MARMITE: Spreading Malicious File Reputation Through Download Graphs

Gianluca Stringhini[♦], Yun Shen[♣], Yufei Han[♣], and Xiangliang Zhang[♠]
[♦]University College London, [♣]Symantec Research Labs, [♠]King Abdullah University of Science and Technology
g.stringhini@ucl.ac.uk, {yun_shen,yufei_han}@symantec.com, xiangliang.zhang@kaust.edu.sa

ABSTRACT

Effective malware detection approaches need not only high accuracy, but also need to be robust to changes in the modus operandi of criminals. In this paper, we propose MARMITE, a feature-agnostic system that aims at propagating known malicious reputation of certain files to unknown ones with the goal of detecting malware. MARMITE does this by looking at a graph that encapsulates a comprehensive view of how files are downloaded (by which hosts and from which servers) on a global scale. The reputation of files is then propagated across the graph using semi-supervised label propagation with Bayesian confidence. We show that MARMITE is able to reach high accuracy (0.94 G-mean on average) over a 10-day dataset of 200 million download events. We also demonstrate that MARMITE's detection capabilities do not significantly degrade over time, by testing our system on a 30-day dataset of 660 million download events collected six months after the system was tuned and validated. MARMITE still maintains a similar accuracy after this period of time.

1 INTRODUCTION

The malware ecosystem is constantly evolving, with cybercriminals both devising new ways to monetize their malicious software (e.g., ransomware [14]) and devising more efficient techniques to deliver malicious payloads to victim computers (e.g., exploit kits [8] or pay-per-install services [5], as well as developing techniques that make their operations stealthier and more resilient — e.g., Fast Flux [9] or Domain Generation Algorithms (DGA) [3].

Traditionally, malware detection is based on static [18, 31] or dynamic [6, 15] analysis. Such techniques, however are prone to evasion [12] and require a considerable amount of computational resources to carry out their inspections. More recently, the security research community has proposed approaches that are content-agnostic, performing detection without looking at the malware sample itself. Some of these techniques detect malware by looking at the characteristics of malware delivery infrastructures [5, 27, 30, 35]. Others identify topological relations among hosts and IP addresses in the hosting infrastructure used by cybercriminals [11, 20, 41], crawl potential malicious hosting sites in a proactive fashion [10, 32], analyze file co-occurrence relationships on infected computers [23, 36], or get insights from downloader-payload relationships of files on a host [19]. The problem with most of these content-agnostic approaches is that they use features that are based on common characteristics of malware delivery infrastructures, but these characteristics can change over time due to normal evolution or attempts to evade detection. For example, if certain characteristics relating to network topology or payload dropping change significantly, the classifiers must be retrained to maintain acceptable detection rates.

More importantly, the security community is now able to collect a very large amount of information at an unprecedented scale, e.g., amassing hundreds of millions of malware samples distributed globally on a daily basis. But how to efficiently identify malware from such large datasets remains a challenge. To address this Big Data challenge, we propose a solution that is principled, in the sense that it does not rely on a specific modus operandi of malware operators or on specific features of malware delivery networks. Instead of trying to understand whether a network delivery infrastructure is malicious, or identify files as malware from their characteristics or local view of file downloads, we leverage the known reputation of a small number of malicious and benign files, and *propagate this information* to other files that shared some part of the delivery infrastructure with them, with the goal of flagging them as benign or malicious. Our system, called MARMITE, first builds a global graph of file delivery, which we call *download graph*. Such a graph embodies a comprehensive view of how files are downloaded (by which hosts and from which servers and which files they drop) on a global scale. MARMITE is agnostic to the type of protocol used to host the files, to the type of the files themselves, to the specific techniques used by malware operators to avoid detection (such as Fast-Flux and DGA) and is therefore generic and resilient to evasion. In the next step, MARMITE performs semi-supervised label propagation with Bayesian confidence to propagate the reputation of known malicious files to unknown ones, allowing us to significantly and efficiently grow our knowledge of malware samples.

A key advantage of MARMITE is that it requires limited ground truth to operate, and it can grow the initial knowledge of malware significantly with a guaranteed linear computational complexity, which is a desirable characteristic in dealing with extremely large-scale datasets. We validate MARMITE on a dataset of 200 million download events collected by Symantec over a period of ten days. We show that our system is able to reach high accuracy (0.94 G-mean on average) and grow our knowledge of malicious samples up to 11 times, requiring a limited number of malware ground truth samples for seeding. An additional desirable property for a malware detection system is to keep their accuracy for long periods of time. This is a particularly important requirement, given the pace at which malware operations evolve. To test whether MARMITE is

ACSAC 2017, San Juan, PR, USA

978-1-4503-5345-8/17/12…$15.00
DOI: 10.1145/3134600.3134604

able to retain similar accuracy over time, we test our system on a 30-day dataset of 660 million download events that was collected by Symantec six months after the system was tuned. We show that Marmite is still effective even after this long time, without any need for re-tuning.

Our analysis highlights idiosyncrasies of malware delivery on the Internet that generate limited but systematic false positives when running label propagation approaches such as Marmite. For instance, it is common to have malware delivered through legitimate Content Delivery Networks (CDNs), and this can cause benign files to be tainted and mistakenly considered as malicious. Similarly, we observe that malicious files, and in particular potentially unwanted programs (PUPs [17]), have the tendency to download legitimate libraries (DLLs) as additional components. The bad reputation of the PUPs can then be propagated to these other files by Marmite. We provide detailed examples of these phenomena, and we propose a simple whitelist approach to reduce Marmite's false detections. We show that these systematic false positives do not change significantly over time, and a whitelist compiled six month before deployment is effective in significantly reducing the false detection rate of Marmite.

In summary, this paper makes the following contributions:

- We propose Marmite, a system based on semi-supervised Bayesian label propagation to propagate the reputation of known files across a download graph that depicts file delivery networks (both legitimate and malicious). The model is designed to be scalable and efficient.
- We validate Marmite on a dataset of 200 million downloads collected in the wild. We show that Marmite does not need carefully crafted seeds to catch malicious files, and a limited set of known malicious files is enough to seed a system with limited false positives.
- We show that Marmite does not require frequent retuning by testing it on a dataset of 660 million downloads collected six months after the system was tuned. We demonstrate that a simple whitelisting of files is enough to dramatically reduce the false positives reported by Marmite.

2 BACKGROUND AND MOTIVATION

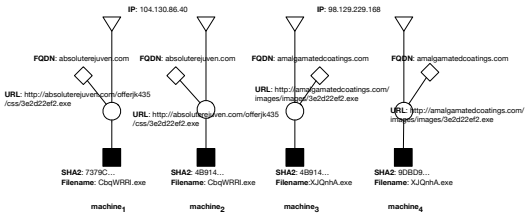

Figure 1: Local view of file distribution network via endpoints. Each graph is built from the local view from one of the end user machines.

To illustrate the complexity of keeping track of malware samples across different distribution infrastructures, consider the real world example in Figure 1. Each graph is built by looking at the

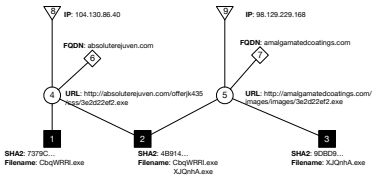

Figure 2: Global view of file distribution network. This graph includes the information that the same malware sample has been delivered by two separate servers.

download activity of a single end user's computer, similarly to what was done in [19]. There are two servers delivering malware, each with a distinct IP address as well as a separate domain (`absoluterejuven.com` and `amalgamatedcoatings.com`). Each malicious server drops two different malicious files over its lifetime, to a different victim computer each. In particular, the malicious file identified by the SHA2 value `4B914...` is dropped by both servers. One solution to flag these files as malware is to blacklist the malicious hosts dropping them, and consequently consider as malware any executable dropped by those hosts. A problem with this technique is that blacklists typically have coverage problems [26, 29], and if only one of the two hosts ended up in the blacklist then there would be a malware sample that would escape detection (either `7379C...` or `9DBD9...`). Another option is to blacklist files as they are observed, for example by running them in a sandbox and flagging anomalous behavior [39]. The problem with this type of techniques is that malware authors are actively trying to evade sandbox systems, and for this reason these approaches have limited coverage too. If a sandbox system was able to detect only the malware sample identified by `4B914...` as malicious, then the other two would escape detection.

To mitigate the aforementioned limitations, previous work focused on studying file dropping relationships after the malicious files are downloaded on victim computers [13, 19]. These approaches build local file dropping graphs extracted from endpoints (similar to Figure 1), extract features from these local graphs, and leverage machine learning algorithms to detect malicious droppers. While these approaches are effective in detecting droppers (i.e., malware samples that download additional components, for example as part of pay-per-install schemes), they do not provide a global view of the malware delivery ecosystem, and therefore their detection is also limited. For example, it is not guaranteed that any of the two aforementioned systems would be able to flag the files `7379C...` on $\texttt{machine}_1$ and `4B914...` on $\texttt{machine}_2$ as malware. This could happen because the localized view leveraged by these systems requires local graphs to be complex to guarantee an effective detection (e.g., to be part of a dropper operation in [19] or to have multiple infections on the same machine in [13]), or because the features relating to malware delivery infrastructures (e.g., the number of unique domains [19]) have changed since the time the detection systems were trained, and the detection efficiency of the systems decreased significantly since then.

As we said, in this paper we propose to approach the problem of detecting malware in a principled way, by looking at the end-to-end file distribution from a global perspective instead of focusing

on local views or relying on specific features of malware delivery networks (as previous systems did [11, 27]). In fact, the only requirement for Marmite is for malicious hosts to drop a multitude of malicious files, possibly dropped by other servers too. As these requirements are needed by the malware delivery process to be effective, they are not likely to change. Figure 2 shows an example of the download graph built by Marmite, obtained by merging the localized information from Figure 1. As it can be seen, this graph includes the information that the malicious file identified by `4B914...` is downloaded by two separate servers. Thanks to this global view, we can leverage structural information to detect malware when we only know that some of the files are malicious — as we will see in Section 3, we do this by probabilistically propagating labels across the download graph. For example, if node 1 in Figure 2 was initially flagged as malicious, node 2 and node 3 could be flagged as well since the boundary between hosts and machines no longer exists in this representation, and the evidence of maliciousness could be propagated along the graph. Ideally, a very limited initial knowledge of malicious samples could be enough to have this reputation propagated across the graph and to identify a large number of unknown malicious files.

Problem statement. Despite of its straightforwardness, the global download graph and the label propagation that we propose bring along several interesting issues regarding their applicability to the real world, especially in the Big Data context. First of all, legitimate content delivery network infrastructures (e.g., `Amazon Web Services`, `Akamai`) have been substantially used to host large amounts of malware alongside benign software, and therefore both benign software and malware are served through them [38]. At the same time, vulnerable websites (typically serving benign files) could also be compromised by cybercriminals to host malware. Moreover, a considerable amount of files could not be analyzed in a timely manner and thus have unknown reputation at the time of check (i.e., we could not confirm that a file is benign or malicious). Inevitably this global download graph is a mixture of benign files, malware, PUPs, and files with unknown status together with their hosting infrastructures. On that account, our *major goal* is to design a scalable graph inference model to reliably detect malicious files and validate this approach via a large-scale analysis on real-world data. This approach complements existing malware detection systems as it offers insights on how various files are hosted and distributed to end-hosts. Our *second goal* is to develop an approach that can be effective over a long period of time, despite the quick changes that the malware delivery ecosystem understakes. Finally, our *third goal* is to gain insights into current malware delivery schemes, which can help our community develop better mitigation techniques.

3 METHODOLOGY

In this section, we start by explaining how Marmite builds download graphs. We then describe the label propagation with Bayesian confidence graph inference model used by Marmite, giving a real world example of how it works. Finally, we provide a theoretical analysis of its scalability in dealing with large scale datasets.

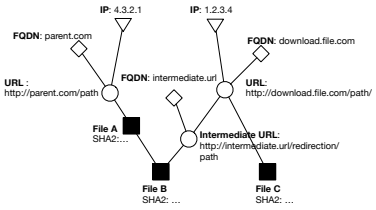

Figure 3: Legend to interpret graphs throughout the paper.

3.1 Building Download Graph

Marmite builds download graphs to perform its label propagation operation. We define the download graph

$$G = (V, E, A), \tag{1}$$

in which V is a set of heterogeneous nodes that represent the following entities: IP addresses, Fully-qualified domain names (FQDNs), URLs, and files. It is important to note that, although the nodes in this graph have a very different nature, some being related to hosts and some others to files, Marmite treats them in the same way, as the file reputation is propagated across them without making a distinction on their types. This keeps our model general and makes it independent to specific characteristics of malware delivery networks. E is a set of edges that represent the relationships among these nodes. As we will explain later, we consider nodes as connected if they appeared in the same download event. A is the symmetric adjacent matrix of the graph G: $A_{i,j} = 1$ if x_i and x_j are linked, otherwise $A_{i,j} = 0$. An example of download graph is shown in Figure 3. As it can be seen, the model captures both file dropping relationships and the file distribution network as discussed in Section 2.

To build the download graph, Marmite takes as input *download events*. We define a download event as a tuple

$$\mathbf{d} =< I, D, U_f, F_f, U_p, F_p, U_{i1}, ..., U_{ik} >, \tag{2}$$

where I is the IP address that the file was downloaded from, D is its FQDN, U_f is the full URL of the download (after removing the URL parameters), while F_f is the file identified by its SHA2. We then have information on the file that initiated the download event. This file can be a malware sample dropping other malware, as part of a pay-per-install scheme [5] or a legitimate program used to download files, such as a web browser. As we will explain in Section 4, we remove information on popular web browsers and file archivers from our datasets, to avoid to have unrelated files connected in the download graph. F_p indicates the SHA2 of this parent file, while U_p indicates the URL that this file was downloaded from. Note that both these values are not necessarily present in a download event. In fact, the information where a parent file was downloaded might not be available. Finally, we include a series of URLs $U_{i1}, ...U_{ik}$, which are the URLs involved in the HTTP redirection chain that terminates with the final download URL U_f. As previous research showed [35] malware operators are commonly using redirection chains to make their infrastructures resilient and avoid detection. These URLs are not always present in a download event, because not all downloads take advantage of any redirection.

Marmite collects download events over an observation period (as we show in Section 5.4, we experiment with variable period lengths, from one hour to one day), and then builds the download graph associated to this observation period. To build the graph, the following steps are taken for each download event **d**:

- For each element d_e in the tuple **d**, we check if d_e already has a node in the graph, if it does not, we create a new node identified by the node's identifier and add it to the set of nodes V. We use the full IP address and the FQDN as identifiers for IP addresses and domain nodes respectively, the full URL without parameters for URL nodes, and the SHA2 hash for file nodes.
- If there is no edge existing yet between any two elements d_{e_1} and d_{e_2} in d, we create one and add it to the set of edges E.

Finally, we populate the $|V| \times |V|$ matrix A, setting $A_{i,j}$ as 1 if there is an edge between those two nodes, and to 0 otherwise. This matrix will be used by the label propagation algorithm described in the next section.

3.2 Label Propagation with Bayesian Confidence

In this section we outline the theoretical foundation of Marmite to detect malware reliably over the download graph. To describe the malware detection problems in formal terms, we introduce the notions used in this section first, then formally present the Bayesian label propagation algorithm employed by Marmite. Note that our proposed Bayesian label propagation is a semi-supervised algorithm and calculates posterior probability of a node's missing labels based on graph structure. It is different from belief propagation, which is a sum-product message passing algorithm and estimates marginal distribution of each variable through the factor graph based presentation of a given Bayesian network.

Notation. Let x_i represent an element d_e in **d** (see Section 3.1), and $(x_1, y_1)\ldots(x_l, y_l)$ be labeled data, denoted as (X_L, Y_L). We assume that all class labels are presented in Y_L. Let $(x_{l+1}, y_{l+1})\ldots(x_{l+u}, y_{l+u})$ be unlabeled data (X_U, Y_U), where Y_U denotes the underlying true class labels of the unlabeled data. In addition, $X = \{x_1, x_2, \ldots x_{l+u}\}$ is used to denote the combination of both labeled and unlabeled data.

Bayesian Label propagation. Label propagation [42], as a transductive semi-supervised learning algorithm, is designed to estimate Y_U from X and the given Y_L by propagating label information from labeled data samples to unlabeled ones. It has shown that graph-based propagation can be very effective at finding the best label distribution across nodes [42]. However, most nodes in a real world download graph tend to have few neighbors, and thus, a small amount of evidence. It is important for us to provide a confidence level to the inference results. In our work, the malware detection algorithm used by Marmite is designed by inheriting the spirit of label propagation and incorporating the technique of Bayesian inference [40]. We use the graph definition $G = (V, E, A)$ presented in Section 3.1 for download graphs, where V represents files, URLs, FQDN and IP nodes, the edge set E represents the relationships among these nodes, and A is the symmetric adjacent matrix of the graph G: $A_{i,j} = 1$ if x_i and x_j are linked, otherwise $A_{i,j} = 0$.

We assume the label (benign or malicious) of each node in the graph as a random categorical variable. The label is 1 or 0, corresponding to the malicious or benign class. What we target is to infer the probability of each node to be malicious as $P(y_i = 1|\theta) = \theta$, where θ is the parameter of the distribution. Changing the value of θ changes the likelihood of the corresponding nodes belonging to the malicious group. Following the neighborhood smoothness hypothesis of label propagation, we assume the graph nodes inside the same neighborhood share the same θ value. The binomial likelihood of labels inside the neighborhood is given as follows:

$$P(\tilde{N}_i|\theta) = \frac{n_i!}{n_i!(\tilde{N}_i - n_i)!}\theta^{n_i}(1-\theta)^{\tilde{N}_i - n_i}, \tag{3}$$

where $\tilde{N}_i$ denotes the all graph nodes in the neighborhood of node i. $n_i = \sum_{j \in \tilde{N}_i} y_j$. For computational convenience, we use the conjugate prior of the binomial distribution, a.k.a Dirichlet distribution as the prior probability over θ, $P(\theta) \propto \theta^{\alpha_1 - 1}(1-\theta)^{\alpha_2 - 1}$, where α_1 and α_2 are the parameters of Dirichlet prior. The posterior distribution of θ given the labels of the nodes inside the same neighborhood is formulated as:

$$P(\theta|\tilde{N}_i) \propto \theta^{n_i + \alpha_1 - 1}(1-\theta)^{\tilde{N}_i - n_i + \alpha_2 - 1}. \tag{4}$$

In a further step, we can write the posterior predictive distribution of y_i:

$$\begin{aligned} P(y_i = 1|\tilde{N}_i, \alpha_1) &= \int_\theta P(y_i = 1|\theta)P(\theta|\tilde{N}_i, \alpha_1)d\theta \\ &= \frac{n_i + \alpha_1}{\left|\tilde{N}_i\right| + 1}. \end{aligned} \tag{5}$$

To derive the formulation of the predictive distribution, we follow the theorem that the predictive distribution $P(y_k|\alpha_k, \tilde{N}_i) = E(\theta|\tilde{N}_i, \alpha_k)$ and we assume $\sum_{k=\{1,2\}} \alpha_k = 1$ and $\left|\tilde{N}_i\right|$ is the number of neighbors of the node i in the graph. By relaxing the definition of n_i and replacing the discrete labels of the neighboring nodes with continuous posterior probability of node labels $P(y_i = 1|\tilde{N}_i, \alpha_1)$, we can further derive a recursive estimation of the confidence of node labeling in the following equation:

$$f_i = \frac{\sum_{j=1}^{N} A_{i,j} f_j + \alpha_1}{\sum_{j=1}^{N} A_{i,j} + 1}. \tag{6}$$

where N is the number of nodes in the graph and A is the adjacency matrix. We use f_i to denote the posterior labeling probability $P(y_i = 1|\tilde{N}_i, \alpha_1)$. With a simple linear algebra calculation, the recursive calculation of labeling posterior probability for each unlabeled node in the download graph can be formulated in a matrix form:

$$F_U = (D_U + I)^{-1}(A_U F + \mathbf{1}\alpha_1) \tag{7}$$

Assuming we have u unlabeled nodes, $\mathbf{1}$ is u dimensional column vector with each entry as 1. F is a vector of f_i for all nodes including both labeled and unlabeled, and F_U is a vector of f_i for u unlabeled nodes. A_U is formed by aggregating the u rows of A corresponding to unlabeled nodes. I is a u-by-u identify matrix, and D_U is a u-by-u diagonal matrix, where the k-th element in diagonal is the sum of all values in the k-th row of A_U. An example of how Marmite infers

probabilistic node labels in a small real-world download graph is shown in Section 3.3.

3.3 Marmite: Inference Example

In this section, we demonstrate how Marmite infers probabilistic node labels in a small real-world download graph. The label f_i is a probabilistic value as defined in Eq (6) and $f_i \in [0, 1]$ (see Section 3.2 for details). In short, the closer f_i to 1, the more likely a file is malicious; the closer f_i to 0, the more likely a file is benign.

Initially file `4.exe` with SHA2 `96D26...` and file `svchos.exe` with SHA2 `DB000...` are identified as malicious files (respectively `Zusy` and a trojan). We label them accordingly using 1 as their probability of being malware. Another file `4.exe` with SHA2 `CB866...` is labeled with 0 (i.e., a known benign file). Marmite uses Eq (6) to recursively update each node's probabilistic label and the results are shown in Figure 4. As we can observe, file `AFFGSDGWIJGWEOG.exe` with SHA2 `18A2F...` is flagged as malicious with high probabilistic confidence score 0.969 as all its neighbors (i.e., `svchos.exe` in this case) are malicious. This follows the neighborhood smoothness hypothesis of label propagation (see Section 3.2). The other nodes' probabilistic node labels (between 0.6 and 0.7) were partially influenced by file `4.exe` with SHA2 `CB866...`, which is benign. For this example, we consider a node as malicious if $f_i > 0.5$. In the end, Marmite is able to flag as malware three previously-unknown samples. As we can see in Figure 4, the Bayesian label propagation algorithm enables Marmite to flag the rest of the files as malicious.

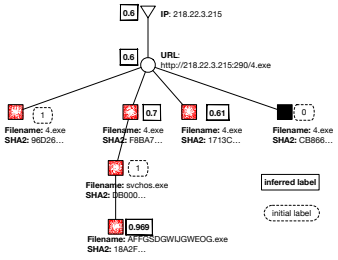

Figure 4: An example of how Marmite infers probabilistic node labels. Initially, two files are known to be malware and one is known to be benign. At the end of process, three additional files are correctly identified as malware.

3.4 Scalability

It is important to note that Eq (6) can be easily parallelized due to its recursive update nature. That is, for each individual node i, it outputs its current label confidence f_i to its neighbors during the *map* phase; at the *reduce* phase, it updates its labeling confidence f_i by using a weighted aggregation $\sum_{j=1}^{N} A_{i,j} f_j + \alpha$ then divided by $\sum_{j=1}^{N} A_{i,j} + 1$ (which is the sum of all edges incidental to the node i). In terms of time complexity, the proposed method iterates over each edge in the download graph a constant amount of times. Given the maximum number of iterations parameter is fixed as c, the graph has $|E|$ edges and $|V|$ node, the overall time complexity is $O((|E| + |V|) * c)$. Therefore, it is safe for us to conclude that Marmite has a *linear* time complexity.

4 DATASETS

To validate and test Marmite we use the *download activity data* from Symantec's data sharing platform. This data was collected from the users who opt in for Symantec's data sharing program to help improving its detection capabilities. To preserve the anonymity of the users, client identifiers were anonymized so that it is not possible to link back the collected data to the user that originated it.

The download activity data provides meta-information about all download activities on the user clients. From this data we extract the following information: the server side timestamp of the download event, the name, SHA2, and size (in bytes) of the downloaded file, the referrer URL, the landing page URL and IP address of the server the file was downloaded from, the full path at which the downloaded file was stored on the server, the SHA2 of the parent file (the file that initialized the file downloading activity) and the URL that this file was downloaded from. Note that SHA2 and URL of the parent file are not always present and we also remove parent SHA2s related to popular web browsers and file archivers from our dataset, to prevent unrelated files from being connected in the download graph — a similar approach was used in previous work that studied local delivery graphs [19]. In this paper, we remove 710 SHA2s that are relating to `Outlook`, `IExplore`, `Firefox`, `Chrome`, `WinZip`, `Filezilla`, `7Zip`, `TeamViewer` and `WinRAR`.

Symantec also employs extensive static and dynamic analysis systems to determine the maliciousness of a binary. We leverage these results and further enrich our dataset using its *binary reputation data* to include metadata about the reputation and prevalence of the downloaded files. This allows us to evaluate Marmite's performance, by looking at its capability of flagging known malware and benign software. On average we extracted 20 million download events generated by 1.5 million machines per day. We compiled two separate datasets from this download data. The first one, which we call $\mathbf{D}_1$, spans a period of 10 days in January 2016, and is composed of over 200 million download events. We use $\mathbf{D}_1$ to tune the parameters needed for Marmite's operation, and to validate our approach. We then compile another dataset, which we call $\mathbf{D}_2$. This dataset is composed of 660 million download events collected over the entire month of June 2016. We use this dataset to test Marmite, and to show that our system can still perform accurate detections without needing re-tuning six months after it was set up.

Data Limitations: It is important to note that the download activity data is collected passively. That is, download events are recorded only when their associated downloading request is initiated (this being generated from a user by clicking on a link or stealthily triggered by a drive-by download exploit or a file dropping event). Any downloads preemptively blocked by existing defenses (e.g., URL/IP blacklists) were not observed.

5 VALIDATION

In this section, we first describe how we collect the ground truth of malicious and benign files in Section 5.1. We then provide a detailed validation of Marmite over the $\mathbf{D}_1$ dataset (see Section 4) collected in January 2016. We identify the optimum values for several parameters used by Marmite. Once the optimal parameters are identified, we validate Marmite's effectiveness at detecting

malware by comparing its detections with the ground truth labels. We show that Marmite can effectively increase the knowledge of malware samples delivered over a download graph, requiring limited seeds of known malicious and benign files. In Section 6.1 we show that the system tuned and tested on the dataset $\mathbf{D}_1$ can still detect malware on a dataset collected six months later.

5.1 Ground Truth

Marmite requires labeled ground truth of malware and benign software for two reasons. The first one is that the approach described in Section 3.2 requires a number of seed files to propagate their reputation across the download graph. The second one is that ground truth is needed to check whether the detections performed by Marmite are accurate or not. We used three separate sources to collect ground truth: VirusTotal, the National Software Reference Library, and data collected by Symantec.

VirusTotal. VirusTotal [2] is a free online service that aggregates the scanning outputs of files and URLs from different antivirus engines, website scanners, and a number of file and URL characterization tools. We query VirusTotal for each file SHA2 to obtain its *first seen* timestamp, the number of AV products (and associated vendor names) that flagged the file as malicious, and the total number of AV products that scanned the file. We consider a file malicious if at least one of the top five AV vendors w.r.t. market share (in no particular order, Symantec, Avast, Microsoft, AVG, Avira) and a minimum of two other AVs detect it as malicious. A similar technique was used by previous work [24].

National Software Reference Library (NSRL). To identify known benign and reputable files we used NSRL's Reference Data Set (RDS) version 2.51 [1]. This dataset provides SHA1 to SHA256 hashes for known benign programs and it was useful for us to identify such files in our dataset.

Complementary Ground Truth. We obtained additional ground truth about malicious files from Symantec, which provided us with the data. This ground truth is based on Symantec's static and dynamic binary analysis platform.

In total our ground truth on dataset $\mathbf{D}_1$ consists of 833,705 malicious files and 1,896,782 benign files. The skewed distribution between benign and malicious files maps the real world observation that benign files are more prevalent in the wild.

5.2 Choose Optimal Parameters

Marmite requires three parameters to operate. These three parameters are the shape parameter of the Dirichlet distribution α, the benign file sampling ratio r_b, and the malware sampling ratio r_m. Different α values affect the inference accuracy (see Section 3.2), while r_b and r_m determine the training data size for benign and malicious files respectively. In this section, we explain how we selected the optimal value for these parameters for Marmite to operate.

Since α, r_b, r_m are not independent, but selecting one of them influences the values of the others, we did not choose them separately. We adopt the *grid search* approach [4] to find the optimal parameters. More precisely, we exhaustively search through a multiple combinations of values for these three parameters and evaluate Marmite by using each of them. We identify the optimal parameters that generate the maximum geometric mean (G-mean) score [25] (G-mean=$\sqrt{TP_{rate} \times (1 - FP_{rate})}$). Our dataset is imbalanced between benign and malicious files. This makes G-mean the ideal metric to evaluate the performance of Marmite, because it balances between the classification performance on the majority and the minority class.

Guided by this methodology, we form the parameter space by setting α between $(0, 1]$, r_b between $[0.4, 0.7]$ and r_m between $[0.05, 0.2]$, and build 64 combinations of these three parameters. The ranges of r_m and r_b are selected to reflect the real world scenario; there are plenty of benign files available while the number of malware samples is limited and we also need to leave some portion of samples out for test purpose.

We then use them to carry out the grid search. We run Marmite on a three day subset of the dataset $\mathbf{D}_1$ for each combination, and average the G-mean score over the three days to measure the performance of each combination. We perform stratified 10-fold cross validation [16] for each combination. Our results show that given $\alpha = 0.1$, $r_b = 0.6$ and $r_m = 0.1$, Marmite outputs the best average G-mean score, 0.953. On that account, we use these values for the rest of the paper. In the next section we provide detection results using these parameters over the remaining part of dataset $\mathbf{D}_1$. In Section 6.1 we demonstrate that these parameters are still effective for Marmite to detect malware six months later.

5.3 Time and Memory Performance

Following our theoretical analysis in Section 3.4, we evaluate our system runtime performance to answer key practical questions such as 1) *how long does* Marmite *take to construct a download graph*, 2) *how long does* Marmite *take to infer maliciousness of all unlabeled nodes in a download graph* and 3) *how long does* Marmite *take to compute the optimal parameters*. Since our typical use case is to identify malicious files on any given day, for the rest of the section, we use one day of data with 21,627,935 download events from $\mathbf{D}_1$ as our runtime evaluation baseline. All tests are performed in a server with a 2.67GHz Xeon CPU X5650 and 64GB memory running Ubuntu Linux 12.04 and Python 2.7.3. In this setup, Marmite takes 890.95 seconds to construct the download graph from one day of data with 15.98GB memory footprint. Once the graph is constructed, Marmite takes 84.498 seconds to finish the inference task (see Section 3.2). In other words, Marmite is able to perform its entire analysis for one day in about 16 minutes. This exemplifies Marmite's computational advantage as we theoretically proved in Section 3.4. In terms of how long it takes to tune the parameters required by Marmite to operate, essentially we need to repeat the aforementioned inference task 64 times (see Section 5.2) on a three day subset of the dataset $\mathbf{D}_1$. Marmite takes therefore 4.82 hours to find the optimal parameter combination. Note that we don't need to run this tuning task frequently as we demonstrate that Marmite is able to retain similar accuracy over time (see Section 6).

5.4 Labeling Performance

We validate the overall performance of Marmite on the labeled data (i.e., ground truth) over a 7 days subset of the $\mathbf{D}_1$ dataset — note that the remaining 3 days for this dataset were used to tune the optimal parameters. In order to assess Marmite's performance

Table 1: Performance of Marmite on labelled ground truth using seven days of data collected during January 2016 (AUC=0.96, G-Mean=0.944). The table reports averaged TPR/FPR results over the seven days.

TPR (FPR=0.5%)	TPR (FPR=1%)	TPR (FPR=2%)	TPR (FPR=3.5%)
0.690	0.786	0.866	0.923

from different perspectives, we also include the area under the ROC curve (AUC) score [7], in addition to the G-mean score that we already used in Section 5.2. Overall, Marmite achieves an average AUC of 0.960 (with a standard deviation of 0.01), and an average G-mean of 0.944 (with a standard deviation of 0.008). This shows that Marmite can offer high accuracy with stable performance over time since the standard deviation of both AUC and G-mean scores over the measurement period of 7 days are small. We further confirm this finding by testing Marmite on a dataset collected six months later in Section 6.1.

We then wanted to understand the true positive rate (TPR) and false positive rate (FPR) reported by Marmite on labeled data. These values are important, because they give us a feeling of how well Marmite would perform if ran in the wild. Table 1 reports the results of this experiment. We start by setting a fairly high false positive rate of 3.5%, and measure that in this setting Marmite has a TPR of 0.923 (which corresponds to a false negative rate of 7.7%) on average. By decreasing the false positive rate, the TPR decreases but remains high. With a FPR of 1%, Marmite reports a TPR of 0.786 (false negative rate of 21.4%) on average. By decreasing the false positive rate even further, to 0.5%, Marmite reports and average TPR of 0.690 (false negative rate of 31%). This result shows that Marmite could be set to have very low false positives and still be useful in practice, flagging a significant amount of malware. In addition, in Section 6.1 we show that whitelisting can be used to further reduce false positives. We also show that false negatives in the wild are lower than what reported by these validation tests, which were only performed on ground truth of known malicious and benign files. Marmite, regardless the size of the graph, can also maintain comparable accuracy in terms of both AUC and G-mean scores thanks to recursively propagated evidence across the whole graph. Details can be found in Section 5.5.

5.5 Modifying the Observation Interval

Our hypothesis is that Marmite, regardless the size of the graph, should maintain comparable accuracy in terms of both AUC and G-mean scores thanks to recursively propagated evidence across the whole graph. To check if this is the case, we build three download graphs from observation periods of one hour (00:00 - 01:00), two hours (00:00 - 02:00), and three hours (00:00 - 03:00), extracted from the data collected on a one day subset of the dataset $\mathbf{D}_1$. We use the same parameters as in the rest of the paper. We list the results of this experiment in Table 2 and compare the results to those obtained on the data extracted over an entire day. As we can see, even if the graphs for shorter time intervals are significantly smaller, both AUC and G-mean scores from all three experiments remain similar to those for the full day data: we have an AUC of 0.975 (G-mean 0.919) for the one-hour interval, an AUC of 0.977 (G-mean 0.923) for the two-hour interval, and an AUC of 0.978 (G-mean 0.957) for the three-hour interval. These results demonstrate that our approach can work well on different observation windows, and underpins Marmite's real-world practicability.

Table 2: Performance of Marmite using one hour, two hour, and three hour observation intervals on one day of data. As it can be seen, Marmite reports good results both in terms of AUC and G-Mean for all interval lengths.

Observation interval	#nodes	#edges	AUC	G-mean
00:00 - 01:00	147,721	196,297	0.975	0.919
00:00 - 02:00	261,686	359,602	0.977	0.925
00:00 - 03:00	355,787	498,745	0.978	0.934
00:00 - 23:59	2,256,984	3,364,215	0.972	0.957

6 EVALUATION

In the previous section we validated Marmite by testing it on ground truth data collected over a period of 10 days (dataset $\mathbf{D}_1$). In this section we evaluate it against dataset $\mathbf{D}_2$, which was collected six months later. We first show that Marmite is still effective in detecting malware without need for re-tuning after this period of time, significantly growing the amount of detected malware samples compared to the ones used for seeding. We also show that Marmite is able to detect malware before VirusTotal.

6.1 Malware Detection In the Wild

We ran Marmite on the entire 30-day dataset $\mathbf{D}_2$. On average, we seeded the system with 111,449 benign files and 7,657 malicious ones every day. We used four different modes of operation for Marmite, which means that for each of them we set the parameters that reported 3.5%, 2%, 1%, and 0.5% FPR in the ground truth validation from Section 5.4. The overall results for our experiments are reported in Table 3.

For each of the settings, we carefully vetted the results provided by Marmite. In particular, we first checked whether the detections performed were either confirmed by Symantec's internal systems or if the SHA2 hashes of the detected files appeared as malicious in VirusTotal. We considered these detections as true positives by Marmite. Note that we split detections between malware and PUP, based on Symantec's feedback. As we show in Section 7.4, it is often difficult to distinguish the two types of operations. As it can be seen in Table 3, the fraction of detections performed by Marmite is generally high. For the 3.5% setting we can confirm 94% of the detections as either malware or PUP. This number gradually increases as we make detection stricter, peaking at 98% for the settings that lead to a 0.5% FPR during the validation phase.

For the false positive analysis, we looked at files whose SHA2s are known as benign by Symantec. As it can be seen in Table 3, false positives decrease as we make Marmite stricter on which files it considers as malware. For the 3.5% case, Marmite reports a 5.8% false positive rate in the wild. This number decreases steadily as the system becomes stricter, up to arriving at 1.1% for the 0.5% setting. In general, we can observe that Marmite's results in the wild are slightly worse than they were by looking at labeled data only (as we did in Section 5.4). As we explained, however, we expect many of

Table 3: Summary of detection performance by MARMITE on our 30-day measurement data. "Setting" reports the FPR obtained when setting the same parameter values during the validation phase.

Setting	Tot. Prediction	Confirmed Malware	Confirmed PUP	FPs before Whitelisting	FPs after Whitelisting	FNs	Unknown
0.005	1,684,439	1,540,877	111,913	24,546 (1.1% FPR)	14,978 (0.6% FPR)	593,643 (26.2% FNR)	7,101
0.01	2,104,897	1,757,027	277,091	53,949 (2.3% FPR)	34,676 (1.5% FPR)	383,634 (15.6% FNR)	16,827
0.02	2,496,526	1,899,359	481,170	89,175 (3.8% FPR)	51,277 (2.2% FPR)	246,175 (9.1% FNR)	26,811
0.035	2,864,481	1,985,959	707,545	134,657 (5.8% FPR)	67,752 (3.0% FPR)	163,940 (5.5% FNR)	36,298

these false positives to be systematic and not change significantly over time, so that they can be easily removed by using a whitelist. In Section 7.1 we show a detailed example of these systematic false positives. To test this hypothesis, we compile a whitelist of known benign files from January 2016, and apply it to our results obtained six months later, in June. As Table 3 reports, the whitelist is able to reduce false positives significantly. For the 3.5% setting, the false positive rate of the filtered dataset is only 3%, while for the 0.5% setting it becomes 0.6%.

While false positives in the wild turn out to be slightly higher than in the validation phase, false negatives are lower, indicating that MARMITE is able to comparatively detect more malware than it was present in our ground truth. As it can be seen in Table 3, the false negative rate for the 3.5% setting is 5.5%, while it was 7.6% on the ground truth. Similarly, for the 0.5% case the FNR is 26.2%, while it was 31% during the validation. Finally, for a small number of files none of our sources could confirm whether these files were benign or malicious. We list them as "unknown" in Table 3.

Based on the results reported in this section, we can see that MARMITE is able to effectively detect malware, and does not require frequent retraining, as the results in the wild six months after the tuning of the systems are generally in line with the original validation results. Depending on how strict the operator wants to be in making detections, MARMITE can increase the original knowledge of malware from the seed files between 11 times (in the 3.5% case) and 6 times (in the 0.5% case).

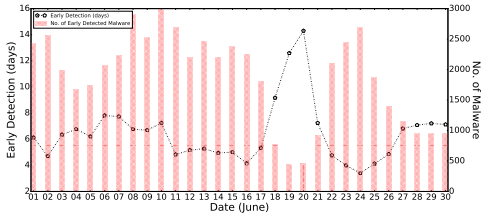

Figure 5: Number of malicious files detected by MARMITE in advance and average early detection time.

6.2 Early Detection of Unknown Malware

We estimate that if MARMITE was run in production it would have been able to detect these files as malicious *before* other antivirus programs. To understand how relevant this early detection would have been, we looked at the average days passed between when MARMITE flagged a file as malicious and when VirusTotal flagged it as such too. Note that we consider a malware sample as detected by VirusTotal if it is flagged by one of the top five AV products plus any other two [24]. For evaluation purpose in this section, we set FPR=3.5%. Figure 5 reports a summary of the number of malware samples detected by MARMITE that were unknown to VirusTotal at the moment, together with how much later (in days) these files appeared on VirusTotal as malicious. On average, MARMITE was able to detect 1,870 files as malware 6.46 days before VirusTotal on a daily basis.

7 LESSONS LEARNED AND CASE STUDIES

In this section, we provide some interesting case studies that we encountered while operating MARMITE. Note that we remove FQDN nodes from all the figures to make them easier to read.

7.1 Malicious programs download legitimate libraries.

In this section, we carry out a detailed case study on PUPs dropping both benign DLLs and further PUPs. This is a typical example of the false positives reported by MARMITE, and shows that many false positives are systematic. Figure 6 shows part of a download graph built by MARMITE, illustrating a typical case that PUPs drop both benign dlls and further PUPs. As we can see in the figure, file `AddonsUI.exe` with SHA2 `5CD12...` is identified as `PUP Bubbledock` and dropped two other PUPs, respectively file `OneSystemCare.exe` with SHA2 `B9BEE...` (identified as `PUP OneSystemCare`) and file `7d27aa...exe` with SHA2 `6307C...` (identified as `Adware ConvertAd`), and three benign DLLs, respectively file `WmiInspector.dll` with SHA2 `A9347...`, `nsdialogs.dll` with SHA2 `1DEC2...`, and `HttpRequest.dll` with SHA2 `97CE1...`. Even though these DLLs are legitimate (we speculate that they are dropped as part of the dependencies used by file `AddonsUI.exe`), these three DLLs are wrongly classified as malicious due to the overwhelming evidence surrounding them. In this paper, we showed that benign files marked as malicious by MARMITE do not change quickly over time, and can therefore be prevented by applying a static blacklist.

7.2 Legitimate content delivery networks are used to deliver malware.

Figure 7 illustrates a typical false positive case by MARMITE. IP address `23.77.202.16` serves four different URLs from very different domains, including an `Apple` domain. We inspected this IP address, and discovered that it belongs to `Akamai`, a well-established content delivery network serving between 15 and 30 percent of all web traffic. Both `http://files4.fastdownload6.com/dl-pure/...` and `http://download.cdn.sharelive.net/cdn/...` host a number

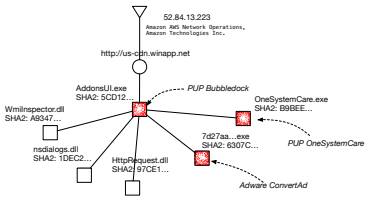

Figure 6: Example of a PUP dropping benign DLLs.

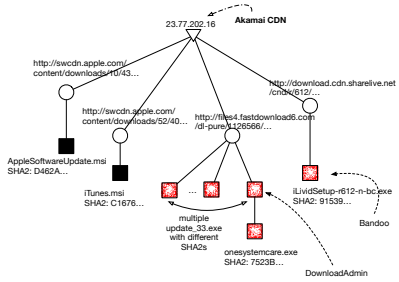

Figure 7: Case study in which a legitimate content delivery network is used to deliver both benign and malicious files.

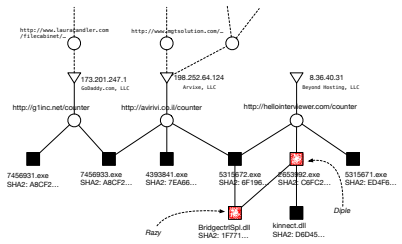

Figure 8: Malware case study before label propagation. At the beginning, only two files are labeled as malware.

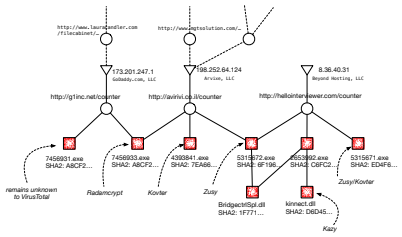

Figure 9: Malware case study after label propagation. At the end of the process, six additional samples are discovered.

of malicious files/PUPs (shown as red blocks in Figure 7). For example, `iLividSetup-r612-n-bc.exe` is identified as the `Bandoo` adware, and `update_33.exe` is identified as `PUP DownloadAdmin`. When running Marmite, the malicious reputation of these files gets propagated to `AppleSoftwareUpdate.msi` and `iTunes.msi`, which end up being wrongly detected as malware. The issue here is that the CDN is both serving benign and malicious content. In Section 8 we discuss how we could deal with this type of false positives to decrease the false positive rate of Marmite.

7.3 Malware operation.

In this section, we carry out a detailed malware distribution case study. Figure 8 shows part of a download graph built by Marmite on January 11, 2016. Nodes connected by dash lines are verified to be benign sites and nodes connected with solid lines are the focus of our discussion. There are three IP addresses that belong to three different hosting infrastructures, respectively `GoDaddy.com`, `Arvixe` and `Beyond Hosting`. Before running Marmite, files `BridgectrlSpl.dll` with SHA2 `1F771...` and `2653992.exe` with SHA2 `D6D45...` were known to belong to the `Razy` and `Diple` malware families respectively. All the other files had unknown reputations. After running Marmite, the rest of the files were identified as malicious (shown in Figure 9), and were accordingly confirmed by VirusTotal, either on that same day (`7456933.exe`) or several days later (`4393841.exe`, `5315672.exe`, `5315671.exe` and `kinnect.dll`). It is worth noting that file `7456931.exe` with SHA2 `A8CF2...` remains unknown to VirusTotal at the moment of writing.

The identified malware samples belong to four different families: (`Radamcrypt`, `Kovter`, `Zusy`, and `Kazy`). Note that We collect the tags used by different vendors, remove general words like "trojan," "backdoor," "malware," etc., and choose the most frequent token as the malware family name. Adding the two malware samples that were provided as ground truth, these three sites dropped malware from 6 different families during a single day. When we look at the URLs that host the files closely, they share the same URL path pattern - a simple path `'/counter/'`. All the files directly hosted by these three URLs also share similar naming pattern - a file name with 7 digits. This leads us to believe that these sites were part of an exploit kit deployment. We also carried out a case study on PUP operation, which can be found in Appendix A.

7.4 Combined malware and PUP operation.

In this section, we study a case that demonstrates that Marmite's label propagation can enable us to detect PUPs starting from malware labels. This case study shows that malicious operations in which malware downloads PUP or vice versa exist, although recent research showed that they are rare [17]. Figure 10 shows part of a download graph built by Marmite on June 1, 2016. Before running Marmite, files `N1NBGXOOKAQD.EXE` with SHA2 `20749...`, `YhLGsb1eNlqf.exe` with SHA2 `D1AFB...` and `coi1.exe` with SHA2 `63B2A...` were known to belong to the `Kryptik`, `SelfDel` and `Trojan.Skeeyah` malware families respectively. All the other files had unknown reputations. After running Marmite, files `EITVUYTPB63C.EXE` with SHA2 `E413D...` and `ts_10051.exe` with SHA2 `D530D...` were detected as malicious. These files were further identified as PUP by VirusTotal, respectively as `PUP.SearchGo` and `PUP.Neobar`. As pointed out in [17], there is a fundamental difference between malware distribution and PUP, as malware is mostly delivered through drive-by downloads while users actively install PUP through deception. This case is therefore very interesting since we see the

malicious group delivers both PUPs and malware via the same infrastructure.

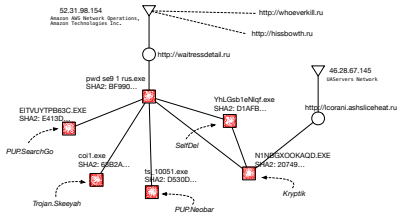

Figure 10: Operation consisting of both PUP and malware.

8 DISCUSSION

Possible practical deployments of Marmite**.** We showed that Marmite can efficiently increase the knowledge of malware samples, identifying malware that was not detected by VirusTotal months after it was observed in the wild. We mainly envision Marmite as a method that security companies can use to improve their internal knowledge of millions of malicious files they collect on a daily basis. The blacklists generated could then be integrated with existing end-user protection programs, such as antivirus tools and browser blacklists [28].

Limitations. Although Marmite is a useful tool for detecting malicious files, it has some limitations. First of all, Marmite relies on an initial set of labeled nodes to infer the probabilistic labels of other unlabeled nodes in the download graph. As we showed, Marmite needs a minimal number of labeled nodes to be effective, but this could be a problem in some settings. Another limitation is that Marmite only infers probabilistic labels of unlabeled nodes in the download graph and does not update the labels of seed nodes during the inference process. This is typically not a problem because AV vendors are conservative in the way they assign labels, but it is a limitation to keep in mind.

Evasion. Marmite does not make any assumptions on how malicious files are delivered, or on the structure of these malware delivery networks. The only assumption that is made is that cybercriminal are either delivering the same file over different servers or are reusing parts of their infrastructure to serve multiple malware samples. This design choice makes Marmite less prone to evasion than previous systems. Assuming Marmite is known to cybercriminals, there are three possible ways that they could try to evade Marmite. They could try to compartmentalize their infrastructures, and serve a single malware sample with unique SHA2 from each malicious server. Although this could be effective in evading our system, we argue that it would not make sense for the malware ecosystem in general, as it would make malware delivery much more expensive (due to the need of setting up a higher number of servers) and it would ultimately break the pay-per-install ecosystem, since these networks would not be able to provide malware samples from multiple customers anymore. They could also try to evade Marmite by dropping more benign files together with malicious binaries, causing our system to potentially flag those benign files as malware. Again, even though this could be effective in evading our system, we argue that this approach would also make malware delivery much more difficult (due to the need of binding a high number of benign files into the payload, leading to increased payload size, etc). As a third and final option, malware operators could start using CDNs exclusively to deliver their malicious files. While Marmite would find it difficult to keep track of such files due to the whitelisting process, this would make malware operators more visible to the CDNs themselves, who could track such malicious activity and terminate their accounts.

9 RELATED WORK

Studying malware delivery networks. In 2011, Caballero *et al.* [5] provided the first large scale measurement of pay-per-install services in the wild. This work confirmed the previous observations that cybercriminals are commonly using other botnets (known as *droppers*) to deliver their malicious payloads [33, 34]. In 2012, Grier *et al.* [8] studied the phenomenon of exploit kits, which are an alternative way to distribute malware. Nelms *et al.* [24] proposed Webwitness, a system that automatically builds the sequence of events followed by users before downloading a malware sample. More recently, researchers have been studying the ecosystem of potentially unwanted software (PUP), which includes toolbars and adware. Thomas et al. [37] performed a systematic study of PUP and its prevalence and its distribution through pay-per-install services. Kotzias et al. [17] identify PUP publishers and then study PPI publishers and their involvement in PUP.

Leveraging malware delivery networks for malware detection. Invernizzi *et al.* [11] proposed Nazca, a system to detect malware in large-scale networks. The system builds malware distribution networks using the HTTP traffic data generated when shellcode downloads the actual malware binary and launches it, and uses this graph to gain insights of various malicious activities associated with the graphs and train a decision tree classifier to detect malicious candidates.Nazca reports 70.59% precision and 100% recall. Marmite favors lowering the number of false detections, and reports 93% TPR for 3.5% TPR in similar settings.

Abu Rajab *et al.* [28] proposed CAMP, a content-agnostic malware detection system which utilizes reputation-based detection to protect users. Leveraging aggregated data sources, CAMP predicts the likelihood that this downloaded binary is malicious. CAMP reports a TPR of 70% and a FPR of around 2%. Accepting a similar FPR, Marmite achieves a TPR of over 90%. Rabharinia *et al.* [27] presented Mastino, a system that performs classification over behavioral characteristics of how malicious files are downloaded by machines on the Internet. Mastino uses domain-specific features such as characteristics of the URLs that files are downloaded from and characteristics of the files themselves. By using this domain-specific information, Mastino reaches 95% TPR with 0.5% FPs. As we mentioned, Marmite does not use any contextual information, and as we show this has the advantage that the system is still capable of efficiently detect malware six months after it was trained. For fairness though, Mastino reaches better results in the short term. Including the whitelisting effort, Marmite reports 1.9% FPs for 93% TPR. Nachenberg *et al.* [23] presented Polonium, a system that builds bipartite graphs of files and machines on which these files are installed and performs belief propagation to detect malware. The pervasiveness of polymorphism and the subsequent diffusion

of singleton files might limit the effectiveness of this system in modern malware delivery. Polonium reported a TPR of 84% and a FPR of 1% over 2011 malware data. This is in line with what was obtained with 2016 malware by Marmite during the validation stage.

The closest work to this paper is [19]. The authors reconstructed and analyzed 19 million downloader graphs from 5 million real hosts, then identified several strong indicators of malicious activity including growth rate, the diameter, and the Internet access patterns of downloader graphs. Building on top of these insights, the authors implemented and evaluated a machine learning system using random forest for malware detection, and proved to achieve high true positive rate and low false positive in detecting malware. This system has the limitation of only taking into account local visibility for single hosts. Marmite, on the other hand, is able to leverage a global view of malware delivery networks.

We showed that Marmite can operate effectively without any re-tuning even six months after the parameters were trained. For a high level comparison, [19] reports 98% TPR with 2% FPR with the full feature set, and 81% TPR and 21% FPR with only the features that are related to the local download graph, and therefore do not take domain-specific information into account. Marmite performs worse than this system compared to the full feature set (88% TPR for 2% FPR during the validation phase) but clearly outperforms it when compared to the graph-only features.

A number of systems aim at detecting malicious hosts based on structural properties of malware delivery networks. Although the goal of Marmite is to detect malicious files, these approaches have similarities with our system in the way they operate. Zhang *et al.* [41] proposed ARROW, a system to detect drive-by download attacks. The system builds a hostname-IP mapping to identify central servers of malware distribution networks (MDNs), and generates corresponding signatures. These signatures are later used to detect malicious webpages.

Li *et al.* [20] performed a large scale study on the topological relations among hosts in the malicious Web infrastructure. The system constructs hostname-IP clusters (HICs) and builds topological relationship between HICs. Utilizing the observation that there is a higher density of interconnections among malicious infrastructures than in the rest of the web, a variation of page-rank algorithm is employed to detect dedicated malicious hosts. Stokes *et al.* [32] proposed WebCop, a bottom up approach to detect malware distribution sites. The system uses the final destination distribution sites as the starting point, and follows the web graph hyperlinks in reverse to identify the higher level landing sites. WebCop further utilizes the identified landing sites to detect unknown distribution sites that share a landing site with a known malware distribution site. Stringhini *et al.* [35] presented SpiderWeb, a system that builds graphs of HTTP redirections used in the delivery of malware, and performs classifications on these graphs for malware detection. Mekky *et al.* [22] expanded on this model, looking not only at automated redirections but also at the links clicked by users. Manadhata *et al.* [21] presented an approach to perform belief propagation over download graphs to detect malicious hosts. This system is designed to operate over proxy logs for a single organization, while Marmite is designed to operate over the entire Internet. Although Marmite could be extended to detect malicious hosts, in its current implementation it is designed to identify malicious files.

Compared to all these approaches, Marmite is generic as it does not rely to particular network structures and protocols. This is an important advantage, because it makes our approach applicable to settings different than HTTP. Marmite also does not rely on features that are typical of how cybercriminals operate (e.g., their use of Domain Generation Algorithms of Fast Flux), and is therefore resilient to evasion — in fact, we showed that Marmite is still able to efficiently detect malware six months after the system was tuned.

10 CONCLUSION

We presented Marmite, a system that is able to detect malicious files by leveraging a global download graph and label propagation with Bayesian confidence. We showed that the global download graph used by Marmite does not significantly change over time, and therefore our system can detect malware for over six months without need of being re-tuned. We were able to grow our knowledge of malware samples by up to eleven times compared to the malicious seeds used, and we showed that 36% of our detections do not appear on VirusTotal three months after they were detected by Marmite. We presented a number of case studies that aim to shed light on malware delivery ecosystems. We hope that these examples will help our community better understand the idiosyncrasies associated with malware delivery, and devise better mitigation systems based on these observations. From our end, we showed that building a whitelist of known benign files can be a simple and durable solution to systematic false positives.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their feedback, and our shepherd Christian Rossow for his help in improving the final version of this paper. This work was supported by UCL through a BEAMS Future Leaders in Engineering and Physical Sciences Award and by the EPSRC under grant EP/N008448/1.

REFERENCES

[1] National Software Reference Library. http://www.nsrl.nist.gov/.
[2] VirusTotal. https://www.virustotal.com.
[3] 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 *USENIX Security Symposium* (2012).
[4] Bergstra, J., and Bengio, Y. Random Search for Hyper-parameter Optimization. *Journal of Machine Learning Research* (Feb. 2012).
[5] Caballero, J., Grier, C., Kreibich, C., and Paxson, V. Measuring pay-per-install: The commoditization of malware distribution. In *USENIX Security Symposium* (2011).
[6] Egele, M., Scholte, T., Kirda, E., and Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computer Surveys 44*, 2 (2012).
[7] Fawcett, T. An introduction to roc analysis. *Pattern Recogn. Lett. 27*, 8 (June 2006), 861–874.
[8] Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C. J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., et al. Manufacturing compromise: the emergence of exploit-as-a-service. In *ACM conference on Computer and communications security (CCS)* (2012).
[9] Holz, T., Gorecki, C., Rieck, K., and Freiling, F. C. Measuring and Detecting Fast-Flux Service Networks. In *Network and Distributed Systems Security Symposium (NDSS)* (2008).
[10] Invernizzi, L., Comparetti, P. M., Benvenuti, S., Cova, M., Kruegel, C., and Vigna, G. EvilSeed: A Guided Approach to Finding Malicious Web Pages. In *IEEE Symposium on Security and Privacy* (2012).

[11] Invernizzi, L., Miskovic, S., Torres, R., Kruegel, C., Saha, S., Vigna, G., Lee, S., and Mellia, M. Nazca: Detecting malware distribution in large-scale networks. In *Network and Distributed System Security Symposium (NDSS)* (2014).

[12] Kapravelos, A., Shoshitaishvili, Y., Cova, M., Kruegel, C., and Vigna, G. Revolver: An automated approach to the detection of evasive web-based malware. In *USENIX Security Symposium* (2013).

[13] Karampatziakis, N., Stokes, J. W., Thomas, A., and Marinescu, M. *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2013, ch. Using File Relationships in Malware Classification.

[14] Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., and Kirda, E. Cutting the gordian knot: a look under the hood of ransomware attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2015).

[15] Kirat, D., Vigna, G., and Kruegel, C. Barecloud: bare-metal analysis-based evasive malware detection. In *USENIX Security Symposium* (2014).

[16] Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI* (1995).

[17] Kotzias, P., Bilge, L., and Caballero, J. Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services. In *USENIX Security Symposium* (2016).

[18] Kruegel, C., Robertson, W., Valeur, F., and Vigna, G. Static disassembly of obfuscated binaries. In *USENIX Security Symposium* (2004).

[19] Kwon, B. J., Mondal, J., Jang, J., Bilge, L., and Dumitras, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In *ACM Conference on Computer and Communications Security (CCS)* (2015).

[20] Li, Z., Alrwais, S., Xie, Y., Yu, F., and Wang, X. Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures. In *IEEE Symposium on Security and Privacy* (2013).

[21] Manadhata, P. K., Yadav, S., Rao, P., and Horne, W. Detecting malicious domains via graph inference. In *European Symposium on Research in Computer Security (ESORICS)* (2014).

[22] Mekky, H., Torres, R., Zhang, Z.-L., Saha, S., and Nucci, A. Detecting malicious HTTP redirections using trees of user browsing activity. In *INFOCOM* (2014).

[23] Nachenberg, C., Wilhelm, J., Wright, A., and Faloutsos, C. Polonium: Tera-scale graph mining and inference for malware detection.

[24] Nelms, T., Perdisci, R., Antonakakis, M., and Ahamad, M. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *USENIX Security Symposium* (2015).

[25] Nguyen, G. H., Bouzerdoum, A., and Phung, S. L. Learning pattern classification tasks with imbalanced data sets. Tech. rep., 2009.

[26] Pitsillidis, A., Kanich, C., Voelker, G. M., Levchenko, K., and Savage, S. Taster's choice: A comparative analysis of spam feeds. In *ACM Conference on Internet Measurement Conference (IMC)* (2012).

[27] Rahbarinia, B., Balduzzi, M., and Perdisci, R. Real-time detection of malware downloads via large-scale url− > file− > machine graph mining. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)* (2016).

[28] Rajab, M. A., Ballard, L., Lutz, N., Mavrommatis, P., and Provos, N. Camp: Content-agnostic malware protection. In *Network and Distributed System Security Symposium (NDSS)* (2013).

[29] Ramachandran, A., Dagon, D., and Feamster, N. Can DNS-based blacklists keep up with bots? In *CEAS* (2006).

[30] Rossow, C., Dietrich, C., and Bos, H. Large-scale analysis of malware downloaders. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2013.

[31] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., Liang, Z., Newsome, J., Poosankam, P., and Saxena, P. Bitblaze: A new approach to computer security via binary analysis. In *Information systems security*. 2008.

[32] Stokes, J. W., Andersen, R., Seifert, C., and Chellapilla, K. Webcop: Locating neighborhoods of malware on the web. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2010).

[33] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., and Vigna, G. Your botnet is my botnet: analysis of a botnet takeover. In *ACM conference on Computer and communications security (CCS)* (2009).

[34] Stone-Gross, B., Holz, T., Stringhini, G., and Vigna, G. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. In *Workshop on lage-scale exploits and emerging threats (LEET)* (2011).

[35] Stringhini, G., Kruegel, C., and Vigna, G. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *ACM conference on Computer and communications security (CCS)* (2013).

[36] Tamersoy, A., Roundy, K., and Chau, D. H. Guilt by association: Large scale malware detection by mining file-relation graphs. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014).

[37] Thomas, K., Crespo, J., Picod, J.-M., Phillips, C., Sharp, C., Decoste, M.-A., Tofigh, A., Courteau, M.-A., Ballard, L., Shield, R., Jagpal, N., Abu Rajab, M., Mavrommatis, P., Provos, N., Bursztein, E., and McCoy, D. Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software. In *USENIX Security Symposium* (2016).

[38] Vadrevu, P., Rahbarinia, B., Perdisci, R., Li, K., and Antonakakis, M. Measuring and detecting malware downloads in live network traffic. In *ESORICS*. 2013.

[39] Willems, C., Holz, T., and Freiling, F. Toward automated dynamic malware analysis using cwsandbox. In *IEEE Symposium on Security & Privacy* (2007).

[40] Yamaguchi, Y., Faloutsos, C., and Kitagawa, H. *PAKDD*. 2015, ch. SocNL: Bayesian Label Propagation with Confidence.

[41] Zhang, J., Seifert, C., Stokes, J. W., and Lee, W. Arrow: Generating signatures to detect drive-by downloads. In *International World Wide Web Conference (WWW)* (2011).

[42] Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML* (2003).

A PUP OPERATION

As an additional experiment, we also wanted to study PUP binaries as discussed in Section 6.1. We use a download graph from January 3rd in this study. Figure 11 illustrates one of these cases. The site `balancer1.amber1glue.com` was hosted by three IP addresses in the course of the day. The first two belong to web hosting company `CloudFlare`, and the last is from `MTS PJSC`, registered in Russia. In total, 72 files dropped from this URL were previously identified and confirmed to belong to the `Mizenota` family (via VirusTotal). 811 files are marked as PUP by Symantec. Both 72 confirmed malicious and 811 newly discovered files share the same naming pattern: '`string__10924_i<10-digit>_il<7-digit>.exe`', e.g. '`MS+Office+2010+Crack+Prod__10924_i18078 21319_il2622354.exe`', '`Structural+analysis +hibbe__10924_i1807858669_il2 64 0857.exe`'. Another binary (classified as `Amonetize`) with SHA2 `9A91B...` also dropped five PUP files with the same name but different SHA2s. This case study shows that different PUP-related PPI services, already studied in previous work [17, 37], often share the same delivery infrastructure.

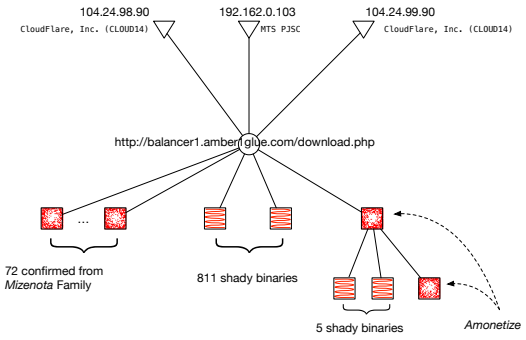

Figure 11: PUP case study detected by Marmite.

This case study illustrates the blurry line between malware and PUP files. While some files are classified as malware by VirusTotal, some other are not, and are considered as PUP by Symantec. The label propagation performed by Marmite is able to flag all of them as malicious.