UNIVERSITY OF CALIFORNIA, SAN DIEGO

Understanding the Role of Malicious PDFs in the Malware Ecosystem

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Moitrayee Gupta

2011

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGEMENTS

I would like to thank the following people whose contributions made this work possible: Mila Parkour, Stephan Chenette, Christian Kreibich and the researchers at Filex, for providing the collections of malicious PDFs that we examined; the members of the Trajectory team, especially Neha Chachra, Chris Grier, He Liu and Tristan Halvorson, for assisting with data retrieval and analysis; Professors Stefan Savage and Hovav Shacham, for serving on my thesis committee and providing valuable feedback on initial versions of this document; and finally, my advisor Professor Geoffrey Voelker, for his constant guidance, encouragement and support.

ABSTRACT OF THE THESIS

Understanding the Role of Malicious PDFs in the Malware Ecosystem

by

Moitrayee Gupta

Master of Science in Computer Science

University of California, San Diego, 2011

Professor Geoffrey M. Voelker, Chair

The Portable Document Format (PDF) is a widely used, cross-platform file format for document exchange. Several applications exist for parsing and rendering PDF documents, with Adobe's Acrobat Reader being the most widely used PDF reader. Starting in 2007, several vulnerabilities in Adobe Reader were discovered being exploited in the wild. PDF-based exploits continued to proliferate during 2008 and 2009, and, although recent security reports have noted a decline in the numbers of PDF-based malware in 2011, malicious PDFs are likely to continue to be a significant threat for the next few years, given the ubiquity of the PDF format and the existence of a large base of unpatched Adobe Reader installations.

In this work, we try to understand the role played by malicious PDFs in the malware and spam ecosystems. We collect data from the execution of a set of about 11,000 malicious PDFs obtained from various sources. We find a correlation between the age of a vulnerability and the number of PDFs exploiting that vulnerability. We also find differences in behavior depending on the distribution vector used. Looking at the final payload of the malicious PDFs, we find that some known pay-per-install services seem to use malicious PDFs as an infection vector. Finally, we see a considerable overlap in malware-hosting domains contacted by malicious PDFs and spam-advertised domains seen in emails collected by various spam feeds, pointing to the use of both vectors for malware distribution.

1 Introduction

The Portable Document Format (PDF) is a cross-platform, open standard file format that is widely accepted as an industry standard for portable document exchange. A PDF file is a self-contained document that includes the actual content of the file as well as formatting and layout information. Several applications exist for parsing and rendering PDF documents, with Adobe's Acrobat Reader being the most widely used PDF reader.

Starting in 2007, several vulnerabilities in Adobe Reader were discovered being exploited in the wild. These exploits used maliciously-crafted PDF documents that, when opened with vulnerable versions of Adobe Reader, resulted in arbitrary code execution and eventual compromise of the host system. Compromised systems can be used for various different purposes, depending on the kind of malware installed on them. For instance, they could be used to send out a large number of email messages advertising different goods — counterfeit software or replicas, pharmaceutical products etc. These email messages are commonly classified as 'spam' and the compromised systems are said to be part of 'spamming botnets'. Several different kinds of botnets exist, with compromised systems being made to participate in activities ranging from click fraud to denial-of-service attacks. Compromised systems can also have other kinds of malware installed on them, such as key-loggers, backdoors or 'scareware', all of which aim at using different methods to monetize information collected from the users of the systems.

PDF-based exploits continued to proliferate during 2008 and 2009, with security reports estimating that PDF-based malware formed 80% of all malware seen by the end of 2009 [1]. New vulnerabilities were discovered in Adobe Reader during 2010 and 2011 as well, and several different types of PDF-based exploits were seen in the wild. Some of these attacks were 'zero-day' attacks, a term used to describe an exploit targeting a vulnerability that is unknown to the software vendor. Zero-day attacks are more potent because they usually succeed in exploiting a large number of systems in the time that it takes the software vendor to develop and distribute a patch for the targeted vulnerability. Though recent security reports have noted a decline in the numbers of PDF-based malware in 2011 [2], malicious PDFs are likely to continue to be a significant threat for the next few years, given the ubiquity of the PDF format and the existence of a large base of unpatched Adobe Reader installations [3].

The purpose of this work is to understand the role played by malicious PDFs in the overall malware ecosystem. Specifically, we focus on analysing how PDF-based malware fits into the current spam ecosystem, as described in the analysis of the spam value chain performed by the Trajectory project [4]. To this end, we collect data from the execution of a set of about 11,000 malicious PDFs obtained from various sources, and analyze the overlap with data about spam-advertised domains collected by the Trajectory project. We also analyze data collected from the malicious PDFs about the vulnerabilities being exploited and the malware intended to be the final payload of the PDF.

We observe a correlation between the age of a vulnerability and the number of PDFs exploiting that vulnerability in our collection of samples — we find more PDFs targeting older vulnerabilities than more recently discovered ones. One exception, however, is a large set of PDFs that exploit a recent vulnerability and also use specific techniques to bypass advanced defenses like DEP and ASLR, which presumably improves the infection rate. The malware executables dropped or downloaded by the malicious PDFs in our collection are predominantly classified as 'Generic Trojans', with a few of them being identified more specifically as password-stealers, backdoors or scareware. We find that some of the executables classified as ‘Generic Trojans’ are downloaders created by known ‘Pay-per-install’ providers [5].

We observe a significant overlap between malware-hosting domains contacted by malicious PDFs and spam-advertised domains seen in spam feeds used in the Trajectory project. We also find an overlap in the third-party services maintaining parked pages used by both spam-advertised domains and malware-hosting domains. These results highlight the sharing of Web infrastructure supporting these two activities, and the possible use of both malicious PDFs and email-based spam to advertise malware-hosting domains.

The remainder of this document is organized as follows. Section 2 gives an overview of related work. Section 3 gives a brief history of Adobe Reader vulnerabilities and describes six of the most commonly exploited ones. Section 4 describes the anatomy of a typical PDF exploit and the structure and content of a malicious PDF document. Section 5 describes our analysis methodology and experimental setup, and details the kind of data that we collected from the PDFs. Section 6 describes the results of the analysis performed on the collected data. Section 7 describes the sandboxing technology called Protected Mode that was recently introduced in Adobe Reader 10. Section 8 briefly lists out alternatives to Adobe Reader. We close with our conclusions in Section 9.

2 Related Work

The mechanics of PDF-based exploits and vulnerabilities are typically described, in great detail, on online security Web sites and blogs (e.g., [6], [7], [8], [9]). To the best of our knowledge, however, this report describes the first study that has been conducted on PDF-based malware with the purpose of understanding its role in the overall malware and spam ecosystems. Previous studies have focused on other aspects of the malware ecosystem that are relevant to the study of malicious PDFs. We mention a few such related papers in this section.

Wepawet [10] is a commonly used Web service that detects malicious JavaScript content in Web pages and files. In [11], the authors describe the approach followed by JSAND, the tool that is used by Wepawet to automatically detect and analyze malicious JavaScript. Since the majority of PDF-based exploits attack JavaScript vulnerabilities in Adobe Reader, Wepawet is able to analyze and identify most malicious PDFs.

Drive-by downloads are widely used as a vector for the dissemination of PDF-based malware, by targeting vulnerabilities in the Adobe Reader plugin present in Web browsers. The authors of [12], [13] and [14] perform in-depth studies of the threat posed by drive-by download attacks, and suggest mitigation techniques. In particular, the authors of [13] state that 24.7% of all exploits in the Mebroot drive-by download campaign that they studied made use of vulnerabilities in the Adobe Reader browser plugin.

In [15], the authors describe a study performed to analyze the life-cycle and network-level behavior of Web-based malware on infected systems. They identify different categories of network traffic sent by compromised hosts, and characterize the traffic exchanged between compromised hosts and command-and-control channels of various botnets. Another study, described in [5], focuses on analysing Pay-per-install services (PPI) that aim at ‘outsourcing’ the distribution of malware and infection of user systems to various third-parties. Using the information described in [5], we find that some known PPI providers are involved in the use of malicious PDFs as a vector for malware distribution.

3 Vulnerabilities in Adobe Reader

The most commonly used PDF reader — Adobe Reader — currently has 223 vulnerabilities listed in The Common Vulnerabilities and Exposures database [16], most of them discovered between 2007 and 2010. In this section, we describe some of the commonly targeted vulnerabilities using the assigned CVE numbers. Table 3.1 summarizes these commonly exploited vulnerabilities.

CVE-2007-5659. This CVE refers to a vulnerability in the JavaScript method *Collab.collectEmailInfo*, which is susceptible to buffer overflow attacks because it does not validate the length of input arguments. This could result in arbitrary code execution with specially designed input data.

CVE-2008-2992. This CVE refers to a vulnerability that exists in Adobe Reader's implementation of the JavaScript *util.printf* method, and is caused by a boundary error when parsing a format string argument containing a floating point specifier. The vulnerable piece of code reads in an input value but returns only the first 16 digits of the input, with the rest of it padded with zeroes. By passing in an input value that is sufficiently long, it is possible to overwrite memory locations in the process' address space and divert control flow to attacker-specified data.

CVE-2009-0927. This CVE refers to a vulnerability that exists in the *Collab.getIcon* method. A call to the *strncpy* function could result in a stack overflow because of improper bounds checking on the third input argument to *strncpy*,

Table 3.1: Commonly exploited Adobe Reader vulnerabilities

CVE Number	Description	Date	Affected Versions
CVE-2007-5659	Input validation bug in Collab.collectEmailInfo	10/2007	<8.1.2
CVE-2008-2992	Boundary error in util.printf	05/2008	<8.1.3
CVE-2009-0927	Input validation bug in Collab.getIcon	07/2008	<7.1.1, 8.1.3, 9.1
CVE-2009-4324	Use-after-free bug in doc.media.NewPlayer	12/2009	<8.2, 9.3
CVE-2010-0188	Integer overflow bug in libtiff library	02/2010	<8.2.1, 9.3.1
CVE-2010-2883	Input validation bug in cooltype.dll module	09/2010	<8.2.5, 9.4

which is the size of input to be copied.

CVE-2009-4324. This CVE refers to a use-after-free bug that exists in the *doc.media.NewPlayer* method in the Multimedia API and is triggered when a null argument is passed to the function. In some of the sample PDFs that used this vulnerability, the JavaScript code that triggered the exploit was contained within a compressed *zlib* stream object, making detection harder.

CVE-2010-0188. This CVE refers to an integer overflow vulnerability in the *libtiff* library that is used for rendering TIFF images. When a TIFF image is included in a PDF, a field called *DotRange* containing a *count* value is defined, which is then used by other modules. In particular, the *AcroForm.api* plugin uses this value without sufficient sanitization, which could result in stack smashing and attacker-specified code execution. A similar integer overflow vulnerability was reported in 2006 in the same library (CVE-2006-3459).

CVE-2010-2883. This CVE refers to a vulnerability in the *cooltype.dll* module which is used when parsing data describing certain TrueType (TTF) fonts. The TTF font specification requires a table that describes the font, which includes a null-terminated string called $UniqueName$. The vulnerable code in the *cooltype.dll* module uses the value specified in this field without checking for null-termination, thus making a buffer overflow possible.

4 Anatomy of a PDF Exploit

In this section, we describe how a typical PDF exploit works, based on the behavior observed in a majority of the samples that we examined.

4.1 Infection Vectors

The two most common vectors used for the dissemination of malicious PDFs are email and drive-by downloads.

4.1.1 Email

Emails with malicious PDFs as attachments are sent out to specific recipients, with the body of the email containing a socially engineered message designed to lure the recipient into opening the attached file. Some instances observed have had the following type of content:

1. ‘Breaking news’ about current political or social events
2. Foreign policy and international relations missives from various countries
3. Notification about an award
4. Agenda for a meeting, event or interview
5. Lessons or tips from sporting professionals

This vector necessarily requires user interaction, as the malicious attachment has to be downloaded and opened by the recipient, using a software reader that is vulnerable to the specific exploit. In addition, spam filters and anti-virus scanners present in email clients could prevent the message from being delivered to the recipient's inbox, possibly reducing the effectiveness of this vector.

4.1.2 Drive-by downloads

A drive-by download [14] is characterized by an absence of user knowledge and/or consent about software being downloaded onto his system. Common drive-by downloads work by exploiting a vulnerability present in one of the browser plugins or in the browser itself. Since many browser plugins are configured to be automatically invoked whenever content of a specific type is seen, this infection method does not require explicit user interaction, apart from getting the user to visit the Web site that is serving the drive-by download.

An attacker can serve up a drive-by download in multiple ways. He could set up a malicious Web site serving the malware, and then use SEO or social engineering techniques to get users to visit his Web site. Alternatively, he could use existing Web sites to serve the malware — for instance, he could buy ad space on a Web site and insert an advertisement containing malicious code that serves up the drive-by download. He could also inject malicious code into legitimate Web sites by exploiting security flaws in web applications used on the web server. Using existing Web sites to serve up drive-by downloads ensures a higher infection rate, especially when the Web site has an established visitor base.

As described in [14], drive-by downloads operate by first checking the user's browser for information about the type and version of the browser, the browser plugins and the operating system. This step is called 'fingerprinting'. Based on the fingerprint obtained, the malicious code serves up an exploit that is designed to take advantage of specific vulnerabilities present in the above-mentioned components. Most of the malicious PDFs that we surveyed exploited vulnerabilities present in the Adobe Reader browser plugin.

4.2 Document Structure

A malicious PDF document typically has the following parts:

4.2.1 Code that exploits the targeted vulnerability

In the case of PDFs exploiting buffer or integer overflow vulnerabilities, this is the part of the PDF that constructs a specially crafted malicious input designed to cause the overflow, followed by a call to the vulnerable method(s) with the malicious input. The purpose of the overflow is to overwrite specific memory locations in the process' address space, with the aim of diverting control flow to a malicious, attacker-specified code block. These malicious code blocks are placed on the process' address space using techniques like heap spraying, as described below.

4.2.2 Heap spray code

Heap spraying is the term used to describe the technique of allocating large chunks of memory on a process' heap and filling it with specific data. Malicious PDFs employ heap spraying to place attacker-specified code blocks in various locations in the process' heap. These code blocks contain malicious code, called *shellcode*, which performs various unauthorized actions on the host system. Since the size of the shellcode is fairly small compared to the size of the process' address space, the attacker usually distributes multiple blocks of shellcode throughout the heap, to improve the chances of execution. Another commonly used technique is to include *NOP sleds* in each code block. A NOP sled is simply a sequence of NOP instructions preceding the shellcode in each code block. By including a large NOP sled in a code block, the attacker can increase the probability of execution of the shellcode. All he has to do now is divert control flow to any part of a NOP sled, and the control flow will 'slide' down the NOP sled to the beginning of the shellcode in that code block. Malicious PDFs employing this technique usually have a small JavaScript code block that performs the heap spraying prior to executing the exploit code.

4.2.3 Shellcode

Shellcode is the term used to describe malicious blocks of code that perform unauthorized actions on a host machine. The shellcode in malicious PDFs usually performs some or all of the following actions:

1. Extracts an executable file that is embedded within the code of the malicious PDF, and drops it to a standard location on the host system. We call this executable the ‘Stage 1 malware’. The shellcode then executes this malware, which in turn could try to download additional malware or install its payload on the system (in the case of key-loggers, rootkits or other kinds of trojans).

2. Connects to a remote server and downloads malware directly. This is different from the behavior described above in that the network connection is made by the malicious PDF itself, not by a dropped executable. If the network connection to the remote server fails or if the server does not respond, there are no executables or other files left behind by the PDF on the system. If the remote connection succeeds, additional malware is downloaded onto the host system and is executed by the shellcode. We call this downloaded malware the ‘Stage 2 malware’.

3. Extracts another PDF from code embedded inside the malicious PDF, and opens the new PDF with the same instance of Reader or Acrobat. This second PDF serves as a decoy and usually contains legitimate text and/or images to divert the user’s attention while the malware is executing on his system.

4. Performs system modifications to ensure the execution of the installed malware at system boot time, by adding entries into the Windows Startup menu or by adding keys to the Windows Registry.

4.2.4 Additional Features

Many of the malicious samples observed in the wild employed specific tricks to bypass detection and/or improve infection rates.

Use of JBIG2Decode filter. The PDF specification defines several different filters that can be applied to uncompress raw compressed data or to decode encoded data. The specification allows multiple layers of the same or different filter to be applied on a single stream in succession. Most malicious PDFs use several layers of filters to obfuscate data and make analysis more difficult. Some of the malicious PDFs observed in the wild exploit a flaw in the specification which allows a certain filter called JBIG2Decode to be applied to any object stream, as long as the object stream has been declared as a 2D monochromatic image in the document. Many anti-virus scanners fail to detect these PDFs [8] because of the unexpected use of this filter, since the specification states that the filter only decodes monochrome images but the content that is encoded in the malicious PDFs is a text stream containing the exploit code.

Code obfuscation. This is a common technique used by malware writers to make malware hard to read and analyze. There are several ways to obfuscate content with the PDF file format, one of which is to use encoding schemes and decoding filters, as described above. Encoding schemes like Base64 or ASCII85 [9] are supported by the PDF format, and can be used to transform sections of a PDF document into an unintelligible character stream that would require decoding to be analyzed. Other common obfuscation techniques include replacing ASCII characters with their hexadecimal or octal equivalents, using ‘eval’ and ‘unescape’ to construct calls to vulnerable functions at runtime, splitting up and storing the malicious code in different parts of the document, etc.

DEP/ASLR bypass. DEP or Data Execution Prevention is a security technique implemented in modern operating systems that aims at preventing code execution from areas of a process’ address space that are traditionally used only for data. By marking these areas of a process’ address space ‘non-executable’, DEP prevents the kind of exploits that work by placing malicious code on the process’ stack or heap and then executing it by diverting control flow to it. ASLR or Address Space Layout Randomization is a security technique that aims at making it harder for malware writers to predict addresses and locations in a process' address space. ASLR works by randomly selecting starting memory addresses for key components in a process' address space, such as the heap, the stack and the addresses at which system libraries are loaded. A subset of the PDF samples examined employed a specific trick to bypass the protection afforded by these mitigation techniques on Windows Vista and Windows 7 systems. DEP was defeated by the use of a technique called Return Oriented Programming [17], where already-loaded system modules with execute permission are used to create a chain of calls that results in attacker-specified code being executed. The samples used a module that does not support ASLR (icucnv.dll) and is therefore loaded into a fixed memory location in the process address space, thus defeating the protection afforded by stack randomization.

5 Methodology

To understand and characterize the behavior of PDF-based malware, we obtained samples of malicious PDFs from various sources, executed them in a sandboxed environment, and collected information about different aspects of their behavior. In this section, we describe our experimental setup and the kind of information that we collected from our samples.

5.1 Samples of malicious PDFs

We obtained samples from two main sources — the *contagiodump* malware collection [6] and the *Filex* malware collection [18]. From [6], we obtained two sets of malicious PDFs — one set distributed via email as attachments and the second set collected from Web sites serving them up as drive-by downloads. In addition to these two primary sources, we also obtained samples from two honeypot feeds that collected email attachments delivered to various accounts.

For each of these collections, we first removed duplicate files by sorting them according to their *md5sum* hashes. We then separated the malicious PDFs from the benign ones using malware scanning tools (as described in Section 6.1).

Table 5.1 gives a summary of all the sources and collections, with the number of malicious and benign PDFs in each collection.

Table 5.1: Summary of sources of malicious PDFs

Name	Source	Distribution vector	Unique PDFs	Malicious PDFs
Collection1	Contagiodump	Email	175	175
Collection2	Contagiodump	Drive-by download	10939	9816
Collection3	Filex	Email,drive-by download	1978	983
Collection4	Honeypot feeds	Email	323	8
Total			13415	10982

5.2 Experimental Setup

To examine the behavior of the malicious PDFs, we created a virtualized environment to run the samples in. We used VirtualBox (version 4.0.8) to create virtual images, each running Windows XP SP 3 with either Adobe Reader 8.0 or 9.0 installed. To decide which version of Adobe Reader would be vulnerable to the exploits used by the samples, we first identified the vulnerabilities being exploited by each sample using malware detection tools, as described in Section 6.1. We installed Cygwin (version 1.7.9) on the images and used shell scripts to automate the execution of the PDFs. We used Wireshark (version 1.4.4) to collect network traces, and common Unix utilities available in Cygwin to record other changes made on the images during the execution of the PDFs.

For each PDF sample, our automation script performed the following steps:

1. Created a fresh virtual image from a clean base image, and copied over required files
2. Took a snapshot of the filesystem on the image prior to running the malicious sample
3. Started a network trace using Wireshark
4. Opened the malicious sample using a specific version of Adobe Reader
5. Closed Reader after 120 seconds, and stopped the network trace

6. Scanned specific locations in the filesystem for dropped or downloaded files
7. Checked for new entries added to the Windows startup menu

While performing this automation on a random subset of samples, we observed that most of the samples were not completely active — while the malicious code in the sample did try to connect to a server and request for additional malware, there was no response from the server for a large majority of the samples. In some cases, the domain names were no longer registered, while in others, the server contacted did not respond or indicated that the resource being requested for was no longer available. Based on this behavior and considering the short lifetime of malware in general, we focused on using the information obtained from the sample's network trace to characterize its behavior, as described in the next section.

The filesystem locations scanned in step 6 were compiled based on the observed behavior of a random subset of samples. We found that the samples in this subset dropped files in certain specific locations on the host system, such as the current working directory, the 'Temp' directory, the 'Application Data' directory, etc. We cross-checked this with accounts of malicious PDF behavior documented on different security Web sites. However, there does still remain a small possibility of some of the examined PDFs dropping files in uncommon locations that were not checked by our automation script.

5.3 Information collected from network traces

A typical malicious PDF opened with a vulnerable reader behaves in the following way:

1. The malicious code in the PDF performs the exploit and the embedded shellcode is executed on the host system.
2. Depending on the kind of PDF, the shellcode either extracts an executable file that is embedded within the PDF, or requests for and downloads an executable file from a remote server.

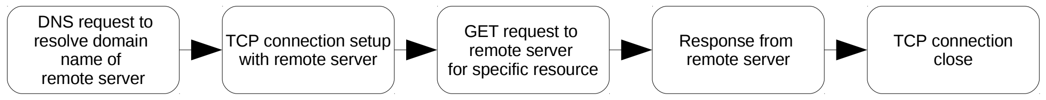
DNS request to resolve domain name of remote server
TCP connection setup with remote server
GET request to remote server for specific resource
Response from remote server
TCP connection close

Figure 5.1: Network flow pattern generated by malicious PDF

3. The dropped or downloaded file from step 2 is executed on the host system, and either downloads additional malware from the same or a different remote server, or installs its payload on the host system.

The network traffic seen on the host system during the execution of a malicious PDF usually has the pattern shown in Figure 5.1.

From the network traces collected during the execution of the samples, we extracted all the domain names from the DNS requests and used them to check for an overlap with known spam-advertised domains. We also extracted tuples of the form {*domain name, resource name*} and used them to look up information about the kind of malware that was intended to be the final payload of the malicious PDF. Many of the samples that requested for malware from a remote server returned a generic parking page in step 4 of Figure 5.1. We extracted the HTML content from these pages and checked for an overlap with parking pages displayed by known spam-advertised domains. The results obtained from analysing these data sets are described in the next section.

6 Results

In this section, we describe the results obtained from clustering and mapping the information that we collected from the malware samples.

6.1 CVE information

The first set of data that we analyzed was the CVE numbers of vulnerabilities being exploited by each PDF. We used the Wepawet [10] and Virustotal [19] public APIs to collect this information. For each PDF, we first checked for an existing report using the *md5sum* hash of the sample, and if a report did not exist, we submitted the sample for analysis and then checked the generated report. A subset of the PDFs were identified as being benign by both tools; we excluded this subset from further consideration. We expect that the use of two different tools for identification reduced the number of false negatives generated during this process.

Figure 6.1 shows the categorization of samples grouped by the exploit(s) present in the sample. Each bar corresponds to a CVE number assigned to a specific vulnerability, and the height of the bar (in logarithmic scale) represents the number of samples in our collection that contained an exploit for that vulnerability. The bar labelled 'Undefined CVE' represents all the PDFs that were identified as being malicious by at least one of the tools, but did not have an assigned CVE number in either of the reports. The bar labelled 'Other' represents all the PDFs that exploited relatively uncommon vulnerabilities, and are all grouped together for ease of representation.

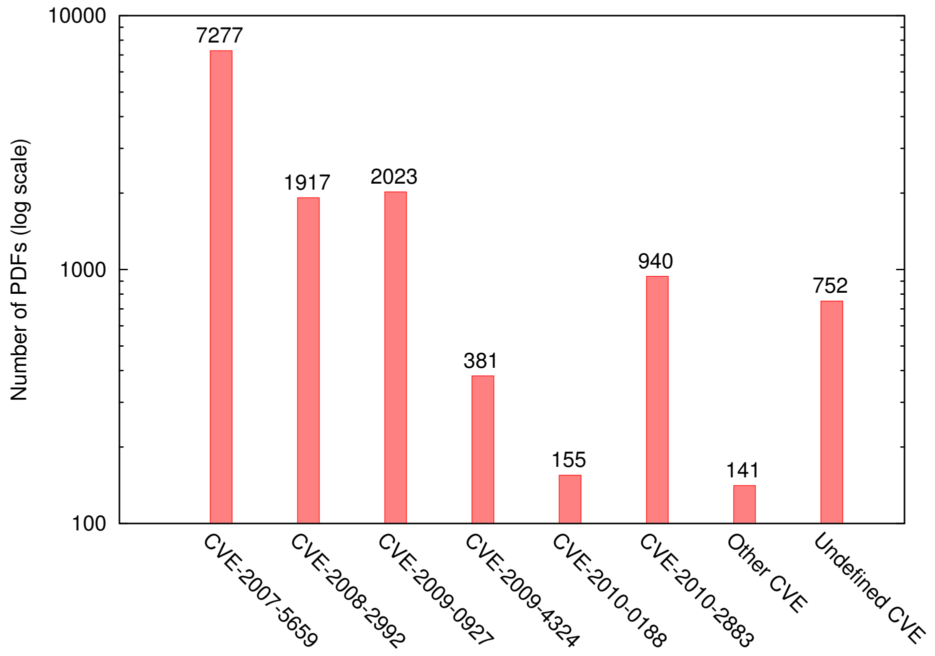

Figure 6.1: Count of malicious PDFs by CVE number

As seen in the graph, there are 6 vulnerabilities that are most widely exploited by the PDFs in our collection of samples. These vulnerabilities (and the corresponding CVEs) are described in Section 3. All the vulnerabilities, with the exception of one, exploit buffer overflow bugs present in various modules of Adobe Reader.

From the graph, it appears that our collection of samples had a larger number of PDFs with exploits for older vulnerabilities (CVE-2007-5659, CVE-2008-2992), and fewer PDFs exploiting newer vulnerabilities (CVE-2010-0188). This is in accordance with observations made by security analysts of a sharp rise in the numbers of malicious PDFs seen in the wild between 2007 and 2009, when vulnerabilities in Adobe Reader started being discovered very frequently. As security researchers and anti-virus vendors became more aware of the nature of these exploits, detection rates improved and patches for vulnerabilities were released faster, as a result of which newer exploits were probably less successful at infecting systems and were, therefore, used less. Thus, the presence of a large number of PDFs exploiting older vulnerabilities could be an indication of the ages of the samples in our collections. However, security vendors have also observed that the majority of systems running Adobe Reader continue to remain unpatched, even when critical security updates are released [3]. This means that attackers can continue to rely on vulnerabilities even after patches have been released for them, presumably because infection rates on older, unpatched systems continue to be satisfactorily high. The absence of security techniques like DEP and ASLR on older systems also increases the chances of an exploit being successful. In our collection of samples, we found several examples of recent PDFs exploiting older vulnerabilities. For instance, in $Collection1$, some of the PDFs sent as email attachments during 2010 were found exploiting CVE-2007-5659, and some PDFs sent during 2011 were found exploiting CVE-2009-4324 [6].

One notable exception in the graph is CVE-2010-2883, with our collection of samples containing a high number of PDFs exploiting this vulnerability when compared to contemporary vulnerabilities. This is most likely due to the fact that the PDFs targeting this vulnerability also used a specific technique to bypass DEP and ASLR (by using ROP and a DLL used by Adobe Reader that does not have ASLR enabled, as described in Section 4.2.4). Given the possibility of a higher infection rate because of the DEP and ASLR bypass, it is perhaps not surprising that higher numbers of this specific exploit were seen in the wild, compared to other vulnerabilities that were discovered around the same time.

Another interesting observation is that many of the PDF-malware samples in our collection contained exploits for more than one vulnerability. These multi-exploit PDFs contained exploit code for up to 3 vulnerabilities targeting different versions of Adobe Reader, improving the chances of a successful infection. The presence of multiple exploits in a malicious PDF could be an indicator that the PDF was generated by one of the many *exploit packs* that are used by Web sites distributing drive-by downloads. As described in [7] and [20], exploit packs

Table 6.1: Activity breakdown of a subset of malicious PDFs

Type of traffic	# of samples	% of samples
No network traffic	626	24.8%
Domain name unresolvable	1142	45.2%
No response from server	258	10.2%
Resource unavailable	234	9.3%
Server returns parked page	233	9.2%
Valid server response	36	1.4%
Total	2526	100%

are toolkits that can be installed on Web servers and be configured to generate exploits that target selected vulnerabilities in a visitor's Web browser or browser plugins. The exploit packs that offer PDF-based exploits target several of the most commonly exploited Adobe Reader CVEs. For instance, the Eleonore exploit pack examined in [7] targets CVE-2007-5659, CVE-2008-2992 and CVE-2009-0927, and generates malicious PDFs containing exploit code for all the three vulnerabilities. We found that 8% of the samples in $Collection1$, 14% of the samples in $Collection2$ and 27% of the samples in $Collection3$ were multi-exploit PDFs. In Figure 6.1, these PDFs were counted once for each vulnerability that they targeted.

6.2 Activity breakdown

For a subset of the malicious PDFs in our collection, we analyzed the network traces and identified the different kinds of network activities seen. Table 6.1 lists out the different activities and the number and percentage of samples for each kind.

The majority of samples (45.2%) in the subset examined tried to contact domains that were no longer registered. For 10.2% of the samples, the server contacted did not complete the TCP handshake. For 9.3% of the samples, the server contacted returned an error code. The most commonly seen codes were 404 (Not Found) and 301 (Moved Permanently). 9.2% of the samples had generic parked pages sent back to them by the contacted servers. For 1.4% of the samples, the server returned a valid response — these were seen to be executable files in a few cases, but most of them were '200 OK' responses with no content in the message body. 24.8% of the samples did not perform any network activity at all, indicating that either the exploit code in the PDF was unsuccessful, or that the final payload was contained in the PDF itself and did not require to be downloaded from a remote server.

While this data describes the network activity of only a subset of the samples examined, it is representative of the behavior observed in our collection of samples as a whole. The large majority of samples in our collection that were supposed to download malware from remote servers were unable to successfully do so. While this is perhaps not too surprising for the older samples, some of the newer PDFs were also found to be inoperable just days after being found in the wild. It remains an open question on how long malicious PDFs remain viable and what their typical window of operability is.

6.3 Analysis of Stage 1 and Stage 2 malware

As described in Section 4.2, a malicious PDF usually drops and/or downloads additional malware during its execution. The first piece of malware that is usually extracted from the malicious PDF itself and dropped to a standard location on the host system is termed 'Stage 1 malware'. Additional malware that may subsequently be downloaded from remote servers, either by the Stage 1 malware or by the malicious PDF, is termed 'Stage 2 malware'.

During the execution of our PDF samples, we observed that most PDFs from $Collection1$ exhibited the two-stage behavior, where Stage 1 malware is first extracted and dropped from the PDF, followed by the download of Stage 2 malware from a remote server. In contrast, most PDFs from $Collection2$ and $Collection3$ did not drop any Stage 1 malware — the PDF directly attempted to contact a remote server and download the Stage 2 malware from it. This could be an indication

Table 6.2: Malware Categories

Category Name	Description
Trojan Downloader	Downloads malware from remote servers
Trojan Dropper	Drops and installs malware onto user system
Generic Trojan	Shows suspicious behavior, performs unauthorized actions on system
Backdoor	Provides a remote attacker unauthorized access to system
Info Stealer	Collects and sends user/system information to remote attacker
Fake AV	Pretends to be anti-virus software, shows warning messages on system
Ransom Trojan	Denies access to system until ‘ransom’ is paid
Worm	Infects host system and other systems on network

of the source of a PDF — the ones that are created for targeted distribution via email (*Collection*1) behave differently from the ones that are created by exploit packs for distribution via drive-by downloads (*Collection*2). However, it seems counter-intuitive that the malicious PDFs created for distribution via email would have malicious executables embedded in the PDF content, since this would make the PDFs larger in size and more likely to be detected and blocked by email anti-virus and spam filters. One possibility is that these PDFs were distributed by PPI providers and affiliates [5] and hence had to contain an embedded downloader created by the PPI provider, which would then connect to a remote malware server hosting malware binaries provided by the client. In contrast, the exploit pack PDFs that did not display the two-stage behavior were more generic in design, and could thus be used to deliver a wider range of malware, since the actual payload was not part of the document content and could be downloaded from any remote server.

We used Virustotal [19] and Anubis [21] to categorize the Stage 1 and Stage 2 malware executables from our PDF samples. Since each AV vendor uses its own heuristics and names for identifying different kinds of malware, we created 8 general categories of malware and placed each identified file in one of the categories, based on the labels given to the file by the AV vendors. Table 6.2 lists the category names and a brief description of each category.

6.3.1 Stage 1 malware

We collected and analyzed some of the Stage 1 malware files that were dropped by the malicious PDF samples during their execution. As mentioned above, most of these dropped files came from the PDFs in $Collection1$. From a total of 150 files, 53 of them were identified by at least one anti-virus vendor on Virustotal. Columns 2 and 3 in Table 6.3 list the number and percentage of files that were found in each category. Most of the files were classified 'Generic Trojan', which includes generally suspicious behavior such as adding entries into the system registry and making outbound network connections.

6.3.2 Stage 2 malware

As mentioned in Section 5.2, most of the samples in our collection of PDFs did not receive the Stage 2 malware requested from the remote servers. The domain names were either unresolvable, or the servers did not respond, or the resource requested for was not found on the server. In many cases, the server returned a generic parking page in response to the request for malware. As a result, we had to look at other ways to analyze the malware that was intended to be the final payload of the malicious PDFs in our collection.

From the network requests made by the PDFs during their execution, we were able to extract complete URLs of the resources requested by many of the PDFs. This included the domain name of the remote server and the path of the resource being requested. Using this information, we searched several malware databases such as [22], [23], [24] and [25]. For each match found, we used the listed $md5sum$ or $sha1sum$ hash of the file, and looked up reports for the file on

Table 6.3: Categorization of Stage 1 and Stage 2 malware

Category Name	# of samples (Stage 1)	% of samples (Stage 1)	# of samples (Stage 2)	% of samples (Stage 2)
Trojan Downloader	7	13.2%	38	20.4%
Trojan Dropper	6	11.3%	24	12.9%
Generic Trojan	24	45.3%	55	29.6%
Backdoor	6	11.3%	17	9.1%
Info Stealer	3	5.7%	21	11.3%
Fake AV	0	0	17	9.1%
Ransom Trojan	0	0	3	1.6%
Worm	0	0	5	2.7%
Other	7	13.2%	6	3.2%
Total	53	100%	186	100%

Virustotal [19] and ThreatExpert [26]. Of a total of 657 unique URLs extracted from the network traces, we found matches for 327 of them in one of the listed databases, and 186 of the matches had hashes listed for the exact file. As in the case of the Stage 1 malware classification, we used the labels given to each file by different AV vendors to place the file in one of the categories listed in Table 6.2. Columns 4 and 5 in Table 6.3 list the number and percentage of samples in each category. While most of the Stage 2 malware samples were classified as being generic Trojans, downloaders or droppers, about 34% of them were identified more specifically as being information-stealing malware, backdoors or 'scareware'.

To gain more insight into the executables that were classified as generic Trojans, we compared some of the specific AV-vendor labels given to the executables with known malware families described in [27] and [5]. We found instances of 'Bredolab', 'Hiloti' and 'Alureon' executables in our collection of samples, all three of which are listed as known Downloader/PPI services in [27]. As described in [5], Pay-per-install (PPI) services are offered by specialized organizations that focus on infecting users' systems, and provide a way to effectively 'outsource' the distribution of malware. PPI providers accept requests from clients for a certain number of 'installs' of the client's programs on targeted hosts, and then arrange for

Table 6.4: Domain overlap percentages for each collection

Collection Name	Distribution Vector	Domain overlap
Collection1	Email	0.41%
Collection2	Drive-by downloads	81.93%
Collection3	Email, drive-by downloads	17.51%
Collection4	Email	0.13%

the required number of compromised systems. The actual infection of hosts could be performed by the PPI provider itself or be outsourced to another third-party, called an 'affiliate', that specializes in specific malware distribution methods. The PPI executables seen in our collection of samples are downloaders that are supposed to connect back to the PPI provider, download a specific 'client' executable and install it on the compromised host system. Depending on the kind of the 'client' malware installed, the compromised host would then be used in different ways, most commonly as a member of a botnet of some kind.

6.4 Overlap with Trajectory Data

To understand the role played by malicious PDFs in the spam ecosystem, we analyzed the overlap of specific data collected from our samples with data collected by the Trajectory project. As described in Section 5, we executed our samples in a virtualized environment and extracted information from the network traces.

6.4.1 Overlap of domains

The Trajectory database has a large corpus of URLs which are collected from different spam feeds. These URLs are then examined by crawlers that tag them based on the nature of their content. To check for an overlap with this dataset, we extracted all the domain names from the DNS requests seen in the network traces. These domain names point to the remote servers contacted by the malicious PDFs for additional malware (step 1 in Figure 5.1).

Table 6.5: Domain overlap percentages for each feed

Feed Name	Feed Source	Domain overlap
Feed A	MX honeypot	2.52%
Feed B	Seeded honey accounts	0.21%
Feed C	MX honeypot	0.77%
Feed D	Seeded honey accounts	0.93%
Feed X	MX honeypot	0.07%
Feed Y	Human identified	95.47%
Feed Z	MX honeypot	0.01%

We searched the Trajectory database for URLs with the same domain names as the domains contacted by the PDFs, and we found that out of a total of 1441 domains extracted from our set of malicious PDFs, 690 or 47.88% of them overlapped with domains listed in the Trajectory database. Table 6.4 contains details of the overlapping domains. Using these results, we can state that a significant percentage of the malicious PDFs from our collection attempted to contact and download malware from domains that were identified as spam-advertised domains appearing in various spam feeds. The collection with the most overlap is $Collection2$, which consists entirely of PDFs distributed via drive-by downloads. While this large overlap with $Collection2$ could be caused simply by the size of that collection relative to the other collections, it also indicates that that some of the same parties that use unsolicited email messages to advertise their domains also try to infect systems by using malicious PDFs distributed by drive-by downloads. This suggests that the Web infrastructure used to support spam-advertised scams is also used, to some extent, to support the infection and compromise of machines on the Internet.

We also identified the exact spam feeds that produced the overlapping domains, by mapping all the URLs containing the overlapping domains to the feeds they came from, and then counting the number of URLs obtained from each feed. Table 6.5 contains a description of each feed source, and lists the percentage of URLs that was mapped back to each feed. Most of the overlapping domains were seen in the content of Feed Y, which consists of URLs present in emails identified as spam by users.

Table 6.6: Tagged URLs by content type

Category	# of tagged URLs	% of tagged URLs
Known affiliate programs	4	0.03%
Spam-advertised goods	191	1.33%
Scheduled drugs	526	3.65%
Minimal	13131	91.24%
Parked pages	158	1.10%
Affiliate/User ID	367	2.55%
Other	15	0.10%
Total	14392	100%

As mentioned earlier, all the URLs present in the Trajectory database are also visited by a crawler to determine the kind of Web resource pointed to by the URL, and are tagged accordingly. To understand the nature of the domain overlap better, we extracted the tags from all the URLs in the database that appeared in the set of overlapping domains. Table 6.6 lists the type of content and the percentage of tagged URLs for each type.

The category 'Spam advertised goods' includes all domains that appeared to advertise or sell goods like software, replicas or pharmaceutical products. The category 'Known affiliate programs' refers to domains in the above category that clearly referenced one of the several identified affiliate programs listed in [4]. The category 'Scheduled drugs' includes all domains that appeared to advertise or sell regulated prescription drugs. The 'Parked pages' category includes domains that displayed generic parked pages, which indicates the availability of the domain for sale. The 'Affiliate/User ID' category includes domains that contained activity-tracking code (e.g., Google Analytics) in the content of the crawled pages. The category 'Other' included all domains that did not fall into one of the above described categories. The majority of URLs pointing to overlapping domains were classified as 'minimal', which means that these URLs pointed to Web resources that were either very simple HTML pages with not much content or resources that displayed an error when visited by the crawler.

There are two overall sets of tagged URLs seen in Table 6.6. The first set consists of URLs pointing to domains that were used both as a spam-advertised domain and a malware-hosting domain, presumably at different points in time. These are the domains that are identified by the crawler as having content of a specific kind (affiliate programs, goods, drugs). The second set consists of URLs pointing to domains contacted by malicious PDFs that are identified as 'minimal' by the crawler. These domains are most likely malware-hosting domains that used email to spread their malware, by sending out email messages containing URLs pointing directly to the malware executables. These emails were captured by the spam feeds and were then visited by the crawler, which classified the URLs as 'minimal' pages because they pointed directly to executable files and thus did not generate any HTML content.

Looking at the percentage and nature of the domain overlap as a whole, we find that 81.9% of overlapping domains are tagged as 'minimal' domains, which implies that they are malware-hosting domains that use at least two vectors to spread their malware and increase infection rates - malicious PDFs that connect to the servers and download malware from them, and email messages that are sent out to users containing direct links to the hosted malware.

6.4.2 Overlap of parked pages

During the execution of the malicious PDF samples, we observed that some of the samples returned HTTP responses containing HTML and JavaScript code from the remote server, instead of the malware that they requested (9.2% in Table 6.1. These responses appeared to be generic 'parked pages' returned by the domain, which indicate that the server was possibly active earlier, but was no longer hosting the resource requested for by the PDF. Parked pages indicate that the domain is registered but is up for sale. Parked domains can be monetized by the owner, the registrar or third-parties by hosting advertisements and showing sponsored content to visitors. Most of the parked pages returned to our set of

Table 6.7: Overlap of parked pages

Category	Parked page overlap
Affiliate/User ID	16.56%
Frame	3.55%
Parked	79.88%

PDFs contained advertisements, text and links to other domains.

The Trajectory database has a set of regular expressions collected from parked pages displayed by spam-advertised domains. We extracted the contents of the network responses containing parked pages that were sent to our malicious PDF samples, and ran the regular expressions on them to check for an overlap. Table 6.7 summarizes the percentage of overlapping parked pages and the tags given to them by the crawler.

Most of the overlapping parked pages were tagged as 'Parked' by the crawler, which confirms our identification of the contents of the network responses. A smaller percentage was tagged as 'Affiliate/User ID', indicating that the page contained some kind of activity-tracking code (e.g., Google Analytics). The category 'Frame' refers to pages that simply contained iframes pointing to other websites.

As in the case of the domain name overlap, there is the possibility that all the overlapping domains were malware-hosting domains that used both malicious PDFs and email spam to increase infection rates. However, in the specific case of parked domains, the overlap is more likely resulting from the reuse of domain names between malware-hosting domains and spam-advertised domains. The 'churn rate' of domain names in the malware ecosystem is known to be fairly high, and malware distributors are constantly on the lookout for new domains that are not on blacklists and have a steady stream of user traffic. Domain parking is one way to ensure that a high-value domain name can be sold and reused many times, by different players and for different purposes. The overlap in parked pages suggests that the same third-parties are involved in providing domains to support both spam-based advertising and malware-hosting activities.

7 Adobe Reader Protected Mode

The latest major release of Adobe Acrobat Reader in November 2010 - version 10 - introduced a new mitigation technique, called Adobe Reader Protected Mode, which implements a sandbox around the renderer component of Adobe Reader. The sandbox limits the actions that the renderer process is allowed to perform on its own, with all other actions requiring communication with a 'broker' running in a separate process. The broker process defines an API for interprocess communication, and uses a set of fixed policies to allow or disallow requests from the renderer process running in the sandbox.

The introduction of the sandboxing features reduces the exploitability of Adobe Reader significantly, because it increases the number of vulnerabilities that have to be found and exploited for a successful take-over of the system [28]. First, the malicious PDF has to find and exploit a vulnerability in the renderer component, allowing it to execute malicious code. However, this code would run inside the sandbox since the renderer process is confined to the sandbox. In order to be able to execute code that will affect the rest of the system, the malicious PDF has to find and exploit a vulnerability in the broker process' API that would allow it to break out of the sandbox. However, since the broker process does not run with administrator privileges, the malicious PDF would then have to find and exploit a privilege escalation vulnerability to have administrative control on the system. In addition, both the renderer and broker processes are executed with DEP and ASLR enabled on systems which support them, so a successful exploit would also have to bypass these protections.

While several new Adobe Reader vulnerabilities have been discovered since the release of version 10, none of the exploits seen in the wild have managed to successfully bypass all the additional security layers added by the Protected Mode sandboxing.

Other security-focused features that were added to Adobe Reader during 2010 and 2011 include the addition of an auto-updater and the selection of 'Update Automatically' as the default configuration of the auto-updater [29]. These changes were introduced, presumably, to reduce the number of unpatched and out-of-date installations that would not benefit from the addition of security measures like sandboxing. According to [30], it was found that, with the new version of the updater, users applied new updates roughly three times faster than they did with earlier versions of the updater.

While security reports have stated that there has been a decline in the number of PDF exploits seen towards the end of 2010 and in 2011, it is not yet clear if this can be attributed to the introduction of these security-focused changes in Adobe Reader, or if it is simply because other attack vectors, like Java, are proving to be more profitable and easier to exploit [2].

8 Alternate PDF Readers

In this section, we look at some alternatives to Adobe Acrobat Reader on Windows, and briefly summarize their track record of vulnerabilities and exploits.

Foxit PDF Reader. The Foxit Reader is a free PDF reader that has gained popularity as the best alternative to Adobe Reader. It has been described as being faster and more lightweight than Reader, and can be run as a standalone executable not requiring installation. The Foxit reader is also generally considered to be more secure than Adobe Reader, with only 14 vulnerabilities listed in the CVE database [16] since 2007. Of these, 3 vulnerabilities were found being exploited in the wild [31]. In our collection of samples, we found 68 malicious PDFs in total that exploited a Foxit vulnerability (CVE-2009-0837).

Google PDF Readers. Google has released two PDF readers in the recent past — the first is a PDF viewer within the Google Docs framework, and the second is a PDF reader built into Google Chrome. The Google Docs PDF viewer renders a PDF document as PNG images with overlays on the text to allow for highlighting. The Google Chrome PDF reader is a PDF reader built into Chrome that is run within the Chrome sandbox for better security and isolation. The earlier versions of Chrome had a Foxit-based PDF reader plugin. There are 8 CVEs listed in the CVE database [16] for Chrome vulnerabilities that can be exploited via malicious PDF documents. However, there have not been any reports so far about actual PDF-based exploits on Chrome that have managed to compromise user systems.

Other readers. Some of the other commonly used PDF readers are Sumatra, PDF-XChange Viewer and NitroPDF. All these readers have vulnerabilities that have been discovered, but none of them have been found being exploited in the wild so far.

To summarize, all the alternatives to Adobe Reader have one major advantage: since they are not as widely used, they are less targeted by attackers and therefore, presumably, safer. Though vulnerabilities have been discovered and disclosed for these readers as well, they are usually not exploited in the wild since it is much more profitable for attackers to focus on targeting Adobe Reader.

9 Conclusion

In this thesis, we described a study performed on a set of malicious PDF documents, with the aim of understanding better the nature of PDF-based malware and how it fits into the overall malware ecosystem. We described our experimental setup, methodology and the data collected from our collection of samples. We then described the analysis performed on the data, and explained in detail the nature of the overlap seen with data on spam-advertised domains collected by the Trajectory project. We found that about 48% of the malware-hosting domains contacted by our collection of malicious PDFs also appear in the Trajectory spam feeds. This points to a possible reuse of domains names, as well as the use of two different distribution vectors — malicious PDFs and email-based spam — to increase malware download and infection rates. We also found that some well-known Pay-per-install providers use malicious PDFs as an infection vector. We listed some alternate PDF readers that are less frequently targeted by attackers, and described two recent security-focused features that were added to Adobe Reader — sandboxing and auto-updates. While current malware trends show a decline in the use of PDF exploits in favor of Java exploits, it remains to be seen if the security features added in Adobe Reader 10 will act as a sufficient deterrent to attackers, or if the existence of older vulnerabilities and a large base of unpatched systems will continue to make PDF-based malware a reliable vector for infecting and compromising systems.

Bibliography

[1] "Scansafe Annual Global Threat Report 2009." http://www.scansafe.com/downloads/gtr/2009_AGTR.pdf .

[2] "Cisco 2010 Annual Security Report." http://www.cisco.com/en/US/prod/collateral/vpndevc/security_annual_report_2010.pdf .

[3] "6 out of every 10 users run vulnerable versions of Adobe Reader." http://public.avast.com/mkt/20110713_6_out_of_10_with_vulnerable_PDF.pdf .

[4] K. Levchenko, N. Chachra, B. Enright, M. Felegyhazi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, D. McCoy, A. Pitsillidis, N. Weaver, V. Paxson, G. M. Voelker, and S. Savage, "Click Trajectories: End-to-End Analysis of the Spam Value Chain," in *Proceedings of 32nd annual Symposium on Security and Privacy*, IEEE, May 2011.

[5] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring Pay-per-Install: The Commoditization of Malware Distribution," in *Proceedings of USENIX Security*, August 2011.

[6] "Contagio Malware Dump." http://contagiodump.blogspot.com/2010/08/malicious-documents-archive-for.html .

[7] "Krebs On Security - A Peek Inside the Eleonore Browser Exploit Kit." http://krebsonsecurity.com/2010/01/a-peek-inside-the-eleonore-browser-exploit-kit/ .

[8] "Avast Security Blog." https://blog.avast.com/2011/04/22/another-nasty-trick-in-malicious-pdf/ .

[9] "Symantec Security Blog." http://www.symantec.com/connect/blogs/fight-against-malicious-pdfs-using-ascii85decode-filter .

[10] "Wepawet." http://wepawet.iseclab.org/index.php .

[11] M. Cova, C. Kruegel, and G. Vigna, "Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code," in *Proceedings of the World Wide Web Conference (WWW)*, 2010.

[12] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost In The Browser: Analysis of Web-based Malware," in *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[13] B. Stone-Gross, M. Cova, C. Kruegel, , and G. Vigna, "Peering Through the iFrame," in *Proceedings of IEEE Infocom*, 2011.

[14] M. Egele, E. Kirda, and C. Kruegel, "Mitigating Drive-by Download Attacks: Challenges and Open Problems," in *Proceedings of the Open Research Problems in Network Security Workshop (iNetSec 2009)*, 2009. http://iseclab.org/papers/inetsec09.pdf .

[15] M. Polychronakis, P. Mavrommatis, and N. Provos, "Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware," in *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[16] "CVE - Common Vulnerabilities and Exposures." http://cve.mitre.org/ .

[17] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *Trans. Info. & Sys. Sec.*, 2011. To appear.

[18] "File Exchange." http://filex.jeek.org/ .

[19] "Virustotal - Free online virus, malware and URL scanner." http://www.virustotal.com/ .

[20] "An Overview of Exploit Packs." http://blogs.mcafee.com/mcafee-labs/an-overview-of-exploit-packs .

[21] "Anubis: Analyzing Unknown Binaries." http://anubis.iseclab.org/ .

[22] "Malc0de Database." http://malc0de.com/database/ .

[23] "Clean MX Realtime Database." http://support.clean-mx.de/clean-mx/viruses.php .

[24] "Malware URL." http://www.malwareurl.com/listing-urls.php .

[25] "Malware Database by abuse.ch." http://amada.abuse.ch/ .

[26] "ThreatExpert Malware Reports." http://www.threatexpert.com/reports.aspx/ .

[27] "World's Top Malware." http://blog.fireeye.com/research/2010/07/worlds_top_modern_malware.html/ .

[28] "Inside Adobe Reader Protected Mode." http://blogs.adobe.com/asset/2010/11/inside-adobe-reader-protected-mode-part-3\ \-broker-process-policies-and-inter-process-communication.html .

[29] "Adobe Ships Security Patches, Auto-Update Feature." http://krebsonsecurity.com/2011/06/adobe-ships-security-patches-auto-update-feature/ .

[30] "Update on the New Updater." http://blogs.adobe.com/adobereader/2010/06/adobe_reader_and_acrobat_933_a.html/ .

[31] "Foxit Reader PDF Handling Multiple Remote Vulnerabilities." http://www.securityfocus.com/bid/34035/info .