Network Connectivity Graph for Malicious Traffic Dissection

Network Connectivity Graph for Malicious Traffic Dissection

Enrico Bocchi*, Luigi Grimaudo*, Marco Mellia*, Elena Baralis*,
Sabyasachi Saha†, Stanislav Miskovic†, Gaspar Modelo-Howard†, Sung-Ju Lee◇

*Politecnico di Torino
{name.surname}@polito.it

†Symantec, Corp.
{name_surname}@symantec.com

◇KAIST
sjlee@cs.kaist.ac.kr

***Abstract*—Malware is a major threat to security and privacy of network users. A huge variety of malware typically spreads over the Internet, evolving every day, and challenging the research community and security practitioners to improve the effectiveness of countermeasures.**

In this paper, we present a system that automatically extracts patterns of network activity related to a specific malicious event, *i.e.*, a *seed*. Our system is based on a methodology that correlates network events of hosts normally connected to the Internet over (i) time (*i.e.*, analyzing different samples of traffic from the same host), (ii) space (*i.e.*, correlating patterns across different hosts), and (iii) network layers (*e.g.*, HTTP, DNS, etc.). The result is a *Network Connectivity Graph* that captures the overall "network behavior" of the seed. That is a focused and enriched representation of the malicious pattern infected hosts exhibit, purified from ordinary network activities and background traffic.

We applied our approach on a large dataset collected in a real commercial ISP where the aggregated traffic produced by more than 20,000 households has been monitored. A commercial IDS has been used to complement network data with alerts related to malicious activities. We use such alerts to trigger our processing system. Results shows that the richness of the Network Connectivity Graph provides a much more detailed picture of malicious activities, considerably enhancing our understanding.

I. Introduction

Information security over the Internet remains a primary concern for consumers, enterprises, and governments alike. Malware infiltrates and spreads over the Internet hiding its traffic among humongous benign traffic. Cyber-attackers also use sophisticated schemes to modify malware and evade detection by security measures. Recent industry reports disclose that existing antivirus software's detection rate of newly crafted viruses is less than 5% [1]. This creates a de facto arm's race between security researchers and cyber-attackers.

Practitioners in the field take different approaches to detect malware, which result in a set of methodologies ranging from instruction set and code analysis, to traffic characterization of infected hosts. For instance, a detection rule can be designed by studying the behavior of infected hosts in a controlled environment (*e.g.*, a honeypot). In this scenario, it is possible to identify the communication channel with the Command and Control (C&C) network, or define signatures for proprietary and obfuscated protocols used by malicious software.

However, it is generally more complicated to obtain a complete picture of the overall malware behavior due to the evolution of malware itself to circumvent counteractions, and the lack of tools that easily extract facts related to malicious activities. Thus, to obtain a thorough and up to date picture, security specialists are called to a manual and cumbersome analysis of data produced by infected hosts in the wild.

In this paper, we present a methodology to automatically extract detailed network patterns generated by infected hosts. We consider the vantage point offered by a network where the traffic aggregate of thousands of possibly infected hosts flows. A passive monitoring tool extracts and logs *events* from the traffic. An event could be a HTTP request, a DNS response, or simply a TCP flow going to a host using an unknown protocol. A security monitor, *e.g.*, an Intrusion Detection System (IDS), analyzes the traffic in parallel, and flags some of the events as malicious, according to a database of already available rules. These flagged events are the *seeds* that trigger our analysis. Given the traffic log and a seed, our system provides a set of "forensic" information to the security analyst for a better understanding of the context in which malicious events take place. Ultimately, it extracts a detailed and complete picture of the malicious activities correlated with the seed event.

Identifying the subset of events that belongs to the same activity is a challenging task as each host generates thousands of events caused by multiple applications running concurrently. For instance, the same host could visit a legitimate web page, poll the mail server, and upload files on a cloud storage service, while a malware is connecting to a C&C node that instructs victims with new malicious instructions. Furthermore, the sequence at which events appear is typically not deterministic with randomness due to diversity (*e.g.*, two hosts visiting the same page can fetch objects in a different order), and system memory (*e.g.*, a DNS request not appearing in the traffic as the server name has been previously resolved and cached).

Our approach is based on a filtering and enrichment process that leverages (i) temporal and (ii) spatial repetitiveness of events generated by different hosts. The intuition is to look for common patterns that are present in different snapshots from the same host, and among different hosts. We explicitly go after repetitive and popular events. In practice, as few as three observations of a malicious seed are enough to trigger our methodology. The result is offered as a *Network Connectivity*

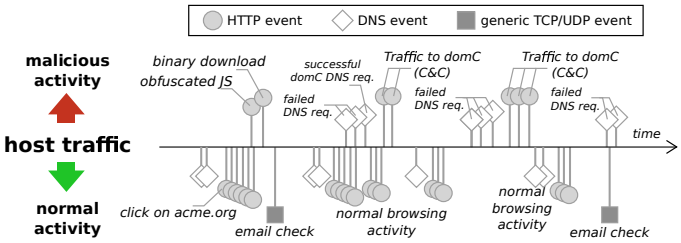


Fig. 1. Example of events generated by a host as seen from the network.

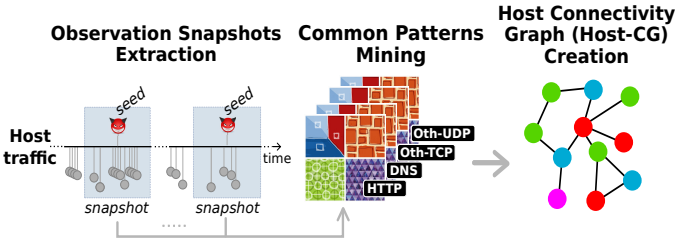


Fig. 2. Host Connectivity Graph generation.

Graph (CG), which models only events highly correlated with the seed, and allows us to easily navigate and extract valuable information using our domain knowledge.

We start from real traffic collected in an operative network where more than 20,000 households are monitored. In a day, more than 336 M events are logged by the passive monitoring tool. A commercial IDS rises alarms for 1,700 unique malicious seeds, generating 42,000 events and belonging to more than 150 different threats. Out of those events, about 40,000 (95%) are processed by our system as they show the required properties of repetitiveness. For each seed, we run the filtering and enrichment process. At the end, the information offered in the final CG grows by a factor of 40, i.e., starting from a single seed, which is represented by three nodes in the CG, 160 nodes are present on average in the final CG.

Visual inspection allows us to immediately spot (i) the malicious infrastructure (*e.g.*, the presence of new C&C nodes), (ii) malicious attacks interfering with legitimate infrastructures (*e.g.*, the exploitation of benign websites to force the download of Exploit Kits), and (iii) some evasion techniques adversaries uses (*e.g.*, the usage of DNS fast-fluxing [2]).

The contributions of our work are as follows:

- We propose a methodology that extracts and represents the network activity surrounding a malicious seed, which is useful to identify and derive a detailed superset of events correlated to it.
- We take a multi-layer approach that combines the connectivity between different protocol layers to uncover hidden behavior and provide forensic information.
- We offer the information in the form a Network Connectivity Graph, that is a straightforward means to represent the common activity of malicious incidents.

We believe that applications of the CG go beyond the simple visualization of the malicious activity. For instance, signatures of the IDS can be updated and enriched, or the CG can be used as a signature itself to design novel behavioral classifiers able to distinguish between CGs derived from malicious or benign seeds. We leave these contributions as future work.

II. Methodology Overview

Before presenting the details of our system, we provide an overview and the intuitions behind its design.

A. Scenario

We consider a scenario in which a sniffer passively monitors the traffic generated by a large group of hosts, *e.g.*, hosts in an enterprise network, or households connected to a Point of Presence (POP) of an Internet Service Provider (ISP). The sniffer extracts information from the packets and logs them in a file where each row corresponds to a different *event*. We assume that, for each TCP and UDP connection, the sniffer logs the flow identifier, the timestamp of the first packet, the flow duration, the number of exchanged packets and bytes, etc. For some protocols, the sniffer can provide multiple events with very detailed information. For instance, it could annotate each HTTP request/response with the requested URL, user-agent, content-type, server response status code, etc.

Consider the timeline generated by a host reported in Fig. 1. It details the logged events generated by Internet applications. DNS and HTTP events are reported using specific markers, while other protocols are reported as generic TCP/UDP events. The user is visiting some web page (*e.g.*, acme.org) while an email client is polling a mail server for new messages. Normal events are reported in the bottom part of the timeline. Unfortunately, acme.org is hosting a Drive-by Download page. Events on the upper part are due to the malicious activity in which the host is unknowingly fooled to download a malware from a malicious JavaScript contained in the web page. We observe the download of the JavaScript object, followed by the download of the malware. Once running at the host, the malware periodically contacts (via HTTP) a C&C server whose hostname is quickly rotated using fast-flux [2]. The periodic polling is visible in the log as a sequence of (failed and successful) DNS requests, and HTTP traffic to the C&C node.

Based on the view of the traffic from all monitored hosts, we design a methodology that extracts and characterizes common network activities. The challenge is how to isolate the events that are possibly correlated with a specific malicious activity from the "background" noise caused by other events.

B. Network Connectivity Graph

Consider a seed and the timeline around it. Intuitively, close-in-time events are likely to be related to it. For instance, in Fig. 1, the DNS request followed by several HTTP requests to the acme.org server could be identified as a typical pattern. However, Drive-by Download attacks [3] can mimic or be hidden in the same behavior. To isolate them, we study snapshots of traffic that contain the specific seeds.

Fig. 2 shows the workflow used to transform the events of a given host into a *Host Connectivity Graph* (Host-CG). Three steps are executed: (i) Snapshots extraction; (ii) Per-layer common patterns mining; and (iii) Host-CG creation.

Algorithm 1 Create Network Connectivity Graph.

input args	s: seed H: set of hosts Δ: snapshot duration
output	Seed Connectivity Graph

```
1:  procedure graphLayer(s, S, layer):
2:    P = findCommonPattern(s, S, layer)
3:    return fromPatternsToGraph(P, layer)

4:  procedure hostConnectivityGraph(s, h, Δ):
5:    S = getSnapshots(s, h, Δ)
6:    g_HTTP = graphLayer(s, S, 'HTTP')
7:    g_DNS = graphLayer(s, S, 'DNS')
8:    g_TCP = graphLayer(s, S, 'TCP')
9:    g_UDP = graphLayer(s, S, 'UDP')
10:   return connectLayers(g_HTTP, g_DNS, g_TCP, g_UDP)

11: procedure seedConnectivityGraph(s, H, Δ):
12:   G_s = ∅
13:   foreach h ∈ H:
14:     G_s ← hostConnectivityGraph(s, h, Δ)
15:   return fuseGraphs(G_s)
```

Snapshots Extraction. For each instance of the seed, we extract a *snapshot* defined as the *ordered* set of events occurring in the temporal window centered at the seed. Two snapshots are presented in Fig. 2 as example.

Common Patterns Mining. We then look for commonalities across snapshots. In particular, we look for *patterns*, defined as the *unordered* set of events, that appear across multiple snapshots. Intuitively, the periodic HTTP requests toward the C&C server would possibly be a repeating pattern on the HTTP-layer. On the contrary, the web browsing events asynchronously generated by the host would be present only in a small subset of snapshots.

We extract separate common patterns by processing the host traffic considering *layers* in isolation. The traffic generated on each layer corresponds to all events of a specific protocol so that HTTP, DNS, other-TCP (*i.e.*, all TCP communication except HTTP on port 80), and other-UDP (*i.e.*, all UDP communication except UDP on port 53) events are separately analyzed. This choice originates from the fact that each protocol has some peculiarities that we would leverage. For instance, in the HTTP layer, we are looking for common and repetitive patterns. On the DNS layer instead, a single failed DNS request may be more interesting than successful DNS requests.

Host Connectivity Graph. For each layer, we represent the common pattern as a graph where nodes and edges are specifically defined to offer a compact yet rich representation. Consider the HTTP-layer. URLs can be represented by separating server hostnames and object paths using two nodes: An edge between the hostname and the path would thus represent a URL. The resulting graph captures the website structure. For example, acme.org/index.html and acme.org/logo.png are represented with a hostname node (acme.org) and two objects nodes. Similarly, in the DNS layer, the request for acme.org is linked to the IP address(es) returned by the resolver.

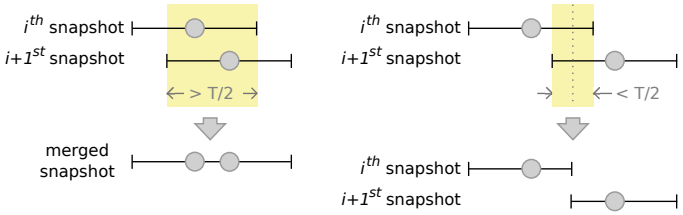


Fig. 3. Snapshots creation when consecutive snapshots overlap.

As last step, we connect each per-layer graph into a final *Host Connectivity Graph*. This is done by adding links across multiple layers. For instance, the acme.org hostname in the HTTP layer is linked to the same node in the DNS layer graph.

The resulting graph is a rich and compact representation of the common network activity related to a specific seed and a given host. Each layer brings a specific characterization of the activity given by a protocol, resulting in an overall integration of common patterns. In the previous example, the HTTP layer highlights the common websites hosting the binary download, while the DNS layer reveals failing requests triggered before or after C&C communications. Focusing only on each activity individually would miss such relationship.

Seed Connectivity Graph. We leverage the fact that the same seed may be present in the timeline of different hosts, offering some "spatial" diversity. To have a broader view of the common activity related to a specific seed, we "fuse" multiple Host-CGs into a single *Seed Connectivity Graph* (Seed-CG). As for the common pattern, we have the freedom to choose between a selective fusion, *e.g.*, retaining only those common nodes from all Host-CGs, or a permissive fusion, *e.g.*, merging all nodes from all Host-CGs.

III. Building the Connectivity Graph

The key aspect of the proposed methodology is the approach used to create Network Connectivity Graphs. The pseudo-code in Alg. 1 details this procedure. This section discusses the design choice taken and the parameters to be controlled when creating a network connectivity graph.

A. Snapshots Extraction

The first step to process host traffic is the extraction of the observation snapshots. We define parameter Δ that controls the duration of the snapshots. In particular, a snapshot is composed by all events occurring in the interval $\pm\Delta/2$ centered around the seed. In case consecutive snapshots overlap, we apply two strategies depicted in Fig. 3 to solve the conflict. If the overlapping window lasts for more than $\Delta/2$, the two snapshots are merged. Otherwise, the overlap is split into two halves, each associated to a different snapshot. These operations are executed by *getSnapshots*() (Alg. 1 line:5) that receives the seed (s), a host (h) presenting at least one instance of the seed, and the snapshot duration (Δ) as inputs. It returns the set of snapshots (S) found.

Different values of Δ can lead to different results: the larger the Δ (*e.g.*, hours), the more the snapshots will merge. This

results in less snapshots on which to perform pattern mining, with each presenting "noisy" data since not many events are filtered. Conversely, a small value of Δ (*e.g.*, seconds) might be too conservative. In the following, we set Δ = 30 minutes. A complete sensitivity analysis is reported in Sec. V-B.

B. Common Patterns Mining

We use the frequent itemset mining technique to extract common patterns [4]. This technique works on unordered sets of simple objects (*e.g.*, strings). Snapshots, however, correspond to ordered sequences of events that may appear multiple times. We thus map each event to an *item* based on the event properties. Specifically:

- a HTTP item is represented by the HTTP URLs, *e.g.*, *http://domain.com/path/file.ext*.
- a DNS item combines the requested hostname with either the list of returned IP addresses, or the query response code, *e.g.*, *DoesNotExists.com–NXDomain*.
- TCP and UDP items are represented by the server IP address and the port contacted, *e.g.*, *10.20.30.40–443*.

For each snapshot, we create a *transaction* containing the set of *distinct* items. We look for common *itemsets*, *i.e.*, sets of items common across multiple transactions. A *support* value is computed for each itemset and indicates the fraction of transactions containing the specific itemset. An itemset is "frequent" if its support is greater than or equal to MinSupport. For a given support value, the itemset presenting the highest number of items is said to be *closed*. The closed attribute implies that no other itemset made by more items has the same support.

Itemsets with a number of items smaller than MinLength could be discarded. By setting MinLength=1, frequent itemsets are equivalent to simple frequent items in terms of Connectivity Graph elements. For MinLength=2, at least pairs of items are considered. For instance, consider acme.org/index.html and acme.org/logo.png that appear in 70% and 45% of snapshots, respectively. The itemset (acme.org/index.html, acme.org/logo.png) may appear from 15% to 45% of snapshots.

Looking for all itemsets is a NP-hard problem [5], but well-known algorithms compute frequent closed itemsets efficiently. Among those, we rely on the Carpenter algorithm [6], which is specifically designed for datasets made of few transactions (*i.e.*, snapshots) that have a huge number of items (*i.e.*, events).

Our system looks for *frequent closed itemsets* that, for simplicity, we call *patterns*. Patterns are extracted by *findCommonPatterns*() (Alg. 1 line:2), that receives the seed (s), the set of snapshots (S) and the layer (layer) to process. It returns the pattern (P). The pattern extraction process is guided by the definition of the value of MinSupport, *i.e.*, events that do not appear with frequency of at least MinSupport are discarded. We set MinSupport = 1/2, *i.e.*, for each host, we discard all events not appearing in at least half of the snapshots. Sensitivity analysis is detailed in Sec. V-B.

C. Host Connectivity Graph

As previously discussed, we individually process each layer to create separate graphs. The *graphLayer*() (Alg. 1 line:1) extracts patterns for a specific layer and maps them into a graph. This mapping exploits a subset of the events properties:

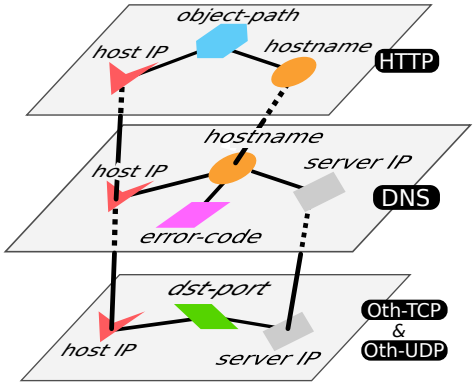


Fig. 4. Graph layers nodes and multi-layer connections.

- The HTTP layer has two node types: hostnames and object paths. An edge connects the hostname and the object path to compose a URL.
- The DNS layer has three node types: server hostnames, server IP addresses, and DNS error codes. An edge connects the hostname to either the IP addresses returned by a DNS response, or to an error code.
- The TCP and UDP layers have two node types: server IP addresses and server ports. An edge connects the two to represent a TCP or UDP connection.

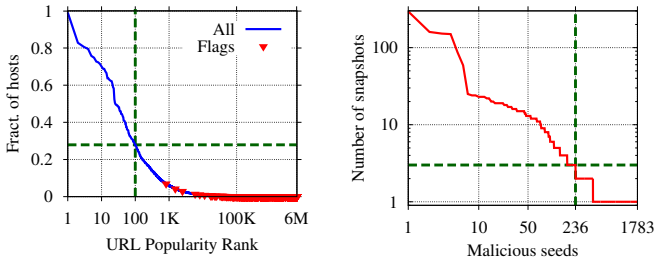
Different graph layers are combined in a single Host-CG using *hostConnectivityGraph*() (Alg. 1 line:4). The function starts by extracting the snapshots (S) related to the seed. The snapshots are then processed to extract the graph layers (g_{HTTP}, g_{DNS}, g_{TCP}, g_{UDP}) through calls to *graphLayer*(). The separate layers are finally integrated to form the Host-CG using the *collectLayers*() function, which looks for common nodes across the layers and links them as represented in Fig. 4. Notice that each graph layer contains the host (h) IP address by construction.

D. Seed Connectivity Graph

To provide the global view of the common behavior seen by observing multiple hosts, we combine all Host-CGs. This operation is performed by the *seedConnectivityGraph*() (Alg. 1 line:11) function. For each host (h) among the subset presenting the seed (H), the function creates the Host-CG calling *hostConnectivityGraph*(). All the output graph are collected into the set G_{hosts}. The graphs are finally merged using *fuseGraphs*(). This operation can consider different strategies. For instance, applying a strict intersection would retain only nodes appearing in all Host-CGs. In the worst case, this results in a Seed-CG containing only the original seed. More complex strategies can instead compute nodes and links popularity among hosts, and discard those below the threshold MinPopularity. In the following, we consider the strict intersection across Host-CGs as the default choice, *i.e.*, MinPopularity=1.

TABLE I. DATASET SUMMARY.

	All		Flagged	
Class	Hosts (%)	Events (%)	Hosts	Events
HTTP	16,217 (*79.1*)	39.7 M (*11.8*)	1,308	42,007
DNS	15,164 (*74.1*)	30.7 M (*9.3*)	-	-
Other TCP	18,911 (*92.31*)	40.8 M (*12.14*)	31	1,543
Other UDP	18,032 (*88.02*)	224.7 M (*66.87*)	-	-
Total	20,486	335.9 M	1,321	43,550



(a) Popularity of HTTP objects. (b) Snapshots per malicious events.

Fig. 5. Dataset characterization. (a).Top-100 HTTP objects are whitelisted (b).Seeds generating at least 3 snapshots are processed by our system.

IV. DATASET

We now describe the traffic traces and tools that we use to extract information to build the dataset that we use.

A. Data Collection

We consider a vantage point located in a commercial ISP where approximately 20,000 customers are connected. Most of the customers are residential users, connected via ADSL modems to the monitored point. Each customer modem is given a static IP address, which can be used to identify all the traffic generated/destined to the same household. In the following, we generalize the term "host" to refer to traffic exchanged by a single household (IP address).[1]

We consider a trace obtained live during one day in April 2012. A commercial monitoring tool processed the packets in real time to generate a text log file in which each TCP and UDP flow is logged. For each flow, a record is stored. It details the flow identifier (the tuple source/destination IP addresses, source destination ports and protocol type), the timestamp of the first packet, the total number of packets/bytes sent and received, the application-protocol used. In case protocol is HTTP, the entry is annotated with server hostname, object path, user-agent, content-type, response status (*e.g.*, `200 OK`), content-length directly extracted from the HTTP header[7]. In case multiple HTTP objects are fetched using the same TCP flow (*e.g.*, due to HTTP-persistent), multiple records are logged. Similarly, for each DNS transaction, the tool logs the requested hostname, the set of returned IP addresses, or the response code in case of an error (*e.g.*, *NXDomain*) [8]. To protect users privacy, sensitive information has been removed.

In parallel to the monitoring tool, a commercial IDS processed the packets in real time, logging alerts if some network activity matches any rule that is present in its database. We consider the IDS as an oracle that reveals which events are to be considered malicious. For each alert, the IDS simply specifies the flow identifier, and a *threat-ID*, *i.e.*, a numerical code that identifies a particular threat. The IDS is very conservative in triggering alerts and hence it is possible that some malicious events do not trigger any alert. Conversely, every alert is related to some malicious activity.

B. Dataset Overview

We consider each record in the log as a different *event*. By matching the flow identifiers, alerts are linked to records, so that records can be *flagged* as malicious. We obtain the *labeled dataset* described in Table I. Overall, 20,486 hosts generated about 336 M total events over the whole day. About 20% of those are related to HTTP and DNS records, with a large majority of the "Other TCP" due to TLS/SSL (HTTPS) traffic, and "Other UDP" events due to Peer-to-Peer applications.

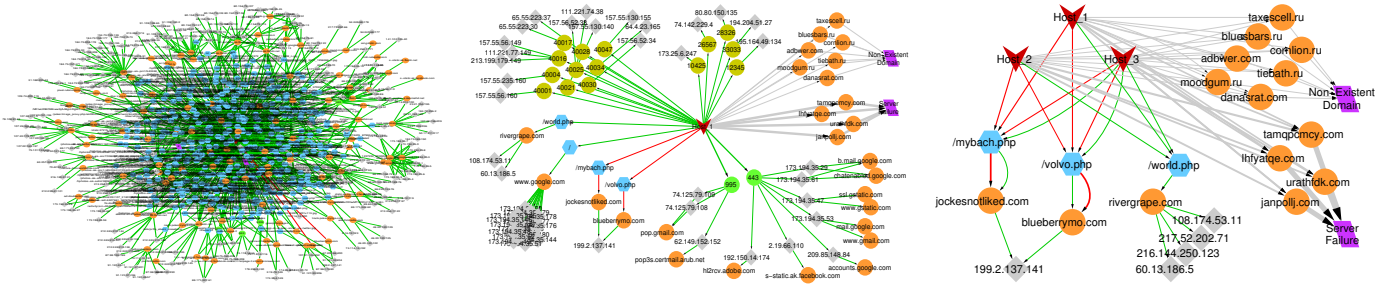
Among all users, 6.4% exhibit some malicious activity (*i.e.*, at least one event is flagged), with 151 different threat-IDs being reported by the IDS. Yet, only 43,550 flags are raised by the IDS. That translates to a negligible 0.013% of all traffic. Most of these records correspond to HTTP traffic, with the exception of some IRC and RPC flows. This confirms on the one hand the very stealthy and low rate activity that malware is typically generating. On the other hand, it confirms the conservative design of IDS. Almost all the flags are related to malicious HTTP activities including Exploit Kits (*e.g.*, Nuclear, Blackhole, ZeroAccess), Drive-by Downloads, Malicious Browser Toolbars (*e.g.*, Ask.com), Trojans and Worms (*e.g.*, Skintrim, Conficker), etc.

C. Whitelisting

Whitelisting is a common technique used to both reduce the amount of information processed, and to discard data that would possibly pollute the analysis. For the same purposes, we built a whitelist that targets very popular events that would be in any CG with high probability but add little information or create noise. Instead of creating a manual list of popular and benign events, we opt for a *dynamic* and *context-aware* approach. We build a whitelist based on events popularity among clients, and select the top-k elements to be ignored during the processing. We whitelist single HTTP events and not the entire websites, as it is known that malware can be hosted and distributed also from legitimate services.

Fig. 5(a) shows the HTTP events popularity, *i.e.*, the fraction of hosts that accessed a given URL (with stripped parameters). Note the log scale on x-axis. Fig. 5(a) shows the classic heavy tailed popularity. Top URLs are clearly very common among most of the hosts. Those include social network buttons (*e.g.*, www.facebook.com/plugins/like.php), analytics services (*e.g.*, www.google-analytics.com/ga.js), software update check (*e.g.*, download.windowsupdate.com/v9/windowsupdate/redir/muv4wuredir.cab), etc. Red triangles highlight those events that are considered malicious by the oracle. The most diffused

[1]Given the popularity of NAT (Network Address Translation) at home, the ADSL modem IP address identifies traffic exchanged by all devices accessing the Internet at each customer household.



(a) Initial graph from a single snapshot. (b) Host-CG of a single client. (c) Final Seed-CG.

Fig. 6. Evolution of Network Connectivity Graphs at several steps of our methodology. The event under study http://jockesnotliked.com/mybach.php is reported as malicious by our oracle. Three clients are flagged for this event: Two generate two analysis snapshots each, while the third client generates eight snapshots.

type of attack - a Drive-by Download threat - infects about 800 hosts (3.8% of hosts). The huge tail confirms the intuition that most of URLs are accessed by few hosts only. We conservatively compile a whitelist made of the Top-100 HTTP events, which equivalently filters those events that are common to more than 23% of hosts. This avoids blurring the common pattern mining and reduces the itemset extraction time, despite not affecting the descriptiveness of CGs.

V. Connectivity Graph Characterization

We next evaluate the benefits and properties of CG creation. We first identify the amount of events eligible of becoming seeds. Recall that our methodology requires a recurrence of seeds over time and over population. The requirement matches basic properties of malicious activities, such as recurrent reporting to the C&C center or recurrent attempts to identify new victims. For this reason, we expect that malware distributors would try to disguise such repetitiveness as much as possible. In our dataset we found 820 malicious hosts that had only one flagged event. If analyzed in isolation (on per host basis), these events would not have any recurrence.

Fig. 5(b) reports the number of snapshots that can be associated to each unique malicious event. By considering 1,783 unique malicious events, we found that 236 events can be uniquely associated to at least three independent snapshots. Setting MinSnapshots=3, these events become fully characterizable by our system. In fact, looking at the absolute numbers, we can provide insights for 95% of the malicious snapshots in our dataset (40k out of 42k events).

A. Connectivity Graph construction steps

In Fig. 6, we present an example of the Network Connectivity Graph evolution according to the steps of the methodology described in Sec. III. We consider the malicious seed *http://jockesnotliked.com/mybach.php*.

- Fig. 6(a) shows the network traffic produced by a host during a single 30 minute snapshot centered on the seed. Obviously, this graph is very difficult to interpret.
- Fig. 6(b) presents the Host-CG of one of the three clients (red marker in the center) involved in the malicious activity. Despite being already clearer and understandable with little effort, it still contains some nodes related to ordinary user's behavior that are not part of the malicious activity. For instance, HTTPS and POP3 Secure transactions on the TCP layer (light green circles in bottom-right part) are related to mail exchange and login to Google services, while flows over UDP (olive green circles in upper part) are mostly directed to the Skype service.
- Fig. 6(c) corresponds to the final Seed-CG generated by our system when fusing the Host-CG of three different hosts. It is now much easier to identify the events involved in the suspicious activities. Events highlighted by red edges are those considered malicious by our oracle. The richness of the provided indications stems from the augmented context that we provide about these clients. First, the clients access three URLs (blue hexagons) hosted by three hostnames (orange circles) all of which now become an indication of a suspicious infrastructure. Next, two of the contacted hostnames (*jockesnotliked.com* and *blueberrymo.com*) use the same IP address (gray diamonds) suggesting for a potential obfuscation by hostname flipping and resources reusage, both common practices among malicious adversaries. The third hostname (*rivergrape.com*) is distributed over several mirrors whose IP addresses belong to very different subnets, a hint of cheap infrastructure or zombies that were previously infected by the malware. Finally, the right side of Fig. 6(c) shows another layer of information indicating multiple failures of DNS queries (purple boxes). This reaffirms our suspiciousness.

Apart from providing more context for the malicious activity, our system discovers new malicious objects and improves the flagging consistency of our oracle IDS. For example, the object *bluberrymo.com/volvo.php* is consistently included in malicious graphs, while the IDS occasionally missed it. Our system also discovered a new object *rivergrape.com/world.php*, and we confirmed its maliciousness across several other security tools such as VirusTotal.[2]

B. Impact of Pattern Filtering

We study the volume of information that Seed-CG creation process extracts from single seeds. Table II shows the average number of nodes included in the final Seed-CGs. Three sets of parameters are reported, from a very selective common pattern

[2]VirusTotal: www.virustotal.com

TABLE II. AVERAGE NUMBER OF NODES FOR DIFFERENT TYPES AMONG SEED-CGS WITH Δ = 30 MIN.

Type	c1	c2	c3
Object-path	6.6	14.9	2351.4
Hostname	7.0	16.8	691.5
Server IP	19.9	95.9	3423.0
Dst-port TCP	0.2	0.5	79.9
Dst-port UDP	2.0	27.8	1335.1
DNS error	0.3	2.4	40.4
Total	36.0	158.4	7921.5

c1 = {MinSupport=1, minPopularity=1}
c2 = {MinSupport=0.5, minPopularity=1}
c3 = {MinSupport=0, minPopularity=0}

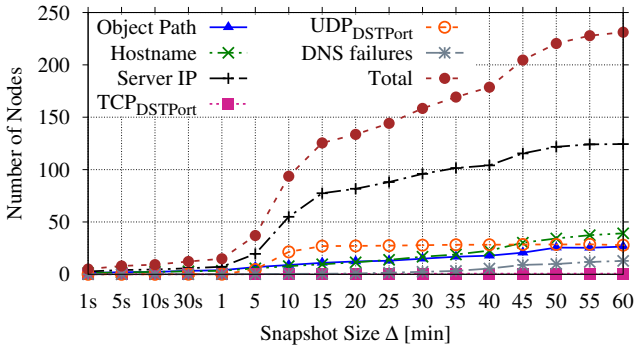


Fig. 7. Average amount of nodes in Seed-CGs with different snapshot sizes.

filtering such as c1 = {MinSupport=1, minPopularity=1}, which selects only the objects that appear in all snapshots and for all hosts, to c3 = {MinSupport=0, minPopularity=0}, which instead merges and fuses all patterns independently of their support and popularity. c2 = {MinSupport=0.5, minPopularity=1} is the suggested default parameter setting.

Results clearly show that starting from a single event, the proposed methodology builds graphs with hundreds of nodes. Note how the number of elements grows consistently when selecting less restrictive thresholds for MinSupport and minPopularity. The number of elements is indeed very large for c3, where thousands of nodes are included in the CGs. This hurts the amount of information offered to the security analyst (see Fig. 6(a) for instance). c2 offers a good trade-off between descriptiveness and richness of the final CG.

Fig. 7 shows the average number of nodes in Seed-CGs according to the selected snapshot size, Δ, for the parameter set c2. As expected, CGs contain only the seed event and few other nodes when selecting small Δ, *e.g.*, lower than 1 min. On the other hand, the number of nodes increases with the snapshot size, peaking at more than 200 nodes on average with a 60 min snapshot. Interestingly, the number of object-paths and hostnames does not increase dramatically with higher Δ, proving the effectiveness of the Common Patterns Mining technique. The only node type showing higher inflation is the Server IP address, owing to malicious activities poking benign infrastructures hosted on Content Delivery Networks (CDNs).

VI. CONNECTIVITY GRAPH EXAMPLES

In the following, we provide some examples of CGs covering several typologies of malware found in our dataset, each presenting different behaviors and network patterns.

A. *Cycbot Backdoor Activity*

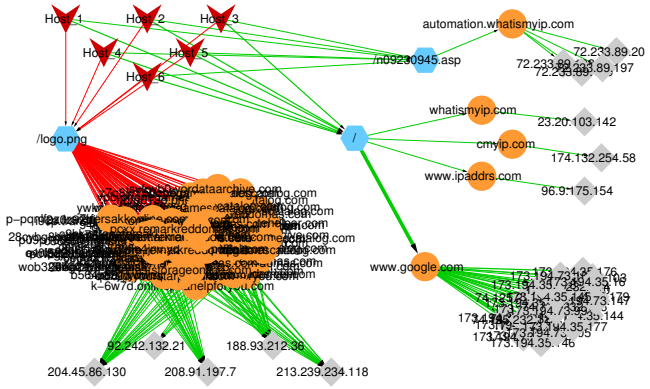


Fig. 8. CG for Cycbot Backdoor Activity backdoor trojan. Notice the *fast-flux* domain name shuffling (orange circles in lower-left part), and the IP address detection webpages (top-right) to assess victims' reachability.

Cycbot is a backdoor trojan that allows cyber-criminals to access infected computers remotely. This causes victims' hosts to be exploited by malicious adversaries for large-scale attacks, and to potential leakages of personal sensitive information.

Fig. 8 shows the CG of the event **/logo.png* for which our oracle raises an alarm. Interestingly, more than 80 hostnames seem to serve the malicious file *logo.png* (cloud of orange circles in Fig. 8). All those hostnames have a third-level domain name exhibiting random strings that are made of both characters and numbers, and are hard to code with regular expressions. This technique, known as *fast-flux*, allows attackers to hide malicious infrastructures by generating hostnames that are registered to the DNS and lately removed with a high frequency. This makes the detection harder, circumvents blacklisting, and guarantees a longer reachability to the infrastructure. Considering only second-level domains, the number of hostnames drops to 10, each presenting an appealing name acting as a lure for potential victims, *e.g.*, *faststorageonline.com*, *phonegamescatalog.com*, *wwwmp3archives.com*, etc. The entire set of domain names is hosted on few servers, heading to 5 IP addresses. Those IPs are not organized in a structured CDN and do not belong to the same subnet, suggesting for the usage of infected machines scattered in the network.

The right part of Fig. 8 includes some benign objects. While those may seem false positives, this is not the case. Looking closer at the top of Fig. 8, it is easy to realize that contacted websites host services aimed at the discovery of the public IP address of the host. Such behavior is coherent considering the intent of the malware we are facing. Being a backdoor trojan, the infected client has to be reachable by the cyber-criminals, but connectivity issues, *e.g.*, hosts behind NATs or firewalls, might preclude the reachability of the victim.

Looking at the bottom-right part of the CG, we observe that the malware is checking Internet connectivity by visiting the *www.google.com* homepage, another test run by the malware to gather connectivity properties of the victim.

B. Downloader.Dromedian Communication

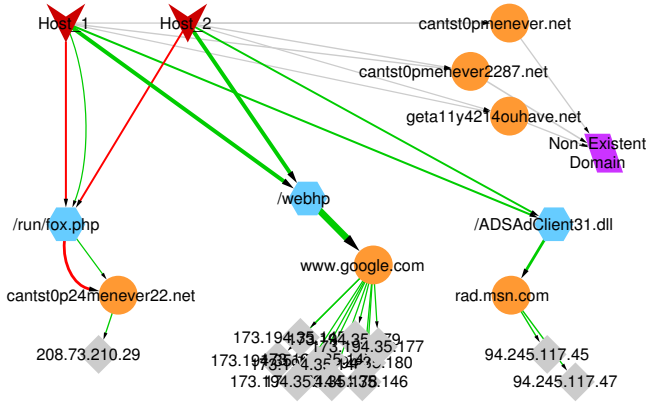


Fig. 9. CG for Downloader.Dromedian Communication trojan horse. Notice the presence of legitimate nodes being contacted by malware.

Downloader.Dromedan is a trojan horse that runs silently on the victim's host, downloading and putting in execution additional threats. This malware connects to remote malicious infrastructures and C&C networks in order to fetch and execute additional malware and potentially unwanted programs (PUP). In most cases, it causes the redirection of ordinary web surfing traffic to malicious websites through the installation of toolbars in web browsers. Such toolbars force the user to visit certain contents in order to generate profit for malicious attackers.

Fig. 9 shows the CG of the malicious event *cantst0p24menever22.net/run/fox.php*, which triggers the alarm related to the Downloader.Dromedian in our reference IDS. Two clients are infected and, in addition to the seed (red arrows), they both connect to *www.google.com/webhp* repeatedly. While the Google webpage is not malicious per se, it is caused to frequently appear due to PUPs related to Conduit search hijacker. Several unwanted toolbars like Delta Search Toolbar, Social Search Toolbar, and Internet Helper Toolbar are related to Conduit search, and cause Google Webhp redirects by hijacking the browser settings. Looking at the top-right part of the CG, infected clients query three hostnames causing an error at the DNS resolver side, owing to the fact that the two exploited hosts try to connect to the remote malicious infrastructure. Our suspiciousness on such event is reinforced by the similarity of the queried hostnames with the successful one (*i.e.*, *cantst0p24menever22.net*), and by the presence of numbers interleaving characters with a random fashion, possibly trying to avoid blacklisting.

C. Mass Injection Website

The Mass Injection Website attack does not target the host of the victim directly, but leverages vulnerabilities of legitimate

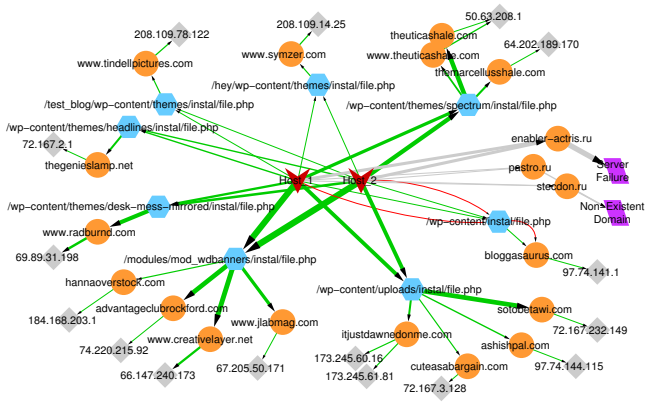


Fig. 10. CG for Mass Injection Website attack. Notice all URLs containing "wp-content", suggesting for a WordPress vulnerability being exploited my malicious adversaries. Victims are forced in a redirection chain through websites hosting Exploit Kits.

websites to inject malicious scripts and hidden *iframes*. In turn, users visiting compromised websites are victims of a redirection chain that forces them to visit third-party websites hosting Exploit Kits. Such Exploit Kits target potential vulnerabilities at the client side so that once the victim lands on the effective Exploit Kit, her machine gets infected as well.

Fig. 10 shows the network behavior of two hosts being victims of the Mass Injection exploitation. Our oracle considers malicious only the seed *bloggasaurus.com/wp-content/instal/file.php* but our CG is populated with other URLs all terminating with */instal/file.php*. Interestingly, many of those URLs also include the substring *wp-content*, suggesting for an exploited vulnerability in the WordPress blogging tool. Our suspiciousness is confirmed by the fact that both victim hosts exhibit the same identical behavior towards all the URLs depicted in the CG. The volume of issued requests is identical, and, considering the contacted URLs in a temporal sequence, it is possible to clearly spot repeated patterns over time, *i.e.*, both hosts visit other URLs in a deterministic way. Moreover, such web pages host different kinds of content, ranging from newscast to music festivals and healthcare. Thus, it is almost impossible that two users visit the same pages, the same number of times, in the same order. This confirms the ongoing automated webscan the user is not aware of.

VII. Related work

The increased ability of malware to spread and infect computers has led to vast amounts of research attempting to identify malware using the network traffic they generate. The work here presented is related to malware detection through graph-based approaches, and multi-protocol traffic correlation.

Graph-based Malware Detection: In [9], the authors build a bipartite graph consisting of domain names of failed DNS queries and host issuing such queries. The intuition is that host infected by the same malware usually query for the same (or similar) set of domain names. Similarly, [10] proposes to build a relationship graph based on DNS historical data.

In this context, suspicious networks are identified by means of two graph measures: graph density and eigenvector centrality. In [11], malicious hosts are detected using a semi-supervised, score-propagation algorithm that utilizes HTTP-communication graph and flow information. All these approaches restrict their efforts to a specific protocol to identify the suspicious graph entities. Alternatively, our system uses the data gathered from multiple protocols to create the CG on which the malware patterns are identified.

Multi-protocol Traffic Correlation: Many efforts have focused on the analysis of a single protocol to identify patterns displayed by malware. The popularity of HTTP has made it the preferred protocol for malware creators and, as such, the target for researchers to analyze and detect malware. [12] presents a system to identify malicious drive-by download activities by exposing the distribution networks necessary to distribute malware. Similarly, [13], [14] propose classifiers based on features from web domains and URLs. Systems that analyze the DNS protocol, usually look at failed DNS queries [15], [10], as this activity can lead to the existence of malware using domain generated algorithms (DGA). The problem with systems relying on a single protocol is their limited scope, as malware can switch among protocols, and the required semantic understanding of the particular protocol considered.

In comparison, a seminal work evaluating multiple protocols is [16], where the lifecycle of botnets is modeled according to a set of phases. An interesting approach is used in [17], [18], where network traffic is presented through generic packet information such as length sequences and encoding differences, allowing to represent the malware activity observed in different protocols. All of these multi-protocol approaches have the limitation of targeting specific type of malware. Our approach is instead general and encompasses different malicious activities. We propose a graph based approach that extracts the behavioral commonalities from multiple clients with a seed event in common.

VIII. Conclusions

We presented a system able to identify and correlate network events with malicious traffic. Starting from a seed, *i.e.*, an alarm raised by a reference IDS, our system leverages both spatial and temporal recurrence of events and frames them in a Network Connectivity Graph, that is a focused representation of the malicious activities over multiple network layers.

In contrast to other security tools, often providing atomic information on malicious attacks, our system delivers an enriched set of network activities related to malicious software running on victims' hosts. Specifically, it is capable of spotting interactions between malicious and legitimate infrastructures, increasing the knowledge on the incident.

We proved our approach is effective against different classes of malware, each showing peculiar behaviors and network patterns. In all cases, it provided a rich and interpretable characterization of the malicious activity, facilitating the understanding of malicious attacks and supporting the forensic activity of the security analyst.

References

[1] iMPERVA, "Assessing the effectiveness of antivirus solutions," http://www.imperva.com/docs/HII_Assessing_the_Effectiveness_of_Antivirus_Solutions.pdf, 2012.

[2] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, 2009, pp. 311–320.

[3] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proc. of WWW*, 2010.

[4] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 2nd ed. Addison-Wesley, 2013.

[5] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma, "Discovering all most specific sentences," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 2, pp. 140–174, 2003.

[6] F. Pan, G. Cong, A. K. Tung, J. Yang, and M. J. Zaki, "Carpenter: Finding closed patterns in long biological datasets," in *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2003, pp. 637–642.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - http/1.1," Tech. Rep., 2006.

[8] P. Mockapetris, "Domain names - concepts and facilities," Tech. Rep., 2003.

[9] N. Jiang, J. Cao, Y. Jin, L. Li, and Z.-L. Zhang, "Identifying Suspicious Activities Through DNS Failure Graph Analysis," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*. IEEE, 2010, pp. 144–153.

[10] Y. Nadji, M. Antonakakis, R. Perdisci, and W. Lee, "Connected colors: Unveiling the structure of criminal networks," in *Research in Attacks, Intrusions, and Defenses*. Springer, 2013, pp. 390–410.

[11] L. Liu, S. Saha, R. Torres, J. Xu, P.-N. Tan, A. Nucci, and M. Mellia, "Detecting Malicious Clients in ISP Networks Using HTTP Connectivity Graph and Flow Information," in *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*. IEEE, 2014, pp. 150–157.

[12] L. Invernizzi, S. Miskovic, R. Torres, S. Saha, S.-J. Lee, C. Kruegel, and G. Vigna, "Nazca: Detecting Malware Distribution in Large-Scale Networks," in *Proc. of the ISOC Network and Distributed System Security Symposium (NDSS '14)*, Feb 2014.

[13] P. K. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting malicious domains via graph inference," in *ESORICS 2014*. Springer, 2014, pp. 1–18.

[14] A. Le, A. Markopoulou, and M. Faloutsos, "PhishDef: URL names say it all," in *Proc. of the 30th IEEE International Conference on Computer Communications*, 2011, pp. 191–195.

[15] M. Antonakakis, R. Perdisci, Y. Nadji, N. V. II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *Proc. of USENIX Security Symposium*, 2012, pp. 491–506.

[16] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: detecting malware infection through IDS-driven dialog correlation," in *Proc. of the 16th USENIX Security Symposium*, 2007, pp. 12:1–12:16.

[17] C. J. Dietrich, C. Rossow, and N. Pohlmann, "Cocospot: Clustering and recognizing botnet command and control channels using traffic analysis," *Computer Networks*, vol. 57, no. 2, pp. 475–486, 2013.

[18] J. François, S. Wang, R. State, and T. Engel, "Bottrack: tracking botnets using netflow and pagerank," in *NETWORKING 2011*. Springer, 2011, pp. 1–14.