Large-Scale Analysis of Malware Downloaders

Christian Rossow †‡[⋆], Christian Dietrich †¶, and Herbert Bos‡

†University of Applied Sciences Gelsenkirchen, Institute for Internet Security, Germany
‡VU University Amsterdam, The Network Institute, The Netherlands
¶Department of Computer Science, Friedrich-Alexander University, Erlangen, Germany
{rossow,dietrich}@internet-sicherheit.de

Abstract. Downloaders are malicious programs with the goal to subversively download and install malware (*eggs*) on a victim's machine. In this paper, we analyze and characterize 23 Windows-based malware downloaders. We first show a high diversity in downloaders' communication architectures (e.g., P2P), carrier protocols and encryption schemes. Using dynamic malware analysis traces from over two years, we observe that 11 of these downloaders actively operated for at least one year, and identify 18 downloaders to be still active. We then describe how attackers choose resilient server infrastructures. For example, we reveal that 20% of the C&C servers remain operable on long term. Moreover, we observe steady migrations between different domains and TLD registrars, and notice attackers to deploy critical infrastructures redundantly across providers. After revealing the complexity of possible counter-measures against downloaders, we present two generic techniques enabling defenders to actively acquire malware samples. To do so, we leverage the publicly accessible downloader infrastructures by replaying download dialogs or observing a downloader's process activities from within the Windows kernel. With these two techniques, we successfully milk and analyze a diverse set of eggs from downloaders with both plain and encrypted communication channels.

Keywords: Malware, Downloader, Dropper, Dynamic Analysis

1 Introduction

A crucial part in a malware's lifecycle is to spread, e.g., via spam, drive-by downloads or exploiting vulnerabilities. Whereas malware such as worms spreads on its own, attackers have begun to separate the task of infecting victim systems and the exploitation or "monetization" of the infected systems. Recent investigations to this business, known as "Pay-per-Install" (PPI), have shown the vast potential of this kind of malware distribution model. Caballero et al. [5] analyzed PPI networks by actively infiltrating and participating a handful of PPI programs. It was shown that PPI networks are responsible for installing a diverse set of malware on infected systems.

[⋆] We thank all malware sample providers and the VirusTotal team for their input. We thank Dennis Fricke and Andrea Lanzi for their support in developing the Windows kernel driver, and Arne Welzel for his reverse engineering efforts. This work was supported by the Federal Ministry of Education and Research of Germany (Grant 01BY1110, MoBE). The research leading to these results has also received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 257007.

Technically, the PPI scheme is only a subset of the malware type that we term a *downloader*. A downloader is a malicious program with the purpose to subversively download and install malware on a victim's machine. The specifics of PPI networks allow attackers to get paid on a per-system and per-affiliate basis, but the effect of PPI or, more generally, downloaders is comparable: Once a downloader is executed, and no matter if related to a PPI network or not, the running system will typically be compromised with multiple different malware families. Thus, downloaders represent a simple yet widely-used way to spread new malware, typically as part of a service model within the underground community.

In this paper, we outline and analyze the landscape of what we think represents a snapshot of prevalent and current downloaders. We identified 23 downloaders, of which many – to the best of our knowledge – have not yet been documented. We characterize these downloaders concerning their communication model. For example, we discuss the communication architectures of downloaders (e.g., P2P), and outline the techniques used to encrypt or even camouflage the malicious activities. We then use dynamic analysis traces to provide a long-term monitoring analysis on these 23 downloaders, identifying 18 downloaders to be still active as of writing this paper. In addition, we show that eleven downloader families are actively distributing malware for more than a year.

Motivated by this observation, we investigate how attackers ensure the resilience of downloader infrastructures. Contrary to our expectation that IP address blacklists would force attackers to change their infrastructure frequently, we show that 219 C&C servers (20%) were actively operated for more than four weeks. For the remaining servers, we analyze how attackers use DNS and IP address fluxing to operate their downloaders, suggesting that isolating downloader infrastructures is much harder than it seems.

As a third part of our analysis, we propose two automated methods to extract the downloaded malware (*eggs*) in a generic and scalable fashion. We hope that these techniques will support future efforts in analyzing downloaders without the manual effort of reverse engineering particular downloader families. We evaluate these two techniques both on downloaders with plaintext and encrypted communication, acquiring a diverse set of malware in the wild.

To summarize our contributions:

- We identify and characterize 23 malware downloaders, describing previously undocumented specimen and their communication models.
- We perform a long-term analysis of these downloaders, revealing that 11 downloaders have been operating for more than a year, and approach to understand the reasons for the infrastructural resilience.
- We propose two automated techniques to actively acquire malware from downloaders, without requiring reverse engineering downloaders.

2 Preliminaries

Malware defense mechanisms, especially anti-virus, have forced attackers to develop increasingly complex malware. This complexity has motivated attackers to specialize and separate duties. For example, services to stealthily install malware on computers may be provided by one group, while other fraudsters specialize in sending spam, and a third group could focus on keylogging. In this work, we focus on the service of installing new malware on systems via *downloaders*. Downloaders are malicious programs that are instrumented to load additional malware via the Internet, which is in turn installed and executed on the victim's system.

2.1 Downloader Architectures

Figure 1 illustrates the architecture of a downloader. Once executed, a downloader contacts its command-and-control (C&C) server(s) via *C&C channels*. After receiving download instructions, it then establishes at least one *download channel* to load malware (*eggs*) via the network.

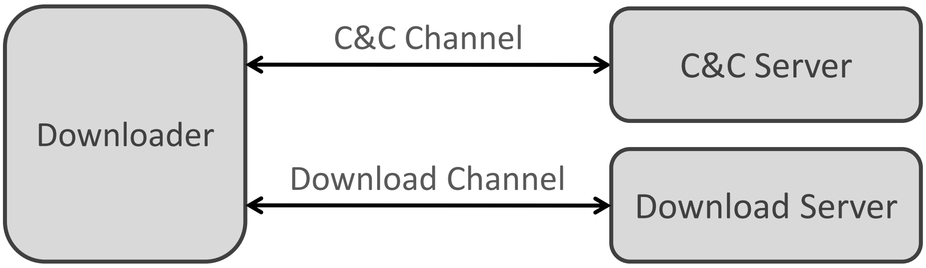


Fig. 1. Simplified architecture of a downloader: Seperation between C&C and download channel.

C&C Channels A downloader's C&C channel is used to get lists of URLs (or similar address information) where eggs can be downloaded from. Next to download instructions, the C&C channel can be used to report back to the C&C server if the download succeeded. In addition, as shown by Caballero et al. [5], C&C channels may exchange affiliate IDs in the economical model of pay-per-install downloaders. Moreover, downloaders send details about the infected host using the C&C channel, such as the OS version, username or device IDs. One characteristic of C&C channels is the *carrier protocol* used to transfer commands. Typical examples for carrier protocols are IRC, HTTP (e.g., if C&C messages are in the HTTP body) or plain TCP/UDP. The information exchanged on C&C channels is critical and highly subjective to counter-measures such as signature-based IDSs, and thus more advanced downloaders encrypt their C&C channel. During our investigations, we also observed downloaders that have download URLs hardcoded in their binaries. We excluded such downloaders from our analysis because of their simplicity and transitory nature.

Download Channels We found the C&C and download channels to be typically well-separated. A download channel shares similar characteristics than the C&C channel, i.e., it has a specific carrier protocol and potential encryption schemes. While we did not see examples of steganographic C&C channels, as we will show, some downloaders tend to camouflage their malicious downloads in normal web traffic. Another typically distinguishing characteristic between C&C and download channels is the number of bytes transferred. C&C commands tend to be small, while eggs – no matter if encrypted or not – have significantly larger file sizes.

2.2 Related Work

First steps to analyze specific downloaders were made by Caballero et al. by analyzing four pay-per-install (PPI) programs [5]. PPI downloaders, a subset of downloaders in general, are based on an economical model cashing out attackers for installing malware on a freshly infected system. Caballero et al. implemented so called *milkers* to download eggs from these four PPI networks, and systematically analyze the ecosystem behind these networks. They show in-depth how egg families relate to download programs, and identified that kinds of malware (e.g., DDoS) were distributed in download campaigns.

Our work was inspired by Caballero et al., and we seek for a broader characterization of downloaders. In fact, we found ourselves at a position not knowing the magnitude and different types of downloaders currently active in the wild. We identify that the number and kinds of downloaders is significantly higher than expected. To the best of our knowledge, we are the first to approach a characterization of downloaders. We then also seek to answer the fundamental but yet unanswered question of how attackers build up infrastructures that are sufficiently resilient for long-term operations of downloaders. We expand a malware acquisition technique as proposed in Botlab [7] with replaying network dialogs as proposed by Newsome et al. [11]. While Botlab fetches malware from URLs found in spamfeeds, our techniques repeatedly acquires malware from downloader URLs. Existing systems like Threatexpert [15] or Anubis [4] can already analyze malware in general, but we are the first to analyze the behavior and infrastructures on downloaders over multiple executions and on long-term.

3 Analysis of the Downloader Landscape

In this section, we characterize and describe the 23 downloaders identified as part of this work, which we will then further analyze later in the paper.

3.1 Dataset Description

Our analyses are based on malware reports from Sandnet [12]. Sandnet executes and dynamically analyzes malware using Windows XP SP3 32bit virtual machines connected to the Internet via NAT. During malware execution, we deploy containment policies that redirect harmful traffic (e.g., spam, infections) to local honeypots. We further limit the number of concurrent connections and the network bandwidth to mitigate DoS activities. An in-path honeywall NIDS watched for security breaches during our experiments. Other protocols (e.g., IRC, DNS or HTTP) were allowed to enable C&C communication. We consider the biases affecting the following experiments due to containment to be negligible. Specifically, given our long measurement period of one hour per malware sample, we did not observe any incomplete download behaviors in our trace. Assuming that downloaders silently operate without the user's consent, we did not deploy user interaction during our experiments. We plan to analyze malware on 64-bit architectures or more recent Windows versions in the future.

Our dataset consists of 243,000 MD5 unique malware samples analyzed in Sandnet at least once between Feb 2010 and Feb 2012. We gratefully received these samples from a variety of sources, including samples submitted to public dynamic analysis environments, feeds by security companies, our own honeypot infrastructures and spamtraps. While we cannot prove that this dataset covers all relevant malware families, it shows a diversity of 38,000 unique malware labels (according to Kaspersky). We extracted the malware family names from these labels and found over 1800 malware families in our dataset.

From this diverse set of samples, we scheduled a random selection on a daily basis, without giving any emphasis to particular malware families. To trigger the malware behavior, we then executed these samples for at least one hour. Obviously, only a minor fraction of these malware samples are in fact downloaders. To build up a dataset covering the most relevant downloaders we did a threefold approach. First, we consulted literature research and asked AV vendors for their expert knowledge on recent and prevalent downloaders. Second, in our dataset covering millions of malware samples, we searched for prevalent AV labels suggesting the malware is a downloader. Third, we manually inspected a random subset of the Sandnet analysis reports for downloader behavior. We manually filtered legitimate programs in our dataset of potential downloaders, such as e.g. Windows Update, Google Updater or programs to update system drivers.

For each identified downloader, we systematically searched for related analysis reports in Sandnet. Typically, we used payload or behavioral signatures to classify and recognize a particular downloader. In rare cases, where a downloader family did not expose any signature, we carefully assembled sets of domains and IP addresses to recognize downloader traffic. Using these techniques, we are able to detect all previous and upcoming executions of a particular downloader family.

3.2 Downloaders Overview

The resulting dataset provides an empirical overview of existing downloaders. Table 1 lists the downloaders that we monitor as part of this work. While this is not necessarily complete, it shows a large diversity in terms of different downloader characteristics. The attributes in Table 1 form two groups: The left-hand attributes characterize the C&C channel, while the right-hand columns characterize the download channel. We labeled three droppers with generic names (dldr-#1 to dldr-#3), as anti-virus vendors either assigned too generic or contradictory labels for those.

Carrier Protocols A first distinction between the downloaders can be made in terms of the *carrier protocol*, that is, the protocol used to communicate with C&C or Download servers. To understand and also classify downloaders, we had to reassemble and parse numerous carrier protocols (UDP, TCP, DNS, HTTP, IRC, TLS). For obfuscated protocols, we define the carrier protocol to be the underlying protocol of the C&C protocol, e.g., “HTTP” for GoldInstall or Renos/Artro and “TCP” for the encrypted variant of Virut C&C. Interestingly, Table 1 shows that C&C channels are not necessarily designed in the same way as download channels. For example, five downloaders

	C&C Channel				*Download Channel*		
Family	*arch*	*Pl?*	*Protocol*	*DNS?*	*Pl?*	*Protocol*	*DNS?*
Renos/Artro	cent	✗	HTTP	✓	✗	HTTP	✓
Sality	cent	✗	HTTP	✓	✗	HTTP	✗
dldr-#1	cent	✗	HTTP	✓	✗	HTTP	✓
Cycbot/Gbot	cent	✗	HTTP	✓	✗	HTTP-inl	✓
Karagany	cent	✗	HTTP	✓	✗	HTTP-inl	✓
Gamarue	cent	✗	HTTP	✓	✓	HTTP-inl	✓
Dofoil	cent	✗	HTTP	✓	✓	HTTP	✓
Emit	cent	✗	HTTP	✓	✓	HTTP	✓
GoldInstall	cent	✗	HTTP	✓	✓	HTTP	✓
Rodecap	cent	✗	HTTP	✓	✓	HTTP	✓
Virut (crypt C&C)	cent	✗	TCP	✓	✓	HTTP	✓
TDSS	cent	✗	TLS	✓	✗	HTTP	✓
Winwebsec	cent	✓	HTTP	✗	✓	HTTP	✗
Dabvegi	cent	✓	HTTP	✓	✗	HTTP	✓
Buzus	cent	✓	HTTP	✓	✗	HTTP	✓
dldr-#3	cent	✓	HTTP	✓	✓	HTTP	✓
Zwangi	cent	✓	HTTP	✓	✓	HTTP	✓
Harnig/LoaderAdv	cent	✓	HTTP	✓	✓	HTTP-inl	✓
dldr-#2	cent	✓	HTTP	✓	✓	HTTP-inl	✓
Virut (plain C&C)	cent	✓	IRC	✓	✓	HTTP	✓
Vobfus/Changeup	cent	✓	TCP	✓	✓	HTTP	✓
Sality P2P	P2P	✗	UDP	✗	✗	TCP	✗
Zeus P2P	P2P	✗	UDP	✗	✗	TCP	✗

Table 1. Overview of downloaders under our analysis. Columns 2–5 characterize the C&C channel, columns 6–9 characterize the download channel. “Pl?” shows if the communication channel was in plain text, and “DNS?” shows if names of the communication endpoints were resolved via DNS prior to contacting them.

use obfuscated or encrypted C&C channels, but at the same time have plaintext HTTP download channels. Similarly, another five downloaders do not separate between C&C and download channels, abbreviated by “inl” to show that malware is served inline with the C&C protocol.

Communication Architectures As Table 1 suggests, almost all downloaders deploy a centralized C&C architecture. Two exceptions are Sality P2P and Zeus P2P. Sality uses a hybrid C&C architecture, i.e., some samples use a centralized HTTP-based C&C channel while others receive their commands via a peer-to-peer network. Zeus P2P is a pure P2P based bot with download functionality. Such distributed networks are attractive to attackers, as the C&C infrastructure cannot be disrupted by taking offline single C&C servers. Both Sality P2P and Zeus P2P[1] initialize their C&C channel by trying to contact hundreds of P2P bootstrapping nodes.

DNS Table 1 reveals that most downloaders make use of DNS to resolve the names of their C&C and/or download servers. However, downloader families such as Winwebsec

[1] Note that while Zeus may not be conceived as downloader, there are references supporting our observation that recent Zeus variants drop other malware

and the P2P-driven downloaders avoid DNS resolution for both C&C and download servers. We speculate that such downloaders either have no technical need for DNS, e.g. the P2P architectures, or want to foil malware domain blacklists. From the attacker's point of view, another disadvantage of using DNS is that taking down domains exposes an additional point of failure in the communication chain. However, on the other hand, DNS would allow to quickly redirect to different IP addresses of download servers. This dilemma basically boils down to: Who is more resilient, the hoster (IP) or the DNS provider (domain)? We try to shed light onto different resilience strategies in Section 4. The fact that most downloaders use DNS resolution shows that developing mitigation techniques based on DNS is promising. Howevr, although we see downloaders using DNS, they may also have a backup communication channel, e.g., using hardcoded IP addresses [10].

Intuitively, one may think that downloaders use DNS to quickly react on server takedowns. Fast flux [9], domain flux and the business of bullet-proof DNS hosting would support this intuition. As we figured, however, some downloaders do not (need to) change DNS records of particular C&C domains. Consequently, while the usage of domains evolved over time, the IP addresses resolved by these domains were relatively static. We will further analyze these observations in Section 4.1.

Communication Encryption Defense mechanisms such as network-based intrusion detection systems or anti-virus scanners scan for URLs and file contents downloaded from the Internet. As a consequence, downloaders deploy a wide set of schemes to obfuscate or encrypt their communication channels. A distinction can be made between deploying well-known or custom encryption techniques. For example, the TDSS downloader relies on TLS within its C&C channel, thus preventing from eavesdropping on C&C communication [13]. Similarly, we observed that the Renos/Artro family encrypts using RC4 with a key hardcoded in the samples.

In contrast, other downloaders use custom encryption/obfuscation algorithms. To give insights, we reverse engineered specific downloader families. For example, Emit deploys an XOR shifting technique to obfuscate traffic. Similarly, Virut picks a random session key and the C&C servers derive these session keys by performing a known-plaintext attack on the ciphertext of the first message sent from the bot to the server. The session key itself is thus never transmitted. Independent from the cryptographic strength of a particular algorithm, understanding and possibly decrypting the ciphertexts often requires tremendous reverse engineering efforts.

Steganography Attackers further disguise the egg downloads with steganography. While encryption prevents eavesdroppers to read exchanged data, steganography tries to hide the existence of egg downloads. We have spotted camouflage techniques used by downloaders that could be interpreted as first steps towards steganography. For example, Renos/Artro hides its eggs in valid GIF files. Although these files look like regular legitimate pictures, eggs are carried as part of the files. Using custom routines, the downloader transforms these files to correct PE binaries.

Downloaders Using Public Services Most downloaders rely on their own infrastructure for hosting malicious software. However, we also observed that particular downloaders make use of publicly accessible services. For example, dldr-#1 retrieves its malicious files from a large public file clouding provider. From a defender's perspective, it is much harder to block access to legitimate services, as a distinction between legitimate or malicious downloads from such sources raises big challenges.

Tracking Mechanisms Among the plaintext downloaders, we could observe downloaders that are client-aware. That is, attackers derive pseudo-unique IDs per system, such as e.g. its MAC address, the gateway's public IP address or the Windows serial. The C&C servers can then keep track of which clients contacted them, and serve binaries accordingly. Similarly to e.g. Torpic [14], the downloader victims are presumable identified to track the number of infections, either to keep an overview or to use this data for payment (e.g., PPI). Another reason would be to observe and defend against potential abuses of the downloader infrastructures (see Section 5). To work around this in our setup, we are modifying fixed strings such as the MAC address for every malware execution in Sandnet since ever.

3.3 Downloader Lifetime

With our understanding that downloaders are a fundamental part of the malware lifecycle, we now analyze the lifetimes of the downloaders. For this lifetime analysis, we are not interested in a particular downloader *binary* (identified by the MD5 hash sum). Instead, we analyze when a particular downloader *family* appears in our dataset, and how long its C&C or download activities continue.

As a first step, we used the mechanisms described in Section 3.1 to identify downloaders of a particular family in our dataset. We specifically designed our signatures to match evolutions of particular downloaders. For example, GoldInstall, a PPI program with diverse affiliation programs [5], was covered by a single signature. We then had to filter C&C flows that reached the C&C server, but the C&C server responded with non-C&C data (e.g., HTTP 404 responses). We enhanced our signatures with heuristics verifying that an endpoint shows active C&C communication, filtering out a significant amount of sinkholed communication.

Figure 2 shows the resulting activity plot. To increase readability, we connected two markers if the gap between these two downloader occurrences in our dataset was less than four weeks. Due to a maintenance period in Sandnet, the graph lacks activity measures of droppers between 03/02/2011 and 08/04/2011. Overall, however, the graph shows that at least 11 of the 23 downloaders (48%) actively operated for more than a year. In addition, 18 downloaders (78%) are still active as of writing this paper. Given that some downloaders are more present in our sample feeds than others, and given that our measurement period started in Feb 2010, the resulting data represents lower bounds of the actual dropper lifetimes. We even noticed that some downloader families were discussed by the community prior to our measurement period, indicating that the lifetimes of some downloaders is significantly longer than two years. We therefore speculate that in fact even more downloaders were successfully operated in long-term. This opposes a long-lasting threat to our community, as apparently downloaders are largely and continuously used to infect PCs.

A few downloaders, such as Dofoil or Gamarue appeared first in our dataset in 2011, underlining active developments in the malware scene. The reasons why other downloaders ceased during the measurement period are twofold. First, in case of GoldInstall,

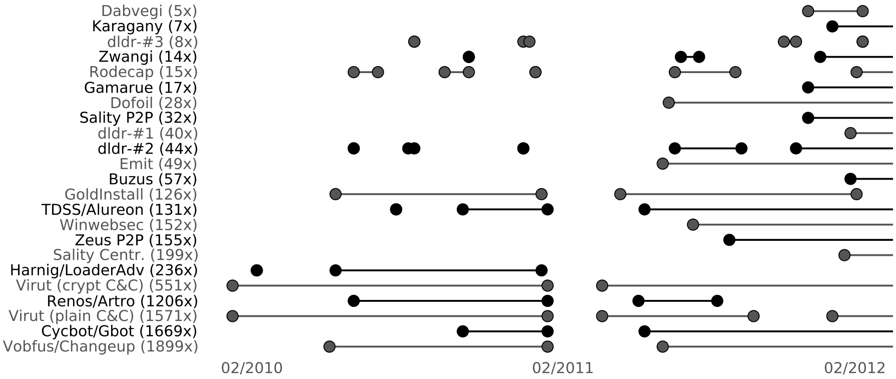


Fig. 2. Lifetime of downloaders, as observed in Sandnet, from Feb 2010 until Feb 2012. The numbers in brackets represent the number of active executions of this downloader in Sandnet.

C&C servers were not responsive for weeks, potentially indicating a downloader was abandoned or undergoes a major evolution. Second, as of August 2011, all specimen of Renos/Artro in our dataset were sinkholed by Shadowserver or Spamhaus.

4 Downloader Infrastructures

Seeing the significant lifetimes of downloaders, and knowing that defenders try to mitigate the threats of malware in general, we asked ourselves: How, technically, do attackers ensure such a high and long-term availability of their downloader infrastructures? In this section, we will therefore investigate the critical infrastructures used by attackers to operate their downloaders, i.e., both C&C and download servers.

4.1 C&C Infrastructure

C&C servers are vital to instrument the downloaders with new download instructions, and thus represent a sensitive part in the architecture of downloaders. From a downloader's perspective, two infrastructural services are crucial. First, most downloaders depend on DNS resolution prior to contacting their C&C server. Second, C&C servers obviously need to be reachable and service correctly. From a defender's perspective, both hosts (IP addresses) and domains represent vantage points to detect and/or disrupt downloaders.

We use the data obtained in Section 3.3 for further analyzing the C&C infrastructure. In particular, we aggregate the number of domains and IP addresses used by a particular downloader as observed in Sandnet. While this does not necessarily give a complete view on the IP addresses and domains used by a downloader, the numbers can serve as lower bounds. Table 2 shows that the resilience strategies differ between the downloaders. In the second major column, we summarize statistics on the specific C&C server IP addresses of a downloader, plus its Autonomous System (AS). In the third major column, Table 2 lists the number of C&C domains per dropper. We highlighted

Downloader	*IPs*		*ASes*		*Domains*		*TLDs*		*Timespan*
Family	*#*	*# LL*	*#*	*# LL*	*#*	*#LL*	*#*	*#LL*	M/Y - M/Y
Buzus	2	1	1	1	3	2	2	1	01/12 - 02/12
Cycbot/Gbot	145	48	56	36	2347	57	6	6	10/10 - 02/12
Dabvegi	5	4	4	3	5	4	3	3	11/11 - 01/12
dldr-#1	69	19	4	2	5	2	3	2	01/12 - 02/12
dldr-#2	41	11	21	5	45	12	7	4	06/10 - 02/12
dldr-#3	10	1	2	1	10	2	4	2	08/10 - 01/12
Dofoil	12	2	7	2	16	0	3	0	06/11 - 02/12
Emit	7	2	2	1	9	4	1	1	06/11 - 02/12
Gamarue	80	3	57	3	12	1	4	1	11/11 - 02/12
GoldInstall	12	5	7	3	13	8	3	2	05/10 - 01/12
Harnig/LoaderAdv	24	11	6	1	42	32	1	1	03/10 - 01/11
Karagany	2	0	1	0	7	0	2	0	12/11 - 02/12
Renos/Artro	27	5	12	3	75	0	3	0	06/10 - 02/12
Rodecap	8	4	2	2	5	4	3	3	06/10 - 02/12
Sality Centr.	239	62	125	47	243	59	31	18	06/11 - 02/12
Sality P2P	9849	1457	900	424	0	0	0	0	11/11 - 02/12
TDSS/Alureon	28	8	21	8	28	3	1	1	08/10 - 02/12
Virut (crypt C&C)	20	6	11	6	44	10	3	2	02/10 - 02/12
Virut (plain C&C)	14	4	9	4	3	3	1	1	02/10 - 02/12
Vobfus/Changeup	19	8	14	7	17	13	3	3	05/10 - 02/12
Winwebsec	5	2	4	2	0	0	0	0	10/10 - 02/12
Zeus P2P	2140	31	446	21	0	0	0	0	08/11 - 02/12
Zwangi	97	7	4	1	10	1	1	1	10/10 - 02/12

Table 2. Statistics on the C&C server distribution infrastructure per downloader family. LL=Long Lasting, i.e., IP addresses/domains had an uptime of more than 4 weeks. In each such case, we increase the corresponding AS/TLD counter by one also.

domains or IP addresses that we have seen in active use for at least four consecutive weeks in Table 2 in the columns annotated with “LL” (long lasting).

Table 2 reveals that most downloaders use multiple C&C server hosts, and tend to distribute their servers across network boundaries. For example, Virut has been in operation during our entire analysis period with about 20 IP addresses in eleven ASes. We speculate that spreading server locations among multiple ASes is a strategic decision by the attackers. The more responsible parties and different national regulations are in place, the higher the complexity for defenders to take actions against specific downloaders. In that sense, Cycbot/Gbot stands out with 146 servers, hosted in more than 50 different networks. Observing such a large diversity may indicate that Cycbot/Gbot is in fact a malware toolkit with downloader functionality, which results in many smaller infrastructures independent from each other. We verified that the Cycbot/Gbot instances in our dataset used different IP addresses at approximately the same time. Another interesting case is dldr-#1, which appears to operate many C&C servers on its own. But instead it uses a large public file sharing company and this hoster’s load balancing techniques, hiding eggs in seemingly benign Bitmap image files. As we are interested in all C&C activities of a downloader, we manually inspected all cases where possibly benign

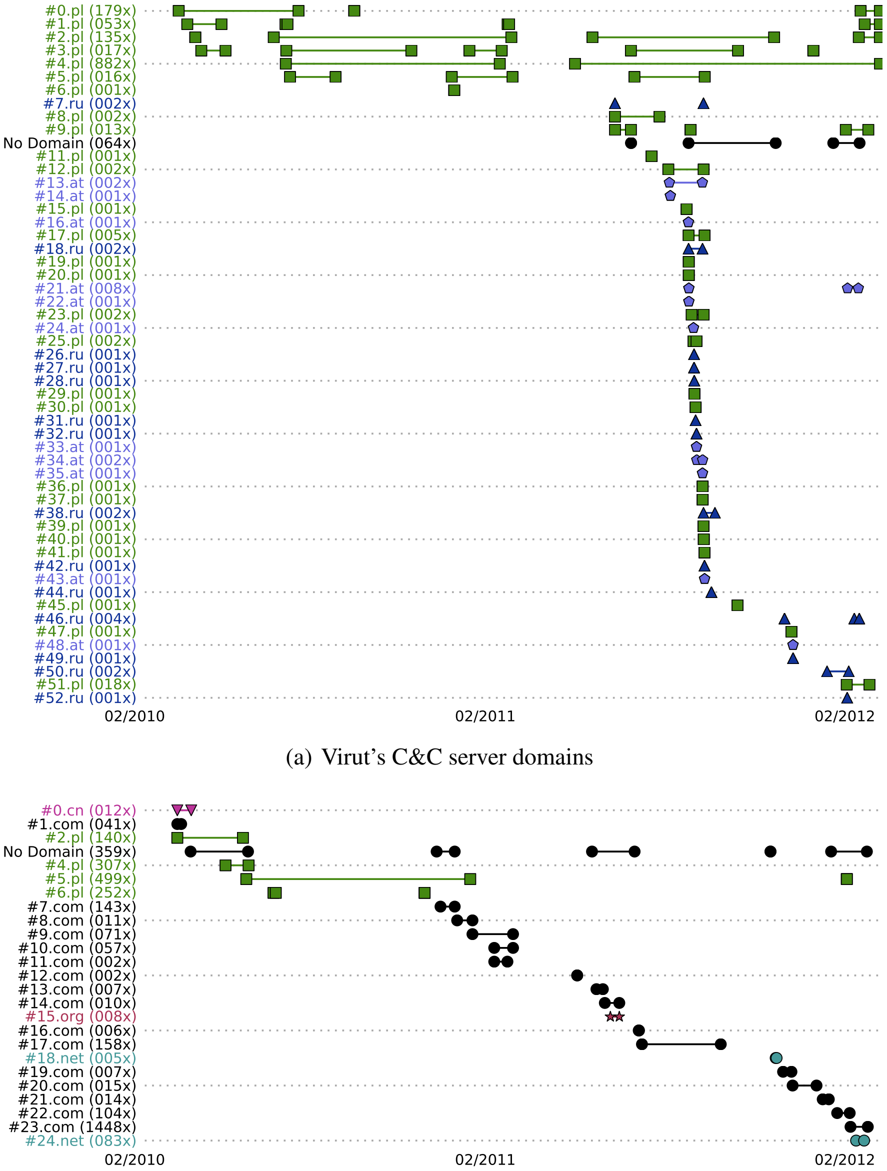
IP addresses or domains (e.g., image hoster) were involved and we explicitly did not exclude them from Table 2 if we also detected C&C.

Outstanding are the P2P variants of Zeus and Sality, with more than thousands of different C&C "server" hosts each. For these downloaders, we consider P2P neighbors that respond to P2P-related UDP requests as active. The large number of ASes involved, 900 for Sality P2P and 446 for Zeus P2P, show that provider-driven initiatives against these P2P networks are deemed to fail. Interestingly, and particularly for Sality P2P, we saw a large fraction of P2P nodes to be lasting for more than four weeks. We first thought this may indicate that defenders joined this particular P2P network, but the high number of long-lasting ASes speaks against this.

The analysis on the domains used by downloaders provides further interesting insights. Zwangi, for example, heavily rotates its C&C IP addresses typically within four /22 networks. Similarly, Gamarue deploys one particular domain pointing to highly fluctuating IP addresses in over 50 different ASes. In both cases the IP addresses are typically reused, i.e., DNS is used to steer downloaders towards multiple C&C servers. On the other hand, we observed downloaders for which the set of IP addresses was relatively constant, but the domains to resolve these IP addresses changed over time. For example, Virut used 45 domains to resolve to its 20 C&C servers, and Renos/Artro pointed its 136 domains to 38 IP addresses. Related to the previous observation that attackers settle their C&C servers in multiple networks, we also show that – for most downloaders – a diverse set of Top Level Domains (TLDs) is chosen. Usually, these C&C domains are even registered across many continents, mostly including European, South-/North-American, and Asian registrars. Again, involving multiple domain registrars is presumably a strategic decision.

It can be seen that a large fraction of C&C servers (20%) remains operable for more than four weeks. Similarly, 217 domains pointing to active C&C servers (7%) remain in active use for at least four weeks. The observed long-levity enables defenders to take actions against downloaders, such as using domain or IP address blacklists. On the other hand, the involvement of numerous registrars and providers shows how complex takedown efforts would be.

As a case study, we compared the usage time spans of Virut's C&C server domains (Figure 3(a)) with the usage time spans of the egg download server domains (Figure 3(b)). Both figures reveal that Virut seems to have a subset of stable domains that have been used throughout the last two years and that are still in active use, for both C&C and egg servers. In addition, several domains have been used only for certain periods. However, the sets of domains for C&C and egg distribution are distinct, i.e. we have not witnessed a single domain being used for both, C&C and egg distribution. Interestingly, we observed a churn of Virut C&C server domain names between June 2011 and January 2012. Our initial hypothesis that these domains were used as backup C&C domains was proven wrong, as many other domains have been actively in use during that period. In addition, our passive DNS database in Sandnet revealed that not a single DNS resolution request for a Virut C&C domain resulted in NXDOMAIN or an empty answer section – Virut thus exhibits a remarkable C&C and egg server availability. We leave a more fine-grained analysis to measure to which extent C&C servers/domains are responsive simultaneously to future work.



(a) Virut's C&C server domains

(b) Virut's egg servers

Fig. 3. Virut's C&C (above) and egg (bottom) server usage by domain over time. Colors/markers denote top level domains.

4.2 Download Server Infrastructure

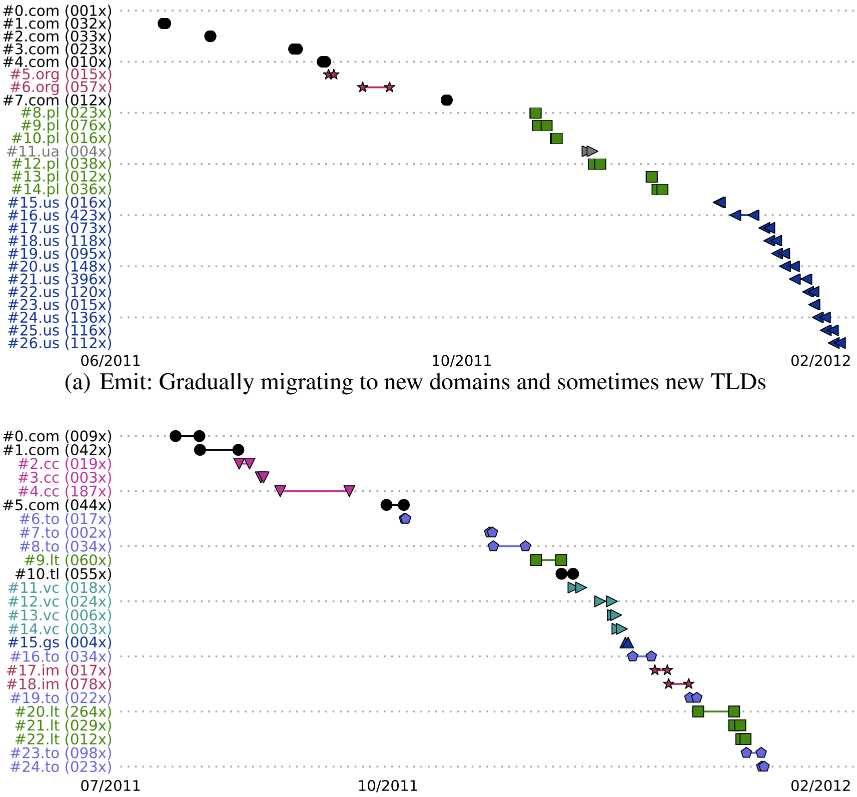
The second pillar of a downloader's infrastructure are the download servers. We will now analyze the infrastructures of downloaders with plaintext download channels. We focus on plaintext downloaders, as we could map download channels to downloaders in these cases with a high accuracy. Table 3 shows statistics on the egg distribution infrastructure for these downloader families. On purpose, we do not consider the C&C infrastructure here, except – unavoidably – in cases where the egg sample download is part of the C&C channel.

Downloader	*IPs*		*Domains*		*Eggs*		*Maximum*	*Packers*	
Family	*#*	*#LL*	*#*	*#LL*	*#*	*#MD5s*	*Uptime*	*#*	*Detected*
dldr-#2	26	7	44	10	1029	110	561 days	6	b,t,u,f,h,y
dldr-#3	8	1	9	1	648	158	114 days	7	u,s,d,n,p,c,e
Dofoil	14	1	29	0	103	93	96 days	3	u,c,Y
Emit	6	2	27	0	5938	698	183 days	2	u,p
GoldInstall	70	25	63	16	13155	971	592 days	8	u,b,s,v,p,w,n,N
Harnig/LoaderAdv	31	12	46	23	1731	735	185 days	9	u,o,f,p,d,N,b,n,M
Rodecap	2	2	8	2	286	23	445 days	2	u,a
Virut (crypt C&C)	30	8	25	6	3852	293	459 days	5	u,x,n,s,d
Vobfus/Changeup	15	7	34	3	2005	424	77 days	1	u
Winwebsec	6	1	1	1	80	22	58 days	n/a	ukn
Zwangi	86	2	8	1	263	138	49 days	n/a	ukn

Table 3. Statistics on the egg sample distribution infrastructure per downloader family. LL=Long Lasting, i.e., uptime of more than 4 weeks. Packers: u=UPX, t=Themida, p=PECompact, e=PEtite, b=BobPack/Bobsoft, a=Armadillo, s=ASPack/ASProtect, x=EXECryptor, h=Thinstall, n=NsPack, f=FSG, d=D1S1G, v=Upack, c=CrypKey, o=ProActivate, y=XtremeProtector, w=WinUpack, N=NET MS, M=MoleBox, Y=y0dasCrypter

Two thirds of the observed plaintext downloader families exhibit more than ten distinct IP addresses for their sample servers. A similar trend is observed concerning the domain names – only the Winwebsec family does not make use of DNS at all in the egg download process.

Per downloader family, the maximum uptime expresses the maximum time span where one single egg server IP address has been witnessed as serving egg samples. Note that, in comparison to Table 2, the measurement in Table 3 is restricted to a family's egg servers and omits its C&C infrastructure. We consider an IP address or domain as long lasting if it serves eggs for at least four weeks. Table 3 shows that over the whole monitoring period, only a small fraction of the IP addresses is actually long lasting. In the cases that we manually inspected, we observed that downloaders typically move their download servers from time to time. For each downloader family of Table 3, we manually inspected the egg server usage over time for both, domains and IP addresses. Interestingly, all downloader families exhibit similar egg server usage patterns where the migration from one domain to another is clearly visible. The same applies to the IP addresses of egg servers, however, egg server domains typically change more often than IP addresses. Some of the servers that we observed to be long lasting, even actively serve eggs for more than a year.



(a) Emit: Gradually migrating to new domains and sometimes new TLDs

(b) Vobfus/Changeup: Migrating to new, previously unseen TLDs

Fig. 4. Egg server usage by domain over time for two downloaders. Domain names have been pseudonymized. Marker styles and colors distinguish the download server's top level domain.

For the downloader family Emit, Figure 4(a) shows each egg server domain on the y-axis and the associated usage time spans. Note that the domain names have been pseudonymized. The egg server domains show hardly any overlap in their usage time spans. In addition to the usage time span, the marker and the color denote the top level domain. We observe that not only does the egg server move from one domain to another – indicated by the pseudonym – it also migrates from one top level domain to another, i.e. from initially .com to .org, .pl and finally to .us. This pattern shows that – in order to strive for a takedown of this downloader's egg serving infrastructure on the DNS level – many different registrars would be required to cooperate.

Figure 4(b) Vobfus/Changeup, which exhibits a strong domain migration pattern for its egg servers. In this case, the domain names are typically only used for a couple of days, and never reused. Not as consistent as Emit, but still, Vobfus exhibits sequential top level domain migration, too, although a few top level domains are used in parallel.

5 Egg Acquisition and Analysis

After investigating the downloader infrastructure, we will now analyze the downloaded eggs. Such an analysis allows us to draw conclusions on how attackers operate the egg infrastructure, e.g., by using polymorphism and aggressively repacking served samples. We will begin with presenting two techniques how to acquire eggs from both plaintext and encrypted downloaders. The resulting dataset of actively acquired eggs will then serve to give first insights into evasive techniques used by downloaders.

5.1 Egg Acquisition Techniques

All downloader infrastructures have one necessity in common: these services must be publicly accessible, as (with the exception of targeted attacks) fraudsters aim for large-scale deployment of their malware. Consequently, attackers cannot easily deploy client authentication mechanisms that prevent their infrastructures from being "abused", raising the difficulty for attackers to control who is accessing the infrastructures. We exploit these necessities to obtain eggs for the downloaders under our analysis. We present two techniques that enable us to acquire the downloaded eggs for plaintext and encrypted droppers. Previous efforts analyzing a few specific downloaders [5] did not require automated egg acquisition techniques. However, given our significantly larger sample set, we seek for a more scalable solution to analyze downloaders. Our techniques may be a potential enabler for future research on malware acquisition methods or egg analysis, as our methods do only require little manual effort compared with reverse engineering.

Plaintext Downloaders For plaintext downloaders, we exploit the fact that eggs are downloaded without disguising or encrypting the communication. Methodically, we replay the egg-download dialog towards each download server and require new egg samples this way. For example, in case of HTTP, once the download server and the egg's URI is known to defenders, downloads can be repeated regularly. We implemented a *dialog repeater* that takes pairs of HTTP request and communication endpoint as input, i.e., payload bytes with a destination IP address and port. For each such pair, the repeater replays the dialog towards the specified destination once an hour, typically resulting in HTTP responses. We feed the repeater with input pairs by searching for requests by downloaders in our dataset that led to egg downloads. Given the prevalence of HTTP in our dataset, we left it open for future work to incorporate further network protocols to the dialog repeater.

In order to avoid such mechanisms, fraudsters could potentially use blacklists of IP addresses of known malware analysis systems [1]. For an attacker, it is straightforward to block all requests from systems as ours. Consequently, instead of using a single Internet outbreak and IP address, we established a proxy network to route the traffic through our home DSL lines. In contrast to well-known proxies such as Tor or open proxies, end-user IP addresses seem to stem from realistic end-users and – in our case –

even change daily. We made sure that our ISPs did to interfere with our measurements by comparing outputs of multiple proxy hosts. Despite its simplicity, as we will show, the repeater is a well-working mechanism to acquire new eggs.

Encrypted Downloads A drawback of the dialog repeater is that it cannot milk eggs from downloaders using encrypted download channels. Even if the download succeeded, we could not make use of the encrypted egg. Therefore, as a complementary technique, we leverage the actual downloader to acquire eggs. The intuition behind this method is simple: whenever a downloader is executed, it will download and execute previously unknown malware samples. We instrumented our Sandnet VMs with a kernel-based Windows system driver that records the file images whenever new processes are forked or system drivers are loaded. For each potential egg being executed, the kernel driver computes the MD5 checksum and records the new processes' image.

However, monitoring new processes results in a large amount of legitimate system files to be interpreted as potential egg. To filter legitimate system files, we built a whitelist of trusted system files by scanning all files of a clean Sandnet VM. In addition, as a further filter to catch only actually dropped and not modified system files, we manually assembled patterns for the file paths where each downloader is storing its eggs. We specifically discard eggs that we identify as exact or repacked/modified copies of the downloaded itself by correlating the time when data was received from the network with the time when the new process was forked. After adding the kernel driver to Sandnet, we additionally scheduled the downloader families with encrypted download channels for execution in Sandnet on a daily basis for seven weeks starting in Jan 2012.

5.2 Egg Sample Distribution

In addition to our passive Sandnet database, we use both active techniques described to obtain a comprehensive egg dataset. Thus, for the plaintext downloaders, we identified the download channels and additionally describe the downloader infrastructure and their uptime. Table 3 (page 13) shows the egg distribution per downloader family that exhibit plaintext egg downloads.

The number of successful egg downloads as well as the number of MD5 unique egg samples differs widely among the plaintext downloader families. Whereas for GoldInstall more than 13,000 egg downloads completed successfully, the number of unique egg samples is much smaller. Other families such as Dofoil show that still a significant fraction of the successful egg downloads expose differing MD5s.

Table 4 summarizes our experiments of actively milking encrypted downloaders in Sandnet. For each downloader, we name the number of executions in Sandnet and show the number of eggs and unique eggs, respectively. For nine of ten downloaders, our technique was able to trace eggs. Despite its short runtime and the relatively small number of execution per downloader, we were able to acquire a high diversity of eggs. For example, although Zeus is well-known for keylogging and information stealing, we can confirm Symantec's recent observation [3] that it also downloads non-Zeus samples. For Sality P2P, we have observed active downloads, but the eggs were never executed during our monitoring period. Consequently, our kernel driver did not record new processes. Renos/Artro drops malware, although from August 2011 on our Renos samples

Family	*Execs*	*Eggs*	*MD5s*	*Packers*
Buzus	316	1898	329	b,u,p
Cycbot/Gbot	181	1030	374	u,c
Dabvegi	278	271	8	unknown
dldr-#1	14	10	3	t
Karagany	256	242	178	z
Renos/Artro	320	2454	23	u
Sality	261	241	59	u
Sality P2P	250	0	0	n/a
TDSS/Alureon	226	652	79	n/a
Zeus P2P	224	221	101	n/a

Table 4. Downloaded egg samples from the encrypted downloaders from Dec '11 to Feb '12. Packers: u=UPX, t=Themida, p=PECompact, b=BobPack/Bobsoft, c=CrypKey, z=StealthPE.

were effectively sinkholed. Independent from the fact that all eggs were included in the original downloader sample, our technique could in fact extract the eggs. Both for Renos/Artro and Sality P2P, we could in general milk the downloaders, if we executed more recent samples or increased the analysis period. The low number of executions of dldr-#1 is due to scheduling this downloader only recently. As a particularly interesting case, TDSS/Alureon dropped all recorded executables by extracting the original sample. In addition to the loaded eggs, however, we recorded that in about half of the executions a kernel driver was loaded, showing that our technique may even work for downloaders with rootkits capability.

5.3 Polymorphism

Malware is well-known for polymorphism in order to evade antivirus signatures. An interesting question in the context of downloaders is whether and how polymorphic code is used. We approached this aspect twofold. First, we classified all egg samples using yara [2] and packer identification rules in order to assign which packer was used to (re)pack an egg sample. In addition, we submitted the egg samples to our sample sharing partners and Virustotal. In turn, querying Virustotal, we were thus able to assign A/V labels to the egg samples.

Sample Packing A large fraction of the egg samples were successfully classified using yara packer rules. Tables 3 and 4 show the number of distinct packers for the eggs of each downloader family. The dominating packers are based on UPX. However, many different packers can be found, such as Armadillo, Themida, ASPack, ASProtect, NsPack and PECompact. In addition, some eggs, such as those of Winwebsec, were packed with unknown packers. The fact that the egg packers vary throughout one downloader family, supports the assumption that there are multiple “clients” per downloader and that it is likely not the download server that repacks the eggs. Instead, we assume that the clients make packed eggs available to the downloaders. In this context, we consider a client to be an attacker willing to distribute malware via downloaders.

Repacking In order to successfully evade signature-based A/V, eggs are repacked at certain intervals. For those families that have plaintext egg downloads, based on Sandnet and dialog repeater traces, we estimate lower bounds on which downloader families distribute polymorphic eggs. In this context, we define an egg to be repacked if different content – in terms of MD5 hash – is served for what can be considered the same egg sample – based on (approximate) file size and A/V label. Thus, for each downloader family, we consider an egg to be repacked if we observe egg downloads with at least 8 distinct MD5 egg hashes all having (nearly) the same file size (rounded to kilobytes) and the same A/V label, within a time span of one month. On average, our filter criteria translate to a repacked egg sample at least once every four days. Furthermore, to ensure statistical significance, we limit our dataset for this experiment to families with at least 90 distinct eggs. Of those 9 families, we observed 8 to exhibit repacked samples. Note that we do not consider repacking to be a property of the downloader family. Instead, we assume that the clients of these downloaders take care of the repacking of their eggs. Wheres at least one client of Emit reached a maximum repacking rate of once every 17 minutes, dldr-#3 only repacked up to once every 2.5 days. For GoldInstall, we measured repacking once a day, and one of the Dofoil clients repacked its eggs once every hour.

This confirms similar analyses by Cabellero et al. [5], only that two downloaders in our dataset (Emit, Dofoil) deploy overly aggressive repacking. Employing our dialog repeater, we looked for server-side polymorphism where the egg sample is repacked upon *each* request. In particular, we tried to measure whether the repacking of Emit eggs takes place via on-the-fly server-side polymorphism, but unfortunately the egg servers have not been reachable during this experiment.

6 Discussion and Future Work

Motivation of this work: Our analyses provide detailed, novel and important insights into malware downloaders, but one may wonder if revealing such data has positive effects to the security community. In particular, revealing the possibility to monitor downloaders may motivate attackers to switch to more advanced techniques. However, given the large numbers of long-term operating downloaders, we see the need to raise attention to this problem domain. Our work also aims to highlight relevant downloader families, as e.g. P2P- or rootkit-driven downloaders, fostering future research on potentially previously unknown malware families.

Evasion: Obviously, our techniques to automatically milk downloaders are evadable by attackers. While it is straightforward to evade our dialog repeater, evading our kernel-based driver requires more thoughts. For example, we face the risk that our current setup may fail for kernel-level rootkits such as TDSS. Similarly, we had to exclude one particular downloader (Wintrim) from our analysis, as it detects virtualized environments. However, hardened dynamic analysis as with Ether [6], hardware-based hosts [8], or developing resilient kernel drivers would be effective against attackers' moves.

Containment: During dynamic analysis, and particularly when allowing network access to malware, we potentially risk to harm others. However, in a best effort to drop all harmful traffic, we strictly control and monitor Sandnet's activity. As a consequence, we have not observed a single abuse complaint concerning Sandnet, so far. Furthermore, a particular risk of executing downloaders is to – unintentionally – financially support PPI downloaders, in that attackers are paid for each installation. However, the cash flow in this case is that attacker A (whose downloader is executed in Sandnet) is paid by attacker B (who asked for his malware being dropped), not causing harm to any innocent uninvolved individual.

Next steps: Until now, we mainly focused on characterizing downloaders, observing their infrastructures and analyzing downloaded eggs. Due to space and time constraints, we did not explore and include all analyses on the downloaded eggs, which we plan to work on as a follow-up of this paper. We also plan to extend our large downloader dataset for active techniques, e.g., to measure the prevalence of downloaders or to explore possible detection/mitigation techniques.

7 Conclusion

We identified and characterized 23 downloader families, showing that the downloader landscape is diverse in terms of architectural design, communication protocols and encryption schemes being used. We observed that many downloaders – albeit sometimes simple – have been actively operated for more than a year. Motivated by this observation, we analyze how attackers ensure the resilient operation of their downloader infrastructure. For example, we show that downloaders migrate their C&C servers aggressively among different Autonomous Systems, often involving multiple countries. Similarly, we observed downloaders not only to alter the C&C domains frequently, but also to involve diverse domain registrars. We revealed further details on the workings of downloaders, such as server-side polymorphism, by analyzing the download server infrastructure. These observations show that mitigating the problem of downloaders is more difficult than it might seem. To foster future research in this area, and as automated mechanism to acquire previously unseen malware samples, we present two generic techniques which extract downloaded eggs from any downloader.

Bibliography

[1] Antivirus tracker. http://avtracker.info/.

[2] yara-project - A malware identification and classification tool. http://code.google.com/p/yara-project/.

[3] Andrea Lelli, Symantec. Zeusbot/Spyeye P2P Updated, Fortifying the Botnet. http://www.symantec.com/connect/blogs/zeusbotspyeye-p2p-updated-fortifying-botnet.

[4] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th EICAR Conference*, 2006.

[5] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *20th USENIX Security Symposium, San Francisco, CA, August 2011*, San Francisco, CA, August 2011.

[6] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In *15th ACM Computer and Communications Security Conference, Alexandria, VA, October 2008*, 2008.

[7] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *NSDI*, 2009.

[8] D. Kirat, G. Vigna, and C. Kruegel. BareBox: Efficient Malware Analysis on Bare-Metal. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2011.

[9] J. Nazario and T. Holz. As the Net Churns : Fast-Flux Botnet Observations Tracking Fast-Flux Domains. In *3rd International Conference on Malicious and Unwanted Software (Malware '08)*, 2008.

[10] M. Neugschwandtner, P. Milani Comparetti, and C. Platzer. Detecting Malware's Failover C&C Strategies with SQUEEZE. In *27th Annual Computer Security Applications Conference (ACSAC), Orlando, Florida, December 2011*, 2011.

[11] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: Automatic Protocol Replay by Binary Analysis. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 06)*.

[12] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *ACM EuroSys BADGERS*, 2011.

[13] Sergey Golovanov, Vyacheslav Rusakov. TDSS. http://www.securelist.com/en/analysis/204792131/TDSS.

[14] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*.

[15] ThreatExpert. http://www.threatexpert.com.