



Towards Measuring and Mitigating Social Engineering Software Download Attacks

Terry Nelms, Georgia Institute of Technology and Damballa; 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/usenixsecurity16/technical-sessions/presentation/nelms>

**This paper is included in the Proceedings of the
25th USENIX Security Symposium**

August 10–12, 2016 • Austin, TX

ISBN 978-1-931971-32-4

**Open access to the Proceedings of the
25th USENIX Security Symposium
is sponsored by USENIX**

Towards Measuring and Mitigating Social Engineering Software Download Attacks

Terry Nelms^{1,2}, Roberto Perdisci^{3,1}, Manos Antonakakis¹, and Mustaque Ahamad^{1,4}

¹Georgia Institute of Technology

²Damballa, Inc.

³University of Georgia

⁴New York University Abu Dhabi

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

Abstract

Most modern malware infections happen through the browser, typically as the result of a drive-by or social engineering attack. While there have been numerous studies on measuring and defending against drive-by downloads, little attention has been dedicated to studying social engineering attacks.

In this paper, we present the first systematic study of web-based social engineering (SE) attacks that successfully lure users into downloading malicious and unwanted software. To conduct this study, we collect and reconstruct more than two thousand examples of in-the-wild SE download attacks from live network traffic. Via a detailed analysis of these attacks, we attain the following results: (i) we develop a categorization system to identify and organize the tactics typically employed by attackers to gain the user's attention and deceive or persuade them into downloading malicious and unwanted applications; (ii) we reconstruct the web path followed by the victims and observe that a large fraction of SE download attacks are delivered via online advertisement, typically served from "low tier" ad networks; (iii) we measure the characteristics of the network infrastructure used to deliver such attacks and uncover a number of features that can be leveraged to distinguish between SE and benign (or non-SE) software downloads.

1 Introduction

Most modern malware infections happen via the browser, typically triggered by social engineering [9] or drive-by download attacks [33]. While numerous studies have focused on measuring and defending against drive-by downloads [14, 17, 28, 38], malware infections enabled by social engineering attacks remain notably understudied [31].

Moreover, as recent defenses against drive-by downloads and other browser-based attacks are becoming harder to circumvent [18, 24, 32, 36, 40], cyber-criminals increasingly aim their attacks against the weakest link,

namely the user, by leveraging sophisticated social engineering tactics [27]. Because social engineering (SE) attacks target users, rather than systems, current defense solutions are often unable to accurately detect them. Thus, there is a pressing need for a comprehensive study of social engineering downloads that can shed light on the tactics used in modern attacks. This is important not only to inform better technical defenses, but may also allow us to gather precious information that may be used to better train users against future SE attacks.

In this paper, we present a study of real-world SE download attacks. Specifically, we focus on studying *web-based SE attacks* that unfold *exclusively via the web* and that do not require "external" triggers such as email spam/phishing, etc. An example of such attacks is described in [9]: a user is simply browsing the web, visiting an apparently innocuous blog, when his attention is drawn to an online ad that is subtly crafted to mimic a warning about a missing browser plugin. Clicking on the ad takes him to a page that reports a missing codec, which is required to watch a video. The user clicks on the related codec link, which results in the download of malicious software.

To conduct our study, we collect and analyze hundreds of *successful* in-the-wild SE download attacks, namely SE attacks that actually result in a victim downloading malicious or unwanted software. We harvest these attacks by monitoring *live* web traffic on a large academic network. Via a detailed analysis of our SE attack dataset, we attain the following main results: (i) we develop a categorization system to identify and organize the tactics typically employed by attackers to gain a user's attention and deceive or persuade them into downloading malicious and unwanted applications; (ii) we reconstruct the *web path* (i.e., sequence of pages/URLs) followed by SE victims and observe that a large fraction of SE attacks are delivered via online advertisement (e.g., served via "low tier" ad networks); (iii) we measure the characteristics of the network infrastructure (e.g., domain names) used to

deliver such attacks, and uncover a number of features that can be leveraged to distinguish between SE and benign (i.e., “non-SE”) software downloads.

Our findings show that a large fraction of SE attacks (almost 50%) are accomplished by *repackaging* existing benign applications. For instance, users often download free software that comes as a bundle including the software actually desired by the user plus some Adware or other Potentially Unwanted Programs (PUPs). This confirms that websites serving free software are often involved (willingly or not) in distributing malicious or unwanted software [4, 7].

The second most popular category of attacks is related to alerting or urging the user to install an application that is supposedly needed to complete a task. For instance, the user may be *warned* that they are running an outdated or insecure version of Adobe Flash or Java, and are offered to download a software update. Unfortunately, by downloading these supposed updates, users are infected. Similarly, users may stumble upon a page that supposedly hosts a video of interest. This page may then inform the user that a specific video codec is needed to play the desired video. The user *complies* by downloading the suggested software, thus causing an infection (see Section 3 for details).

Another example of an SE download attack is represented by fake anti-viruses (FakeAVs) [35]. In this case, a web page alerts the user that their machine is infected and that AV software is needed to clean up the machine. In a way similar to the SE attack examples reported above, the user may be persuaded to download (in some cases after a payment) the promoted software, which will infect the user’s machine. However, while FakeAVs have been highly popular among attackers in the recent past, our study of in-the-wild SE malware downloads finds that they represent less than 1% of modern SE attacks. This sharp decline in the number of FakeAV attacks within the last few years is consistent with the recent development of technical countermeasures against this class of attacks [5] and increased user awareness [6].

As mentioned earlier, a large fraction of SE download attacks (more than 80%) are initiated via advertisements, and that the “entry point” to these attacks is represented by only a few low-tier advertisement networks. For instance, we found that a large fraction of the web-based SE attacks described above are served primarily via two ad networks: `onclickads.net` and `adcash.com`.

By studying the details of SE download attacks, we also discover a number of features that aid in the detection of SE download attacks on live web traffic. We train a classifier using these features and measure its effectiveness at detecting SE downloads.

Summary of Contributions:

- We present the first systematic study of modern web-based SE download attacks. For instance, our analysis of hundreds of SE download attack instances reveals that most such attacks are enabled by online advertisements served through a handful of “low tier” ad networks.
- To assist the process of understanding the origin of SE download attacks, we develop a categorization system that expresses how attackers typically gain a user’s attention, and what are the most common types of deception and persuasion tactics used to trick victims into downloading malicious or unwanted applications. This makes it easier to track what type of attacks are most prevalent and may help to focus user training programs on specific user weaknesses and particularly successful deception and persuasion tactics currently used in the wild.
- Via extensive measurements, we find that the most common types of SE download attacks include *fake updates* for Adobe Flash and Java, and that fake anti-viruses (FakeAVs), which have been a popular and effective infection vector in the recent past, represent less than 1% of all SE downloads we observed in the wild. Furthermore, we find that existing defenses, such as traditional anti-virus (AV) scanners, are largely ineffective against SE downloads.
- Based on our measurements, we then identify a set of features that allow for building a statistical classifier that is able to accurately detect ad-driven SE download attacks with 91% true positives and only 0.5% false positives.

2 Study Overview

Our study of SE download attacks is divided in multiple parts. To better follow the results discussed in the following sections, we now present a brief summary of their content.

In Section 3, we analyze the range of deception and persuasion tactics employed by the attackers to victimize users, and propose a categorization system to systematize the in-the-wild SE tactics we observed.

In Section 4, we discuss how we collect software downloads (including malicious ones) from live network traffic and reconstruct their *download path*. Namely, we trace back the sequence of pages/URLs visited by a user before arriving to a URL that triggers the download of an executable file (we focus on Windows portable executable files). We then analyze the collected software download events, and label those downloads that result

from SE attacks. This labeled dataset is used in the following sections to enable a detailed analysis of the characteristics of the SE download attacks.

We analyze our dataset of in-the-wild SE download attacks in Section 5. Specifically, we measure how the SE attack tactics are distributed, according to the categorization system proposed in Section 3, and highlight the most popular *successful* SE malware attacks. According to our dataset, the majority of SE attacks are promoted via online advertisement. Therefore, in Section 5 we also present an analysis of the network-level properties of ad-based SE malware attacks, and contrast them with properties of ad-driven benign software downloads.

In Section 6, we focus on detecting ad-based SE download attacks. We first show that anti-virus products detect only a small fraction of all SE attacks, leaving most “fresh” SE download events undetected. We then devise a number of statistical features that can be extracted from the network properties of ad-driven software download events, and show that they allow us to build an accurate SE attack classifier, which could be used to detect and stop SE download attacks before they infect their victims.

Finally, we discuss possible limitations of our SE attacks study and detection approach in Sections 7, and contrast our work to previously published research in Section 8.

3 SE Download Attacks

In this section, we analyze the range of deception and persuasion tactics employed by the attackers to victimize users (Section 3.1). We also provide some concrete examples of SE download attacks, to highlight how real users may fall victim to such attacks (Section 3.2).

SE Attacks Dataset. Our analysis is based on a dataset consisting of 2,004 real-world SE download attacks. We collected these attacks by monitoring the network traffic of a large academic network (authorized by our organization’s IRB), passively reconstructing the download of executable binary files and tracing back the browsing path followed by the users to reach the file download event. We then analyzed the observed file download events to identify possible malware, adware or PUP downloads. Finally, we performed an extensive manual analysis of the suspicious downloads to identify and label those downloads that were triggered by SE attacks, and to precisely reconstruct the attack scenarios. A detailed description of our dataset collection and labeling approach is provided in Section 4. Furthermore, in Section 5 we measure properties of the collected attacks, such as what types of SE attacks are the most prevalent, and provide information on the network-level characteristics of SE download distribution operations.

In the following, we will focus on analyzing our SE download attack dataset with the goal of categorizing the different types of deception and persuasion tactics used by attackers to lure victims into downloading malicious and unwanted software.

3.1 Categorizing SE Download Tactics

The dataset of 2,004 SE download attacks that we reconstructed and labeled via extensive manual analysis efforts (detailed in Section 4) gives us an excellent opportunity to study the wide range of depiction and persuasion tactics employed by the attackers. To better understand how SE attacks work, we develop a categorization system that aims to provide a systematization of the techniques used by successful SE download attacks. Specifically, we categorize different SE attacks according to; (1) the ways the adversaries get the user’s attention and (2) the type of deception and persuasion tactics employed. Our categorization of SE attacks is summarized in Figure 1.

Gaining the user’s attention. The first step in a SE attack is to get the user’s attention. This is accomplished for example by leveraging online advertisement (ad), search engine optimization (SEO) techniques or by posting messages (and clickable links) on social networks, forums, and other sites that publish user-generated content.

As we will show in Section 5, the most popular of these methods is *ads*. On-line advertisement allows the attacker to easily “publish” their deception/persuasion ad on a site that is likely already popular among the targeted victims. In addition, ads help hide the deception/persuasion campaign and attack infrastructure (i.e., hide it from users as well as security researchers), simply because SE ads are exposed only to targeted users via search keywords, the user’s cookies, etc.

Another method employed to attract the user’s attention is *search*. For instance, search engines can be abused via black hat SEO attacks to pollute the search results with harmful links. In addition, in our categorization of SE attacks we use a generic definition of “search” that does not only include search engines; anytime a user perform a query to locate specific content on a website, we classify it as a search event. For instance, we have observed users that become victims of SE attacks while simply searching for content within a website that hosts video streaming (e.g., movies, video clips, etc.).

Attackers also use *web posts* to attract the user’s attention. We define a web post as content that has been added to a website by a visitor and is now available for display to others. For instance, many of the web posts used in the SE download attacks we observed were located within groups of legitimate posts about a topic of interest. The majority of such web posts were related to content (e.g.,

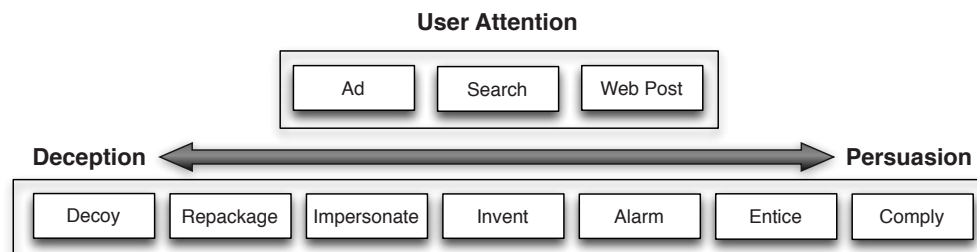


Figure 1: Categorization of SE downloads on the web.

clickable links) such as free software, books, music and movies.

It is not uncommon for these three techniques (ads, search, and web posts) to be combined. For instance, attackers will use search and ads in combination to get the user's attention. Targeted search engine ads related to the search terms entered by users are often displayed before the real search results, thus increasing the likelihood of a click. This common search engine feature is often abused by attackers. Also, users may search for certain specific terms on web forums, social network sites, etc., and may fall victim to targeted web posts.

Deception and persuasion tactics. After an attacker gains the user's attention, they must convince them to download and install their malicious or unwanted software. This typically involves combining a subset of the deception and persuasion techniques summarized in Figure 1. As one scrolls from left to right in the figure, the techniques move from deception towards persuasion. Notice that none of the techniques we list involve only deception or only persuasion; instead, the different techniques vary in their levels of each. We now provide a description of the deception and persuasion classes shown in Figure 1. We will then present examples of real-world SE download attacks that make use of a combination of these techniques.

- (1) **Decoy:** Attackers will purposely place decoy “clickable” objects, such as a hyperlink, at a location on a web page that will attract users to it and away from the actual object desired (or searched) by the user. An image of a “flashy” download button (e.g., delivered as an ad banner) on a free download site located prior to the actual download link desired by the user is an example of this technique.
- (2) **Repackage:** To distribute malicious and unwanted software, attackers may group benign and PUP (or malware) executables together, and present them to the user as a single application. An example technique from this class is adware bundled with a benign application and served as a single software download on a free software distribution website.
- (3) **Impersonate:** Using specific images, words and colors can make an executable appear as if it was a

known popular benign application. Also, claiming that a software provides desirable features or services (though it does not supply them) is a way to convince the user to download and install the application. Malicious executables pretending to be an Adobe Flash Player update, e.g., by using logos or icons similar to the original Adobe products and keywords such as “adobe” and “flash,” is an example of impersonation techniques from this class.

- (4) **Invent:** Creating a false reality for the user may compel them to download a malicious or unwanted executable. For example, alerting the user stating that their machine is infected with malware and instructing them to download (malicious) clean-up software (e.g., a fake AV) is an example of the *invent* tactic.
- (5) **Alarm:** Using fear and trepidation aims to scare the user into downloading (malicious) software that promises to safeguard them. For instance, an online ad claiming that the user's browser is out-of-date and vulnerable to exploitation is an example of *alarm* techniques.
- (6) **Entice:** Attackers often attempt to attract users to download a malicious or unwanted executable by offering features, content or advantages. As an example, a user may be shown an ad for a system optimization utility stating that it will “speedup” their PC, but hides malicious software.
- (7) **Comply:** A user may be (apparently) required to install an (malicious) application before she can continue. For instance, a user visiting a video streaming website may be prompted to install a necessary “codec” before she can watch a free movie. As the user is motivated to watch the movie, she complies with the codec installation request, thus getting infected with malware.

It is important to note that none of the SE attacks in our study fall into a single class. Instead the in-the-wild SE attacks we collected often use techniques across two or more of the above classes to trick the user into infecting their machine. Labeling a download using these classes involves understanding the motivations employed to convince a user to install the malicious software. These are

typically easy to identify by examining the words and images used in an attack. For instance, an attack that *impersonates* will claim to be software that it is not, such as Adobe Flash Player. On the other hand, an attack that *entices* a user will often use words like “free” and describe all the benefits of installing the software. Entice and impersonate are not mutually exclusive and are used together in some SE attacks. Allowing an SE attack to be assigned to more than one class simplifies the labeling process because all perception/deception tactics can be included, not just the one believed to be the primary tactic.

3.2 Examples of In-The-Wild SE Attacks

In this section, we present two examples from our dataset of reconstructed SE download attacks, and classify their SE tactics using our categorization system (see Figure 1). To aid in our discussion we define the notation “*attention:deception/persuasion*,” where the *attention* string refers to how attackers attempt to attract users’ attention, and the *deception/persuasion* string refers to the combination of the deception/persuasion techniques used to trigger the malware download. For example, if an SE attack relies on an *ad* and uses *alarm* and *impersonate* deception/persuasion tactics, then we label the attack using our notation as “ad:alarm+impersonate.”

Attack 1. A user searches for “Gary Roberts free pics” using a popular search engine. A page hosted on a compromised website is returned as a top result. The page contains various content referring to “Gary Roberts”, but this content is incoherent and likely only present for blackhat search engine optimization (SEO). However, the user never sees the content because the javascript code located at the top of the served page immediately closes the document, and then reopens it to inject a script that redirects the user to a page that says “gary-roberts-free-pics is ready for download. Your file download should automatically start within seconds. If it doesn’t, click to restart the download.” But the downloaded file does not contain any pictures, and instead carries malicious code that is later flagged as malware by some AV vendors.

Using our categorization system we classify this attack as “search:entice+decoy+impersonate.” *Search* is the method of gaining the users attention. In this example this is obvious because the SE page appeared in the results page provided by a search engine. The *entice* part of the attack is the offering of “free” pics of the subject of interest. *Decoy* is due to the fact that blackhat SEO was used to elevate the SE page in the search results above other legitimate pages. Lastly, what the user downloads is not pics of Gary Roberts; instead, it is a malicious executable *impersonating* what the user wants (e.g., Gary



Figure 2: SE ad for Ebola early warning system.

Roberts free pictures).

Attack 2. A user is watching an episode of “Agents of Shield” on a free video website when they are presented with an *ad*. The ad, similar to the one shown in Figure 2, presents the user with the option of downloading an early warning system for Ebola. However, the downloaded file does not provide information about an Ebola outbreak; instead, it infects the user’s system with malicious software.

We classify this attack as “ad:alarm+impersonate” using our categorization system. The user’s attention is gained through an *ad*, in which their fear of Ebola is used to *alarm* the user into downloading a tracking system. Unfortunately, what the user downloads only *impersonates* a tracking systems, and instead contains malicious code.

4 Collecting and Labeling SE Attacks

In this section we discuss in detail how we collected and labeled our dataset of 2,004 SE download attacks. We will then present an analysis of the prevalence and characteristics of the collected attacks in Section 5.

4.1 Data Collection Approach

To collect and reconstruct SE download attacks, we monitor all web traffic at the edge of a large network (this study was authorized by our organization’s Institutional Review Board). Using deep packet inspection, we process the network traffic in real-time, reconstructing TCP connections, analyzing the content of HTTP responses and recording all traffic related to the download of executable files (Windows executables).

While monitoring the traffic, we maintain a buffer of all recent HTTP transactions (i.e., request-response pairs) that occurred in the past few minutes. When an HTTP transaction that carries the download of an executable file is found, we passively reconstruct and store a copy of the file itself. In addition, we trigger a dump of the traffic buffer, recording all the web traffic generated

by the same client that initiated the file download during the past few minutes before the download started. In other words, we store all HTTP traffic a client generated up to (and including) the executable file download.

We then process these HTTP transaction captures using the trace-back algorithm presented in [30]. The trace-back algorithm builds a graph where each node is an HTTP transaction. Given two nodes T_1 and T_2 , they are connected by an edge if T_2 was “likely referred to” by T_1 . For instance, a directed edge is drawn from T_1 to T_2 if the user clicked on a link in T_1 ’s page and as a consequence the browser loaded T_2 . Then, starting from the download node, the algorithm walks backwards along this graph to trace back the most likely path (i.e., sequence of HTTP transactions) that brought the user to initiate the download event. For more details, we refer the reader to [30].

These reconstructed download paths are later analyzed to identify and categorize SE download attacks. It is important to note that we do not claim automatic download path traceback as a contribution of this paper. Instead, our focus is on the collection, analysis, and categorization of SE download attacks, and on the detection and mitigation of ad-based SE infections. Automatic download traceback is just one of the tools we use to aid in our analysis.

We deployed the data collection process described above on a large academic network serving tens of thousands users for a period of two months. To avoid unnecessarily storing the download traces related to frequent software updates for popular benign software, we compiled a conservative whitelist consisting of 128 domain names owned by major software vendors (e.g., Microsoft, Adobe, Google, etc.). Therefore, executable files downloaded from these domains were excluded from our dataset.

Overall, during our two month deployment, we collected a total of 35,638 executable downloads. The process we used to identify the downloads due to SE attacks is described in the following sections.

4.2 Automatic Data Filtering

Even though we filter out popular benign software updates up front, we found that the majority of executable downloads observed on a network are updates to (more or less popular) software already installed on systems. As we are interested in new infections caused by web-based SE attacks, we aim to automatically identify and filter out such software updates.

To this end, we developed a set of conservative heuristics that allow us to identify and filter out most software update events based on an analysis of their respective download path. First, we examine the length of the download path. The intuition is that software updates

tend to come from very short download paths, which often consist of a single HTTP request to directly fetch a new executable file from a software vendor’s website (or one of its mirrors). Conversely, the download path related to SE download attacks usually consists of a number of navigation steps (e.g., the user may navigate through different pages before stumbling upon a malicious SE advertisement).

For the next step in the analysis, we review the user-agent string observed in the HTTP requests on the download path. The user-agent string appearing in software update requests is typically not the one used by the client’s browser (similar observations were made by the authors of [30]), because the user-agent found in these requests often contains the name of the software that is being updated (e.g., Java or Acrobat reader). Since web-based SE attacks happen to users browsing the web, HTTP requests on the download path typically carry the user-agent string of the victim’s browser.

Therefore, to automatically identify and filter out update downloads we use the following heuristics. If the download path contains a single HTTP transaction (the update request itself), and the user-agent string does not indicate that the request has been made by a browser, we filter out the event from our dataset.

Overall, the conservative filtering approach outlined above allowed us to reduce the number of download paths to be further analyzed. Specifically, we were able to reduce our download traces dataset by 61%, leaving us with a total of 13,762 that required further analysis and labeling.


4.3 Analysis of Software Download Events

After filtering, our dataset consists of 13,762 software download events (i.e., the downloaded executable files and related download paths) that required further detailed analysis and labeling. As our primary goal is to create a high quality dataset of labeled SE download attacks, we aim to manually analyze and perform a detailed reconstruction of the attacks captured by our archive of software download events.

To aid in the manual analysis process and reduce the cost of this time-consuming effort, we leveraged unsupervised learning techniques. Specifically, we identify a number of statistical features that allow us to discover clusters of download events that are similar to each other. For instance, we aim to group different downloads of the same benign software by different clients. At the same time, we also aim to group together similar download events triggered by the same SE attack campaign.

To identify and automatically cluster similar download events, we developed a set of statistical features. We would like to emphasize that none of the features we describe below is able to independently yield high-quality

clustering results. Rather, it is the combination of these features that allows us to obtain high quality clusters of related software download events.

 Notice also that the purpose of this clustering process is simply to reduce the time needed to manually analyze the software download events we collected. By using a conservative clustering threshold (discussed below) and by manually reviewing all obtained clusters, we minimize the impact of possible noise in the results.

To perform the clustering, we leverage a number of simple statistical features, some of which (e.g., URL similarity, domain name similarity, etc.) are commonly used to find the similarity between network-level events. Notice, however, that our main goal in this clustering process is not to design novel features; rather, we simply aim to reduce the manual analysis and labeling efforts needed to produce a high-quality dataset of in-the-wild SE download attacks.

We now describe our clustering features:

- (1) **Filename Similarity:** Benign executable files distributed by the same organization (e.g., an application distributed by a given vendor or software distribution site) tend to have similar filenames. Notice that often this also holds for SE attack campaigns, because the files distributed by the same campaign often follow a consistent “theme” to aid in the *deception* of the end users. For instance, the malware files distributed by a fake Flash Player upgrade attack campaign (see Section 3) may all include the word “Adobe” in the filename to convince the user that the downloaded file is legitimate.
- (2) **File Size Similarity:** Benign files that are identical or variants (i.e., different versions) of the same software are usually very close in size. Similarly, SE campaigns typically infect victims with a variant of the same malware family. While the malware file’s size may vary due to polymorphism, the size difference is typically small, compared to the total file size.
- (3) **URL Structure Similarity:** A benign website that serves software downloads will often host all of its executable files at the same or very similar structured URLs. In a similar way, SE campaigns often use malware distribution “kits” and go weeks or even months before a noticeable change in the structure of their URL paths is observed. This is unlike malicious URLs, which frequently change to avoid blacklisting.
- (4) **Domain Name Similarity:** While the domain names used to distributed malware files belonging to the same SE attack campaign may change, some campaigns will reuse some keywords in their domains that are meant to deceive the user. For instance, the domains used in a Fake AV malware campaign may contain the keyword “security” to convince the user of its legitimacy. Also, download events related to (different versions of) the same benign software are often distributed via a handful of stable domain names.
- (5) **Shared Domain Predecessor:** SE attacks that share a common node (or predecessor) in the download path are often related. For instance, an SE malware campaign may exploit an ad network with weak anti-abuse practices. Therefore, while the final domain in the download path from which the malware is actually downloaded may change (e.g., to avoid blacklisting), the malware download *paths* of different users that fall victim to the same SE campaign may share a node related to the abused ad network, for example. On the other hand, in case of benign downloads both the download and “attention grabbing” domain tend to be stable, as the main goal is quality of service towards the end user.
- (6) **Shared Hosting:** While domains involved in malware distribution tend to change frequently, SE malware campaigns often reuse (parts of) the same hosting infrastructure (e.g., some IPs). The intuition is that hosting networks that tolerate abuse (knowingly or otherwise) are a rare and costly resource. On the other hand, domain names are significantly easier to obtain and can be used as crash-and-burn resource from the adversary. On the benign downloads side, legitimate software distribution websites are usually stable and do not change hosting very frequently, for quality of service reasons.
- (7) **HTTP Response Header Similarity:** The headers in an HTTP response are the result of the installed server-side software and configuration of the web server. The set of response headers and their associated values offer a lot of variation. However, most of the web servers for a benign site tend to have common configurations so they respond with similar headers. Also, some SE campaigns use the same platform for their attacks and do not change their server-side configurations even when they move to new domains.

For each of the 13,762 downloads we compute a feature vector based on the features listed above, and calculate the pairwise distance between these feature vectors. We then apply an agglomerative hierarchical clustering algorithm to the obtained distance matrix. Finally, we cut the dendrogram output by the hierarchical clustering algorithm to obtain the final clusters of download events. To cut the dendrogram we chose a conservative cut height to error on the side of not grouping related downloads and significantly reduce the possibility of grouping unrelated ones. This process produced 1,205 clusters, thus resulting in an order of magnitude reduc-

tion in the number of items that require manual inspection. In the following section we explain how we analyzed and labeled these clusters.

4.4 Labeling SE Download Attacks

After clustering similar software download events, we manually examine each cluster to distinguish between those that are related SE download attack campaigns, and clusters related to other types of software download events, including benign downloads, malware downloads triggered by drive-by downloads, and (benign or malicious) software updates. This labeling process allows us to focus our attention on studying SE download attacks, and to exclude other types of benign and malicious downloads (notice that because in this paper we are primarily interested on SE attacks, we exclude non-SE malware attacks from our study).

To perform the cluster labeling, each cluster was manually reviewed by randomly sampling 10% of the download events grouped in the cluster, and then performing a detailed manual analysis of the events in this sample set. For small clusters (e.g. clusters with < 50 events) we sampled a minimum of 5 download events. For clusters containing less than 5 download traces, we reviewed all of the events. As discussed earlier, our clustering process uses a conservative cut height. The net effect is that the clusters tend to be “pure,” thus greatly reducing the possibility of errors during the cluster labeling process. At the same time, some groups of download events that are similar to each other may be split into smaller clusters. However, this does not represent a significant problem for our labeling process. The only effect of having a larger number of highly compact clusters is to create additional manual work, since a random sample of events from each cluster is manually analyzed.

In addition to manually reviewing the download paths contained in the clusters, to assist our labeling we also make use of antivirus (AV) labels for the downloaded executable files. To increase AV detections we “aged” the downloads in our dataset for a period of two months, before scanning them with more than 40 AV engines using virustotal.com. Notice that AV labels are mainly used for confirmation purposes. The actual labeling of SE attacks is performed via an extensive review of each download path (i.e., sequence of pages/URLs visited to arrive to the executable file download). If we suspect a cluster is malicious (based on our manual analysis), having one or more AV hits offers additional confirmation, but is not required if we have strong evidence that the download was triggered by an SE attack.

Updates: Even though the heuristics we described in Section 4.2 filter out the vast majority of software updates, our heuristics are quite conservative and therefore some update events may still remain. To determine if

a download event is related to a (malware or benign) update, we examine the length of the download path and the time between requests. If the length of the total download path is < 4 or the time between requests is < 1 second, we consider the download event for detailed manual review. In this case, we analyze the HTTP transactions that precede the download by examining the content for artifacts, such as text and clickable buttons, that are indicators of human interaction. If none are found we label the download as an *update*.

Drive-by: Next we look for drive-by download indicators. To assist our labeling, we borrow some of the features proposed in [30]. Notice that the labeling of drive-by downloads is *not* a contribution of our paper. This is only a means to an end. The novel contributions of this paper are related to studying the characteristics of SE download attacks.

To label drive-by attacks, we look for “exploitable content,” such as pdf, flash, or java code on the path to a malware download. Browser plugins and extensions that process this type of content often expose vulnerabilities that are exploited by attackers. If we suspect the download event under analysis is a drive-by, we reverse engineer the content of the HTTP transactions that precede the suspected attack. This typically requires deobfuscating javascript and analyzing potentially malicious javascript code. For instance, if we identify code that checks for vulnerable versions of browser software and plugins (an indication of “drive-by kits”), we label the download as *drive-by*. We identify and label 26 drive-by downloads.

Social Engineering: If the cluster is not labeled as *update* or *drive-by*, we further examine the download events to determine if they are due to SE attacks. For this analysis, we inspect the content of all HTTP transactions on the download path. This content was saved at the time of the download and does not require revisiting of the URLs on the download path. Because SE downloads are attacks on the user, they are initiated by a user-browser interaction (e.g., clicking a link). Therefore, our goal is to identify the page on the download path containing the link likely clicked by the user that ultimately initiated the executable file download. By manually reviewing the web content included in the download path, we attempt to determine if *deception* or *questionable persuasion* tactics were used to trick the user into downloading the executable file (see Section 3). In case of positive identification of such tactics, we label the cluster as *social engineering*; otherwise, we label it as “likely” benign.

Notice that the analysis and labeling of SE download attacks is mainly based on the identification of deception tactics to trick a user to download an executable file. However, we also use AV scanning for confirmation pur-

poses. By doing so, we found that the majority of the clusters we label as social engineering contained one or more of downloaded files that were labeled as malicious by some AVs. This provides additional confirmation of our labeling of SE download attacks.

Overall, among 1,205 clusters in our dataset, we labeled 136 clusters as *social engineering*. In aggregate, these clusters included a total of 2,004 SE download attacks. In Section 3 we analyzed these SE download attacks and developed a categorization system that allows us to organize these attacks based on the deception and persuasion tactics used to attack the user. In the next section, we measure the popularity of these tactics based on the data we collected.

5 Measuring SE Download Attacks

In this section we measure the popularity of the tactics attackers employ to gain the user’s attention and of the deception and persuasion techniques that convince users to (unknowingly) download malicious and unwanted software. In addition, we measure properties of ad-based SE download attacks, which can be used to inform the development of effective defenses against SE attacks that leverage ad campaigns.

5.1 Popularity of SE Download Attacks

Table 1 shows the number and percentage of SE download attacks for each tactic employed by the attackers to gain the user’s attention. Over 80% of the SE attacks we observed used ads displayed on websites visited by the user. An additional 7% employed both *search* and *ad*, whereby the user first queries a search engine and is then presented with targeted ads based on the search terms. The popularity of ads in SE download attacks is likely due to the fact that they are a very efficient way for attackers to reach a large audience, thus maximizing the number of potential victims. Furthermore, well-crafted targeted ads are naturally highly effective at attracting a user’s attention.

Table 1: Popularity of SE techniques for gaining attention.

User’s Attention	Total	Percentage
Ad	1,616	80.64%
Search+Ad	146	7.29%
Search	127	6.34%
Web Post	115	5.74%

Gaining the user’s attention is not sufficient for an SE download attack to succeed. A user must also be tricked into actually “following the lead” and downloading the malicious or unwanted software. Table 2 shows the popularity of the deception and persuasion techniques we observed in our dataset of SE download attacks. The most

popular combination of deception and persuasion techniques is *repackage+entice*, making up over 48% of the observations. In most of these cases, the user is offered some type of “free” software of interest (e.g., a game or utility). Unfortunately, while the free software itself may not be malicious, it is often bundled with malicious applications such as *adware* or PUPs.

Table 2: Popularity of SE techniques for tricking the user.

Trick	Total	Percentage
Repackage+Entice	972	48.50%
Invent+Impersonate+Alarm	434	21.66%
Invent+Impersonate+Comply	384	19.16%
Repackage+Decoy	155	7.74%
Impersonate+Decoy	46	2.30%
Impersonate+Entice+Decoy	12	0.60%
Invent+Comply	4	0.20%
Impersonate+Alarm	1	0.05%

The next two most popular combinations of deception and persuasion tactics are *invent+impersonate+alarm* and *invent+impersonate+comply*, comprising 22% and 19% of the SE downloads we observed. An example of *invent+impersonate+alarm* is a Fake Java update attack, whereby the user is shown an ad that states “WARNING!!! Your Java Version is Outdated and has Security Risks, Please Update Now!” and uses images (e.g., logos or icons) related to Java (notice that this and all other examples we use throughout the paper represent real-world cases of successful SE attacks from our dataset). Ads like this are typically presented to users while they are visiting legitimate websites. In this example, the attacker is *inventing* the scenario that the user’s Java VM is out-of-date, *alarming* them with “WARNING!!!” displayed in a pop-up ad, and then *impersonating* a Java update that must be installed to resolve the issue. Notice that this is different from *repackage+entice*, in that the real Java software update is never delivered (only the malware is). Furthermore, the attacker leverages alarming messages about a well-known software to more aggressively “push” the user to download and install malicious software.

The difference between *invent+impersonate+alarm* and *invent+impersonate+comply* is in the persuasion component; i.e., *alarm* vs. *comply*. *Alarm* leverages fear (e.g., the computer may be compromised) to compel the user to download and install malicious software. On the other hand, *comply* leverages a pretend requirement necessary to complete a desired user task. For instance, a user may be presented with an ad on a video streaming website that says “Please Install Flash Player Pro To Continue. Top Video Sites Require The Latest Adobe Flash Player Update.” In this example, the attacker is *inventing* the need to install “Flash Player Pro” and tells the user they must *comply* before they can continue with watching the desired video. Unfortunately, this results

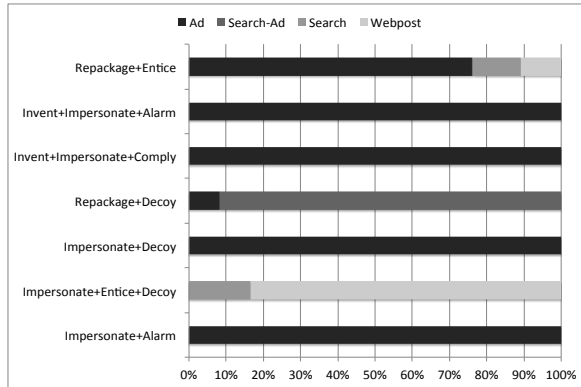


Figure 3: How attackers gain the user’s attention per deception/persuasion technique.

in the user downloading malicious software that simply *impersonates* a popular benign application and offers no actual utility to the user.

Table 3: Popularity of different “scam” tactics in the Ad:Invent+Impersonate subclasses.

	Alarm	Comply
Fake Flash	68%	20%
Fake Java	30%	0%
Fake AV	1%	0%
Fake Browser	1%	0%
Fake Player	0%	80%

Table 3 shows the popularity of different “scam” tactics in the invent+impersonate subclasses *alarm* and *comply*. Fake Flash and Java updates are the two most popular in the *alarm* class. In this same class we also observe Fake Browser updates and Fake AV alerts, but they are much less common, each comprising only about 1% of our observations. Fake Flash updates are also common in the *comply* class; however, the most popular scam tactic is telling the user they must update or install a new video player before they can continue. In these Fake Player attacks, images that resemble Adobe Flash Player are often used, but the terms “Adobe” or “Flash” are not directly mentioned. Therefore in Table 3 we distinguish between explicit Fake Flash and Fake Player.

Figure 3 shows how attackers gain the user’s attention for each of the observed deception and persuasion techniques. For instance, ads are the most common way used to attract users’ attention for repackage+entice, comprising 75% of our observations. *Search* and *web posts* contribute the remaining 25%. All observations for invent+impersonate+alarm, invent+impersonate+comply, impersonate+alarm and impersonate+decoy rely exclusively on ads to gain the user’s attention. At the other extreme, none of our observations for impersonate+entice+decoy use ads. This is likely due to the fact that this combination of deception and persuasion

techniques is more effective when the user’s attention is gained through *search* and *web posts*. However, notice that this comprises less than 1% of all SE downloads in our dataset (see Table 2).

5.2 Ad-based SE Download Delivery Paths

As shown in Table 1, the majority of SE attacks we observed use online ads to attract users’ attention. To better understand these attacks, we examine the characteristics of their *ad delivery path*. We begin by reconstructing the web path (i.e., the sequence of URLs) followed by the victims to arrive to the download URL (see Section 4). Then, we identify the first ad-related node on the web path using a set of regular expressions derived from the Adblock Plus rules [1]. We define the set of nodes (i.e., HTTP transactions) on the web path beginning at the first ad node and ending at the download node as the *ad delivery path*.

Table 4: Top five ad entry point domains.

Comply	Alarm	Entice
26% onclickads.net	16% adcash.com	20% doubleclick.net
10% adcash.com	7% onclickads.net	16% google.com
10% popads.net	7% msn.com	12% googleservices.com
7% putlocker.is	6% yesadsv.com	11% msn.com
3% allmyvideos.net	4% yu0123456.com	8% coupons.com

Table 4 shows the top 5 “entry point” domain names on the ad delivery paths (i.e., the first domain on the ad paths) for the *comply*, *alarm* and *entice* attack classes. Almost 50% of the ad entry points for the *comply* class begin with one of the following domains: `onclickads.net`, `adcash.com` or `popads.net`. By investigating these domains, we found that they have also been abused by adware in the past. Specifically, these domains are the source of pop-up ads injected into the user’s browsing experience by several well known adware programs and ad-injecting extensions [43].

Table 4 also shows that the top two ad entry points for the *alarm* class are the same as the *comply* class, though in reverse ranking. The third domain is `msn.com`, which has a good reputation. However, this domain is appearing at the top of the ranking probably because it sometimes redirects (via syndication) to less reputable ad networks, which in turn direct the user to an SE download. Notice also that the top entry domains in the *entice* class all have very good reputations (`doubleclick.net` is owned by Google). This is likely due to the fact that the majority of downloads in this class are for legitimate software that is simply bundled with “less aggressive” PUPs.

Besides performing an analysis of the “entry point” to the ad delivery path, we also analyze the last node on the path, namely the HTTP transaction that delivers the download. Table 5 shows the most popular SE download domains for the *comply*, *alarm* and *entice* classes. We

Table 5: Top five ad-driven SE download domains.

Comply	Alarm	Entice
17% softwaare.net	7% downloaddabs.com	41% imgfarm.com
5% newthplugin.com	4% downloaddado.com	17% coupons.com
5% greatsoftfree.com	4% whensoftisupdated.net	11% shopathome.com
4% soft-dld.com	3% safesystemupgrade.org	5% crusharcade.com
3% younplugin.com	3% onlinelivevideo.org	3% ilivid.com

found that the domains listed for the *comply* and *entice* classes serve mostly adware and PUPs.

To better understand the network-level properties of SE downloads, we also measure the “age” of these domains by leveraging a large passive DNS database (pDNS-DB) that stores historic domain name resolutions. Specifically, we define the domain age as the difference in days from the time the domain was first queried (i.e., first recorded in the pDNS-DB) to the day of the download. All the domains in Table 5 that are part of the *comply* and *alarm* classes are less than 200 days old, with the majority being less than 90 days. On the other hand, the domains in Table 5 for the *entice* class are all at least several years old. This is because most of the downloads in this class are for legitimate software that is simply bundled with adware or PUPs. For instance, we find a large variety of “free” software ads that direct users to the domain `imgfarm.com` for download. This is the reason that over 40% of the downloads in the *entice* class are from that domain.

The “middle of path” domains, namely the ones between the ad entry point and the download domain itself, tend to be a mix of recent and old domains. In fact, the most popular *comply* and *alarm* class “middle of path” domains are a 50/50 split of recent and old. However, this is not the case for the *entice* class, for which most domains are several years old. At the same time, the majority of ad delivery paths for all three classes include at least one middle domain name with an age that is less than 200 days.

5.3 Ad-Driven Benign Downloads

As mentioned earlier, more than 80% of the SE download attacks we observed use ads to gain the user’s attention (see Table 1). Based on common experience, it may seem unlikely that many benign software downloads would result from clicking on an ad. As a result, one may think that if software is downloaded after clicking on an ad, that software is unlikely to be benign. Somewhat surprisingly, we found that this may not necessarily be the case, as we explain below.

First, to automatically identify ad-driven benign software downloads, we first derive a set of ad detection regular expressions from the rules used by the popular AdBlock Plus browser extension [1]. We match these regular expressions against the nodes on the reconstructed download path for each benign download (the down-

load labeling process is described in Section 4.4). If an AdBlock rule matches the download path, we label that benign software download as *ad-driven*. We find that around 7% of all benign software downloads are ad-driven. Even though the percentage is low compared to SE downloads, in absolute terms this number is significant because the vast majority of software downloads observed in a network are benign. In fact, by considering the overall number of confirmed malware and benign software downloads and how many of these downloads are ad-driven, we find that if a software download is reached by clicking on an ad there is a 40% chance that that software is benign.

Table 6: Ad-based benign download popularity.

Category	Percentage
Games	32%
Utilities	30%
Music	15%
Business	11%
Video	8%
Graphics	2%
Social	2%

To further understand what type of benign software downloads are ad-driven, we investigate through manual analysis 100 randomly selected samples of benign software download events. Table 6 shows the type of software that is represented in our sample and their relative popularity. Games and utilities are the most popular categories, comprising 62% of all downloads. For example, the game “Trion Worlds” is the most popular with 17 downloads, followed by “Spotify” with 10 downloads.

6 Detecting SE Download Attacks

In this section, we measure the antivirus (AV) detection rate for SE downloads and group them into broad malware classes using AV labels. Also, we show that it is possible to accurately detect SE download attacks by constructing a statistical classifier that uses features derived from the SE attack measurements we presented in Section 5.

6.1 Antivirus Detection

To assess how AV products cope with SE downloads, we scan each SE download sample in our dataset with more than 40 AV engines (using VirusTotal.com). We scanned each of the samples on the day we collected them. Then, we also “aged” the samples for a period of one month, and rescanned them with the same set of AV engines. If at least one of the top five AV vendors (w.r.t. market share) and a minimum of two others detect it, we label the executable as malicious.

We first group malicious downloads into three broad classes, namely *malware*, *adware* and *PUP*. These

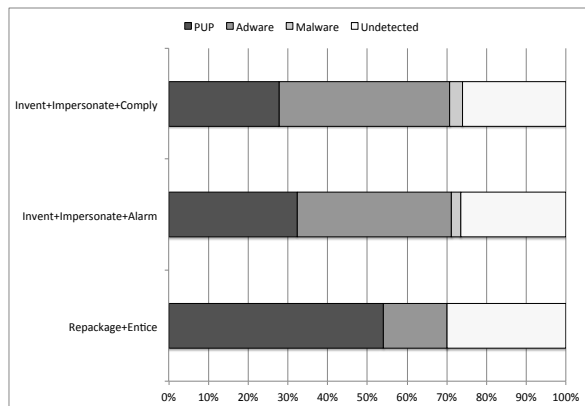


Figure 4: AV detections one month after download.

classes are meant to roughly indicate the potential “aggressiveness” of the malicious software. We assign these class labels based on a conservative set of heuristics that consist of matching keywords on the labels provided by the AVs. For instance, to identify adware we look for the string “adware” as well as the names of several popular adware applications (e.g., “amonetize,” etc.). Similarly, for the PUP class we look for the strings such as “PUP”, “PUA” and popular PUP application names. If both PUP and adware match, we label the sample based on a majority voting rule (across the different AV vendor labels). In the case of a tie, we conservatively label the sample as PUP. The remaining samples are labeled as malware, and manually reviewed for verification purposes.

Figure 4 shows the percentage of attacks that result in the download of malware, adware, and PUP, respectively. We show a breakdown for the top three deception/persuasion categories, which in aggregate represent more than 95% of all ad-driven SE attacks. The majority of malicious downloads in the invent+impersonate+comply and invent+impersonate+alarm deception/persuasion tactics belong to the *adware* class. A smaller percentage of downloads in these categories, 3.2% and 2.4% respectively, are labeled as *malware*. For repackage+entice the majority of downloads are labeled as *PUP*; no malware is found in this deception/persuasion category.

Notice that the relatively small percentage of *malware* is likely due to the fact that our heuristics for grouping malicious downloads into broad classes is very conservative, because it requires only a single AV label to contain an “adware” or “PUP” string to label the sample as belonging to the adware or PUP classes (see Section 4.4). In addition, adware is simply much more prevalent than malware, making the malware set size look relatively small compared to the rest of the dataset. Lastly, notice that only 70% – 75% of all malicious executables were labeled by the AVs, even after “aging” and rescanning the

samples. The remaining 25% – 30% is therefore labeled as *undetected* in Figure 4.

Table 7 shows the AV detection rate on the day of download for malware, adware and PUPs. Namely, we consider all SE download samples that were detected by AVs after aging and rescanning (i.e., after one month), and show the percentage of these samples that were also detected on the day we first observed them being downloaded. As can be seen, the detection rate in this case is quite small, thus confirming the reactive nature of AV detection.

Table 7: Zero day AV detections.

	Malware	Adware	PUP
Invent+Impersonate+Comply	0%	0%	6.9%
Invent+Impersonate+Alarm	5.9%	11.6%	26.9%
Repackage+Entice	NA	28.7%	34.4%

6.2 SE Detection Classifier

Guided by our measurements around SE attacks that we presented in Section 5, we devise a set of statistical features that can be used to accurately detect ad-driven SE downloads. We focus on ad-driven attacks because they are responsible for more than 80% of all SE downloads we observed (see Table 1).

Problem Definition. Given an executable file download event observed on the network, we first automatically reconstruct its *download path*, i.e., the sequence of pages/URLs the user visited to arrive to the HTTP transaction carrying the file, as explained in Section 4. Then, given a set of labeled ad-driven SE download events (Section 4.4) and also ad-driven benign software downloads (Section 5.3), we first *translate the download path of each event into a vector of statistical features*. Finally, we use the obtained labeled dataset of feature vectors to *train a statistical classifier using the Random Forest [10] algorithm* that can detect future ad-driven SE download attacks and distinguish them from benign ad-driven software downloads.

Statistical Features. We now present the set of detection features we derived and the intuitions behind their utility. In the following, we assume to be given as input the download path related to a software download event observed on the network, which we translate into a feature vector. Notice that no single feature by itself enables accurate detection; it’s their combination that allows us to reach high accuracy.

- **Ad-Driven** (binary feature). We check whether the download path contains an ad-related URL. This feature is computed by matching Adblock [1] rules against the sequence of URLs on the download path. *Intuition:* while the majority of SE downloads are

promoted via advertisement (Section 5), only 7% of benign downloads result from clicking on an ad (Section 5.3).

- **Minimum Ad Domain Age.** We measure the age of each domain on the *ad path*, namely the subsequence of the download path consisting of ad-related domains, and use the minimum age across these domains. *Intuition:* ad-serving domains that consistently direct users to malicious ads are often blacklisted, so they move to new domains. In essence, this feature is a way of (approximately) measuring the reputation of the *ad path*. Our measurements show that the majority of ad paths for the *comply*, *alarm* and *entice* attack classes all have domains less than one year in age. For benign download paths, this is true in only less than 5% of the cases.
- **Maximum Ad Domain Popularity.** Using our dataset (Section 4), we first consider all ad-related domains involved in the download paths observed in the past (i.e., in the training set). Then, for each domain, we count the number of distinct download paths on which the domain appeared, for both ad-driven SE attacks and the benign download paths. If the domain is found in more than 1% of the benign download paths, it is discarded. Otherwise, we compute the number of distinct SE attack paths in which the domain appeared. Finally, given all ad-related domains in the download path we are currently considering, we take maximum number of times a domain along this path appeared in an SE attack path. *Intuition:* some ad networks, and the domains from which they serve ads, are more abused than others, e.g., due to scarce policing of ad-related fraud and abuse in lower-tier ad networks. Therefore, they tend to appear more frequently in the download path of SE downloads. For instance, Table 4 in Section 5.2 shows the popularity of *ad entry points* for SE downloads.
- **Download Domain Age.** We measure the number of days between the download event and the first time we observed a DNS query to the effective second level domain for the download URL (final node of the web path) using a large historic passive DNS database. *Intuition:* the vast majority of benign downloads are delivered from domains that have been active for a long time because it takes time for a website to establish itself and attract visitors. On the other hand, SE domains are often “young” as they change frequently to avoid blacklisting. Our data shows that the download domain of over 80% of the invent+impersonate SE subcategories *comply*

and *alarm* are less than one year in age, whereas for benign download this only holds in 5% of the cases.

- **Download Domain Alexa Rank.** We measure the Alexa rank of the domain that served the software download. We compute this features using the effective second level domain for the download URL and the Alexa top 1 million list. *Intuition:* malicious executables are more likely to be hosted on unpopular domains because of their need of avoiding blacklisting. Conversely, benign software downloads are often hosted on popular domains. For instance, measurements on our data show that over 60% of the benign downloads are from domains with an Alexa rank in the top 100,000. On the other hand, the more “aggressive” SE downloads, such as those from the *alarm* class, are primarily delivered from very unpopular domains (very few are in the top 1 million). At the same time, the domains involved in SE attacks that trigger the download of PUP fall somewhere between, in terms of domain popularity.

6.3 Evaluating the SE Detection Classifier

In this section, we present the results of the evaluation of our SE detection classifier. We start by describing the composition of the training and test dataset, and then present an analysis of the false and true positives.

Datasets. To measure the effectiveness of the SE classifier, we use two separate datasets. The first dataset, \mathcal{D}_1 , which we use to train the classifier, consists of the software downloads described and measured in Sections 4 and 5. Specifically, this dataset includes 1,556 SE download paths (we consider all ad-driven SE attacks from the dataset described in Section 4), and 11,655 benign download paths.

The second dataset, \mathcal{D}_2 , consists of new executable downloads (and their reconstructed download paths) that we collected from the same deployment network in the three months following the completion of the measurements we presented in Section 5. Notice also that, \mathcal{D}_2 was collected *after* the feature engineering phase and after building our detection classifier. Namely, both the feature engineering and the training of the classifier were completed with no access to the data in \mathcal{D}_2 . Overall, \mathcal{D}_2 contains 1,338 ad-driven SE downloads, and 9,760 benign downloads paths. We label \mathcal{D}_2 following the steps outlined in Section 4.4.

Classification Results. After training our SE detection classifier using dataset \mathcal{D}_1 and the Random Forest learning algorithm, we test the classifier on dataset \mathcal{D}_2 .

Table 8 reports the confusion matrix for the classification results. The classifier correctly identified over 91% of the ad-driven SE downloads. Furthermore, it has a very low false positive rate of 0.5%.

Table 8: Confusion matrix for the SE detection classifier.

	Predicted Class	
	Benign	Ad-Based SE
Benign	99.5%	0.5%
Ad-Based SE	8.8%	91.2%

Table 9: SE Subclass Performance.

	True Positives
Repackage+Entice	65%
Invent+Impersonate+Alarm	98%
Invent+Impersonate+Comply	90%

Figure 9 shows a breakdown of the classification results for the subclasses of ad-based SE downloads. The *invent+impersonate+alarm* and *invent+impersonate+comply* categories have 98% and 90% true positive rates, respectfully. The lower performance for *repackage+entice* is due to downloads of legitimate software bundled with PUPs from well established domains. Because these domains are “mixed use,” and have high popularity or Alexa ranking, they make the detection task more difficult.

Feature Importance. We estimate feature importance by performing forward feature selection [20]. The single feature that provides the largest information gain is *download domain age*. Using only that feature we have a 69% true positive rate and a 6% false positive rate. By adding *maximum ad domain popularity*, we obtain a true positive rate above 80% with less than 3% false positives. As we add other features (using the forward search), both the true positives and false positives continue to improve. Thus, all the features help achieve high accuracy.

7 Discussion

In this paper, we focus exclusively on *successful* web-based SE download attacks (we consider the attacks we collected and study successful because they actually trigger the delivery of malicious software to the victim’s machine). Social engineering attacks carried over different channels (e.g., email) and that have different objectives (e.g., phishing attacks to steal personal information, rather than malware infections) are not part of our measurements, and are therefore also not reflected in the categories of SE tactics we described in Section 3. However, we believe ours is an important contribution. In fact, as defenses for drive-by downloads continue to improve (e.g., through the hardening of browser software and operating system defenses) we expect the attackers to increasingly make use of web-based SE attacks for malware propagation. Therefore, the reconstruction and analysis of SE download attacks is important because in-the-wild SE attack samples could be used to better train users and mitigate the impact of future attacks; thus, complementing automatic attack detection solutions.

Our study relies on visibility over HTTP traffic and

deep packet inspection. One might think that the inability to analyze HTTPS traffic represents a significant limitation. However, it is important to take into account the following considerations. When a user browses from an HTTPS to an HTTP site, they are often redirected through an unsecured intermediate URL, so that the *Referer* field can be populated with the domain and other information related to the origin site [3]. Alternatively, the origin site can set its referrer policies [2] to achieve the same result without need of intermediate redirections. As an example, even though many of the searches performed using search engines such as Google, Yahoo and Bing occurred over HTTPS, we were able to identify the search engines as the origin of web paths because the related domain names appeared in the *Referer* field of the subsequent HTTP transactions. Furthermore, modern enterprise networks commonly employ SSL man-in-the-middle (MITM) proxies that decrypt traffic for inspection. Therefore, our SE attack detection system could be deployed alongside SSL MITM proxies.

Throughout the study, we use the term *malicious* to describe the software downloaded as the result of an SE attack. However, there exist many shades of maliciousness and some malicious software (e.g., ransomware, botnets, etc.) are more “aggressive” than others (e.g., adware and PUPs). Therefore, in several parts of the analysis we broke down our results by distinguishing between *malware*, *adware* and *PUPs*. As shown in Section 6.1, only a relatively small percentage of the SE downloads collected for our measurements were categorized by our AV-label-based heuristics as *malware*. The majority were labeled as adware or PUP. However, we should notice that AV labels are known to be noisy and that our labeling heuristics are very conservative (see Section 4.4). Furthermore, over 25% of the malicious downloads remained unlabeled due to lack of AV detection (Section 6.1). Therefore, it is possible that the number of malware is somewhat higher than reflected in Section 6.1. However, the categorization system, network-level properties and detection results for SE attacks that deliver adware apply to attacks that result in malware downloads as well.

While the software downloads and traffic we collected for our study were collected from a single academic network, we should consider that the deployment network was very large, serving tens of thousands diverse users, consisting of users from different ages, cultures and backgrounds.

Because our SE detection classifier is designed to detect ad-based SE download attacks, an attacker could evade the system by using tactics other than online ads to attract the user’s attention (e.g., *search* or *web post*, as discussed in Section 3). However, advertisements are the

predominant tactic used by attackers because they allow them to “publish” their SE attacks on sites that already popular with the targeted victims. In addition, ads are only shown to the users that “match” their delivery criteria, thus reducing exposure to others (including security researchers) that could result in the discovery and mitigation of these attack vectors.

Another way an attacker may try to evade detection, is to specifically attempt to evade our statistical features (see Section 6.2). For instance, to evade the *download domain age* and *domain Alexa rank* features, the attacker could host the malicious files on a free file sharing site. This could result in a download domain with an age > 1 year and a high Alexa ranking. However, the *ad-driven*, *minimum ad domain age* and *maximum ad domain popularity* features, which are harder for the attacker to control, could still allow to identify most attacks. For example, simply knowing that a software download resulted from an online ad puts its probability of being malicious at more than 50%, according to the real-world data we collected (see Section 5.3). Furthermore, if hosting malicious downloads on free hosting sites became popular, then a *Free File Hosting* feature could be added to our feature set, as it is unlikely that many ad-driven benign software downloads are served from free file hosting sites.

8 Related Work

Social engineering is primarily an attack on users, not systems. The fundamental concepts that are employed to exploit the user are rooted in modern psychology, specifically in the study of persuasion [13] and deception [41]. SE attacks have been studied in [21, 27]. While these works study SE tactics in general, they do not focus on SE download attacks. To the best of our knowledge, the only systematic study on SE malware is [8], which discusses the psychological and technical ploys adopted by SE attacks and some trends in SE malware. However, [8] focuses on malware that spreads via e-mail and on SE tactics used by malware to lure the user to activate (i.e., run) the malicious code. In addition, the data analyzed in [8] is limited to malware attack case studies and statistics published by others in the *VirusBulletin* journal until 2010. In contrast, we focus on web-based SE downloads, and on reconstructing and analyzing how users are tricked into downloading malicious software in the first place. Also, our analysis is based on recent real-world instances of successful SE attacks collected from a live network.

Malware downloads have been studied in a number of works [11, 22, 34, 39]. For instance, [34, 39] use a content-agnostic detection approach that relies on computing a reputation score for the domains/IPs from which malware downloads are served. However, [34, 39] are

generic malware download detection systems that offer no understanding of what caused a malware download in the first place. In other words, they cannot identify the origin of the attack, but only its side effects (i.e., only the malware download even itself), and therefore offer no clue on whether an infection was caused by a SE attack. Other works focus on the properties of malware droppers [11, 22], whereby already infected machines download malware updates or new malware strains. In contrast, we study how users fall victim to web-based SE download attacks, and design a detection system that can accurately detect ad-driven SE downloads.

Researchers have also separately examined specific types of SE malware attacks, such as FakeAVs [15, 16, 19, 25, 37]. Our work is different because we propose a general approach to studying, measuring and classifying SE download attacks on the web. We do not limit ourselves to specific attack types such as Fake AVs. Therefore, our work has broader applications, and also provides mitigation against generic ad-based SE download attacks, which represent a large percentage of all SE download attacks we observed in the wild.

Other works have focused on traffic redirection chains to understand and detect malicious websites and attack delivery [23, 26, 38]. Among these systems, Mad-Tracer [23] studies malicious advertisement, including ad chains that deliver malware downloads. This is done by crawling popular websites, and using a supervised classifier trained on data labeled by leveraging domain name blacklists (e.g., Google SafeBrowsing). While part of our work, namely our ad-driven SE download detection system, also leverages some properties of advertisement chains to detect ad-driven malicious downloads, it is important to notice that we focus specifically on *in-the-wild SE download attacks* and are able to identify a large variety of SE download attacks. For instance, [23] only reports fake anti-viruses (AVs) as malware delivered via ad-based *scams*. Instead, in our study we find many other types of SE-driven downloads that leverage a variety of deception and persuasion tactics. In fact, our measurements show that fake AVs represent only a small fraction (less than 1%) of all SE attacks. In addition, rather than actively looking (or crawling) for possible malware downloads on popular websites, we collect *live* SE attacks by directly witnessing successful attacks against users in a large academic network. This allows us to collect *successful SE attacks*, rather than *possible attacks* as done in [23].

In our work, we use a combination of web traffic reconstruction and analysis to trace back the origin of the attacks, namely the SE tactic that tricked the user into downloading malicious software. Researchers have studied web traffic reconstruction in [12, 29, 30, 42]. Among these, the closest to our work is WebWitness [30], a re-

cently proposed incident investigation system that aims to provide context to malicious downloads by reconstructing the path taken by the user to download executable files. WebWitness is able to classify the cause of a malicious download as drive-by, social engineering or update. In [30], the main focus is on studying drive-by downloads and developing a new defense against drive-by download attacks. However, social engineering attacks are not studied. WebWitness is able to separate drive-by downloads from social engineering downloads once an *oracle* identifies a download as malicious, and is not able to independently detect SE attack. Although we utilize WebWitness' trace-back algorithm, our contributions are very different from [30], because we study SE download attacks in depth, focusing on their collection, analysis and categorization, as well as the detection and mitigation of ad-based SE-driven infections.

9 Conclusion

In this paper, we presented the first systematic study of social engineering (SE) attacks that trigger software downloads. To this end, we collected and reconstructed more than two thousand examples of in-the-wild SE download attacks captured at a large academic network. We performed a detailed analysis and measurements on the collected data, and developed a categorization system to identify and organize the tactics typically employed by attackers to make SE download attacks successful. Furthermore, by measuring the characteristics of the network infrastructure used to deliver such SE attacks, we were able to engineer a number of features that can be leveraged to distinguish between SE and benign (or non-SE) software downloads with a true positive rate of 91% at a false positive rate of only 0.5%.

References

- [1] Adblock Plus. <https://adblockplus.org/>.
- [2] ReferrerPolicies. <https://www.w3.org/TR/referrer-policy/#referrer-policy-origin>.
- [3] Protecting privacy with referrers, 2010. <https://www.facebook.com/notes/facebook-engineering/protecting-privacy-with-referrers/392382738919/>.
- [4] The download.com debacle: What CNET needs to do to make it right, 2011. <https://www.eff.org/deeplinks/2011/12/downloadcom-debacle-what-cnet-needs-to-make-it-right>.
- [5] Huge decline in fake av following credit card processing shakeup, 2011. <http://krebsonsecurity.com/2011/08/huge-decline-in-fake-av-following-credit-card-processing-shakeup/>.
- [6] Fake virus alert malware (fakeav) information and what to do, 2013. <http://helpdesk.princeton.edu/kb/display.plx?ID=1080>.
- [7] Heres what happens when you install the top 10 download.com apps, 2015. <http://www.howtogeek.com/198622/heres-what-happens-when-you-install-the-top-10-download-com-apps/?PageSpeed=noscript>.
- [8] ABRAHAM, S., AND CHENGALUR-SMITH, I. An overview of social engineering malware: Trends, tactics, and implications. *Technology in Society* 32, 3 (2010), 183 – 196.
- [9] BOTT, E. Social engineering in action: how web ads can lead to malware, 2011.
- [10] BREIMAN, L. Random forests. *Mach. Learn.* 45, 1 (Oct. 2001).
- [11] CABALLERO, J., GRIER, C., KREIBICH, C., AND PAXSON, V. Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the 20th USENIX Conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 13–13.
- [12] 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, pp. 186–195.
- [13] CIALDINI, R. B. *Influence: Science and Practice*, 5th ed. Pearson Education, 2000.
- [14] 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, pp. 281–290.
- [15] DIETRICH, C. J., ROSSOW, C., AND POHLMANN, N. Exploiting visual appearance to cluster and detect rogue software. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2013), SAC '13, ACM, pp. 1776–1783.
- [16] DUMAN, S., ONARLIOGLU, K., ULUSOY, A. O., ROBERTSON, W., AND KIRDA, E. Trueclick: Automatically distinguishing trick banners from genuine download links. In *Proceedings of the 30th Annual Computer Security Applications Conference* (New York, NY, USA, 2014), ACSAC '14, ACM, pp. 456–465.
- [17] GRIER, C., BALLARD, L., CABALLERO, J., CHACHRA, N., DIETRICH, C. J., LEVCHENKO, K., MAVROMMATIS, P., MCCOY, D., NAPPA, A., PITSILLIDIS, A., PROVOS, N., RAFIQUE, M. Z., RAJAB, M. A., ROSSOW, C., THOMAS, K., PAXSON, V., SAVAGE, S., AND VOELKER, G. M. Manufacturing compromise: The emergence of exploit-as-a-service. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 821–832.
- [18] GRIER, C., TANG, S., AND KING, S. T. Secure web browsing with the op web browser. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2008), SP '08, IEEE Computer Society, pp. 402–416.
- [19] KIM, D. W., YAN, P., AND ZHANG, J. Detecting fake anti-virus software distribution webpages. *Comput. Secur.* 49, C (Mar. 2015), 95–106.
- [20] KOHAVI, R., AND JOHN, G. H. Wrappers for feature subset selection. *Artificial Intelligence* 97, 12 (1997), 273 – 324. Relevance.
- [21] KROMBOLZ, K., HOBEL, H., HUBER, M., AND WEIPPL, E. Advanced social engineering attacks. *J. Inf. Secur. Appl.* 22, C (June 2015), 113–122.
- [22] KWON, B. J., MONDAL, J., JANG, J., BILGE, L., AND DUMITRAS, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1118–1129.

- [23] LI, Z., ZHANG, K., XIE, Y., YU, F., AND WANG, X. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 674–686.
- [24] 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, pp. 440–450.
- [25] MAVROMMATIS, P., BALLARD, L., PROVOS, N., INC, G., AND ZHAO, X. The nocebo effect on the web: An analysis of fake anti-virus distribution. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2010).
- [26] 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* (April 2014), pp. 1159–1167.
- [27] MITNICK, K. D., AND SIMON, W. L. *The Art of Deception: Controlling the Human Element of Security*, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [28] 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, pp. 1–20.
- [29] NEASBITT, C., PERDISCI, R., LI, K., AND NELMS, T. Click-miner: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 1244–1255.
- [30] NELMS, T., PERDISCI, R., ANTONAKAKIS, M., AND AHAMAD, M. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2015), SEC'15, USENIX Association, pp. 1025–1040.
- [31] POWER, R., AND FORTE, D. Social engineering: attacks have evolved, but countermeasures have not. *Computer Fraud and Security* 2006, 10 (2006), 17 – 20.
- [32] PROJECT, T. C. Out-of-process iframes (OOPIFs). <https://www.chromium.org/developers/design-documents/oop-iframes>.
- [33] 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* (Berkeley, CA, USA, 2007), HotBots'07, USENIX Association, pp. 4–4.
- [34] RAJAB, M. A., BALLARD, L., LUTZ, N., MAVROMMATIS, P., AND PROVOS, N. Camp: Content-agnostic malware protection.
- [35] 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 *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (Berkeley, CA, USA, 2010), LEET'10, USENIX Association, pp. 3–3.
- [36] REIS, C., AND GRIBBLE, S. D. Isolating web programs in modern browser architectures. In *Proceedings of the 4th ACM European Conference on Computer Systems* (New York, NY, USA, 2009), EuroSys '09, ACM, pp. 219–232.
- [37] STONE-GROSS, B., ABMAN, R., KEMMERER, R. A., KRUEGEL, C., STEIGERWALD, D. G., AND VIGNA, G. The underground economy of fake antivirus software. In *In Proc. (online) WEIS 2011* (2011).
- [38] 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 & Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 133–144.
- [39] VADREVU, P., RAHBARINIA, B., PERDISCI, R., LI, K., AND ANTONAKAKIS, M. Measuring and detecting malware downloads in live network traffic. In *Computer Security ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds., vol. 8134 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 556–573.
- [40] WANG, H. J., GRIER, C., MOSHCHUK, A., KING, S. T., CHOUDHURY, P., AND VENTER, H. The multi-principal os construction of the gazelle web browser. In *Proceedings of the 18th Conference on USENIX Security Symposium* (Berkeley, CA, USA, 2009), SSYM'09, USENIX Association, pp. 417–432.
- [41] WHALEY, B. Toward a general theory of deception, 1982. *Military Deception and Strategic Surprise*.
- [42] XIE, G., ILIOFOTOU, M., KARAGIANNIS, T., FALOUTSOS, M., AND JIN, Y. Resurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference, 2013* (2013), IEEE, pp. 1–9.
- [43] XING, X., MENG, W., LEE, B., WEINSBERG, U., SHETH, A., PERDISCI, R., AND LEE, W. Understanding malvertising through ad-injecting browser extensions. In *Proceedings of the 24th International Conference on World Wide Web* (New York, NY, USA, 2015), WWW '15, ACM, pp. 1286–1295.