Measuring *Pay-per-Install*: The Commoditization of Malware Distribution

Juan Caballero[†]*, Chris Grier*[*‡]*, Christian Kreibich*[*‡]*, Vern Paxson*[*‡]
[†]*IMDEA Software Institute* [*]*UC Berkeley* [‡]*ICSI*
juan.caballero@imdea.org {*grier, vern*}*@cs.berkeley.edu* *christian@icir.org*

Abstract

Recent years have seen extensive diversification of the "underground economy" associated with malware and the subversion of Internet-connected systems. This trend towards specialization has compelling forces driving it: miscreants readily apprehend that tackling the entire value-chain from malware creation to monetization in the presence of ever-evolving countermeasures poses a daunting task requiring highly developed skills and resources. As a result, entrepreneurial-minded miscreants have formed pay-per-install (PPI) services—specialized organizations that focus on the infection of victims' systems.

In this work we perform a measurement study of the PPI market by infiltrating four PPI services. We develop infrastructure that enables us to interact with PPI services and gather and classify the resulting malware executables distributed by the services. Using our infrastructure, we harvested over a million client executables using vantage points spread across 15 countries. We find that of the world's top 20 most prevalent families of malware, 12 employ PPI services to buy infections. In addition we analyze the targeting of specific countries by PPI clients, the repacking of executables to evade detection, and the duration of malware distribution.

1 Introduction

Recent years have seen extensive diversification of the "underground economy" associated with malware and the subversion of Internet-connected systems. This trend towards specialization has compelling forces driving it: miscreants readily apprehend that tackling the entire value-chain from malware creation to monetization in the presence of ever-evolving countermeasures poses a daunting task requiring highly developed skills and resources. As a result, market forces foster a *service culture* that has brought about a wide range of specialized providers for all stages in the malware-monetization lifecycle, such as malware toolkits [3, 15], packing tools to evade antivirus (AV) software [21], "bullet-proof" hosting [4], and forums for buying and selling ill-gotten gains [10].

At the heart of this ecosystem lies the *infection* of victim computers. Virtually every enterprise in this market ultimately hinges on access to compromised systems. To meet the demands for wholesale infection of Internet systems, a service called *pay-per-install* (PPI) has risen to predominance. Such PPI services play a key role in the modern malware marketplace by providing a means for miscreants to *outsource* the global dissemination of their malware. Miscreants simply determine the raw number of victim systems (including specific geographical distribution, if desired) that fits within their budget, supply a PPI service with payment and malware executables of the miscreants' choice, and in short order their malware is installed on thousands of new systems. In today's market, the entire process costs *pennies* per target host—cheap enough for botmasters to simply rebuild their ranks from scratch in the face of defenders launching extensive, energetic, take-down efforts [6].

In this work we perform a measurement study of the PPI market by *infiltrating* four PPI services. We develop infrastructure that enables us to (1) interact with PPI services by mimicking the protocol interactions they expect to receive from *affiliates* with whom they have contracted, and (2) gather and classify the resulting malware executables as distributed by the PPI services. We report results of infiltrations we conducted in the six months between August 2010 and February 2011.

To our knowledge, our work reflects the first systematic study of the PPI ecosystem as seen from the perspective of the downloads pushed out by PPI services down to their victims. Security analysts have previously exam-

ined PPI services in a top-down manner, by becoming affiliates of particular services [7, 29]. Our study is instead based on infiltrating PPI services in a bottom-up manner, by creating custom programs that can continuously download malware specimens that the PPI services distribute, enabling us to track the infiltrated PPI services over time.

We harvested over a million client executables using vantage points spread across 15 countries. The month of August 2010 yielded 57 malware families, including many of the most prevalent infections at the time. They include spam bots (*Rustock*, *Grum*), fake antivirus (*Securitysuite*, *Securityessential*), information-stealing trojans (*Zbot*, *Spyeye*), rootkits (*Tdss*), DDoS bots (*Russkill*, *Canahom*), clickers (*Gleishug*), and adware (*SmartAdsSolutions*).

Using our geo-diverse vantage points, we measure differences in the geographical preferences of the different malware families. We identify families that exclusively target the US, the UK, and a variety of European countries. We also analyze the rate at which malware authors *repack* their wares to evade hash-based signatures. On average, they repack specimens every 11 days, and some malware families repack up to twice daily. We track the dynamics of *campaigns* during which a service disseminates a given malware family in an ongoing push, observing a wide temporal range, from specimens that are continually distributed over weeks, to pointwise efforts lasting only a few hours. We also analyze the particulars of how different PPI services interact with their affiliates, including surprising evidence suggesting that some affiliates who sell installs to a particular PPI service not only buy installs from rival PPI services, but also from the very service to which they sell installs—apparently to exploit arbitrage.

2 An Overview of Pay-Per-Install

The PPI market, as depicted in Figure 1, consists of three main actors: *clients*, *PPI providers* (or *services*), and *affiliates*. We begin with an overview of these actors, followed by discussion of the transactions they perform (Section 2.1) and the means and importance of evading detection (Section 2.2).

Clients are entities that want to install programs onto a number of target hosts. They wish to *buy installs* of their programs. The PPI provider receives money from clients for the service of installing their programs onto the target hosts, where installation comprises distributing the programs to the target hosts, executing the client programs, and tracking successful executions for accounting.

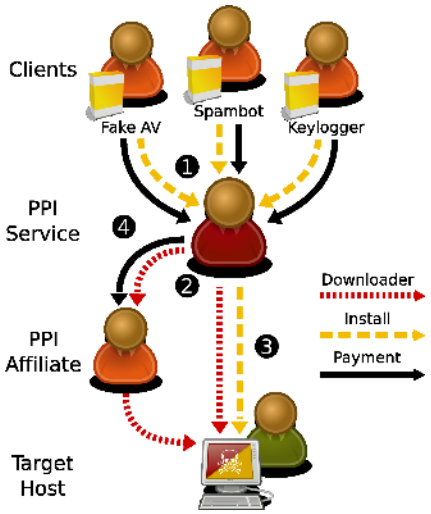

Figure 1: The typical transactions in the PPI market. PPI clients provide software they want to have installed, and pay a PPI service to distribute the software (❶). The PPI service conducts downloader infections itself or employs affiliates that install the PPI's downloader on victim machines(❷). The PPI service pushes out the client's executables (❸). Affiliates receive commission for any successful installations they facilitated (❹).

The PPI provider develops a program, called a *downloader*, that retrieves and runs client's executables upon installation. The PPI provider may conduct the installation of the downloader itself or may outsource distribution to third parties called *affiliates*. When a provider has affiliates, the provider acts as a middle man that sells installs to the clients while buying installs from affiliates that specialize in some specific distribution method (e.g., bundling malware with a benign program and distributing the bundle via file-sharing networks; drive-by-download exploits; or social engineering). PPI providers pay affiliates for each target host on which they execute the provider's downloader program. Once the downloader runs, it connects to the PPI provider to download the client programs. If the PPI provider does the distribution itself, we call the service a *direct PPI service*. If the PPI provider runs an affiliate program, we call it an *affiliate PPI service*.

In general, both reputable and not-so-reputable entities use PPI services. In this paper we focus on the use of PPI services as a distribution mechanism for malware, e.g., bots, trojans, fake AV software, and spyware. To

avoid determining what constitutes malware, we limit the scope of the paper to PPI services that perform (or allow their affiliates to perform) *silent installs* on the target hosts, i.e., installations that lack the *informed consent* of the owner of the system. Hereafter we use the term *PPI providers* to refer exclusively to those providers that perform or facilitate silent installs.

2.1 The PPI Ecosystem

We describe the PPI ecosystem in terms of the transactions that take place between clients and PPI providers, and between PPI providers and their affiliates.

Clients. Clients profit from the malicious activities enabled by malware they want to deploy on target hosts, such as click fraud, stealing user information (e.g., credit card numbers, credentials), or selling software to the user under false pretense (e.g., fake AV).

PPI providers allow clients to choose the geographic distribution of target hosts. This distinction creates price differentiation in the market due to varying demand for machines in certain regions and varying target host supply. Clients pay only per *unique* install, i.e., for one installation of their program on a given target host.

PPI providers. PPI providers profit from installation fees paid by the clients. PPI install rates vary from \$100–\$180 for a *thousand* unique installs in the most demanded regions (often the US and the UK, and more recently other European nations), down to \$7–\$8 in the least popular ones (predominantly Asia) [12, 13, 19]. In this study, we observe PPI providers installing multiple client programs on the same target host, and have not observed attempts to secure exclusive use of a target host on behalf of a client. Exclusivity of a host is difficult to guarantee because a PPI provider cannot generally know whether a target host already runs other malware (e.g., a rival PPI downloader that installs competitors of the client program). In addition, it is very difficult for clients to validate that the PPI service only installed their malware on a host.

Affiliate PPI services give their affiliates a PPI downloader program personalized with their unique affiliate identifier. The service credits affiliates for executing their specific PPI downloader on a target host. Affiliates only receive credit for *confirmed installs* of their PPI downloader. The confirmation takes the form of the PPI downloader sending the personalized affiliate identifier to the PPI provider after downloading and executing the client programs. Thus, affiliates receive credit only after delivering the installs.

Affiliates. Affiliates profit from the installs performed on behalf of the PPI provider, with the distribution method remaining transparent to the clients. Affiliates might in fact be botmasters that compromise hosts, install their own malware, and then task their malware with downloading and installing the PPI downloaders as one means for monetizing their botnet. When doing so, the botmaster relinquishes exclusive control of the hosts in exchange for the install payments from the PPI service. The same botmasters might work with multiple PPI providers simultaneously to maximize the income from each bot, installing multiple affiliate binaries on each of their hosts.

Indeed, the market has a somewhat fundamental conflict-of-interest, in that the more installs a botmaster/affiliate provides, the more payment they receive; but each install degrades the quality of previous installs, because the likelihood of the owner of the system discerning they have become infected, and remedying the situation, rises with the volume of malicious installs on the system.

2.2 Evading Detection

AV software may detect and block any program in the installation chain, making it difficult to sustain installs. Therefore, providing stealthy executables is a key objective for both PPI providers and clients. In the PPI ecosystem, clients are often in charge of making their programs stealthy before giving them to the PPI provider, while affiliates rely upon the PPI provider to provide them with a stealthy downloader.

To render programs stealthy, both PPI providers and clients employ *packer* programs sold by third parties [21, 23]. Packers change the program content so that its signature (e.g., MD5 hash) differs even though the program's functionality has not changed. Sophisticated packers may also change the program size and add detection techniques for debuggers and virtual machines, which are commonly used by analysts. PPI providers have responsibility for packing the PPI downloaders for each affiliate and testing that the resulting executable remains undetected by AV software. In addition, PPI providers instruct affiliates and clients *not* to test their programs on free malware scanners [30, 32], because these services often redistribute samples to AV vendors. The vendors may then add new signatures to their databases, thus uncloaking the programs. We analyze

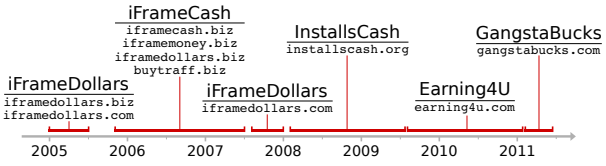

Figure 2: Brands used by the LoaderAdv PPI service over time. The domains under each brand correspond to known front-ends for affiliates.

how frequently clients repack their programs in Section 4.2.

3 Infiltrating PPI Infrastructure

In this section, we first describe how we identified the four PPI services we infiltrate, and evaluate our coverage of the PPI ecosystem. We then explain the processing pipeline we have developed for *milking* executables from PPI services and classifying them.

3.1 Identifying PPI services

A good starting point for identifying PPI services to infiltrate is PPI forums [27, 28], which mainly serve as a means for advertising affiliate PPI services to attract new affiliates. General underground forums sometimes offer the same advertisements. One challenge when studying PPI services concerns how to identify the different brands used by the same PPI service over time. We approached this task by analyzing public information, including copies of any old front-ends [14], forums used to advertise affiliate PPI services [27, 28], and previous analysis by security analysts [7, 29].

We selected four affiliate PPI programs for infiltration: *LoaderAdv*, *GoldInstall*, *Virut*, and *Zlob*. We use these names to refer to the respective PPI services, regardless of their branded program names over time. Figure 2 illustrates such branding, employed by the *LoaderAdv* service.

Our coverage. Several other PPI services exist that we did not infiltrate. To get an idea of our coverage of the malware ecosystem, we compare our malware harvest with contemporary reports by the security industry. In July 2010, FireEye posted the list of the top 20 malware families they observed using their network during April–June 2010 [22]. Table 1 correlates these 20 families with the contents of our "milked" malware corpus for August 2010. The column labeled *kit* designates families

	NAME	%	MONETIZATION	KIT	SEEN
1	Palevo	7.50	DoS,Info stealer	✓	✓
2	Hiloti	4.69	Downloader/PPI		✓
3	Zbot	3.62	Info stealer	✓	✓
4	FakeRean	3.47	Rogue AV(s)		✓
5	Onlinegames	2.94	Info stealer		?
6	Rustock	2.66	Spam		✓
7	Ldpinch	2.64	Info stealer	✓	?
8	Renos	2.58	Rogue AV(s)		?
9	Zlob	2.54	Rogue software		✓
10	Autoit	2.53	Downloader/PPI		
11	Conficker	2.48	Worm		
12	Opachki	1.95	Click Fraud		✓
13	Buzus	1.91	Info stealer		
14	Koobface	1.17	Downloader		
15	Alureon	1.16	Downloader	✓	✓
16	Bredolab	1.15	Downloader/PPI	✓	✓
17	Piptea	1.13	Downloader/PPI		✓
18	Ertfor	0.91	Rogue AV(s)		✓
19	Virut	0.91	Downloader/PPI		✓
20	Storm 2.0	0.80	Spam		

Table 1: FireEye's top 20 malware families observed in their MAX Cloud network on the April–June 2010 time period [22] and whether we observe them in our milk for August 2010.

that are *crimeware kits*, software that one can purchase and customize in order to build botnet variants. Each kit sold may represent an individual botnet with a separate owner. For popular kits such as *zbot*, many distinct botnets instances exist [33]. The column labeled *seen* indicates whether we see samples of the family in our milking data. We milk 12 of the top 20 families, remain unsure about the phylogeny of 3, and miss 5 (*AutoIt*, *Buzus*, *Conficker*, *Koobface*, *Storm 2.0*). We contacted FireEye to inquire about the 3 unknown families, and based on their response we believe they reflect generic tags used by AV vendors, rather than specific families of malware.

3.2 "Milking" PPI Providers

This section starts the description of our milking operations. Figure 3 illustrates its architecture from milking the executables until their classification.

PPI "milker" requirements. Each PPI service uses at least one downloader program. A PPI downloader has three main tasks to perform: download the client programs, execute them, and communicate successful installation to the PPI service for accounting. For each downloader used by a PPI service that we infiltrated, we built our own program that mimics the network com-

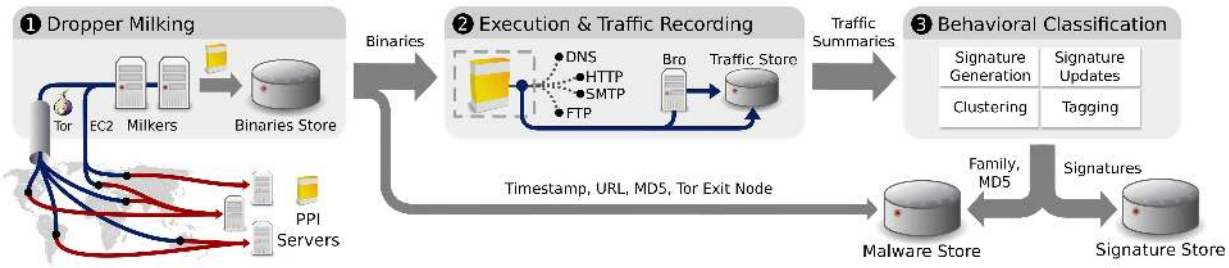

Figure 3: Architecture of our PPI milking system. The milkers contact the PPI services through Tor and store the executables for processing (❶). We then use Bro to distill network traffic summaries from packet traces recorded for each sample's contained execution (❷). A behavioral classifier then processes these summaries and stores clustering and tagging results to a database (❸).

munication used by the downloader to obtain the client programs, but does not implement the rest of the downloader's functionality, namely executing the client programs and accounting. In particular, we do our best to identify and avoid any accounting communication to prevent the PPI service from crediting an affiliate. We call such programs *milkers* because we use them to *milk* the client programs that the PPI provider distributes.

Although each PPI downloader program uses a different method to download the client programs from the PPI service, we observe two large classes. Basic PPI downloaders use plain HTTP and have a set of hard-coded URLs supplying client programs. The downloads remain unencrypted and could be spotted easily by any network monitoring device. The *LoaderAdv* and one of the *GoldInstall* downloaders (*GoldInstall-dl*) belong to this class. Advanced PPI downloaders have a proprietary, often encrypted, C&C protocol. These downloaders first contact the C&C infrastructure to receive the list of URLs supplying client programs. The *Zlob*, *Virut*, and an alternative *GoldInstall* downloader (*GoldInstall-list*) fall into this category. These downloaders still use HTTP for the downloads, at times encrypting the executables or disguising them as a benign file (e.g., by prefixing them with a fake GIF header).

Building the milkers. Building a milker is most challenging for downloaders using undocumented C&C protocols and encryption routines. Our approach leverages previously proposed techniques for automatic binary code reuse [5,16], which, given an executable, identify and extract parts of the executable related to a given function or specific functionality defined by the analyst. Our milker building process is semi-automatic because we also manually decompile parts of the extracted binary code. The final milker uses a mixture of C source code and assembly instructions. For this project, building and testing a basic milker required on average one day of full work, while the advanced milkers required from two to five days of work. It is worth noting that while building and testing the milker it is important to minimize the amount of traffic exchanged with the real C&C servers, which the PPI administrators may monitor. We learned this the hard way when the *Zlob* PPI service banned one of our computers during the testing phase. Moving to a different IP address fixed the issue.

Updating the milkers. All PPI services frequently change their download URLs to bypass blacklists. When a PPI service changes its download URLs, our advanced milkers simply download the updated list from the PPI C&C infrastructure and keep milking. However, our basic milkers, which have the old download URLs hard-coded, stop working until we update the URLs. To update the download URLs for the basic milkers, we first develop network signatures for the basic PPI downloaders. Then, we use two different approaches. First, we use the network signatures to look for new PPI downloaders within the executables we milk. If we find a match, our processing automatically extracts new URLs and adds them to our basic milkers. In addition, we also periodically query search engines and repositories that perform malware analysis [30] for any new traffic that matches the network signatures. Due to the prevalence of the PPI services in this study, we often find the new URLs in public repositories immediately after URLs change.

Anonymity and geographical diversity. To provide anonymity and geographical diversity for the milkers, we route them, when possible, through Tor [31]. A milker achieves geographical diversity by using 15 Tor circuits in parallel, each circuit terminating in an exit node in a different country. We chose these countries

in accordance with different price points advertised by PPI providers. We verify with the MaxMind GeoIP database [20] that the exit node's IP address indeed resides in the desired country. For *GoldInstall*, *Loader-Adv*, and *Virut*, we conduct all network communication through Tor. We cannot access *Zlob* through Tor. We suspect the *Zlob* operators blacklist the Tor exit nodes, which are publicly known. To achieve geographical diversity for this provider, we run its milkers on Amazon's EC2 cloud [9] from hosts in two different countries, without using Tor. We discuss the targets and results of geographically diverse milking in Section 4.4.

3.3 Running the Executables

We run each new milked executable under containment in the GQ malware farm [18], a platform for hosting all manner of malware-driven research in safe, controlled fashion. GQ confines each piece of malware in its execution by a custom, manually created containment policy that allows us to decide per-flow whether to allow traffic to interact with the outside, drop it, rewrite it, or reflect it to other machines inside the environment. In our scenario, the malware family and behavior is completely unknown when we run a newly milked sample. Thus, we create a containment policy that allows us to run all of our samples safely, and to classify them based on their network traffic.

We use this containment policy, called SinkAll, to automatically run thousands of executables, fully unsupervised. This policy blocks network connections and redirects them to internal *sink* servers within the farm. The only traffic from the malware allowed on the Internet is DNS. The reason for allowing DNS is to try to get the malware sample to attempt C&C communication, since part of our classification process (Section 3.4) examines the traffic content. While our DNS sink server could simply reply to all DNS requests with a valid response that includes a fixed IP address, some malware samples resolve benign domains (e.g., `microsoft.com`, `google.com`) and check the returned IP addresses against a hard-coded list in the malware. Thus, our DNS sink server proxies DNS requests and responses. If the DNS response is a failure, the sink server spoofs a successful DNS response with a fixed IP address to try to get the malware to attempt C&C communication.

SinkAll forwards all non-DNS TCP traffic from the malware to internal sink servers. For some well-known protocols, e.g., HTTP and SMTP, these servers mimic a valid session. This is important because some malware samples will test connectivity first using these protocols, and a valid session may entice them to attempt C&C communication. All other TCP traffic goes to a generic sink server that accepts arbitrary connections but does not provide a response; it simply completes the TCP handshake and accepts any data sent by the malware.

Finally, to detect anti-virtualization capabilities, samples that do not send any traffic are rerun on a bare (non-virtualized) host, also within the farm. (This did not often make a difference in practice.)

3.4 Classifying the Executables

We classify executables based on the network traffic they produce. First, we manually cluster them based on traffic similarity and create a *cluster signature*. Then, when possible, we tag clusters with names used by the community such as *Rustock* or *Palevo*.

Each run of a malware sample in the farm produces a trace of its network communication. We process the network trace with the Bro intrusion detection system [24], using a number of custom analysis scripts we developed. The scripts first check whether the sample generated any network traffic at all. If it did not, then we queue the executable for running on a bare host to check for anti-virtualization techniques. If the sample did generate traffic, we extract a number of features to characterize the network traffic that we later use during clustering.

The first feature is the list of protocols used by the sample. To extract this feature, we leverage Bro's dynamic protocol detection capabilities, which detects traffic for well-known protocols (e.g., DNS, HTTP, SMTP, and IRC), regardless of the port with which the communication happens [8]. Another feature is the list of end-points that communicate with the sample. For this, we extract from the DNS traffic the domains requested by the sample. If the sample starts a connection without a previous DNS request, we also add the IP address it contacts to the list of end-points. Another feature is the list of TCP/UDP destination ports for connections started by the sample. Finally, we extract a content feature from the payload of any connection. For any HTTP request originated by the malware, the content feature is the method and the list of parameters from the URL. We ignore the path in the URL and the parameter values because they tend to change often between samples. For other protocols, the content feature is simply the first 16 bytes sent by the malware.

We use the extracted features for clustering executables with similar network behaviors. In contrast to ex-

MILKER	DOWNLOADS	DISTINCT	START DATE
LoaderAdv	696,714	4,334	Aug 1, 2010
GoldInstall	361,325	4,488	Aug 1, 2010
Virut	4,841	72	Aug 1, 2010
Zlob	504	259	Jan 3, 2011
Total	1,060,895	9,153	

Table 2: Number of downloads and distinct MD5s collected from each PPI service, starting August 1, 2010 and ending February 1, 2010.

isting clustering systems for domain names [26], HTTP requests [25], and similar communication patterns [11], our system must accommodate any type of C&C, including custom binary protocols. In this work we therefore use our own, simple, clustering method, based primarily on manual inspection, but forsee integrating other approaches as the need arises.

Our clustering first groups all executables with identical features into a single cluster, with the list of features acting as the initial cluster signature. We then manually merge similar clusters, assigning the new cluster a signature of simply the disjunction of the signatures of each merged cluster. Using this process on the August 2010 milk, we identify 57 clusters. The cluster signatures vary from a domain list—of limited value due to continual updates to C&C domains—to binary and HTTP signatures that prove more useful long-term.

For tagging, we prioritize clusters by the total number of times we milked them. For each cluster we manually check if we can find labeled traffic that matches the cluster signature in public repositories and malware analysis reports. If so, we change the cluster tag to match the publicly available name. This process is painful due to the disparity of names used for the same families (and binaries) in the community. We were able to tag 35 of the 57 clusters. In Section 4.1 we describe the results from our classification.

4 Insights into the PPI Business

We now present results from our infiltration by analyzing the executables we collected. We began our milking operations on August 1, 2010. As of February, 2011, we downloaded 1,060,895 client executables, yielding 9,153 distinct binaries during approximately 6 months of infiltration. The modest proportion (0.8%) of unique executables arises due to our frequent milking, and the fact that our geo-diverse milking frequently retrieves the same executable from multiple locations. We began our infiltration with *LoaderAdv*, *GoldInstall*, and *Virut*, adding *Zlob* in Jan. 2011. Table 2 shows the breakdown of our harvest by PPI service. The download rate varies across PPI providers since each PPI has a different number of endpoints to download malware and our milkers access each through geo-diverse locations.

FAMILY	MILKED	DIST.	DAYS	CLASS	PPI
Rustock	61,017	15	31	spam	L
LoaderAdv-ack	60,770	62	31	ppi	L
CLUSTER: A	11,758	8	31	clickfraud	G
Hiloti	10,045	43	31	ppi	L
CLUSTER: B	8,194	9	31	?	G
Gleishug	7,620	15	31	clickfraud	L
Nuseek	5,802	2	30	clickfraud	G
Palevo2	16,101	21	29	botnet	G,L
Securitysuite	15,403	100	29	fakeav	L
Zbot	3,684	49	29	infosteal	G,L
CLUSTER: D	5,723	1	28	?	G
SmartAdsSol.	18,317	6	26	adware	L
Spyeye	4,522	16	25	infosteal	G,L
Securitysuite-avm	4,732	45	20	fakeav	L
Grum	2,974	54	20	spam	G,L
Tdss	4,893	12	19	ppi	G,L
Otlard	677	7	16	botnet	G,L
Blackenergy1	1,135	15	15	ddos	L
Palevo	2,594	2	14	botnet	G
Harebot	1,617	13	14	botnet	G,L,V

Table 3: Top 20 malware families we milked during August 2010. The columns indicate the total number of executables milked, distinct executables per family, the number of days seen, the families' general class, and PPI services that distribute the family: *LoaderAdv* (L), *GoldInstall* (G), *Virut* (V).

4.1 Family Classification

We developed a set of classification signatures and vetted them based on extensive manual analysis of the 313,791 executables we milked during August 2010. These signatures classify 92% of the total August downloads. If we then apply these same signatures to milk from September 2010, the proportion matched only diminishes to 86%, and for October 2010, 77%. Thus, in terms of classifying the most prevalent downloads, the power of such milk-derived signatures decays fairly slowly with time. (Certainly we do expect their power to diminish, however, as PPI providers acquire new clients, and existing clients release variants of their malware that no longer manifest the behavior targeted by our signatures.)

For the 8% of August downloads unmatched by our signatures, we have assigned a general label reflecting absence of any generated traffic. We manually evaluated the behavior of 243 executables in this group and confirmed that the executables appear corrupted and do not execute. We also ran most on bare hardware and confirmed that their failure to execute does not reflect anti-virtualization checks.

While our signatures work quite effectively for classifying the bulk of downloads, the picture changes if we instead consider *distinct* binaries (only 0.6% of the overall volume). For these, we classify only 36%. However, it is unclear that this latter figure holds much significance: a single malware specimen whose behavior we have not specifically classified can account for a large number of failures to classify distinct binaries if the specimen happens to be repacked frequently.

To examine the malware families distributed by each PPI provider, we limit our discussion to the August 2010 milk. Since the distributed malware changes over time, focusing on a single month facilitates a clear presentation of our results, while still spanning a significant breadth of activity. Table 3 lists the top 20 malware families we milked during August 2010, the number of times milked, the number of distinct executables, the number of days we saw the family being dropped, the overall class for the family's predominant activity ("botnet" represents generic malware platforms), and the different PPI services that distributed the family.

Some of the malware families are crimeware kits (*Palevo2*, *Spyeye*, *Zbot*, *Bredolab*), which means they may be distributed by otherwise independent clients. When computing statistics for individual clients, we thus remove these kits to avoid potential aliasing. We observe that out of the 20 malware families, 7 are distributed by more than one PPI service. If we assume each (non-kit) malware family belongs to one actor, the results show that clients do not feel tied to a single PPI provider.

Distribution over time. Figure 4 shows distribution timelines for each family we could label by activity class, for August 2010. We visualize availability continuously whenever a family was available at least once in three hours. We make several observations. Programs push clickbots at virtually all times, but DDoS platforms much more sporadically. The latter perhaps reflects some sort of *Just-In-Time* DDoS-for-hire service. With the exception of the *GoldInstall-list* downloader, we see PPI downloaders pushed for weeks at a time. Spambots show no uniform availability pattern: relatively short-lived push-outs for *Pushdo* and *Grum*, but continual push-outs for *Rustock*. In the PPI setting, botmasters can afford to push out their bots as convenient, which will keep the installs relatively "silent"; by contrast, propagation campaigns driven by social engineering (e.g., as used by Storm [17]) require more careful design and timing.

		DAYS TO REPACK		
FAMILY	# DISTINCT	MEAN	MIN	MAX
Rustock	15	2.12	0.00	8.51
LoaderAdv-ack	62	2.21	0.00	7.14
CLUSTER: A	8	7.46	2.63	12.34
Hiloti	43	0.76	0.00	2.58
CLUSTER: B	9	4.42	0.34	23.62
Gleishug	15	3.57	0.00	8.60
Nuseek	2	14.08	5.04	23.13
Palevo2	21	1.77	0.00	10.15
Securitysuite	100	0.37	0.00	1.17
CLUSTER: D	1	28.22	28.22	28.22

Table 4: Repacking rates for the 10 most-milked families (Aug. 2010), excluding crimeware kits. The columns show the number of distinct binaries and the mean/minimum/maximum time to repack, in days. A minimum time of zero means that one of the distinct executables appeared in only a single milking instance.

4.2 Repacking Rate

The rate at which malware distributors repack their products reflects their concern about content-driven AV signatures. In this section we analyze the repacking rate for the client programs that we milk, which are typically repacked by the client themselves. In addition, we describe how the *Zlob* service repacks their affiliate *downloader* binaries on-the-fly.

In the milk from August 2010, a malware family is repacked on average at least once every 11 days. Table 4 summarizes the individual repacking rate for the top 10 families (excluding crimeware kits) milked in August 2010. The data for the top 10 families shows that they are repacked on average every 6.5 days. This indicate that the top malware families are repacked more often than the average malware family. Among these families, the most often repacked are *Securitysuite* (more than twice a day) and *Hiloti* (at least once per day). *CLUSTER:D* has the slowest repacking rate, only 1 executable was seen during the month, followed by *Nuseek* (2 executables).

In Figure 5, we contrast the repacking of the *Rustock* and *Securitysuite* families (with two variants of the latter) over the course of August. We plot distinct vari-

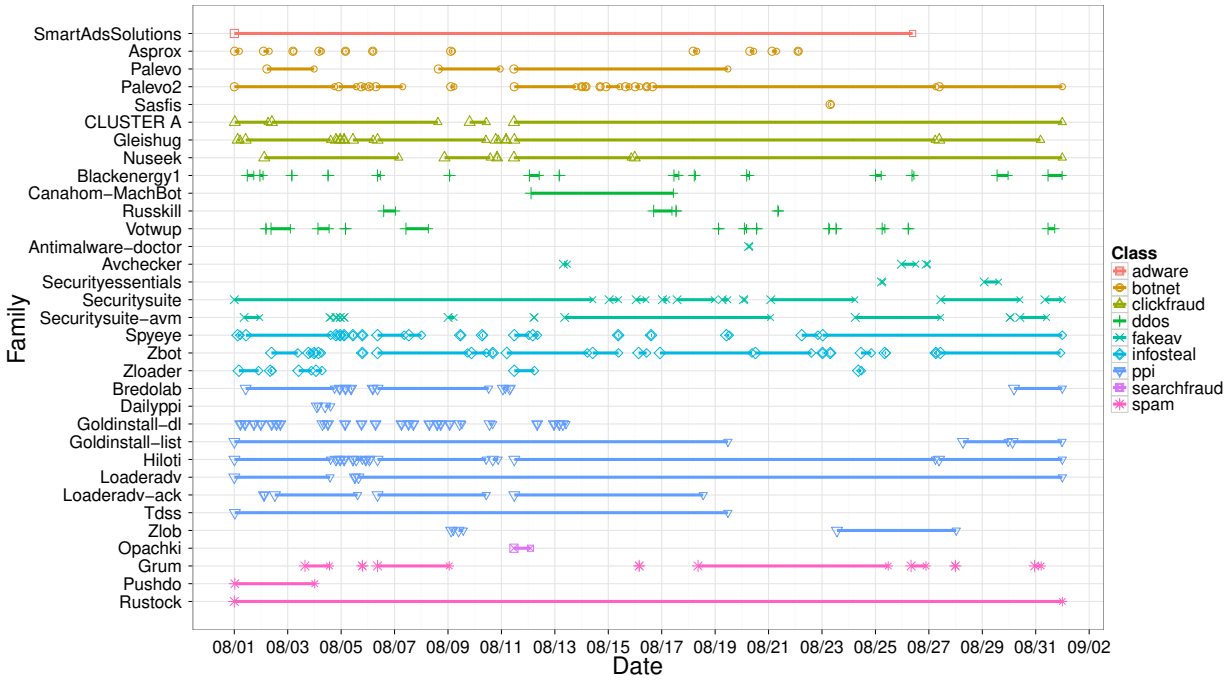

Figure 4: Malware family availability via infiltrated PPI services in August 2010. We only show families with a known activity class.

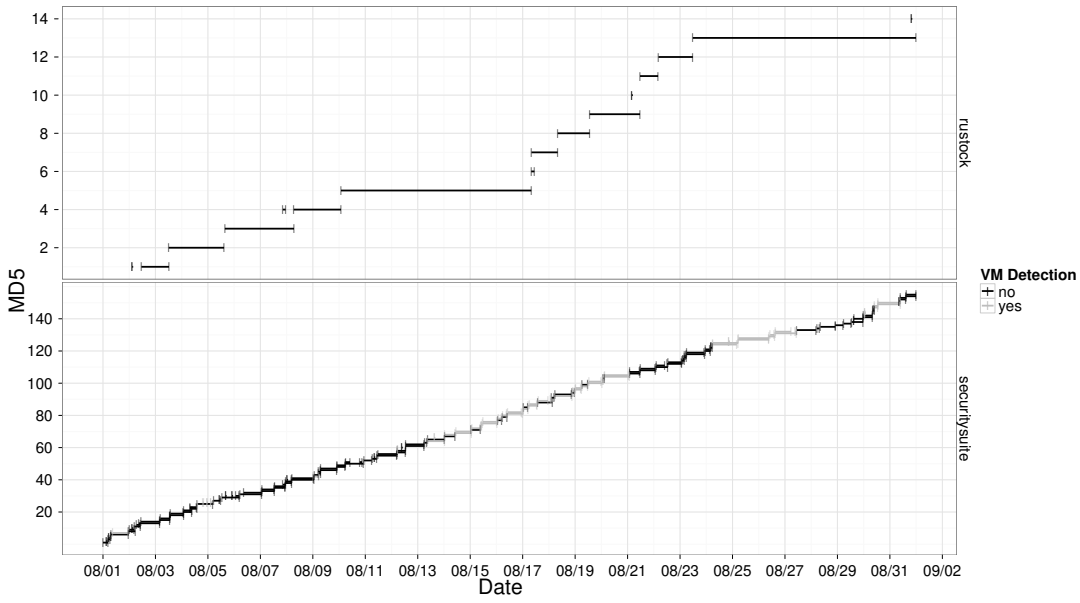

Figure 5: Repacking activity according to binary changes over time for the *Securitysuite* and *Rustock* families. Some *Securitysuite* binaries detected virtualized execution; we separate these by color.

ants on the y-axis, with entries ordered by first appearance. *Rustock* changes executables less frequently, and with little version overlap. Furthermore, the program dropped *Rustock* during the whole month, while *Securitysuite* has more complex availability: *Securitysuite-avm*, a *Securitysuite* subfamily with anti-VM capabilities (for VMware, specifically), filled the availability gaps when *Securitysuite* was not pushed out.[1] In aggregate, *Securitysuite* was thus likewise available throughout, though with differing anti-VM capabilities. One possible explanation is that the *Securitysuite* gang uses two off-the-shelf packers, but only one provides anti-VM capabilities.

Zlob affiliate downloader repacking. Unlike for the malware that their clients provide, PPI providers typically repack affiliate *downloader* binaries on a periodic basis and notify their affiliates to switch to the fresh downloader [29]. We found that the *Zlob* service has incorporated a twist on this approach. They provide a web service for affiliates to request a fresh binary, which, interestingly, apparently repacks the affiliate binaries on-the-fly. We requested the downloader for a single affiliate 27 consecutive times, resulting in 27 distinct, working *Zlob* binaries with identical sizes but differing MD5 hashes. Attackers could likewise apply such on-the-fly packing to other areas, such as drive-by-downloads, to create unique malware for each compromised host.

4.3 PPI Behavior

In this section we look at the behavior and distinct structure manifested by each PPI provider for managing their downloads.

LoaderAdv. The *LoaderAdv* downloader has hard-coded two domains and a set of file paths that it combines with the two domains to create the URLs to locate the malware executables. If we ignore the domain part of the URL (the second domain is only used for redundancy) we observe two classes of URLs: single-client and multi-client. Single-client URLs always return the same family of malware, while multi-client URLs cycle through a set of clients that changed over the course of our infiltration. These latter also yielded different downloads based on the geo-location of the milker's IP address, an aspect we examine further in Section 4.4.

Figure 6 shows the behavior of a single multi-client URL as seen by our milkers. We show the different families in separate boxes, and the y-axis represents the countries involved. (The gaps on August 5 and 11 arise due to failures of the milkers to connect through Tor.) As we milk binaries from this URL, we typically see *Securitysuite* or *SmartAdsSolutions* binaries. We also obtain *Zbot* for a brief 11-hour period and *GoldInstall-list* for about three days. During August 2010 our *LoaderAdv* milker downloads malware from a total of 19 unique URLs (ignoring the domains). Three of these are single-client URLs only serving *Rustock*, while the remaining 16 drop malware matching 31 of our signatures.

GoldInstall. *GoldInstall* has two downloaders. The *GoldInstall-list* downloader contacts the PPI C&C server to obtain a list of URLs hosting the client executables. The received list varies based on the geographic location. *Goldinstall-dl* has a hard-coded list of URLs in the binary that serve executables independent of geographic location. Both the *GoldInstall-list* and *GoldInstall-dl* downloaders fetch the executables using HTTP, with each distinct URL representing a single family of malware. Often, the service hosts the same client executable in multiple locations, with the path components of the URL (such as `1.exe`) remaining constant. When the path is the same, typically so is the family of malware, though we also observed common URL paths used for multiple families (e.g., `bot.exe`). The download locations show no evidence of checking the geo-location of the downloader before serving malware. Thus, the *GoldInstall-dl* downloader does not download executables based on geographic location. Throughout the month, the program periodically distributed new URLs to the PPI executable, 41 total. These on average continued to return valid executables for 36 days after first provided by the C&C (maximum 162 days, minimum 14 hours).

Virut. The *Virut* downloader uses a custom IRC-based C&C protocol to receive a list of URLs hosting the client executables. We observe a total of six distinct URLs throughout August 2010, distributing 15 distinct executables matching signatures for three families. Four of the URLs use a domain with the same *whois* entries as the *Virut* C&C, and each URL can return a different executable for each request.

Zlob. The *Zlob* downloader uses a custom encrypted C&C protocol to request a list of URLs to locate client programs. The received list varies based on the geographic location. The service replicates the list of URLs so that every two received URLs correspond to one executable, at two locations apparently for redundancy.

[1] Detecting the presence of *Securitysuite-avm* versus *Securitysuite* was the only significant identification we obtained by using our "bare metal" setup in addition to our VM-based execution environment.

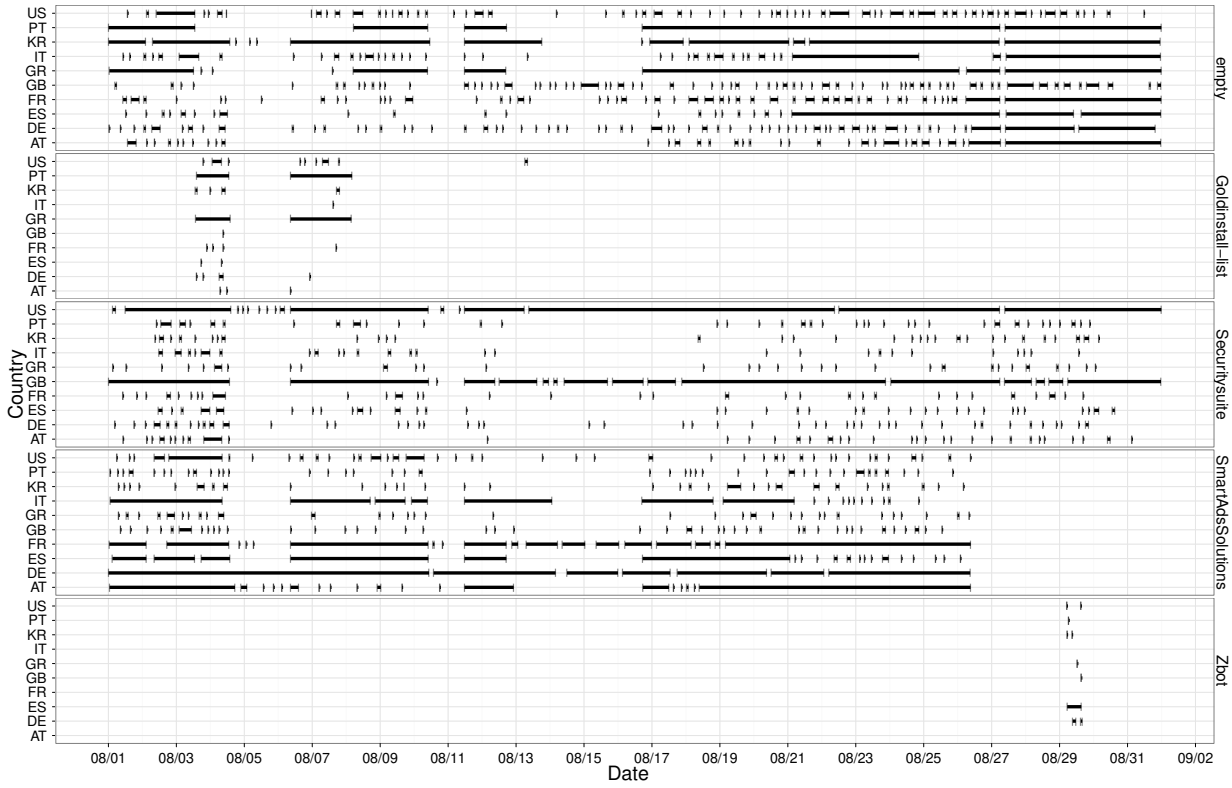

Figure 6: Availability of malware families over time, from a single *LoaderAdv* URL. The empty family shows when the URL provided a non-executable response.

4.4 Geographic Breakdown

To investigate the geographical preferences of the different malware families, we analyze the milk from the *LoaderAdv*, *GoldInstall*, and *Virut* services, since as explained in Section 3.2 for these three services the milker used 15 Tor circuits in parallel, each terminating in a different country. We selected 15 countries using price points advertised by PPI providers: AT, BR, DE, ES, FR, GB, GR, IT, JP, KR, NL, PL, PT, RU, and US.

For most malware families we observe clear geographical preferences. Figure 7 shows the frequencies with which we obtained a sample of the *Ertfor*, *Gleishug*, *Rustock*, *Securitysuite*, and *SmartAdsSolutions* families, each of which our milkers downloaded at least 100 times during August. We selected these groups to highlight characteristics we observe in geographical distribution; other families exhibit similar patterns.

Three trends in geographical distribution emerge. First, we commonly see families of malware preferentially targeting Europe and the US (e.g., *Ertfor*, *Securitysuite*, and *SmartAdsSolutions*). Second, some families exclusively target the US or another single country (e.g., *Gleishug*). Finally, we observe families with no geographical preferences (e.g., *Rustock*).

Several factors can influence a PPI client's choice of country. First, the class of activity in which the client's executable engages. A spam bot such as *Rustock* requires little more than a unique IP address to send spam, while fake AV such as *Securitysuite* often targets speakers of a specific language, and may need to support user payment methods specific to some areas. In addition, the install rate a client pays also varies depending on the targets' countries. We find the US and Great Britain generally at the high end ($100–180 per thousand), other European countries in the middle ($20–160), and the rest of the world at the bottom (< $10) [12, 13, 19].

4.5 Affiliate–PPI Interactions

Surprisingly, among the binaries that we milk we find a number of affiliate PPI downloaders. That is, download-

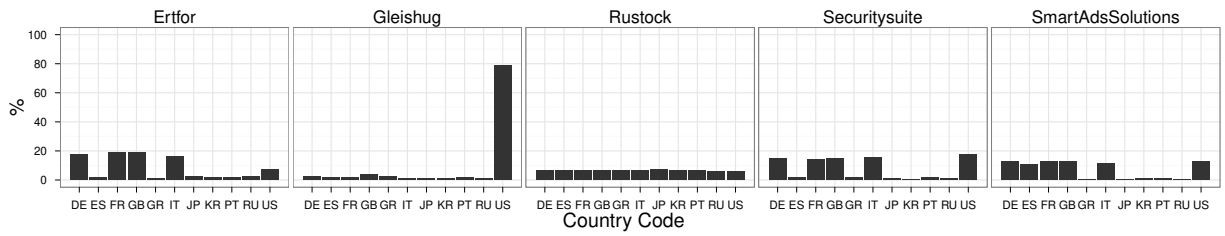

Figure 7: Prevalence of six malware families seen by our milkers from different country vantage points.

ers not infrequently *download other downloaders*. This indicates that some PPI affiliates have also signed up as *clients* of PPI services. To understand these affiliate–PPI interactions, we extracted the unique affiliate identifier embedded in each of the PPI downloaders found in our milk, which we can observe from its transmission (presumably for accounting purposes) during the C&C exchange.

Using these identifiers, we observe that affiliates from one PPI service themselves sometimes act as clients of other PPI services. This behavior manifests by our milker, impersonating affiliate *X* for PPI service *A*, fetching an executable for installation that corresponds to a downloader for affiliate *Y* of PPI service *B*.

We speculate that some of these multi-PPI-service affiliates represent arbitrageurs who try to take advantage of pricing differentials between the (higher) install rates paid to the affiliates of one service for some geographical regions versus the (lower) install rates charged to clients of another PPI service. For example, we observe that *LoaderAdv*'s affiliate `701` signed as a client of *GoldInstall*, using the latter to distribute `701`'s personalized *LoaderAdv* downloader for four days. Here, the price differential includes the US, Canada, and Europe, from which our *GoldInstall* milkers collected this executable.

Perhaps even more surprising, we find affiliates from one PPI service who are also *clients of the same PPI service*. For example, *LoaderAdv*'s affiliate `515` distributed their personalized *LoaderAdv* downloader over Europe and Brazil using the *LoaderAdv* service for a total of 20 hours. We see a similar behavior from affiliates `0625` and `go1` of the *GoldInstall* service, both clients and affiliates of *GoldInstall*. We conjecture that this happens when affiliates try to take advantage of the price differential between the (higher) install rates paid to the affiliates for some geographical regions over the (lower) install rates paid by the clients for installing on the same regions. Note that such price differential is possible because the PPI service oversells installs: *multiple* clients can pay the service for installs that cost the service only a *single* affiliate payout. We suspect the PPI service can detect this behavior would not credit both affiliates for the install.

In a yet more convoluted case, we observed a *GoldInstall* affiliate, `e4u`, signing up as a client for both *GoldInstall* and *LoaderAdv*. We speculate that `e4u` most likely stands for "*earning4u*", the brand for the *LoaderAdv* PPI service at that time. (Presumably this affiliate simply took advantage of price differentials within the *GoldInstall* service and with the *LoaderAdv* service, but possibly `e4u` in fact represents the *LoaderAdv* gang itself.)

4.6 The Download Tree

One important observation of our work regards how the nesting of downloaders-downloading-additional-downloaders can quickly grow strikingly complex. To capture such nesting we use a *download tree*. Nodes in the tree represent programs identified by hashes of their binary. At each branch in the tree, children represent programs installed by the parent. Figure 8 shows an example download tree. We term any node with children a downloader. Nodes with a single child may be specialized downloaders for the child family, while nodes with multiple children may reflect PPI downloaders that charge the children for the installs. Leaf programs may implement any of a number of recognizable malware behaviors, including sending spam, performing click-fraud, and stealing personal information.

Generating the download tree requires carefully identifying the dependencies between installed programs, e.g., which program downloads and executes other programs. To build the tree in Figure 8, the client malware programs need the freedom to download other executables from the Internet. For this experiment we used a different containment policy that sinks everything but HTTP and C&C. In addition, we rate-limited the outgoing HTTP and C&C traffic, and a human operator mon-

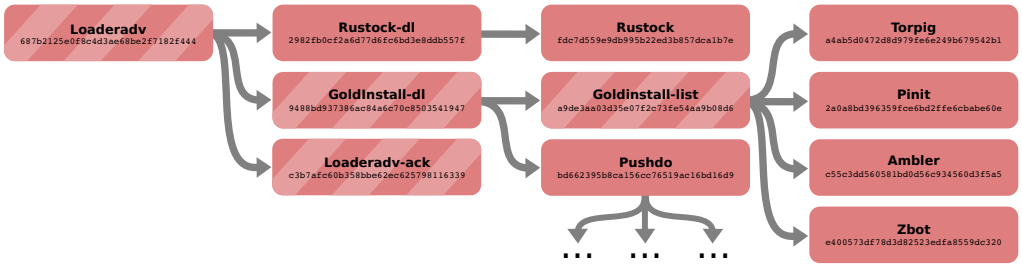

Figure 8: A download tree starting with a single *Loaderadv* downloader. Stripes indicate PPI service-related binaries.

itored the execution in real-time to stop the process if anything unexpected happened.

The download tree in Figure 8 comes from the live execution of (originally) a single *LoaderAdv* PPI downloader that we ran in our controlled environment. Strikingly, the entire execution required under 10 minutes—with several additional leaf nodes *omitted* for clarity! Thus, the example illustrates how quickly an exploited system can transform from unmolested operation to hosting a veritable ecosystem of malware.

5 Discussion

Our findings have a number of implications, as follows.

Malware classification. Our work shows that we should conceptually separate the exploitation mechanism compromising a system from the malware that the system subsequently hosts. For example, it may not make sense to characterize malware by its infection method beyond malware that self-propagates and malware that does not. Botmasters might simply *purchase* installation of their malware from PPI services which can use a variety of distribution methods.

The installation of malware from multiple clients on a single target host has important implications for behavior-based malware classification. For example, when writing a malware analysis report it is easy to confuse a downloader with malware that it happens to install during one particular execution. Such confusion can then result in misleading statistics characterizing the prevalence of malware families. Furthermore, malware analysis platforms that execute malware with Internet connectivity [1, 2, 30] should carefully track program downloads and their execution, to allow separation of each program's runtime behavior. Without a download tree, behavioral reports may reflect the aggregate behavior of multiple types of malware. These aggregate reports may result in incorrect classifications, and in the worst case the produced signature may fail to detect individually executing malware.

Regarding classification techniques, we note that our work does not aim to pursue advances in the field of behavioral malware signature generation, and instead employs straightforward techniques. We could fruitfully incorporate much of the published research in this space into our classification approach.

Defenses. As defenders, we need to understand and appreciate the threat posed by the "silent installs" industry. PPI services have direct implications for takedown efforts: even if defenders can completely clean up a botnet (as opposed to merely severing its C&C master servers), the botmaster could return to business-as-usual through modest payments to one or more PPI services. Given that multiple malware authors share use of the same PPI services, and that the number of PPI services seems to be significantly smaller than the number of malware families, PPI services are good targets for future takedown efforts. However, the commoditization of the malware industry could make it easy to recreate PPI services elsewhere after takedown, so the focus should be on identifying and apprehending the people that run such services.

Regarding detection techniques, we observe that the content-based features of our signatures perform better than the endpoint-based features. The former wins over the latter in our handling of the periodic replacement of stale URLs PPI services employ for hosting the malware executables, likely to bypass URL blacklists. We also observe that many downloaders employ a simple download-and-execute strategy, which in turn suggests that defenders might realize significant protections by employing taint-based approaches that identify the execution of downloaded data.

Evasion. Infiltrating the PPI C&C protocols required significant reverse-engineering effort on our part. As miscreants become aware of this possibility and more parties launch infiltration attempts, adversarial evolution will surely complicate this process. In particular, we expect PPI services to harden their C&C protocols with more robust use of cryptographic techniques and incorporation of anti-virtualization and triggering mechanisms to increasingly hamper dynamic analysis. On the other hand, the fact that a relatively modest infiltration effort sufficed to gain insight into many of today's top malware families is encouraging. Analysts should remain on the lookout for opportunities to infiltrate core components of the modern malware ecosystem, which may offer broad insights into the malware landscape.

6 Conclusion

We have presented the results of the first systematic study of the pay-per-install (PPI) ecosystem, conducted by infiltrating the malware distribution mechanism of PPI services. The ability to "milk" malware binaries directly from the source provides an unprecedented intelligence capability to defenders. We leveraged this approach to measure technical aspects of the market surrounding malware installation.

Starting with a network-behavioral classification of a one-month corpus of 313,791 binaries, we identified 12 of the 20 most prevalent families of malware. We illustrated how infection with several clickfraud and fake-AV families specifically target the United States and Europe, while other malware classes, such as spam bots, are distributed worldwide. Our examination of repacking rates of PPI-distributed malware showed that on average binaries are repacked every 11 days, with one family of malware repacking up to twice a day. Finally, we illuminated the relationships among actors in the PPI ecosystem, including the identification of *LoaderAdv* and *GoldInstall* affiliates that apparently engage in pricing arbitrage by becoming clients to other PPI providers.

7 Acknowledgements

We would like to thank Atif Mushtaq for his help understanding FireEye's top 20 malware classification. We also thank the anonymous reviewers for their insightful comments.

This work was partially done while Juan Caballero was a visiting student researcher at the University of California, Berkeley, and a graduate student at Carnegie Mellon University. This work was supported in part by the National Science Foundation under grants NSF-0433702, CNS-0905631, and CNS-0831535, and by the Office of Naval Research under MURI Grant No. N000140911081. Juan Caballero was also partially supported by Grants FP7-ICT No. 256980 and FP7-PEOPLE-COFUND No. 229599. Opinions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Anubis: Analyzing Unknown Binaries. `http://anubis.iseclab.org/`. Accessed on June 2011.

[2] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A Tool for Analyzing Malware. In *European Institute for Computer Antivirus Research Annual Conference*, April 2006.

[3] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *International Conference on Privacy, Security and Trust*, August 2010.

[4] AS-Troyak Exposes a Large Cybercrime Infrastructure , March 2010. `http://blogs.rsa.com/rsafarl/as-troyak-exposes-a-large-cybercrime-infrastructure/`.

[5] J. Caballero, N. M. Johnson, S. McCamant, and D. Song. Binary code extraction and interface identification for security applications. In *Proceedings of the Network and Distributed System Security Symposium*, February 2010.

[6] C. Y. Cho, J. Caballero, C. Grier, V. Paxson, and D. Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats*, April 2010.

[7] N. Doshi, A. Athalye, and E. Chien. Pay-Per-Install: The New Malware Distribution Network, April 2010. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/pay_per_install.pdf`.

[8] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection. In *Proceedings of the USENIX Security Symposium*, April 2006.

[9] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/. Accessed on June 2011.

[10] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *Proceedings of ACM Conference on Computer and Communications Security*, October 2007.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol and Structure Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium*, July 2008.

[12] InstallsDealer. http://installsdealer.com/. Accessed on June 2011.

[13] InstallsForYou. http://installsforyou.biz/. Accessed on June 2011.

[14] Internet archive. http://www.archive.org/. Accessed on June 2011.

[15] B. Koehl and J. Mieres. SpyEye Bot (Part two) Conversations with the Creator of Crimeware, February 2010. http://www.malwareint.com/docs/spyeye-analysis-ii-en.pdf.

[16] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector GAdget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[17] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G.M. Voelker, V. Paxson, and S. Savage. Spamcraft: An inside look at spam campaign orchestration. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[18] Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. GQ: Practical Containment for Measuring Modern Malware Systems. Technical Report TR-11-002, International Computer Science Institute, May 2011.

[19] LoadsSell. http://loadssell.net/. Accessed on June 2011.

[20] MaxMind. Resources for Developers. http://www.maxmind.com/app/api. Accessed on June 2011.

[21] J. Mieres. Russian prices of crimeware, March 2009. http://evilfingers.blogspot.com/2009/03/russian-prices-of-crimware.html.

[22] A. Mushtaq. World's Top Malware, July 2010. http://blog.fireeye.com/research/2010/07/worlds_top_modern_malware.html.

[23] J. Oberheide, M. Bailey, and F. Jahanian. PolyPack: An Automated Online Packing Service for Optimal Antivirus Evasion. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, August 2009.

[24] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463, December 1999.

[25] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation*, April 2010.

[26] D. Plonka and P. Barford. Context-Aware Clustering of DNS Query Traffic. In *Proceedings of the 8th ACM SIGCOMM conference on Internet Measurement*, October 2008.

[27] Best Pay-Per-Install Affiliate Program Reviews. http://pay-per-install.com. Accessed on June 2011.

[28] Pay-Per-Install Programs. https://www.pay-per-install.org/. Accessed on June 2011.

[29] K. Stevens. The Underground Economy of the Pay-Per-Install PPI Business, February 2010. Black Hat DC, Arlington, VA.

[30] ThreatExpert – Automated Threat Analysis. http://www.threatexpert.com/. Accessed on June 2011.

[31] The Tor Project. http://www.torproject.org/. Accessed on June 2011.

[32] Virustotal – Free Online Virus, Malware and URL Scanner. http://www.virustotal.com/. Accessed on June 2011.

[33] ZeuS Tracker. https://zeustracker.abuse.ch/. Accessed on June 2011.

A Examples of Signatures

This appendix provides a concrete view of several of the malware signatures that appear in Table 3. We include four popular-but-untagged clusters, and two versions of *Palevo* for reference.

Each signature consists of three parts: URL, DOMAIN, and PAYLOAD statements followed by associated contents. The URL can contain regular expressions; we only use it for HTTP-based protocols. DOMAINS can list IP addresses or domains (with or without subdomains). PAYLOAD statements specify the parameters for `where`, `type`, `contents`, and `len`. `where` specifies the location to match, with "`begin`" meaning at the beginning of the payload. We use `type` to inform the engine whether to interpret the contents as a string or as an array of bytes. Finally, `len` restricts the length of the checked packet: a signature that specifies a `len` will only match if the packet has exactly the given length in bytes.

CLUSTER: A

```
URLS
/svc.php\?ver=

DOMAINS
sy.perfectexe.com
sy2.perfectexe.com
sy3.perfectexe.com
```

CLUSTER: B

```
URLS
/get.cgi\?.+
/data.cgi

DOMAINS
f19dd4abb8b8bdf2.cn
2bff2694930d2e21.cn
697fe322c995da1a.net
89e3aaecc2ba1734.net
ade34ea82c4f7f2f.net
```

CLUSTER: C

```
DOMAINS
ds.perfectexe.com

URLS
/active.asp\?[0-9]{2}
```

CLUSTER: D (URLs truncated for space)

```
DOMAINS
x.liruna.com

URLS
/x.ashx\?
ashx\?a=get&v=
ashx\?a=[^&]+v=[^&]+&fid=[^&]+&id=...
```

Palevo

```
DOMAINS
193.104.186.88
76.76.99.186
f5v9w.com
e7j0h7.cn
mp1r3n.ru

URLS
/hygtrve.exe
/htrgef.exe
/htgref.exe
/hybtvr.exe

PAYLOAD
where    : begin,
type     : bytes,
contents : [[0x61]],
len : 7
```

Palevo2

```
DOMAINS
ff.fjpark.com
fifa2012terra.com
converter50.com

URLS
/rip.exe, /usa.exe, /575.exe,
/adv.exe, /adv2.exe, /rip2.exe,
/prr.exe, /4757exe.exe

PAYLOAD
where: begin,
type : bytes,
contents : [[0x18]],
len : 21,
```