Targeted Attacks Detection With SPuNge

Marco Balduzzi, Vincenzo Ciangaglini, Robert McArdle
Trend Micro Research, EMEA

Marco Balduzzi
Trend Micro Research, EMEA

Vincenzo Ciangaglini
Trend Micro Research, EMEA

Robert McArdle
Trend Micro Research, EMEA

Contents

Abstract

Over the past several years there has been a noticeable rise in the number of reported targeted attacks, which are also commonly referred to as advanced persistent threats (APTs). This is seen by security experts as a landscape shift from a world dominated by widespread malware that infect indiscriminately, to a more selectively targeted approach with higher gain. One thing that is clear about targeted attacks is that they are difficult to detect, and not much research has been conducted so far in detecting these attacks. In this paper, we propose a novel system called SPuNge that processes threat information collected on the users' side to detect potential targeted attacks for further investigation. We use a combination of clustering and correlation techniques to identify groups of machines that share a similar behavior with respect to the malicious resources they access and the industry in which they operate (e.g., oil & gas). We evaluated our system against real data collected by an antivirus vendor from over 20 million customers installations worldwide. Our results show that our approach works well in practice and is helpful in assisting security analysts in cybercrime investigations.

Introduction

Over the last several years there has been a noticeable rise in the number of reported targeted attacks, which are also commonly referred to as APTs or advanced persistent threats. These attacks are carried out by attackers with several different motivations – with financial gain and espionage being main driving factors. Financial gain is of course also a factor for widespread attacks, but espionage is more limited to attacks of a targeted nature. Overall this is seen by security experts worldwide as a landscape shift from a world dominated by widespread malware that infects indiscriminately, to a more selectively targeted approach with higher gain.

While it is unlikely that widespread malware will vanish completely (or even decrease noticeably), almost all industry commentators agree that targeted attacks will continue to rise. This view is also echoed by the media and AV company customers, who are very concerned about attacks aimed at their organizations. Notable examples of recent targeted attacks include Red October [1] and IXESHE [12].

One difficulty in talking about targeted attacks is that everyone has a different understanding of what it means to them. For the purposes of this paper we will use the following definition:

> "A targeted attack refers to an electronic attack carried out by a group of attackers against a specific organization, country or industry with the goal of theft of data or gaining control of company resources".

In fact, what sets a targeted attack apart from a widespread attack is purely the motivation behind the attackers and their victims (targets), while the actual malware or technology adopted is largely irrelevant. For example a banker infection across 50 countries (e.g., Zeus) would be considered a widespread attack, while the same attack against two nuclear power plants – and nowhere else – is an example of a targeted attack. The tool is identical, but the motivations of the attackers, and the targeted victims set this apart.

One thing that is clear about targeted attacks is that they are difficult to detect, and not much research has been conducted so far in detecting these attacks automatically. In this paper, we propose a novel system that processes threat information collected on the users' side to detect potential targeted attacks.

Often these attacks have generic detections which do not call them out as targeted in an obvious way, but using our approach we are able to reduce the millions of normal malicious events down to a more manageable amount for further in-deep analysis. Our contributions are:

- We propose a system to find potential targeted attacks (and victims). We use a combination of clustering techniques to identify groups of machines that share a similar behavior with respect to the malicious resources they request (e.g., exploit kits, drive-by downloads or C&C servers).
- We correlate the location and industry information in which these machines operate (e.g., oil & gas or government) to discover interesting attack operations.
- We implemented our system in a working prototype that we called SPuNge.
- We evaluated our system against one week of threat data collected by an antivirus vendor from over 20 million users installations worldwide. Our empirical results show that our approach works well in practice and is helpful in assisting security analysts in cybercrime investigations.

The remainder of the paper is structured as follows: Section II presents our approach for the detection of potential targeted attacks and how our system is designed. Section II-B details how clustering is used in SPuNge. Later, in Section III we discuss the implementation details and the solutions we introduced to analyze the data in an efficient way. We address the ethical concerns in Section IV-C, while in IV and IV-B we describe how we conducted the experiments and our findings. We give the related work in Section V and conclude with Section VI.

Target Attacks Detection with SPuNge

We defined a targeted attack as an electronic attack carried out by a group of attackers against a specific organization, country or industry, with the goal of theft of data or gaining control of company resources – i.e., the victims are often located within one, or few, geographic locations, or they all operate in the same industry. Given this premise, hereby we introduce our approach for the detection of potential targeted attacks.

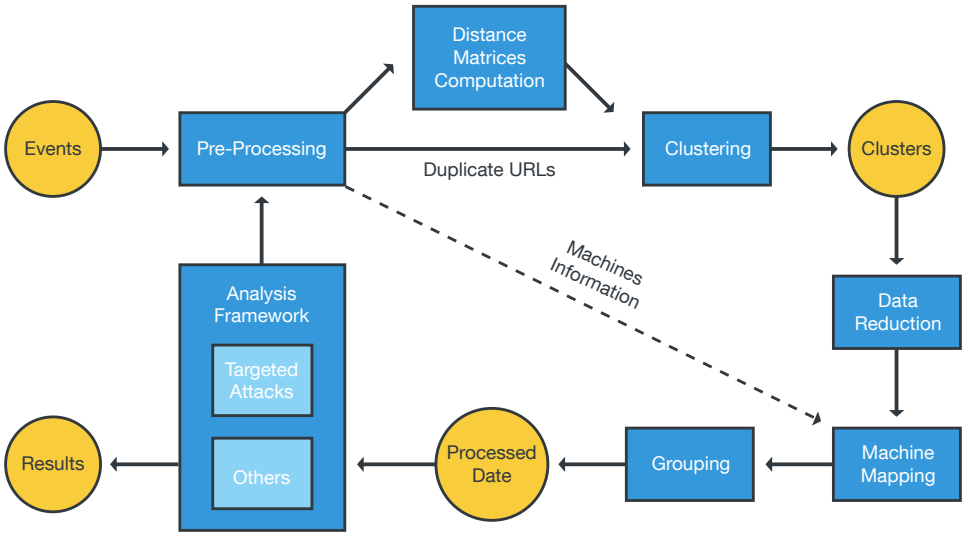

Figure 1. SPuNge Architecture

Our approach consists of two phases. In first stage, we analyze the malicious URLs that regular users machines access over HTTP(S) with an Internet browser, another HTTP client or because infected by a malware. We identify those machines that present a similar network behavior – e.g., accessing web pages used within the same phishing campaign or malware infection. We apply a combination of clustering techniques to group together similar malicious URLs, and we "organize" the machines based on the clusters of URLs that they requested. Our system, that we called SPuNge, is composed of six main components that do *pre-processing, distance matrices computation, clustering, data reduction,*

machine mapping and grouping. We introduce them in the rest of the Section and we present the general architecture in Figure 1.

In the second phase, we correlate the clusters of machines presenting a similar behavior, and we identify those machines, networks or organizations that are more likely to be involved in a targeted attack. For example, because operating in the same industry (e.g., oil & gas). We developed an analysis framework to analyze the results of the processing and to automatically generate a report for the security analyst. We discussed this more in detail in Section II-G.

Pre-Processing

We start our discussion by introducing the pre-processing component together with its tasks: (i) to load and parse the threat data to be analyzed, (ii) to get rid of irrelevant information to the detection of targeted attacks and (iii) to identify duplicates and redundancies in the URLs.

Within SPuNge we process collections of threat events, i.e. malicious URLs that regular users access over HTTP(S) with an Internet browser, another HTTP application or because infected by a malware with network capabilities (e.g., a bot that connects to the C&C server). These URLs are known to be malicious and are blocked at client side by a security program (e.g., an antivirus). Examples include: web pages that harvest malware, fake AVs, RATs, drive-by-download code, phishing campaigns, and finally C&C server hosting.

cr5aigslist.com	craigsli8st.com	crauglist.com	craigslistnc.com
crageslist.com	craigslisg.com	craigslisrt.com	craigslistny.com
craigaslist.org	craigdlist.com	craighlist.org	craigilist.com
creagslist.com	craiclist.com	craigslit.ca	craigllist.org
	craigsliost.com	cragslists.com	

Table 1. Example of 16 craigslist's typosquatting domains.

We process threat events "cleverly", by trying to keep only those events that are more relevant for the detection of possible targeted attacks. Note that an event consists of the URL and of the information on the machine that requested it.

In particular, we apply the following set of filtering:

1. Classification: We ignore "parental controlled" URLs because they are not relevant for the detection of targeted attacks (e.g., sites related to pornography and violence).

2. Network Sampling: We keep a single infected candidate per network and URL (i.e. one IP per Class B). We store information on which networks are infected and not for each machines.

3. Events Sampling: If a URL is requested by a single machine multiple times, we store this information once (i.e. the first request). A typical scenario is a botnet controlled machine that pulls commands out of the C&C server regularly.

4. Duplicates: If a single URL is widely requested by a large number of machines (N), we ignore the event if N is greater than a threshold, because it is not relevant for the detection of targeted attacks.

5. Whitelisting: We ignore clusters of popular URLs, for example URLs being massively requested. This is discussed later in Section IV-A.

Clustering in APT detection

Clustering is an operation that, given a set of arbitrary elements, identifies and assigns the elements, without prior information, to groups (called clusters), so that elements that are similar to one another according to a chosen criteria may fall in the same group. As such, clustering constitutes an important stage of our processing, as it permits to identify patterns in the collected data without prior knowledge, to reduce the number of elements to be examined. Therefore, it allows for the presentation of aggregate views of possible attacks. More specifically, clustering enables the identification of groups of malicious events sharing similarities in the URLs' hostname or in the request path.

Attackers are known to register and exploit group of similar domains, for example using typosquatting techniques, to conduct attacks like drive-by-downloads, phishing, scamming and click fraud by leveraging the popularity of the benign domain that is "squatted".

Recently, Avast! reported that craigslist advertisements website had been heavily typosquatted with hundreds of domains to lure the visitors into running a fake "quiz" where they receive an offer of a "free" prize such as an iPhone [1]. These sites typically make money through premium phone calls, selling advertisements, and reselling the emails collected from visitors. Some of these domains are shown in Table 1. Obviously this is a fraud.

Exploit Kit	URL's Host	URL's Request
Blackhole	hxxp://77.79.13.88	/content/w.php?f=52&e=4
Blackhole	hxxp://188.127.249.241	/image/l.php?f=553&e=2
Blackhole	hxxp://brown.mydomxd.org	/root/w.php?f=2293&e=6
Nuclear	hxxp://zeak.rghil.info	/a456gh/9493af39692e[...].jar
Nuclear	hxxp://163.1.32.2	/1rg54e/55c2b44e0c8a[...].jar
Nuclear	hxxp://31.184.244.9	/6ju9a2/bb136b125774[...].jar

Table 2. Example of URLs used by BLACKHOLE and NUCLEAR exploit kits and detected with SPUNGE.

In another report, Hernandez [6] claimed that ten of the top 50 financial institutions, offering online banking services to their customers, had been "phished" with hundreds of fake domains that seemingly looked like the original websites. Attackers, for example, have registered more than 50 typosquatting domains of chase.com and deployed malicious variants of Chase Bank's online portal for stealing the bank credentials of unaware victims.

While domain clustering is an effective approach to detect malicious URLs sharing the same domain characteristics, a complementary approach is to ignore the domain part of the URL and to cluster accordingly to the HTTP request string (path and query string). In fact, botnets and exploit kits authors normally don't rely on a single domain to operate, which is generally seen as a single point of failure, but prefer to organize their infrastructure across multiple (often compromised) machines and locations, by switching from one to another regularly.

Two notable examples of exploits kits are Blackhole and Nuclear, which are currently among the most popular ways of delivering malicious payloads to a victim's computer like fake AVs, bankers (e.g., Zeus) or ransomware. The attack consists of installing the exploit kit on a webpage that acts as landing server and attracting the most number of victims, via spamming for example. The landing page normally contains some obfuscated Javascript that detects the configuration of the victim's machine and serves the appropriate exploit. By looking at Table 2, the reader can see that the URL's request characterizes the threat as exploit kit and the family as Blackhost or Nuclear.

What we designed in SPuNge is a clustering algorithm that allows to identify groups of malicious URLs that share either similar hostnames (domains) or similar requests. As specified in RCF 39862, a URI schema is defined as a hostname (e.g., domain name or IP address) and a request path, as a sum of the path itself and the query string (a series of parameter-value pairs). Thus, clustering URL information involves computing two different sets of distances:

1. Distances measuring the similarity of URLs' hostname (*host distances*);

2. Distances measuring the similarity of URLs' request, intended as path and query string (*request distances*).

Therefore, it becomes important to choose an appropriate distance, i.e. an appropriate similarity criteria. The similarity criteria has to be chosen depending on the type of data to cluster, and is usually in the form of a distance function that can estimate how "close" two elements of the dataset are to one another, or a metric, i.e. a function that calculates the unique coordinates of each element in a multidimensional algebraic space. The ability of either placing a dataset in a metric space or simply to compute distances between elements is also one of the factors that drive the choice of the clustering algorithm to use.

Clustering in SPuNge

Clustering has been widely used in literature, with the most renowned algorithms being adopted being k-means [7], used for example in [17], x-means [13], used in works such as [2] or [5], or hierarchical clustering [7], used in [14]. One of the requirements driving our work it is being able to process the data rapidly and efficiently. Because of that, and because of the prominently textual nature of our data, the clustering algorithm we introduced in SPuNge is a *hierarchical single-linkage clustering*, chosen for the following reasons:

- Algorithms like k-means usually require the number of clusters to be initially set, whereas in our scenario said number is not only unknown, but one of the variables to be computed. Even though variants, such as x-means, do not require an initial knowledge of the number of clusters, they involve several iterations over the same dataset and an additional cluster validation at the end of each iteration, thus adding computational cost to the processing.
- Algorithms like k-means require an Euclidean distance to compute the cluster centroids, while hierarchical clustering can flawlessly cope with non metric distances like the ones used on our dataset.
- Hierarchical clustering, in its single-linkage variant, does not require to recompute new distances when clusters are created, thus saving time and resources.

In the first phase, two distinct distance matrices are computed: One measuring the similarity of each URL pair according to its respective hostnames (host distance) and another measuring the similarity of their requests (request distance). The similarity is computed using the distance functions described later in the Section. The algorithm processes both distances matrices and groups similar URLs into two distinct cluster sets *CHost* and *CReq*: One with clusters of URLs sharing similar hostnames (*host clusters*) and another with clusters sharing similar requests (*request clusters*). To create each set, the clustering algorithm runs as follows:

1. Each distance *d* is considered, in ascending order;
2. The list of all URL pairs (e_1, e_2) that measured d is parsed;
3. Each pair is assigned to a new cluster C_{new};
4. If e_1 or e_2 were already assigned to a previous cluster C_{old}, C_{new} assimilates all the elements in C_{old}, and then C_{old} is discarded;
5. The operation repeats until a given threshold *T* is reached, i.e. until there are no more pairs at a distance lower than *T*.

	cr5aigslist.com	craigsli8st.com	crauglist.com	craeglist.com	google.com
cr5aigslist.com	0	0.0666	0.1428	0.1428	0.520
craigsli8st.com	0.0666	0	0.1428	0.1428	0.520
crauglist.com	0.1428	0.1428	0	0.0769	0.478
craeglist.com	0.1428	0.1428	0.0769	0	0.478
google.com	0.520	0.520	0 .478	0.478	0

Table 3. Example of distance matrix for hostnames (normalized Levenshtein).

As such, *T* is the maximum allowed distance to cluster URLs together. The threshold serves a double purpose of (i) acting as a termination condition for the clustering, by defining size and quality of each cluster, and (ii) limiting the amount of processing resources (i.e. every pair having a distance higher than the threshold can be memory freed).

1. *Host Distance:* In this section we explain the distance function that we used to measure the similarity between hostnames.

As hostnames are strings, similarity between hostnames can be computed with one of the several well-known functions that are capable of quantifying how two strings of text are similar to one another: The Hamming distance, for example, counts the number of bits having same position and different value; the Jaccard distance treats text as a set of characters and counts how many characters two sets do not have in common; the Levenshtein distance is probably the most common.

The Levenshtein distance measures the similarity of two strings as minimum number of edit operations needed to change one string into the other, with an edit operation being insertion, deletion or modification of a character in a string. As an example, two edit operations are needed to change the word “Robert” to “Roger”: “Robert” → “Rogert” and “Rogert” → “Roger”. Levenshtein also allows a comparison of strings of different length (unlike the Hamming distance) and keeps the information on duplicate characters and their position (that Jaccard does not).

In SPuNge, we use the Levenshtein function, normalized in the interval [0; 1] to compute the distance dhost between two hostnames. The example in Table 3 shows the distances computed over the typosquatting domains of Table I plus an additional domain (google.com), which is very different from the others. As we can see, there are several desirable properties that made it a suitable candidate:

A. The distance is 0 for the same domain;

B. The distance is symmetric;

C. There is a clearly quantifiable difference between similar and non-similar domains: Different domains show a distance much higher than similar one, making it relatively simple to differentiate the two (gray color).

2. *Request Distance*: As we have already said before, a URI schema is defined as consisting of a hostname and a request path, as a sum of the path itself dpath and the query string d_{qsl} (a sequence of parameters and values).

As a consequence, the request distance d_{req} between two URL requests is more complex than d_{host}, because composed of two distinct metrics. That is, we use the normalized Levenshtein function to compute the similarity between paths and the Jaccard function to measure the similarity of the query strings. Note that we apply the Jaccard only to the query string's parameters (and not their values), by counting how many parameters the two requests have in common. We ignore the values because they often change and poorly characterize a request.

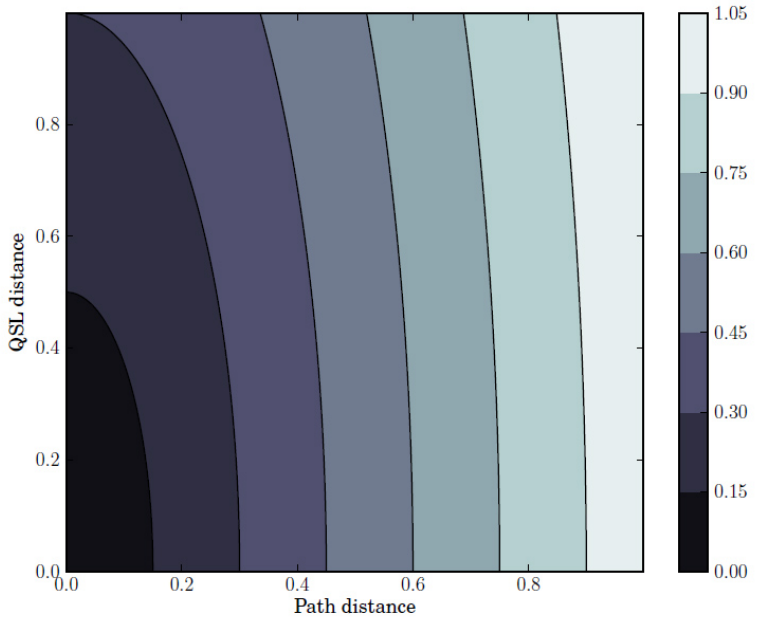

Figure 2. The request distance d_{req} as contribution of d_{path} and d_{qsl} similarities.

Cluster	Cluster Label	Event	Hostname
C_1	H zfmudav4aaq33r5.com	e_1	zfmudav4aaq33r5.com
		e_2	zfmudav4aaq35r5.com
		e_3	zfmudav3aap36r5.com
		e_4	zfmudav2acq35r4.com
C_2	H facebookc.com	e_5	facebookc.com
		e_6	facaebook.com
		e_7	faceboook.com
		e_8	facebopok.com
C_3	H h-aelameftzgj4vxient.com	e_9	h-aelameftzgj4vxient.com
		e_{10}	h-aelameftxcd5vxient.com
		e_{11}	h-aelameftssd6vxient.com
		e_{12}	h-aelanfftzgj1vxient.com

Table 4. Example of host clusters.

The request distance, as shown in Figure 2, is the contribution of two sub-distances and obtained by the following formula:

$$d_{req}(e_1, e_2) = \sqrt{d_{path}(e_1, e_2)^2 + (WeightFactor \times d_{qsl}(e_1, e_2))^2} \quad (1)$$

WeightFactor is a numerical factor that rescales dqsl so that d_{qsl} and dpath bear an equal contribution to d_{req}. The rule of thumb to calculate the WeightF actor is that the threshold T_{path} and T_{sql} for both components should be the same. D_{path} uses the same Levenshtein distance as d_{host}, and the two thresholds T_{path} and T_{host} have the same value (0.15); two paths with a distance lower than 0.15 are considered similar. However, we consider two query strings similar if they have at least half of their parameters in common, hence the threshold of T_{sql} is 0.5. In order to normalize T_{sql} to 0.15, we set WeightFactor to 0.333; applying the Equation 1 to T_{path} and T_{sql} gives a compound threshold T_{req} of 0:15$\sqrt{2}$ = 0:212.

Labeling and Data Reduction

When the clustering is over, the processed events are “organized” in two orthogonal cluster sets that contain URLs grouped according to our two criteria of similarity (hostname and request). Examples of the two cluster sets are shown in Tables 4 and 5.

In this section, we describe the operation of cluster labeling, which consists of assigning to a cluster a "human-friendly" label (i), and of data reduction, which consists of reducing the number of clusters by the identification of redundancies and merging (ii).

Cluster Label	Event	Request
C_4: R /get2.php?c=BLMEUGUBd=266	e_1	/get2.php?c=BLMEUGUBd=266
	e_2	/get.php?c=ZLXULJNRd=266
C_5: R /9MzImdHA9MCZmbD0w0	e_3	/9MzImdHA9MCZmbD0w0
	e_4	/9MzImdHB9MCZmbD0w1
C_6: R /qKA0rO4d8l7qBhS7Y2xrPTQu	e_9	/qKA0rO4d8l7qBhS7Y2xrPTQu
	e_{10}	/lkG1yP3L8q5YPtU7Y2xrPTQu
	e_{11}	/BAq3T78d8l5Q7bs0Y2xrPTQu
	e_{12}	/pA71gKND6P5MTls9Y2xrPTQu

Table 5. Example of request clusters

We introduced labels to rapidly visualize the content of a cluster (i.e. without the need of inspecting the URLs contained within). We label each cluster with the following convention:

- Host clusters' label have prefix H followed by the hostname of a cluster's event;
- Request clusters' label have prefix R followed by the query string of a cluster's event.

The *data reduction* operation consists of going through both clusters sets, henceforth referred to as *CHost* and *CReq*, and performing a clusters merge when certain conditions are met. Merging two clusters of different type involves discarding one of the two and updating the label of the "survivor" to reflect the merge operation. In detail, we have the following possibilities:

1. If *CHost* and *CReq* contain the same information, *CReq* is discarded [3] and *CHost*'s label is updated as `CHOST-LABEL =: CREQ-LABEL`

2. If *CHost* is a subset of *CReq*, *CHost* is discarded and *CReq*'s label is updated as `CREQ-LABEL >: CHOST-LABEL`

3. If *CReq* is a subset of *CHost*, *CReq* is discarded and *CHost*'s label is updated as `CHOST-LABEL >: CREQ-LABEL`

Table 6 shows the results of the merging stage on 4 and 5. The labeling convention we adopted provides a quick understanding of the clusters content and the relationships between the merged clusters. For example, the C_1's label tells us that C_1 is a cluster of URLs having hostnames similar to

`zfmudav4aaq33r5.com` and requests in two groups: The first similar to `/get2.php?c=BLMEUGUBd=266` and the second to `/9MzImdHA9MCZmbD0w0`. A second cluster C_3 contains URLs having both similar hostname `h-aelameftzgj4vxient.com` and request `/qKA0rO4d8I7qBhS7Y2xrPTQu`.

Clusters	Cluster Label	Event	URL
C_1	H zfmudav4aaq33r5.com >: R /get2.php?c=BLMEUGUBd=266 >: R /9MzImdHA9MCZmbD0w0	e_1 e_2 e_3 e_4	zfmudav4aaq33r5.com/get2.php?c=BLMEUGUBd=266 zfmudav4aaq35r5.com/get.php?c=ZLXULJNRd=266 zfmudav3aap36r5.com/9MzImdHA9MCZmbD0w0 zfmudav2acq35r4.com/9MzImdHB9MCZmbD0w1
C_2	H facebookc.com	e_5 e_6 e_7 e_8	facebookc.com facaebook.com faceboook.com facebopok.com
C_3	H h-aelameftzgj4vxient.com =: R /qKA0rO4d8I7qBhS7Y2xrPTQu	e_9 e_{10} e_{11} e_{12}	h-aelameftzgj4vxient.com/qKA0rO4d8I7qBhS7Y2xrPTQu h-aelameftxcd5vxient.com/IkG1yP3L8q5YPtU7Y2xrPTQu h-aelameftssd6vxient.com/BAq3T78d8I5Q7bs0Y2xrPTQu h-aelanfftzgj1vxient.com/pA71gKND6P5MTls9Y2xrPTQu

Table 6. Example of merged clusters

Machine Mapping

Up to this point the processed results contain information about similar malicious URLs, which we clustered together. The *machine mapping* component aims at identifying and correlating the source of the malicious requests, i.e. by finding out which group of machines have performed requests to the same cluster, or vice versa to which resources (clusters) have a machine connected to. In fact, we are interested in knowing which users machines behave similarly with respect to their malicious network behavior, for example because targeted by the same phishing campaign or botnet.

To achieve this result, the following transformations are performed on the merged clusters set:

1. For each cluster: We extract from all cluster's events an identifier of the user machine, i.e. its IP address in an anonymized form (we take into consideration possible privacy issues, as we describe in Section IV-C.

2. We produce an association table in the form *cluster* → *machine*, as shown in Table 7.

3. We build a second table in the form *machine* → *cluster*, as shown in Table 8. – Note that the two tables are one the reverse format of the other.

Cluster	Cluster Label	Event	Source Machine
C_1	H zfmudav4aaq33r5.com >: R /get2.php?c=BLMEUGUBd=266 >: R /9MzImdHA9MCZmbD0w0	e_1 e_2 e_3 e_4	M_1 M_2 M_3 M_4
C_2	H facebookc.com	e_5 e_6 e_7 e_8	M_1 M_2 M_5 M_6
C_3	H h-aelameftzgj4vxient.com =: R /qKA0rO4d8l7qBhS7Y2xrPTQu	e_9 e_{10} e_{11} e_{12}	M_3 M_4 M_5 M_7

Table 7. Example of *cluster* → *machine* associations.

Source Machine	Clusters
M_1	C_1, C_2
M_2	C_1, C_2
M_3	C_1, C_3
M_4	C_1, C_3
M_5	C_2, C_3
M_6	C_2
M_7	C_3

Table 8. Example of *machine* → *cluster* associations.

Grouping

With this last processing stage, called grouping, we want to identify groups of machines that request the same set of clusters (more than one). Attacks often involve a multi-step process, i.e. the attack is carried out in different phases. A well-known example is a drive-by-download infection in which the victim is first redirected to the malicious page and then served with the right exploit.

While so far the information we collected in form of *machine → cluster* association tells us which individual machine has connected to which destination(s), individually, we perform one last operation to group clusters and machines together. With grouping, we are able to identify:

- If there are groups of uncorrelated resources (e.g., websites) that are used within the same attack or campaign (e.g., malware or phishing campaign);
- If there are groups of machines requesting the same malicious resources, for example because being infected by the same variant of malware;
- If there are machines connecting to a significantly high number of malicious resources, for example because being heavily infected.

Groups	Machine Set	Clusters Set
G_1	M_1, M_2	C_1, C_2
G_2	M_3, M_4	C_1, C_3
G_3	M_5	C_2, C_3
G_4	M_6	C_2
G_5	M_7	C_3

Table 9. Example of groups (machine and clusters)

The result is shown in Table IX: In here we see, for example, two machines $\{M_1, M_2\}$ belonging to the same group of clusters $\{C_1, C_2\}$, and a second group of machines $\{M_3, M_4\}$ sharing the group G_2.

Analysis Framework

Until now, we discovered which group of machines presents a similar malicious network behavior. For example, because accessing web sites used within the same phishing or malware campaign. We used a combination of clustering techniques to cluster malicious URLs that are similar to each other and “organize” the machines based on the clusters of URLs that they accessed.

Next, we developed an automated *analysis framework* to analyze each group of machines and identify potential candidates of targeted attacks. We correlate the information of these candidates, i.e. the industry in which they operate (oil and gas or government, for example) and the location (country).

We run two types of analysis: A first on the clusters set, and a second on the groups of clusters (more than one). In the cluster analysis, we look for *N*+ machines operating in the same industry or country (or a combination of both), and generating requests to URLs being clustered together because similar. We make N varying between 2 and 5. In the group analysis, we search for groups of *N*+ machines that share *C* or more clusters. We use the results of the grouping step previously described and we make both *N* and *C* varying, between 2 and 5 for example. We verify that the identified machines do not share their behavior with other groups – i.e. a sign of widespread attack.

For those machines that match our criteria, we automatically generate a report for the security analyst. An example of the results that our system is able to provide is given later in Section IV.

Implementation

We implemented SPuNge as a Python 2.7 application, a prototype that analyzes threat information with clustering and correlation techniques to discover potential targeted attacks (and victims).

Our implementation faced several challenges, i.e.:

1. the amount of threat data to process was in the order of millions of requests (as shown in Table 10),
2. some processing stages have high complexity (e.g., given *N* requests, we compute (N2) ≈ $N^2/2$ distances),
3. to allow for continuous run on a single machine, the analysis over *H* hours of threat data must last less than *H* hours and
4. the processing has to be recoverable in the eventuality of crash (i.e., as modern filesystems guarantee).

In this section, we detail the design choices we put in place to cope with these constraints.

Duplicates identification and optimization

On top of the pre-processing's filtering operations, which we introduced previously in Section II-A, we perform some additional data optimization at the initial stage. In fact, threat data can be redundant, for example because the same URL address is requested several time and by multiple infected machines. The strategy we adopted to handle this "special case" is to organize the data in two groups of duplicate and unique URLs during the pre-processing stage. We then restrict the computation of the distance matrices to a sample per duplicate group.

In particular, we rely on a hash-based algorithm to identify duplicate elements. Upon parsing each URL, we compute two MD5 hashes (i.e. one per hostname and one per request) and we group together URLs having the same hostname or requests. Then, during the clustering stage, we randomly pick one "candidate" URL for each duplicate group to include in the computation of the distance matrices.

Our empirical experiments over millions of requests showed that our approach reduced the clustering overhead time of 19% – about 10% less URLs to "distance-compute".

Distributed distance computation

Distance computation is demanding on computing resources. Fortunately, it is also an operation over uncorrelated data that can run in a distributed form. We designed SPuNge with an algorithm that supports multi-processing, which is nowadays commonly found in modern multi-core CPUs. It is relatively straightforward, with almost no code change, to extend the system to distributed computing (over multiple machines) with one of the many frameworks available for Python [4].

We use this algorithm to compute the two distance matrices (host and request) and the grouping table described in II-F. Our algorithm works as following:

- We build a linearized events matrix as list of events pairs (also known as the *distances list*).
- To restrict the memory consumption, we split the distances list into *N iteration segments* to process sequentially. In this way, we can run the processing on memory-bound systems by configuring the number of iteration segments *N* accordingly.
- At each iteration, we split the iteration segment into *worker segments* (a sub-list of request pairs) and a pool of worker processes is instantiated to process each segment (one process per segment). Workers have exclusive read-access to the memory area containing the segments by avoiding computational overhead due to message passing, mutex locking or possible race conditions. The results are returned in a shared queue.
- A collector waits for the results, receives the computed distances and organizes them as list of pairs that measured that much; this makes easier to fetch the distances during the clustering operation. To optimize the processing, the garbage collector is controlled manually: We suspend it during the intense distance computation stage and re-enable it once each iteration has ended.
- At the end of each iteration, each partial result is stored on disk to make the whole process faulttolerant. Note that we perform this operation after every processing stage. We use an efficient C-based Python serialization library called cPickle [5] that we found to be very fast.

Experiments

We ran SPuNge against real data collected on the users' side from an antivirus vendor [6], with the goal of evaluating the detection of potential targeted attack campaigns. Trend Micro's smart protection network (hereby called *SPN* [7]) is a cloud-based infrastructure that collects about 6 terabytes of threat data per day from over 20 million customers installations worldwide. The threat data is collected by the SPN network on an hourly basis, and made available to the analyst in the form of data feeds with an API and a web application.

We based our analysis on a data feed that collects information on the malicious URLs that are accessed by users over HTTP(S). When a user requests a URL that is known to be malicious, for example because hosting a RAT, a malware, or a C&C channel, the network security component of the antivirus generates an event for the smart protection network. This event contains information on the requested URL and the user's application that requested it, together with the configuration of the machine. The event consists of the following aggregate fields (summarized):

- The time when the event occurred (GMT converted);
- The URL being requested and the IP address of the web server (at the time of the request);
- The process that generated the HTTP(S) request (name, size and hash);
- The IP address, location (country) and OS version of the machine that generated the event;
- The industry in which the attacked entity operates. A comprehensive list is given in [8].

We ran our experiments over one week of threat data – from Sunday 11th to Saturday 17th of November – by deploying two physical machines (*A* and *B*) in our testing infrastructure. We used machine *A* to process the data with SPuNge and machine *B* to analyze the results, that we transferred from *A* to *B* over network. *B* runs a PostgreSQL database. Their configuration is as following:

- Machine A: Intel Xeon 2.40GHz, 16 Cores, 72GB RAM, 3.5TB Hard Disk
- Machine B: Intel Xeon 2.83GHz, 8 Cores, 16GB RAM, 4.0TB Hard Disk

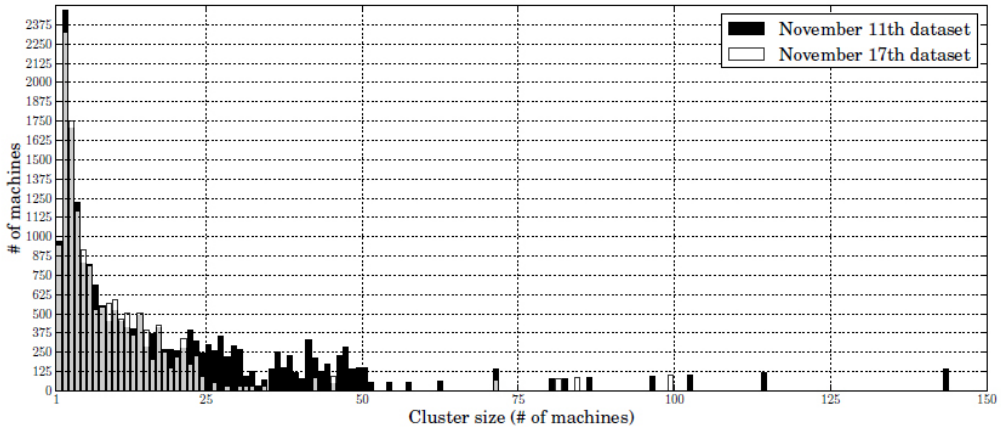

Figure 3. Machines distribution across the different clusters.

Dataset Optimization

When we ran the experiments, we decided to analyze each day individually, i.e. to make use of the results of day N as input for the following processing $N + 1$. In fact, the dataset that we used to evaluated SPuNge revealed to contain "polluted" information, for example webpages that did not appear to be malicious any longer or that have been taken down. In addition, as a consequence of the use of inaccurate signatures in URLs-pattern-matching, URLs having, for example, */icon.ico* or */favicon.ico* as path are considered malicious regardless if the file exists and is indeed malicious. Finally, when the reputation is computed at hostname level, we noticed some webpages being filtered because hosted on a malicious IP or domain, regardless of the nature of the page. As a consequence, we observed several machines requesting the same group of URLs, i.e. generating events that are not necessarily associated with potential targeted attacks.

In order to cope with this limitation, after each day of analysis we automatically extracted the clusters of URLs being requested by more than N machines – because these are more likely to be used in widespread, rather then targeted operations – and we excluded them from the following processing. In this way, we first avoided to re-process the same URLs and we focused on analyzing fresh data; second we followed up in keeping more interesting information. From an implementation point of view, we empirically choose $N = 25$ as threshold – i.e. the same value we used in the elimination of the duplicated URLs during the pre-processing stage (ref. Section II-A).

Number of:	Sun. 11	Mon. 12	Tue. 13	Wed. 14	Thu. 15	Fri. 16	Sat. 17
Raw events (million)	2.79	5.17	5.58	5.68	5.22	4.91	2.62
Processed events	387,339	536,524	256,270	221,954	230,758	269,103	329,458
Processed machines	10,866	15,581	15,413	15,391	14,165	14,364	8,406
Detected clusters	4,106	8,825	8,195	7,825	7,196	7,281	3,869
Detected groups	2,144	3,941	3,579	3,528	2,679	2,896	1,069
Hostnames clustering							
Clustered events	20,433	46,361	44,362	38,663	37,232	41,349	24,596
Computed pairs (mil.)	208	1,074	983	747	693	854	302
Iterations	1	5	4	3	3	4	2
Requests clustering							
Clustered events	111,073	192,075	159,294	117,271	146,279	136,137	138,335
Computed pairs (mil.)	6,168	18,446	12,687	6,876	10,698	9,266	9,568
Iterations	25	74	51	28	42	38	39
Processing time (sec)							
Pre-processing	268	769	665	667	604	527	302
Matrices computation	8,708	21,273	15,307	9,988	13,395	11,675	9,683
Clustering	3,198	10,665	4,906	2,500	4,983	4,739	12,987
Merging	2,089	31,370	23,040	9,632	18,614	13,581	5,082
Machine mapping	274	1,214	488	187	441	414	1,187
Grouping	51	93	74	62	67	61	48
Total (hrs:min)	04:15	18:09	12:32	06:31	10:44	08:47	08:28

Table 10. Processing results.

URL's Host	URL's Request	Machine	Country	Process Name
hxxp://83.133.124.191	HVM2wppE5M7mnPC7Y2xrPTQuOCZ	M1	US	ping.exe
hxxp://83.133.124.191	vA21k6yD7N5XAKC8Y2xrPTQuOCZ	M2	US	ping.exe
hxxp://46.249.59.47	4zk3oUup7K7xjOS0Y2xrPTQuNyZ	M3	US	ping.exe
hxxp://63.223.106.17	rK61TBkp5a3mCgS4Y2xrPTQuNyZ	M4	AU	ping.exe

Table 11. *Ping.exe* Malware. Cluster's label is *R /sVv4VmLE8Z5Mdzc9Y2xrPTQuNyZia*.

Our findings showed that our approach works. In fact, after the first two days, where we massively learned on the poor quality URLs, we reduced the number of events to process from 300-500,000 to roughly 200,000 events, by halving the processing time and incrementing the quality of results – i.e., less number of clusters with a big number of machines (ref. Table 10). The URLs that we daily identified as "polluted" and added to the exclusion list are respectively 252,998, 191,891, 112,627, 3,459, 2,255 and 2,413.

This is confirmed from Figure 3, which shows the machines allocation in the different clusters, i.e. how many machines are clustered into clusters of increasing size. The black bars show the allocation of the first run performed, the 11th of November, where no exclusion was made due to lack of previous information, while the white bars represent the machine allocation on the 17th of November, when a whole week of information coming from previous clusters has been used to skim the raw data. The peak on the left is made of a large number of small clusters (2-3 machines), while the tail on the right is made of few clusters resulting by the merge of numerous events. The interesting result comes from the elimination of a big chunk of clusters having more than $N = 25$ machines as discussed previously.

To summarize, we configured SPuNge with the following parameters and we ran our automated processing over a whole week of threat data: clustering iteration size: 250 million, grouping iteration size: 20 million, duplicate and pollution thresholds (N): 25 events, clustering thresholds (T): 0.15 (host) and 0.15 $\sqrt{2}$ (request).

Findings

We start describing our findings with Table 10. Our datasetconsists of about 5 million events on weekly days (from Monday to Friday) and 2.5 million on Sunday and Saturday – non-working days in most of the users' countries. After having applied the pre-processing filters described in Sections II-A and IV-A, out of the total number, we analyzed an average of 200-300,000 events.

We then further reduced this number down to those events that are unique with respect to URL's hostname or requested path, and we ran the distance matrices computation only for these URLs, as previously discussed in III-A. We ended up in creating two groups of data: A first of about 20-40,000 unique URLs – with respect to the hostname – that we "host clustered", and a second of 100-200,000 unique URLs – with respect to the request – that we "request clustered". This second group is much larger because malware authors often use variation patterns of the path or the query string parts of the request. A typical example are exploit kits, such as Blackhole and Nuclear, that use a variation of different path's parameters to identify the right exploit to serve to the victim based on its configuration.

We used clustering to achieve this result. Nevertheless, clustering is, by definition, a slow approach. As we previously described in Section III-B, we carefully took into account this challenge, and we designed and implemented appropriate solutions. Our empirical results on a real dataset proved that our system is

able to process data in an *online* fashion (i.e. the processing time is smaller than the dataset interval). In fact, our timing measurements showed that SPuNge can handle millions of raw threat events collected each day from over 20 million users installations in less then half-day and with a single 16 cores machine. Note that it is relatively straightforward, with almost no code change, to extend the system to distributed computing (over multiple machines)

The results of this processing are presented in Table 10. Each day our system builds aggregate clustering information out of millions of raw events collected at machine level. In particular, we detected an average of 7,882 clusters and 3,306 groups per day – Table 11 gives an example. This is a group of machines infected by a newly discovered bot that disguises itself by injecting in the standard Windows's application ping.exe. This cluster, labeled as `R /sVv4VmLE8Z5Mdzc9Y2xrPTQuNyZia` (request base cluster), has been identified as consequence of infected machines that communicate with the C&C server by using similar URLs request.

In the following, we give two examples of detection with SPuNge and the correlation framework introduced in Section II-G. By looking for machines presenting a similar behavior and correlating their information, i.e. the industries and countries in which they operate, we reveal victims of potential targeted attacks.

```
Cluster 7543 - H146.185.246.116 >:R /p98a.exe >:R /dd.exe
http://146.185.246.111/p98a.exe NETWORK 1
notepad.exe 2012-11-13 09:50:35
http://146.185.246.116/p18a.exe NETWORK 1
notepad.exe 2012-11-13 09:50:37
[...]
http://146.185.246.121/mailsa.exe NETWORK 1
notepad.exe 2012-11-13 09:50:24
http://146.185.246.101/lmqa.exe NETWORK 1
notepad.exe 2012-11-13 09:50:26
http://146.185.246.63/dd.exe NETWORK 2
svchost.exe 2012-11-13 11:45:27
http://146.185.246.63/dd.exe NETWORK 3
svchost.exe 2012-11-13 20:58:55
http://146.185.246.104/dqs.exe NETWORK 1
notepad.exe 2012-11-13 09:47:36
NETWORK 1  Technology  Mexico   Windows 5 . 1
NETWORK 2  Technology  Turkey   Windows 5 . 1
NETWORK 3  Technology  Morocco  Windows 5 . 1
```

Listing 1. RBN Example - Technology Industry

In Listing 1, we have 3 distinct class B networks, all belonging to companies operating in the technology industry, and located in three separate and remote locations: Mexico, Turkey and Morocco. These events have been clustered together because sharing similar hostnames, e.g. IP addresses of contiguous space, and paths that consist of binary files with short name. This piece of malware is used to keep control of

the infected machines via a backdoor. The malware injects into the memory space of legitimate Windows programs (i.e. `notepad.exe` and `svchost.exe`) to avoid easy detections. When looking at the IP addresses that have been connected, they all belong to the same netblock, from Russia, registered with information belonging to actors operating malicious activities. This netblock has a history of badness and is associated with the infamous Russian Business Network (RBN), which is known to provide support, custom services and malware for targeted attacks operations [4], [15].

The next example of Listing 2 shows two networks, which are both identified as Malaysian companies in the oil and gas industry. The machines seated in these two networks reached out on November 14th two clusters of C&C servers, with a process called `r18nwn.exe`. Our system grouped these clusters together because exclusively accessed by the same victims and part of the same attack.

```
Group 1245,2 Clusters , 2 Networks
Cluster 1725,
Label: R/list.php?c=140C3 [...] =:Hw.nucleardiscover.com:888
E1: http://w.nucleardiscover.com:888/list.php?c=140C34E31DAB3B9746[...]&t=0.689831&v=2
E2: http//w.nucleardiscover.com:888/list.php?c=D8C08B5CD1670FA396[...]&v=1&t=0.9288141
Cluster1932, Label: R/gggg_r.jpg?t=0.1424164
E1: http://61.147.99.179:81/gggg_r.jpg?t=0.1424164
E2: http://ru.letmedo.net:2011/myck.jpg?t=0.3245672
NETWORK1: Oil and Gas Malaysia Windows 5.1
r18nwn.exe (HASH HERE) 2012-11-14
NETWORK2: Oil and Gas Malaysia Windows 5.1
r18nwn.exe (HASH HERE) 2012-11-14
```

Listing 2. Example of Cluster Group - Oil&Gas Industry.

McAfee [9] also sees this attack localized to the south Asia region and refers to it as a malware that “spreads by transmission to a removable medium such as a removable disk, writable CD, or USB drive”. This is a common methodology followed by malware authors targeting networks that might not have an easy access to the Internet, for example companies operating in industrial environments. The malware waits for commands from an attacker (as opposed to carrying out automated activity). This is very uncommon in widespread attacks, but part of the standard modus operandi of targeted attack groups, especially those based in China. We used the historical data provided by the DomainTools service [10] to verify that the domain was originally registered to a person located in China. His name is also linked to multiple other malicious domains employed in targeted attack operations.

Ethical Considerations

Real-world experiments involving data collected at customer side may be considered an ethically sensitive issue. Clearly, one question that arises is if it is ethically acceptable and justifiable to conduct experiments that involve real users. Similar to the experiments conducted by Jakobsson et al. [8], [9] and previous work in the field [17], we believe that realistic experiments are the only way to reliably estimate success rates of detection systems in the real-world. During the experiments we described in the paper, we took into account the privacy of the users, and the sensitivity of the data that was collected. In particular, identifiers (e.g., IP addresses of the machines) were anonymized, and any information that could reveal the user identity previously removed. Finally, all experiments were performed in Europe. Hence, we consulted with the legal department of our institution and our handling and privacy precautions were deemed appropriate and consistent with the European legal position.

Related Work

Thonnard et al. [16] provide an analysis of APT campaigns, focusing on studying the characteristic of a known set of targeted attacks delivered with email attachments, and evaluate the prevalence and sophistication level of said attacks by analyzing the malicious attachments used. Another paper focusing on the analysis of APTs campaign is [10]: The authors use graphs to give a broad view of APT campaigns, rather than analyzing them individually. By associating attacks with shared targets, it is possible to build a map of the APTs activities and identify clusters that could represent common activities from a single team. In [11], authors show how to determine, from a known targeted attack, the N most likely victims of the attack, by improving the performance in terms of detection rate and false positives with regards to N-gram based approaches. Despite providing the reader with insights of APTs, the above works focus on analyzing already identified campaigns, while the scope of our work is to present a new methodology that leverages clustering techniques to identify and single out possible attacks yet to be discovered.

Clustering has been widely used in data mining as a technique to find common patterns and similarities in huge sets of data; recently security researchers have applied it as a way to perform traffic aggregation and classification to identify botnets and other threats. Perdisci et al. in [14] propose a clustering-based approach to the analysis of HTTP-based malware traffic, in order to improve the generation of signatures, while in [5] they introduce a solution to identify botnets via the analysis of network traffic that require no prior knowledge, i.e. it does not require previous assumptions on the botnets' network behavior. The authors are able to identify IRC-, HTTP- and P2P-based botnets with success. In [2], a novel approach to identify DGA-based malware domains is proposed by the same authors. In lieu of reverse engineering DGA algorithms to identify domains used by botnets, the authors propose the use of clustering for the analysis of NXDomain traffic, under the assumption that most of DGA-generated DNS queries would result in NXDomain responses. Finally, [17] offers a solution to identify candidates of botnet-infected machines by analyzing the aggregation patterns of network traffic, while in our case we rely on a network traffic who has already been classified as malicious. On a different note, in [3], Ulrich et al. exploit clustering techniques to perform behavior-based malware classification. Unlike network clustering, this work clusters malware samples based on host-based features like system calls and memory objects.

Given the related works presented above, none of them address explicitly the problem of identifying potential targeted attacks, by means of malicious traffic aggregation and correlation of machines information. We believe that the opportunity for using this methodology in APTs detection is big and, to the best of our knowledge, still unexplored.

Conclusion

We introduced a novel system, that we called SPuNge, to process threat information collected on the users' side to detect potential targeted attacks for further investigation. We used a combination of clustering techniques to identify groups of machines, and networks, that are possibly involved in the same attack. In addition, we showed how we correlate industry and country information of the victims to reduce the millions of normal malicious events down to a more manageable amount for further in-deep analysis. We evaluated our system against one week of data collected by an antivirus vendor from over 20 million users installations worldwide. Our results show that our approach works well in practice and is helpful in assisting security analysts in cybercrime investigations.

REFERENCES

1. AlienVault and Kaspersky. Operation Red October. http://news.softpedia.com/news/AlienVault-and-Kaspersky-Help-Organizations-Neutralize-Red-October-Attack-322919.shtml.

2. M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX conference on Security symposium*, 2012.

3. U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA*, 2009.

4. J. Carr. RBN Connection to Kaspersky's Red October Espionage Network. http://jeffreycarr.blogspot.ca/2013/01/rbn-connectionto-kasperskys-red.html.

5. G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, 2008.

6. J. Hernandez. Typo squatting, The Threat Network Defense Teams Overlook. http://www.academia.edu/818541/TypoSquatting_-_Malicious_Domains_Malware_Domains_Part_Of_The_Miasmatic_Threat_.

7. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv*., 1999.

8. M. Jakobsson, P. Finn, and N. Johnson. Why and how to perform fraud experiments. *Security Privacy, IEEE*, 2008.

9. M. Jakobsson and J. Ratkiewicz. Designing ethical phishing experiments: a study of (rot13) ronl query features. In *Proceedings of the 15th international conference on World Wide Web*, 2006.

10. M. Lee and D. Lewis. Clustering disparate attacks: mapping the activities of the advanced persistent threat. http://www.academia.edu/2352875/CLUSTERING_DISPARATE_ATTACKS_MAPPING_THE_ACTIVITIES_OF_THE_ADVANCED_PERSISTENT_THREAT.

11. S.-T. Liu, Y.-M. Chen, and H.-C. Hung. N-victims: An approach todetermine n-victims for apt investigations. In *Information Security Applications*. 2012.

12. T. Micro. IXESHE, Targeting East Asian Governments and Electronics Companies. https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp_ixeshe.pdf.

13. D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

14. R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of httpbased malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2010.

15. Symantec. Anatomy of a Data Breach. http://www.symantec.com/articles/article.jsp?aid=20091023.

16. O. Thonnard, L. Bilge, G. O'Gorman, S. Kiernan, and M. Lee. Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat. In *Research in Attacks, Intrusions, and Defenses*, 2012.

17. T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2008.

TREND MICRO™

Trend Micro Incorporated, a global cloud security leader, creates a world safe for exchanging digital information with its Internet content security and threat management solutions for businesses and consumers. A pioneer in server security with over 20 years experience, we deliver top-ranked client, server, and cloud-based security that fits our customers' and partners' needs; stops new threats faster; and protects data in physical, virtualized, and cloud environments. Powered by the Trend Micro™ Smart Protection Network™ infrastructure, our industry-leading cloud-computing security technology, products and services stop threats where they emerge, on the Internet, and are supported by 1,000+ threat intelligence experts around the globe. For additional information, visit **www.trendmicro.com**.

Securing Your Journey
to the Cloud