WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths

Terry Nelms, *Damballa, Inc. and Georgia Institute of Technology;* Roberto Perdisci, *University of Georgia and Georgia Institute of Technology;* Manos Antonakakis, *Georgia Institute of Technology;* Mustaque Ahamad, *Georgia Institute of Technology and New York University Abu Dhabi*

https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/nelms

This paper is included in the Proceedings of the
24th USENIX Security Symposium

August 12–14, 2015 • Washington, D.C.

ISBN 978-1-931971-232

WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths

Terry Nelms[1,2], Roberto Perdisci[3,2], Manos Antonakakis[2], and Mustaque Ahamad[2,4]

[1]Damballa, Inc.
[2]Georgia Institute of Technology
[3]University of Georgia
[4]New York University Abu Dhabi

tnelms@damballa.com, perdisci@cs.uga.edu, manos@gatech.edu, mustaq@cc.gatech.edu

Abstract

Most modern malware download attacks occur via the browser, typically due to social engineering and drive-by downloads. In this paper, we study the "origin" of malware download attacks experienced by real network users, with the objective of improving malware download defenses. Specifically, we study the *web paths* followed by users who eventually fall victim to different types of malware downloads. To this end, we propose a novel *incident investigation system*, named WebWitness. Our system targets two main goals: 1) automatically *trace back and label* the sequence of events (e.g., visited web pages) preceding malware downloads, to highlight how users reach attack pages on the web; and 2) leverage these automatically labeled in-the-wild malware download paths to better understand current attack trends, and to *develop more effective defenses*.

We deployed WebWitness on a large academic network for a period of ten months, where we collected and categorized thousands of *live* malicious download paths. An analysis of this labeled data allowed us to design a new defense against drive-by downloads that rely on injecting malicious content into (hacked) legitimate web pages. For example, we show that by leveraging the incident investigation information output by WebWitness we can decrease the infection rate for this type of drive-by downloads by almost *six times*, on average, compared to existing URL blacklisting approaches.

1 Introduction

Remote malware downloads currently represent the most common infection vector. In particular, the vast majority of malware downloads occur via the browser, typically due to social engineering attacks and drive-by downloads. A large body of work exists on detecting drive-by downloads (e.g., [10, 11, 19, 23, 33, 40]), and a few efforts have been dedicated to studying social engineering attacks [6, 31, 37]. However, very little attention has been dedicated to investigating and categorizing the *web browsing paths* followed by users *before* they reach the web pages from which the attacks start to unfold.

Our Work. In this paper, we study the *web paths* followed by *real users* that become victims of different types of malware downloads, including social engineering and drive-by downloads. We have two primary goals: 1) provide context to the attack by automatically identifying and labeling the sequence of web pages visited by the user *prior to the attack*, giving insight into how users reach attack pages on the web; and 2) leverage these annotated in-the-wild malware download paths to better understand current attack trends and to *develop more effective defenses*.

To achieve these goals we propose a novel malware download *incident investigation system*, named WebWitness, that is designed to be deployed passively on enterprise scale networks. As shown in Figure 1, our system consists of two main components: an *attack path traceback and categorization* (ATC) module and a *malware download defense* (MDD) module. Given all (live) network traffic generated by a user's browsing activities within a time window that includes a malware download event, the ATC module is responsible for identifying and linking together all HTTP requests and responses that constitute the web path followed by the user from an "origin" node (e.g., a search engine) to the actual malware download page, while filtering out all other irrelevant traffic. Afterwards, a statistical classifier automatically divides all collected malware download paths into *update*, *social engineering* and *drive-by* attacks. We refer to the output of the ATC module as *annotated malware download paths* (AMP).

The AMPs are continuously updated as new malware downloads are witnessed in the live traffic, and can therefore be used to aid the study of recent attack trends. Furthermore, the AMP data is instrumental in designing and building new defenses that can be plugged into the MDD

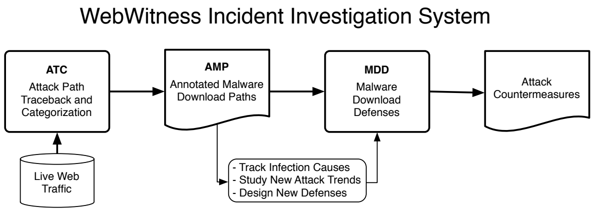
WebWitness Incident Investigation System
ATC
Attack Path Traceback and Categorization
AMP
Annotated Malware Download Paths
MDD
Malware Download Defenses
Attack Countermeasures
Live Web Traffic
- Track Infection Causes
- Study New Attack Trends
- Design New Defenses

Figure 1: WebWitness – high-level system overview.

module (see Figure 1). As an example, by investigating real-world web paths leading to drive-by malware downloads, we found that it is often possible to automatically trace back the domain names typically used in drive-by attacks to inject malicious code into compromised web pages (e.g., via the source of a malicious `script` or `iframe` tag). The injected code is normally used as an attack trigger, directing the browser towards an actual exploit and finally to a "transparent" malware download and execution. We empirically show that automatically discovering and promptly blocking the domain names serving the injected malicious code is a much more effective defense, compared to the more common approach of blacklisting the URLs that directly serve the drive-by browser exploits themselves or the actual malware executables (see Section 4.4).

Main Differences from Previous Work. Most previous works that study the network aspects of malware downloads focus on building malware detection systems, especially for drive-by exploit kits and related attacks (e.g., [13,30,38,40]).

Our work is different from these studies, because our goal is *not* to build a drive-by detection system; rather, we aim to passively trace back and automatically label the network events that *precede* different types of in-the-wild malware downloads, including both drive-by and social engineering attacks. We show that our investigation approach can aid in the design of more effective malware download defenses.

Some recent studies focus primarily on detecting malicious redirection chains as a way to identify possible malware download events [16,18,20,36]. WebWitness is different because we devise a *generic path trace back* approach that does not rely on the properties of redirection chains. Our work aims to provide *context* around malicious downloads by reconstructing the *full web path* (not just redirection chains) that brought the victim from an "origin" page to the download event. In addition WebWitness is able to classify the *cause* of the download (e.g., drive-by or social engineering) and to identify the roles of the domains involved in the attack (e.g., trick page, code injection, exploit, or malware hosting). We further discuss related work in Section 6.

Summary of Contributions. In summary, we make the following contributions:

- We investigate the *web paths* followed by *real network users* who eventually fall victim to different types of malware downloads, including social engineering and drive-by downloads. Through this investigation, we provide quantitative information on attack scenarios that have been previously explained only anecdotally or through limited case studies.
- To enable a continuous collection and study of web paths leading to malware download attacks, we build a system called WebWitness. Our system can automatically trace back and categorize in-the-wild malware downloads. We show that this information can then be leveraged to design more effective defenses against future malware download attacks.
- We deployed WebWitness on a large academic network for a period of ten months, where we collected and categorized thousands of *live* malicious download paths. Using these web paths, we were able to design a new defense against drive-by downloads that rely on injecting malicious content into (hacked) legitimate web pages. For example, we show that by leveraging the incident investigation information output by WebWitness, on average we can decrease the infection rate for this type of drive-by downloads by almost *six times*, compared to existing URL blacklisting approaches.

2 In-The-Wild Malware Download Study

Goals: In this section we report the results of a large study of *in-the-wild* malware downloads captured on a live academic network. Through this study, we aim to create a labeled dataset of download paths that can be used to design (including feature engineering), train, and evaluate the ATC and MDD modules of WebWitness shown in Figure 1. A detailed discussion of ATC and MDD is reported in Sections 3.

2.1 Collecting Executable File Downloads

To collect executable file downloads we use deep packet inspection to perform on-the-fly TCP flow reconstruction, keeping a buffer of all recent HTTP transactions (i.e., request-response pairs) observed on a live network. For each transaction, we check the content of the response to determine if it contains an executable file. If so, we retrieve all buffered HTTP transactions related to the client that initiated the download. Namely, we store all HTTP traffic a client generated preceding (and including) an executable file download; this allows us to study what *web path* users follow *before* falling victim to malware downloads. All data is saved in accordance with the policies set forth by our Institutional Review Board and are protected under a nondisclosure agreement.

2.2 Identifying Malicious Executables

Since many legitimate applications are installed or updated via HTTP (e.g., Windows Update), we immediately exclude all executable downloads from a manually-compiled whitelist of domain names consisting of approximately 120 effective second level domains (e2LDs) of popular benign sites (e.g., `microsoft.com`, `google.com`, etc.). For the remaining downloads, we scan them with more than 40 antivirus (AV) engines, using `virustotal.com`. In addition, we rescan them periodically because many "fresh" malware files are not immediately detected by AV scanners, allowing us to also take into account some "zero-day" downloads. We label a file as malicious if at least one of the top five AV vendors (w.r.t. market share) and a minimum of two other AVs detect it as malicious. The remaining downloads are considered benign until the rescan. In addition, we discard binary samples that are assigned labels that are too generic or based purely on AV detection heuristics.

2.3 Overview of Study Data

To gather our study data we deployed our collection agent (Section 2.1) on a large academic network serving tens of thousands of users for a period of 6 months. Notice that the system was deployed for a total of 10 months, with the study conducted in the first 6 months and the evaluation in the 4 months that followed (see Section 4 details on the evaluation). During these 6 months, we collected a total of $174,376$ executable downloads from domains that were not on our whitelist. Using the malicious executable identification process defined in Section 2.2, we labeled $5,536$ downloads as malicious.

However, many of these malicious downloads were related to *adware*. As we are primarily interested in studying *malware* downloads, because they are potentially the most damaging ones, we devised a number of "best effort" heuristics to separate *adware* from *malware*. For example, given a malicious file, if the majority of AV labels contain the term "adware", or related empirically derived keywords that identify specific unwanted applications (e.g., "not-a-virus", "installer", "PUP", etc.), we label the file as *adware*. The malicious executables not labeled as *adware* by our heuristics were manually reviewed to determine if they were truly *malware*. This resulted in $1,064$ *malware* downloads, with a total of 533 unique samples.

For these 533 unique malware downloads, we performed extensive manual analysis of their download paths, including reverse engineering web pages, heavy javascript deobfuscation, complex plugin content analysis, etc. This time-consuming analysis produced a set of labeled paths, with 164 drive-by, 41 social engineering and 328 update/drop malware download events.

Study Data Limitations: Our collection agent was deployed on an existing production network monitoring sensor. This sensor had limited hardware resources; in addition, our data collection system had to run alongside production software whose functionality could not be disrupted. We therefore collected downloads only during off-peak hours, due to traffic volumes that would oversubscribe the sensor and result in dropped packets during other periods of the day. Thus, the malicious downloads in our study represent only a sample of the ones that occurred during the six month monitoring period. In addition, our system monitors the network in a *purely passive* way; therefore, any malicious downloads preemptively blocked by existing defenses (e.g., URL blacklists such as Google Safe Browsing) were not observed. Yet, based on our extensive manual analysis, we believe the 533 malware downloads to be sufficiently diverse and representative of the overall set of malware downloads that occurred during our study period.

2.4 Download Path Traceback Challenges

One of the goals of our system is to automatically trace back the sequence of steps (i.e., HTTP transactions) that lead victims to be infected via a malware download. One may think that reconstructing the *web path to infection* is fairly easy, because we could rely on the `Referer` and `Location` header fields to link subsequent HTTP transactions together (see RFC2616). For example, a simple strategy would be to start from the download transaction and "walk back" the sequence of transactions by following the `Referer` header found in the HTTP requests.

Unfortunately, in practice download path traceback is much more difficult than it may seem at first. Depending on the particular version of the browser, JavaScript engine, and plugin software running on the client, the `Referer` and/or `Location` headers may be suppressed (e.g., see [14]), resulting in the inability to correctly reconstruct the entire sequence of download path transactions in a given network trace.

Deriving and Measuring Surrogate Features: As part of our study, we reviewed hundreds of malicious download traces. In most cases we cannot rely completely on the `Referer` and `Location` headers, and we therefore derive surrogate "referrer indicator" features and heuristics, which can be used to perform a more complete download path traceback. Next, we define each of the features we observed, and then provide a measure of how prevalent they are for malware download paths. While in this section we simply *measure their prevalence*, we later use these features to automate path traceback (Section 3).

First, let us more precisely define what we mean with *download path traceback*. Let T_d indicate an HTTP transaction carrying an executable file download initiated by client C. Given the recording of all web traffic generated by C during a time window preceding (and including) T_d, we would like to reconstruct the sequence of transactions $(T_1, T_2, \ldots, T_d)$ that led to the download, while filtering out all unrelated traffic. This sequence of transactions may be the consequence of both explicit user interactions (e.g., a click on a link) and actions taken by the browser during rendering (e.g., following a page redirection). Notice that the traffic trace we are given may contain a large number of transactions that are completely unrelated to the download path, simply because the user may have multiple browser tabs open and multiple web-based applications active in parallel. Thus, potentially producing a large amount of overlapping unrelated traffic.

Let T_1 and T_2 be two HTTP transactions. We found that the features/heuristics listed below can be used to determine whether T_1 is a *likely source* of T_2, therefore allowing us to "link" them with different levels of confidence. Table 1 summarizes the prevalence of each feature in both drive-by and social engineering downloads (we discuss how we can distinguish drive-by from social engineering later in Section 2.5). A detailed discussion of how WebWitness uses these features for automated download path traceback is given in Section 3.

(1) **Location**: According to RFC2616, if transaction T_2's URL matches T_1's `Location` header, it indicates that T_2 was reached as a consequence of a server redirection from T_1.

(2) **Referrer**: Similarly, if T_1's URL matches T_2's `Referer` header, this indicates that the request for T_2 originated (either directly or through a redirection chain) from T_1, for example as a consequence of page rendering, a click on a hyperlink, etc.

(3) **Domain-in-URL**: We observed that advertisement URLs often embed the URL of the page that displayed the ad. So, if T_1's domain name is "embedded" in T_2's URL, it is likely that T_1 was the "source" of the request, even though the `Referer` is not present. This is especially true if there is only a small time gap between the transactions.

(4) **URL-in-Content**: If T_1's response content includes T_2's URL (e.g., within an HTML or non-obfuscated JavaScript code), this indicates there is (potentially) a "source of" relationship that links T_1 to T_2.

(5) **Same-Domain**: By investigating numerous drive-by malware downloads, we found that in many cases the exploit code and the malware executable file itself are served from the same domain. This approach is likely chosen by the attackers because if the exploit is successfully served, it means that the related malicious domain is currently reachable and serving the malware file from the same domain helps guarantee a successful infection (a similar observation was made in [13]). Therefore, if T_1 and T_2 share the same domain name and are temporally close, this likely indicates that T_1 is the "source of" T_2.

(6) **Commonly Exploitable Content (CEC)**: In our observations, most drive-by downloads use "commonly exploitable" content (e.g., .jar, .swf, or .pdf files that carry an exploit) to compromise their victims. The exploit downloads the malicious executable; thus, if T_1 contains commonly exploitable content (CEC) and T_2 is an executable download that occurred within a small time delta after T_1, this indicates that T_1 may be the "source of" T_2.

(7) **Ad-to-Ad**: In some cases, we observed chains of ad-related transactions where the `Referer` and `Location` header are missing (e.g., due to JavaScript or plugin-driven redirections). Therefore, if T_1 and T_2 are consecutive ad-related requests (e.g., identified by matching their URLs against a large list of known ad-distribution sites) and were issued within a small time delta, this indicates there may be a "source of" relationship.

Table 1: Success rate of traceback method and "Source-of" relationships in malware download paths. The numbers indicate the percentage of analyzed download paths.

Traceback method success rate	Drive-by	Social Eng.
Only Referrer and Location	0%	53%
All surrogate referrer features	96%	95%

Feature	Drive-by	Social Eng.
Location	69%	73%
Referrer	97%	100%
Domain-in-URL	0%	5%
URL-in-Content	17%	17%
Same-Domain	97%	20%
CEC	5%	0%
Ad-to-Ad	6%	10%

As a confirmation to the fact that tracing back malware download paths is challenging, we found that not a single drive-by download in our dataset could be traced back by relying only on the `Referer` and `Location` headers. For example, even if 97% of the drive-by download paths contained at least one pair of requests linked via the `Referer`, all drive-by paths contained at least some subsequence of the path's transactions that could not be "linked" by simply using the `Referer` header.

For social engineering paths, we found that 53% of the downloads could be traced back using only the `Referer` and `Location` headers. When this was not possible, the main cause was the presence of requests made via JavaScript and browser plugins. In some cases, we were not able to fully trace back the download path. The cause for the majority of the untraceable drive-by (4%) and social engineering (5%) downloads, when using all the features, was missing transactions likely due to our system not observing all related packets.

2.5 Drive-by vs. Social Engineering

We label a malware download path as *social engineering* if explicit user interaction (e.g., a mouse click) is required to initiate a malware download. In contrast, we label as *drive-by* those malware downloads that are transparently delivered to the victim via a browser exploit. As mentioned earlier (Section 2.3), during our study, we were able to manually review and label 164 drive-by and 41 social engineering malware downloads.

What distinguishes drive-by from social-engineering: In the following we report the characteristics that we observed for different types of paths. In particular, some of these characteristics could be leveraged as statistical features to build a classifier that automatically distinguishes between drive-by and social engineering downloads (see Section 3). We also discuss characteristics of malware updates/drops that could be used to filter out download paths that belong neither to the drive-by nor to the social-engineering class. Table 2 summarizes the prevalence of each of the characteristics described below.

Table 2: Download path properties.

Feature	Drive-by	Social eng.
Candidate Exploit Domain Age	0	-
Drive-by URL Similarity	69%	0%
Download Domain Recurrence	0.6%	34%
Download Referrer	0.6%	95%
Download Path Length	6	7
User-Agent Popularity	95%	98%

(1) **Candidate Exploit Domain "Age":** Drive-by download attacks often exploit their victims by delivering exploits via files of popular content types such as `.jar`, `.swf`, or `.pdf` files; we simply refer to these file types as "commonly exploitable" content (CEC). For example, during our study, we found that 94% of the drive-by download paths at some point delivered the exploit via CEC. The domains serving these exploits tend to be short-lived compared to domains serving benign content of the same type. Therefore, CEC served from a recently registered domain is an indicator of a possible drive-by download path. On the other hand, none of the social engineering download paths we observed during our study had this property. Table 2 reports the median domain name "age", computed as the number of days of activities for the domain of a page serving CEC, measured over a very large passive DNS database. The median age is less than one day for drive-by paths, and is not indicated for social engineering paths, because none of the nodes in the social engineering path served content of the type we consider as CEC (the overall traffic traces included HTTP transactions that carried content such as `.swf` files, but none of those were on the download path).

(2) **Drive-by URL Similarity:** The majority of drive-by downloads (about 70% of our observations) are served by a small number of exploit kits. Therefore, in many cases the exploit delivery URLs included in drive-by download paths share a structural URL similarity to known exploit kit URLs. Table 2 reports the fraction of drive-by download paths that had a similarity to known exploit kit URLs greater than 0.8, measured using the approach proposed in [26].

(3) **Download Domain Recurrence:** Most domains serving drive-by and social engineering malware download are contacted rarely, and often only once by one particular client at the time of the attack. On the other hand, malicious software regularly checks for executable updates. To approximately capture this intuition, we measured the number of queries to the malware download domain. As shown in Table 2, only 0.6% of the malware download domains in our drive-by paths are queried mulitple times within a small time window (two days, in our measurements). The higher percentage of social engineering malware paths with download domain recurrence is due to the fact that a significant fraction of the ones we observed used a free file sharing website for the malware download and that we count the domain query occurrences in aggregate, rather than per client.

(4) **Download Referrer:** In case of social engineering attacks, the HTTP transaction that delivers the malicious file download tends to carry a `Referer`, usually due to the direct user interaction that characterizes them. On the other hand, drive-by attack malware file delivery happens via a browser exploit. The request initiated from the shell code typically does not have a `Referer` header. Similarly, malware updates/drops initiated by malicious applications are already running on a compromised machine, and usually do not carry any referrer information. Table 2 shows that only 0.6% of all drive-by paths, in contrasts to 95% of social engineering paths, carried a `Referer` in the download node.

(5) **Download Path Length:** Drive-by and social engineering attacks typically generate download paths consisting of several nodes, mainly because a user has to first browse to a site that eventually leads to the actual attack. In addition, the malware distribution infrastructure is often built in such ways that enables malware downloads "as a service", which entails the use of a number of "redirection" steps. In contrast, download paths related to malware updates or drops tend to be very short. Table 2 reports the median number of nodes for drive-by and social engineering paths. In case of malware updates/drops, the median length for the path was only one node.

(6) **User-Agent Popularity:** The download paths for both drive-by and social engineering downloads typically include several nodes that report a popular browser user-agent string, as the victims use their browser to reach the attack. On the other hand, in most cases of a malware drop/update, it is not the browser, but the update software making the requests. In practice, we observed that the majority of malware update download paths did not report a popular user-agent string (only 36% of them did). Table 2 reports the percentage of paths that include a popular user-agent string.

3 WebWitness

Inspired by our study of real-world malware download paths, we develop a system called WebWitness that can automate the investigation of new malware download attacks. The primary goal of this system is to provide *context* around malicious executable downloads. To this end, given a traffic trace that includes all web traffic recorded during a time window preceding (and including) a malicious executable file download, WebWitness automatically traces back and categorizes the web paths that led the victim to the malicious download event.

In this section, we describe the components of our system, which are shown in Figure 2.

3.1 ATC - Download Path Traceback

Given a malicious file download trace from a given client, WebWitness aims to trace back the *download path* consisting of the sequence of web pages visited by the user that led her to a malware download attack (e.g., via social engineering or to a drive-by exploit). As detailed in Section 2.4, the trace may contain many HTTP transactions that are unrelated to the download. Furthermore, it is not always possible to correctly link two related consecutive HTTP transactions by simply leveraging their HTTP `Referer` or `Location` headers.

To mitigate the limitations of referrer-only approaches and more accurately trace back the download path, we devise an algorithm that leverages the features and heuristics we identified during our initial study of in-the-wild malware downloads presented in Section 2.4. In summary, we build a *transactions graph*, where nodes are HTTP transactions within the download trace, and edges connect transaction according to a "probable source of" relationship (explained in detail below). Then, starting from the node (i.e., the HTTP transaction) related to the malware file download, we walk back along the most probable edges until we find a node with no predecessor, which we label as the "origin" of the download path. In the following, we provide more details on our traceback algorithm.

Transactions Graph. Let D be the dataset of HTTP traffic generated by host A before (and including) the download event. We start by considering all HTTP transactions in D, and construct a weighted directed graph $G = (V,E)$. The vertices are A's HTTP transactions and the edges represent the relation "probable source of" for pairs of HTTP transactions. As an example, the edge $e = (v_1 \rightarrow v_2)$ implies that HTTP transaction v_1 likely produced HTTP transaction v_2, either automatically (e.g., via a server-imposed redirection, javascript, etc.) or through explicit user interaction (e.g., via a hyperlink click). Thus, we can consider v_1 as the "source of" v_2. Each edge has a weight that expresses the level of confidence we have on the "link" between two nodes (the weights are ordinal so their absolute values are not important). For example, the higher the weight assigned to $e = (v_1 \rightarrow v_2)$, the stronger the available evidence in support of the conclusion that v_1 is the "source of" v_2 (edge weights are further discussed below). Also, let t_1 and t_2 be the timestamp of v_1 and v_2, respectively. Regardless of any available evidence for a possible edge, the two nodes may be linked only if $t_1 \leq t_2$.

Heuristics and Edge Weights. To build the graph G and draw its edges, we leverage the seven features that we indentified in Section 2.4. Specifically, given two nodes (essentially, two URLs) in the directed graph G described earlier, an edge $e = (v_1 \rightarrow v_2)$ is created if any of the seven features is satisfied. For example, if v_1 and v_2 can be related via the "Domain-in-URL", we draw an edge between the two nodes. We associate a weight to each of the seven features; the "stronger" the feature, the higher its weight. For example, we assign a weight value $w_e = 7$ to the "Location" feature, $w_e = 6$ to the "Referrer" feature, and so on, with the "Ad-to-Ad" receiving a weight $w_e = 1$. The weight values are conveniently assigned simply to express relative importance and precedence among the edges to be considered by our greedy algorithm. If more than one feature happens to link two nodes, the edge will be assigned a weight equal to the maximum weight among the matching features.

Traceback Algorithm. Once G has been built, we use a *greedy algorithm* to construct an approximate "backtrace path". We start from the graph node related to the executable download event, and walk backwards on the graph by always choosing the next edge with the highest weight. Consider the example graph in Figure 3, in which thicker edges have a higher weight. We start from the download node d. At every step, we walk one node backwards following the highest weight edge. We proceed until we reach a node with no predecessor, which we mark as the *origin* of the download path. If a node has more than one predecessor whose edges have the same weight, we follow the edge related to the predecessor node with the smaller time gap to the current node (measured w.r.t. the corresponding HTTP transactions).

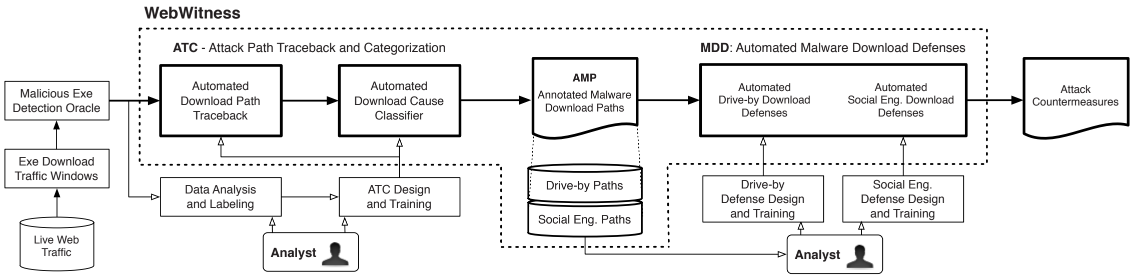
WebWitness
ATC - Attack Path Traceback and Categorization
MDD: Automated Malware Download Defenses
Malicious Exe Detection Oracle
Exe Download Traffic Windows
Live Web Traffic
Automated Download Path Traceback
Automated Download Cause Classifier
Data Analysis and Labeling
ATC Design and Training
Analyst
AMP
Annotated Malware Download Paths
Drive-by Paths
Social Eng. Paths
Automated Drive-by Download Defenses
Automated Social Eng. Download Defenses
Drive-by Defense Design and Training
Social Eng. Defense Design and Training
Analyst
Attack Countermeasures

Figure 2: WebWitness system details.

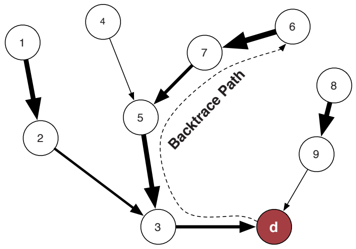

Figure 3: Example of download path traceback.

Possible False and Missing Edges: Naturally, the heuristics we use for tracing back the download path may in some cases add “false edges” to the graph or miss some edges. However, notice that these challenges are mitigated (though not always completely eliminated) by the following observations:

i) Our algorithm and heuristics aim to solve a much narrower problem than finding the correct “link” between all possible HTTP transactions in a network trace, because we are only concerned with tracing back a sequence of HTTP transactions that terminate into a malicious executable download.

ii) The “false edge” problem is mitigated by the fact that we always follow the strongest evidence. For example, consider Figure 3. Suppose the edge $(2 \rightarrow 3)$ was drawn due to rule (6), while edge $(5 \rightarrow 3)$ was drawn due to rule (2). In this case, even though edge $(2 \rightarrow 3)$ was mistakenly drawn (i.e., nodes 2 and 3 have no real “source of” relationship), the mistake is irrelevant, because our algorithm will choose $(5 \rightarrow 3)$ as part of the path, which is supported by stronger evidence.

iii) Our algorithm can output not only the sequence of HTTP transactions, but also the nature (and confidence) of every edge. Therefore, a threat analyst (or a downstream post processing system) can take the edge weights into account, before the reconstructed download path is used to make further decisions (e.g., remediation or takedown of certain domains in the download path).

3.2 ATC - Download Cause Classification

After we trace back the download path, we aim to label the reconstructed path as either *social engineering* or *drive-by* download. As shown in Figure 2, the output of this classification step allows us to obtain the annotated malware download paths (AMPs), which are then provided as input to the defense module (MDD).

While we are mainly interested in automatically identifying social engineering and drive-by download paths, we build a three-class classifier that can distinguish between three broad download causes, namely *social engineering*, *drive-by*, and *update/drop*. Essentially the *update/drop* class allows us to more easily identify and exclude malware downloads that are not caused by either social engineering or drive-by attacks.

To automatically classify the “cause” of an executable file download, WebWitness uses a supervised classification approach. First, we describe how we derive the features needed to translate malware download events into feature vectors that can be given as input to a statistical classifier. Then, we discuss how we derive the dataset used to train the classifier. To actually build the classifier, we used the random forest algorithm [7] (see Section 4).

Features: To discriminate between the three different classes, we engineered six statistical features that reflect, with a one-to-one mapping, the six characteristics of drive-by and social-engineering malware download paths that we discussed and measured in Section 2.5. For example, we measure binary feature (1) “Download Referer” as true if the HTTP request that initiated the download has a `Referer` header; a numerical feature (2) representing the “age” of domains serving “commonly exploitable” content; etc.

Training dataset: To train the classifier, we use the dataset of in-the-wild malware download paths that we collected and manually labeled during our initial investigation of in-the-wild malware downloads discussed in Section 2.5. Our training dataset contained the following number of labeled download paths: 164 instances of *drive-by* download paths, 191 instances of *social engineering* paths, and 328 *update/drop* samples.

3.3 MDD - Drive-by Defense

The annotated download paths output by ATC provide a large and up-to-date dataset of real-world malware download incidents, including the web paths followed by the victims (see Figure 2). This information is very useful for studying new attack trends and developing more effective defenses. As new defenses are developed, they can be plugged into the MDD module, so that as new malware download paths are discovered we can automatically derive appropriate countermeasures.

As an example that demonstrates how WebWitness can enable the development of more effective malware download defenses, we develop a new defense against drive-by download attacks based on code injections. While code injection attacks are not new, current defenses rely mainly on blacklisting the URLs serving the actual drive-by exploit or malware download, rather than blocking the URLs from which malicious code is injected. Our results (Section 4) show that by automatically tracing back drive-by download paths and identifying the code injection URLs, we can enable better defenses against future malware attacks.

Identifying code injection URLs: Given a *drive-by* download path output by the ATC module, we aim to automatically identify the *landing*, *injection*, and *exploit* nodes within the download path.We tackle this problem using a supervised classification approach. Namely, we train a separate classifier for each of the three types of nodes on a drive-by download path. The final output is a labeled drive-by download path.

Exploit Page Classifier: The exploit classifier takes as input a drive-by download path and labels its nodes as *exploit* or *non-exploit*. We define an exploit node as a page that carries content that exploits a vulnerability on the victim's machine, causing it to eventually download a malicious executable. The search for exploit nodes proceed "backwards", starting from the node prior to the executable download and ending at the root. It is not uncommon to have more than one exploit node in one path (e.g., some exploit kits try several exploits before success). Thus, multiple nodes could be labeled as *exploit*.

To build the classifier, we use the following features:

(1) *Hops to the download page*. Number of nodes on the download path between the considered node and the final malware download node. *Intuition*: It is typical for the exploit node to only be a few hops away from the actual download. In many cases, the node prior to the download event is an exploit node, because once the exploit succeeds the executable is downloaded immediately.

(2) *"Commonly exploitable" content*. Boolean feature that indicates if a node contains content for Java, Silverlight, Flash or Adobe Reader. *Intuition*: Browser plug-ins are a popular exploitation vector. The exploit is typically delivered though their content.

(3) *Domain age*. The number of days since the first observation of the node's effective second level domain in a large historic passive DNS database. *Intuition*: Exploit domains tend to be short-lived and often only active for one day.

(4) *Same domain*. Boolean feature that is true if the node's domain is equal to the download domain. *Intuition*: It is common for the exploit and download to be served by the same domain, as also noted in [13].

Landing Page Classifier: Once the exploit node(s) is labeled, we attempt to locate the landing page URL. Essentially, the landing page is the web page where the drive-by attack path begins. Often, the landing page itself is a non-malicious page that was previously compromised (or "hacked"). The landing page classifier calculates the probability that a node preceding the exploit node (labeled by the exploit page classifier discussed earlier) is a landing page. Nodes with a probability higher than a tunable detection threshold (50% in our experiments) are classified as "candidate landing" nodes. If there are multiple candidates, the one with the highest probability is labeled as the landing node.

To label a node as either *landing* or *non-landing*, we engineered the following statistical features:

(1) *Hops to the exploit page*. This feature set consists of the number of non-redirect nodes and unique effective second level domains between the node and the exploit node. *Intuition*: Often, all the nodes between the landing and exploit node are redirects [36]. Also, most drive-by downloads use one to three types of malicious domains (injection, exploit, download). Therefore, in most cases there are zero or one domains (the one being the injection domain) on the download path between the landing and exploit nodes.

(2) *Domain age*. We use two features based on domain age. The first feature is the age of the node's effective second level domain as computed from a passive DNS database. *Intuition*: The domains associated to ("hacked") landing pages tend to be long-lived. Furthermore, "older" landing pages tend to offer more benefits to the attackers, as they often attract more visitors (i.e., potential victims), because it takes time for legitimate pages to become popular. The second feature is the age of the oldest domain between the node and the exploit node. *Intuition*: Nodes on the download path between the landing and exploit nodes tend to be less than a year in age. This is because they are typically malicious and recently registered.

(3) *Same domain*. Boolean feature that is true if the node's domain is equal to the exploit domain. *Intuition*: It is uncommon for an exploit to be served

from the same domain as the landing page. They are typically kept separate because installing an exploit kit on a compromised website may increase the likelihood of detection by the legitimate site's webmaster. In addition, it is much easier to manage a centralized exploit kit server than keep all the compromised websites up-to-date with the latest exploits.

Injection Page Classifier: We define the injection page to be the source of the code inserted into the "hacked" landing page. Typically, the injection and exploit nodes are separate and are served via different domain names. This provides a level of indirection that allows the exploit domain to change without requiring an update to the landing page. The injection node by definition is a successor to the landing page, but depending on the injection technique it may or may not be directly present in the download path traced back by the ATC module. Therefore, the classifier calculates the injection page probability for each direct successor of the landing node in the transactions graph, instead of only considering nodes in the reconstructed download path. The successor of the landing page node with the highest probability is labeled as the injection page node.

To identify the *injection* page, for each successor of the *landing* node we measure the following features:

(1) *On path*. Boolean feature indicating if the node is on the download path. *Intuition*: Being on the download path and a successor of the landing page, makes it a good candidate for the injection node. However, the injection node is not always on the download path due to the structure of some drive-by downloads.

(2) *Advertisement*. Boolean feature that is true if the node is an ad. *Intuition*: By definition, the injection page is not an ad, but code injected into the landing page. It is common for ads that are not related to the malicious download to be served on a landing page. This feature help us exclude those ad nodes.

(3) *Domain age*. The number of days since the first observation of the node's effective second level domain in passive DNS. *Intuition*: Injection pages typically have the sole purpose of injecting malicious code. They are rarely hosted directly on compromised pages, because this would expose the malicious code to cleanup by the legitimate site owners, ending the attacker's ability to exploit visitors. Consequently, injection pages are hosted on "young" domains that are typically active for the lifetime of a website compromise.

(4) *Successors*. There are two features that are derived from the node's successors. First is the number of direct successors. *Intuition*: Injection nodes tend to have only one direct successor. They typically perform an HTTP redirect or dynamically update the DOM to include the URL of the exploit domain. Benign pages often have more than one direct successor because they load content from many different files or sources. The second feature is boolean and it is true if one of the node's successors is on the download path. It indicates there is a possible "source of" relationship between it and a node on the download path. Even though the node itself may not be on the download path.

(5) *Same domain*. There are two boolean features that compare domain names. The first checks for equality between the node's domain and the landing domain. *Intuition*: It is uncommon for the landing domain to equal the injection domain for reasons similar to those described in the landing page classifier's "same domain" feature described earlier. The second feature compares the node's domain to the exploit domain. *Intuition*: In approximately 70% of the observations in our measurement study (Section 2), the exploit and injection domains were different.

4 Evaluation

In this section, we evaluate WebWitness' ATC and MDD modules. We also demonstrate the overall benefits of our new defense approach against drive-by downloads, by measuring the effectiveness of blacklisting the injection domains discovered by WebWitness. We show that while blacklisting the injection domains provides a better defense, compared to blacklisting only the exploit and download domains, injection domains appear very rarely in current blacklists, including Google Safe Browsing and a variety of large public blacklists.

4.1 ATC - Download Cause Classification

The download cause classifier uses a supervised learning approach to label each download path as either *social engineering*, *drive-by* or *update/drop* (Section 3.2). To evaluate its accuracy, we use WebWitness to traceback and classify all malicious downloads collected from the large academic network (Section 2) in the months *following* our initial study and development of the system. Specifically, all download events and samples used during evaluation have *no overlap* with the data we used for the study presented in Section 2, to design WebWitness' features and heuristics, or to train our classifiers. Each malicious download observed during the *testing period* was then classified as one of the following: *drive-by*, *social engineering* or *update*. From each of the three predicted classes we randomly sampled 50 downloads for manual verification. We limited the sample size to a total of 150 downloads because of the extensive manual analysis required to determine the ground truth, including reverse engineering web pages, heavy javascript deobfuscation, complex html and plugin content analysis, etc. This time consuming review process allowed us to identify the correct web path and the *true cause* of download, creating our *ground truth* for the evaluation. Table 3 reports the confusion matrix for the cause classifier.

Table 3: Cause Classifier - Confusion Matrix Results

		Predicted Class		
	Class	Drive-by	Social	Update/Drop
Ground Truth	Drive-by	47	1	0
	Social	2	46	3
	Update/Drop	1	3	47

The classifier correctly labeled over 93% of the downloads. Notice that these results represent the overall system performance of the ATC module, because the download paths used in the experiment (i.e., input to the cause classifier) were extracted using our download path traceback algorithm (Section 3.1). The two social engineering samples classified as drive-by downloads both had commonly exploitable content (CEC) on the download path. They were misclassified even though the CEC domain ages were greater than 200 days. The three update/drop samples classified as social engineering was caused by invalid download paths resulting from the false edges described in the next section. Finally the three social engineering downloads misclassified as update/drop was a result of small downloads paths (all were length 3) and high download domain recurrence (all greater than 20 of the 48 hourly buckets).

4.2 ATC - Download Path Traceback

To evaluate the accuracy of our download path traceback algorithm (Section 3.1), we use the 150 manually reviewed downloads; i.e., our ground truth, from Section 4.1. For path traceback, we consider two types of errors for review: (1) missing nodes: the traceback stops short, before reaching the origin of the download path (recall that the traceback algorithm works its way backwards from the download node to the path origin); (2) false node: a node that should not appear in the download path. Table 4 summarizes the results of our evaluation.

Table 4: Download Path Traceback Results.

	Paths	Correctly Traced Back	Missing	False
Drive-By	48	45	3	0
Social	51	46	2	3
Update/Drop	51	47	0	4

The results show that 92% of the download paths were correctly traced back by our system. The 5 with missing nodes all had a *referer* header in the origin node's request, but a matching URL was not contained in the trace. This was likely due to our system not observing all the packets related to those transactions. The 7 with the false nodes were all caused by the "same-domain" heuristic incorrectly connecting the paths of an update and a social engineering download. The heuristic failed because the updates were performed by a malicious executable seconds after the user was socially engineered into downloading it from the same domain as the update.

4.3 MDD - Detecting Injection Domains

As discussed in detail in Section 3.3, we aim to automatically identify the malicious code injection domains often employed in drive-by download attacks. To achieve this goal, we use a cascade of three classifiers: an *exploit*, a *landing*, and an *injection* classifier (Section 3.3). In the following, we evaluate the performance of each one.

To build the training dataset, we use 117 drive-by malware downloads collected and manually labeled during our six-month malware study described in Section 2. These 117 drive-by paths contained 246 exploit nodes (notice that it is not uncommon for a drive-by attack to serve more than one exploit, especially when the first exploit attempt fails). There is only one landing node and one injection node per download path.

Table 5: Node Labeling for Drive-By Download Paths

Experiment	Classifier	Correctly Labeled	Incorrectly Labeled
Cross-Validation	Exploit	99.19%	0%
	Landing	96.58%	0.17%
	Injection	94.87%	0.07%

We performed 10-fold cross-validation tests using the dataset described above. Table 5 summarizes the results. As can be seen, all classifiers are highly accurate. The results of the the injection page classifier represent the performance of the final injection domain detection task. This is due to fact that all tests were conducted using the three classifiers (exploit, landing, and injection) in cascade mode to mirror an actual deployment of WebWitness' MDD module. Thus, overall, we obtained a minimum of 94.87% detection rate at 0.07% false positives.

There were a total 7 domains mislabeled as injection by our system. The most common error was labeling the exploit domain as the injection domain; i.e., missing the fact that a separate injection domain existed. This was the case for 5 of the 7 mislabeled domains. Since these domains are malicious, blacklisting them will not cause false positives. The other two domains were benign. One of them had an Alexa rank over $260,000$ and the other above $1,600,000$. To mitigate such false positives, the newly discovered injection domains could be reviewed by analysts before blacklisting. As WebWitness provides the analyst with full details on the traffic collected before the download and the reconstructed download path, this information can make the analyst's verification process significantly less time-consuming.

4.4 MDD - Defense Efficacy & Advantages

Domain name and URL blacklisting are commonly practiced defenses [2]. However, blacklists are only effective if the blacklisted domains remain in use for some period of time after they are detected. The longer-lived a malicious domain, the more useful it is to blacklist it. As discussed in Sections 3.3 and 4.3, WebWitness is able not only to identify the domains from which malware files are downloaded, but also to identify the malicious code injection and exploit domains within drive-by malware download paths. Clearly, these domains are all candidates for blacklisting.

To evaluate the efficacy of blacklisting the code injection domains, we demonstrate the advantages this provides compared to the currently more common approach of blacklisting the exploit and download domains. To this end, we use a set of 88 "complete" injection-based drive-by download paths that we were able to collect from a large academic network. These samples were "complete" paths in the sense that they were manually verified to have an injection, exploit, and malware download node (and related domain).

We evaluate the effect of blocking the different types of drive-by path domains by counting the number of potential victims that would be saved by doing so. Specifically, we define a potential victim as a unique client host visiting a blacklisted domain. Notice that the actual number of hosts that get infected may be smaller than the number of potential victims, because only some of the hosts that visit a malicious domain involved in a drive-by download attack will "successfully" download and run the malware file (e.g., because an anti-virus blocked the malware file from running on the machine). However, we can use the potential victim count to provide a relative comparison on the effectiveness of blacklisting injection versus exploit and malware download domains.

To count the potential victims, we rely on a very large passive DNS (pDNS) database that spans multiple Internet Service Providers (ISPs) and corporate networks. This pDNS dataset stores the historic mappings between domains and IP addresses, and also provides a unique source identifier for each host that queries a given domain name. This allows us to identify all the unique hosts that queried a given domain in a given timeframe (e.g., a given day). For each injection-based drive-by download paths in our set, we compute the potential victims saved by counting the number of unique hosts that query the injection, exploit, and file download domains in the 30 days following the date when we observed and labeled the download event. Figure 4 shows our results, in which day-0 is the day when we detected a malicious download path (the victims counts are aggregated, per day, for all hosts contacting a malicious domain). We can immediately see that the number of potential victims that query the exploit or file download domains rapidly drops as the exploit domain ages. On the other hand, injection domains are longer lived, and blacklisting them would prevent a much larger number of potential victims from being redirected to new (unknown) and frequently churning exploit and file download domains. Blacklisting the injection domain saves almost 6 *times* more potential victims, compared to blacklisting the exploit domain.

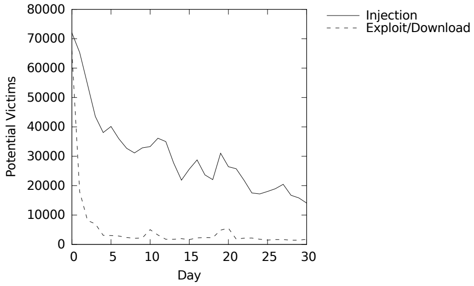

Figure 4: Potential victims saved by blocking the injection versus exploit/download domains on drive-by paths.

4.5 Blacklists & Google Safe Browsing

In this section, we aim to gain additional insights into the advantages that could be provided by our WebWitness' MDD module, compared to existing domain blacklists.

Public Domain Blacklists First, given the entire set of malicious domain names related to drive-by downloads discovered during our study and deployment of WebWitness, we counted how many of these domains appeared in popular public blacklists. We also measured the delay between when we first discovered the domain on a malware download path and when it appeared on a blacklist. This was possible because we repeatedly collected all domain names reported by the following set of public blacklists every day for more than a year: support.clean-mx.de, malwaredomains.com, zeustracker.abuse.ch, phishtank.com and malwaredomainlist.com. Table 6 summarizes our findings.

Table 6: Public Blacklisting Results.

	Uniq. Domains		Days: Detect to Blacklist		
	Observed	Blacklisted	Min.	Med.	Mean
Exploit/Download	152	9	1	20	29
Injection	52	6	20	31	36

As shown in Table 6, from all drive-by download paths that we were able to identify, reconstruct, and label, we collected a total of 52 unique drive-by code injection domains and 152 unique drive-by exploit and malware file download domains. Overall, less than 10% of these domains ever appeared on a public blacklist. As we can see, more exploit/download domains (a total of 9) were blacklisted, compared to the injection domains (only 6). Furthermore, we can see that the minimum time it took for an injection domain to appear in at least one blacklist was 20 days, whereas some exploit domains were blacklisted almost immediately (after only one day).

Because injection domains are typically longer lived than exploit domains, and because the same injection domain is often used throughout the course of a drive-by download campaign to redirect users to different (short-lived) exploit domains, identifying and blocking injection domains has a significant advantage. By helping to quickly identify and blacklist injection domains, WebWitness enables the creation of better defenses against drive-by downloads, thus helping to significantly reduce the number of potential malware victims, as we also demonstrated in the previous Section 4.4.

Google Safe Browsing For the last few weeks of our deployment of WebWitness, we checked the domain names related to the drive-by download paths reconstructed by our system against Google Safe Browsing (GSB) [2]. Specifically, given a malware download path and its malicious domains, we queried GSB on the next day, compared to the day the malware download was observed. Overall, during this final deployment period we observed 34 drive-by download paths. GSB detected a total of 6 malicious domains that were related to only 4 out of the 34 downloads. The domains GSB detected were used to serve drive-by exploits, the malware file themselves, or were related to ads used to lead the victims to a browser exploit. *None of the domains detected by GSB were injection domains*, even though our 34 download paths included 12 unique injection domains.

It is important to notice, however, that while GSB detected malicious domains related to only 4 out of our 34 drive-by download paths, there may be many more malware downloads that WebWitness cannot observe, simply because they are blocked "up front" by GSB. Because WebWitness passively collects malware download traces from the network whenever a malicious executable file download is identified in the traffic, it is very possible that in many cases GSB simply prevented users who were about to visit a drive-by-related domain from loading the malicious content, and therefore from downloading the malware file in the first place. Nonetheless, the fact that WebWitness automatically discovered 30 drive-by download paths that were not known to GSB demonstrates that our system can successfully complement existing defenses.

4.6 Case Studies

4.6.1 Social Engineering

Figure 5 shows the *download path* for an in-the-wild social engineering attack, including the "link" relationships between nodes in the path. The user first performs a search on `www.youtube.com` (A) for a "*facebook private profile viewer*", which is the *root* of the path. Next, the user clicks on the top search result leading to a "*trick*" page on `www.youtube.com` (B), which hosts a video demonstrating a program that supposedly allows the viewing of the private profiles of Facebook users. A textual description under the video provides a link to download a "profile viewer" application through a URL shortener `goo.gl` (C). This shortened URL link redirects the user to `uploading.com` (D), a free file sharing site that prompts the user with a link to start the download. This leads to another `uploading.com` (E) page that thanks the user for downloading the file and opens a new `uploading.com` (F) page that includes an `<iframe>` with source `fs689.uploading.com` (G), from which the executable file is downloaded. The file is labeled as "Trojan Downloader" by some anti-virus scanners.

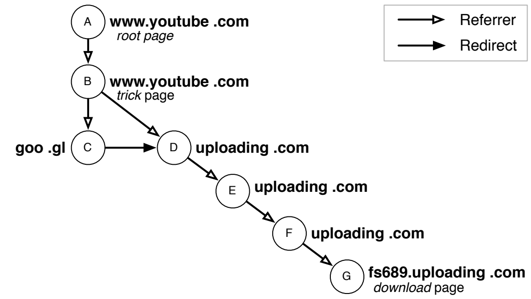

Figure 5: Social engineering download example.

Notice that no exploit appears to be involved in this attack, and that the user (highly likely) had to explicitly click on various links and on the downloaded malware file itself to execute it.

4.6.2 Drive-by

Figure 6 shows the *download path* related to an in-the-wild drive-by download.

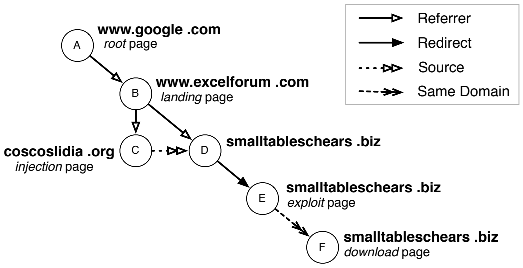

Figure 6: Drive-by download example.

The download path originates from (A) `www.google.com` (the *root* page), where the user entered the search terms "*add years and months together*." The first link in the search results, which the user clicked on, is for a webpage (B) on `www.excelforum[dot]com` (the *landing* page). Sadly, the page the user landed

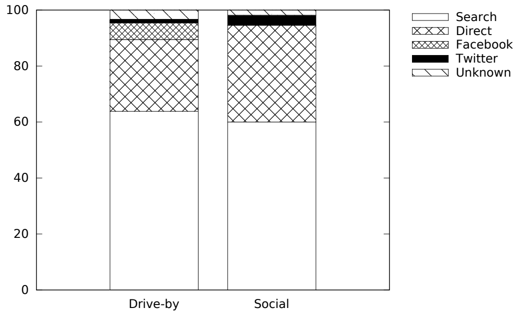

Figure 7: “Root” of malware download paths.

on was compromised several days earlier, resulting in the addition of a `<script>` tag with source at `coscoslidia[dot]org`, which is the *injection* page. The script is automatically retrieved from (C) and executed, forcing an `<iframe>` to be added and rendered. The source of the frame (D) is on the site `smalltableschears[dot]biz`, from which the content is immediately fetched and included in the page. The newly loaded javascript served by (D) then checks for the presence of vulnerable versions of several browser plugins. It quickly matches a version of the installed Adobe Flash Player to a known vulnerability and dynamically adds another `<iframe>` to the page, which pulls a malicious Flash exploit file from (E) on the same `smalltableschears[dot]biz` site (the *exploit* page). The Flash exploit succeeds and the shellcode fetches a malware binary (labeled as ZeroAccess by some AVs) from (F) on the same domain `smalltableschears[dot]biz` (the *download* page).

4.7 “Origin” of Malware Download Paths

Figure 7 shows a breakdown of the drive-by and social engineering “origins” behind the malware downloads. For drive-by downloads, 64% of the download paths started with a search. We noticed that the search query keywords were typically very “normal” (e.g., searching for a new car, social events, or simple tools, as shown in the example in Section 4.6.2), but unfortunately the search results linked to hacked websites that acted as the “entry point” to exploit distribution sites and malware downloads.

For social engineering downloads, about 60% of the web paths started with a search. Search engine queries that eventually led to social engineering attacks tended to be related to less legitimate content. For example, the search queries were often related to free streaming links, pirated movies, or pirated versions of popular expensive software. In these cases, the search results contained links offering content relevant to the search, but the related search result pages would also encourage the user to install malicious software disguised as some required application (e.g., a video codec or a software key generator).

The second most common origin is direct links, whereby a user arrives to a webpage directly (e.g., by clicking on a link within a spam email), rather than through a link from another site. Most of these direct links point to a benign website that is either hacked or displays malicious ads.

Facebook and Twitter represent a relatively infrequent origin for malware downloads (7% and 3% of the cases, respectively). While both Facebook and Twitter usually rely on encrypted (HTTPS) communications, we were able to determine if a download path originated from their sites by noticing that Facebook makes sure that all external requests carry a generic `www.facebook.com` referrer [14]. On the other hand, requests initiated by clicking on a link published on twitter carry a referrer containing a `t.co` shortening URL. During our entire deployment, we only observed one case in which a link from Facebook or Twitter led directly to a drive-by exploit kit. In all other cases, the links led first to a legitimate page that was hacked or that displayed a malicious ad.

For the remaining malicious downloads (less than 3%, overall) we were unable to trace them back to their origin (e.g., due to missing traffic).

During our deployment, we also found that malicious ads are responsible for a significant fraction of the malware downloads in our dataset. Specifically, malicious ads were included in the web path of about 25% of drive-by and 40% of social engineering malware downloads. The malicious ads we observed were typically displayed on relatively unpopular websites. We observed only one example of a malicious ad served on a website with a US Alexa ranking within the top 500.

5 Discussion and System Limitations

Our system only collected data during off-peak hours because it was sharing hardware resources with a production network monitoring system whose functionality could not be disrupted. Thus, our data is just a sample of the malicious downloads that occurred during this period. Also, due to the significant efforts required to analyze complex malware download traces, our evaluation ground truth is limited to a representative sample of the malicious downloads that occurred in the monitored network. However, based on our extensive manual analysis, we believe the samples to be very diverse because of the various exploit kits, exploits, social engineering tricks and malware observed, and therefore representative of the overall set of malware downloads that occurred during our deployment.

One may think that attackers could avoid detection by simply distributing malicious files over encrypted web traffic, using HTTPS. However, it is worth noting that in sensitive networks (e.g., enterprise and government networks) it is now common practice to deploy SSL Man-In-The-Middle (MITM) proxies, which allow for inspecting and recording the content both HTTP and HTTPS traffic (perhaps excluding the traffic towards some whitelisted sides, such as banking applications, etc.). WebWitness could simply work alongside such SSL MITM proxies.

Because the detection of malicious executable files is outside the scope of this paper, we have relied on a "detection oracle" to extract malicious download traces from the network traffic. For the sake of this study, we have chosen to rely on multiple AV scanners. It is well known, though, that AV scanners suffer from false positive and negatives. In addition, the labels assigned by the AV are often not completely meaningful. However, we should consider that using multiple AV scanners reduces the false negatives, and the set of filtering heuristics we discussed in Section 2.2 can mitigate the false positives. In addition, we used re-scanning over a period of a month for each of the downloaded executable files we collected, to further improve our ground truth. Finally, we used the AV labels to filter out adware downloads, because we are mainly interested in the potentially most damaging malware infections. We empirically found that the AV labels usually do a decent job at separating the broad adware and malware classes. Also, we manually reviewed all samples of malware downloads in our dataset, to further mitigate possible mislabeling problems.

Attackers with knowledge of our system may try to evade it by using a purposely crafted attack in attempt to alter some of the features we use in Section 3 to perform path traceback, categorization and for node labeling. Most likely, the attacker will have as a primary goal the evasion of our traceback algorithm. This, for example, could be done by forcing a "disconnect" between the final malware download node and its true predecessors. Such an attack theoretically may be possible, especially in case of drive-by attacks. In such events the browser is compromised and is (in theory) under the full control of the attacker. Now, if the malware download node is isolated in the reconstructed download path, the cause classifier may label the download event a malware update, thus preventing any further processing of the download path (i.e., any attempt to identify the exploit and injection domains).

However, we should also notice that most drive-by downloads are based on what we refer as "commonly exploitable" content in Section 3.3 (e.g., .jar, .swf, or .pdf files that carry an exploit). For such type of drive-by download attacks, the "commonly exploitable" content feature should connect the exploit and the download, if they occur in a small time window. If needed, the time window could be extended by requiring the domain severing the content to be young by checking its "age" before making a connection. Since the exploit must occur before the attacker has control of the browser, it is more difficult to evade.

6 Related Work

Client honeypots actively visit webpages and detect drive-by downloads though observing changes to the system [1, 21, 22, 28, 29] or by analyzing responses for malicious content [3, 10, 24]. These systems tend to have a low false positive rate, but only find malicious websites by visiting them with exploitable browser configurations; also, they have limited range in the quantity of pages they can crawl because they are much slower than static crawlers. Often candidate URLs are selected by filtering content from static crawlers [28, 29, 35], using heuristics to visit parts of the web that are likely more malicious [8] or using search engines to identify webpages that contain content similar to known malicious ones [12].

A number of techniques have been developed to detect drive-by downloads through examining content [10, 11, 15, 32, 34]. Signature based intrusion detection systems, such as Snort [34], passively search network traffic content for patterns of known attacks. Both static [11] and dynamic [10] analysis of JavaScript has been used to detect attacks. The disadvantages of using content is that it is complex and under the control of the attacker. Polymorphic malware and code obfuscation results in missed attacks for signature and static analysis systems, and dynamic analysis can be detected by malware and subverted by altering its execution path [15].

Other systems focus on the redirection chain that leads to drive-by downloads. Stringhini et al [36] create redirection graphs by aggregating redirection chains that end at the same webpage. Features from the redirection graph and visiting users are then used to classify the webpage as malicious or benign. Mekky et al [20] build browsing activity trees using the referrer and redirection headers as well as URLs embedded in the content. Features related to the redirection chain for each tree are extracted and used to classify the activity as malicious or benign. Li et al [17] apply page rank from the dark and bright side of the web to a partially labeled set of redirection chains to separate benign and malicious web paths. They find the majority of malicious paths are directed through traffic distribution systems. Using features from the redirection chain, Surf [18] detects malicious websites found in search engine results due to search poisoning and WarningBird [16] identifies malicious webpages posted on Twitter. These systems focus on the redirection chain and features extracted from it to classify a web activity as benign or malicious. Whereas, WebWitness provides context to malicious downloads by reconstructing the full download path (not just the redirection chain), classifying the cause of the download (drive-by, social, update) and identifying the roles of the domains involved in the attack.

Static blacklists [2] of domains/URLs and domain reputation systems [4, 5] identify malicious websites to prevent users from visiting them. Many of the domains on static blacklists are exploit and download domains that change frequently rendering them less effective. On the other hand, reputation systems only provide a malicious score for a domain and do not indicate their role or give context to an attack. By analyzing the structure of a malicious download, WebWitness can identify the type of attack and the domain roles; providing the highest value domains for blocking and reputation training data.

Recently researchers have proposed executable reputation systems [13,30,38] due the limitations of signature AV [27]. Instead of using content features from the executable content, they focus on properties of the malware distribution infrastructure. These systems can be very effective at identifying malicious downloads. However, they do not provide any context such as how and why the user came to download a malicious executable. Providing download context is the goal of WebWitness not malicious executable detection. We see these systems as complementary to WebWitness and as good candidates to replace our current oracle (signature AV) for malicious executable detection.

Web traffic reconstruction has been studied for example in [9, 25, 39]. WebPatrol [9] uses a client honeypot and a modified web proxy to collect and replay web-based malware scenarios. Unlike WebPatrol, WebWitness is not limited to drive-by downloads invoked through client honeypots and can provide context to drive-by and social engineering attacks on real users observed on live networks. ReSurf [39] uses the referrer header to build graphs of related HTTP transactions to reconstruct web-surfing activities. As discussed in this paper and evaluated in [25], this approach is very limited especially in reconstructing the entire download path of a malicious executable. Lastly, ClickMiner [25] reconstructs user-browser interactions by replaying recorded network traffic through an instrumented browser. Its focus is on the user's behavior that led to a webpage; whereas, WebWitness identifies the cause and structure of an attack that led to a malicious download.

7 Conclusion

We proposed a novel incident investigation system, named WebWitness. Our system targets two main goals: 1) automatically *trace back and label* the chain of events (e.g., visited web pages) preceding malware downloads, to highlight how users reach attack pages on the web; and 2) leverage these automatically labeled in-the-wild malware download paths to better understand current attack trends, and to *develop more effective defenses*.

We deployed WebWitness on a large academic network for a period of 10 months, where we collected and categorized thousands of *live* malware download paths. An analysis of this labeled data allowed us to design a new defense against drive-by downloads that rely on injecting malicious content into (hacked) legitimate web pages. For example, we show that on average by using the results of WebWitness we can decrease the infection rate of drive-by downloads based on malicious content injection by almost *6 times*, compared to existing URL blacklisting approaches.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation (NSF) under grant No. CNS-1149051 and US Department of Commerce (Commerce) under grant no. 2106DEK. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF and Commerce.

References

[1] Capture-hpc client honeypot. `https://projects.honeynet.org/capture-hpc`.

[2] Google safe browsing api. `https://developers.google.com/safe-browsing/`.

[3] Honeyc. `https://projects.honeynet.org/honeyc`.

[4] ANTONAKAKIS, M., PERDISCI, R., DAGON, D., LEE, W., AND FEAMSTER, N. Building a dynamic reputation system for DNS. In *the Proceedings of 19th USENIX Security Symposium (USENIX Security '10)* (2010).

[5] ANTONAKAKIS, M., PERDISCI, R., LEE, W., DAGON, D., AND VASILOGLOU, N. Detecting Malware Domains at the Upper DNS Hierarchy. In *the Proceedings of 20th USENIX Security Symposium (USENIX Security '11)* (2011).

[6] BAKHSHI, T., PAPADAKI, M., AND FURNELL, S. A practical assessment of social engineering vulnerabilities. In *2nd International Symposium on Human Aspects of Information Security & Assurance (HAISA 2008)* (2008), pp. 12–23.

[7] BREIMAN, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001).

[8] CANALI, D., COVA, M., VIGNA, G., AND KRUEGEL, C. Prophiler: A fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web* (New York, NY, USA, 2011), WWW '11, ACM.

[9] CHEN, K. Z., GU, G., ZHUGE, J., NAZARIO, J., AND HAN, X. Webpatrol: Automated collection and replay of web-based malware scenarios. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2011), ASIACCS '11, ACM.

[10] COVA, M., KRUEGEL, C., AND VIGNA, G. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web* (New York, NY, USA, 2010), WWW '10, ACM.

[11] CURTSINGER, C., LIVSHITS, B., ZORN, B., AND SEIFERT, C. Zozzle: Fast and precise in-browser javascript malware detection. In *Proceedings of the 20th USENIX Conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association.

[12] INVERNIZZI, L., BENVENUTI, S., COVA, M., COMPARETTI, P. M., KRUEGEL, C., AND VIGNA, G. Evilseed: A guided approach to finding malicious web pages. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (2012), SP '12, IEEE Computer Society.

[13] INVERNIZZI, L., LEE, S.-J., MISKOVIC, S., MELLIA, M., TORRES, R., KRUEGEL, C., SAHA, S., AND VIGNA, G. Nazca: Detecting malware distribution in large-scale networks.

[14] JONES, M. Protecting privacy with referrers, 2010. `https://www.facebook.com/notes/facebook-engineering/protecting-privacy-with-referrers/392382738919`.

[15] KAPRAVELOS, A., SHOSHITAISHVILI, Y., COVA, M., KRUEGEL, C., AND VIGNA, G. Revolver: An automated approach to the detection of evasiveweb-based malware. In *Proceedings of the 22Nd USENIX Conference on Security* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association.

[16] LEE, S., AND KIM, J. Warningbird: A near real-time detection system for suspicious urls in twitter stream. *IEEE Trans. Dependable Secur. Comput. 10*, 3 (May 2013).

[17] LI, Z., ALRWAIS, S., XIE, Y., YU, F., AND WANG, X. Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (2013), SP '13.

[18] LU, L., PERDISCI, R., AND LEE, W. Surf: Detecting and measuring search poisoning. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2011), CCS '11, ACM.

[19] LU, L., YEGNESWARAN, V., PORRAS, P., AND LEE, W. Blade: An attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM.

[20] MEKKY, H., TORRES, R., ZHANG, Z.-L., SAHA, S., AND NUCCI, A. Detecting malicious http redirections using trees of user browsing activity. In *INFOCOM, 2014 Proceedings IEEE* (2014).

[21] MIN WANG, Y., BECK, D., JIANG, X., ROUSSEV, R., VERBOWSKI, C., CHEN, S., AND KING, S. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *In NDSS* (2006).

[22] MOSHCHUK, E., BRAGIN, T., GRIBBLE, S. D., AND LEVY, H. M. A crawler-based study of spyware on the web.

[23] NAPPA, A., RAFIQUE, M. Z., AND CABALLERO, J. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (Berlin, Heidelberg, 2013), DIMVA'13, Springer-Verlag.

[24] NAZARIO, J. Phoneyc: A virtual client honeypot. In *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (Berkeley, CA, USA, 2009), LEET'09, USENIX Association.

[25] NEASBITT, C., PERDISCI, R., LI, K., AND NELMS, T. Clickminer: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer &#38; Communications Security* (New York, NY, USA, 2014), CCS '14, ACM.

[26] NELMS, T., PERDISCI, R., AND AHAMAD, M. Execscent: Mining for new c&#38;c domains in live networks with adaptive control protocol templates. In *Proceedings of the 22Nd USENIX Conference on Security* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association.

[27] PERDISCI, R., LANZI, A., AND LEE, W. Classification of packed executables for accurate computer virus detection. *Pattern Recogn. Lett.*.

[28] PROVOS, N., MAVROMMATIS, P., RAJAB, M. A., AND MONROSE, F. All your iframes point to us. In *Proceedings of the 17th Conference on Security Symposium* (Berkeley, CA, USA, 2008), SS'08, USENIX Association.

[29] PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, N. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets* (2007), HotBots'07.

[30] RAJAB, M. A., BALLARD, L., LUTZ, N., MAVROMMATIS, P., AND PROVOS, N. Camp: Content-agnostic malware protection. In *Proceedings of Annual Network and Distributed System Security Symposium, NDSS (February 2013)* (2013), Citeseer.

[31] RAJAB, M. A., BALLARD, L., MAVROMMATIS, P., PROVOS, N., AND ZHAO, X. The nocebo effect on the web: An analysis of fake anti-virus distribution. In *3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (Berkeley, CA, USA, 2010), LEET'10, USENIX Association.

[32] RATANAWORABHAN, P., LIVSHITS, B., AND ZORN, B. Nozzle: A defense against heap-spraying code injection attacks. In *Proceedings of the 18th Conference on USENIX Security Symposium* (Berkeley, CA, USA, 2009), SSYM'09, USENIX Association.

[33] RIECK, K., KRUEGER, T., AND DEWALD, A. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference* (New York, NY, USA, 2010), ACSAC '10, ACM.

[34] ROESCH, M. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration* (1999).

[35] STOKES, J. W., ANDERSEN, R., SEIFERT, C., AND CHELLAPILLA, K. Webcop: Locating neighborhoods of malware on the web. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (2010), LEET'10, USENIX Association.

[36] STRINGHINI, G., KRUEGEL, C., AND VIGNA, G. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security* (2013), CCS '13, ACM.

[37] TOWNSEND, K. R&d: The art of social engineering. *Infosecurity 7*, 4 (2010), 32–35.

[38] VADREVU, P., RAHBARINIA, B., PERDISCI, R., LI, K., AND ANTONAKAKIS, M. Measuring and detecting malware downloads in live network traffic. In *ESORICS*. 2013.

[39] XIE, G., ILIOFOTOU, M., KARAGIANNIS, T., FALOUTSOS, M., AND JIN, Y. Resurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference, 2013* (May 2013), pp. 1–9.

[40] ZHANG, J., SEIFERT, C., STOKES, J. W., AND LEE, W. Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th International Conference on World Wide Web* (New York, NY, USA, 2011), WWW '11, ACM.