). Random or malicious data can also cause problems for a parser; vulnerabilities related to incorrect handling of Unicode text have been frequently reported for many software products.

To facilitate development, it is often recommended to convert all incoming text to a single encoding; with this approach, only “boundary” code will need to deal with Unicode encoding while the core application logic remains shielded from these problems. However, as we have seen, determining the correct encoding for a web page is not a trivial problem. Therefore for *Inner-Eye* a different approach has been followed: data retrieved from the network and controlled by a potential attacker is always stored as-is without parsing. If an encoding was specified through an HTTP header, that value is also stored.

During the processing phase, the retrieved text will be parsed by the robust HTML `BeautifulSoup` parsing library, which will determine the correct encoding of the data using the same heuristics employed by Firefox. It should be noted that a large number of Unicode parsing errors has been encountered while analyzing the available datasets. The `BeautifulSoup` library has been modified to be slightly more tolerant in case of decoding errors, in an attempt to extract as much information as possible from the retrieved data.

4.3.2 Capturing Windows executables

As we have seen in Section 2.2.1, rogue antivirus software is often distributed in the form of Microsoft Windows executables.

As many of the analysis capabilities of *Inner-Eye* depend exclusively on the appearance of the window presented to the user, extending it to deal with binaries was seen as a natural choice, especially to deal with the scarcity of live fake antivirus webpages. Binaries are usually less dependent on network resources than webpages and often lend themselves to analysis even after the pages that distributed them have been taken down (the payment infrastructure, however, is usually web-based).

To achieve a good level of realism while isolating the sample from the internal network, the binary is executed in a Windows virtual machine. The machine reproduces a common environment currently found on user PCs, namely a Windows XP installation with some common auxiliary programs such as browsers and plugins such as the Adobe Flash player. The binary is executed with full Administrator capabilities, as it is common for malware to expect such a situation (Windows XP home users often run all their software under an administration-capable account), and is able to freely modify the environment as it wants.

Network communication can be restricted as desired; Internet access is allowed by the current configuration. If this is undesired, honeypot-like systems exist that monitor traffic and send simulated responses from common services such as IRC or HTTP; the idea is to trick the malware into believing that it has network connectivity while in fact keeping it in complete isolation (a similar system is implemented in the Anubis analysis system [6]).

Through the use of non-persistent disk storage, the VM can be restarted in a clean state every time. Modifications are stored in a temporary file as differences from the base image; persistent storage can be re-enabled whenever modifications to the capture environment are needed (e.g. to install patches)

The virtualization solution chosen (VMware) also offers simple communication capabilities between the host and the guest environment. These are

used to automatically to automatically update the internal scripts, transfer the sample in the guest, launch the analysis and periodically take screenshots of the virtual screen.

4.3.2.1 Simulating user interaction

Although rogue antivirus programs can be distributed through drive-by install vulnerabilities, many samples rely solely on the user to voluntarily run them.

To mask their fraudulent nature, the first binary executed by the user is often designed to look like a regular installation program. Legitimate antivirus products generally require installation before they can be used: even not very experienced users would find it strange for a program to pop up security warnings as soon as it was started.

As a bonus, by running a convincing installation procedure, the user can be tricked into ignoring warnings displayed by products like Windows Defender: these products keep track of programs set to automatically start at boot time and warn the user whenever the list changes; however, legitimate software will often trigger similar warnings during setup and the user is likely “trained” to ignore them whenever confronted with new software.

Some usability guidelines are published by Microsoft, but there are no real standards that govern the construction of setup programs. However, most setup programs are fairly predictable and will present a sequence of dialog pages linked by “Next” and “Previous” commands. As the Win32 API allows enumerating and controlling windows on the same desktop (even if they belong to other processes), *Inner-Eye* uses a Perl script that attempts to recognize common installation windows and automatically interact with them until the setup procedure is complete.

Example heuristics employed by the script include:

- Advancing through the pages of the installation dialog.
- Automatically accepting the license terms.

- Simulating a click on the “Finish” button, but only if no “Next” button is still present.
- Disabling automatic restart.
- Confirming and dismissing pop-up messages.

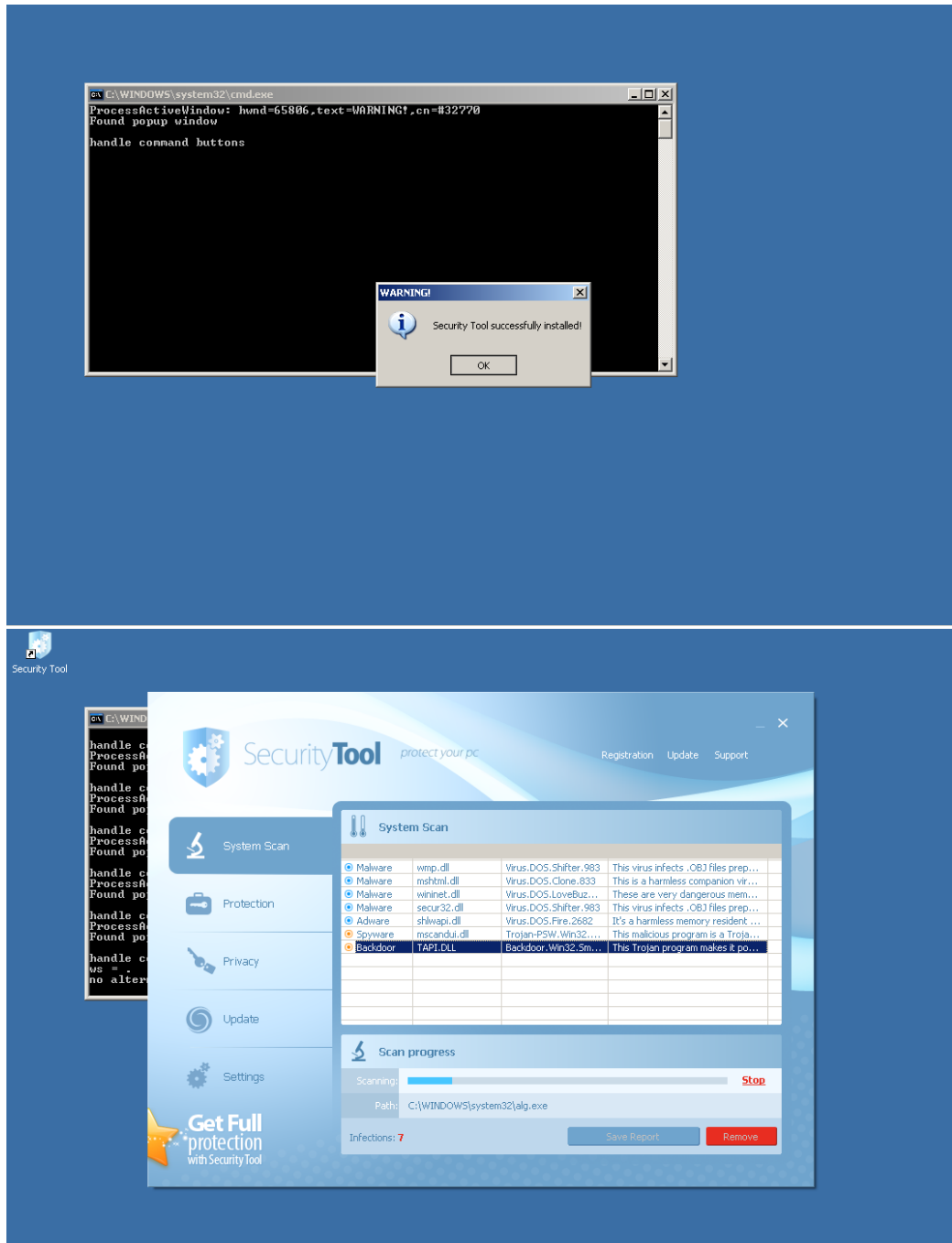


Figure 4.6: Sample interaction with the setup procedure for a fake antivirus program. After simulating a click on the “OK” button, the fake scan interface is displayed. For testing purposes, messages from the simulation script are visible in the background window (this can be avoided if necessary).

4.4 Processing phase

The successful storage of data from the capture phase will trigger further processing. As a first step, the captured screen image undergoes Optical character recognition (OCR); the complete analysis is performed when the OCR results become available.

The capture, OCR and processing modules are completely decoupled and can operate at different speed without problems. The native filesystem monitoring API exposed by the Operating System is used to trigger processing of new files (information about the state of the system is kept in a database).

4.4.1 Optical Character Recognition

OCR programs employ a series of heuristics and artificial intelligence techniques to automatically recognize text areas in an image, recognize the single characters and reconstruct the original text; they are commonly used on scanned documents, to quickly sort mail, to verify bank checks, etc.

Traditional OCR applications require high resolution images. It is not uncommon for printers and scanners to support resolutions of thousands of DPI (dots per inch). Computer screens, however, can render text only at a much lower resolution, and webpages, images and applications are not generally able to scale to arbitrary resolutions. To this day, for example, the dominant imaging formats for web content (PNG, JPG, etc.) are *raster* formats: compression aside, they represent a matrix of pixels with fixed dimensions. While the commercial OCR solution used for this work was usually able to cope with low resolution text, in order to overcome some inaccuracies a manual training phase has been performed.

Of course no amount of training can completely eliminate errors and whenever OCR is employed one must be prepared to compensate for them. Both classification and zoning errors are possible [38]. Classification errors are the simplest: they occur whenever the OCR engine mis-recognizes or omits one or more characters in the text (extraneous characters can also be added). As we will see below, the impact of classification errors can be reduced with compensation techniques.

Zoning errors can also occur. Before attempting character recognition, the OCR engine has to determine which regions of the image contain text and must order them. This work uses the mere appearance of words as a classification signal: ordering errors are therefore inconsequential. Misrecognition of text areas can have a stronger impact. Manual analysis of some samples revealed false-positive cases (e.g. meaningless character sequences resulting from attempting to “read” drawings and photographs) but relatively few false negatives (unrecognized text areas). False positives have no impact on the classification performed by *Inner-Eye*, since it is very unlikely for them to introduce sensitive words in the text.

4.4.1.1 Mitigating OCR errors

In the experiment performed in this work, the main quirks exhibited by the OCR system after manual training were:

- Some trouble in recognizing the spacing between words: spaces are sometimes missing from the reconstructed text and words are concatenated.
- Confusion between the lowercase letter “r”, “m” and “n”, likely due to the fact that the beginning “leg” of these characters is identical in many fonts.

To compensate for the first problem, two different types of matching are performed. In the first, *exact*, matching, words are required to appear surrounded by whitespace or punctuation, as they naturally do in written text. A second matching algorithm treats the words as sequences of characters (*n-grams*) and any occurrence of the sequence is considered a match. Classifiers based on *n*-grams are commonly used to compensate for OCR and spelling errors [11]: the most powerful approaches involve computing the frequency of all *n*-grams over a certain length and using the frequency table to compute the distance between documents. In this work the simpler technique of looking for pre-determined *n*-grams has been employed.

Compensation for the second problem is even simpler: words that are likely to be mis-recognized by the OCR engine are included in the word list in both spellings. For example, both *scan* and *scar* are included. They still appear as distinct words to the classifier, in order to avoid over-generalization: in the above example, for instance, *scar* is likely too generic to give meaningful classification results, while *protection* and *protector* are very likely to be paired by the classifier.

4.4.2 Features

When text from the OCR engine becomes available, the following elements are available to the processing phase in the current implementation:

- The raw HTML source of the page.
- The text extracted from the dynamic DOM of the page.
- The text recognized by the OCR.
- Metadata such as the URL, the remote IP address, the source of the submission, the Alexa ranking of the domain, the encoding specified in the HTTP headers, ...

As mentioned in Section 4.3.1.1, the HTML source is passed to the python `BeautifulSoup` library, a robust HTML parsing library that can reconstruct a sensible DOM tree even from malformed document. This is necessary both as a robustness measure and because in many cases even benign pages don't strictly adhere to the correct syntax for HTML. Text nodes are then extracted from the constructed tree and the original text is retrieved.

4.4.2.1 Extracting words

After retrieving the textual content, we have to extract useful data for the classifier. Very complex algorithms exist to classify text, ranging from text distance metrics to systems that attempt to parse natural language and reconstruct the meaning of the text. Besides being very complex and CPU-intensive, these methods also tend to be very fragile.

As we have seen, OCR engines are prone to commit several errors. Zoning errors are particularly detrimental to algorithms that are sensitive to the ordering of text; moreover, the “correct” way to linearize text in the presence of complex table and image-based layouts is subjective and even a human reader could order the text of similar pages in different ways. Therefore, a word-based approach was chosen: since with this method the ordering of the text does not have any impact, zoning errors do not generally constitute a problem.

Correct feature selection is also very important. The easiest way to build a word-based classifier is to consider every possible word: this will feed the classifier a large amount of irrelevant data, needlessly increase its complexity, increase the likelihood of overfitting and decrease the overall performance of the classifier. Our list of sensitive words (Sec. 3.6, Fig. 3.3 on page 49) serves as a first selection that will filter out useless data.

In the current implementation *Inner-Eye* operates on two kinds of words: exact word matching and n -gram matching. The kind of match is exposed to the classifier, so that it can differentiate between the two.

4.4.3 Detecting obfuscation

The mere presence of a certain set of words can sometimes be enough to classify a page as malicious. However, this approach is prone to false positives.

As we have seen, the authors of scam and fake antivirus pages employ obfuscation to evade code-based detection engines. The OCR-based detection implemented in *Inner-Eye* can not only easily defeat this obfuscation, but can also turn it into an additional detection mechanism.

By comparing the number of occurrences of each sensitive word in the raw source text (`count_src`), the dynamically-generated DOM tree (`count_dom`) and the OCR-reconstructed text (`count_ocr`), the system exposes to the classifier features that can reveal obfuscation.

The following possibilities arise:

`count_dom > count_src`

This case typically occurs when JavaScript-based obfuscation is em-

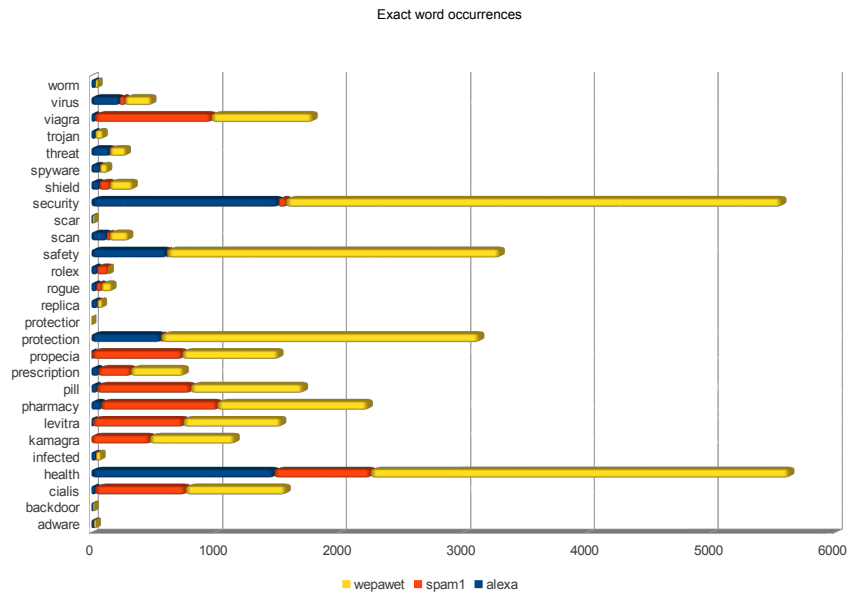


Figure 4.7: *Exact word occurrences grouped by source.*

ployed. The raw code of the page looks like random garbage (or contains decoy text to fool search engines), plus a small decoding script. The malicious content will be decoded and added to the page at runtime. This style of obfuscation is typical (and almost required to escape detection) for drive-by download and plugin vulnerability attacks, but is also commonly employed in fake antivirus pages.

count_ocr > count_dom

This case typically occurs in cases of image-based obfuscation. The page will evade even sophisticated scanners that execute JavaScript and parse generated text sections. False positives can occur: as a way to work around platform limitations (e.g. the availability of fonts) and browser differences, web designers sometimes use images to display stylized text for titles, banners, etcetera.

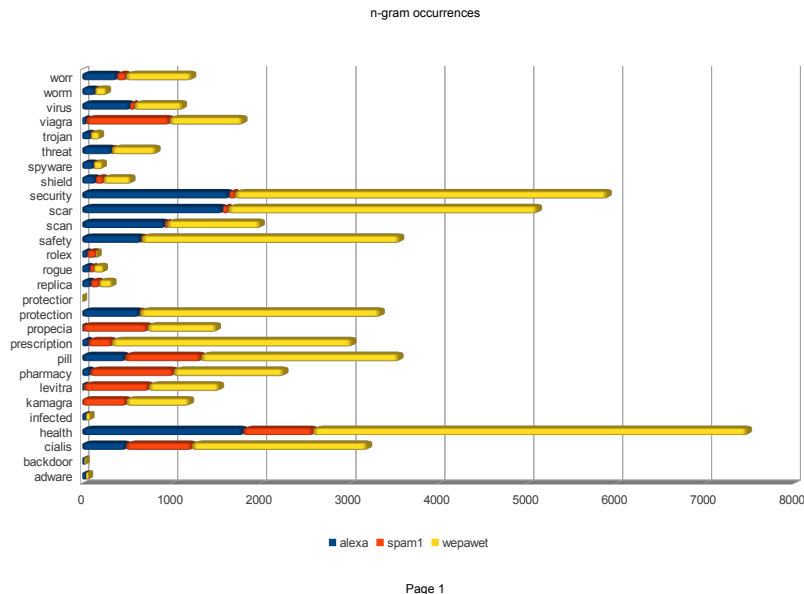


Figure 4.8: *N*-gram word occurrences grouped by source.

`count_ocr > count_src`

This condition may indicate that both JavaScript based and image-based exploits are in use.

`count_dom < count_src`

This case is relatively rare. A rule based on this condition would detect text in comments or not yet added to the DOM tree. Since this text is not visible to the user and cannot be used for obfuscation, this case is not exposed to the classifier.

`count_ocr < count_dom`

This condition indicates element hiding. Modern web applications often pre-load data in the DOM and visualize it only when necessary. Tab-based interfaces are also implemented in this way. As this condition has many legitimate reasons to occur and almost no potential for abuse, it is not exposed to the classifier.

count_ocr < count_src

Besides OCR errors, this condition can occur due to element hiding, text in comments, etc. Since there is no potential for abuse, this case is not exposed to the classifier.

Equal number of occurrences

This is the expected case for a page that does not perform obfuscation. It has been observed that by hiding these cases from the classifier we can exclude a good number of false positives while still catching all pages that attempt obfuscation. Given the scarcity of malicious pages that do not attempt any obfuscation, *Inner-Eye* does not currently consider this case in the classification: the presence of a word causes features to be exposed to the classifier only if obfuscation is also detected.

It can be interesting to mention that *Inner-Eye*'s detection system works in pretty much the opposite way a SEO trick detection would. Search Engine Optimization refers to a series of techniques to improve a site's ranking in search engines. SEO tricks include legitimate, benign practices such as ensuring accessibility of all content, avoiding duplicates, structuring sites along a clear hierarchy and other site construction guidelines. These practices are widespread and their use is often recommended by the search engines themselves [23]. Questionable practices include attempts to artificially inflate the page ranking by including content and links that will be visible to the search engine (and potentially lure the bot to visit more of the website, give it a higher ranking, include it in unrelated search result pages, ...) but invisible to the users.

Each feature employed by *Inner-Eye* has a descriptive name such as “dom_vs_ocr viagra”: this feature is sensitive to the mismatch in the number of occurrences of the n -gram “viagra” in the DOM tree and in the OCR-reconstructed text.

4.4.4 Classification

As the final step of the processing of a sample, *Inner-Eye* has to use the features gathered and exposed in the previous phases to determine whether

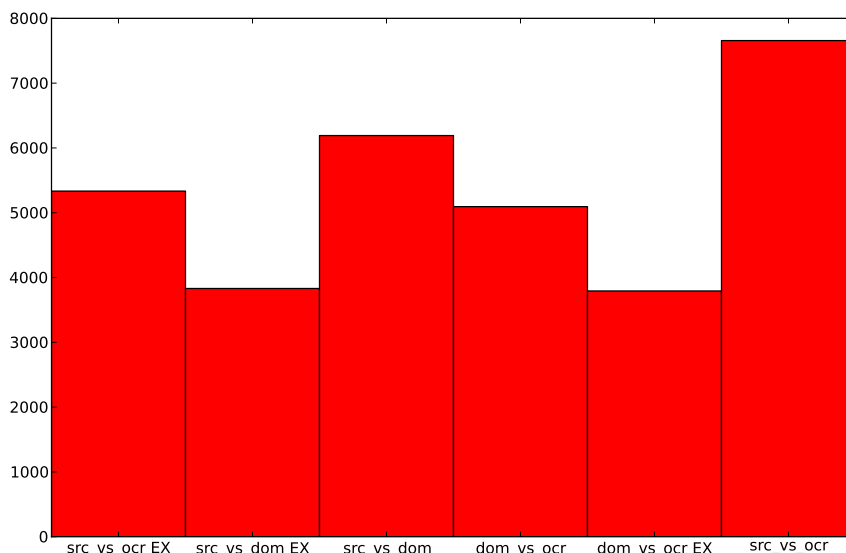


Figure 4.9: Categories of features that will be exposed to the classifier and number of occurrences. Features with “EX” in the name correspond to exact word matchings, the others to n -gram matchings.

the page is benign or a scam.

Due to their good performance in text classification tasks, ease of use and resilience to overfitting, *Inner-Eye* employs an SVM-based classifier.

Each sample is represented as a point in the feature space. In the case of webpage classification, for instance, each URL can be represented as a vector with one coordinate for each possible obfuscated word and obfuscation method. For example, coordinate `dom_vs_ocr viagra` will be one if the n -gram “viagra” appeared more times in the text from the OCR than in the one from the DOM tree. Alternatively, the value could be a real number that takes into account the total number of occurrences versus the number of obfuscated occurrences; while this may seem a good idea to reduce the number of false positives, experimentation has shown that binary features perform better in our case.

Since in our case each sample generally exhibits a small number of features, it would be represented by a vector with many zeros and a few ones.

To conserve space, a *sparse* representation is used both in the database and in the input to the classifier.

1	15:1	21:1	24:1	25:1	26:1
---	------	------	------	------	------

Figure 4.10: *Sparse representation of a sample URL from the spam dataset in libsvm format [12]. The first number is the label that will be used during the training phase. The rest of the line represents the training vector (features `src_vs_dom cialis`, `src_vs_dom pill`, `src_vs_dom cialis EX`, `src_vs_dom levitra` and `src_vs_dom viagra` have a non-zero value for the sample.)*

4.4.4.1 Support Vector Machines

The classification problem can be expressed as the division of the feature space in regions that correspond to each classification label. If we limit ourselves to a binary classification problem (scam or benign), one might try to use a hyperplane to separate the two classification regions. Single layer perceptrons are a possible implementation of this kind of linear classifier.

It is well known, however, that these simple classifiers can solve only problems where the interesting regions are linearly separable in the feature space. An important indication on how to overcome this limitation comes from the Cover theorem [19], which in qualitative terms states that a good way to make a problem space linearly separable is to map it (non-linearly, of course) into a space with higher dimensionality.

The idea of mapping the original problem space to an intermediate space where separation will be easier is the foundation of many existing classifiers. Multi-layer perceptrons, for instance, are built with a “hidden” intermediate layer which generally has more connections than the input and output layers, which are constrained by the original problem statement. Since the number of neurons in the hidden layer is generally fixed before training, one is confronted with the problem of estimating the “true” dimensionality of the problem: given too many neurons, the classifier may exhibit overfitting, given too

few, the classifier may not be able to distinguish the classes. Moreover, the internal representation of the data by the perceptron is generally not meaningful to humans, therefore checking the internal consistency of the model becomes very hard [8].

Support Vector Machines largely avoid these problems. Conceptually, during the training phase the SVM finds the hyperplane that will divide the space in the clearest possible way: this means solving the optimization problem of finding the surface that will leave the widest gap between samples belonging to the different classes.

Instead of optimizing exclusively with respect to the distance, which may depend only on a few (possibly misclassified!) points and finding a *hard margin*, modern SVM training methods usually associate a classification score to each point and find a *soft margin* that takes into account more of the training set and is resilient to noise and outliers. A parameter, generally designated as C , controls the “softness” of the margin: a harder margin will classify correctly most (all, in the case of a traditional hard margin) of the points in the training set and risk overfitting. To select a good value for C , we can test many different values and use cross-validation: at each step, a different part of the training set is left out as a validation set and the performance of the classifier is evaluated using those points.

SVM classifiers can be particularly efficient because the mapping of points to the highly-dimensional space can be bypassed by using *kernel* functions. The key observation is that the absolute value of the coordinates is not important to classify a point: only its relative position with respect to other points (as expressed, for instance, by the inner product of their coordinates) must be known. *Kernel* functions directly compute the inner product without actually determining the coordinates of the points in the high-dimensionality space. Many different kernel functions can be used; in our case, a Gaussian *radial basis function* has been used:

$$e^{-(\gamma \|\mathbf{u}-\mathbf{v}\|)^2}$$

As suggested by the *libsvm* authors [20] the value of γ has been determined

with the same procedure used to determine C .

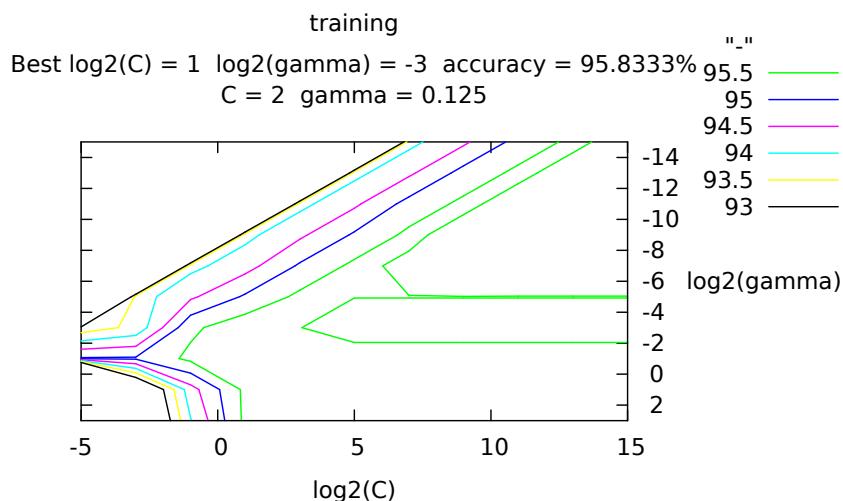


Figure 4.11: Grid search conducted by libsvm for the C and γ parameters, showing the correct classification rate as determined by cross-validation.

Due to timing and data availability constraints, classification has been performed only for spams. The training set employed for the grid search displayed in Figure 4.11 consists of approximately half of the *spam* and *alexa* datasets, with binary features and no scaling. The grid search determined $C = 2$ and $\gamma = 0.125$ to be good values for the SVM.

With these parameters, the classifier has a final accuracy rate of over 97.2% on the training set. The rest of the *spam* and *alexa* datasets (never seen by the classifier) has been used as a final testing set. On the testing set, the classifier achieved an accuracy rate of 96.5%, confirming the soundness of the procedure used.

Training set Accuracy: 97.2% (887 / 912)

Testing set Accuracy: 96.5% (965 / 1000)

Figure 4.12: Accuracy for the SVM classifier (Gaussian RBF kernel, $C = 2$, $\gamma = 0.125$).

Chapter 5

Building a scalable system

Image processing is a demanding task in terms of computing power. Since many of the promising heuristics presented in Chapter 3 are image-based, one may question their real-world applicability.

Inner-Eye, for instance, relies on OCR, a notoriously expensive technique. OCR-based spam detection systems have been proposed in the past [18, 7], but their implementation has always been limited to offline analysis. On-line detection systems operate on each item (emails, in the above cases) as it passes through the system; items are not allowed to reach users until they have been classified as benign. Latency is the most important property in this case, and the complexity of the detection system is severely constrained by it.

The target of *Inner-Eye* is not the on-line processing of web-pages or binaries. Given the real-time nature of web traffic, it is very hard to create detectors that do not impose an unacceptable performance tax on the network; deep-packet inspection systems, for example, typically require specialized hardware. However, *Inner-Eye* still aims to have good performance, both for testing and training needs (a large number of pages has been analyzed in the course of this work) and for future applications (as part of the Wepawet system, for instance). The next sections will elaborate on the performance limits of the current system and on how it handles a large volume of pages to analyze.

5.1 Latency

Network latency is a significant limiting factor for the performance of the current implementation of the system: besides pre-caching the page elements, no workarounds exist for this problem. Moreover, servers involved in scams often become unresponsive, both because of takedowns and because fraudsters abandon them: network timeouts are to be expected.

To better observe the page and allow initial JavaScripts events to fire, the capture system also includes an artificial delay of a couple of seconds before capturing the screenshot of the page. This delay is introduced *after* the browsers signals that it has finished loading the page; for slow servers this can be a relatively long time¹.

Finally, the commercial OCR solution currently used for *Inner-Eye* can take as long as 3 seconds to analyze a complex webpage.

5.2 Parallelism

Latency is only one of the factors that influence performance. Even if the analysis of a single webpage can take a long time, a good overall speed can still be maintained by processing several pages in parallel. Care must be taken to ensure that the performance benefits of a parallel infrastructure are fully realized and that no unnecessary delays and interdependencies are introduced, while still performing the amount of synchronization needed to maintain a consistent state.

Numerous parallel programming paradigms exist. Operating systems, for instance, support the concurrent execution of multiple tasks by virtualizing the CPU and maintaining the illusion of a dedicated CPU and memory space for each *process*; they also provide inter-process communication facilities so that multiple processes can cooperate in carrying out a complex task. Many platforms also allow programs to spawn multiple *threads* in the context of a

¹Browsers typically have relatively long timeouts. While capturing pages in the *alexa* dataset, it has been observed that many sites had very heavy homepages that took several seconds to load even on a very fast connection. Asian websites were particularly affected: latency and poor international connectivity issues probably exacerbate the problem.

single process: threads share the same memory space but execute on different virtual CPUs.

5.3 Virtualization

Threads and processes allow code to use the existing resources in a more efficient way. Besides being able to exploit multiple core systems, long operations (such as network communication) performed by one thread do not block others. Ideally, the processor is never idle: as one threads awaits the completion of a background task, others run and perform useful work. Indeed, purely event-base programming paradigms exist: in this model, a single thread of execution is shared by many small pieces of code, each triggered by a specific event (timers, user action, completion of network requests, . . .); JavaScript is a prominent language that adopts this model.

The first implementations of multiprocessing supported only *cooperative* multitasking: in order to share the CPU with other tasks, each process had to explicitly save its state and relinquish control to the Operating System before the CPU could be allocated to other processes. While the OS API could hide much of this complexity by automatically performing these operations whenever a system call was performed, programs still had to periodically poll the OS during long computations that would not otherwise require system calls. Moreover, a rogue or defective process could hang the entire system. Modern operating systems completely virtualize the CPU: all context switches are performed automatically and code can be actually written as if a fully dedicated CPU existed for each process.

One of the benefits provided by multiprocessing is isolation. Each process is started in a clean memory space and has no access to information outside it, nor can be influenced by other processes running on the same machine. OS-mediated interactions are an obvious exception to this rule, and the OS must be careful to enforce security policies correctly. Ideally, a malicious program started with very restrictive privileges would be innocuous and unable to influence the system in any way. Modern browsers run dangerous tasks in

specifically created, unprivileged, “sandboxed” processes, in an attempt to mitigate the impact of vulnerabilities that might be present in the code.

Sandboxing processes, however, is a complex task. Unless the code has been specifically written with sandboxing in mind, it will likely rely on the rich environment provided by the default OS permission model, which typically includes access to the network and to at least part of the filesystem. Once again, virtualization can help solve the problem. Through special hardware support or via dynamic recompilation, hypervisors can create multiple *virtual machines* on a single physical computer. Each machine runs its own OS and can be completely isolated from other VMs running on the same computer.

5.4 Exploiting virtualization

Inner-Eye exploits virtualization in many ways. First of all, each module runs in a separate process. This allows the system to take advantage of multi-core processors. Moreover, each stage of the analysis is only loosely coupled with the preceding and the following one. This allows easy distribution of the system over multiple virtual or physical machines.

The modules are interconnected as follows:

Submission

An automated submission system feeds URLs retrieved from Weapawet into the capture module. Submissions from pre-constructed lists (such as the Alexa topsites list) is also supported. Each URL is tagged with source information and becomes a *task*.

Task descriptions are passed to a central queue manager. RabbitMQ² has been chosen as the central message broker in the current implementation. RabbitMQ, together with the Celery³ helper library, offers a completely transparent implementation of the Advanced Message

²<http://www.rabbitmq.com/>

³<http://celeryproject.org/>

Queuing Protocol (AMQP). Manual submissions can optionally bypass the message queue and go straight to the capture module.

Capture

The first thing the capture script has to do is to obtain a unique identifier for the analysis task being conducted. This ID will be used to associate capture, OCR and processing information. Since URL data has to be saved by the capture module in the database, we take advantage of the write operation to let the database generate a unique identifier for us: the code does not need any synchronization beyond the one natively implemented by the database. (Databases such as MySQL provide `AUTO_INCREMENT` primary key fields and allow clients to atomically retrieve the ID associated with the most recently-performed insertion.)

The rest of the capture procedure can then be performed as described in Section 4.3. The capture script, the instrumented browser and the queue client are ready for deployment in separate virtual machines.

Besides communicating with the central database and the message queue, a common file storage for the captured data exists. In the current implementation a single file share is used, but (since the ID is already known at this point) it could be easily split over multiple storage spaces.

OCR

The commercial OCR solution currently used for *Inner-Eye* can automatically monitor a directory and perform OCR on new files as they are created. The OCR engine also runs in a separate virtual machine (a single one in the current implementation, due to license restrictions).

Processing

The completion of OCR triggers the final stage of processing. Features will be extracted from the data produced by the previous stage and inserted into the database; the classifier can also run and produce a verdict in this stage, if desired.

Notice how each stage can proceed at its own pace without interference from the others. Each stage can have one or multiple workers, implemented as processes on the same machine or on different machines.

5.5 Scalability

The features described in the previous paragraph are the key to building a *scalable* system. A scalable system can easily be adapted to increased load (e.g. more pages, or pages that take longer to analyze) by increasing the available resources.

Should the processing stage of *Inner-Eye* prove to be too slow to keep up with the previous stages, for instance, more processing workers can be created to share the workload.

During the course of this work, the slowest and most problematic stage has been the capture, mainly due to the intrinsically unpredictable latency introduced by network and rendering operations. Since the capture system is the only part of *Inner-Eye* exposed to potentially malicious code, running it in disposable virtual machines seemed a natural move.

The RabbitMQ central broker keeps track of the state of each worker VM. As tasks arrive, they are automatically distributed to the available workers. The system also monitors the performance of the workers and keeps statistics; fast workers also have the option to pre-fetch tasks from the queue so that they are always performing useful work.

5.6 Cloud-based workers

The scalability of the system comes from the ability to painlessly add new workers in order to handle load spikes or increased demand on the system. During the analysis of the *alexa* dataset, for instance, additional workers were used so that the system could continue analyzing URLs provided by Wepawet without delay.

Obviously, it would be impractical to acquire new physical machines and configure them from scratch every time additional capacity was needed. Moreover, if resources are added to cope with a load spike, it would be wasteful to permanently commit them to a system that has no ongoing need for them.

These considerations motivated developers to create the so-called *cloud computing* paradigm: processing power is seen as a service that can be provided in a flexible way. For instance, customers can purchase CPU time and network resources from large services such as Amazon EC2; no advance reservation is needed, and users simply pay for what they use.

Cloud computing services are usually provided by a pool of physical machines that run several virtual machine *instances*. A central manager keeps track of the resources in current use, keeps logs and accounting and allocates pool-wide shared resources such as IP addresses. Thanks to system virtualization, the manager can move instances among different physical machines, spawn new ones if requested, etcetera.

Instances are created from stored system images. A system image captures the state of a virtual machine; generally, only the hard drive contents are needed. When the user requests a new instance, the cloud manager creates a new virtual machine with the requested features (amount of RAM, number of processors, etc.) and a copy of the image as hard drive. The VM will then boot up and reconfigure itself with the new name and IP address assigned by the cloud manager: after this, the machine is ready to perform useful work for the project.

A system image has been constructed for the capture module of *Inner-Eye*. The workers automatically update their own system software, connect to the centralized file store, fetch the updated code for the capture system and register themselves with the queue manager. Configuration changes can be made without modifying the base image thanks to Puppet⁴, an event-based remote configuration tool. A secure shell is also exposed by the worker in case manual monitoring is needed.

⁴<http://puppetlabs.com>

The workers have been deployed on a locally-administered cloud managed by Eucalyptus ⁵, an open-source implementation of the Amazon EC2 platform, where they shared processing resources with several other projects.

⁵<http://open.eucalyptus.com/>

Chapter 6

Conclusions

By exploring the threats posed to users by modern malware authors and fraudsters, this work has revealed a trend towards the exploitation of the “human factor” in computing. In the first chapters, we have seen how scammers can easily build profitable malware that escapes common existing detection methods.

Some possible detection methods for computer frauds have been explored, with a particular focus on rogue AV and pharmacy scams. As information is mainly presented visually, image-based techniques have been identified as the most promising way to approach the problem. Instead of succumbing to obfuscation tricks, these methods are often able to leverage them to improve their efficacy.

In particular, an obfuscation-sensitive method based on a list of trigger words has been chosen as the basis of the *Inner-Eye* detection system. A complete analysis system has been implemented that is safe, accurate and efficient in capturing and analyzing web pages. The system’s classifier, based on a Support Vector Machine, has been trained to detect pharmacy scams with very good accuracy.

As a final remark, it should be restated that while eliminating all computer frauds may not be a feasible goal, the bar is currently very low for fraudsters to make a profit from their illicit activities. Forcing criminals to develop more complex and more costly systems could put many of them out

of business and, by removing some of the economic incentives, deter potential scammers from dedicating themselves to these activities.

6.1 Future work

While the current implementation of *Inner-Eye* targets web-based pharmacy scams, its design should lend itself to the addition of more detection capabilities.

6.1.1 Detecting fake AV webpages

A very limited amount of rogue antivirus webpages was available for submission to the system, too few to meaningfully train an automated classifier. Finding more (and more varied) web-based rogue antivirus scams is the prerequisite for the extension of *Inner-Eye* to target this type of fraud. A simple text-based rogue antivirus detector has been recently added to the Wepawet system and could provide useful data for this task.

The current design of the system should not need particular modifications to handle this task.

6.1.2 Detecting fake AV binaries

As described in Section 4.3.2 on page 65, an almost complete capture module for rogue antivirus executables has been implemented during this work.

As binary obfuscation techniques differ substantially from those used for malicious web pages, the heuristics employed by *Inner-Eye* will need amending. Simpler, purely text-based detection methods could be employed. More interesting detection avenues could be opened by the application of the techniques proposed in the next paragraphs.

An additional challenge that arises when dealing with rogue AV binaries is that real, legitimate, security products often use questionable marketing techniques such as exaggerating the risks faced by the user if she uses only the free functionality-limited version of an antivirus, or if an integrated security

suite (that may include additional elements such as firewalls and backup services) is not in use. Running an antivirus on a clean machine should, in theory, easily allow discrimination of real and fake antiviruses; unfortunately, those practices complicate the matter and create ample opportunity for false positives.

6.1.3 Detecting graphical elements

One of the most promising detection heuristics described in Chapter 3 involves the recognition of “scary” graphical elements. This technique is expected to be relatively powerful and equally applicable for the detection of binary and web-based rogue antiviruses. However, operating on low-resolution images has proven hard.

Possible ways to tackle the issue include a two-phase method in which a simple image-wide algorithm (such as line or edge detection) is applied first. Heuristics applied to the result of the first phase could be able to point to a number of candidate positions for the sensitive graphical elements. If the number of possible positions is low enough, it could be feasible to use simple matching methods (possibly repeated to compensate for scale and offset mismatches) that would be too complex to run on the entire image.

Color information is intuitively expected to play a big part in any object recognition algorithm that is suited for this task.

6.1.4 Improvements to the current heuristic

The current approach based on sensitive words has proved reasonably accurate for the detection of pharmacy scams. Nevertheless, some improvements could be necessary.

In particular, a traditional distance-based n -gram based text classifier could be useful to improve accuracy and supplement the manually constructed list of sensitive words.

6.1.5 Other improvements

Heuristics based on network features have not been implemented in *Inner-Eye*, even though the current system is already equipped to check for the Alexa rank of sites. While network-related heuristics have proven useful for the detection of on-line scams, they have been disregarded in favor of more general approaches. Nevertheless, they could provide very useful data for the classifier.

While implementing the capture module for Windows executables, it has become apparent that modern malware can expect (and even require) user interaction to activate its malicious characteristics. It could be useful to simulate similar interactions with web pages, in the hope to expose more content to the capture module. A possible approach would be to detect (through introspection) the click-sensitive elements in the web page and randomly simulate mouse events for some of them. A more sophisticated approach could be to model which areas of the page would be more likely to be clicked by the user. Additionally, analysis of the JavaScript control flow and event graph could be used to identify events that would trigger interesting behavior; this would allow the system to simulate only the most promising interactions.

Bibliography

- [1] Microsoft Security Intelligence Report, Volume 11. <http://www.microsoft.com/security/sir/>, 2011.
- [2] H.B. Aradhye, G.K. Myers, and J.A. Herson. Image analysis for efficient categorization of image-based spam e-mail. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 914–918. IEEE, 2005.
- [3] P. Azad, T. Asfour, and R. Dillmann. Combining harris interest points and the SIFT descriptor for fast scale-invariant object recognition. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4275–4280. IEEE, 2009.
- [4] K. Baumgartner. Malvertizing continued - spotify's ad networks outed. http://www.securelist.com/en/blog/6158/Malvertizing_Continued_Spotify_s_Ad_Networks_Outed, 2011.
- [5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.
- [6] U. Bayer, C. Kruegel, and E. Kirda. Anubis: Analyzing unknown binaries, 2009.
- [7] A. Bergholz, G. Paass, F. Reichartz, S. Strobel, M.F. Moens, and B. Witten. Detecting known and new salting tricks in unwanted emails. In *Fifth Conference on Email and Anti-Spam, CEAS*, pages 21–22, 2008.
- [8] M. Berthold and D.J. Hand. *Intelligent data analysis: an introduction*. Springer Verlag, 2003.

- [9] E. Blanzieri and A. Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.
- [10] R. Böhme and T. Holz. The effect of stock spam on financial markets. *Information Security*, (June):1–24, 2006.
- [11] W.B. Cavnar and J.M. Trenkle. N-gram-based text categorization. In *Proceedings of the Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*, pages 161–175, 1994.
- [12] C.C. Chang and C.J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.
- [13] T.M. Chen and J. Robert. *Statistical methods in computer security*, chapter The Evolution of Viruses and Worms, page 265. CRC Press, 2005.
- [14] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J.C. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, 2004.
- [15] M. Cova, S. Ford, C. Kruegel, and G. Vigna. Wepawet, 2009.
- [16] M. Cova, C. Leita, O. Thonnard, A. Keromytis, and M. Dacier. Gone rogue: An analysis of rogue security software campaigns. In *Computer Network Defense (EC2ND), 2009 European Conference on*, pages 1–3. IEEE, 2009.
- [17] A. Decker, D. Sancho, L. Kharouni, M. Goncharov, and R. McArdle. A study of the pushdo / cutwail botnet. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/study_of_pushdo.pdf, 2009.
- [18] G. Fumera, I. Pillai, and F. Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7:2699–2720, 2006.

- [19] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice-Hall, 1999.
- [20] C.W. Hsu, C.C. Chang, C.J. Lin, et al. A practical guide to support vector classification, 2003.
- [21] BitTorrent Inc. Security Incident (Updated 9/14). <http://blog.bittorrent.com/2011/09/13/security-incident/>, 2011.
- [22] Google Inc. Google Safebrowsing API. <http://code.google.com/apis/safebrowsing/>.
- [23] Google Inc. Google Webmaster guidelines. <http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=35769>, 2011.
- [24] E. Kirda and C. Kruegel. Protecting users against phishing attacks. *The Computer Journal*, 49(5):554–561, 2006.
- [25] C. Lioma, M.F. Moens, J.C. Gomez, J. De Beer, A. Bergholz, G. Paass, and P. Horkan. *Anticipating Hidden Text Salting in Emails*. 2008.
- [26] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999.
- [27] E. Mendoza, J. Manuel, J.D. Torre, and R.D. Paz. Unmasking FAKEAV. <http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/unmaskingfakeav.pdf>.
- [28] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th international conference on World Wide Web*, pages 83–92. ACM, 2006.
- [29] H. O’Dea. The modern rogue-malware with a face. In *Virus Bulletin Conference*, 2009.
- [30] Nigerian Code of Law. Nigerian criminal code act, chapter 38.

- [31] M.A. Rajab, L. Ballard, P. Mavrommatis, N. Provos, and X. Zhao. The nocebo effect on the web: an analysis of fake anti-virus distribution. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.
- [32] J. Segura. Fake AV network steps up its game with rootkit. <http://blogs.paretologic.com/malwarediaries/index.php/2011/08/19/fake-av-network-steps-up-its-game-with-rootkit/>, 2011.
- [33] J. Sobrier. The most common obfuscation techniques in fake AV pages. <http://research.zscaler.com/2011/05/most-common-obfuscation-techniques-in.html>, 2011.
- [34] Steven K. XyliBox Blog. <http://xylibox.blogspot.com/>.
- [35] B. Stone-Gross, R. Abman, R.A. Kemmerer, C. Kruegel, D.G. Steigerwald, and G. Vigna. The underground economy of fake antivirus software. *Proc.(online) WEIS*, 2011.
- [36] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.
- [37] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna. *The Underground Economy of Spam: A Botmaster’s Perspective of Coordinating Large-Scale Spam Campaigns*, page 4. USENIX, 2011.
- [38] K. Taghva, R. Beckley, and J. Coombs. The effects of OCR error on the extraction of private information. In *Document Analysis Systems*, pages 348–357, 2006.
- [39] J. Von Neumann and A.W. Burks. Theory of self-reproducing automata. 1966.

- [40] L. Wenyin, G. Huang, L. Xiaoyue, X. Deng, and Z. Min. Phishing web page detection. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 560–564. IEEE, 2005.
- [41] Andrew G West, Jian Chang, Krishna Venkatasubramanian, Oleg Sokolsky, and Insup Lee. Link spamming wikipedia for profit. In *CEAS 11 Proc of the 8th Annual Collaboration Electronic Messaging AntiAbuse and Spam Conference*, 2011(September):152–161.