Int. J. Inf. Secur.
DOI 10.1007/s10207-014-0248-7

REGULAR CONTRIBUTION

The MALICIA dataset: identification and analysis of drive-by download operations

Antonio Nappa · M. Zubair Rafique · Juan Caballero

Abstract Drive-by downloads are the preferred distribution vector for many malware families. In the drive-by ecosystem, many exploit servers run the same exploit kit and it is a challenge understanding whether the exploit server is part of a larger operation. In this paper, we propose a technique to identify exploit servers managed by the same organization. We collect over time how exploit servers are configured, which exploits they use, and what malware they distribute, grouping servers with similar configurations into operations. Our operational analysis reveals that although individual exploit servers have a median lifetime of 16 h, long-lived operations exist that operate for several months. To sustain long-lived operations, miscreants are turning to the cloud, with 60 % of the exploit servers hosted by specialized cloud hosting services. We also observe operations that distribute multiple malware families and that pay-per-install affiliate programs are managing exploit servers for their affiliates to convert traffic into installations. Furthermore, we analyze the exploit polymorphism problem, measuring the repacking rate for different exploit types. To understand how difficult is to takedown exploit servers, we analyze the abuse reporting process and issue abuse reports for 19 long-lived servers. We describe the interaction with ISPs and hosting providers and monitor the result of the report. We find that 61 % of the reports are not even acknowledged. On average, an exploit server still lives for 4.3 days after a report. Finally, we detail the MALICIA dataset we have collected and are making available to other researchers.

Keywords Drive-by download operations · Malicia dataset · Malware distribution · Cybercrime

A. Nappa (✉) · J. Caballero
IMDEA Software Institute, Madrid, Spain
e-mail: antonio.nappa@imdea.org

J. Caballero
e-mail: juan.caballero@imdea.org

A. Nappa
Universidad Politécnica de Madrid, Madrid, Spain

M. Z. Rafique
iMinds-DistriNet, KU Leuven, Leuven, Belgium
e-mail: zubair.rafique@cs.kuleuven.be

1 Introduction

Drive-by downloads have become the preferred distribution vector for many malware families [19,38]. A major contributing factor has been the proliferation of specialized underground services such as exploit kits and exploitation-as-a-service that make it easy for miscreants to build their own drive-by distribution infrastructure [19]. In this ecosystem, many organizations license the same exploit kit, essentially running the same software in their exploit servers (upgrades are free for the duration of the license and promptly applied). This makes it challenging to identify which drive-by operation a exploit server belongs to. This is fundamental for understanding how many servers an operation uses, which operations are more prevalent, how long operations last, and for prioritizing takedown efforts and law enforcement investigations.

A drive-by operation is a group of exploit servers managed by the same organization and used to distribute malware families the organization monetizes. An operation may distribute multiple malware families, e.g., for different monetization schemes. A malware family may also be distributed by different operations. For example, malware kits such as zbot or spyeye are distributed by many organizations building their own botnets. And, pay-per-install (PPI) affiliate programs

Published online: 21 June 2014

give each affiliate organization a customized version of the same malware to distribute [7].

In this paper, we propose a technique to identify exploit servers managed by the same organization, even when those exploit servers may be running the same software (i.e., exploit kit). Our technique enables reducing the large number of individual exploit servers discovered daily, to a smaller, more manageable, number of operations. Our intuition is that servers managed by the same organization are likely to share parts of their configuration. Thus, when we find two servers sharing configuration (e.g., pointed by the same domain, using similar URLs, or distributing the same malware), this is a strong indication of both being managed by the same organization. To collect the configuration information, we track exploit servers over time and classify the malware they distribute.

Our clustering can be used by law enforcement during the pre-warrant (plain view) phase of a criminal investigation [53]. During this phase, criminal activity is monitored and targets of importance are selected among suspects. The goal of the plain view phase is gathering enough evidence to obtain a magistrate-issued warrant for the ISPs and hosting providers for the servers in the operation. Our clustering can identify large operations that use multiple servers, rank operations by importance, and help understanding whether they belong to individual owners or to distribution services.

Results. Our data collection has been running for one year and has tracked 502 exploit servers. Our analysis reveals two types of drive-by operations. Two-thirds of the operations use a single server and are short lived. The other third of the operations uses multiple servers to increase their lifetime. These multi-server operations have a median lifetime of 5.5 days and some live for several months, despite individual exploit servers living a median of 16 h. Miscreants are able to run long-lived operations by relying on pools of exploit servers, replacing dead servers with clones. We also observe a few short-lived multi-server operations (lasting less than a day) that use over a dozen exploit servers in parallel to achieve a burst of installations. While most short-lived operations distribute a single malware family, we observe multi-server operations often distributing more than one. Furthermore, we identify two PPI affiliate programs (the *winwebsec* fake antivirus and the *zeroaccess* botnet) that manage exploit servers so that their affiliates can convert their traffic into installations, without investing in their own drive-by infrastructure.

We also analyze the hosting infrastructure. We find that to sustain long-lived multi-server operations, in the presence of increasing pressure from defenders, miscreants are turning to the cloud. Over 60 % of the exploit servers belong to cloud hosting services. Long-lived operations are using pools of exploit servers, distributed among different countries and autonomous systems (ASes) for resiliency, replacing dead servers with clones. Miscreants are taking advantage of a booming cloud hosting services market where hosting is cheap, i.e., virtual private servers (VPS) start at $10 per month and dedicated servers at $60 [26]. These services are easy to contract (e.g., automated sign-up procedures requiring only a valid credit card), and short leases are available (e.g., daily billing) so that the investment loss if the exploit server is taken down can be less than a dollar. In this environment, cloud hosting providers have started reporting that 50 % of their automated VPS subscriptions are being abused [29].

In addition, we analyze the exploits used by the monitored exploit servers. In particular, we measure the exploit polymorphism, which as far as we know has not previously been studied. Our results show that different exploit types (e.g., Java, PDF, Windows fonts) exhibit different repacking rates. For example, some exploit kits like BlackHole 2 have integrated automated repacking of PDF exploits for each exploitation attempt. On the opposite side, font exploits are not being repacked likely due to the lack of repacking tools. Exploits for Java vulnerabilities are repacked at a lower and varying frequency indicating that the repacking tool is likely invoked manually and not fully integrated in the exploit kit.

To understand how difficult is to takedown exploit servers, we issue abuse reports for 19 long-lived servers. We analyze the abuse reporting process, as well as the interaction with the ISPs and hosting providers. We use our infrastructure to monitor the result of the report (i.e., whether the server is taken down). The results are disheartening. Over 61 % of the reports do not produce a reply and the average life of a exploit server after it is reported is 4.3 days.

Our work reveals a growing problem for the takedown of drive-by download operations. While miscreants enjoy a booming hosting market that enables them to setup new exploit servers quickly, defenders face a tough time reporting abuse due to uncooperative providers and inadequate business procedures. Takedown procedures need to be rethought. There is a need to raise the cost for miscreants of a server being taken down, monitor short-lived VPS subscriptions, and shift the focus to prosecuting the organizations that run the operations, as well as the organizations behind specialized underground services supporting the ecosystem.

Finally, this work has produced a dataset that includes the malware binaries we collected, the metadata of when and how it was collected, and the malware classification results. To foster further research, we make our MALICIA dataset available to other researchers [27].

Contributions:

- We propose a technique to identify drive-by operations by grouping exploit servers based on their configuration and the malware they distribute.

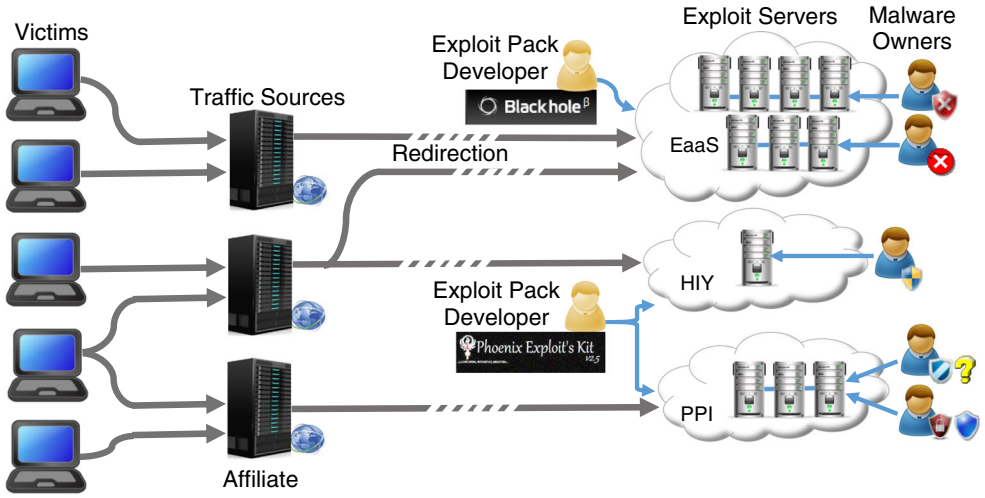

Fig. 1 Exploit kit ecosystem

– We report on aspects of drive-by operations such as the number of servers they use, their hosting infrastructure, their lifetime, and the malware families they distribute.
– We measure the polymorphism of the exploits served by the monitored exploit servers.
– We analyze the abuse reporting procedure by sending reports on exploit servers.
– We build a dataset with the collected malware, their classification, and associated metadata. We make this dataset available to other researchers.

2 Overview

Drive-by downloads are a popular malware distribution vector. To distribute its products over drive-by downloads, a malware owner needs three items: exploitation software, servers, and traffic. To facilitate the process, three specialized services exist (Fig. 1). A malware owner can license an exploit kit (host-it-yourself), rent a exploit server with the exploit kit installed (exploitation-as-a-service), or simply buy installs from a pay-per-install service that provides the exploit server and the traffic.

2.1 Roles

The exploit kit ecosystem has four main roles: *malware owner*, *exploit kit developer*, *exploit server owner*, and *exploit server manager*. Exploit kit developers offer a software kit including a set of exploits for different platforms (i.e., combination of browser, browser plugins, and OS), web pages to exploit visitors and drop files on their hosts, a database to store all information, and an administration panel to configure the functionality and provide installation statistics. Exploit kits are offered through two licensing models: host-it-yourself (HIY) and exploitation-as-a-service (EaaS). In both models, access to the exploit kit (or server) is time limited and clients obtain free software updates during this time. Also in both models, the client provides the traffic as well as a domain name to which the kit is linked. The client pays for domain changes (e.g., $20 for BlackHole [58]) unless it buys a more expensive multi-domain license.

The exploit server provider is the entity that contracts the hosting and Internet connectivity for the exploit server. It can be the malware owner in the HIY model or the exploit kit developer in EaaS. Exploit kits are designed to be installed on a single host that contains the exploits, malware files, configuration, and statistics. Thus, exploit servers are typically dedicated, rather than compromised, hosts. A robust hosting infrastructure is needed to launch long-lived operations as most exploit servers are short lived. Exploit server providers acquire a pool of servers and favor hosting providers and ISPs where exploit servers live longer, i.e., those that are not diligent in handling abuse reports, or those who offer a specific abuse protection policy.

The exploit server manager is the entity that manages the exploit server through its administration panel. The manager is a client of the exploit kit developer and corresponds to the malware owner or a PPI service. PPI affiliate programs may run their own exploit server providing each affiliate with a unique affiliate URL. Affiliates credit installs by installing their affiliate-specific malware executable in hosts they have compromised, or by sending traffic to their affiliate URL, which would in turn install their affiliate-specific malware if exploitation succeeds. In these programs, affiliates can point their traffic sources to their affiliate URL in the program's exploit server or to their own exploit server. The latter requires investment, but has two advantages: They can configure their exploit server to install other malware on the compromised machine, and they can avoid the affiliate pro-

gram skimming part of their traffic for their own purposes. Our operation analysis reveals both exploit servers managed by individual affiliates and by PPI affiliate programs.

2.2 Exploit server clustering

In this work, we cluster exploit servers under the same management using information about the server's configuration. Two servers sharing configuration, (e.g., pointed by the same domain, hosting similar URLs, using the same exploits, or distributing the same malware) indicates that they may be managed by the same organization. We focus on server configuration because the software is identical in many exploit servers since kit updates are free and promptly applied (19 days after the launch of BlackHole 2.0, we could no longer find any live BlackHole 1.x servers).

Our results show that attackers are using pools of exploit servers to sustain malware distribution operations over time. Such operations often run multiple exploit servers simultaneously behind a traffic direction system (TDS) that distributes the incoming traffic among them [25]. When one of their exploit servers goes down, attackers replace it with another server from their pool. The intuition behind our clustering is that in this environment, new exploit servers often reuse the configuration of older servers. This happens because, when installing a new exploit server, the attacker simply clones an existing server including its configuration, e.g., by uploading to a new cloud hosting provider a previously created virtual machine image file where an exploit server is installed and configured.

3 Methodology

To collect the information needed to cluster servers into operations, we have built an infrastructure to track individual exploit servers over time, periodically collecting and classifying the malware they distribute. Our pipeline is described in Fig. 2. We receive feeds of drive-by download URLs (Sect. 3.1), use honeyclients as well as specialized milkers to periodically collect the malware from the exploit servers those URLs direct to (Sect. 3.2), classify malware using icon information and behavioral reports obtained through execution in a contained environment (Sect. 3.3), store all information in a database, and use the collection and classification data for clustering exploit servers into operations (Sect. 4) and for abuse reporting (Sect. 5). An earlier version of the milking and classification components was used to collect the BlackHole/Phoenix feed in [19]. Since that work, we have upgraded those two components. This section describes their latest architecture, detailing the differences with [19].

3.1 Feeds

To identify exploit servers for the first time, we use two publicly available feeds: Malware Domain List (MDL) [28] and urlQuery [51]. MDL provides a public forum where contributors report and discuss malicious URLs. The reported URLs are manually checked by volunteers. Once verified, they are published through their webpage and feeds. urlQuery is an automatic service that receives URLs submitted by analysts and publishes the results of visiting those URLs on their webpage. We periodically scan the webpages of MDL and urlQuery for URLs matching our own regular expressions for the landing URLs of common exploit kits. The volume of URLs in urlQuery is much larger than in MDL, but the probability of finding a live exploit server is larger in MDL because URLs in urlQuery are not verified to be malicious and URLs long dead are often re-reported. We selected these feeds based on their public availability, but the infrastructure is designed to work with any URL feed.

3.2 Milking

Our milking component differs from the one used to collect the BlackHole/Phoenix feed in [19] in that it identifies an exploit server by its *landing IP*, i.e., the IP address hosting the landing URL, which provides the functionality (typically some obfuscated JavaScript code) to select the appropriate exploits for the victim's platform. In [19], we identified exploit servers by the domain in their URLs. This was problematic because a large number of domains often resolve to the IP address of an exploit server. When the domains in the URLs known to us went down, our milking would consider the exploit server dead, even if it could still be reachable through other domains. Currently, if all domains in the land-

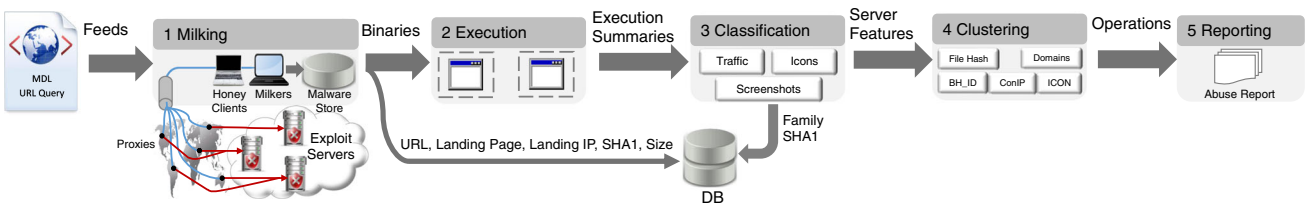

Fig. 2 Architecture of our milking, classification, and analysis

ing URLs of a server stop resolving, the milking queries two passive DNS services [6,44] for alternative domains recently observed resolving to the exploit server. If no alternative domain is found, the milking continues using the landing IP.

In addition, our infrastructure now resolves the malicious domains periodically, which enables locating previously unknown exploit servers if the same domain is used to direct traffic to different exploit servers over time. This information is used in our clustering (Sect. 4). Using this separate resolution, we discover an additional 69 servers not present in our feeds and another 30 servers before they appear in the feeds.

Another difference is that in [19], we relied exclusively on lightweight *specialized milkers*, i.e., custom HTTP clients that collect the malware from the exploit server, without running a browser or going through the exploitation process, simply by replaying a minimized network dialog of a successful exploitation. Our specialized milkers take advantage of the lack of replay protection in the BlackHole 1.x and Phoenix exploit kits. For details on the construction of the specialized milkers, we refer the interested reader to our technical report [40]. Since then we have added support for milking other exploit kits by adding *honeyclients*, i.e., Windows virtual machines installed with an unpatched browser (and browser plugins), which can be navigated to a given landing URL [30,54].

Milking policy. Our milking tries to download malware from each known exploit server every hour on average. If no malware is collected, it increments a failure counter for the exploit server. If a failure counter reaches a threshold of six, the state of its exploit server is changed to offline. If malware is collected before 6 h, its failure counter is reset. This allows milking to continue through temporary failures of the exploit server. In addition, the milking component runs a separate process that checks if an offline exploit server has resurrected every two days. If three consecutive resurrection checks fail, the exploit server is considered dead. If the server has resurrected, its failure and resurrection counters are reset.

3.3 Malware classification

Our classification process leverages icon information extracted statically from the binary as well as network traffic and screenshots obtained by executing the malware in a contained environment. Compared with the classification process in [19], we propose the automated clustering of malware icons and screenshots using perceptual hashing and have implemented a novel malware clustering and signature generation tool [39]. In addition, we evaluate the accuracy of the icon and screenshot clustering using a manually generated ground truth.

Fig. 3 Icon polymorphism. Each pair of icons comes from two different files of the same family and is perceptually the same, although each icon has a different hash

Additional behavioral features could be used for malware classification such as the system and API calls invoked by the malware during execution [4,5]. We have not yet added those due to resource constraints. Their addition would likely increase our malware classification results. However, our current classification only leaves 2 % of the malware samples unclassified, so the benefit would be limited.

Malware execution. We execute each binary in a virtualized environment designed to capture the network traffic the malware produces and to take a screenshot of the guest VM at the end of the execution. We use Windows XP Service Pack 3 as the guest OS and only allow DNS traffic and HTTP connections to predefined benign sites to leave our contained environment. All other traffic is redirected to internal sinks.

Our classification applies automatic clustering techniques separately to the icons, the screenshots, and the network traffic. Then, an analyst manually refines the generic labels by comparing cluster behaviors against public reports. Finally, majority voting on the icon, screenshot, and network labels decide the family label for an executable.

Icons. A Windows executable can embed an icon in its header. Many malware families use icons because it makes them look benign and helps them establish a brand, which is important for some malware classes such as rogue software. Icons can be extracted statically from the binary without running the executable, so feature extraction is very efficient. A naive icon feature would simply compute the hash of the icon. However, some malware families use polymorphism to obfuscate the icons in their executables, so that two malware of the same family has icons that look the same to the viewer, but have different hashes (Fig. 3). To capture such polymorphic icon variants, we use a perceptual hash function [60]. Perceptual hash functions are designed to produce similar hashes for images that are perceptually (i.e., visually) similar. A good perceptual hash returns similar hashes for two images if one is a version of the other that has suffered transformations such as scaling, aspect ratio changes, or small changes in brightness, contrast, and color. We have experimented with two different perceptual hash functions: average hash (avgHash) [23] and pHash [60]. We use the Hamming distance between hashes as our distance metric. If the distance is less than a threshold, both icons are clustered together using the aggressive algorithm in Sect. 4.2. We experimentally select the threshold value for each feature. Table 1 (top) shows the clustering

Table 1 Clustering results for icons (top) and screenshots (bottom)

	Feature	Th.	Clus.	Precision (%)	Recall (%)	Time
I	avgHash	3	141	99.7	91.3	2.4 s
I	pHash	13	149	99.8	89.6	1 m17 s
S	avgHash	1	64	99.2	60.4	7 m37 s
S	pHash	13	43	97.9	52.4	16 m8 s

results on 5,777 icons compared with the manually generated ground truth, which an analyst produces by examining the clusters. The results show very good precision for both features and slightly better recall and runtime for avgHash.

Screenshots. The screenshot clustering also uses perceptual hashing. Table 1 (bottom) shows the clustering results on 9,896 screenshots. This time avgHash achieves better precision, but slightly worse recall. The lower recall compared with the icons is due to the perceptual hashing distinguishing error windows that include different text or the icon of the executable. Still, the clustering reduces 9,896 screenshots to 50–60 clusters with very high precision, so it becomes easy for an analyst to manually label the clusters. We ignore clusters that capture generic error windows or do not provide family information, e.g., the Windows firewall prompting the user to allow some unspecified traffic.

Network traffic. Our network clustering has evolved over time. In our earlier works [19,32], the features used by the network clustering were the C&C domains contacted by the malware and tokens extracted from the URLs and User-Agent headers in HTTP requests sent by the malware. Those features did not handle malware using non-HTTP traffic for C&C communication. To support non-HTTP C&C traffic, we developed a novel malware clustering and signature generation tool called FIRMA [39].

FIRMA takes as input a set of network traces obtained by running unlabeled malware binaries in a contained environment. It outputs: (1) a *clusters file* with a partition of the malware binaries that produced the network traces into *family clusters*, (2) a *signature file* with network signatures annotated with the family cluster they correspond to, and (3) an *endpoints file* with the C&C domains and IP addresses that the malware binaries in each family cluster contacted across the input network traces.

The network signatures produced by FIRMA enable classifying new executables from previously seen malware families efficiently, without having to re-run the clustering. More importantly, they can be used to identify malware-infected computers in network monitored by an IDS. FIRMA produces network signatures in the format supported by the open source signature-matching IDSes Snort [47] and Suricata [49], so that those popular IDSes can be used to match the signatures on network traffic.

In addition to not being limited to any type of traffic (e.g., HTTP) or specific fields (e.g., HTTP URL, User-Agent), other salient characteristics of FIRMA are that it produces a set of network signatures for each malware family (i.e., family cluster) and that the signature generation is protocol-aware. A set of signatures is produced for each family cluster so that each signature captures a different network behavior of the family, e.g., one signature for HTTP traffic and another for a binary C&C protocol or different signatures for different protocol messages. Using a protocol-aware traffic clustering and signature generation means that if the C&C traffic uses a known application protocol such as HTTP, IRC, or SMTP, the traffic is parsed into fields and the signatures capture that a token may be specific to a field and should only be matched on that field, increasing signature accuracy.

It is important to note that FIRMA supports obfuscated C&C protocols, commonly used by malware. In fact, much of the malware in our datasets uses such obfuscation. It is still possible to generate signatures on those because obfuscated C&C protocols often contain value invariants. For fully polymorphic C&C protocols [42], i.e., protocols where every single byte changes in each request making it impossible to generate a signature on the ciphertext, FIRMA can still cluster malware samples from the same family if they reuse the same C&C domains or IP addresses. We refer the reader to the FIRMA paper for more details [39].

Once FIRMA outputs the family clusters, an analyst tries to map the automatically generated cluster labels (e.g., CLUSTER:A) to well-known traffic families (e.g., zbot). Overall, our classification produces traffic labels for 80 % executables, icon labels for 51 %, and screenshot labels for 21 %. It classifies 93 % of the executables, 5 % fail to execute, and 2 % remain unclassified.

3.4 Exploit analysis

An exploit is an input to a program that takes advantage of a software vulnerability, which affects some versions of the program, to cause an unintended behavior of the program. An exploit is able to drive the program execution to the vulnerability point where the vulnerability can be triggered (e.g., an arithmetic operation) and to trigger the vulnerability condition (e.g., make the arithmetic operation overflow).

Similar to malware, exploits can also exhibit polymorphism. That is, the same exploit can be repacked multiple times to generate different *exploit instances*, i.e., variants of the same exploit. For example, a PDF exploit, in addition to the data that lead the program to the vulnerability point and triggers the vulnerability condition, may also contain much other data that can be modified to generate a different PDF file that still exploits the same vulnerability in the same way. In this example, we say that the old and new PDF files are two instances of the same exploit.

Exploit polymorphism can be introduced by exploit kit developers to bypass exploit signatures used by AV vendors. It also happens when the payload of the exploit (e.g., the code run after exploitation and the data used by that code) needs to be updated, without affecting how the vulnerability is exploited. For example, an exploit may embed URLs from where some malware is downloaded after exploitation. Those URLs may need to be changed periodically to bypass URL blacklists. Every time they are changed, a new exploit instance is created.

In this section, we perform what we believe is the first analysis of how exploit polymorphism works in prevalent exploit kits. In particular, we examine how often different exploit types (e.g., Java, PDF, Windows fonts) are repacked to introduce polymorphism.

Exploit collection. To analyze the exploits used in the drive-by downloads, we leverage that (starting on November 19, 2012) every time a honeyclient visits a potentially malicious URL, a trace of the complete network communication is stored. This produces 14,505 network traces, of which 19.7 % correspond to exploitation attempts. The rest corresponds to landing URLs that no longer lead to an exploit server, or exploit servers temporarily down. Of those exploitation attempts 74.5 % (14.7 % of all network traces) lead to malware installation. The unsuccessful exploitation attempts may be due to unreliable exploits and, rarely, to the honeyclient execution finishing before malware was downloaded. The collected exploits come from the BlackHole 2.x (BH2) and CoolExploit exploit kits. For other exploit kits, for which, we use milkers, e.g., Phoenix and BlackHole 1.x, no network traces are generated.

The network traces contain multiple instances of the exploits that the exploit servers use against our honeyclients. Extracting and classifying all exploit instances in a network trace are challenging. We focus on exploits that are individual files, namely Java JAR archives, PDF documents, and EOT font files. There may be other types of exploits embedded in the landing page, e.g., JavaScript exploits. We do not attempt to identify and classify those because the landing page is obfuscated and while it could be deobfuscated, understanding what parts of it correspond to an exploit is difficult, as the landing page contains additional functionality like checking the victim's software configuration and redirecting the user after exploitation. We extract a total of 172 unique (by SHA1 hash) exploit instances: 114 JAR, 55 PDF, and 3 EOT files. The JAR and PDF files target vulnerabilities in the Java and Adobe Reader browser plugins, respectively, and the EOT files target Windows vulnerabilities.

Exploit kits select which exploits to use from their exploit pool based on the software configuration of the victim, typically checked by some JavaScript code in the landing page. Thus, the exploits in our network traces depend on the configuration of our honeyclients. Our honeyclients have the same configuration: Windows XP SP3 with IE 7, Java JRE 6u14, and Adobe Reader 8. It is important to note that our goal is not to analyze all the exploits that a exploit kit contains. Our approach cannot see exploits that exploit servers do not serve against our honeyclients. For example, the BlackHole and CoolExploit kits are known to contain Flash exploits [3]. However, we do not observe Flash exploits in our network traces, likely because our honeyclients do not have Flash installed; thus, the exploit kits do not serve those exploits. Rather, our goal is to analyze a novel question: Whether exploit kits periodically repack their exploits and if so, how often.

Exploit classification. To verify that the extracted files are exploits, we leverage two online services: VirusTotal [52] and Wepawet [55]. We use VirusTotal for JAR and EOT files and Wepawet for PDF files, as VirusTotal does not support PDF files. The three EOT files and 112/114 JAR files were present in VirusTotal and flagged by at least one AV vendor to be malicious. We submitted to VirusTotal the two JAR files that they had not seen yet, which were both flagged as malicious. We submit all 55 PDF documents to Wepawet, which identifies them as exploits. Thus, all extracted files are indeed exploits, which shows that the file extraction from the network traces does not identify benign files as exploits.

We classify the exploits according to the vulnerability they target. We first submit the exploit files to VirusTotal, which scans them using a variety of AV products and publishes the AV detection labels. We leverage the fact that some AV vendors include the CVE identifier of the vulnerability in their labels. For each AV that detects the file as malicious, we check if a CVE identifier is included in the detection label using the following regular expression: CVE[-_]?([0-9]{4})[-_]+([0-9]{4}). Different AV vendors may not agree on the vulnerability targeted by the exploit. To address this issue, we use majority voting to select the most commonly seen CVE identifier for a exploit file. In case of a tie, we choose the most frequent CVE identifier between the tied ones, where frequency is computed across all exploits that present no tie. For the PDF files, Wepawet outputs a single CVE for each exploit, so majority voting is not needed.

Using AV labels for classification is not ideal as it is well known that AV vendors focus on detection and their classification is often inaccurate [4]. Indeed, we found two exploit instances where AV vendors disagreed on the targeted vulnerability, and the majority voting was wrong. In this case, it was possible to spot the error because the vulnerabilities output by the majority voting did not affect the Java versions run by our honeyclients. These examples illustrate the need for developing accurate exploit classification techniques.

Table 2 presents the exploit classification results. For each exploit type, it shows the CVE identifier of the different vulnerabilities or *Unknown* if no CVE identifiers were found. For

Table 2 Exploit classification

Type	CVE	Instances	Collected	Ratio (%)
JAR	CVE-2012-0507	88	2,419	3.5
	CVE-2012-1723	15	203	7.4
	CVE-2010-4476	3	103	2.9
	Unknown	8	158	5.0
	Subtotal	114	2,894	3.9
EOT	CVE-2011-3402	3	777	0.3
PDF	CVE-2009-0927	43	47	91.4
	CVE-2010-0188	12	12	100
	Subtotal	55	69	93.2
Total		172	3,740	4.6

each vulnerability, it presents the number of unique exploit files labeled with that vulnerability, the total number of times those instances appears in the network traces, and the ratio of both numbers as a percentage. The larger the percentage the more likely it is that when we collect a exploit instance for a vulnerability, we have never seen that instance before. The results show that Java vulnerabilities are most targeted against our honeyclients, followed by the CVE-2011-3402 TrueType font parsing vulnerability in Windows, with PDF exploits comprising only 1.8 % of all exploits used against our honeyclients. The most common Java vulnerability is CVE-2012-0507, which is targeted by 65 % of the collected exploits. As explained earlier, if we configured our honeyclients differently, exploit kits may serve them other exploits.

Exploit polymorphism. We analyze whether exploit kits have integrated packing tools to automatically provide polymorphism for their exploits. For this, we analyze each exploit file type (JAR, PDF, EOT) separately. For PDF files, the ratio in Table 2 shows that 93.2 % of the times that we collect a PDF exploit instance it is new, i.e., we have not collected it before. This indicates high polymorphism as we periodically visit each exploit server until it dies, indicating that the same server distributes different exploit instances for the same vulnerability over time. A detailed analysis of PDF exploit distribution reveals 11 BlackHole 2.0 (BH2) servers distributing PDF exploits. Consecutive visits to the same BH2 exploit server, separated by as little as 4 min, yield different PDF exploits for the same vulnerability. In addition, when we collect multiple instances of the same PDF exploit, all the instances appear in a single network trace (i.e., on the same drive-by download session). These indicate that automatic repacking of PDF exploits is likely integrated into the BH2 exploit kit and occurs for every drive-by download session. This means that byte signatures are unlikely to help in detecting PDF exploits.

In contrast, the ratio for EOT exploits is very low (0.3 %), which indicates low polymorphism. Detailed analysis of the EOT files shows that of the three files, one is served by the CoolExploit kit and the other two by different operations that use BH2. Each exploit instance uses a different URL to download the malware after exploitation and is never repacked, i.e., BH2 and CoolExploit servers always distribute the same EOT exploit instance over time. For example, the longest lived server in our dataset distributes the same EOT exploit for over two months. This likely indicates that there are no repacking tools available for EOT exploits. Once an EOT exploit is created with an embedded URL, attackers need to ensure that the URL stays up over long periods of time, e.g., by using bullet-proof hosting. This produces an easy avenue for detection. We expect that in the near future repacking tools will become available due to market demand or that EOT exploits will be replaced by easier to repack exploit types.

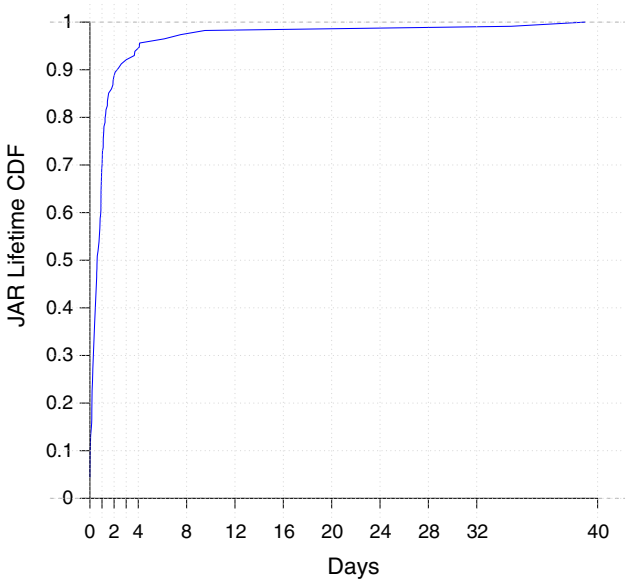

Fig. 4 CDF of the lifetime of Java exploits

The ratio for JAR exploit instances is 3.9 % indicating some polymorphism, but much lower than PDF exploits. Figure 4 shows the CDF of the lifetime of JAR exploit instances, where the lifetime of an instance is measured as the difference between the last and first time it was collected. The median lifetime is 14.4 h, the average is 39.3 h, and the standard deviation is 121.9 h (i.e., 5 days). In total, 69.2 % of the exploits have a lifetime less or equal to one day, 19.2 % have a lifetime between one and two days, and the remaining more than two days. The high standard deviation in lifetime and the overall low repacking rate indicate that Java exploits are not automatically repacked by the monitored exploit kits. They are likely repacked using tools that need to be manually invoked by the exploit server manager. These tools may be shipped (but not integrated) with the exploit kit or may be commercial-off-the-self (COTS) tools [1,61].

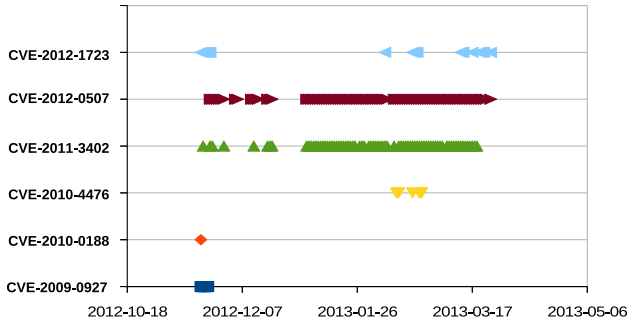

Fig. 5 Exploit distribution of different vulnerabilities over time

The fact that EOT exploits are not repacked means that different exploit servers in the same operation may deliver the same EOT exploit instance. In addition, since the repacking of Java exploits requires some manual steps, exploit server managers are likely to copy the newly packed exploit instances across their exploit servers. We leverage these to incorporate exploit polymorphism as one of the features used in our exploit server clustering, presented in the next section.

Vulnerabilities exploited over time. Figure 5 shows the distribution over time of the exploits for each vulnerability. Of the eight vulnerabilities, exploits for five of them were collected from the time we started producing network traces with the honeyclients. Exploits for the three other vulnerabilities started to be collected three months later, which indicates that those vulnerabilities were added later to the exploit kits. Two of these (CVE-2013-0431 and CVE-2013-0422) are recent vulnerabilities. This illustrates the benefit of the exploit kit ecosystem where exploit kit writers focus on developing exploits for new vulnerabilities, and new exploits for old ones, and customers achieve improved infection rates by adding those exploits.

4 Exploit server clustering

To identify exploit servers managed by the same organization, we propose an unsupervised clustering approach that groups exploit servers that have similar exploit kit configurations. The intuition is that exploit servers in the same operation are more likely to be similarly configured because attackers may reuse parts of the configuration across their servers. For example, attackers may clone an existing exploit server to create a new one in a different hosting provider by copying the image file with the exploit kit configured. Or, they may use a malicious domain to point to different exploit servers over time. In contrast, exploit servers in different operations are less likely to be similarly configured.

To cluster the exploit servers, we define a distance metric between two exploit servers based on their configuration. The distance metric combines seven features that capture how a exploit server is configured including the domains that point to the server, the server's IP address, the URLs it hosts, the exploits it serves, and the malware it distributes. These features are derived from our milk data.

This section details our seven similarity features (Sect. 4.1) and the clustering algorithms we use (Sect. 4.2).

4.1 Features

We define seven boolean server similarity features:

1. *Landing URL feature:* The landing URL of a exploit server contains elements that are specific to the configuration of the exploit kit. In particular, the file path in the landing URL (the directory where the kit's files are installed and the name of those files) and the parameter values (typically used to differentiate traffic sources) are configurable and changed from the default by the manager to make it difficult to produce URL signatures for the kit. This feature first extracts for each landing URL the concatenation of the file path (including the file name) and the list of parameter values. The similarity is one if the set intersection is non-empty, otherwise it is zero.
2. *Domain feature:*If the same DNS domain has resolved to the IP addresses of two exploit servers, that is a strong indication that both exploit servers belong to the same organization, i.e., the one that owns the domain. This feature first extracts the set of DNS domains that have resolved to the IP address of each server. The similarity between two servers is one if the set intersection is non-empty, otherwise the similarity is zero.
3. *File hash feature:* A malware owner can distribute its malware using its own infrastructure (HIY or EaaS) or a PPI service. However, it is unlikely that it will use both of them simultaneously because outsourcing distribution to a PPI service indicates a willingness to avoid investing in infrastructure. Thus, if the same malware executable (i.e., same SHA1 hash) is distributed by two servers, that is a strong indication of both exploit servers belonging to the same organization. This feature first extracts the set of file hashes milked from each exploit server. The similarity is one if the set intersection is non-empty, otherwise it is zero.
4. *Icon feature:* The icon in a malware executable is selected by the creator of the executable, i.e., malware owner or an affiliate PPI program (the program is typically in charge of repacking the affiliate-specific malware [7]). In both cases, a shared icon in files distributed by different servers is a strong indication of both servers distributing malware from the same owner. This feature is related to the file hash feature, but covers files that may have been repacked, while keeping the same icon. This feature first extracts the set of icons in files milked from each exploit

server. The similarity is one if the set intersection is larger than one, otherwise it is zero.

5. *Family feature:* If two servers distribute the same malware family, and the malware family is neither a malware kit (e.g., zbot, spyeye) nor an affiliate program, then the two servers distribute malware of the same owner and thus share management. This feature is optional for the analyst to use because it requires a priori knowledge of which malware families are malware kits or affiliate programs, otherwise it may overcluster. This boolean feature first extracts the set of non-kit, non-affiliate malware families distributed by each exploit server. The similarity is one if the set intersection is non-empty, otherwise it is zero.
6. *Consecutive IP feature:* If two exploit servers have consecutive IP addresses that is a strong indicator that both servers have been contracted by the same entity because the probability of two separate exploit server owners selecting the same ISPs and those ISPs selecting consecutive IP addresses for their servers is very small. The similarity between two servers is one if their IP addresses are consecutive, otherwise it is zero.
7. *Exploit hash feature:* Similar to malware, exploits may also be periodically repacked. Such repacking is either performed on the fly by the exploit kit or manually using a repacking tool. In both cases, if the same exploit (i.e., same SHA1 hash) is distributed by two servers, that is a strong indication of both exploit servers being managed by the same entity. This feature first extracts the set of exploit hashes collected from each exploit server. The similarity is one if the set intersection is non-empty, otherwise it is zero. Note that exploit types that are automatically repacked in each exploitation session, e.g., PDF exploits, will not capture similarity in this feature since each exploit server serves different PDF exploit instances. In our experiments, this feature captures similarity on the Java and EOT exploits only. However, with other exploit kits, it is unknown a priori whether automatic repacking for a exploit type has been integrated in the kit. Thus, it is important to include all exploit types when computing this feature.

4.2 Clustering algorithms

We experiment with two clustering algorithms: The partitioning around medoids (PAM) [22] and an aggressive clustering algorithm that groups any servers with some similarity.

Partitioning around medoids. The input to the PAM algorithm is a distance matrix. To compute this matrix, we combine the server similarity features into a boolean server distance metric as $d(s_1, s_2) = 1 - (\bigvee_{i=1}^{5} f_i(s_1, s_2))$, where f_i is the server similarity feature i. Note that the features compute similarity (one is similar), while the distance computes dissimilarity (zero is similar). Once a distance matrix has been computed, we apply the PAM algorithm. Since PAM takes as input the number k of clusters to output, the clustering is run with different k values, selecting the one which maximizes the Dunn index [16], a measure of clustering quality.

Aggressive clustering. Our aggressive clustering first computes a boolean server similarity metric: Two servers have similarity one if any of the server feature similarities is one (logical OR). Then, it iterates on the list of servers and checks if the current server is similar to any server already in a cluster. If the current server is only similar to servers in the same cluster, we add the server to that cluster. If it is similar to servers in multiple clusters, we merge those clusters and add the current server to the merged cluster. If it is not similar to any server already in the clusters, we create a new cluster for it. The complexity of this algorithm is $O(n^2)$, but since the number of servers is on the hundreds, the clustering terminates in a few seconds.

5 Reporting

Reporting abuse is an important part of fighting cybercrime, largely overlooked by the research community. In this section, we briefly describe the abuse reporting process and the challenges an abuse reporter faces. In Sect. 6.5, we detail our experiences reporting exploit servers and discuss the current situation.

Five entities may be involved in reporting an exploit server: The *abuser*, the *reporter*, the *hosed* that owns the premises where the exploit server is installed, the *abuser's ISP* that provides Internet access to the exploit server, and *national agencies* such as CERTs and law enforcement. Sometimes, the ISP is also the hoster because it provides both hosting and Internet access to the exploit server. The abuser can also be the hoster if it runs the exploit server from its own premises.

The most common practice for reporting exploit servers (and many other abuses[1]), is to first email an abuse report to the ISP's abuse handling team, who will forward it to their customer (i.e., the hoster) if they do not provide the hosting themselves. If this step fails (e.g., no abuse contact found, email bounces, no action taken), the reporter may contact the CERT for the country where the exploit server is hosted or local law enforcement. There are two main reasons to notify first the abuser's ISP. First, in most cases, a reporter does not know the abuser's or hoster's identity. But, the abuser's

[1] This practice also applies to other types of abuse such as C&C servers, hosts launching SSH and DoS attacks, and malware-infected machines. However, spam is commonly reported from a receiving mail provider to the sender mail provider and web server compromises are commonly first reported to the webmaster.

ISP is the entity that has been delegated the IP address of the exploit server, which can be found in the WHOIS databases [15]. Second, ISPs that are provided evidence of an abuse of their terms of service (ToS) or acceptable use policy (AUP) by a host unlikely to have been compromised (e.g., an exploit server) can takedown the abusing server without opening themselves to litigation. This removes the need for law enforcement involvement, speeding the process of stopping the abuse.

Next, we describe three challenges a reporter faces when sending abuse reports.

Abuse report format and content. The Messaging Abuse Reporting Format (MARF) [17,18,45] defines the format and content for spam abuse reports. Unfortunately, it does not cover other types of abuse and proposals for extending it (e.g., X-ARF [57]) are still work in progress. In this work, we use our own email template for reporting exploit servers. The key question is what information will convince an ISP of the abuse. The goal is to provide sufficient evidence to convince the ISP to start its own verification. The key evidence we include is a network trace of a honeyclient being exploited by the exploit server. We also include the IP address of the server, the first day we milked it, and pointers to public feeds listing the server.

Abuse contact address. Finding the correct abuse contact is not always easy (or possible). For spam, RFC 6650 states that abuse reports should only be sent to email addresses clearly intended to handle abuse reports such as those in WHOIS records or on a web site of the form abuse@domain [18]. Unfortunately, not all ISPs have an abuse@domain address. Such addresses are only required for ISPs that (care to) have an abuse team [12] and have not been mandatory in WHOIS databases until recently. Even now, they are often only mandatory for new or updated WHOIS entries and the objects and attributes holding this information are not consistent across databases. We are able to find abuse addresses for 86 % of all exploit servers we milk. In practice, reporters use WHOIS to identify the organization that has been delegated the abuser's IP address. If an abuse email does not exist for the organization (or cannot be found in its website), abuse reports are sent to the organization's technical contact, which is mandatory in WHOIS. Unfortunately, after finding an email address to send the report, there is no guarantee on its accuracy.

Sender's identity. Abuse reports may end up being received by malicious organizations (e.g., bullet-proof ISPs or hosters). Thus, using an individual's real identity in an abuse report can be problematic. On the other hand, abuse teams may be suspicious of pseudonyms. Organizations that issue many abuse reports such as SpamHaus [50] can rely on their reputation, but they do not act as abuse aggregators. In this work, we use a pseudonym to hide our identities and still get access to the communication with ISPs and hosters.

Table 3 Summary of milking operation

Milking period	2012-03-07–2013-03-25
Malware executables milked	46,514
Unique executables milked	11,363
Domains milked	603
Servers milked	502
ASes hosting servers	242
Countries hosting servers	57
Malware executions	21,765
Total Uptime days	383

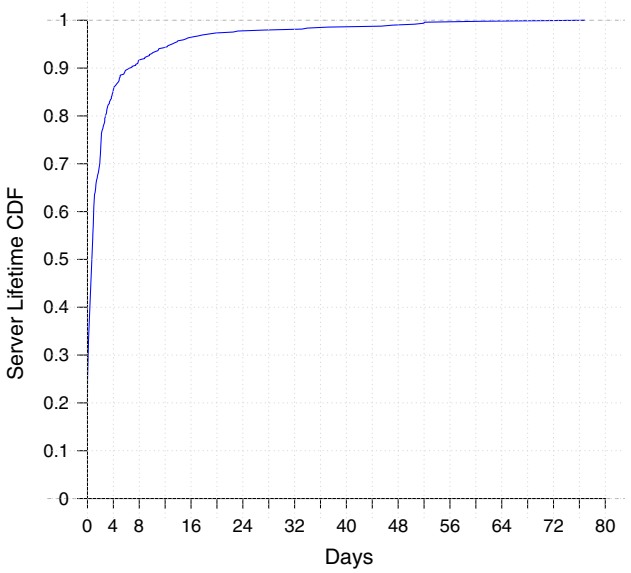

Fig. 6 CDF of exploit server lifetime

6 Analysis

Table 3 summarizes our milking, which started on March 7, 2012 and has been operating for 12 months (the BlackHole/Phoenix dataset in [19] covered only until April 20). We have milked a total of 502 exploit servers, hosted in 57 countries and 242 ASes, and downloaded from them 46,514 malware executables, of which 11,363 are unique (by SHA1 hash). A total of 603 DNS domains were observed pointing to the 502 servers.

6.1 Exploit server lifetime

To understand how well defenders are reacting to the drive-by download threat, we measure the exploit server lifetime, i.e., the period of time during which it distributes malware. For this measurement, we use only exploit servers found after we updated our infrastructure to identify servers by landing IP (Sect. 3.2) and remove servers for which we have sent abuse reports (Sect. 6.5). Figure 6 presents the CDF for the exploit

server lifetime. The majority of exploit servers are short lived: 13 % live only for an hour, 60 % are dead before one day, and the median lifetime is 16 h. However, it is worrying to observe a significant number of long-lived servers: 10 % live more than a week, 5 % more than two weeks, and some servers live up to 2.5 months.

The median exploit server lifetime we measure is more than six times larger than the 2.5 h median lifetime of a exploit domain (a domain resolving to the landing IP of an exploit server) measured by Grier et al. using passive DNS data [19]. This shows the importance of identifying exploit servers by their IP address, accounting for multiple domains pointing to the same server over time.

6.2 Hosting

In this section, we analyze the hosting infrastructure. We find that miscreants are abusing cloud hosting services. We also find, similar to prior work [46,48], autonomous systems hosting an inordinate number of exploit servers, compared with the size of their IP space.

Cloud hosting services. Using WHOIS, we can first determine which organization has been delegated the IP address of an exploit server and then use web searches to determine whether it offers cloud hosting services. Our results show that at least 60 % of the exploit servers belong to cloud hosting services, predominantly to Virtual Private Server (VPS) providers that rent VMs where the renter gets root access. This number could be larger because ISPs do not always reveal in WHOIS whether an IP address has been delegated to a customer, who may be a hosting provider. This indicates that drive-by operations have already embraced the benefits of outsourcing infrastructure to the cloud.

AS distribution. Table 4 shows the top ASes by the cumulative uptime (in days) of all exploit servers we milked in the AS. It also shows the number of exploit servers in the AS, the CAIDA ranking of the AS by the number of IPv4 addresses in its customer cone (the lower the ranking the larger the AS) [8], and the FIRE ranking for malicious ASes [48]. The two ASes with the largest number of exploit servers are in Europe, and the average life of an exploit server in those ASes is 10 days and four days, respectively, well above the median lifetime of 16 h. Some small ASes host an inordinate number of exploit servers compared with their ranking such as `awas` and `infiniumhost`, both located in Russia. There are also three ASes in eastern Europe that do not advertise any IP addresses or no longer exist, which could indicate that they were setup for such operations. We milked servers in three ASes that appear in the 2009 FIRE ranking. Two of them (`ovh` and `leaseweb`) appear also among our top ASes, which indicates that practices at those ASes have not improved in three years.

Table 4 Top ASes by cumulative exploitation time

ASN	Name	CC	Days up	ES	AS rank size	FIRE
16276	ovh	FR	194.84	21	5,623	10
701	uunet	US	139.60	1	8	–
44038	swisscom	CH	76.8	1	8,496	–
47869	netrouting	NL	70.0	18	4,395	–
43637	sol	AZ	61.1	1	6,828	–
48716	ps	KZ	52.0	1	8,530	–
56964	rmagazin	RO	49.5	2	8,337	–
12695	di-net	RU	47.6	9	175	–
36992	etisalat	EG	47.1	1	1,136	–
197145	infiumhost	RU	44.8	8	1,384	–
36444	nexcess.net.l.l.c	US	37.4	4	5,798	–
56413	proservis	LT	37.1	1	8,553	–
16265	leaseweb	NL	36.8	8	3,089	7
58182	kadroviy	RU	30.5	3	–	–
5577	root	LU	28.7	7	1,171	–
40676	psychz	US	28.1	5	6,939	–
21788	burst	US	27.8	14	3,942	–
28762	awax	RU	27.0	15	4,644	–
44784	sitek	UA	23.2	1	–	–
15971	ecosoft	RO	19.1	5	–	–

Table 5 Top malware families by number of exploit servers observed distributing the family

Family	Kit	ES	Files	Milk	Repack rate
zbot	Kit	170	2,168	11,619	18.6
cridex		64	74	2,555	2.8
zeroaccess	Aff	21	1,306	3,755	18.0
winwebsec	Aff	18	5,820	16,335	59.5
spyeye	Kit	11	7	342	2.0
CLUSTER:A		9	14	266	2.2
securityshield		5	150	307	11.8
CLUSTER:B		4	45	51	30.4
CLUSTER:C		4	1	4	1.0
smarthdd		4	68	453	3.1
CLUSTER:D		3	3	32	3.0
CLUSTER:E		3	1	4	1.0
CLUSTER:F		3	9	531	0.7
webprotect		3	3	26	3.9
cleaman		2	32	103	7.7
CLUSTER:G		2	5	148	1.5
CLUSTER:H		2	24	43	21.7
CLUSTER:I		2	9	17	9.4

6.3 Malware families

Our classification has identified a total of 55 families. Table 5 shows the top 18 families sorted by the number of exploit

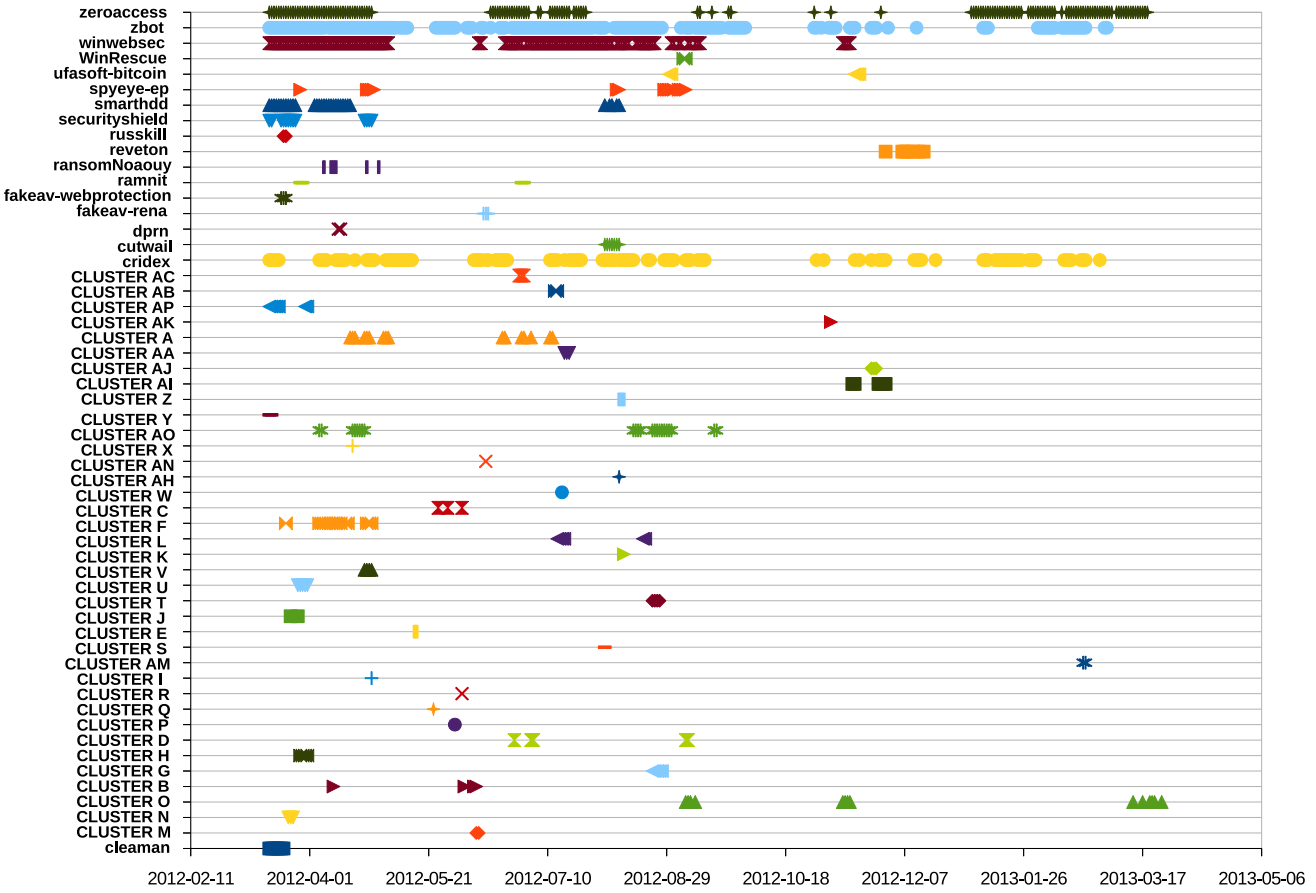

Fig. 7 Malware family distribution

servers observed distributing the family. For each family, Table 5 shows whether the family is a known malware kit or affiliate program, the number of servers distributing the family, the number of unique files milked, the total number of binaries milked from that family, and its repacking rate. Overall, the most widely distributed families are information stealers (zbot, cridex, spyeye), PPI downloaders (zeroaccess), and rogue software (winwebsec, securityshield, webprotect, smarthdd). The family most milked was winwebsec, a fake antivirus affiliate program, while the one distributed through most servers was zbot, a malware kit for stealing credentials.

Figure 7 shows the distribution of malware families over time. While most families are distributed through short operations, there are a few families such as zeroaccess, cridex, and zbot, which have been distributed throughout most of our study.

Families with shared ownership. Since different malware families target different monetization mechanisms, malware owners may operate different families to maximize income from compromised hosts. There are 50 servers distributing multiple malware families. Nine servers distribute different malware families through the same landing URL, during the same period of time, and to the same countries, e.g., a visit from the US with no referer would drop family one, another visit from the US a few minutes later family two, and then again family one. This indicates those families share ownership, as there is no way to separate the installs from the different families. Some families that manifest shared ownership are as follows: CLUSTER:D and cleaman, and securityshield and smarthdd. There is also shared ownership involving families known to be malware kits or affiliate programs such as winwebsec affiliates installing zbot and CLUSTER:L, and zbot botmasters installing ramnit.

Repacking rate. Malware owners repack their programs periodically to avoid detection by signature-based AV. On average, a malware family (excluding kits and affiliate programs) is repacked 5.4 times a day in our dataset. This is a sharp rise compared with the 0.1 times a day prior work reported during August 2010 [7]. This trend will further harm the detection rate of signature-based AVs. The rightmost column in Table 5 shows the repacking rate for our top families. The rate for families known to be kits or affiliate programs is artificially high, covering multiple botnets or affiliates. There are other families with high repacking rates such as securityshield, CLUSTER:B, and CLUSTER:H. This could indicate that those families are malware kits or affiliate programs.

Table 6 Operation clustering results

Alg.	Features 1–4			Features 1–5			Features 1–4,6,7			Features 1–7		
	Clusters	Largest	Singletons	Clusters	Largest	Singletons	Clusters	Largest	Singletons	Clusters	Largest	Singletons
Agg.	174	64	121	109	142	71	156	86	111	101	195	73
PAM	256	31	188	204	31	141	294	31	229	261	34	199

6.4 Operations analysis

In this section, we evaluate our clustering approach to identify operations that use multiple exploit servers. Unfortunately, we do not have ground truth available to evaluate our clustering results in a quantitative fashion. In fact, if such ground truth was available, then there would be no need for the clustering. Instead, we argue qualitatively that our clustering identifies meaningful and interesting drive-by operations.

Table 6 summarizes the clustering results. It compares the clustering results with both algorithms and using four different feature sets. The two leftmost feature sets (Features 1–4 and 1–5) correspond to the ones used in our prior work [32], while the two rightmost feature sets correspond to adding the new features on consecutive IP addresses and exploit polymorphism. For the new feature sets, we include the clustering results with and without the family feature for comparison. However, for the operation analysis below, we focus on the results without the family feature (Features 1–4, 6, 7), since we suspect some families like securityshield to be affiliate programs. Since those are distributed alongside other malware, the family feature can overcluster. For each clustering algorithm and feature set, the table shows the number of clusters, the size of the largest cluster, and the number of singleton clusters with only one server. As expected, the aggressive algorithm groups the most, minimizing the number of clusters.

We first present a number of operations our clustering reveals (for the aggressive clustering with six features unless otherwise noted), evaluating their correctness with information not used by our features such as which kit was installed in the exploit server and for affiliate programs, which affiliate a malware executable belongs to (we extract the affiliate identifier from the network traffic). Finally, we summarize the types of operations the clustering reveals and their distribution properties including the number of servers used, their hosting, and the operation lifetime.

Phoenix operation. Using both PAM and aggressive, all 21 Phoenix servers are grouped in the same cluster, which exclusively distributes zbot. Here, the clustering reveals that the Phoenix servers belong to the same operation *without* using any features about the exploit kit. Both algorithms do not include servers from other kits in the cluster, so they are not overclustering.

Reveton operation. We observe two clusters exclusively distributing the Reveton ransomware, which locks the computer with fake police advertisements. One cluster has 14 CoolExploit servers, the other three CoolExploit and one BlackHole 2.0. This agrees with external reports on the Reveton gang switching from BlackHole to the newer CoolExploit kit [14]. Here, the clustering captures an operation using different exploit kits, but possibly underclusters as both clusters likely belong to the same operation.

Winwebsec operation. We observe the winwebsec fake AV affiliate program distributed through 22 different servers in eight clusters. There exists three singleton clusters, each exclusively distributing the winwebsec executable of a different affiliate. Another cluster of eight servers distributes affiliate 60830 as well as another unknown malware family and zbot. The other four clusters distribute the executables of multiple affiliates. Here, there exist two possibilities: The same group could have signed up to the winwebsec program multiple times as different affiliates, or the affiliate program is managing the exploit server so that affiliates can convert their traffic into installs. To differentiate between both cases, we check their landing URLs. One of these clusters uses the same landing URL to distribute the executables of affiliates 66801, 66802, and 66803. In this case, there is no way to separate the installs due to each affiliate, which indicates those affiliates belong to the same entity. The other three clusters use different landing URLs for each affiliate, which indicates those servers are run by the affiliate program, which provides a distinct landing URL to each affiliate.

We confirm that the winwebsec program manages their own exploit servers through external means. We leverage a vulnerability on old versions of BlackHole, where the malware URLs used a file identifier that was incremented sequentially, and thus could be predicted. On March 12, we tried downloading file identifiers sequentially from one of the servers distributing multiple winwebsec affiliates. We found 114 distinct executables, of which 108 were winwebsec executables for different affiliates, one did not execute, and the other five corresponded to other malware families, including smarthdd and the Hands-up ransomware [59]. This indicates that on March 12, the winwebsec program had 108 affiliates and that the winwebsec managers, in addition to their own program, were also distributing other rogue software.

Zeroaccess operations. Zeroaccess is also an affiliate program [56]. With the aggressive algorithm, there are eight clusters distributing zeroaccess: five distribute a single affiliate identifier, the other three multiple. For two of these three, the distribution is simultaneous and on a different landing URL for each affiliate, which indicates that the zeroaccess affiliate program also manages their own exploit server. The other distributes two affiliate identifiers on the same URL, indicating those affiliates belong to the same entity.
Zbot operations. There are 51 clusters distributing zbot in the aggressive clustering. Of these, 38 clusters distribute exclusively zbot, the largest using 21 servers over six days. For each of these 38 clusters, we compute the set of C&C domains contacted by the malware milked from servers in the cluster. Only three of the 38 clusters have C&C overlap, which indicates that our non-family features capture enough shared configuration to differentiate operations distributing the same malware kit.
Broken malware operation. We identify a cluster with 13 servers that operates on a single day and distributes a single file. Surprisingly, the file does not execute. Apparently, the malware owners realized the malware was corrupt and stopped the operation.
Operations summary. The clustering reveals two types of operations. Two-thirds of the clusters are singletons. They correspond to small operations with one server that lives on average 14 h. Most singletons distribute a single family, which is often zbot or one of the generic families for which we have not found a published name. The remaining is operations that leverage multiple servers for their distribution. Multi-server operations use on average 6.2 servers and diversify their hosting. On average, each multi-server operation hosts 1.2 servers per country, and two servers per AS. Multi-server operations last longer with a median life of 5.5 days and only 1.2 servers operate on the same day. This indicates that they are replacing servers over time to sustain distribution, rather than using them for sudden bursts of installs (although we observe bursts like the broken malware operation mentioned earlier).
Feature set comparison. We compare the feature sets in our prior work [32] with the new feature sets, which add the consecutive IP addresses and exploit hash features. With aggressive clustering, the new features reduce the number of clusters and increase the average cluster size, while with PAM the two new features further separate the clusters, slightly increasing their number. With aggressive clustering, the exploit hash feature is most helpful, merging 13 small clusters into the largest cluster (86 servers). To verify that indeed the exploit hash feature is capturing exploit servers that share management, we analyze the time distribution of the exploit instances that are served by multiple servers, finding that their distribution happens very close in time. This supports our hypothesis that once the exploits are repacked by the managers, they are distributed to the different exploit servers in the operation. The consecutive IP addresses feature merges five old clusters into two new clusters. These new clusters correspond to operations distributing zbot and cridex. The new clusters show that the miscreants are running 4–6 servers in the same cloud hosting provider, with all exploit servers using consecutive IPs. This indicates that if a cloud hosting server is abused the miscreants may install multiple servers in that hosting provider.

6.5 Reporting analysis

We started sending abuse reports on September 3, 2012 for exploit servers that we had been milking for 24 h. Most abuse reports did not produce any reply. Of the 19 reports we sent, we only received a reply in seven; 61 % of the reports were not acknowledged. For two of the ISPs, we were unable to locate an abuse@domain address in WHOIS. One of these had no technical support contact either, so we resorted to web searches to find an email address. The absence of an abuse@domain address indicates a lack of interest in abuse reports. As expected, those reports did not produce a reply.

All initial replies contained a ticket number, to be included in further communications about the incident. Three of them also provided a URL for a ticket tracking system. Two of the replies came from ISPs to whom we had sent more than one report (on different dates). Surprisingly, only one of the two reports produced a reply. This lack of consistency indicates manual processing and that the response to an incident may depend on the abuse team member that first reviews the report.

After reporting a server, we keep milking it to understand how long it takes to act on a report. Note that, these reaction times are lower bounds because the servers could have been reported earlier by other parties. On average, an exploit server lives 4.3 days after a report. Exploit servers whose report did not generate a response lived on average for 5.1 days after our report. Servers whose report produced a reply lived for 3.0 days. Thus, the probability of action being taken on the report when no reply is received is significantly smaller. Next, we detail the reactions to the seven reports with replies.
Report 1. The most positive report. The exploit server was a VPS hosted by the ISP, which immediately disconnected it and notified us of the action (which we confirmed).
Report 2. This large US ISP replied with an automated email stating that they take abuse reports seriously, but cannot investigate or respond to each of them. No further reply was received, and the server lived for four days.
Report 3. A ticket was open with medium priority promising further notification. No further response was received, and the server lived for another day.
Report 4. The report was forwarded to a customer. After a day, the server was still alive so we sent a second report

stating that the customer had not taken action and the ISP proceeded to disconnect the server.
Report 5. The report was forwarded to a customer, and our ticket closed without waiting for the customer's action. The server was still alive for 1.7 days.
Report 6. The reply stated they would try to get back within 24 h and definitely before 72 h. The server lived two more hours, and we never heard back.
Report 7. The initial reply stated that it was a customer's server and that according to the Dutch Notice and Take-down Code of Conduct [31], we had to notify the customer directly. Only if the customer did not reply after five days, or their reply was unsatisfactory, we could escalate it to them. We reported it to the client and after five days the server was still alive. We re-reported the exploit server to the ISP who told us to contact the customer again, which we did copying the ISP. This time the customer replied, but was not willing to act on the response unless we reveal our real identity, which we declined. It seems that the ISP called them requesting the disconnection. The ISP later notified us about the disconnection. As far as we can tell, the five days waiting time is not part of the Dutch Notice and Take-down Code of Conduct.

These reports show that if the exploit server is hosted by a hosting provider who is a customer of the ISP, the ISP simply forwards them the abuse report and does no follow-up. It is up to the reporter to monitor the customer's action and re-report to the ISP in case of inaction. They also show how painful abuse reporting can be and the need for an homogeneous code of conduct for takedowns.

7 Malicia dataset

To foster future research, e.g., in malware classification, we have compiled the data collected in this work into the MALICIA dataset, which we make available to other researchers. The MALICIA dataset can be requested by researchers at academia, research laboratories, and industry laboratories following the instructions at the dataset's webpage [27]. It is released under an agreement to not redistribute the dataset and only to researchers under contract from a research institution. Students need to ask their supervisors to request to the dataset on their behalf. We use basic identity checks (e.g., that the email address from which the request is sent belongs to the institution requesting it) before releasing the dataset. At the time of writing, the dataset has been released to 17 institutions: 15 universities and two industrial research laboratories.

This section briefly describes the publicly available 1.0 release of the MALICIA dataset. Then, it describes the updates that we plan to add into release 1.1, which we will make available upon publication of this manuscript.

7.1 Release 1.0

The current release of the MALICIA dataset contains data for all the milking period (March 7th, 2012–March 25th, 2013). The dataset comprises five files. The core of the dataset is a MySQL database with all the milking metadata including when the malware was collected, from where the malware was collected, the malware classification, and exploit server information. In addition, there is a figure that captures the database schema, a tarball with the malware binaries, another tarball with the icons extracted from those malware binaries, and a signature file for the Snort IDS produced by our FIRMA tool.
Database. The database comprises eight tables. The most important table in the DB is the MILK table, which contains a row for every time a malware was milked from a exploit server. Every row contains the timestamp of when the malware was milked, the landing URL, as well as identifiers linking to the other DB tables. Other two important tables are the FILES and LANDING_IP tables. The FILES table contains a row for each unique malware binary (identified by SHA1 hash) and its classification information. The LANDING_IP table contains a row for each exploit server (identified by the landing IP), the exploit kit installed in the server, the server's autonomous system number, and its country code.
Malware. The malware tarball contains 11,363 samples (.exe and .dll files). For each malware binary, its network traffic, icon, and screenshot labels, as well as the final family label, can be found in the FILES table in the DB.
Icons. The icons tarball contains 5,777 icons extracted from the executables. We provide the icons for convenience, since they can be extracted from the provided malware.

7.2 Release 1.1

In the next release of the MALICIA dataset, we plan to include the exploits extracted from the network captures of our honeyclients (Sect. 3.4) and additional malware classification information.
Exploits. This release will include an exploits tarball with the 172 exploits collected, named using their SHA1 hash. The database will contain an additional table where each row represents one exploit file and contains its hash, size, file type, and CVE label.
Updated malware classification. The 1.0 release includes one month of unclassified malware samples, which were collected after the DIMVA 2013 paper was written, but before the dataset was released. The 1.1 release will update the malware classification so that it covers all samples and corresponds to the classification data used in this manuscript.

8 Discussion

In this section, we discuss implications of our work, avenues for improvement, and suggest other applications.

Takedown challenges. Setting up a exploit server in the cloud is a simple and cheap process, while taking down a exploit server can be a complicated one and may cost little to the exploit server owner, which can simply move to another provider. We need to raise the cost for the exploit server managers of one of their exploit servers being taken down and make it less anonymous to rent them. Hosting providers should incorporate processes to verify the identity of the renter, assign reputation to payment accounts, and closely monitor short leases, as these are more likely to be abused. More emphasis is also needed on the attribution of the organizations running drive-by operations, and those providing the specialized underground services supporting them. We believe that this work is a step in that direction.

Criminal investigations. Our clustering is designed to be used by law enforcement during the pre-warrant (plain view) phase of a criminal investigation [53]. During this phase, criminal activity is monitored and targets of importance are selected among suspects. Our clustering can identify large operations among all reported exploit servers, satisfying this requirement. The goal of the plain view phase is gathering enough evidence to obtain a magistrate-issued warrant for the ISPs and hosting providers for the servers in the operation.

Additional features. Other features can be incorporated to our clustering to further identify servers with shared configuration. For example, we could incorporate the web server version running in the exploit server and the registrant for DNS domains pointing to the servers.

Improving coverage. In this work, we show that even with a small number of drive-by download feeds, we can identify exploit servers in the same operation. Adding more feeds would improve our coverage, identifying more operations. Acquiring feeds is challenging because many security vendors are careful about sharing them since they consider them a competitive advantage.

Other applications. The problem of distinguishing from a pool of servers running the same software which servers are managed by the same organization is not exclusive to exploit servers. Malware kits pose similar challenges because they provide configurable bot and C&C server software, shared among all organizations buying the kit. Our technique could be applied to this scenario to identify C&C servers in the same operation.

9 Related work

A number of works have analyzed drive-by downloads. Wang et al. [54] build honeyclients to find websites that exploit browser vulnerabilities. Moshchuk et al. [30] use honeyclients to crawl over 18 million URLs, finding that 5.9 % contained drive-by downloads. Provos et al. [38] describe a number of exploitation techniques used in drive-by downloads. They follow-up with a large-scale study on the prevalence of drive-by downloads finding that 67 % of the malware distribution servers were in China [37]. Recently, Grier et al. [19] investigate the emergence of exploit kits and exploitation as-a-service in the drive-by downloads ecosystem, showing that many of the most prominent malware families propagate through drive-by downloads. Our work differs from prior drive-by downloads analysis in that we focus on identifying and understanding the properties of drive-by operations, rather than individual exploit servers. Other work proposes detection techniques for drive-by downloads [11,13,62] and could be incorporated into our infrastructure.

Cho et al. [10], infiltrated the MegaD spam botnet and collected evidence on its infrastructure being managed by multiple botmasters. In contrast, our work shows how to automate the identification of servers with shared management, grouping them into operations. In simultaneous work, Canali et al. [9] analyze the security of shared hosting services. Similar to their work, we also issue abuse reports to hosting providers, but our focus is on VPS services, which are more adequate for hosting exploit servers.

Prior works on running malware in a controlled environment have influenced our malware execution infrastructure [21,24,43]. Our classification builds on a number of prior works on behavioral classification techniques [2,4,5,7,19,34,36,41] and incorporates the automated clustering of malware icons using perceptual hashing. We could also incorporate techniques to reduce the dimensionality in malware clustering [20], to evaluate malware clustering results using AV labels [35], and to "personalize" the generated network fingerprints to the network where they are deployed [33].

10 Conclusion

We have proposed a technique to identify drive-by download operations by clustering exploit servers under the same management based on their configuration, the exploits they serve, and the malware they distribute. Our analysis reveals that to sustain long-lived operations miscreants are turning to the cloud. We find that 60 % of the exploit servers are hosted by specialized cloud hosting services. We have performed what we believe is the first quantitative analysis of exploit polymorphism. We observe that different types of exploits are repacked differently; repacking can be integrated in the exploit kit to be performed automatically, invoked manually using external tools, and some exploit types (e.g., fonts) may not be repacked at all. We have also analyzed the abuse reporting procedure with discouraging results: most abuse reports

go unanswered and even when reported, it still takes several days to takedown an exploit server.

Acknowledgments The authors would like to thank Chris Grier and Kurt Thomas for their help and the anonymous reviewers for their insightful comments. This work was supported in part by the European Union through the FP7 network of excellence NESSoS (Grant FP7-ICT No. 256980), by the Spanish Government through the StrongSoft project (Grant TIN2012-39391-C04-01) and a Juan de la Cierva Fellowship for Juan Caballero, by the N-Greens CM project, by the Research Fund KU Leuven, and by the Fight against Crime Programme of the European Union (B-CCENTRE). Opinions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

1. Allatori java obfuscator. http://www.allatori.com/
2. Anderson, D.S., Fleizach, C., Savage, S., Voelker, G.M.: Spamscatter: characterizing internet scam hosting infrastructure. In: USENIX Security Symposium, Boston, MA (August 2007)
3. An overview of exploit packs (update 20) Jan (2014) http://contagiodump.blogspot.com.es/2010/06/overview-of-exploit-packs-update.html
4. Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: International Symposium on Recent Advances in Intrusion Detection, Queensland, Australia (September 2007)
5. Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: Network and Distributed System Security Symposium, San Diego, CA (February 2009)
6. Bfk: Passive dns replication. http://www.bfk.de/bfk_dnslogger.html
7. Caballero, J., Grier, C., Kreibich, C., Paxson, V.: Measuring pay-per-install: the commoditization of malware distribution. In: USENIX Security Symposium, San Francisco, CA (August 2011)
8. Caida: As Ranking (October 2012). http://as-rank.caida.org
9. Canali, D., Balzarotti, D., Francillon, A.: The role of web hosting providers in detecting compromised websites. In: International World Wide Web Conference, Rio de Janeiro, Brazil (May 2013)
10. Cho, C.Y., Caballero, J., Grier, C., Paxson, V., Song, D.: Insights from the inside: A view of botnet management from infiltration. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats. San Jose, CA, April (2010)
11. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: International World Wide Web Conference, Raleigh, NC (April 2010)
12. Crocker, D.: Mailbox Names for Common Services, Roles and Functions. RFC 2142 (May 1997)
13. Curtsinger, C., Livshits, B., Zorn, B., Seifert, C.: Zozzle: low-overhead mostly static javascript malware detection. In: USENIX Security Symposium, San Francisco, CA (August 2011)
14. Cool exploit kit—a new browser exploit pack. http://malware.dontneedcoffee.com/2012/10/newcoolek.html/
15. Daigle, L.: Whois Protocol Specification. RFC 3912 (September 2004)
16. Dunn, J.C.: Well-separated clusters and optimal fuzzy partitions. J. Cybern. **4**(1), 95–104 (1974)
17. Falk, J.: Complaint Feedback Loop Operational Recommendations. RFC 6449 (November 2011)
18. Falk, J., Kucherawy, M.: Creation and Use of Email Feedback Reports: An Applicability Statement for the Abuse Reporting Format (arf). RFC 6650 (June 2012)
19. Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C.J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., Provos, N., Rafique, M.Z. Rajab, M.A., Rossow, C., Thomas, K., Paxson, V., Savage, S., Voelker, G.M.: Manufacturing compromise: the emergence of exploit-as-a-service. In: ACM Conference on Computer and Communications Security, Raleigh, NC (October 2012)
20. Jang, J., Brumley, D., Venkataraman, S.: Bitshred: feature hashing malware for scalable triage and semantic analysis. In: ACM Conference on Computer and Communications Security. Chicago, IL (October 2011)
21. John, J.P., Moshchuk, A., Gribble, S.D., Krishnamurthy, A.: Studying spamming botnets using Botlab. In: Symposium on Networked System Design and Implementation, Boston, MA (April 2009)
22. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis, vol. 4. Wiley, New York (1990)
23. Krawetz, N.: Average Perceptual Hash (May 2011). http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html
24. Kreibich, C., Weaver, N., Kanich, C., Cui, W., Paxson, V.: GQ: Practical containment for measuring modern malware systems. In: Internet Measurement Conference, Berlin, Germany (November 2011)
25. Li, Z., Alrwais, S., Xie, Y., Yu, F., Wang, X.: Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In: IEEE Symposium on Security and Privacy, San Francisco, CA (May 2013)
26. Love vps http://www.lovevps.com/
27. Malicia project http://malicia-project.com/
28. Malware domain list http://malwaredomainlist.com/
29. Morrison, T.: How Hosting Providers can Battle Fraudulent Sign-ups (October 2012). http://www.spamhaus.org/news/article/687/how-hosting-providers-can-battle-fraudulent-sign-ups
30. Moshchuk, A., Bragin, T., Gribble, S.D., Levy, H.M.: A crawler-based study of spyware on the web. In: Network and Distributed System Security Symposium, San Diego, CA (February 2006)
31. New Dutch Notice-and-Take-Down Code Raises Questions (October 2008). http://www.edri.org/book/export/html/1619
32. Nappa, A., Rafique, M.Z., Caballero, J.: Driving in the cloud: an analysis of drive-by download operations and abuse reporting. In: SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment, Berlin, Germany (July 2013)
33. Nelms, T., Perdisci, R., Ahamad, M.: Execscent: mining for new c&c domains in live networks with adaptive control protocol templates. In: USENIX Security Symposium, Washington, DC (August 2013)
34. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: Symposium on Networked System Design and Implementation, San Jose, CA (April 2010)
35. Perdisci, R., Vamo, M.U.: Towards a fully automated malware clustering validity analysis. In: Annual Computer Security Applications Conference, Orlando, FL (December 2012)
36. Polychronakis, M., Mavrommatis, P., Provos, N.: Ghost turns zombie: exploring the life cycle of web-based malware. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Francisco, CA (April 2008)
37. Provos, N., Mavrommatis, P., Rajab, M.A., Monrose, F.: All your iframes point to us. In: USENIX Security Symposium, San Jose, CA (July 2008)
38. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser: analysis of Web-based malware.

In: USENIX Workshop on Hot Topics on Understanding Botnets, Cambridge, UK (April 2007)
39. Rafique, M.Z., Caballero, J.: Firma: Malware clustering and network signature generation with mixed network behaviors. In: International Symposium on Recent Advances in Intrusion Detection, St. Lucia (October 2013)
40. Rafique, M.Z., Huygens, C., Caballero, J.: Network Dialog Minimization and Network Dialog Diffing: Two Novel Primitives for Network Security Applications. Technical Report TR-IMDEA-SW-2014-001, IMDEA Software Institute, Madrid, Spain (March 2014). https://software.imdea.org/~juanca/papers/TR-IMDEA-SW-2014-001.pdf
41. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment, Paris, France (July 2008)
42. Rossow, C., Dietrich, C.J.: Provex: Detecting botnets with encrypted command and control channels. In: SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment, Berlin, Germany (July 2013)
43. Rossow, C., Dietrich, C.J., Bos, H., Cavallaro, L., van Steen, M., Freiling, F.C., Pohlmann, N.: Sandnet: network traffic analysis of malicious software. In: Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, Salzburg, Austria (April 2011)
44. Ssdsandbox. http://xml.ssdsandbox.net/dnslookup-dnsdb
45. Shafranovich, Y., Levine, J., Kucherawy, M.: An Extensible Format for Email Feedback Reports. RFC 5965 (August 2010). Updated by RFC 6650
46. Shue, C., Kalafut, A.J., Gupta, M.: Abnormally malicious autonomous systems and their internet connectivity. IEEE/ACM Transactions of Networking **20**(1), (2012)
47. Snort http://www.snort.org/
48. Stone-Gross, B., Christopher, Kruegel, Almeroth, K., Moser, A., Kirda, E.: Fire: Finding rogue networks. In: Annual Computer Security Applications Conference, Honolulu, HI (December 2009)
49. Suricata http://suricata-ids.org/
50. The spamhaus project (October 2012) http://www.spamhaus.org/
51. Urlquery. http://urlquery.net/
52. Virustotal. http://www.virustotal.com/
53. Walls, R.J., Levine, B.N., Liberatore, M., Shields, C.: Effective digital forensics research is investigator-centric. In: USENIX Workshop on Hot Topics in Security, San Francisco, CA (August 2011)
54. Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.: Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In: Network and Distributed System Security Symposium, San Diego, CA (February 2006)
55. Wepawet. https://wepawet.iseclab.org/
56. Wyke, J.: The Zeroaccess Botnet: Mining and Fraud for Massive Financial Gain (September 2012). http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess-botnet.asp:x
57. X-arf: Network abuse reporting 2.0. http://x-arf.org/
58. Xylitol: Blackhole exploit kits update to v2.0 (September 2011). http://malware.dontneedcoffee.com/2012/09/blackhole2.0.html
59. Xylitol: Tracking Cyber Crime: Hands Up Affiliate (Ransomware) (December 2011). http://www.xylibox.com/2011/12/tracking-cyber-crime-affiliate.html
60. Zauner, C.: Implementation and Benchmarking of Perceptual Image Hash Functions. Master's thesis, Upper Austria University of Applied Sciences, Hagenberg, Austria (July 2010)
61. Zelix klassmaster heavy duty protection. http://www.zelix.com/klassmaster/
62. Zhang, J., Seifert, C., Stokes, J. W., Lee, W.: Arrow: Generating signatures to detect drive-by downloads. In: International World Wide Web Conference, Hyderabad, India (April 2011)