



ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates

Terry Nelms, Damballa, Inc. and Georgia Institute of Technology;

Roberto Perdisci, University of Georgia and Georgia Institute of Technology;

Mustaque Ahamad, Georgia Institute of Technology and New York University Abu Dhabi

**This paper is included in the Proceedings of the
22nd USENIX Security Symposium.**

Washington, D.C., USA

ISBN 978-1-931971-03-4

**Open access to the Proceedings of the
22nd USENIX Security Symposium
is sponsored by USENIX**

ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates

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

¹Damballa, Inc.

²Georgia Institute of Technology – College of Computing

³University of Georgia – Dept. of Computer Science

⁴New York University Abu Dhabi

Abstract

In this paper, we present ExecScent, a novel system that aims to mine new, previously unknown C&C domain names from *live* enterprise network traffic. ExecScent automatically learns control protocol templates (CPTs) from examples of known C&C communications. These CPTs are then *adapted* to the “background traffic” of the network where the templates are to be deployed. The goal is to generate *hybrid* templates that can *self-tune* to each specific deployment scenario, thus yielding a better trade-off between true and false positives for a given network environment. To the best of our knowledge, ExecScent is the first system to use this type of adaptive C&C traffic models.

We implemented a prototype version of ExecScent, and deployed it in three different large networks for a period of two weeks. During the deployment, we discovered many new, previously unknown C&C domains and hundreds of new infected machines, compared to using a large up-to-date commercial C&C domain blacklist. Furthermore, we deployed the new C&C domains mined by ExecScent to six large ISP networks, discovering more than 25,000 new infected machines.

1 Introduction

Code reuse is common practice in malware [18, 20]. Often, new (polymorphic) malware releases are created by simply re-packaging previous samples, or by augmenting previous versions with a few new functionalities. Moreover, it is not uncommon for the source code of successful malware to be sold or leaked on underground forums, and to be reused by other malware operators [14].

Most modern malware, especially botnets, consist of (at least) two fundamental components: a *client agent*, which runs on victim machines, and a *control server* application, which is administered by the malware owner.

Because code reuse applies to both components¹, this naturally results in *many different malware samples sharing a common command-and-control (C&C) protocol*, even when control server instances owned by different malware operators use different C&C domains and IPs.

In this paper, we present ExecScent, a novel system that aims to mine new, previously unknown C&C domain names from *live* enterprise network traffic (see Figure 1). Starting from a *seed* list of known C&C communications and related domain names found in malware-generated network traces, ExecScent aims to discover new C&C domains by taking advantage of the commonalities in the C&C protocol shared by different malware samples. More precisely, we refer to the C&C protocol as the set of specifications implemented to enable the malware control application logic, which is defined at a higher level of abstraction compared to the underlying transport (e.g., TCP or UDP) or application (e.g., HTTP) protocols that facilitate the C&C communications. ExecScent aims to automatically learn the unique traits of a given C&C protocol from the seed of known C&C communications to derive a *control protocol template* (CPT), which can in turn be deployed at the edge of a network to detect traffic destined to new C&C domains.

ExecScent builds *adaptive* templates that also learn from the *traffic profile* of the network where the templates are to be deployed. The goal is to generate *hybrid* templates that can *self-tune* to each specific deployment scenario, thus yielding a better trade-off between true and false positives for a given network environment. The intuition is that different networks have different traffic profiles (e.g., the network of a financial institution may generate very different traffic compared to a technology company). It may therefore happen that a CPT could (by chance) raise a non-negligible number of false posi-

¹For example, web-based malware control panels can be acquired in the Internet underground markets and re-deployed essentially *as is*, while the client agents can be obtained using *do-it-yourself* malware creation kits [10].

tives in a given network, say Net_A , while generating true C&C domain detections and no false positives in other networks. We take a pragmatic approach, aiming to automatically identify these cases and lowering the “confidence” on that CPT *only* when it is deployed to Net_A . This allows us to lower the overall risk of false positives, while maintaining a high probability of detection in other networks. We further motivate the use of adaptive templates in Section 3.

ExecScent focuses on HTTP-based C&C protocols, because studies have shown that HTTP-based C&C communications are used by a large majority of malware families [26] and almost all known mobile bots [31]. Moreover, many enterprise networks employ strict egress filtering firewall rules that block all non-web traffic. This forces malware that target enterprise networks to use HTTP (or HTTPS) as the communication protocol of choice. It is also important to notice that many modern enterprise networks deploy web proxies that enforce SSL man-in-the-middle² (SSL-MITM). Therefore, enterprise networks can apply ExecScent’s templates at the web proxy level to discover new C&C domains even in cases of HTTPS-based C&C traffic.

Our system is different from previously proposed URL-based C&C signature generation systems [23, 30]. Unlike previous work, we build templates that can adapt to the deployment network and that model the entire content of HTTP requests, rather than being limited to the URL string. We show in Section 5 that this allows us to obtain a much better trade-off between true and false positives, compared to “statically” modeling only the URL of C&C requests.

Anomaly-based botnet detection systems such as [15, 16] typically require that more than one host in the monitored network be compromised with the same malware type, do not scale well to large networks, and are not able to directly attribute the suspected C&C communications to a specific malware family. Unlike these systems, ExecScent can detect C&C communications initiated by a single malware-infected machine with low false positives, and can attribute the newly discovered C&C domains to a known malware family name or malware operator (i.e., the name of the cyber-criminal group behind the malware operation). In turn, the *labeled* C&C domain names discovered by ExecScent may also be deployed in existing lightweight malware detection systems based on DNS traffic inspection, thus contributing to the detection and attribution of malware infections in very large networks (e.g., ISP networks) where monitoring all traffic may not be practically feasible.

Currently, we identify the seed of known C&C traffic required by ExecScent to learn the control protocol tem-

²See <http://crypto.stanford.edu/ssl-mitm/>, for example.

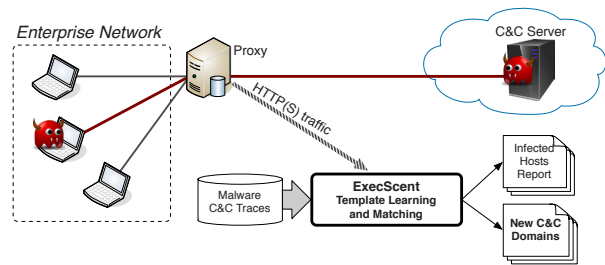


Figure 1: ExecScent deployment overview. Adaptive control protocol templates are learned from both malware-generated network traces and the live network traffic observation. The obtained adaptive templates are matched against new network traffic to discover new C&C domains.

plates by leveraging blacklists of known C&C domain names. C&C discovery systems based on dynamic analysis such as Jackstraws [17] may also be used for this purpose. However, unlike ExecScent, while Jackstraws may be useful to find “seed” C&C traffic, its system-call-based detection models [17] cannot be deployed to detect new C&C domains in live network traffic.

In summary, we make the following contributions:

- We present ExecScent, a novel system for mining new malware C&C domains from live networks. ExecScent automatically learns C&C traffic models that can adapt to the deployment network’s traffic. This *adaptive* approach allows us to greatly reduce the false positives while maintaining a high number of true positives. To the best of our knowledge, ExecScent is the first system to use this type of adaptive C&C traffic models.
- We implemented a prototype version of ExecScent, and deployed it in three different large networks for a period of two weeks. During the deployment, we discovered many new, previously unknown C&C domains and hundreds of new infected machines, compared to using a large up-to-date commercial C&C domain blacklist.
- We deployed the new C&C domains mined by ExecScent to six large ISP networks, discovering more than 25,000 new infected machines.

2 System Overview

The primary goal of ExecScent is to generate control protocol templates (CPTs) from a seed of known malware-generated HTTP-based C&C communications. We then use these CPTs to identify new, previously unknown C&C domains.

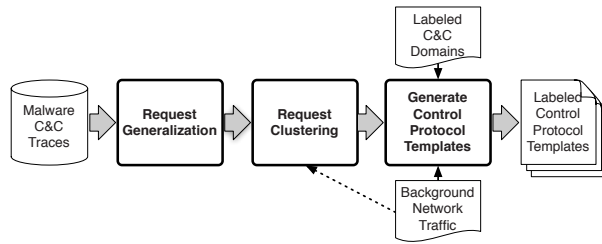


Figure 2: ExecScent system overview.

ExecScent automatically finds common traits among the C&C protocol used by different malware samples, and encodes these common traits into a set of CPTs. Each template is labeled with the name of the malware family or (if known) criminal operator associated with the C&C traffic from which the CPT is derived. Once a CPT is deployed at the edge of a network (see Figure 1), any new HTTP(S) traffic that matches the template is classified as C&C traffic. The domain names associated with the matched traffic are then flagged as C&C domains, and attributed to the malware family or operator with which the CPT was labeled.

Figure 2 presents an overview of the process used by ExecScent to generate and label the CPTs. We briefly describe the role of the different system components in this section, deferring the details to Section 4.

Given a large repository of malware-generated network traces, we first reconstruct all HTTP requests performed by each malware sample. Then, we apply a *request generalization* process, in which (wherever possible) we replace some of the request parameters (e.g., URL parameter values) with their data type and length, as shown in the example in Figure 3. Notice that ExecScent considers the entire content of the HTTP requests, not only the URLs (see Section 3.2), and the generalization process is applied to different parts of the request header. The main motivation for applying the generalization step is to improve the accuracy of the *request clustering* process, in which we aim to group together malware-generated requests that follow a similar C&C protocol.

Once the malware requests have been clustered, we apply a *template learning* process in which we derive the CPTs. Essentially, a CPT summarizes the (generalized) HTTP requests grouped in a cluster, and records a number of key properties such as the structure of the URLs, the set of request headers, the IP addresses contacted by the malware, etc. Furthermore, the templates associate a malware-family label to each template (see Section 4.4 for details).

Before the templates are deployed in a network, we *adapt* the CPTs to the “background traffic” observed in

that network. In particular, for each template component (e.g., the generalized URL path, the user-agent string, the request header set, etc.), we compute how frequently the component appeared in the deployment network. CPT components that are “popular” in the background traffic will be assigned a lower “match confidence” for that network. On the other hand, components that appear very infrequently (or not at all) in the traffic are assigned a higher confidence. We refer to these “rare” components as having high *specificity*, with respect to the deployment network’s traffic. The intuitions and motivations for this approach are discussed in more detail in the next section.

After deployment, an HTTP request is labeled as C&C if it matches a CPT with *high similarity and specificity*. That is, if the request closely matches a CPT and the matching CPT components have high specificity (i.e., rarely appeared) in that particular deployment network.

3 Approach Motivations and Intuitions

In this section, we discuss the intuitions that motivated us to build adaptive control protocol templates. Furthermore, we discuss the advantages of considering the entire content of C&C HTTP requests, rather than limiting ourselves to the URL strings, as done in previous work [23, 30].

3.1 Why Adaptive Templates?

As most other traffic models, ExecScent’s CPTs, which are derived from and therefore can match C&C traffic, may be imperfect and could generate some false positives. To minimize this risk, ExecScent builds *adaptive* control protocol templates that, besides learning from known malware-generated C&C traffic, also learn from the traffic observed in the network where the templates are being deployed. Our key observation is that different enterprise networks have different traffic profiles. The traffic generated by the computer network of a financial institute (e.g., a large bank) may look quite different from traffic at a manufacturing company (e.g., a car producer) or a technology company (e.g., a software-development company). It may therefore happen that a CPT could (by chance) raise a non-negligible number of false positives in a given network, say Net_X , and several true detections with no or very few false positives in other networks. Intuitively, our objective is to automatically identify these cases, and lower the “confidence” on that template when it matches traffic from Net_X , while keeping its confidence high when it is deployed elsewhere.

For example, assume Net_B is a US bank whose hosts have rarely or never contacted IPs located, say, in China. If an HTTP request towards an IP address in China is found, this is by itself an anomalous event. Intuitively, if

the request also matches a CPT, our confidence on a correct match (true C&C communication) can be fairly high. On the other hand, assume Net_A is a car manufacturer with partners in China, with which Net_A 's hosts communicate frequently. If an HTTP request in Net_A matches a CPT but is directed towards an address within one of the IP ranges of the manufacturer's partners, our confidence on a correct match should be lowered.

More specifically, consider the following hypothetical scenario. Assume we have a template τ that matches an HTTP request in both Net_A and Net_B with a *similarity* score s . For simplicity, let us assume the score s is the same for both Net_A 's traffic and Net_B 's traffic. Suppose also that the server's IP (or it's /24 prefix) associated with the matching traffic is ip_a for Net_A and ip_b for Net_B . Also, suppose that ip_a is "popular" in network Net_A , whereas ip_b has very low popularity in Net_B because it has never been contacted by hosts in that network. Because ip_a is very popular in Net_A , meaning that a large fraction (e.g., more than 50%) of the hosts in Net_A has contacted the domain in the past, it is likely that the template τ is fortuitously matching benign traffic, thus potentially causing a large number of false positives in Net_A . On the other hand, because ip_b has very low popularity in Net_B , it is more likely that the match is a true detection, or that in any case τ will generate very few (potentially only one) false positives in Net_B . Consequently, based on a model of recent traffic observed in Net_A and Net_B , we should lower our confidence in τ for the matches observed in Net_A , but not for Net_B . In other words, τ should automatically *adapt* to Net_A to "tune down" the false positives. At the same time, keeping the confidence in τ high for Net_B means that we will still be able to detect C&C communications that match τ , while keeping the risk of false positives low. We generalize this approach to *all* other components of ExecScent's templates (e.g. the structure of the URLs, the user-agent strings, the other request headers, etc.), in addition to the destination IPs.

Overall, our confidence on a match of template τ in a given network Net_X will depend on two factors:

- *Similarity*: a measure of how closely an HTTP request matches τ .
- *Specificity*: a measure of how specific (or *rare*) are the components of τ with respect to Net_X 's traffic.

An HTTP request is labeled as C&C if it matches a CPT with both high similarity and high specificity. We show in Section 5 that this approach outperforms C&C models that do not take such specificity into account.

3.2 Why Consider All Request Content?

Malware C&C requests typically need to carry enough information for a malware agent running on a victim to (loosely) authenticate itself with the C&C server. Intuitively, the C&C server wants to make sure that it is talking to one of its *bots*, thus avoiding exposure of its true nature or functionalities to crawlers or security researchers who may be probing the server as part of an investigation. This is often achieved by using a specific set of parameter names and values that must be embedded in the URL for the C&C requests to be successful. Previous work on automatic URL signature generation has shown promising results in such cases [23,30]. However, some malware (e.g., TDL4 [1]) exchanges information with the C&C by first encrypting it, encoding it (e.g., using base-64 encoding), and embedding it in the URL path. Alternatively, identifier strings can also be embedded in fields such as *user-agent* (e.g., some malware samples use their MD5 hash as *user-agent* name), encoded in other request headers (e.g., in the *referrer*), or in the body of POST requests. Therefore, only considering URLs may not be enough to accurately model C&C requests and detect new C&C domains, as supported by our experimental results (Section 5).

4 System Details

We now detail the internals of ExecScent. Please refer to Section 2 for a higher-level overview of the entire system.

4.1 Input Network Traffic

As we mentioned in Section 1, ExecScent focuses on HTTP-based malware, namely malware that leverage HTTP (or HTTPS) as a base network protocol on top of which the malware control protocol is "transported". To this end, ExecScent takes in as input a feed of malware-generated HTTP traffic traces (in our evaluation, we use a large set of malware traces provided to us by a well-known company that specializes in malware defense).

It is worth remembering that while some malware may use HTTPS traffic as a way to evade detection, this does not represent an insurmountable obstacle in our deployment scenarios (see Figure 1). In fact, many enterprise networks, which represent our target deployment environment, already deploy web proxy servers that can perform SSL-MITM and can therefore forward the clear-text HTTP requests to ExecScent's template matching module, e.g., using the ICAP protocol (RFC 3507). Also, malware samples that appear to be using HTTPS traffic may be re-analyzed in a controlled environment that includes an SSL-MITM proxy interposed between the (vir-

tual) machine running the sample and the egress router. After all, HTTPS-based malware that do not support or choose not to run when an SSL-MITM proxy is present will also fail to run in enterprise networks that have a similar setting, and are therefore of less interest.

4.2 Request Generalization

As we discuss in the following sections, to obtain quality control protocol templates we first need to group similar C&C requests. To this end, an appropriate similarity metric needs to be defined before clustering algorithms can be applied. Previous works that propose URL-centric clustering systems [23,30] are mainly based on string similarity measures. Essentially, two URLs are considered similar if they have a small edit distance, or share a number of substrings (or tokens). However, these systems do not take into account the fact that URLs often contain variables whose similarity is better measured according to their data *type* rather than considering specific sequences of characters. Consider the two hypothetical C&C requests in Figure 3. Taken as they are (Figure 3a), their distance is relatively large, due to the presence of several different characters in the strings. To avoid this, ExecScent uses a set of heuristics to detect strings that represent data of a certain type, and replaces them accordingly using a placeholder tag containing the data type and string length (Figure 3b).

For example, we would identify “fa45e” as lowercase hexadecimal because it contains numeric characters and the alphabetic characters are all valid lowercase hexadecimal digits. The data types we currently identify are *integer*, *hexadecimal* (upper, lower and mixed case), *base64* (standard and “URL safe”) and *string* (upper, lower and mixed case). In addition, for integer, hexadecimal and string we can identify the data type plus additional punctuation such as “:” or “.” (e.g., 192.168.1.1 would be identified as a data type of integer+period of length 11). Furthermore, our heuristics can easily be extended to support data types such as IP address, MAC address, MD5 hash and version number.

This generalization process allows us to define a better similarity metric (Section 4.7), which is instrumental to obtaining higher quality C&C request clusters. Notice also that while previous works such as [23,30] focus only on URL strings, ExecScent takes the entire request into account. For example, in Figure 3 the user-agent strings are MD5s, and can be generalized by replacing the specific MD5 strings with the appropriate data type and length information.

(a)	Request 1: GET /Ym90bmV0DQo=/cnc.php?v=121&cc=IT Host: www.bot.net User-Agent: 680e4a9a7eb391bc48118baba2dc8e16 ...
	Request 2: GET /bWFSd2FyZQ0KDQo=/cnc.php?v=425&cc=US Host: www.malwa.re User-Agent: dae4a66124940351a65639019b50bf5a ...
(b)	Request 1: GET /<Base64;12>/cnc.php?v=<Int;3>&cc=<Str;2> Host: www.bot.net User-Agent: <Hex;32> ...
	Request 2: GET /<Base64;16>/cnc.php?v=<Int;3>&cc=<Str;2> Host: www.malwa.re User-Agent: <Hex;32> ...

Figure 3: Example C&C requests: (a) original; (b) generalized.

4.3 Request Clustering

Before extracting the templates, we group together similar C&C requests. This clustering step simply aims to assist the automatic CPT generation algorithm, improving efficiency and yielding templates that are at the same time *generic* enough to match similar (but not identical) C&C communications in new traffic, and *precise* enough to generate very few or no false positives.

We perform C&C request clustering in two phases. During the first phase, we coarsely group C&C requests based on their destination IPs. Specifically, given two C&C requests, we group them together if their destination IPs reside in /24 (or class C) networks that *share a DNS-based relationship*. Namely, we consider two /24 networks as related if there exists at least one domain name that within the last 30 days resolved to different IP addresses residing in the two different networks. To find such relationships, we rely on a large passive DNS database [12].

In the second phase, we consider one coarse-grained cluster at a time, and we further group a cluster’s C&C requests according to a content similarity function. We use an agglomerative hierarchical clustering algorithm to group together C&C requests within a coarse-grained cluster that carry similar generalized URLs, similar user-agent strings, similar numbers of HTTP header fields and respective values, etc. When measuring the similarity between two requests, we take into account both the *similarity* and *specificity* of the requests’ content, where the specificity (or low “popularity”) can be measured with respect to a dataset of traffic recently collected from different networks (dashed arrow in Figure 2). For a more detailed definition of the similarity

τ_1) Median URL path: /<Base64;14>/cnc.php
τ_2) URL query component: {v=<Int;3>, cc=<String;2>}
τ_3) User Agent: {<Hex;32>}
τ_4) Other headers: {(Host;13), (Accept-Encoding;8)}
τ_5) Dst nets: {172.16.8.0/24, 10.10.4.0/24, 192.168.1.0/24}
Malware family: {Trojan-A, BotFamily-1}
URL regex: GET /.*\?(cclv)=
Background traffic profile: specificity scores used to adapt the CPT to the deployment environment

Figure 4: Example CPT.

function used in the clustering step, we refer the reader to Section 4.7.

4.4 Generating CPTs

Once C&C requests have been clustered, a control protocol template (CPT) is generated from each cluster. At this stage, we consider only clusters that contain at least one HTTP request to a known C&C domain. Each template represents a summary of all C&C requests in a cluster, and contains the following components, as also shown in Figure 4:

- τ_1) *Median URL path*: median path string that minimizes the sum of edit distances from all URL paths in the requests (see [11] for a definition of median string). *Intuition*: although the URL path may vary significantly from one malware installation to another, we observed many cases in which there exist “stable” path components that are unique to a specific malware family or operation.
- τ_2) *URL query component*: stores the set of parameter names, value types and lengths observed in the query component [5] of each of the URLs. *Intuition*: URL parameters are often used by malware to convey information about the infected host, such as its OS version, a unique identifier for the infected machine, etc.
- τ_3) *User-agent*: the set of all different (generalized) user-agent strings found in the requests. *Intuition*: the user-agent is one of the most abused HTTP headers by malware, and is sometimes used as a loose form of authentication.
- τ_4) *Other headers*: the set of other HTTP headers observed in the requests. For each header, we also store the length of its value string. *Intuition*: the set of header names, their order and values are sometimes unique to a malware family.

τ_5) *Dst. networks*: the set of all destination /24 networks associated with the C&C requests in the cluster. *Intuition*: in some cases, the C&C server may be relocated to a new IP address within the same (possibly “bullet-proof”) network.

- *Malware family*: the (set of) malware family name(s) associated to the known C&C requests in the cluster.

In addition, each CPT includes the following deployment-related information:

- *URL regex*: to increase the efficiency of the template matching phase (Section 4.6), each template includes a regular expression automatically generated from the set of URL strings in the requests. The URL regex is intentionally built to be very generic and is used during deployment for the sole purpose of filtering out traffic that is extremely unlikely to closely match the entire template, thus reducing the cost of computing the similarity between HTTP requests in live traffic and the template.
- *Background traffic profile*: information derived from the traffic observed in the deployment environment within the past W days (where W is a system parameter). This is used for computing the specificity of the CPT components, thus allowing us to adapt the CPT to the deployment network, as explained in detail in Section 4.5.

Notice that a CPT acts as the *centroid* for the cluster from which it was derived. To determine if a new request is similar enough to a given cluster, we only need to compare it with the CPT, rather than all of the clustered C&C requests. Therefore, CPTs provide an efficient means of measuring the similarity of a new request to the C&C protocol used by the clustered malware samples.

4.5 Adapting to a Deployment Network

As explained in Section 3.1, once the CPTs are deployed, an HTTP request is labeled as C&C if it matches a CPT τ with both high *similarity* and *specificity*. To this end, we first need to compute a specificity score for each element of the k -th component τ_k of τ , which indicates how “unpopular” that element is with respect to the traffic profile in the deployment network (notice that $k = 1, \dots, 5$, as shown in Figure 4 and Section 4.4).

For example, to compute the specificity scores for τ_3 , we first compute a *host-based popularity* score hp_{ua_i} for each user-agent string ua_i in the set τ_3 . We consider the number of hosts hn_{ua_i} in the deployment network that generated an HTTP request containing ua_i during the last

W days, where W is a configurable time-window parameter. We define $hp_{ua_i} = \frac{hn_{ua_i}}{\max_j \{hn_{ua_j}\}}$, where the max is taken over all user-agent strings ua_j observed in the deployment network's traffic. Similarly, we compute a domain-based popularity score dp_{ua_i} , based on the number of distinct destination domain names dn_{ua_i} with one or more HTTP requests that contain ua_i . We define $dp_{ua_i} = \frac{dn_{ua_i}}{\max_j \{dn_{ua_j}\}}$. The intuition is that a user-agent string can only be considered truly popular if it spans many hosts and domains. On the other hand, we do not want to consider a ua_i as very popular if it has high host-based popularity (e.g., "Windows-Update-Agent") but low domain-based popularity (e.g., because the only domain on which it is used is microsoft.com). Finally, we define the specificity score for ua_i as $\sigma_{3,ua_i} = 1 - \min(hp_{ua_i}, dp_{ua_i})$. In a similar way, we compute a specificity score σ_{4,hd_i} for each header element hd_i in τ_4 .

To compute the specificity scores for τ_5 , we simply compute the host-based popularity hp_{net_i} for each /24 network prefix $net_i \in \tau_5$, and we define a separate score $\sigma_{5,net_i} = (1 - hp_{net_i})$ for each prefix.

4.5.1 URL Specificity

Computing the specificity of the components of a URL is more complex, due to the large variety of unique URLs observed every day on a given network. To address this problem, we rely on a supervised classification approach. First, given a dataset of traffic collected from a large network, we extract all URLs, and learn a map of URL word frequencies, where the "words" are extracted by tokenizing the URLs (e.g., extracting elements of the URL path, filename, query string, etc.). Then, given a new URL, we translate it into a feature vector in which the statistical features measure things such as the average frequency of single "words" in the tokenized URL, the average frequency of word bigrams in the query parameters, the frequency of the file name, etc. (to extract the frequency values for each word found in the URL we lookup the previously learned map of word frequencies).

After we translate a large set of "background traffic URLs" into feature vectors, we train an SVM classifier [8] that can label new URLs as either *popular* or *unpopular*. To prepare the training dataset we proceed as follows. We first rank the "background URLs" according to their domain-based popularity (i.e., URLs that appear on requests to multiple sites on different domain names are considered as more popular). Then, we take a sample of URLs from the top and from the bottom of this ranking, which we label as *popular* and *unpopular*, respectively. We use this labeled dataset to train the SVM classifier, and we rely on the max-margin approach used by the SVM [9] to produce a model that can generalize

to URLs not seen during training.

During the operational phase (once the SVM classifier is trained and deployed), given a URL u_i , we can first translate u_i into its corresponding feature vector v_i , as described above, and feed v_i to the SVM classifier. The classifier can then label u_i as either *popular* or *unpopular*. In practice, though, rather than considering these class labels, we only take into account the classification score (or confidence) associated with the *popular* class³. Therefore, the SVM's output can be interpreted as follows: the higher the score, the more u_i "looks like" a popular URL, when compared to the large set of URLs observed in the background traffic. Finally, the specificity score for the URL is computed as $\sigma_{u_i} = 1 - p_{u_i}$, where p_{u_i} is the SVM output for URL u_i .

Now, let us go back to consider the template τ and its URL-related components τ_1 and τ_2 (see Figure 4). We first build a "median URL" u_m by concatenating the median URL path (τ_1) to the (sorted) set of generalized parameter names and values (τ_2). We then set the similarity scores $\sigma_1 = \sigma_2 = \sigma_{u_m}$, where σ_{u_m} is the specificity of u_m .

4.6 Template Matching

Template matching happens in two phases. As mentioned above, each template contains an URL regular expression automatically derived from the C&C requests in a cluster. Given a new HTTP request r , to test whether this request matches a template τ , we first match r 's URL to τ 's URL regex. It is worth noting that, as mentioned in Section 4.4, the URL regex is intentionally built to be very generic, and is merely used to efficiently filter out traffic that is extremely unlikely to match the entire template. Furthermore, we check if the destination IP of r resides within any of the /24 prefixes in τ (specifically in component τ_5). If neither the URL regex nor the destination IP have a match, we assume r does not match τ . Otherwise, we proceed by considering the entire content of request r , transforming r according to the request generalization process (see Section 4.2), and measuring the overall matching score $S(r, \tau)$ between the (generalized) request r and the template τ .

In summary, the score S is obtained by measuring the similarity between all the components of the request r and the respective components of the template τ . These similarity measures are then weighted according to their specificity, and the matching score $S(r, \tau)$ is computed as the average of all weighted component similarities. A detailed definition of the similarity functions and how specificity plays an explicit role in computing $S(r, \tau)$ is given in Section 4.7.

³We calibrate the classification scores output by the SVM classifier using the method proposed by Platt [24].

If $S(r, \tau)$ exceeds a tunable detection threshold θ , then the request r will be deemed a C&C request and the domain name associated with r (assuming r is not using a hardcoded IP address) is classified as C&C domain and labeled with the malware family associated to τ . Furthermore, the host from which the request r originated is labeled as compromised with τ 's malware family.

4.7 Similarity Functions

4.7.1 CPT matching score

To determine if a new HTTP request r matches a CPT τ , we compute a *matching score* $S(r, \tau)$ as follows:

$$S(r, \tau) = \frac{\sum_k w_k(s_k, \sigma_k) \cdot s_k(r_k, \tau_k)}{\sum_k w_k(s_k, \sigma_k)} \cdot \sigma_d \quad (1)$$

where s_k is a similarity function that compares each element τ_k of τ (Section 4.4) with its respective counterpart r_k of r , and where w_k is a *dynamic weight* (whose definition is given below) that is a function of both the similarity s_k and the specificity σ_k of the k -th component of τ . The denominator scales $S(r, \tau)$ between zero and one.

The factor σ_d is the *specificity of the destination domain* d of request r , which is computed as $\sigma_d = 1 - \frac{m_d}{\max_i \{m_{d_i}\}}$, where m_d is the number of hosts in the deployment network's traffic that queried domain d , and $\max_i \{m_{d_i}\}$ is the number of hosts that queried the most "popular" domain in the traffic. Accordingly, we use σ_d to decrease the matching score $S(r, \tau)$ for low-specificity domains (i.e., domains queried by a large number of hosts). The intuition is that infections of a specific malware family often affect a relatively limited fraction of all hosts in an enterprise network, as most modern malware propagate relatively "slowly" via drive-by downloads or social engineering attacks. In turn, it is unlikely that a new C&C domain will be queried by a very large fraction (e.g., > 50%) of all hosts in the monitored network, within a limited amount of time (e.g., one day).

In the following, we describe the details of the similarity functions $s_k(\cdot)$ used in Equation 1. In addition, we further detail how the specificity value of each component is selected, once the value of $s_k(\cdot)$ has been computed (for the definition of specificity, we refer the reader to Section 4.5).

s_1 - Given the path of the URL associated with r , we measure the normalized edit distance between the path and the CPT's median URL path τ_1 . The URL path specificity σ_1 is computed as outlined in Section 4.5.

s_{2a} - We measure the Jaccard similarity⁴ between the set of parameter names in the URL query-string of r

⁴ $J = \frac{|A \cap B|}{|A \cup B|}$

and the set of names in τ_2 . The specificity of the parameter names σ_{2a} is equal to σ_2 (see Section 4.5).

s_{2b} - We compare the data types and lengths of the values in the generalized URL query-string parameters (see Section 4.2). For each element of the query string, we assign a score of one if its data type in r matches the data type recorded in τ_2 . Furthermore, we compute the ratio between the value length in r and in τ_2 . Finally, s_{2b} is computed by averaging all these scores, whereby the more data types and lengths that match, the higher the similarity score. As in s_{2a} , we set $\sigma_{2b} = \sigma_2$.

s_3 - We compute the normalized edit distance between the (generalized) user-agent string in r , and each of the strings in the set τ_3 . Let d_m be the smallest of such distances, where m is the closest of the template's user-agent strings. We define $s_3 = 1 - d_m$, and set the specificity $\sigma_3 = \sigma_{3,m}$.

s_4 - Given the remaining request header fields in r , we measure the similarity from different perspectives. First, we compute the Jaccard similarity j between the set of headers in r and the set τ_4 . Furthermore, we consider the order of the headers as they appear in r and in the requests from which τ was derived. If the order matches, we set a variable $o = 1$, otherwise we set $o = 0$. Finally, for each header, we compare the ratio between the length of its value as it appears in r and in τ_5 , respectively. The similarity s_4 is defined as the average of all these partial similarity scores (i.e., of j , o , and the length ratios). We set the specificity score $\sigma_5 = \min_l \{\sigma_{5,hd_l}\}$, where the hd_l are the request headers.

s_5 - Let ρ be the destination IP of request r . If ρ resides within any of the /24 network prefixes in τ_5 , we set $s_5 = 1$, otherwise we assign $s_5 = 0$. Assume ρ is within prefix $n \in \tau_5$ (in which case $s_5 = 1$). In this case, we set the specificity $\sigma_5 = \sigma_{5,n}$.

The dynamic weights $w_k(\cdot)$ are computed as follows:

$$w_k(s_k, \sigma_k) = \hat{w}_k \cdot \left(1 + \frac{1}{(2 - s_k \cdot \sigma_k)^n} \right) \quad (2)$$

where $\hat{w}_k$ is a *static weight* (i.e., it takes a fixed value), and n is a configuration parameter. Notice that $w_k \in [\hat{w}_k(1 + \frac{1}{2^n}), 2\hat{w}_k]$, and that these weights are effectively normalized by the denominator of Equation 1, thus resulting in $S(r, \tau) \in [0, 1]$ (since $s_k \in [0, 1], \forall k$, and $\sigma_d \in [0, 1]$, by definition).

The intuition for the dynamic weights $w_k(\cdot)$ is that we want to give higher weight to components of a request r that match their respective counterpart in a CPT τ with

both high similarity and high specificity. In fact, the weight will be maximum when both the similarity and specificity are equal to one, and will tend to the minimum when either the similarity or specificity (or both) tend to zero.

In summary, *similarity measures the likeness of two values*, whereas *specificity measures their uniqueness* in the underlying network traffic. The dynamic weights allow us to highlight the *rare structural elements* that are common between a CPT and a request, so that we can leverage them as the dominant features for detection. Because rare structural elements differ in their importance across malware families, by emphasizing these “unique features” we are able to detect and distinguish between different malware families.

4.7.2 Similarity function for clustering phase

In Section 4.3, we have described the C&C request clustering process. In this section we define the function used to compute the similarity between pairs of HTTP requests, which is needed to perform the clustering.

Given two HTTP requests r_1 and r_2 , we compute their similarity using Equation 1. At this point, the reader may notice that Equation 1 is defined to compare an HTTP request to a CPT, rather than two requests. The reason why we can use Equation 1, is that we can think of a request as a CPT derived from only one HTTP request. Furthermore, if we want to include the specificity scores, which are used to make the weights w_k dynamic, we can use a dataset of traffic previously collected from one or more networks (see dashed arrow in Figure 2).

5 Evaluation

In this section, we describe the data used to evaluate ExecScent (Section 5.1), how the system was setup to conduct the experiments (Section 5.2), and present the experimental results in different live networks (Section 5.3). Furthermore, we quantify the advantage of modeling entire HTTP requests, rather than only considering URLs, and of using adaptive templates over “static” C&C models (Section 5.4). In addition, we show the benefits obtained by deploying new C&C domains discovered by ExecScent into large ISP networks (Section 5.5).

5.1 Evaluation Data

5.1.1 Malware Network Traces

We obtained access to a commercial feed of malware intelligence data (provided to us by a well known security

company), which we used to generate the control protocol templates (CPTs). Through this feed, we collected about 8,000 malware-generated network traces per day that contained HTTP traffic. Each network trace was marked with a hash of the malware executable that generated the network activity, and (if known) by the related malware family name.

5.1.2 Live Network Traffic

To evaluate ExecScent, we had access to the *live* traffic of three large production networks, which we refer to as UNETA, UNETB, and FNET. Networks UNETA and UNETB are two different academic networks based in the US, while FNET is the computer network of a large North-American financial institution. Table 1 reports statistics with respect to the network traffic observed in these three networks. For example, in UNETA we observed an average of 7,893 distinct active source IP addresses per day. In average, these network hosts generated more than 34.8M HTTP requests per day, destined to 149,481 different domain names (in average, per day).

Table 1: Live Network Traffic Statistics (Avg. per day)

	UNETA	UNETB	FNET
<i>Distinct Src IPs</i>	7,893	27,340	7,091
<i>HTTP Requests</i>	34,871,003	66,298,395	58,019,718
<i>Distinct Domains</i>	149,481	238,014	113,778

5.1.3 Ground Truth

To estimate true and false positives, we rely on the following data:

- CCBL: we obtained a large black-list containing hundreds of thousands of C&C domains provided by a well known security company, which we refer to as CCBL. It is worth noting that CCBL is different from most publicly available domain black-lists for two reasons: 1) the C&C domains are carefully vetted by professional threat analysts; 2) the domains are labeled with their respective malware families and, when available, a malware operator name (i.e., an identifier for the cyber-criminal group that operates the C&C).
- ATWL: we derived a large white-list of *benign* domain names from Alexa’s top 1 million global domains list (a.alex.com). From these 1M domains, we filtered out domains that can be considered as *effective* top level domains⁵ (TLDs), such as domains related to dynamic DNS services (e.g., dyn-dns.org, no-ip.com, etc.). Next, we discarded domains that have not been in the top 1M list for at

⁵<http://publicsuffix.org>

least 90% of the time during the entire past year. To this end, we collected an updated top domains list every day for the past year, and only considered as benign those domains that have *consistently* appeared in the top 1M domains list. The purpose of this filtering process is to remove possible noise due to malicious domains that may became popular for a limited amount of time. After this pruning operations, we were left with about 450,000 popular domain names⁶.

- **PKIP:** we also maintain a list of parking IPs, PKIP. Namely, IP addresses related to *domain parking* services (e.g., IPs pointed to by expired or unused domains which have been temporarily taken over by a registrar). We use this list to prune ExecScent's templates. In fact, CPTs are automatically derived from HTTP requests in malware-generated network traces that are labeled as C&C communications due to their associated domain name being in the CCBL list (Section 4). However, some of the domains in CCBL may be expired, and could be currently pointing to a parking site. This may cause some of the HTTP requests in the malware traces to be erroneously labeled as C&C requests, thus introducing noise in ExecScent's CPTs. We use the PKIP to filter out this noise.
- **Threat Analysis:** clearly, it is not feasible to obtain complete ground truth about all traffic crossing the perimeter of the live networks where we evaluated ExecScent. To compensate for this and obtain a better estimate of the false and true positives (compared to only using CCBL and ATWL), we performed an extensive manual analysis of our experimental results with the help of professional threat analysts.

5.2 System Setup

To conduct our evaluation, we have implemented and deployed a Python-based proof-of-concept version of ExecScent. In this section we discuss how we prepared the system for live network deployment.

5.2.1 Clustering Parameters

As discussed in Section 4.3, to generate the CPTs, we first apply a request clustering step. The main purpose of this step is to improve the efficiency of the CPT learning process. The clustering phase relies on a hierarchical clustering algorithm that takes in as input the height at which the dendrogram (i.e., the “distance tree” generated

⁶More precisely, second level domains (2LDs).

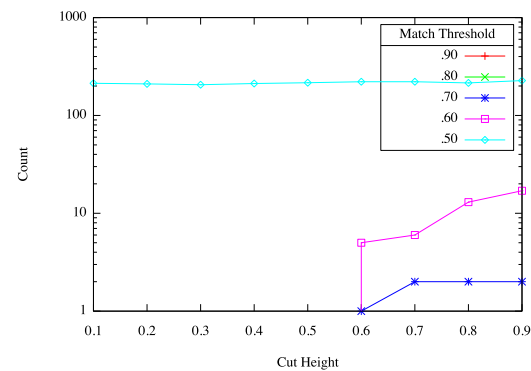


Figure 5: Effect of the dendrogram cut height (FPs).

by the clustering algorithm) needs to be cut to partition the HTTP requests into request clusters.

To select the dendrogram *cut height*, we proceeded as follows. We considered one day of malware traces collected from our malware intelligence feed (about 8,000 different malware traces). We then applied the clustering process to these traces, and produced different clustering results by cutting the dendrogram at different heights. For each of these different clustering results, we extracted the related set of CPTs, and we tested these CPTs over the next day of malware traces from our feed with varying matching thresholds. The obtained number of false positives, i.e., misclassified benign domains (measured using ATWL), and true positives, i.e., new correctly classified C&C domains (measured using CCBL), are summarized in Figure 5 and Figure 6, respectively. Notice that although in this phase we tested the CPTs over malware-generated network traces, we can still have false positives due to the fact that some malware query numerous benign domain names, along with C&C domains.

As Figures 5 and 6 show, per each fixed CPT matching threshold, varying the dendrogram cut height does not significantly change the false positives and true positives. In other words, the CPT matching results are not very sensitive to the specific value of the clustering parameter. We decided to finally set the value of the cut height to 0.38, which we use during all remaining experiments, because this provided good efficiency during the CPT generation process, while maintaining high CPT quality.

5.2.2 CPT Generation

To generate the CPTs used for the evaluation of ExecScent on live network traffic (Section 5.3), we initially used two weeks of malware traces collected from our malware intelligence feed. To label the *seed* of C&C HTTP requests in the malware traces, we used the CCBL black-list. We also use the list of parking IPs PKIP to

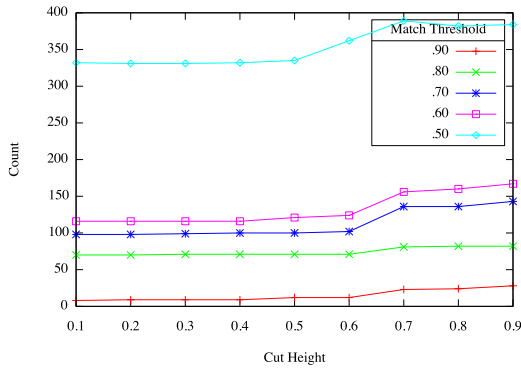


Figure 6: Effect of the dendrogram cut height (TPs).

prune CPTs related to parked C&C domains, as mentioned in Section 5.1.3. Once this initial set of CPTs was deployed, we continued to collect new malware traces from the feed, and updated the CPT set daily by adding new CPTs derived from the additional malware traces. More precisely, let D_1 be the day when the initial set of CPTs was first deployed in a live network, and let C_1 be this initial CPT set. C_1 is generated from the malware traces collected during a two-week period immediately before day D_1 . The CPTs set C_1 was then used to detect new C&C domains during the entire day D_1 . At the same time, during D_1 we generated additional CPTs from the malware traces collected on that day, and added them to set C_1 . Therefore, at the end of day D_1 we had an expanded set C_2 of CPTs, which we deployed on day D_2 , and so on. At the end of the deployment period we had just over 4,000 distinct CPTs.

To adapt the CPTs to the traffic of each deployment network (see Section 4.5), we proceeded in a similar way. We built a background traffic profile based on all HTTP traffic observed at each deployment network during the two days immediately before day D_1 , and used this profile to adapt the initial set of CPTs C_1 . Then, every day we updated the traffic profile statistics based on the new live traffic observed on that day, and used this information to further adapt all the CPTs. Notice that the set of CPTs deployed to different networks are different, in that they adapt differently to each deployment network (using that network's background traffic profile).

5.3 Live Network Deployment Results

To evaluate ExecScent, we deployed it in three different large networks, UNETA, UNETB, and FNET, for a period of two weeks. We generated the set of adaptive CPTs as explained above (Section 5.2.2), using a total of four weeks of malware-generated network traces (two weeks before deployment, plus daily updates during the

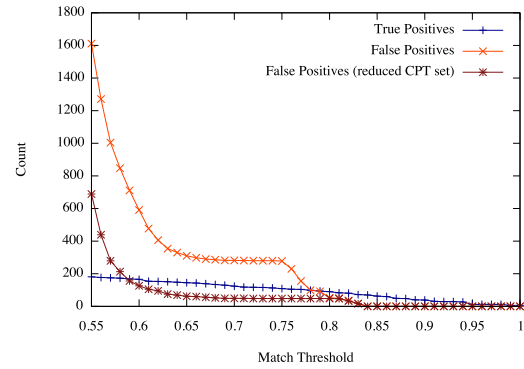


Figure 7: CPT detection results for varying detection thresholds.

two-week deployment period). The CPT matching engine was deployed at the edge of each network.

The detection phase proceeded as follows. For each network, we logged all HTTP requests that matched any of the adapted CPTs with a matching score $S \geq 0.5$, along with information such as the destination IP address of the request, the related domain name, the source IP address of the host that generated the request, and the actual value of the score S . This allowed us to compute the trade-off between the number of true and false positives for varying values of the detection threshold θ . Specifically, let h be a request whose matching score S_h is above the detection threshold θ , and let d be the domain name related to h . Consequently, we label h as a C&C request, and classify d as a C&C domain. We then rely on the CCBL and ATWL lists and on manual analysis (with the help of professional threat analysis) to confirm whether the detection of d represents a true positive, i.e., if d is in fact a C&C domain, or a false positive, in which case d is not a C&C domain.

Figure 7 summarizes the overall number of true positives and false positives obtained during the two-week deployment period over the three different live networks, while Table 2 shows a breakdown of the results on the different networks for a set of representative detection thresholds. For example, in Table 2, consider UNETA with a detection threshold of 0.65. During the two-week deployment period, we detected a total of 66 C&C domains, of which 34 are new, previously unknown C&C domains that were not present in our commercial black-list, CCBL. The 66 C&C domains were related to 17 distinct malware families. Overall, we detected 105 infected hosts, 90 of which were new infections related to the 34 previously unknown C&C domains. This means that 90 ($\approx 86\%$) of the infected hosts could not be detected by simply relying on the CCBL black-list.

The CPTs generated 118 false positives, namely domain names that we misclassified as C&C domains. We

Table 2: Live network results over a two-week deployment period

Detection Threshold	UNETA				UNETB				FNET			
	.62	.65	.73	.84	.62	.65	.73	.84	.62	.65	.73	.84
<i>All C&C Domains</i>	68	66	46	25	36	32	24	10	2	2	2	1
<i>New C&C Domains</i>	35	34	26	13	21	18	15	4	2	2	2	1
<i>Distinct Malware Families</i>	17	17	14	8	14	12	10	4	1	1	1	1
<i>Number of Infected Hosts</i>	114	105	98	37	185	150	147	21	7	7	7	7
<i>Number of New Infected Hosts</i>	91	90	86	25	145	135	133	11	7	7	7	7
<i>FP Domains</i>	133	118	114	0	152	117	105	0	109	63	49	0
<i>FP Domains (reduced CPT set)</i>	25	13	10	0	40	26	22	0	30	23	16	0

noticed that most of these false positives were generated by only two CPTs (the same two CPTs generated most false positives in all networks). By subtracting the false positives due to these two “noisy” CPTs, we were left with only 13 false positives, as shown in the last row of Table 2. The false positives marked with “*reduced CPT set*” in Figure 7 are also related to results without these two CPTs. Overall, within the entire two-week test period ExecScent generated a quite manageable number of false positives, in that a professional threat analyst could analyze and filter out the false C&C domains in a matter of hours.

Notice that the low number (only two) of new C&C domains found in the FNET network was expected. In fact, FNET is a very sensitive financial institution, where many layers of network security mechanisms are already in use to prevent malware infections. However, our findings confirm that even well guarded networks remain vulnerable.

5.3.1 Pushdo Downloader

It is worth clarifying that all results reported in Figure 7 and Table 2 have been obtained after discounting the domains detected through a single CPT that was causing hundreds of misclassifications. Through a manual investigation, we easily found that ExecScent had correctly learned this CPT, which actually models the HTTP-based C&C communications of a PUSHDO downloader variant [28]. This particular variant purposely replicates its C&C requests, and sends them to a large number of *decoy* benign domain names. The malware does this to try to hide the true C&C domain in plain sight, among a large set of benign domains. However, while this makes it somewhat harder to find the true C&C among hundreds or even thousands of benign domains (which requires some manual analysis effort), it makes it very easy to identify the fact that the source hosts of these requests, which matched our PUSHDO CPT, are infected with that specific malware variant.

We further discuss the implications of similar types of *noisy* or *misleading* malware behaviors in Section 6.

5.3.2 UNETB Deployment Results

The results we obtained for the UNETB deployment have been obtained in a slightly different way, compared to UNETA and FNET. Because of the higher volume of traffic in UNETB our proof-of-concept implementation of the CPT match engine could not easily keep pace with the traffic. This was due especially to the fact that our match engine software was sharing hardware resources with other production software that have to be given a much higher priority. A few weeks after conducting the experiments reported here, we implemented an optimized version (written in C, rather than Python) that is almost 8x faster; thus, it can easily keep up with the traffic on UNETB.

To compensate for the performance problems of our prototype implementation, during the two-week deployment period we only considered the traffic for every other day. That is, we only matched the CPTs over about seven days of traffic in UNETB, effectively cutting in half the traffic volume processed by ExecScent.

5.4 “Static” and URL-Only Models

In this section we compare the results of ExecScent’s *adaptive templates*, to “static” (i.e., non-adaptive) templates, which only learn from malware-generated traces and do not take into account the traffic profile of the deployment network, and to URL-based C&C request models, which only use information extracted from URLs.

To obtain the “static” models, we simply took ExecScent’s CPTs and “turned off” the specificity parameters. In other words, we set the specificity scores in Equation 1 to zero (with the exception of σ_d , which is set to one), essentially turning the dynamic waits w_k into their static counterparts $\hat{w}_k$ (see Section 4.7). In the following, we refer to these static (non-adaptive) templates as “Specificity-Off” models.

To obtain the URL-based models, again we “turn-off” the specificity information, and also ignore all components of ExecScent’s CPT apart from URL-related components. Effectively, in Equation 1 we only use the similarity functions s_1 , s_{2a} , and s_{2b} defined in Section 4.7. We refer to these templates as “URL-Only” models.

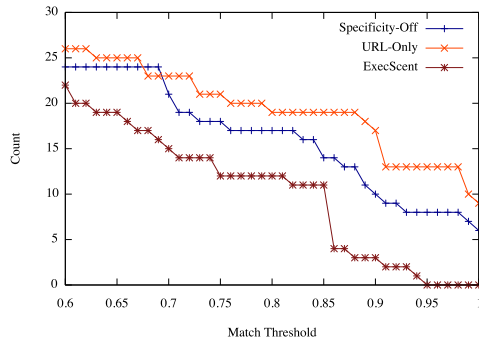


Figure 8: Comparing C&C Models - True Positives

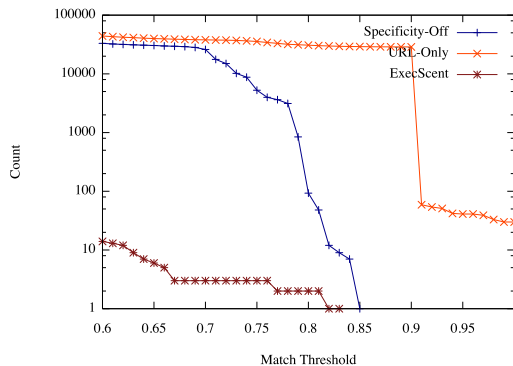


Figure 9: Comparing C&C Models - False Positives

To perform a comparison, we deployed the ExecScent CPTs and their related “Specificity-Off” and “URL-Only” models to UNETA, UNETB, and FNET for a period of 4 days. Figure 8 and 9 summarize the overall true and false positives, respectively, obtained by varying the detection threshold $\theta \in [0.6, 1]$. As can be seen from the figures, ExecScent’s adaptive templates outperform the two alternative models, for detection thresholds $\theta < 0.85$. Unless we are willing to sacrifice a large fraction of all true positives, compared to the numbers obtained at $\theta = 0.6$, the “Specificity-Off” and “URL-Only” models will generate a very large, likely unsustainable, number of false positives (notice the log scale on the y axes of Figure 9).

5.5 Deployment in ISP Networks

We were also able to evaluate the results of ExecScent over six large ISP networks serving several million hosts. We proceeded as follows: given 65 new C&C domains discovered by ExecScent during the live network deployment described in Section 5.3, we deployed the domains to the six ISPs for an entire week, during which we monitored all DNS traffic. Each day, we counted the number

of distinct source IP addresses that queried any of the 65 C&C domains. We found a maximum of 25,584 of distinct source IPs that in any given day queried these C&C domains. In other words, the new C&C domains discovered by ExecScent allowed us to identify 25,584 new potential malware infections across the six ISP networks.

6 Limitations

An attacker who gains knowledge of how ExecScent works may try to avoid detection by mutating her botnet’s C&C protocol every time the C&C server is relocated to a new domain. One possible approach would be to implement a new protocol that can be deployed on all the clients (i.e., malware agents) and servers (i.e., malware controllers) before switching to the new domain. However, this would substantially increase the complexity of managing the botnet and hurt its *agility*. Furthermore, for moderate to large botnets the updates would take time to deploy and a mistake in the update procedure could result in losing parts of or the entire botnet.

Another evasion approach may consist in injecting noise into the C&C protocol to make it appear “different”. For example, an attacker may randomly generate the C&C URL path or name-value pairs in the query-string, when making a request. However, if a malware agent needs to convey enough information to (loosely) authenticate itself to the C&C server, then at least one request component must have some form of “structured” data. Since ExecScent measures similarity by protocol structure and gives more weight to the shared unique components, it is non-trivial for an attacker to avoid detection on all deployment networks. In fact, several malware families we detect during our evaluation of ExecScent use such types of techniques to try to avoid detection via regular expressions.

An attacker may also try to “mislead” the detector by injecting noise into the domain name matches. For instance, an attacker may send requests to many decoy benign domains using the same malware C&C requests sent to the true C&C server. This is the approach used by the PUSHDO malware variant we discovered during our evaluation. This type of noisy malware is actually easy to identify, because of the number of unique destination domains contacted by a single host that match one particular CPT within a short period of time. Thus, detecting the infected hosts is easy. However, this makes it somewhat more difficult to determine the true C&C domains among all other domains. In this case, a threat analyst must review the domains, before they can be added to a blacklist; but at the same time, a security administrator can be immediately alerted regarding the infected hosts, thus enabling a prompt remediation.

Blending into the background traffic is another technique that may be used to avoid detection. For example, an attacker may choose “common” data types and values for their C&C protocol components. For some components such as the URL path it may be easy to select a popular value (e.g., “index.html”). However for many of the components, the “commonality” is relative to the deployment network’s traffic profile. Therefore, an attacker would need to customize the protocol based on the infected machine’s network. This may be difficult to do, because most network hosts have limited or no visibility into the traffic produced by other hosts in the same network. Therefore, although a C&C protocol may carry some “common” components, ExecScent’s adaptive CPTs may still be able to use those components that are *specific* (i.e., non-popular) in the deployment network to detect the C&C requests.

Finally, ExecScent’s CPTs depends on the malware traces and labeled C&C requests from which they are derived. Thus, ExecScent requires at least one or a few malware samples from a malware family, before its C&C protocol can be modeled and detected. In this case, though, malware code reuse plays to our advantage. A few samples of a malware family whose code has been reused elsewhere (because it was sold or leaked) will in fact facilitate the detection of future malware strains. Note that ExecScent in principle requires only a single sample to generate a CPT, thanks in particular to the request generalization process (Section 4.2). That being said, the quality of a CPT can be significantly improved when more than one sample sharing the same C&C protocol are available.

7 Related Work

Malware Clustering and Signature Generation: Grouping malware based on features extracted from HTTP requests has been studied for example in [7, 22, 23, 25]. Specifically, Perdisci et al. [22, 23] proposed a system for clustering malware samples that request similar sets of URLs. In addition, token-subsequences are extracted from the URLs, and used to detect infected hosts on live networks. In [7], information about HTTP request methods and URL parameters are used to cluster similar malware samples. The authors describe their clustering technique as a manual process and mention replacing it with an automated system in the future.

A recently proposed system FRIMA [25] clusters malware samples into families based on protocol features (e.g., same URL path) and for each family creates a set of network signatures. The network signatures are token-sets created from byte strings that are common to a large percentage of the network traffic within a clus-

ter. To reduce false positives, network signatures are pruned by removing the ones that match any communication in the authors’ benign traffic pool. Automated network signature generation has also been studied for detecting worms [19, 21, 27]. The generated signatures typically consist of fixed strings or token subsequences that can be deployed in an intrusion detection system. AutoRE [30] extends the automated signature generation process to produce regular expressions that can be used to match URLs in emails for the purpose of detecting spam emails and group them into spam campaigns.

Our work focuses on automatic *template* generation for detecting C&C communications and attributing them to a known malware family. In particular, our main focus is not on clustering malware samples per se. Rather, we apply clustering techniques mainly as an optimization step to generate high quality control protocol templates. Furthermore, we do not limit ourselves to only considering URLs or to extracting sets of common tokens. More importantly, our C&C templates are *adaptive*, in that they learn from the traffic of the network where they are to be deployed, thus self-tuning and automatically yielding a better trade-off between true and false positives.

Botnet Detection and C&C Identification: A number of studies have addressed the problem of detecting botnet traffic, for example [15, 16, 29]. BotSniffer [16] and BotMiner [15] are anomaly-based botnet detection systems that look for similar network behavior across hosts. The idea is that hosts infected with the same bot malware have common C&C communication patterns. Furthermore, BotMiner [15] leverages the fact that bots respond to commands in a coordinated way, producing similar malicious network activities. This type of systems require multiple infected hosts on the same monitored network for detection. In addition, being anomaly-based, they are not capable of attributing the infections to a specific malware family, and tend to suffer from relatively high false positive rates.

Our work is different, because ExecScent can detect botnets’ C&C even when only one bot is present in the monitored network. Furthermore, unlike previous work, ExecScent uses a *hybrid* detection approach, learning from both known C&C communications and the deployment network’s traffic to generate *adaptive* templates that can detect new C&C domains with high true positives and low false positives.

Wurzinger et al. [29] propose to isolate C&C traffic from mixed malicious and legitimated traffic generated by executing malware samples in a controlled environment. They propose to first identify malicious network activities (e.g., scanning, spamming, etc.), and then analyze the network traffic going back in time until a network flow is found that is likely to represent the command sent to the malware that caused the previously

identified malicious activities to be initiated. However, finding commands in malware network traces is not always possible. In fact, most datasets of malware network traces are obtained by running thousands of malware samples, with only a few minutes of execution time allocated to each sample. Therefore, the chances of witnessing a valid command being sent to a sample within such a small amount of time is intuitively small. On the other hand, malware samples typically attempt to contact the C&C server as soon as they run, even though no command to perform malicious activities may be issued at first contact. For this reason, ExecScent does not focus on identifying malicious network activities performed by the malware, and the related commands. Rather, ExecScent leverages any type of (HTTP-based) communication with a C&C server to learn control protocol templates that can be later used to identify new C&C communications and related C&C domains, even when malicious activities are not directly observable.

Jackstraws [17], executes malware in an instrumented sandbox [13] to generate behavior graphs of the system calls related to network communications. These system-level behavior graphs are then compared to C&C graph templates to find new C&C communications. ExecScent is different because it relies only on network information, and does not require malware to be executed in an instrumented sandbox (e.g., it can use traces collected from “bare metal” execution or live networks) to learn the templates. Furthermore, unlike Jackstraws [17], ExecScent learns *adaptive* templates, which allow us to identify new C&C domains in live networks.

Malicious Domains: Recently, a number of approaches for identifying malicious domains by monitoring DNS traffic have been proposed [2–4, 6]. These systems classify domains as malicious or benign, but do not attribute them to a specific malware family. Also, [2, 6] are mainly domain reputation system, and may assign a low reputation score to generic malicious domains, not only C&C domains, without providing any explicit distinction. On the other hand, [4] focuses only on malware that use pseudo-random domain generation algorithms. Kopis [3] is the only system that focuses explicitly on generic malware domains, but it requires the ability to monitor DNS traffic at the upper DNS hierarchy, which is difficult to obtain.

Unlike the DNS-based systems mentioned above, ExecScent focuses on detecting new C&C domains in live enterprise networks by inspecting HTTP(S) traffic, and using adaptive C&C protocol templates.

8 Conclusion

We presented ExecScent, a novel system that can discover new C&C domain names in *live* enterprise network traffic. ExecScent learns *adaptive* control protocol templates (CPTs) from both examples of known C&C communications and the “background traffic” of the network where the templates are to be deployed, yielding a better trade-off between true and false positives for a given network environment.

We deployed a prototype version of ExecScent in three large networks for a period of two weeks, discovering many new C&C domains and hundreds of new infected machines, compared to using a large up-to-date commercial C&C domain blacklist. We also compared ExecScent’s adaptive templates to “static” (non-adaptive) C&C traffic models. Our results show that ExecScent outperforms models that do not take the deployment network’s traffic into account. Furthermore, we deployed the new C&C domains we discovered using ExecScent to six large ISP networks, finding over 25,000 new malware-infected machines.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This material is based in part upon work supported by the National Science Foundation under Grant No. CNS-1149051. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

- [1] ANTONAKAKIS, M., DEMAR, J., STEVENS, K., AND DAGON, D. Unveiling the network criminal infrastructure of tdss/tcl4. https://www.damballa.com/downloads/r_pubs/Damballa_tdss_tcl4_case_study_public.pdf.
- [2] ANTONAKAKIS, M., PERDISCI, R., DAGON, D., LEE, W., AND FEAMSTER, N. Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX conference on Security* (Berkeley, CA, USA, 2010), USENIX Security’10, USENIX Association, pp. 18–18.
- [3] ANTONAKAKIS, M., PERDISCI, R., LEE, W., VASILOGLOU, II, N., AND DAGON, D. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX conference on Security* (Berkeley, CA, USA, 2011), SEC’11, USENIX Association, pp. 27–27.
- [4] ANTONAKAKIS, M., PERDISCI, R., NADJI, Y., VASILOGLOU, N., ABU-NIMEH, S., LEE, W., AND DAGON, D. From throw-away traffic to bots: detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX conference on Security symposium* (Berkeley, CA, USA, 2012), Security’12, USENIX Association, pp. 24–24.
- [5] BERNERS-LEE, T., FIELDING, R., AND MASINTER, L. RFC3986 - Uniform Resource Identifier (URI): Generic Syntax, 2005.

- [6] BILGE, L., KIRDA, E., KRUEGEL, C., AND BALDUZZI, M. Exposure: Finding malicious domains using passive dns analysis. In *NDSS* (2011), The Internet Society.
- [7] 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.
- [8] CHANG, C.-C., AND LIN, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2 (2011), 27:1–27:27. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [9] CRISTIANINI, N., AND SHAW-TAYLOR, J. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.
- [10] DANCHEV, D. Leaked DIY malware generating tool spotted in the wild, 2013. <http://blog.webroot.com/2013/01/18/leaked-diy-malware-generating-tool-spotted-in-the-wild/>.
- [11] DE LA HIGUERA, C., AND CASACUBERTA, F. Topology of strings: Median string is np-complete. *Theoretical computer science* 230, 1 (2000), 39–48.
- [12] EDMONDS, R. ISC Passive DNS Architecture, 2012. <https://kb.isc.org/getAttach/30/AA-00654/passive-dns-architecture.pdf>.
- [13] EGELE, M., SCHOLTE, T., KIRDA, E., AND KRUEGEL, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.* 44, 2 (Mar. 2008), 6:1–6:42.
- [14] FISHER, D. Zeus source code leaked. http://threatpost.com/en_us/blogs/zeus-source-code-leaked-051011.
- [15] GU, G., PERDISCI, R., ZHANG, J., AND LEE, W. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium* (Berkeley, CA, USA, 2008), SS'08, USENIX Association, pp. 139–154.
- [16] GU, G., ZHANG, J., AND LEE, W. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)* (February 2008).
- [17] JACOB, G., HUND, R., KRUEGEL, C., AND HOLZ, T. Jackstraws: picking command and control connections from bot traffic. In *Proceedings of the 20th USENIX conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 29–29.
- [18] JANG, J., BRUMLEY, D., AND VENKATARAMAN, S. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security* (New York, NY, USA, 2011), CCS '11, ACM, pp. 309–320.
- [19] KIM, H.-A., AND KARP, B. Autograph: toward automated, distributed worm signature detection. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 19–19.
- [20] KOLTER, J. Z., AND MALOOF, M. A. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.* 7 (Dec. 2006), 2721–2744.
- [21] NEWSOME, J., KARP, B., AND SONG, D. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2005), SP '05, IEEE Computer Society, pp. 226–241.
- [22] PERDISCI, R., ARIU, D., AND GIACINTO, G. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks* 57, 2 (2013), 487 – 500. Botnet Activity: Analysis, Detection and Shutdown.
- [23] PERDISCI, R., LEE, W., AND FEAMSTER, N. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 26–26.
- [24] PLATT, J. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers* 10, 3 (1999), 61–74.
- [25] RAFIQUE, M. Z., AND CABALLERO, J. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Proceedings of the 16th international conference on Research in Attacks, Intrusions, and Defenses* (2013), RAID'13, Springer-Verlag. To be published. Research conducted concurrently and independently of ExecScent.
- [26] SANTORELLI, S. Developing botnets - an analysis of recent activity, 2010. <http://www.team-cymru.com/ReadingRoom/Whitepapers/2010/developing-botnets.pdf>.
- [27] SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. Automated worm fingerprinting. In *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 4–4.
- [28] STONE-GROSS, B. Pushdo downloader variant generating fake HTTP requests, 2012. http://www.secureworks.com/cyber-threat-intelligence/threats/Pushdo_Downloader_Variant_Generating_Fake_HTTP_Requests/.
- [29] WURZINGER, P., BILGE, L., HOLZ, T., GOEBEL, J., KRUEGEL, C., AND KIRDA, E. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on Research in computer security* (Berlin, Heidelberg, 2009), ESORICS'09, Springer-Verlag, pp. 232–249.
- [30] XIE, Y., YU, F., ACHAN, K., PANIGRAHY, R., HULTEN, G., AND OSIPKOV, I. Spamming botnets: signatures and characteristics. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 171–182.
- [31] ZHOU, Y., AND JIANG, X. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE, pp. 95–109.