The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics

Bum Jun Kwon
University of Maryland
College Park, MD, USA
bkwon@umd.edu

Jayanta Mondal
University of Maryland
College Park, MD, USA
jayanta@cs.umd.edu

Jiyong Jang
IBM Research
Yorktown Heights, NY, USA
jjang@us.ibm.com

Leyla Bilge
Symantec Research Labs
France
leyla_bilge@symantec.com

Tudor Dumitraș
University of Maryland
College Park, MD, USA
tdumitra@umiacs.umd.edu

ABSTRACT

Malware remains an important security threat, as miscreants continue to deliver a variety of malicious programs to hosts around the world. At the heart of all the malware delivery techniques are executable files (known as *downloader trojans* or *droppers*) that download other malware. Because the act of downloading software components from the Internet is not inherently malicious, benign and malicious downloaders are difficult to distinguish based only on their content and behavior. In this paper, we introduce the *downloader-graph abstraction*, which captures the download activity on end hosts, and we explore the growth patterns of benign and malicious graphs. Downloader graphs have the potential of exposing large parts of the malware download activity, which may otherwise remain undetected. By combining telemetry from anti-virus and intrusion-prevention systems, we reconstruct and analyze *19 million downloader graphs* from *5 million real hosts*. We identify several strong indicators of malicious activity, such as the *growth rate*, the *diameter*, and the *Internet access patterns* of downloader graphs. Building on these insights, we implement and evaluate a machine learning system for malware detection. Our system achieves a *96.0% true-positive* rate, with a *1.0% false-positive* rate, and detects malware an average of 9.24 days earlier than existing anti-virus products. We also perform an external validation by examining a sample of unlabeled files that our system detects as malicious, and we find that 41.41% are blocked by anti-virus products.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

Keywords

Downloader Graph; Malware classification; Early detection

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813724.

1. INTRODUCTION

Because cyber criminals constantly re-pack and obfuscate malicious software (malware) to evade detection, the security community has turned its attention to *content-agnostic* techniques for detecting malware [3, 5, 10, 14, 20, 28, 30, 31]. For example, research has generally focused on understanding the properties of the global malware distribution networks, such as their business models [3], their network-level behavior [10, 30], or their server-side infrastructure [14,31]. Prior research has also investigated how malware samples are *downloaded* on end hosts, e.g., through drive-by-download attacks [20] (which exploit vulnerabilities in web browsers when users are surfing the Web), pay-per-install infrastructures [3] (which are paid services that distribute malware on behalf of their affiliates), or self-updating malware (e.g., worker bots from recent botnets [35]). Comparatively, less attention has been given to the executable files (colloquially called *downloader trojans* or *droppers*) that download a variety of supplementary malware (called *payloads*) at the heart of malware distribution techniques.

It is not trivial to distinguish benign and malicious downloaders simply *based on their content and behavior* because the act of downloading software components from the Internet is not a sign of inherently malicious intent. For example, many benign applications download legitimate installers and software updates. Owing to social engineering or drive-by-download attacks, benign applications (e.g., web browsers) download malicious programs, which may then download additional malware. Some malware droppers are known to be active for over two years [23] due to the difficulty of determining their maliciousness.

However, the *downloader-payload relationship* of executable files on a host, and the *downloader graphs* generated by the relationship, can provide unique insights into malware distribution. Figure 1 shows a real downloader graph example that illustrates this opportunity. A benign web browser (node A) downloads two files from the same domain: nodes B and D that are unlabeled (i.e., not known to be malicious or benign). However, node B downloads additional files, some of which (nodes C and F) are detected as malicious, suggesting that node B, and potentially node D downloaded from the same domain as node B, as well as *all of the nodes reachable* from them in the downloader graph, are likely involved in malware distribution. Consequently, by analyzing the downloader graphs on a host we can identify large parts of the malware download activity, which *may otherwise remain undetected*.

In this paper, we conduct a systematic analysis of downloader graphs in the wild, exposing the differences between the *growth*

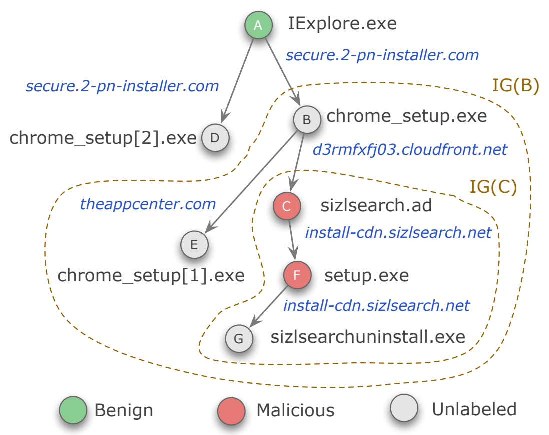

Figure 1: Example of a *real* downloader graph and the influence graphs of two selected downloaders.

patterns of benign and malicious graphs. We construct downloader graphs by combining telemetry collected by Symantec's intrusion prevention systems (IPS) and anti-virus (AV) products on real hosts targeted by malware distribution networks. This data is collected from users who opt in for Symantec's data sharing program and is available on the WINE analytics platform [7]. Specifically, we correlate reports of Portable Executable (PE) file downloads over HTTP with records of executables present on end hosts to reconstruct the download activity on *5 million Windows hosts* around the world. We further introduce the notion of *influence graph* (marked by dotted outlines in Figure 1), which is the subgraph reachable from a given downloader on the downloader graph. Intuitively, the influence graph represents the download activity that a downloader has caused on a host. We identify *19 million influence graphs* in our data. We label 15 million of these graphs as benign and 0.25 million as malicious, using data from VirusTotal, the National Software Reference Library (NSRL), and an additional ground truth data set we received from Symantec.

We demonstrate that the growth rate of influence graphs is a strong indicator of malicious activity, as successful malware campaigns deliver their payloads slowly to evade detection. Specifically, almost 88% of influence graphs with *average inter-download times greater than 1.5 days/node* are malicious, and 65% of the malicious influence graphs have inter-download times above this threshold. Additionally, 84% of the influence graphs with *diameter* $\geq$ 3 *are malicious*, as competition and arbitrage opportunities in the underground economy [3] result in situations where malicious droppers download other droppers. We also find that the average number of downloaders accessing a domain and the average number of files downloaded from a domain, computed per influence graph, are relevant to certain classes of benign and malicious downloaders. Surprisingly, *55.5% of malicious downloaders are digitally signed* (22.4% have valid signatures), suggesting that the organizations responsible for a large part of the malware download activity are *not trying to evade attack attribution*.

We use the properties of influence graphs as features to learn a classifier for identifying malicious executables. Because of the differences in the influence graphs depending on malware classes and maturity stages produced by the slow growth rates (which could be perceived as a noise by a learning algorithm), we choose to employ a *random forest classifier* that is known to perform well with such variability in the data. Our classifier achieves a *96.0% true positive rate*, with a *1.0% false positive rate*, using 10-fold cross validation. We also estimate, conservatively, that our classifier would be able to detect unknown malware an average of *9.24 days earlier* than the anti-virus products employed by VirusTotal. Finally, we perform an *external validation* of the classifier by querying VirusTotal for some of the unlabeled samples predicted to be malicious, and find that 41.41% of them are reported as malicious by anti-virus products.

In summary, the paper makes the following contributions:

- We build a large data set of malicious and benign download activities on 5 million real hosts by reconstructing download events from IPS and AV telemetry. We also build a ground truth of malicious and benign downloaders by combining three data sources.
- We propose a graph-based abstraction to model the download activity on end hosts, and perform a large measurement study to expose the differences in the growth patterns between benign and malicious downloader graphs.
- We use insights from our measurements to build a malware detection system, using machine learning on downloader graph features, and evaluate it using both internal and external performance metrics.

Outline. The rest of the paper is organized as follows. In Section 2 we overview the threat of malicious downloaders and we state our goals. In Section 3 we formally introduce the downloader graph abstraction, and we discuss our data sources and our method for constructing downloader graphs. We then present our measurement results in Section 4, followed by the design and evaluation of our classifier in Section 5. In Section 6 we discuss the implications of our results, and in Section 7 we review the related work.

2. THREAT MODEL

In this section, we present an overview of the security threats imposed by downloaders and articulate the goals of our research.

Overview of Downloaders. Applications often have a particular component that is responsible for determining which software components are required to be installed or updated, and for downloading them from remote servers. In this work, we refer to this component as the *downloader*.

The software delivery process benefits from the ubiquitous access to the Internet, e.g., efficient updating mechanisms, and customizable software installation. As a matter of fact, these benefits make the distribution of both legitimate and malicious software reliable and easy. Multi-phase malware first gains a foothold on a system, and installs a small executable—called *droppers* or *downloader trojans*—which then downloads additional components, such as the rest of the malicious *payload*. An early example of such multi-phase malware was the Sobig email worm discovered in 2003, which downloaded additional files to set up spam relay servers on infected machines and to update itself [27].

The next evolution step was the emergence of general-purpose droppers that could be configured to propagate different malware families (e.g., Bredolab trojan [29]). Some general-purpose droppers represent the client-side part of *pay-per-install* (PPI) infrastructures, which allows malware authors to install arbitrary executables on thousands of hosts and to select their targets based on the properties of those hosts (e.g., their geographical locations) [3]. These infrastructures also have server-side systems designed to be resilient to takedown efforts [3, 14, 18], which may continue to operate for over two years [23]. They employ sophisticated techniques to disseminate malicious payloads to a large number of hosts, e.g., drive-by-downloads [20], social engineering, search engine poisoning [13], and provide this functionality as a service [3]. PPI downloaders represent only a fraction of the current population of down-

Influence graphs	19 million
Files (*graph nodes*)	24 million
Total Downloaders	0.9 million
Benign downloaders	87,906
Malicious downloaders	67,609
Download events (*graph edges*)	50.5 million
Domains accessed	0.5 million
Hosts	5 million

Table 1: Summary of our data sets and ground truth.

loaders, as simpler and older forms continue to operate [23]. In consequence, malware samples often rely on the functionality provided by downloaders.

Problem Statement. In this paper, we conduct a systematic analysis of downloaders in the wild with a special focus on the relationship between the downloaders and the supplementary executables they download. We represent the transitive closure of this relationship using a graph abstraction. Our *first goal* is to uncover the differences between the graph structures constructed from benign and malicious download activities. By leveraging the insights obtained from the analysis, our *second goal* is to propose a new method to detect malicious downloaders. Our method could improve the overall performance of malware detectors by providing earlier warnings for malicious download activities. This approach complements existing host-based malware detection systems as it helps comprehend how various files are delivered to end-hosts using the dropper-payload relationship to connect the dots.

We also have some *non-goals*: in this paper, we do not aim to analyze the server-side infrastructure and network-level behavior of malware delivery networks, to determine for how long these networks remain operational, to profile the organizations involved in malware distribution, or to improve network security. Rather, our main goal is to complement existing approaches focusing on server-side infrastructures [14, 31] and network-level behaviors [10, 30].

3. DOWNLOADER GRAPH ANALYTICS

We start our discussion by describing our data sets, followed by a formal description of the *downloader graph* abstraction and how we construct the downloader graphs from the collected data.

3.1 Data Sources

We infer the download activities on 5 million end-hosts using data available on the Worldwide Intelligence Network Environment (WINE) [7], a platform for data intensive experiments in cyber security provided by Symantec. WINE contains security telemetry collected on real hosts that are often targeted by cyber attacks. The data is collected from the users who opt in for Symantec's data sharing program, and does not include personally identifiable information. Specifically, we use two WINE data sets: (a) *binary reputation*, and (b) *intrusion prevention system telemetry*. The data sets are stored as SQL relations consisting of multiple columns. We extract relevant columns from each relation, and join them to generate the data sets we require for our study. Table 1 summarizes our data.

The binary reputation data includes summarized information about all binary download activities on the Symantec's customers' machines. From this data we collect the following information: the server-side timestamp of the event, the unique identifier for the machine that initiated the download, the name and SHA2 hash of the downloaded file, and the SHA2 hash of the parent file (an archive including the downloaded file or the actual downloader). We extract information about 24 million distinct files, including 0.4 million parent files, from the binary reputation data set.

The intrusion prevention system (IPS) telemetry data provides information about malicious activities detected on network streams. In addition, the IPS telemetry includes logs of network activity on the host that might not be necessarily tied to any malicious activities. The IPS telemetry reports downloads of Portable Executable (PE) files over HTTP. From this data set, we extract the unique identifier of the host, the MD5 hash of the process initiated the network activity (*portal* in IPS jargon), the server-side timestamp and the URL from which the portal downloads the binary. From the IPS telemetry, we extract information about 0.5 million portals.

3.2 Ground Truth Data For Executables

Our ground truth consists of a large number of known-malicious and known-benign files, recorded in VirusTotal, the National Software Reference Library (NSRL), and an additional data set received from Symantec.

VirusTotal. VirusTotal[1] is a service that provides a public API for scanning files with up to 54 different anti-virus (AV) products, and for querying hashes to retrieve their previous analysis reports. We query VirusTotal for each downloader in our data set to obtain its first-seen timestamp, the number of AV products that flagged the binary as malicious, the total number of AV products that scanned the binary, and the corresponding file signer information. We then compute the percentage r_{mal} of products that flagged the binary as malicious. We consider that a file is malicious if $r_{mal} \geq 30\%$; because AV vendors are typically worried about false positives, we believe that this represents a conservative threshold. We verified that approximately 80% of the files above this threshold are also labeled as malicious in the Symantec ground truth described below.

National Software Reference Library. NSRL[2] is a project that collects software installers from leading software vendors with the purpose of creating a reference data set (RDS) of benign software. RDS is a collection of digital signatures (mainly SHA1, MD5, and other metadata) of known, traceable software applications. NSRL releases the RDS four times per year. We used the RDS 2.47 version that was released in December 2014. We treat all the downloaders with matching hashes in NSRL as benign.

Additional Ground Truth for Files. Due to the limitations imposed by the VirusTotal API, we were unable to query all the 24 million file hashes. We therefore complement our ground truth with an additional ground truth maintained by Symantec. This step increases the coverage of our ground truth.

After combining all these sources of ground truth, we identify 87,906 benign and 67,609 malicious downloaders; the rest of the downloaders remain unlabeled.

3.3 Downloader Graph

In this section, we introduce the notion of *downloader graph* (DG) and *influence graph* (IG). A DG is a directed graph, defined for each host machine, where a node represents a downloaded file and an edge from downloader d_a to d_b indicates that d_a has downloaded d_b on the corresponding host machine. On the other hand, an IG is defined for each individual downloader (called *root*) on a given host machine, and is a subgraph of the DG on that host. *Influence graph* captures the *impact* of its *download root* by encoding the downloads (both direct and indirect) caused by the root.

Figure 1 illustrates an example of a DG and two IGs (in looped dotted lines) in it. In this example, the nodes are annotated with their file names, and edges are annotated with the domains of

[1]https://www.virustotal.com/

[2]http://www.nsrl.nist.gov/

the download URLs. We discuss additional properties of the nodes and edges of the downloader graph in Section 3.4. Note that the downloader graph per machine could be disconnected, unlike a connected one in this example. Another observation is that the influence graph of a downloader could be contained in the influence graph of another downloader; in particular, the influence graph of a malicious dropper can be a part of the influence graph of a benign downloader.

Now we provide the formal definition of the DG abstraction. Considering $\mathcal{V}$ to be the set of all executables in our data set, and $\mathcal{M}$ to be the set of all host machines, downloader graph $\mathcal{G}_i$ for machine $\mathcal{M}_i$ is defined as $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \alpha, \beta)$, where:

- $\mathcal{V}_i \subseteq \mathcal{V}$ denotes the set of executable files downloaded on machine $\mathcal{M}_i$. Note that the same executable could be downloaded across multiple machines.
- $\mathcal{E}_i \subseteq \mathcal{V}_i$ X $\mathcal{V}_i$ denotes a set of *directed edges* that correspond to the download relations between executables.
- α denotes a set of properties defined on the nodes of the downloader graphs. Node properties could be: (a) machine-dependent (unique across all machines), and (b) machine-independent (unique to a host machine) properties.
- β denotes a set of properties defined on the edges of the downloader graphs.

Now, we define influence graph. The influence graph $\mathcal{I}_g(d_{\mathcal{M}_i})$ of a *download root* d in machine $\mathcal{M}_i$ is defined as the subgraph of $\mathcal{G}_i$ which is reachable by traversing $\mathcal{G}_i$ starting at d. Note that, the influence graphs are defined for every downloader (i.e., an executable that has downloaded at least one executable) in a DG, but not for the other executables in a DG.

3.4 Constructing DGs and Labeling IGs

In this section, we start by describing how we construct the DGs for each machine using the data described in Section 3.1. Then for each possible *root* of the DGs, we extract the IGs and label as *benign*, *malicious*, or *unlabeled* using the ground truth data for the individual downloaders.

Constructing Downloader Graphs. We start by extracting the names of the downloaded files from the URL column of the IPS telemetry data set (example entry: `http://somedomain.com/file_name.exe`); this corresponds to the name of the file created on a disk. 95% of the URLs from which users downloaded PE files include the name of the file downloaded. We then search for these files in the binary reputation data set, which reports file creation events and includes the corresponding SHA2 hashes. If a matching filename and source domain appear in the binary reputation data set within ± 2 days from the IPS event timestamp, we create a graph node for the file, add the file hash as a node attribute, and add an incoming edge from the portal reported in the IPS event. In consequence, we may miss certain graph edges in some rare cases (e.g., the user changes the filename, the malware renames itself), but the edges we create correspond to genuine download events. We look 2 days before or after the IPS event because server-side timestamps may be affected by transmission delays and different submission orders between the two data sets. We employ an approximate file name matching algorithm by computing the edit distance between file names and by accepting pairs with distance below 3 as matches. This allows us to handle differences in the file name caused by duplicate downloads (e.g., `setup.exe` and `setup[1].exe`). We also extract the domain name from the URL and add it as an attribute to the edge.

We also analyze the relationship between files and their parents in the binary reputation data set. If there exists a parent for the downloaded file, we consider that the parent downloaded the file and add a directed edge from the parent to the file; then we assign the domain name extracted from the downloaded file's source URL as an attribute of the new edge. The step was motivated by our observation that many of the parents recorded in the binary reputation data set are the same as the portals in the IPS telemetry.

During the downloader graph construction, we add the following properties to each node in the graph:

- *Number of outgoing edges:* This captures the download volume of a downloader.
- *Number of incoming edges:* This captures how often an executable is downloaded on a host.
- *Time interval between a node and its out-neighbors:* This captures how quickly, on an average, a downloader tends to download other executables after it was downloaded on a host.
- *File score (based on digital signatures):* We assign a file score in the range 0–3 for every downloader in our data set based on their digital signatures. Specifically, we take into account the availability of the following information: (a) signature, (b) publisher information, and (c) reputation of the certification authorities. The executables without file signatures or records in VirusTotal will be assigned with score 0. Otherwise, if the signature contains information about the publisher, we assign score 1. Then, if the certificate validation chain contains a certificate authority among Comodo, VeriSign, Go Daddy, GlobalSign, and DigiCert, we add 1 to the score. Finally, if the signature is valid, we also add 1 to the score.
- *Number of out-neighbors with score 0 or 1:* This represents a rudimentary quantification of the malicious intent of a downloader.

We add the following properties to each edge in the graph:

- *URL has IP instead of domain:* denotes if the corresponding download URL had a domain name or an IP address.
- *URL is Localhost:* denotes if the corresponding download URL was relative to a localhost address.
- *URL is in Alexa top 1 million:* denotes if the download URL is in Alexa[3] top 1 million websites.

We also extract some aggregated properties, which we will leverage to derive some features of influence graphs:

- *File prevalence (FP):* For every file, we count the number of hosts it appears on, in the binary reputation data set.
- *Number of unique downloaders accessing given URL (UDPL):* For every URL (domain), we count the number of unique downloaders that used the domain to download new executables, aggregated across all machines.
- *Number of unique downloads from a given URL (UDFL):* For every URL (domain), we count the number of unique executables downloaded from the domain, aggregated across all machines.

Figure 2 illustrates the distribution of nodes, edges, and life span for our influence graphs. There are 3.66 nodes on average per influence graph, with a minimum 2 nodes and a maximum of 66,573 nodes. Influence graphs have between 1–66572 edges, with an average of 2.66 edges. These graphs have an average life span (difference between the timestamps of the first and the last node in the graph) of 75.3 days, ranging from 0 to 2199.9 days. Because this paper focuses on analyzing the properties of downloader graphs, in the rest of our analysis we exclude the influence graphs with fewer

[3] `http://www.alexa.com/`

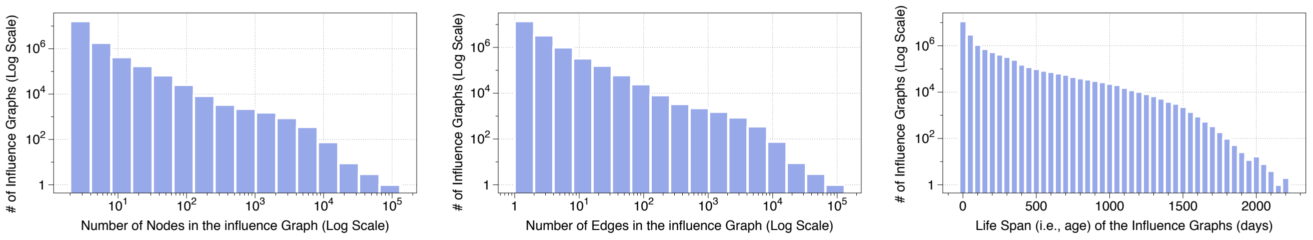

Figure 2: Distributions of influence-graph properties: (a) Number of nodes, (b) Number of edges, (c) Life span.

than 3 nodes, which might not provide sufficient insight for our problem.

Labeling Influence Graphs. We label the influence graphs, using the benign and malicious labels of downloaders, determined as described in Section 3.2. We label only the IGs whose root downloader is known to our ground truth. Specifically, we consider that an IG is malicious if its root downloader is labeled as malicious. On the other hand, we consider that an IG is benign if one of the following three conditions is true: (1) the root is in NSRL, (2) the digital signature of the root is from a well known publisher and verified by VirusTotal, or (3) the root is $r_{mal} = 0$ and is benign in Symantec ground truth, and the next two are also true: (1) none of the other nodes in the IG have $r_{mal} > 0$, (2) none of the influence graph nodes is labeled as malicious in Symantec ground truth. This results in 14,918,097 benign and 274,126 malicious influence graphs.

4. INSIGHTS INTO INFLUENCE GRAPHS

We now present our insights into the ecosystem of benign and malicious influence graphs. We first investigate the ability of malicious downloaders to stay under the radar of the security community, while delivering malware. This allows us to assess the magnitude of this threat and to create a data set of malicious download graphs, unbiased by interference from anti-virus products, for further empirical analysis. We then analyze the properties of influence graphs that correspond to malicious and benign downloaders, and identify properties that can be utilized in malware detection.

4.1 Unknown Droppers

The functionality of benign and malicious downloaders is similar: both retrieve software components from the Internet, sometimes in response to commands received from a remote server. We therefore expect that it takes a while until the security community recognizes that a downloader is involved in malicious activities.

For each downloader labeled as malicious in our data set, we compute the earliest file timestamp in the binary reputation data from WINE, which approximates the date when the sample appeared in the wild. We compare this timestamp with the time when the file was first submitted to VirusTotal, which approximates the date when the malicious dropper became known to the security community. These are not perfect approximations. Antivirus companies change their settings specifically for VirusTotal compared to their commercial versions[4], so we cannot conclude that the droppers it fails to label as malicious are undetectable. Similarly, because the dropper may be delivered initially to a host not covered by WINE, the interval when the dropper remains unknown may be under-estimated. However, given the large number of hosts where data is collected (5 million) and the fact that several anti-virus products (up to 54) must agree, unanimously, that a downloader is not known to be malicious, we believe that the structure of downloader graphs analyzed in this section is representative of the way malicious droppers operate in the wild before they become known to the security community.

[4]`https://www.virustotal.com/en/faq/`

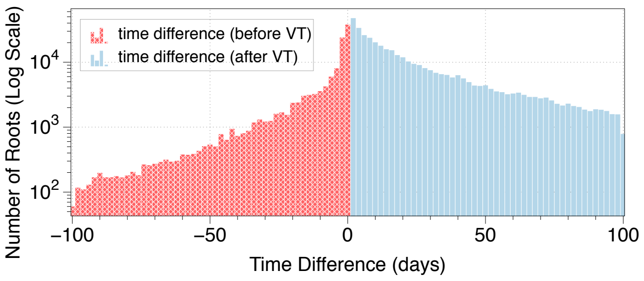

Figure 3: Distribution of the interval between the earliest file timestamp in WINE and the first seen time from VirusTotal, for all the malicious downloaders.

Figure 3 shows the distribution of the time interval between the timestamp of the downloader on each host and the first-seen timestamp in VirusTotal. Negative time intervals correspond to unknown droppers. We identify 140,062 influence graphs rooted in unknown malicious droppers. Among the 67,609 malicious downloaders, 36,801 appear at least once before VirusTotal first seen timestamp. In 27.1% of these cases, the dropper appears in the wild one day before it is uploaded to VirusTotal, suggesting that many of these malicious downloaders rouse suspicion. Nevertheless, the distribution has a long tail, with an average of 80.6 days (approximately 2.7 months) before discovery. This suggests that many downloaders are able to deliver malware stealthily for several months.

These results illustrate the magnitude of the downloader threat and the opportunities for improving malware detection. For the empirical results presented in this section, we focus on malicious graphs rooted in an unknown dropper, in order to minimize the bias caused by anti-virus products that may block the growth of the graphs.

4.2 Dynamics of Malware Delivery

Web browsers, updaters and instant messengers. To understand how malware is delivered to end hosts, we first analyze the programs responsible for most downloads in our data set. The top downloaders are well-known programs, which appear in our benign ground truth and which have valid digital signatures. For example, we identify the top-3 browsers by searching the digital signatures for the following <publisher, product> pairs: <Microsoft Corporation, Internet Explorer>, <Google Inc, Google Chrome>, <Mozilla Corporation, Firefox>; we also check that the file name contains the keywords `chrome`, `firefox` or `explore`. In addition to browsers, the top downloaders include software updaters and Skype (an instant messaging program).

Who Drops the Droppers? By analyzing the incoming edges for all the malicious droppers, we determine that 94.8% of them are downloaded using the top-3 Web browsers. This illustrates the

Downloader file name	Payloads
CSRSS.EXE	14801
EXPLORER.EXE	1717
JAVA.EXE	892
DAP.EXE	749
OPERAUPGRADER.EXE	584
SVCHOST.EXE	547
WMPLAYER.EXE	247
IDMAN.EXE	237
CBSIDLM-CBSI145-SUBTITLESSYNCH-ORG-10445104.EXE	209
MODELMANAGERSTANDALONE.EXE	187
KMPLAYER.EXE	140
JAVAW.EXE	105

Table 2: Top benign downloaders dropping malware.

fact that download graphs may include a mix of benign and malicious software and emphasizes the importance of focusing on the influence graph rooted in each downloader in our analysis.

Benign Programs Dropping Malware. In addition to browsers, we observe a number of benign programs that drop malicious executables (see Table 2). These include three Windows process, `EXPLORER.EXE`, `CSRSS.EXE` and `SVCHOST.EXE`. In the case of `CSRSS.EXE`, the 10 most frequently downloaded executables are adware programs (detected by several AV products); 8 out of 10 are from Mindspark Interactive Network. For `EXPLORER.EXE`, the 10 most frequently downloaded executables are adware programs from Conduit, Mindspark, Funweb, and Somoto. In the case of `SVCHOST.EXE` most of top 10 payloads are generic trojan droppers. Executables downloaded from `JAVA.EXE` are specific trojans (Zlob, Genome, Qbot, Zbot, Mufanom, Cycbot, Gbot and FakeAV), and a hack tool (passview). `DAP.EXE` and `IDMAN.EXE` are downloader managers, and the top 5 executables they drop are products signed by Mindspark. For `JAVAW.EXE`, Bitcoin mining executables made the top of the list. Another Java related process, `MODELMANAGERSTANDALONE.EXE`, is also used for dropping trojans. In many of these cases it is difficult to pinpoint the exact program that is responsible for delivering malware; for example, `SVCHOST.EXE` is a process that can contain a variety of Windows services, while `JAVA.EXE` is an interpreter that runs Java programs. However, this illustrates the fact that malware programs, spanning a broad functionality range, often try to remain undetected by infiltrating benign software ecosystems.

Signed Malicious Downloaders. Surprisingly, among the 67,609 malicious downloaders, 22.4% (15,115 downloaders) have a valid digital signature. Moreover, the droppers may be uploaded to VirusTotal after the expiration of their signing certificate; if we count the invalid signatures, 55.5% of malicious downloaders are signed. These signed malicious downloaders form 128,436 influence graphs, accounting for 46.9% of all the malicious influence graphs. The top-5 downloaders with valid signatures are from Softonic International, Amonetize Ltd, InstallX, Mindspark Interactive Network, SecureInstall. All of these downloaders are known to deliver third-party software. Among these, Amonetize is known to be a pay-per-install provider[5]. These software publishers release 1.70, 1.55, 1.98, 0.35, and 1.79 new droppers per day, respectively. This practice likely stems from a desire to evade detection, as benign software is usually signed. However, this also illustrates that the organizations responsible for almost half of the malware download activity identify themselves voluntarily by signing their droppers.

[5]`http://www.amonetize.com/about-us/`

4.3 Properties of Malicious Influence Graphs

We now turn our attention to the question: *How do malicious influence graphs differ from benign influence graphs?* We compared multiple features and feature combinations; for brevity, we report only the strongest indicators of malicious activity.

Large diameter IGs are mostly malicious. The diameter of influence graphs (the maximum length of the shortest path between two nodes) ranges between 2–5. Figure 4(a) shows the distribution. A graph with diameter 2 (a single downloader with multiple payloads) is equally likely to be benign or malicious. However, when the diameter is 3 and above a high percentage (84% in our case) of graphs are malicious. More importantly, almost 12% of the malicious influence graphs have diameter of 3 and larger. These findings are consistent with prior observations of pricing arbitrage in the underground economy [3], where PPI providers distribute their droppers through competing PPI infrastructures.

IGs with slow growth rates are mostly malicious. Figure 4(b) shows the distribution of the *average inter-download time* (AIT) of the influence graphs. We define AIT to be the average amount of time taken to grow by one node. Almost 88% of the influence graphs that grow slowly (AIT > 1.5 days/node) are malicious. The ratio of malicious graphs further increases with slower growth rates. We also observe that over 65% of the malicious influence graphs have AIT > 1.5 days/node. This suggests that successful malware campaigns, which are able to evade detection for a long time, tend to deliver their payloads slowly.

URL access patterns vary across subclasses of malicious/benign downloaders. Figure 4(c) shows the distribution of the average number of distinct downloaders accessing an Internet domain. We compute this number by determining the set of source domains for the nodes in an influence graph and by taking the average of the number of downloaders accessing them, across all the hosts. Even though the distribution does not clearly separates malicious and benign behavior, we found some interesting patterns. The large number of IGs with between 1,100–1,200 downloaders per domain, towards the right side of the plot, is mostly caused by *adware*. In fact, three adware programs, `LUCKYLEAP.EXE`, `LINKSWIFT.EXE`, and `BATBROWSER.EXE` comprise 26% of the IGs in that distribution bucket. This suggests that the organizations behind these programs have resilient server-side infrastructures (and the domains used to host the adware do not have to change very often) and that they frequently re-pack their droppers (in order to evade detection on the client side). Figure 4(d) zooms in on the left side of the same distribution. Most of the benign IGs have up to 10 downloaders per domain. However, we also identified several malicious IGs in this bucket; the top-3 are fake antivirus programs (`EASYVACCINESETUP.EXE`, `LITEVACCINESETUP.EXE`, and `BOAN-DEFENDERSETUP.EXE`), which access Korean domains and seem to be part of the same campaign. Windows Update IGs have between 60–70 downloaders per domain in our data set.

Malware tend to download fewer files per domain. Figure 4(e) shows the distribution of the average number files downloaded from the domains accessed from an influence graph. The figure also illustrates the diversity in malicious behavior. Most of the malicious droppers that download large number of files per domain correspond to adware. For example, 40% of the IGs in the 4000–5000 files-per-domain histogram bucket (the tallest malicious bar) correspond to three adware programs (`LUCKYLEAP.EXE`, `LINKSWIFT.EXE`, and `BATBROWSER.EXE`). Apart from the adware, most of the other malicious droppers (around 40% of all malware) download 1–5 files per domain, as they have to move to new domains after the old ones are blacklisted. Another interesting obser-

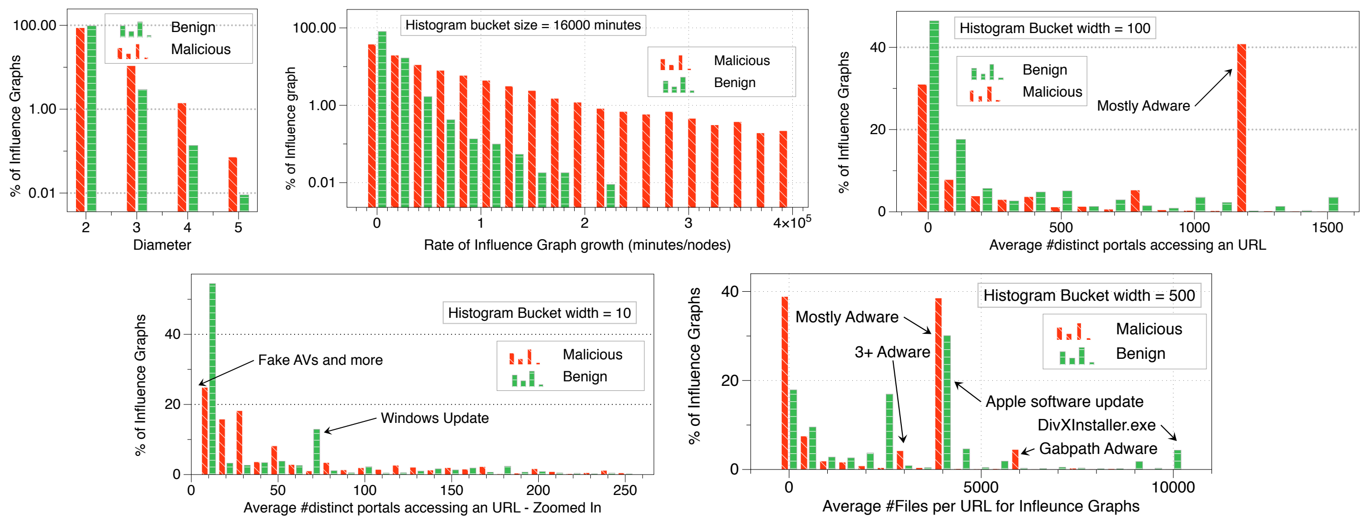

Figure 4: (a) Distribution of influence graph diameter (b) Growth rate of influence graphs (expressed as the average inter-download time) (c) Distribution of the average number distinct portals accessing the domains of an influence graph (d) Zoomed in version of the previous plot (e) Distribution of the number of executables downloaded from the source domains of an influence graph.

vation is that benign downloaders also exhibit diverse behaviors. For example, the influence graphs of Apple Software Update are also in the 4000–5000 histogram bucket, while DivXInstaller's influence graphs download around 10000 files per domain.

5. MALWARE CLASSIFICATION

While in the previous section we identify several features that indicate malicious download activity, none of these features, taken individually, are sufficient for detecting most of the malware in our data set. We therefore build a malware detection system that employs *supervised machine* learning techniques for automatically selecting the best combination of features to separate the malicious and benign influence graphs. Specifically, we train a random-forest classifier using influence graphs labeled as described in Section 3.4; unlike in Section 4, we employ malicious graphs that correspond to both known and unknown droppers, as this data set is more representative of the state of malicious downloaders in the wild. We test our classifier using both *internal* (cross-fold validation and early detection) and *external* (VirusTotal results for some of the unlabeled samples predicted to be malicious) performance metrics.

5.1 Handling Data Skew

The ground data consists of 274,126 malicious and 14,918,097 benign influence graphs. Training a classifier on such a skewed data set may bias the model toward the abundant class (the benign IGs) and may focus on tuning the unimportant features, i.e., the ones that do not contribute to the classification but rather model noise in the dataset. We address this problem through *stratified sampling*: we sample the abundant and rare classes separately to select approximately equal numbers of IGs for the training set. In practice, we do not need to exclude any examples from the rare class, and some examples from the abundant class can be identified easily and filtered to reduce the variability within that class. We therefore filter out all the Web browsers (identified as described in Section 4.2) from the benign set of IGs, as the set of files they download is not predictable and includes both benign and malicious executables. We then keep all the malicious graphs and we downsample the remaining set of benign graphs (sampling uniformly at random). We manually examined the properties of several random samples created in this manner and observed that they closely match the properties of the abundant class. Our balanced training set consists of 43,668 malicious and 44,546 benign influence graphs.

5.2 Feature Engineering

Table 3 provides the features that we compute based on the properties of the influence graphs. We organize the features into five *semantic categories*: internal dynamics, life cycle, properties of downloaders, properties of domains, and globally aggregated behavior. For each feature, we also provide the high level *intuition* for why we expect it would separate malicious and benign graphs.

Depending on how we compute the features, we distinguish two *feature types*. Local features (LF) use information contained in the influence graph. Global features (GF) are also computed for each influence graph; however, they use properties aggregated across all the hosts. An example of a GF is the Average Distinct File Affinity, illustrated in Figure 4(e), which reflects the *tendency of an influence graph* to download files from domains that are known to serve a large number of distinct files.

We quantify the worth of each feature in terms from distinguishing benign and malicious influence graphs using the *gain ratio*[6] with 10-fold cross validation. We select this metric because it is known to be more robust than alternative metrics, such as the information gain or the Gini index, when the features differ greatly with respect to their range [21]. Table 4 shows a summary of the top-10 features in descending order of their gain ratio. The most useful features are the average file prevalence and the features illustrated in Figures 4(c)–(e). We emphasize that, because the features are computed per influence graph, the power of these global features helps classify all the downloaders in the graph. For example, a dropper that normally evades detection because it is present on many hosts, but that always downloads unique files, will likely be classified as malicious because of the low average prevalence of the files in its influence graph.

5.3 Choosing the Classifier

Our data set presents several challenges for supervised machine learning. Some classification algorithms work best on data sets

[6] `http://www.csee.wvu.edu/~timm/cs591o/old/Lecture6.html`

Categories	Name	Type	Explanation
Internal Dynamics (FI)	$\mathbf{f_1}$. Diameter	LF	Diameter capture a chain of download relations (e.g., $A \rightarrow B$, $B \rightarrow C$, and so on). High diameter could imply malicious behavior such as droppers or Pay-per-install ecosystem where there is an affiliate.
	$\mathbf{f_2}$. Clustering Coefficient	LF	Clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together, i.e., create triangles (e.g., (A $\rightarrow$ B, A $\rightarrow$ C, B $\rightarrow$ C) or (A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ A)). Intuitively, we would expect the benign software to show low clustering as compared to malware.
	$\mathbf{f_3}$. Density	LF	A graph with high density means that the binaries are downloading each other actively, and there are binaries that are downloaded by multiple downloaders.
	$\mathbf{f_4}$. Total Download	LF	Total number of downloads made by the influence graph.
Domain Properties (FU)	$\mathbf{f_5}$. Number of Unique Domains	LF	Some malware droppers may access more domains, e.g. if they employ domain generation algorithms to bypass blacklists.
	$\mathbf{f_6}$. Domain Name Similarity	LF	We compute the similarity between all pairs of domains accessed from the graph: $$similarity = 1 - \frac{EditDistance(D_1, D_2)}{\min(length(D_1), length(D_2))}$$
	$\mathbf{f_7}$. Alexa top-1M	LF	Percentage of domains in the influence graph that appear in Alexa top 1 Million list
Downloader Score Properties (FD)	$\mathbf{f_8}$. Average Score	LF	Average score (based on signatures—see Section 3.4) of all the downloaders in an influence graph. Intuitively, even if the root has high score (signed malware) it might download low score downloaders, indicating that the root might be malicious.
	$\mathbf{f_9}$. Standard Deviation of the Score	LF	A malicious influence graph can achieve high average score by downloading known high score downloaders. Standard deviation of downloader score in the IG might be relatively robust in that regard.
Life Cycle (FL)	$\mathbf{f_{10}}$. Influence Graph Life Span	LF	The life span of an influence graph is defined as the time interval between the newest and oldest node. The life span of malicious IGs tends to be shorter, as A/V programs eventually start detecting the droppers.
	$\mathbf{f_{11}}$. Growth Rate	LF	Average inter-download time for the nodes in an influence graph (= $\frac{IG_{lifespan}}{\#Nodes}$). Malicious IG trying to remain stealthy tend to grow slowly, as shown in Figure 4(b).
	$\mathbf{f_{12}}$. Children Spread	LF	Average time difference between the root and the children of an influence graph.
	$\mathbf{f_{13}}$. Intra-children Time Spread	LF	Average difference in download timestamp among the children of the root. Intuitively, we would expect this value to be smaller for malware, as they tend to be more aggressive as downloaders.
Gloablly Aggregated Behavior (FA)	$\mathbf{f_{14}}$. Average File Prevalence	GF	Average FP (see Section 3.4) of the executables that appear in an influence graph. Benign binaries are expected to show high prevalence.
	$\mathbf{f_{15}}$. Average Distinct File Affinity	GF	Intuitively, this depicts whether an influence graph tends to prefer/avoid domains that download less/more distinct binaries. This feature is illustrated in Figure 4(e).
	$\mathbf{f_{16}}$. Average Distinct Dropper Affinity	GF	Intuitively, this depicts the bias of a dropper toward a specific set of domains in order download new binaries. This feature is illustrated in Figures 4(c)–(d).

Table 3: Feature categories and the high-level intuition behind some of the important features

Features	Gain Ratio	Avg. Rank
$\mathbf{f_{14}}$. Avg. File Prevalence	.16 $\pm$ 0	1.1 $\pm$ .3
$\mathbf{f_{15}}$. Avg. Distinct File Affinity	.16 $\pm$ 0	1.9 $\pm$ .3
$\mathbf{f_{16}}$. Avg. Distinct Dropper Affinity	.12 $\pm$ 0	3 $\pm$ 0
$\mathbf{f_{10}}$. IG Life Span	.1 $\pm$ 0	4.3 $\pm$ .64
$\mathbf{f_7}$. Alexa Top-1M	.1 $\pm$ 0	5 $\pm$ 0.5
$\mathbf{f_{11}}$. Growth Rate	.09 $\pm$ 0	5.7 $\pm$ 0.6
$\mathbf{f_{13}}$. Children Spread	.08 $\pm$ 0	7.4 $\pm$ .66
$\mathbf{f_1}$. Diameter	.06 $\pm$ 0	8.5 $\pm$ 1.1
$\mathbf{f_{12}}$. Intra-children Time Spread	.06 $\pm$ 0	11.4 $\pm$ 1.1

Table 4: Gain ratio of top 10 features of influence graphs.

with low variability and few outliers [25]. These assumptions may not hold true for download graphs, for several reasons:

- *Fitting multiple malware types in a two-class model:* The subclasses of malicious and benign IGs have different properties (e.g., the average distinct file affinity, illustrated in Figure 4(e)). From the point of view of a classifier trying to label each example as benign or malicious, this means that the training data has high variability or many outliers. It is also difficult to establish how many subclasses of malware exist. Moreover, one subclass of benign downloaders may exhibit similar properties of that of a subclass of malicious downloaders (e.g., software components that update themselves), confusing the classifier even further.
- *Influence graph evolution:* Influence graphs evolve over time. Depending on when a downloader was dropped on different machines, the corresponding influence graphs may be different. For example, a malicious IG might initially have diameter 2, and the value of this feature might increase as the graph grows.
- *Cross-machine behavioral differences:* Even for similar ages, the influence graphs of a downloader may depend on the host environment and user behavior. In fact, downloaders may also encode host-specific download instructions [3].

Given these properties, the classes of malicious and benign influence graphs *may not be linearly separable* in the feature space, which raises a challenge for Support Vector Machine (SVM) classifiers. Additionally, we want our classifier to be scalable and parameter-free, as tuning parameters for such diverse data could be difficult. After experimenting with several algorithms, including SVM and k-nearest neighbor (kNN), we found that ensemble classification algorithms provide the best fit for our problem. We select a *random forest classifier* (RFC) [2], for the following reasons:

- *Bias-variance trade-off:* RFCs are known to provide a good bias-variance trade-off, especially if the data is diverse (i.e., noisy from the point of the view of a learning algorithm). An RFC consists of a set of *decision trees* (DT), where each DT is trained using only a subset of features (randomly chosen). The final class prediction is made by aggregating the votes from all the individual DTs. Thus, an RFC is less prone to overfitting as compared to a single DT. In other words, the *bias* of the entire forest is bounded by the bias of the individual DTs [2]. Moreover, since the class predication is made by aggregating the votes from a number of smaller decision trees, RFCs exhibit low variance as well. Intuitively, we can expect that different DTs will specialize on the different subclasses of the malicious/benign binaries and champion their cause in the final classification.
- *Parameter-free:* RFCs do not require parameter tuning[7] and are robust to outliers. While we need to specify the number of deci-

[7]In contrast, SVM requires selecting: (a) the kernel, (b) the kernel's parameters, (c) a soft margin parameter that encodes the penalty factor for non-separable points.

sion trees (N_t) used and the number of features (N_f) per decision tree, these parameters are independent of the nuances of the data and have standard selection rules[8]. Moreover, they exhibit monotonicity and diminishing returns, i.e., increasing them leads to an increase in accuracy but only to a certain limit.

- *Scalability:* Random forest classifiers are very fast to train, compared to SVM and kNN. Our experiments show that training an RFC on the same dataset is 10-15 times faster than SVM.

5.4 Internal Validation of the Classifier

In this section we evaluate the performance of our RFC classifier in three ways: (a) using 10-fold cross validation on the balanced training dataset, (b) using all the labeled data as a test set, and (c) computing the detection lead time, compared to the anti-virus products employed by VirusTotal.

10-fold Cross Validation. We choose $N_t = 40$ and $N_f = \log_2(\#\text{ of features}) + 1$, for our experiments. We observe that increasing N_t and N_f beyond these numbers results in very small improvement in accuracy; however, the training cost goes up significantly. We use the `RandomForestClassifier` module from Python's `scikit-learn` package for the experiment. We run a 10-fold cross validation on the balanced set. We use all the features described on Table 3 for training the classifier. We report the classification result at 1.0% FP rate, which achieves 96.0% TP rate[9].

These results with a default `scikit-learn` threshold are listed in the first row of Table 5, labeled "All Features".

Feature Evaluation. While in Table 4 lists the most useful features for our classifier, we also want to know how using different feature combinations would affect the performance of the classifier. Starting from only using FI features from Table 3, we combine other feature categories one by one (FL, FD, FU, then FA) and evaluate how the classification performance increases. The result is shown as a Receiver Operating Characteristic (ROC)[10] plot in Figure 5, where the X axis is the false positive rate and the Y axis is the true positive rate. ROC plots show the TP/FP trade-off: the top-left corner corresponds to a perfect classifier, which never makes mistakes. The curves for the different feature combinations are obtained by varying internal thresholds of the random forest classifier. We observe large jumps in performance at two points, first is when we add the FL features and them when we add the FA features. We believe the classifier is capturing the insights we discussed in Section 4.3, such as the slow growth rate and specific numbers of distinct downloaders accessing a domain for the malicious influence graphs. At FP rate 1%, the corresponding TP rates are 19.9%, 13.4%, 3%, 17.6%, 28.5% and 96.0%. Interestingly, only using FI rate was performing better than adding FL and FS at this point. This is consistent with our observation that almost 12% of the malicious IGs have diameter 3 and more and 84% of the IGs are malicious at diameter 3 and beyond (Section 4.3). At FP rate 3%, the numbers for TP rate are 24.7%, 33.5%, 42.1%, 59.6%, 98.7%. At 10%, TP rates are 41.7%, 64.5%, 71.2%, 84.8%, and 99.8%.

Evaluation of the False Positives. We perform a manual investigation of the misclassified IGs. We take the average of all the feature values, for the TP, TN, FP sets and compared the difference between <TP, FP> and <TN, FP>, to see which feature is closer to the TP set. The number of distinct files downloaded by a single URL, density, distribution of the number of incoming/outgoing edges, prevalence of the executables, and the children time spread features are showing closer value to the TP set. The IGs in the FP set have a smaller number of files downloaded per URL, denser structure, smaller number of outgoing edges, lower prevalence, and faster dropping rate. Most prevalent downloaders in this TP set turned out to be P2P downloaders and download managers.

[8] N_t is typically data size independent and few applications go beyond 100 trees. For N_f, standard value used is Log(Total number of features) + 1

[9] $TP_{rate} = \frac{TP}{TP+FN}$, $FP_{rate} = \frac{FP}{FP+TN}$, $F_1 = 2 * \frac{Precision*Recall}{Precision+Recall}$

[10] `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`

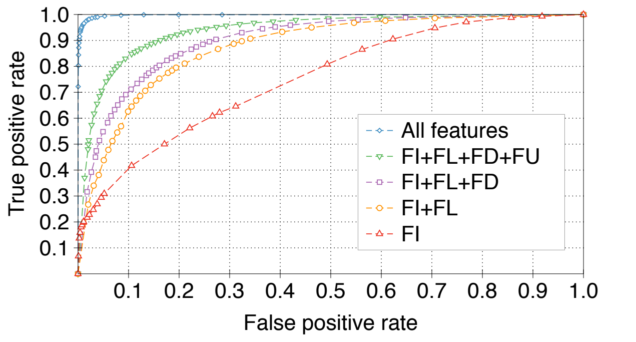

Figure 5: ROC curve for different feature groups

Algorithms	TP rate	FP rate	F-score	ROC Area
All Features	0.980	0.020	0.980	0.998
FI+FL+FD+FU	0.868	0.124	0.870	0.939
FI+FL+FD	0.831	0.184	0.823	0.902
FI+FL	0.811	0.211	0.801	0.876
Only FI	0.602	0.261	0.644	0.752

Table 5: Classifier performance on malicious class.

Testing the Classifier on the Entire Labeled Data. We apply the RFC trained on the balanced labeled data to the entire labeled data, which is skewed toward the benign class. There are 1,433,670 IGs in this set, where 43,668 are malicious and 1,390,092 are benign. The classification result showed 100% TP rate and 2% FP rate on the malicious class. From the ROC curve, we get 91.8% TP rate at FP rate 1%. We also got 99% G-mean[11]score [12] which is a standard way of measuring the classification accuracy on unbalanced data where both classes are important [26].

Early Detection. We also evaluate how early we can detect malicious executables that are previously unknown. We define "early detection" as "we are able to flag unknown executables as malicious before their first submission to VirusTotal". As discussed in Section 4.1, we approximate the date when malware samples becomes known to the security community using the VirusTotal first-submission time. We estimate our detection time in three ways: (a) an executable is detected at its earliest timestamp in the TP set (Lower bound), (b) an executable is detected at the timestamp when the last node was added to the newest influence graph it resides as a node (Upper bound) , and (c) the average timestamp of the executables in the TP set (Average).

We apply random forest with 10 fold cross validation on our balanced labelled data set using all 58 features. The outcome of 10 fold cross validation is TP=42,683, FP=822, FN=985 and TN=43,724, in terms of the number of influence graphs. For distinct executables in the TP set excluding the ones in the FN set, we compared the first seen timestamps in VirusTotal and our lower-bound/upper-bound/average detection timestamps. Among 31,104 distinct executables that are in TP set but not in FN set, 20,452 files had scanning records in VirusTotal. Among them, 17,462 executables had at least one detection in VirusTotal ($r_{mal} > 0$), and 10,323 executables had detection rates over 30% ($r_{mal} \geq 30$).

[11] $G - mean = \sqrt{TP \times TN}$

$r_{mal} > 0$	**Lower bound**	**Upper bound**	**Average**
Distinct Executables	6,515 (37.3%)	3,344 (19.2%)	4,871 (27.9%)
Early detection avg.	20.91	-23.73	9.24
$r_{mal} \geq 30$	**Lower bound**	**Upper bound**	**Average**
Distinct Executables	3,939 (38.2%)	2,041 (19.8%)	3,002 (29.1%)
Early detection avg.	35.86	-7.69	25.24

Table 6: Early detection.

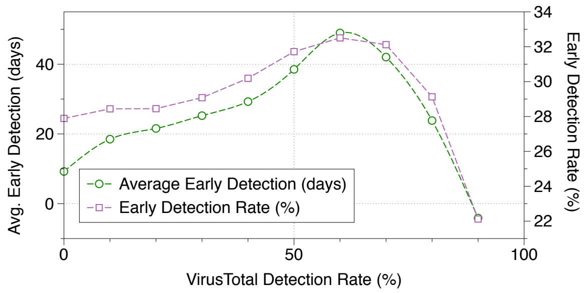

Figure 6: Detection rate vs. Early detection ratio / Early detection avg. (days)

Table 6 lists our results. For $r_{mal} > 0$, the time difference between the VirusTotal first seen timestamp and our lower bound detection timestamp is 20.91 days on average, and we are able to detect 6,515 executables earlier than VirusTotal. Our upper bound is 23.73 after VirusTotal, with 3,344 executables detected early. On average, we detect malware 9.24 days before the first VirusTotal detection. Interestingly, for $r_{mal} \geq 30$, we detect malware on average 25.24 days earlier than VirusTotal. We further investigate this trend in Figure 6, where we plot the portion of executables that we are able to detect early, in our average detection scenario, against the VirusTotal detection rate r_{mal}. Up to $r_{mal} = 80\%$, our early detection lead time increases with r_{mal}. This suggests that executables that are more likely to be malicious present stronger indications of maliciousness in their downloader graphs, allowing our classifier to detect them earlier than current anti-virus products.

5.5 External Validation of the Classifier

In this section, we evaluate the performance of our RFC classifier by drawing three random samples from the *unlabeled IGs* and querying the corresponding file hashes in VirusTotal. Out of the 580,210 unlabeled IGs, our classifier identifies 116,787 as malicious.

As discussed in Section 3.2, we were unable to query VirusTotal for all the file hashes, which leaves many of the leaf nodes in our graphs unlabeled. We draw three random samples, of approximately 3000 influence graphs each, from the set of unlabeled graphs and try to estimate the accuracy of our predictions by presenting the results of each set. We consider that an IG labeled as malicious is misclassified if none of the AV products in VirusTotal detect it as malicious. We consider that an IG labeled as benign is misclassified if its nodes were detected by more than 30% of the AV vendors, to account for the fact that AV products may also produce false positives in gray-area situations, such as benign executables that are sometimes involved in malware delivery.

Table 7 shows our results. On average 41.41% of the binaries that construct the IGs are known to be malicious also by other AV vendors, while only 0.53% of the binaries in benign IGs were labeled as malicious. Moreover, on average 78% of the IGs labeled as malicious have at least one internal node that AV products detect as malware, and only 1.58% of the IGs labeled as benign carry a malicious node. As many malware samples are discovered late by the security community (see Section 4.1), we expect that the number of samples considered malicious by VirusTotal will grow in the future.

#Run	Predictions	#IGs	$\#IG_{mal}$ by VT	$\#Binary_{mal}$ by VT
#1	Malicious	582	444 (76%)	1093(41%)
	Benign	2456	43 (1.7%)	60 (0.5%)
#2	Malicious	590	471 (80%)	1249 (43%)
	Benign	2454	30 (1.2%)	38 (0.3%)
#3	Malicious	592	466 (79%)	1041 (41%)
	Benign	2495	44 (1.7%)	67 (0.6%)

Table 7: Testing Classifier on the Unlabeled Influence Graphs.

5.6 Online Detection Experiment

Finally, we perform an experiment where we simulate the way our classifier would be employed operationally, for detecting malware in an online manner. We prepare the training set of 21,543 malicious and 21,755 benign IGs from the data before 2014. For the testing set, we build IGs based on the data from the year 2014. There are 12,299 malicious and 12,594 benign IGs for the test set. Since we are trying online detection, we assume here that the data from 2014 may not had enough time to collect the prevalence of the executables, so we disregard the prevalence features for the classification. Note that prevalence is one of the top features that perform well in the classification. We train the RFC on the training set, with the same parameters used on section 5.4. Then we apply the trained classifier on the testing set. We obtain 99.8% TP rate, 1.9% FP rate and 0.2% FN rate, with 99.0% F1-score. This suggests that our features do not exhibit a significant *concept drift* over time and show the potential for using our classifier as a robust online detection technique.

6. DISCUSSION

We now discuss the lessons learned from our experiments, focusing on the implications for malware detection and for attack attribution.

Opportunity for Improving Malware Detection. Our research represents a first step towards understanding the properties of downloader graphs in the wild. We demonstrate that some of these properties, such as the large graph diameter, high growth interval of the influence graphs, large number of distinct droppers accessing a domain, represent strong indication of malicious activities. This insight can lead to *deterministic* detection techniques that help existing anti-virus products to block certain classes of malware in early phases of their lifecycle. The intuition behind these benefits is that the properties of downloader graphs can provide evidence of malicious activity before new malware samples can be investigated by the security community. Moreover, our results highlight the benefits of incorporating downloader graph features into *probabilistic* detection techniques along with existing host-based and network-based features. The information extracted from downloader graphs complements the existing approaches as it (a) captures the client-side activity of malware delivery networks, (b) reflects the relationships among malware families, and (c) helps increase the detection performance and reduce the detection latency. We note that our techniques operate on end hosts. Although the analysis of network traffic may indicate when downloading is in progress [10, 30], it cannot determine which executable triggers the download; thus it may not be possible to construct a complete downloader graph.

Blocking Malicious Droppers. Our results raise the question, *Should we require user approval for all downloads of executable pro-*

grams? If an operating system, or an anti-virus program, quarantines every executable downloaded and presents a user with a dialog to ask for the user's approval, the operation of malicious downloaders would be severely impaired. Unfortunately, this approach would also harm security by inhibiting the deployment of security patches, as manual or semi-automated software updating approaches are considerably less effective than silent updates which download and install security patches without user interactions [6, 16]. A more effective approach would be to require digital signatures for programs that download other executables, as is currently done for device drivers [15]. This approach would also be more efficient than the attempts to whitelist all benign software [24], because only a few benign programs (87,906 in our data set) download other executables.

Implications for Attack Attribution. Attack attribution to identify attackers is generally considered as a hard problem because attackers may employ various methods to conceal their identities, e.g., obfuscating or re-packing binaries, changing the URLs and domains of servers, launching attacks from geographically-distributed compromised machines. However, 55.5% of malicious downloaders are signed, accounting for 73.8% of the malicious influence graphs. For example, one PPI provider consistently specifies "Amonetize LTD" in the publisher field of the digital signature. Attackers may distribute signed programs to avoid raising suspicion and to remain undetected for possibly longer periods of time. This suggests that, for the miscreants operating malware dissemination networks, the benefits of being able to remain stealthy for a while outweigh the benefits of thwarting attribution efforts. Similarly, attackers may transmit malicious payloads over the HTTP protocol, which is one of the most common traffic crossing organizations network perimeters, to go around firewall restrictions. In fact, this has facilitated several efforts to profile the organizations involved in malware distribution [3, 8, 17]. Our detection approach based on the downloader graph analytics may force attackers to employ stealthier techniques to download malicious payloads, such as utilizing custom ports and protocols, but these techniques are more likely to raise alarms in firewalls and intrusion-prevention systems.

Limitation. Droppers with rootkit functionality would evade our technique. However, rootkits are typically employed to hide more incriminating functionality, while our technique relies mainly on the ability to track downloader-payload relationships. This provides a new signal, complementary to current AV engines, and our experiments suggest that it can detect malware that are not currently being detected by existing AVs.

7. RELATED WORK

Several research efforts have focused on characterizing the properties of malware delivery networks and on taking advantage of these properties for detecting malware. Provos et al. [20] described drive-by-download attacks caused by exploiting vulnerabilities in web browsers, analyzed the tree-like structure of redirection paths leading to the main malware distribution sites, and identified several mechanisms to inject malicious web content into popular pages (e.g., comments on blogs or syndication in Ad serving networks). Li et al. [14] further analyzed URL redirection graphs and identified a set of topologically dedicated malicious hosts (e.g., Traffic Distribution Systems) that are long-lived and receive traffic from new attack campaigns over time. Perdisci et al. [19] analyzed the structural similarities among malicious HTTP traffic traces which include a variety of activities, such as receiving commands from C&C servers, sending spam, exfiltrating private data, and downloading updates. Xu et al. [31] fingerprinted several types of malicious servers, including exploit servers (for malware distribution through drive-by downloads), C&C servers (for command and control), redirection servers (anonymity), and payment servers (for monetization). Inspired by these studies, several techniques were proposed for detecting malware download events from network traffic [10, 30]. Unlike our work, these studies focus on the server side of malware distribution networks, and they are unable to identify which program triggered the download, and cannot be used to reconstruct complete downloader graphs.

Prior research on the behavior of downloaders [3, 17, 23] focused on executing malware droppers in *lab environments*, typically for short periods of time (e.g., up to one hour), in order to observe the communication protocols they employ and to *milk* their server-side infrastructures (i.e., to download the payloads). For example, Caballero et al. [3] described pay-per-install infrastructures, which distributed malware on behalf of their affiliates, and analyzed their structure and business model. They also provided an example of a download tree, which reached diameter 4 within 10 minutes. Follow-up work reported that downloaders might remain active for over 2 years [23], and characterized several malware distribution operations [17]. In contrast, we analyze malicious droppers that remain undetected for 80.6 days on average *in the wild*. We compare the properties of malicious and benign downloader graphs, in order to assess their real impact on end-user security and the potential benefits of downloader graph analytics. This approach allows us to determine that some features, such as high graph diameter, slow growth rate of the influence graphs, large number of distinct portal accessing a domain are strong indication of malicious behavior, and to train a generic classifier for malware detection using information extracted from downloader graphs.

Closest to our work are recent techniques for assigning *reputation* scores to executable files by performing belief propagation on bipartite graphs [5, 28]. Chau et al. [5] constructed a graph that encoded the relationship between files and the hosts they are present on, building on the intuition that hosts with poor cyber-hygiene tended to contain more malware. Tamersoy et al. [28] constructed a graph that encoded the relationship between different files that were present on the same host, building on the intuition that several malware samples were often distributed together (guilt-by-association). In our work, we construct graphs that encode a semantic relationship between files—the fact that one file downloads another file—which can provide deeper insights into malware distribution activities.

There is a rich literature on graph mining techniques [4]. We cite here the Oddball approach [1], which consists of extracting features from the k-hop neighborhoods of every node, identifying patterns for these features and analyzing the outliers that do not seem to conform to these patterns, for its similarity with the technique we employ in Section 4. Graph analytics have also been employed for analyzing function call graphs in malware samples [9, 11, 33], spamming operations [32, 34], and vote gaming attacks [22]. In contrast to these approaches, we propose downloader graph analytics for analyzing the malware delivery activities on the client side.

8. CONCLUSIONS

We analyze downloader graphs in the wild and uncover the differences in growth patterns between benign and malicious graphs. Because downloader graphs capture the relationships between downloaders and the supplementary executables they download, we can identify large parts of the malware download activity on each host by analyzing the upstream download chain. We identify deterministic techniques for detecting certain classes of malware based on properties of the downloader graphs, including their growth rate, diameter and the diversity in the set of Internet domains accessed.

We also describe a generic malware detection system, which uses machine learning techniques for automatically learning models of malicious download graphs. We evaluate our system on 19 million graphs and show that it detects malware with high accuracy and earlier than existing anti-virus systems.

Acknowledgments

We thank Amol Deshpande, Jonathan Katz, and Michel Cukier for early feedback. We also thank VirusTotal for access to their service and Symantec for access to WINE (the data analyzed in this paper corresponds to the reference data set WINE-2015-002). This research was partially supported by the Maryland Procurement Office (contract H98230-14-C-0127) and by the Department of Defense.

9. REFERENCES

[1] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *KDD*. 2010.

[2] L. Breiman. Random forests. 2001.

[3] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *USENIX Security Symposium*, 2011.

[4] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys (CSUR)*, 2006.

[5] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *SDM*, 2011.

[6] T. Dübendorfer and S. Frei. Web browser security update effectiveness. In *CRITIS Workshop*, September 2009.

[7] T. Dumitraș and D. Shou. Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In *EuroSys BADGERS Workshop*, Salzburg, Austria, 2011.

[8] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *CCS*, 2012.

[9] X. Hu, T. Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *CCS*, 2009.

[10] L. Invernizzi, S.-J. Lee, S. Miskovic, M. Mellia, R. Torres, C. Kruegel, S. Saha, and G. Vigna. Nazca: Detecting Malware Distribution in Large-Scale Networks. In *NDSS*, 2014.

[11] D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification. In *KDD*, 2013.

[12] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine learning*, 1998.

[13] N. Leontiadis, T. Moore, and N. Christin. A Nearly Four-Year Longitudinal Study of Search-Engine Poisoning. In *CCS*, 2014.

[14] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang. Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In *IEEE S&P*, 2013.

[15] Microsoft. Driver signing policy. `https://msdn.microsoft.com/en-us/library/windows/hardware/ff548231(v=vs.85).aspx`.

[16] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitraș. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *IEEE S&P*, San Jose, CA, 2015.

[17] A. Nappa, M. Z. Rafique, and J. Caballero. The MALICIA dataset: identification and analysis of drive-by download operations. *IJIS*, 2015.

[18] A. Nappa, Z. Xu, M. Z. Rafique, J. Caballero, and G. Gu. Cyberprobe: Towards internet-scale active detection of malicious servers. In *NDSS*. The Internet Society, 2014.

[19] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, 2010.

[20] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *USENIX Security Symposium*, 2008.

[21] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1986.

[22] A. Ramachandran, A. Dasgupta, N. Feamster, and K. Weinberger. Spam or ham?: characterizing and detecting fraudulent not spam reports in web mail systems. In *CEAS*, 2011.

[23] C. Rossow, C. Dietrich, and H. Bos. Large-scale analysis of malware downloaders. In *DIVMA*. 2013.

[24] A. Sedgewick, M. Souppaya, and K. Scarfone. Guide to application whitelisting. Technical Report Special Publication 800-167 (Draft), National Institute of Standards and Technology, 2014.

[25] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE S&P*, 2010.

[26] Y. Sun, A. K. Wong, and M. S. Kamel. Classification of imbalanced data: A review. *IJPRAI*, 2009.

[27] Symantec. W32.Sobig.F. `http://www.symantec.com/security_response/writeup.jsp?docid=2003-081909-2118-99`, 2003.

[28] A. Tamersoy, K. Roundy, and D. H. Chau. Guilt by association: large scale malware detection by mining file-relation graphs. In *KDD*, 2014.

[29] G. Tenebro. The Bredolab Files. Symantec Whitepaper. `http://securityresponse.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_bredolab_files.pdf`, 2009.

[30] P. Vadrevu, B. Rahbarinia, R. Perdisci, K. Li, and M. Antonakakis. Measuring and Detecting Malware Downloads in Live Network Traffic. In *ESORICS*, 2013.

[31] Z. Xu, A. Nappa, R. Baykov, G. Yang, J. Caballero, and G. Gu. AUTOPROBE: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. In *CCS*, 2014.

[32] C. Yang, R. C. Harkreader, and G. Gu. Die Free or Live Hard? Empirical Evaluation and New Design for Fighting Evolving Twitter Spammers. In *RAID*, 2011.

[33] M. Zhang, Y. Duan, H. Yin, and Z. Zhao. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *CCS*, 2014.

[34] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: Large Scale Spamming Botnet Detection. In *NSDI*, 2009.

[35] C. C. Zou and R. Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *DSN*, 2006.