DynaMiner: Leveraging Offline Infection Analytics for On-the-Wire Malware Detection

Birhanu Eshete and V.N. Venkatakrishnan
University of Illinois at Chicago
{eshete5, venkat}@uic.edu

***Abstract*—Web-borne malware continues to be a major threat on the Web. At the core of malware infection are for-crime toolkits that exploit vulnerabilities in browsers and their extensions. When a victim host gets infected, the infection dynamics is often buried in benign traffic, which makes the task of inferring malicious behavior a non-trivial exercise.**

In this paper, we leverage web conversation graph analytics to tap into the rich dynamics of the interaction between a victim and malicious host(s) without the need for analyzing exploit payload. Based on insights derived from infection graph analytics, we formulate the malware detection challenge as a graph-analytics based learning problem. The key insight of our approach is the payload-agnostic abstraction and comprehensive analytics of malware infection dynamics pre-, during-, and post- infection. Our technique leverages 3 years of infection intelligence spanning 9 popular exploit kit families.

Our approach is implemented in a tool called DynaMiner and evaluated on infection and benign HTTP traffic. DynaMiner achieves a 97.3% true positive rate with false positive rate of 1.5%. Our forensic and live case studies suggest the effectiveness of comprehensive graph abstraction malware infection. In some instances, DynaMiner detected unknown malware 11 days earlier than existing AV engines.

***Index Terms*— malware detection, graph analytics, machine learning.**

I. Introduction

Malware infection via drive-by-downloads has become a day-to-day encounter in the current state of the Web. As cybercriminals step up their evasion tactics on content-based defense, the research community has responded with behavioral techniques (e.g., [5, 13, 19]) for malware analysis and detection. To this end, prior work explored approaches centered around the *malware download* phenomenon. In particular, on the analysis of binary reputation [21], exploit behavior [7, 11], redirection graphs [14, 25], malware download paths [16, 28], download graphs [9, 12], and botnet C&C dialogue [8, 15].

While these dimensions of malware defense are effective when considered in isolation, practice shows that a typical malware infection often exhibits *behavioral dynamics* which encapsulates a set of related interactions among actors in HTTP conversation. The interactions involve (i) *pre-download dynamics* —through a series of redirections, (ii) *payload download dynamics* —where malware payload gets downloaded to a victim system and, (iii) *post-download dynamics* —which involves connecting back to a C&C server of a cybercriminal. At the same time, multiple actors participate in a typical malware infection. These actors include compromised sites, traffic distribution services (TDSs), landing page servers, and exploit servers. Given this nature of malware infection dynamics and these crucial actors, we argue that a malware infection detection scheme that takes advantage of the interaction relationships between the various actors is desirable to (i) synthesize a comprehensive understanding of malware infection behavior, and (ii) develop more robust defense grounded to the inherent facets of malware infection.

In this paper, we present a novel system called DynaMiner that leverages *payload-agnostic* web conversation graph (WCG) analytics. DynaMiner taps into the rich dynamics of the interaction between a victim and malicious hosts to learn insights for building an effective malware defense. The key observation in DynaMiner is the *payload-agnostic*[1] and *comprehensive*[2] abstraction and analysis of malware infection dynamics. Our system reasons over a combination of payload download dynamics, pre-download redirections, and post-download dynamics of a web conversation, without the need to analyze exploit payload content. In a nutshell, DynaMiner leverages *offline analysis* of real-world malware infection to perform *on-the-wire* detection.

Motivated by the reality that the majority of malware use the Web as an entry point, DynaMiner focuses on malware infection over HTTP. To this end, given a stream of HTTP transactions, DynaMiner first abstracts the transactions into a WCG that captures *who* is related to whom (e.g., hosts, IP addresses), *what* relates the participants of the conversation (e.g., download, redirection), and *how* the conversation plays out temporally (e.g., delay between redirections). To annotate the WCG with artifacts of pre-download redirection, payload download, and post-download dynamics, DynaMiner employs a number of heuristics (Section III-B). On the annotated WCG, DynaMiner then performs graph analytics to infer payload-agnostic properties of the graph (Section IV). Finally, it determines whether the WCG is a malware infection or benign (Section V). DynaMiner relies on 37 payload-agnostic features, of which 27 are *novel features*. We note that among the novel features, 15 are in the top-20 highly distinguishing features for infection detection (Section VI-A).

In leveraging offline analysis of malware infection for on-the-wire detection, the major challenge is the inevitable benign background traffic that can potentially cause noise for statistical learning. We tackle this challenge via a careful choice of a learning algorithm that suits the variable nature of HTTP traffic

[1]**Payload agnostic**: does not require analysis of exploit payload (e.g., an executable binary) to perform detection. Hence, our malware detector will be resilient against morphed malware that evades content-based detection.

[2]**Comprehensive**: capturing pre-infection fingerprinting and redirection, payload download, and post-infection C&C call-back.

data. To this end, we employ the Ensemble Random Forest (ERF) [1] that is proved to perform reliably in the presence of noisy data. The robustness of ERF comes from the fact that sub-structures learned in a form of trees during training can capture distinct dynamics pertinent to redirection, download, and post-download sub-structures in the WCGs. In addition, during real-time detection, our system also weeds out obvious sources of noise (e.g., downloads from trusted software vendors).

On a ground truth data which spans 3 years of infection intelligence, the ERF classifier in DYNAMINER achieves an overall accuracy of 97.3% with a false positive rate of 1.5%. On a validation dataset disjoint with the ground truth, our classifier outperforms *VirusTotal* detectors by 11.5%. Furthermore, on a forensic and a live case study, our classifier outperforms *VirusTotal* detectors — suggesting the practical viability of DYNAMINER for offline and real-time malware detection. We also confirmed through a forensic case study, DYNAMINER detected an unknown malware 11 days ahead of *VirusTotal*.

This paper makes the following contributions:

- an empirical analysis that suggests a set of novel insights on malware pertinent to payload-agnostic analytics and detection,
- a payload-agnostic malware infection analysis technique based on analytics that taps into the rich dynamics of HTTP conversation graphs.
- a system that can be deployed at the network level for real-time malware detection.

The remainder of this paper is organized as follows. In Section II, we present a motivational study of malware infection. We introduce the formulation of our approach in Section III. In Section IV, we discuss the payload-agnostic features. Training and detection are discussed in Section V. We then present the results of our evaluation in Section VI. We discuss limitations and related work in Sections VII and VIII respectively. Section IX concludes the paper.

II. BACKGROUND: INFECTION DYNAMICS STUDY

To motivate our approach, in this section, we report insights from our analysis of *real malware infection episodes* captured on an enterprise network over a period of 3 years.

Analysis context and sanitization. To extract interesting artifacts from our study, we employ deep packet inspection to "connect the dots" in HTTP transactions of malware infection. Given our goal of examining malware infections from HTTP traffic, the traces we analyze may carry content not relevant to actual infection (e.g., binaries downloaded as part of system update). We therefore exclude non-malicious HTTP transactions and associated payloads using common-sense heuristics (e.g., binaries from known software vendors). As a second layer of sanity check, we pass all malicious payloads through *VirusTotal* [4] and verify their maliciousness via a conservative ensemble of detection systems.

Challenges in connecting the dots. Given an HTTP traffic capture, it might seem obvious to reconstruct the infection scenario. Unfortunately, it is more than a just mere stitching of HTTP transactions. With regards to recovering pre-download dynamics (e.g., redirections), one has to carefully comb through the HTTP traffic to pinpoint redirection footprints via `Referrer` headers on the client, `Location` headers on the malicious host, custom redirections such as `JavaScript` and `META` redirects. The most challenging situation is miscreants often use obfuscation of client-side code to conceal redirection chains. We therefore infer redirection and post-infection insights by deciphering obfuscated infection traffic to recover hidden redirection dynamics.

II-A. *Ground Truth Data and Collection Methodology*

Table I shows the ground truth dataset for this paper. Next, we describe the dataset and its collection context.

Infection Ground Truth. Our experimental insights in this paper are based on our analysis of 770 distinct PCAP traces of exploit kit malware infections. The dataset spans 3 years (06/2013 - 07/2016), covers 9 popular families of exploit kits, and involves 6 predominant exploit payloads —which, according to recent estimates [23], account for 2/3 of web-borne malware infection. Each trace represents a confirmed infection whereby a victim host interacts with an exploit kit site. Given the 3 years duration it spans, the presence of the most prevalent exploit kits, and the abundance of well-known exploit types, we believe that our dataset is representative of a realistic snapshot of the recent threat landscape with regards to web-based malware infections. All the samples we use for ground truth are obtained from `malware-traffic-analysis.net`.

Benign Ground Truth. To build a labelled ground truth of infection-free web surfing, we used multiple browsing sessions to capture 980 unique PCAP traces during 05/2015 - 05/2016. In all the browsing sessions, we keep multiple tabs open in the browser. As web search accounts for significant percentage of online activities, we collect traces of typical searching on the Google and Bing search engines. To simulate dynamics close to malware infection, we also click on links of a few of top search results. Visiting social networking sites, such as Facebook and Twitter, is also included in our data collection with common user operations such as clicking on links shared by friends and opening links in tweets. Another scenario we monitored is the use of web-based email services (e.g., Gmail, Yahoo! Mail) which includes the dynamics of downloading attachments of different file formats (e.g., PDF, executables, and MS office documents). To account for email-based exposure, we also captured dynamics on clicking links embedded in email messages. To simulate watching videos and clicking on advertisement links, we captured various scenarios of visits to `youtube.com`. Lastly, we also include traces from visits to randomly selected sites from Alexa Top 1 million sites. Given the scenarios we used for trace collection, our dataset is realistic enough to stress-test the discriminating power of features we discuss later in Section IV.

Next, we highlight empirical insights on malware infection focusing on exposure, graph properties, and HTTP header properties.

II-B. *Insights on Exposure*

In a typical malware infection, unsuspecting victims are lured to a seemingly harmless site via a range of exposures. Popular enticement methods include: links embedded in email messages, search results pointing to remote URLs, URLs shared by friends

Trace Family	No. of PCAPs	No. of Hosts			No. of Redirects			Unique Payload Counts					
		Min.	Max	Avg.	Min.	Max	Avg.	*.pdf	*.exe	*.jar	*.swf	*.crypt	*.js
Benign	980	2	34	3	0	2	0	60	30	3	0	0	138
Angler	253	2	74	6	0	18	1	0	80	133	0	64	1163
RIG	62	2	17	4	0	3	1	0	35	74	13	0	240
Nuclear	132	2	213	8	0	18	1	8	730	146	13	11	935
Magnitude	43	2	231	20	0	12	2	0	862	22	0	2	330
SweetOrange	33	2	90	8	0	6	1	0	310	22	0	0	227
FlashPack	29	2	15	5	0	8	2	0	556	35	0	0	159
Neutrino	40	2	30	6	0	14	2	0	45	31	5	6	217
Goon	19	2	90	9	0	30	2	0	78	15	10	0	71
Fiesta	89	2	182	7	0	3	1	21	226	72	63	0	414
Other Kits	70	2	68	4	0	5	1	1	420	13	4	0	271

TABLE I: Ground truth dataset. "No. of PCAPs" is the number of PCAP files which represent a single infection episode. The minimum for "No. of Hosts" is always 2 since the smallest conversation involves a client and one remote host. Columns "*.pdf", "*.exe", "*.jar", "*.swf", "*.crypt", and "*.js" refer to the count (per family) of payloads of the indicated file extensions. Since ransomware payloads come in more than 45 different file extensions, we used "*.crypt" to collectively refer to crypto-locker file types in our ground truth dataset. Note also that "*.js" shows the count of distinct JavaScript files (local and remote) all of which may not necessarily be malicious.

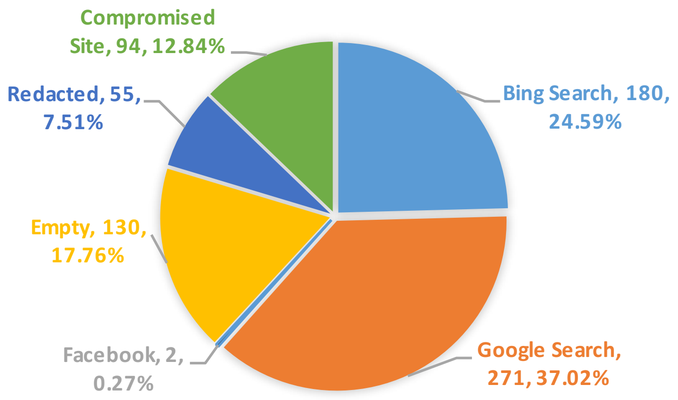

Fig. 1: Overall distribution of enticement strategies (Legend shows category, count, and percentage respectively).

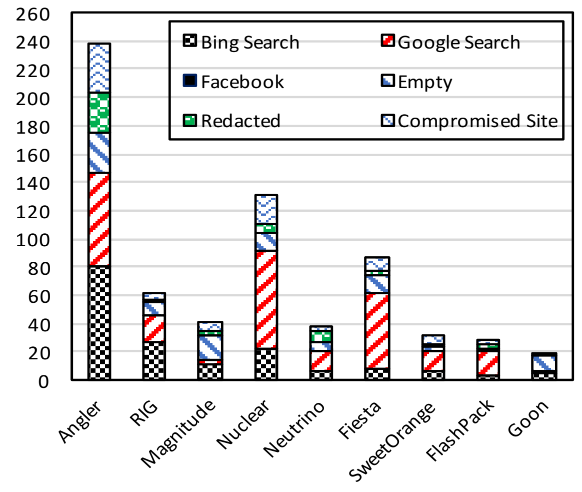

Fig. 2: Infection origin distributions for 9 exploit kits.

via social networking sites, ad banners with links to other URLs, and legitimate sites with links to other (possibly)malicious sites. A common aspect of all these enticement strategies is that the victim is tricked to visit a certain URL.

Knowing how the victim ended up visiting a malware site is crucial in establishing a case as to whether a malware infection is about to unfold. In this regard, we examine our ground truth dataset for clues about enticement strategies used in each infection. In particular, we infer from the infection traffic whether the victim was (i) visiting a search engine site (ii) visiting a social networking site and (iii) checking out a seemingly legitimate site.

Search engines drive infection exposure. Figure 1 shows the overall distribution of enticement strategies used in exploit kits. Search engines take the lion's share (62%) of the enticement strategies —Google with (37%) followed by Bing (with 25%). Given the market share of these two search engines, the distribution in Figure 1 is not surprising. A noteworthy observation, however, is that despite the popularity of social networking sites such as Facebook, less than 1% of the enticements originated from such sites. Interestingly, our findings are inline with a recent study [16] that found out search engines as the predominant origins of malware downloads. The same study also confirms our observation on the infrequent use of social networking sites to initiate malware infection.

In about 17.76% of the traces, we found the referrer fields to be empty — which we believe is due to intentional removal of referrer header values to conceal origin. The 7.51% traces for which the referrers have been redacted are sites that could have otherwise violated user privacy on victim hosts.

Weaponization of compromised sites. In 12.84% of the infection traces, enticements happened via compromised sites. This evidence shows the steady rise in the use of vulnerable sites for infection by cybercriminals. Recent studies (e.g., [6]) have also confirmed the rise of exploit kit infection via compromised sites. As can be seen from Figure 1, about 94 of the infections are initiated by visiting compromised sites. In quest for specifics, we matched the URI patterns of the 94 sites with URI patterns of default installations for commonly used CMSs (in particular, WordPress, Joomla, and Drupal). We found out that 56/94 infections have enticement URIs that match typical installations of WordPress sites. We also examined the longitudinal distribution of these sites in our infection ground truth dataset. To this end, we confirmed that while 15 of the 56 WordPress sites are scattered over a 2.5 years period (06/2013 - 01/2016), 41 out of the 56 appeared

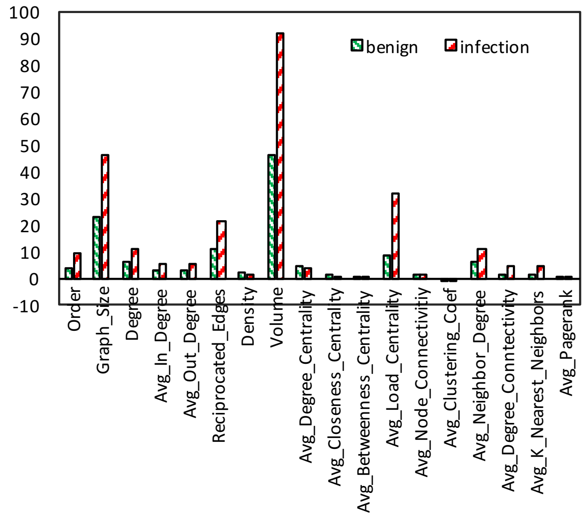

Fig. 3: Average measures for various graph properties.

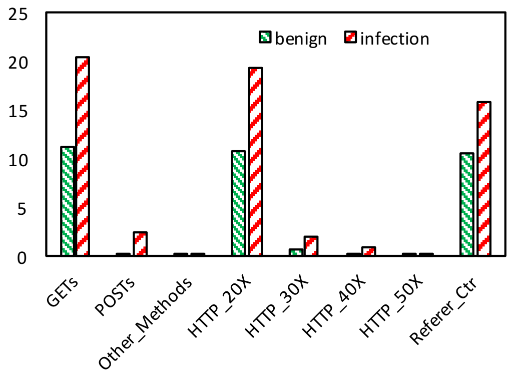

Fig. 4: Average counts for HTTP header elements.

within 90 days (01/2016 - 04/2016) —showing that such sites are increasingly utilized by cybercriminals.

Per-family distribution. Figure 2 shows the per-family distribution of enticement in exploit kit malware infection. Search engines and compromised sites consistently rank as the top enticement strategies used by the currently popular exploit kits. This similarity is attributed to the fact that exploit kit authors often employ similar black-hat SEO schemes or they could compromise similar set of vulnerable sites.

II-C. Insights on Graph Properties

Malware infection involves actors (i.e., various hosts) and relations (e.g., request, response, download, redirect) that link actors. Such a structure is essentially a (directed) graph. Inspired by this inherent graph structure, we studied graph properties of malware infection. A snapshot of our analysis is shown in Figure 3 which shows the average measurement of different graph properties. While the detailed insights from our graph abstraction are presented in Sections III and IV, here we briefly summarize the graph properties we examined in our dataset.

Basic Graph Properties. On average, infection graphs have higher number of nodes and edges. In addition, infection graphs tend to have higher diameter, degree, and volume.

Centrality. Except for load centrality, on average infection graphs have lower degree-centrality, closeness-centrality, and betweenness-centrality.

Connectedness. Due to high frequency of request-response and redirection relations, infection graphs on aggregate tend to have higher measures for degree-connectivity, neighbors, and page-rank.

II-D. Insights on HTTP Header Properties

Infections standout. While HTTP headers are interesting for pattern matching, in this work, we examine a more specialized set of header elements relevant to payload-agnostic dynamics analysis. Figure 4 captures the summary of our analysis on the infection dataset. Overall, we notice a contrasting statistical distribution of infection traces in comparison to benign traces. In particular, the average number of `GET` and `POST` requests, redirection chains, and HTTP 40X response codes is visibly higher (in some cases more than double) than the benign counterparts. A typical infection has at least 2 chains of redirection, while a typical infection-free trace has none. In some infection traces, we noticed exceptionally long redirection chains in some families of exploit kit infections (e.g., as long as 30 in the Goon exploit kit). Exploit kits such as Angler, Nuclear, and Neutrino are also known for their elaborate chains of redirections (see the "No. of Redirects" column in Table I).

Post-infection calling back attempts. In 708 of the 770 infection traces, we confirmed at least one attempt of "calling back home". In all of the cases, our analysis reveals that the hosts (IP addresses) to which post-download requests were initiated from the victim host have never been seen prior to or during the download dynamics. We found this characteristic to confirm our intuition behind analyzing the whole-spectrum of malware infection to engineer effective features for detection.

III. Web Conversation Graph Analytics

We now present our web conversation analytics focusing on graph abstraction, construction, and annotations.

Overview. Figure 5 shows a high-level overview of DynaMiner with two major stages, an *Offline Web Conversation Analytics* (Stage 1) and an *On-the-Wire Malware Detection* (Stage 2). In stage 1, DynaMiner analyzes web conversation traces (PCAPs) to construct a WCG and performs analytics on it. The result of the analytics is payload-agnostic features that are used to train a classifier. In stage 2, DynaMiner continually receives real-time HTTP request-response transactions and infers clue(s) of infection. Whenever it finds an *infection clue* (e.g., a redirection chain above a threshold followed by a download of a likely-malicious payload type), it builds a potential infection WCG around the clue(s). The constructed WCG is then passed to a feature extraction engine to extract the WCG properties to be evaluated by the classifier. Finally, DynaMiner gives a verdict (infection or benign) on the WCG. If DynaMiner deems a WCG infectious, the corresponding session is terminated. For each WCG deemed benign, DynaMiner keeps monitoring it as it grows, until the corresponding web session is terminated (usually by the user) or the WCG stops to grow.

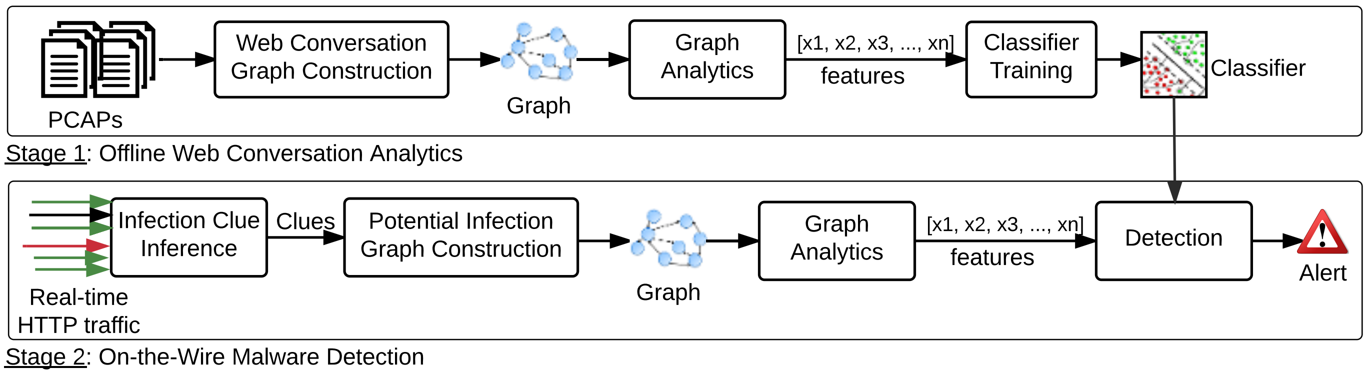

Fig. 5: High-level overview of DYNAMINER.

III-A. Graph Abstraction

We now describe how we build WCGs as they make the foundation of our analytics, learning, and detection. A WCG is a directed graph that captures the interaction between a host and one or more remote hosts. A node in the WCG represents a unique host, which can be a victim host, a malicious host, or a redirect intermediary host. A victim host is the one to which malware payload is downloaded (and executed). Alternatively, a victim host is also characterized by post-infection exfiltration to the attacker's server. A node is designated malicious if there is at least one download of malware payload from it to a victim host. An intermediary redirect host is a host that does nothing more than chaining hosts beginning from the victim host. An edge from host H_i to host H_j could mean (i) H_i sends a request (e.g., `GET`, `POST`) to H_j (ii) victim H_j receives response from host H_i (which could be payload download) or (iii) H_i is redirected to H_j.

Let us formally define our abstraction for the WCG. Suppose that H denotes the set of all hosts in a WCG dataset, R_q denotes the set of all requests, and R_s denotes the set of all responses, and R_r denotes the set of all redirects. Then the WCG G_i for a client H_i is defined as G_i = (Φ_i, Ψ_i, Σ_i, α, β) where:

- $\Phi_i \subseteq R_q i \times R_q i$ denotes a set of directed edges that correspond to requests initiated by H_i.
- $\Psi_i \subseteq R_s i \times R_s i$ denotes a set of directed edges that correspond to responses received by H_i.
- $\Sigma_i \subseteq R_r i \times R_r i$ denotes a set of directed edges that correspond to redirection relations in which H_i participates.
- α denotes a set of attributes about nodes and may include node type, IP address, and port number.
- β denotes a set of attributes about edges between nodes such as protocol, payload details (e.g., type, size), timestamp, and user-agent.

III-B. Graph Construction

Given an HTTP transaction stream, to construct a WCG we first extract unique hosts to populate the nodes. Next, we group the HTTP transactions into conversations between pairs of hosts. For each conversation pair, we identify one or more *request*, *response*, or *redirection* edges. We then annotate the nodes and the edges with relevant conversation attributes. Finally, we leverage the source-destination information to connect nodes via edges. The construction of the WCG begins by adding what we refer to as *origin node*. An origin node is a special node that indicates the enticement source. When the origin of a web conversation is known, the origin node takes the name of the referrer. Otherwise, it is marked "empty".

Real infection example. Figure 6 shows a simplified WCG we constructed from an infection trace of the *Angler* exploit kit. The "bing.com" node is the *referrer node* which is the origin that initiated to the malware site. Counting in the origin node, the WCG has 8 nodes and 31 edges (not all are shown in the graph). In the pre-download dynamics (blue dotted region), the victim is redirected (via a search engine) to a compromised site `A` which then leads it to `B` (an exploit kit landing page). An iframe redirection in `B` then leads the victim to the exploit kit server `C` that serves Flash exploits. The download dynamics (red dotted oval) ends with a download of a flash file from host `C`. Finally, the post-download dynamics (purple dotted oval) shows the malware contacting, via `POST` requests, remote hosts `D`, `E`, and `F`. These three hosts point to 3 unique IP addresses that serve the infamous *CryptoWall* ransomware. For the sake of brevity, we have not shown all the attributes of nodes and edges on the WCG in Figure 6. However, our approach annotates nodes and edges of a WCG with attributes that we use for computing features. In the following, we discuss annotation of nodes, edges, and the WCG as a whole.

III-C. Graph Annotations

Node-Level: The nodes in the WCG are annotated with the following attributes:

- *Basic attributes*: These include hostname and IP address.
- *URIs per host*: For each host, we count the unique number of URIs that have the hostname (IP address) of a host.
- *Payload summary*: This captures the count of different payload types that originate from or received by a node. The payload types include: known exploit types (e.g., *.jar, *.exe, *.pdf, *.xap, *.swf) and commonly exchanged payloads (e.g., images, HTML, JavaScript code, compressed files, text files). The summary also includes ransomware exploit payloads. Since ransomware comes with variable file extensions, we match file extensions in the web conversation against 45 distinct file extensions that we compiled from industry reports on ransomware [10].

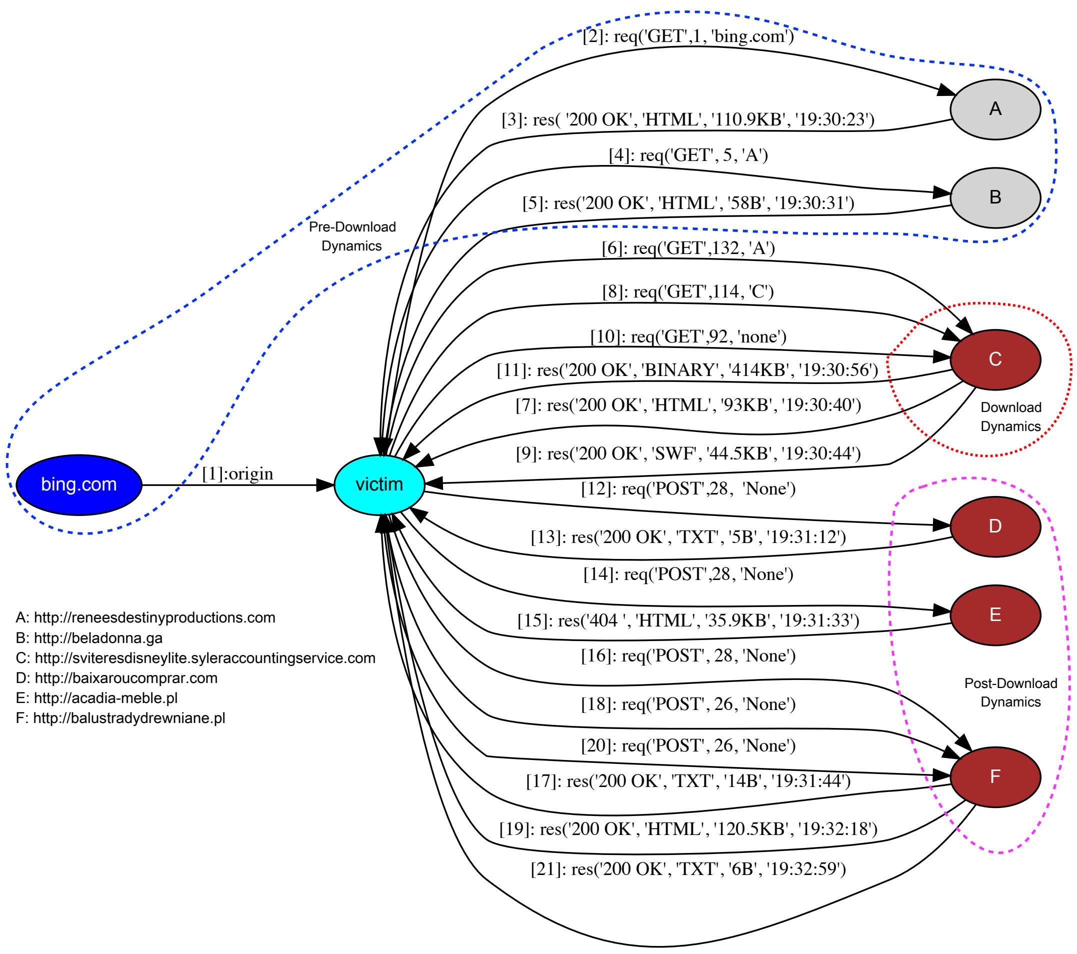
[2]: req('GET',1, 'bing.com')
[3]: res('200 OK', 'HTML', '110.9KB', '19:30:23')
[4]: req('GET', 5, 'A')
[5]: res('200 OK', 'HTML', '58B', '19:30:31')
[6]: req('GET',132, 'A')
[8]: req('GET',114, 'C')
[10]: req('GET',92, 'none')
[11]: res('200 OK', 'BINARY', '414KB', '19:30:56')
[7]: res('200 OK', 'HTML', '93KB', '19:30:40')
[9]: res('200 OK', 'SWF', '44.5KB', '19:30:44')
[12]: req('POST',28, 'None')
[13]: res('200 OK', 'TXT', '5B', '19:31:12')
[14]: req('POST',28, 'None')
[15]: res('404 ', 'HTML', '35.9KB', '19:31:33')
[16]: req('POST', 28, 'None')
[18]: req('POST', 26, 'None')
[20]: req('POST', 26, 'None')
[17]: res('200 OK', 'TXT', '14B', '19:31:44')
[19]: res('200 OK', 'HTML', '120.5KB', '19:32:18')
[21]: res('200 OK', 'TXT', '6B', '19:32:59')
[1]:origin
bing.com
victim
A
B
C
D
E
F
Pre-Download Dynamics
Download Dynamics
Post-Download Dynamics
A: http://reneesdestinyproductions.com
B: http://beladonna.ga
C: http://sviteresdisneylite.syleraccountingservice.com
D: http://baixaroucomprar.com
E: http://acadia-meble.pl
F: http://balustradydrewniane.pl

Fig. 6: Example WCG (Angler exploit kit captured on 12/21/2015). For "req" edges, the attributes shown are respectively: HTTP method, URI length, and referrer. We excluded the exact URI in requests to maximize the readability of the graph. For "res" edges, the attributes are respectively: HTTP response code, payload type, payload size in bytes, and timestamp. The victim's user agent is MSIE8.0 on Windows 7.

Edge-Level: The edges in the WCGs are annotated with the following properties:

- *Timestamp*: the time at which the event represented by the edge happens.
- *Conversation stage*: this property is assigned 0 for an edge that appears in pre-download stage, 1 for an edge in download stage, and 2 for edges in post-download stage. To determine the stage of an edge, we take a request-response pair and reason over a combination of: timestamp, HTTP method, response codes. For instance, if a request uses `GET` as a method, no known exploit payload is downloaded to a victim client prior to that, and the response code is 30x, then we assign the request-response pair to a pre-download stage. All the remaining request-response pairs are assigned to download stage after correlating their timestamp for inconsistencies. Note that the last 30x response code is the end of the pre-download stage and the beginning of the payload download stage. With the same token, the last 20x response whose content type is one of the known exploit payloads is considered the end of the payload download stage. For request-response pairs that use `POST` as a method to contact nodes from which no known exploit payload is downloaded, and the response code is either 200 or 40x, we assign them to a post-download stage.
- *HTTP method*: This captures which HTTP method is used in a request edge.
- *URI length*: The URI length of a request edge.
- *Response code*: The HTTP response code of a response edge.
- *Payload file type*: The payload type of a response edge.
- *Payload size*: The payload size in bytes of a response edge.

Graph-Level: In addition to node and edge attributes, we also compute aggregate properties that we use as foundations of features for classifying infectious WCGs. The following are annotations we add to a WCG.

- *Do not track*: It is assigned 1 if the "DNT" property is enabled. Otherwise, it is assigned 0.
- *X-Flash version*: If version of "X-Flash" is set, we capture its value. Otherwise, it is assigned 0.
- *Average payload counts*: The graph-level count of different payload types.
- *Average payload sizes*: The graph-level measure of payload sizes in bytes.
- *Total methods for requests*: The total count of `GET`, `POST`, and other request methods in the graph.
- *Total referrers*: The graph-level total count of requests for

which referrer is set.

- *Cross-domain redirection*: The count of redirections that happen between two different origins.
- *Redirection length*: The number of unique hops involved in a redirection chain.
- *TLD diversity*: The number of unique top-level domains involved in redirection.
- *Graph dynamics*: The graph-level properties including order, size, degree, density, volume, different measures of centrality and connectivity, neighborhood dynamics, and clustering coefficient.
- *Conversation duration*: The total duration in seconds of the conversation in the graph.
- *Average inter-transaction time*: this captures the average inter-arrival time of HTTP transactions in the graph.
- *Average delay between successive redirects*: this captures an estimate of the average time spent between two successive redirections. This property is useful in identifying infectious redirections from benign redirections since infections tend to have shorter delays between consecutive redirects.

III-D. Notes on Heuristics and Global Properties

Heuristics. In order to comprehensively capture behavior and relations in WCGs, we employ a number of heuristics. We infer pre-download redirections primarily from referrer and redirection header values of the HTTP transactions. However, redirection evidence is often embedded in obfuscated HTML or Javascript. We reverse engineer obfuscated JavaScript and HTML code to mine evidence the enriches redirection chains that the WCG follows before the first download event. Although we capture the pre-download dynamics in order to capture how redirection plays out, our observation suggests that, in some cases, one can find redirection chains in WCGs after downloads happen. In particular, while infectious WCGs perform redirections before dropping an exploit payload, we noticed instances of benign WCGs where redirections happen even after downloading a payload (e.g., when clicking on ad banners). For such cases, our pre-download redirection inference method is modified such that we take the sum of all redirections in a WCG.

Global properties. From the analytics on our ground truth dataset, we found out that there are 10 nodes on average per malware infection graph with a minimum of 2 nodes and a maximum of 404 nodes. The range of edges is between 2 and 1778 edges, with an average of 46 edges. As for lifetime, the graphs have an average lifetime of 123 seconds with a range between 0.5 and 4061 seconds.

IV. Payload-agnostic Features

The complete description of our features is shown in Table II. We group the features in to *high-level* aggregates, *graph-centric* properties, properties of *HTTP headers*, and *temporal* dynamics. Features for which the last column of Table II has a checkmark (✓) are *novel* features that we introduce (use for the first time) in this work. For features reused from prior work, we show citations to those works. Notice also that the last column shows an indirect comparison of our feature set against closely related work ([9, 12, 16, 25]). While the descriptions in Table II are self-explanatory, in the rest of this section we provide more context on graph properties and HTTP header features. Note that the reference dataset for all the distributions we present in this section is the ground truth data in Table I.

IV-A. Graph Features (f_7 - f_{25})

Our focus on graph dynamics is motivated by the downloader graph analytics in [12] and redirection graph analysis in [25]. While [12] also uses diameter, density, and clustering coefficient of downloader graphs, our WCG abstraction semantically differs from [12] in three respects. Firstly, we use payload as an edge attribute and the URL from which it is downloaded as a node. Differently from our technique, [12] uses a downloaded executable as a node and the source URL as an edge. Secondly, [12, 16] and [25], analyze downloader graph and redirection graph respectively. In DynaMiner, we rather combine the download graph with the redirection graph. In addition, we also include post-download graph dynamics whenever available. In fact, our study confirmed that 92% of the infection WCGs contain at least 1 post-download edge. Thirdly, we compute and show the effectiveness of *new* graph features compared to the state of the art techniques [12, 16, 21, 25]. Given the comprehensiveness of the graph abstraction we discussed in Section V, we believe the graph features reflect a much accurate dynamics of malware infections. The distributions of selected graph features shown in Figures 7-9 confirm the discriminating power of our graph features.

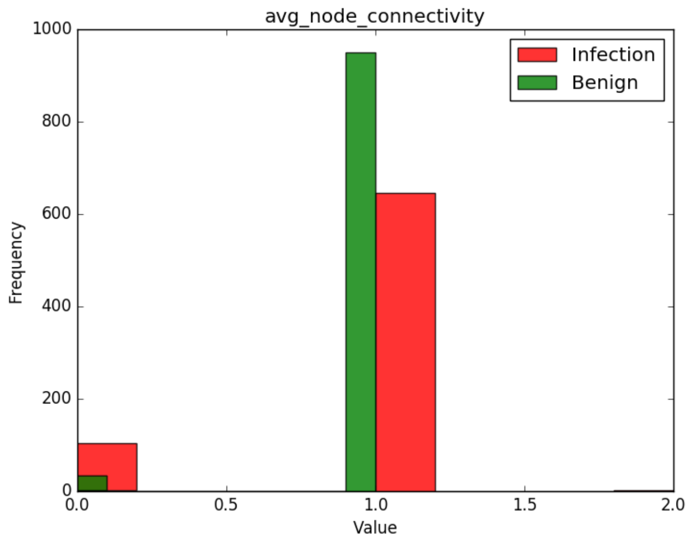

Fig. 7: Average node connectivity.

IV-B. Header Features (f_{26} - f_{35})

After a successful infection, malware often "calls back home" to exfiltrate valuable information from the victim host or to get more payloads for future missions. Alternatively, the malware can also use the infected host as a bot to conduct further attacks (e.g., spam campaigns). The key insight here is to leverage the malware infection dataset to pinpoint WCG structures that reveal a malware contacting a C&C server of the attacker. For

High-Level Features (HLFs)	**Brief Description**	**Novel**
$\mathbf{f_1}$: Origin	whether origin is known or not.	[25]
$\mathbf{f_2}$: X-Flash-Version	whether X-Flash version is set or not.	✓
$\mathbf{f_3}$: WCG-Size	size of a WCG.	[12]
$\mathbf{f_4}$: Conversation-Length	number of unique hosts involved in the WCG.	✓
$\mathbf{f_5}$: Avg-URIs-per-Host	average URIs per host computed as: $\frac{\sum URIs}{num\_hosts}$.	[9]
$\mathbf{f_6}$: Average-URI-Length	average URI length computed as: $\frac{\sum len(URIs)}{num\_URIs}$.	✓
Graph Features (GFs)		
$\mathbf{f_7}$: Order	number of nodes in a WCG.	[12, 25]
$\mathbf{f_8}$: Size	number of edges of a WCG.	[12]
$\mathbf{f_9}$: Degree	number of edges the node shares with other nodes in the graph.	✓
$\mathbf{f_{10}}$: Density	measure of how close the number of edges is to the maximum number of possible edges.	[12]
$\mathbf{f_{11}}$: Volume	sum of node degrees over all nodes in the graph.	✓
$\mathbf{f_{12}}$: Diameter	longest distance between any pair of nodes.	[12]
$\mathbf{f_{13}}$: Avg-In-Degree	average number of incoming edges to a node in the graph.	✓
$\mathbf{f_{14}}$: Avg-Out-Degree	average number of outgoing edges from a node in the graph.	✓
$\mathbf{f_{15}}$: Reciprocity	likelihood of nodes to be mutually linked.	✓
$\mathbf{f_{16}}$: Avg-Degree-Centrality	average of number of ties a node has.	✓
$\mathbf{f_{17}}$: Avg-Closeness-Centrality	average of the reciprocal of the sum of a node's distances from all other nodes.	✓
$\mathbf{f_{18}}$: Avg-Betweenness-Centrality	average number of shortest paths from all nodes to all others that pass through that node.	✓
$\mathbf{f_{19}}$: Avg-Load-Centrality	average of the fraction of all shortest paths that pass through a node.	✓
$\mathbf{f_{20}}$: Avg-Node-Centrality	average of the smallest number of nodes whose removal disconnects the graph.	✓
$\mathbf{f_{21}}$: Avg-Clustering-Coefficient	average of measure of the degree to which nodes in a graph tend to cluster together.	[12]
$\mathbf{f_{22}}$: Avg-Neighbor-Degree	average degree of neighbors of a node in the graph.	✓
$\mathbf{f_{23}}$: Avg-Degree-Connectivity	average degree for connected nodes.	✓
$\mathbf{f_{24}}$: Avg-K-Nearest-Neighbors	average number nodes at k-nodes distance from each node.	✓
$\mathbf{f_{25}}$: Avg-PageRank	average value for the importance measure of a node in the graph.	✓
Header Features (HFs)		
$\mathbf{f_{26}}$: `GET`s	total number of `GET` methods in a WCG.	✓
$\mathbf{f_{27}}$: `POST`s	total number of `GET` methods in a WCG.	✓
$\mathbf{f_{28}}$: Other-Methods	total number of less common methods (e.g., PUT, DELETE) in a WCG.	✓
$\mathbf{f_{29}}$: HTTP-10Xs	total number of informational responses in a WCG.	✓
$\mathbf{f_{30}}$: HTTP-20Xs	total number of success responses in a WCG.	✓
$\mathbf{f_{31}}$: HTTP-30Xs	total number of redirection responses in a WCG.	✓
$\mathbf{f_{32}}$: HTTP-40Xs	total number of client error responses in a WCG.	✓
$\mathbf{f_{33}}$: HTTP-50Xs	total number of server error responses in a WCG.	✓
$\mathbf{f_{34}}$: Referrer-Ctrs	total number of URIs which have referees set in a WCG.	[16, 25]
$\mathbf{f_{35}}$: No-Referrer-Ctrs	total number URIs for which referrer is empty in a WCG.	[16, 25]
Temporal Features (TFs)		
$\mathbf{f_{36}}$: Duration	average duration to access a single URI in a WCG session in seconds.	✓
$\mathbf{f_{37}}$: Avg-Inter-Transact-Time	average time (in seconds) between two consecutive web transactions.	✓

TABLE II: Feature types and brief explanations on how they are derived from WCGs.

instance, if we encounter `POST` requests leaving the victim after the completion of malware payload download, such an event is a strong evidence of post-download behavior. In this regard, HTTP headers carry statistical insights into the post-infection dynamics of a WCG. Our rationale for using such features from the WCGs is that HTTP methods such as `GET` and `POST`, and response codes are exhibit distinct distributions in benign and infectious WCGs. Thus, studying these properties reinforces the other graph properties in DYNAMINER.

V. CLASSIFIER TRAINING AND DETECTION

In this section, we provide an overview of learning a classifier and the detection method in DYNAMINER.

V-A. Classifier Training

Given the features we described in Section IV, we use an ensemble random forest (ERF) [1] to train our classifier. Our choice of ERF is driven by the nature of our WCGs and the underlying theory of the learning algorithm. The WCGs we build are likely to have sub-classes within the whole WCG due to distinct dynamics pertinent to redirection, download, and post-download sub-structures. In fact, a tree-based classifier such as a decision tree seems a natural choice for our WCG classification problem. However, decision trees tend to overfit training data that exhibits internal variability. Instead of taking the majority vote in the standard ERF, our implementation of the ERF combines classifiers by averaging their probabilistic prediction (which reduces variance). An ERF is therefore less prone to overfitting as compared to a decision tree.

V-B. On-the-Wire Detection

The intuition behind our on-the-wire detection is as follows. DYNAMINER sits at the edge of a network or as a web proxy to inspect individual web transactions from different hosts.

Infection clue inference. In the course of HTTP conversation, after each request-response transaction, the infection clue inference module of DYNAMINER determines on the presence of an infection clue. An infection clue, intuitively, is a likely indicator of malware infection. For our purpose, for instance,

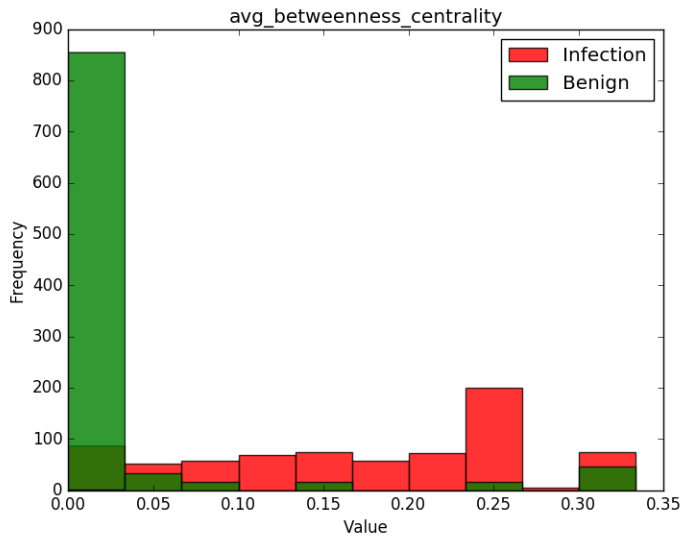

Fig. 8: Average betweenness centrality.

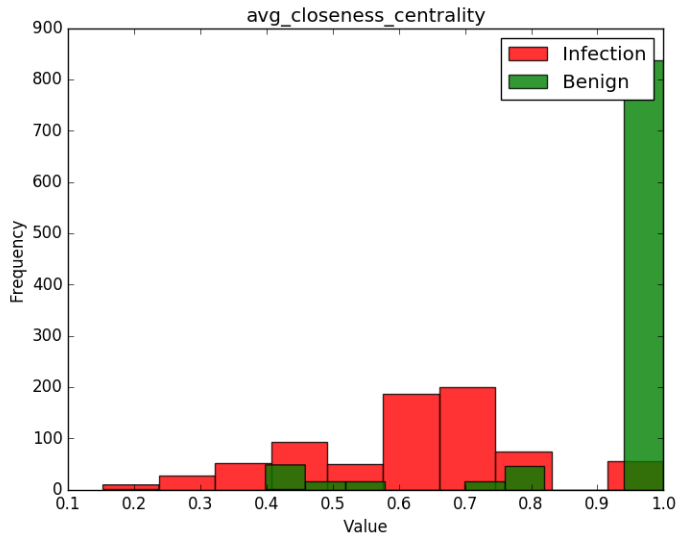

Fig. 9: Average closeness centrality.

an infection clue is flagged when a redirection chain of length $>= l$ is followed by a download of a file type t. The threshold for l and the download likelihood of the payload type x to be infectious are determined from a statistical analysis of the ground truth data.

Potential Infection WCG construction. Using the infection clue, DYNAMINER then goes back in time to construct a potential infection WCG using the WCG construction scheme we discussed earlier. To avoid mixup of HTTP transactions from multiple remote hosts, the session ID [18] of the download and the redirection chains that triggered the infection clue are used to guide the grouping of HTTP transactions that are likely to go into the same WCG. In case of multiple session IDs that a client is identified with when it interacts with multiple remote hosts, we use a heuristic that leverages referrer values and timestamps to cluster transactions into groups of sessions. Each cluster then becomes a WCG to watch as the various hosts communicate with remotes hots on the Web. Note that to reduce noise from benign HTTP traffic, we weed out HTTP transactions that originate from known vendors. For instance, assuming that these sources are trusted, we exclude traffic that involve downloads from online application stores / software repositories.

WCG classification and update. After the potential infection WCG is constructed, DYNAMINER extracts the features and queries the classifier. If the classifier predicts that a WCG is infectious, then DYNAMINER issues an alert. If the WCG is found benign, DYNAMINER keeps watching the WCG. In the course of watching WCGs, for each request-response transaction in the HTTP stream, DYNAMINER updates the respective WCG (again based on session IDs). Each update of a WCG then triggers feature extraction and invoking of the ERF classifier. DYNAMINER continues to watch each potential infection WCG until either the WCG stops growing or the session between the client and the remote host(s) is terminated.

VI. EVALUATION

We evaluate the effectiveness of DYNAMINER with regards to the ground truth dataset, an independent validation dataset, a forensic case study on recorded traffic, and a live case study in a mini-enterprise setting.

VI-A. *Features and Classifier Effectiveness*

Features. Table IV shows the ranking of the top-20 features in our ERF classifier on the ground truth dataset. For computing the ranks, we use the gain ratio metric with 10-fold cross validation. This metric is known for reducing bias towards multi-valued features in its criteria for selecting features. In favor of our claim about *capturing comprehensive dynamics*, graph-centric features make it to 15 of the top-20 features —showing how useful the graph dynamics is in distinguishing benign and infection WCGs.

Classifier. We evaluated our ERF classifier (a) using 10-fold cross validation on the training dataset in Table I and (b) using a labeled independent test set (Table V). The training was ran by varying the number of trees (N_t) and number of features (N_f) to get the best balance between true positive and false positive rates. The best performance our EDF classifier is with $N_t = 20$ and $N_f = \log_2(NumFeatures) + 1$ over all the 37 features described on Table II. Figure 10 shows the ROC curve of the ERF classifier we use to perform independent test on a separate dataset (Section VI-B).

As can be seen from Table III, training the classifier using the HLFs, HFs, and TFs from Table II (by excluding graph features) achieves the lowest true positive rate (0.860) with the highest false positive rate of 0.304. On the other hand, using graph features alone, the true positive rate jumps to 0.978 and the false positive rate drops to 0.059. Note that when all the features are combined, the false positive rate clearly drops (from 0.059 to 0.015) while improving the true positive rate (from 0.958 to 0.973). This is consistent with our observation that malicious WCGs have distinct distribution of features as compared to benign WCGs (see Figures 7-9). Our manual verification of the trees generated by the ERF shows that, when combined with the other features, the graph features improve the classifier accuracy.

Features	TPR	FPR	F-score	ROC Area
All	0.973	0.015	0.972	0.978
GFs	0.958	0.059	0.954	0.928
HLFs+HFs+TFs	0.806	0.304	0.848	0.860

TABLE III: Impact of features on classifier accuracy.

Feature	Gain Ratio	Average Rank
Avg-inter-trans-time	0.484 $\pm$ 0.015	1 $\pm$ 0
Duration	0.454 $\pm$ 0.021	2 $\pm$ 0
Order	0.309 $\pm$ 0.011	4.3 $\pm$ 1.27
Avg-load-centrality	0.309 $\pm$ 0.011	5.6 $\pm$ 2.15
Avg-closeness-centrality	0.309 $\pm$ 0.011	5.9 $\pm$ 1.92
Avg-betweenness-centrality	0.309 $\pm$ 0.011	6.2 $\pm$ 2.14
Avg-pagerank	0.309 $\pm$ 0.011	6.8 $\pm$ 1.4
Avg-neighbor-degree	0.306 $\pm$ 0.011	9.5 $\pm$ 1.8
Avg-k-nearest-neighbor	0.306 $\pm$ 0.011	9.6 $\pm$ 1.2
Avg-degreee-connectivity	0.306 $\pm$ 0.011	10.7 $\pm$ 1.55
Avg-in-degree	0.29 $\pm$ 0.02	11.4 $\pm$ 2.87
Avg-out-degree	0.29 $\pm$ 0.02	11.6 $\pm$ 2.8
Convs-length	0.302 $\pm$ 0.01	12 $\pm$ 1.9
Reciprocated-edges	0.248 $\pm$ 0.051	14.4 $\pm$ 6.55
Graph-size	0.245 $\pm$ 0.026	16.1 $\pm$ 0.94
HTTP-20X	0.251 $\pm$ 0.044	16.1 $\pm$ 2.77
HTTP- GETs	0.225 $\pm$ 0.047	16.8 $\pm$ 3.22
Avg-clustering-coeff	0.255 $\pm$ 0.008	17 $\pm$ 1.18
Volume	0.245 $\pm$ 0.026	17.1 $\pm$ 0.94
Degree	0.209 $\pm$ 0.053	18 $\pm$ 5.02

TABLE IV: Feature rankings for the top-20 features.

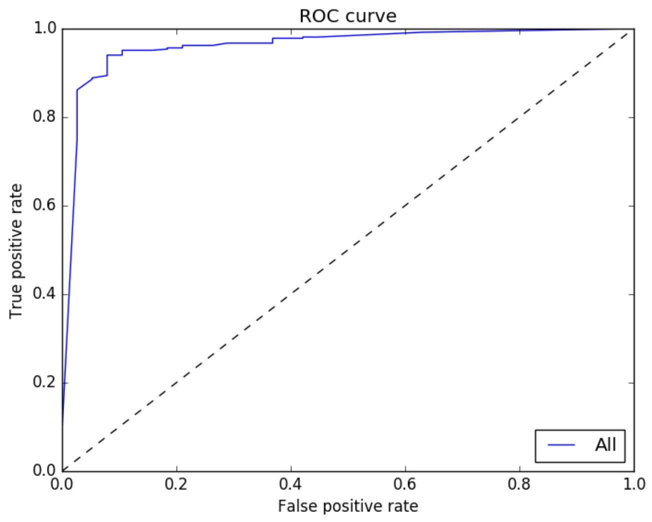

Fig. 10: ROC curve for ERF classifier on all features.

VI-B. Detection on a Separate Validation Set

To get insights about how our classifier would perform on samples it has never seen, on a dataset disjoint with the ground truth, we perform an experiment on a test set of 7489 malicious and 1500 benign WCGs. Note that the benign traces are collected the same way we collected the benign ground truth (as described in Section II). The infection samples are drawn from pre-verified *ThreatGlass* [3] malware infection intelligence. We submit the same test set to *VirusTotal* and compare the accuracy with the results of our classifier.

Overall effectiveness. As shown in Table V, DYNAMINER outperforms the current state of practice malware detectors by a visible margin (11.5% to be precise). Given the unique angle we explored to tackle malware infection, the results confirm that our payload-agnostic, graph-centric, and comprehensive analytics of infection dynamics is practically effective. Being a learning-based system, DYNAMINER faces challenges in terms of false signals. In the following, we analyze the reasons behind misclassifications we encountered in our tests.

Analysis of false positives. False positives are benign WCGs flagged as infection by our ERF classifier. As we indicated in Section II, legitimate download sources are excluded from our WCG construction. However, users often download benign content from unofficial sources. By cross-checking the hash values of downloaded payloads, we confirmed that 37 of the 49 false positives in DYNAMINER have download dynamics that connects to unofficial sites that deliver benign content. Since DYNAMINER does payload-agnostic analysis, based primarily on graph-centric properties, it flagged those WCGs as infections. In the remaining 12 cases of false positives, we noticed the client downloaded large binaries (in the range 246MB - 1.1GB) and a long list of videos downloaded from torrent sites. In addition to the size of downloads, the overall duration is also exceptionally long —which resulted in flagging the benign WCGs as infectious.

Analysis of false negatives. False negatives are infection WCGs flagged as benign by our ERF classifier. Through manual analysis of the 206 infection WCGs that DYNAMINER flagged as benign, we identified two major causes. Firstly, we noticed the absence of redirections but delivery of compressed malicious payload. In particular, we confirmed 89 cases of no redirections but compressed payload download. The second source of false negatives is the absence of post-download graph dynamics. This is not totally surprising since we have already confirmed that about 8% of our ground truth dataset does not contain post-download dynamics. Note that we could have avoided such misclassifications by excluding infection WCGs that have no post-download dynamics. We decided to keep them in the training set because of the rich dynamics they have on top of the pre-download and download stages.

VI-C. Case Study 1: Forensic Detection

Potentially infectious web session. To evaluate DYNAMINER on a web session that is potentially infectious, we selected a free live streaming service due the high likelihood of malware infections according to a recent large-scale study [20] on free streaming sites. Our case study was conducted on a PCAP capture of a user who watched the final game of EURO2016 Soccer Tournament on July 10 2016 on a free live streaming site (`http://atdhe.net`). On the browser, there were 18 distinct tabs open during the streaming session. The user occasionally switched to one of the tabs and clicked on links. In the course of the 90 minutes streaming, the service was interrupted 3 times and every time the page was reloaded, there was a JavaScript pop-up that asked the user to click on a download link that claims to fix a "out-of-date player". The user intentionally clicked on the links all of which led to other sites which appear to serve executables. The whole session resulted in the download of 32 payloads and the longest redirection chain was 4. The user's host conversed with 12 unique remote domain names during the whole session.

Detection and comparison with *VirusTotal*. We deployed DYNAMINER on a host and replayed the traffic using a local

System	# of WCGs Tested	Correctly Classified	False Positives	False Negatives
DYNAMINER	benign: 1500, infection:7489	benign =1471 (**98.1%**), infection=7283 (**97.38%**)	29	206
VirusTotal	benign: 1500, infection:7489	benign=1409 (94.0%), infection =6310 (84.3%)	91	1179*

TABLE V: Classifier performance on independent test data. * For 110 of the 1179 infection WCGs, *VirusTotal* timed-out.

web server. In the course of the replay, DYNAMINER issued 5 infection alerts on a total of $3,011$ HTTP transactions (i.e., request-response pairs) in the stream. The redirection threshold used for this evaluation was 3. Of the 5 alerts, 3 involved an Adobe Flash player executable download, while the remaining 2 were a JAR file and a PDF. We submitted all the 32 downloaded files to *VirusTotal* and at least 3 of the detectors in *VirusTotal* flagged as malicious 4 of the 5 payloads that DYNAMINER alerted as malicious. On the PDF payload, all the 56 *VirusTotal* detectors flagged it as clean. As a follow-up, after 11 days, we resubmitted to *VirusTotal* the same PDF that was flagged clean. Interestingly, 3 detectors flagged it as malicious for the first time. This shows that DYNAMINER can flag a malware that took *VirusTotal* detectors 11 days to pick up. Note that prior work [12] has also confirmed similar experience of *VirusTotal* detectors lagging an average of 9.25 days in flagging malware.

VI-D. Case Study 2: On-the-Wire Detection

Mini-enterprise setup. In this setup, DYNAMINER is deployed as a web proxy in a 3-host mini-enterprise network. The three hosts are a MacOS host with Google Chrome, a Ubuntu host with Firefox, and a Windows host with Internet Explorer. DYNAMINER intercepts all HTTP transactions from the three hosts and performs live analysis. This setup was run for 48 hours while the users were performing their routine web browsing. The Windows host is setup with COTS AV engine while the other two did not have one. The three hosts downloaded 62 files during the course of the case study and the average redirection length is 2 with the maximum redirection chain of 6.

Live alerts. Table VI summarizes the live case study in terms of the various payload types downloaded on each host, the maximum and average redirection chain length, and the breakdown of alerts issued by DYNAMINER. As shown in the last row, DYNAMINER issued 8 alerts (4 on the Windows host, 3 on the Linux Host, and 1 on the MacOS host). 3 of the 4 alerts on the Windows host are issued right after a download of an Adobe Flash payload while 1 is after JAR payload was downloaded. Note that the AV on the windows host, which was enabled during the case study, did not issue any alert on when these payloads were downloaded. On the Ubuntu host, all the 3 alerts are related to a download of JAR payloads, while the one on the MacOS host is a ".dmg" executable. We submitted all the 62 files to *VirusTotal* and it flagged (malicious) all the 8 that are relevant to the alerts by DYNAMINER. In addition, *VirusTotal* flagged as malicious 2 PDF files that were downloaded on the Windows host, but DYNAMINER issued no alert pertinent to the download of these PDF files.

False signal investigation. As a payload-agnostic system, if the maliciousness of an exploit payload manifests on its content or its behavior, DYNAMINER will likely flag it as benign. To understand why DYNAMINER could not issue alerts around the download of the 2 PDF files, we conducted a tool-supported investigation. In particular, we analyzed the PDF files with PDF Stream Dumper [2], which revealed Flash files embedded in the PDFs. We believe that *VirusTotal* deemed these files based on analysis results of its signature- and/or content-based malware detectors (3/56 malicious detections are all from AV engines for both PDF files).

Total	Windows Host	Ubuntu Host	MacOS Host
PDF	11	15	6
Executable	6	0	8
Flash	0	0	0
Silverlight	0	0	0
JAR	5	8	3
Avg. Redirection Chain	2	2	2
Max. Redirection Chain	6	4	3
DYNAMINER Alert	4	3	1

TABLE VI: Live detection summary on 48 hours of HTTP traffic streaming.

VII. DISCUSSION AND LIMITATIONS

We demonstrated how we can tap into the rich dynamics of a web conversation to learn distinguishing insights for payload-agnostic malware detection. DYNAMINER is driven by an assumption that the web conversation exhibits a certain degree of dynamism centered around download, redirections, and post-download dynamics. We also note that our system operates on unencrypted HTTP conversation. In what follows, we highlight evasion attempts that a determined adversary may employ to circumvent our ERF classifier.

Cloaked download dynamics. Although there is a trend in shifting to in-memory infections by exploit kits (e.g., Angler) [22], the infection trend we studied shows that file-based infections are by far the most predominant and consistent for the last 3 years. In fact, we found out that all the infection sessions we analyzed for training our classifier involved a download of an exploit payload for accomplishing an attack. We note that even if a WCG misses download dynamics, but has redirections and post-infection call-back, we believe it will still be classified as infectious due to the prediction score averaging by the ERF classifier which reduces the variance.

Cloaked redirection dynamics. Sometimes, infections may skip redirections to directly lead a victim to an exploit server. Although we have very few instances (11 in our dataset) of WCGs without redirects, it is a trade-off that attackers consider to complicate detection efforts. In theory though, an attacker may attempt to evade DYNAMINER by avoiding redirections and directly infecting the victim via drive-by or fileless infection. If she chooses fileless infection, DYNAMINER may not be able to detect as the resulting WCG will miss the most revealing features.

Post-download tweaks. Contrary to the common case of post-download dynamics we observed in our dataset, a malware author may cloak post-download by either (a) doing nothing after a successful infection or (b) delaying the call to the C&C server. For the former, it significantly limits the effectiveness of the attack in exfiltrating valuable information from the victim host. Hence, if at all it happens, it is rather in favor of the defender. In the latter case, DYNAMINER may miss the post-download dynamics as it is not trivial to learn the timing pattern of what plays out after infection.

VIII. RELATED WORK

We discuss related work focusing on graph-based malware detection and exploit kit analysis and detection.

Graph and tree-based infection abstraction. Kwon *et al.* [12] capture download activity on end hosts and explore the growth patterns of benign and malicious graphs to build a classifier. SpiderWeb [25] leverages redirections browsers go through to detect malicious web pages. BotHunter [8] synthesizes evidence of malware infection by tracking dialogue between internal host and external entities of a network. WebWittness [16] studies the origin of malware by tracing back the web paths followed by users who fall victim to malware downloads. In a follow-up work to WebWitness, Nelms *et al.* [17] present a systematic study on characterization and detection of social engineering attacks that lure users to download malware. In a malware distribution context, Nazca [9] identifies infectious downloads and installations in large scale networks. BotGrep [15] localizes botnet members based on the unique communication patterns arising from their overlay topologies used for command-and-control. Mekky *et al.* [14] developed a decision tree classifier based on HTTP redirection trees of browsing traces. CAMP [21] is an in-browser system for content-agnostic malware protection based on binary reputation. Amico [28] detects malicious downloads based on provenance of downloaded files in a windows host. DYNAMINER differs from this body of work in its richer abstraction and comprehensive analytics of WCGs.

Exploit kit malware analysis and detection. In [26], Taylor *et al.* detect exploit kit malware through tree similarity of HTTP flows. In a follow-up work [27], they leverage honeyclient-based detection of exploit kits on a network. Compared to [26, 27], our methodology in DYNAMINER differs in the payload-agnostic graph abstraction of infection dynamics. WebWinnow [7] leverages honey-clients to capture exploit kit behaviors for malicious URL detection. Kizzle [24] employs hierarchical clustering of unpacked malicious JavaScript code to generate exploit kit signatures. DYNAMINER complements these works by abstracting WCGS in a comprehensive manner.

IX. CONCLUSION

This paper presented DYNAMINER, a payload-agnostic system that performs web conversation graph analytics to uncover malware infection insights. By enriching the malware download phenomenon with pre-infection and post-infection dynamics, we demonstrate the effectiveness of graph features to distinguish malware infections in an evasion-resilient fashion. We evaluated DYNAMINER on multiple test sets of infection-free and infection WCGs and it achieved a TP rate of 97.3% with a FP rate of 1.5%. We also demonstrated the forensic and live detection capabilities of DYNAMINER with two case studies that demonstrate the effectiveness of our approach in detecting unknown malware days before *VirusTotal* detectors.

X. ACKNOWLEDGEMENTS

This material was supported in part by National Science Foundation grant CNS-1514472 and by a Defense Advanced Research Projects Agency / Air Force Research Lab contract FA8650-15-C-7561. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the NSF, DARPA, AFRL, or the US government.

REFERENCES

[1] Ensemble random forest classifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier/.
[2] Pdf stream dumper. https://github.com/dzzie/pdfstreamdumper/.
[3] Threatglass. http://www.threatglass.com/pages/.
[4] Virustotal. https://www.virustotal.com/.
[5] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *WWW*, 2010.
[6] Brad Duncan. How the EITest Campaign's Path to Angler EK Evolved Over Time. http://researchcenter.paloaltonetworks.com/tag/angler-exploit-kit/, 03 2016.
[7] Birhanu Eshete and V. N. Venkatakrishnan. Webwinnow: Leveraging exploit kit workflows to detect malicious urls. In *ACM CODASPY*, pages 305–312, 2014.
[8] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation. In *USENIX SEC*, 2007.
[9] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. Nazca: Detecting Malware Distribution in Large-Scale Networks. In *ISOC NDSS*, 2014.
[10] Jaymesned. List of ransomware extensions and known ransom files created by Crypto malware. https://www.reddit.com/r/sysadmin/comments/46361k/list_of_ransomware_extensions_and_known_ransom/, 03 2016.
[11] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *USENIX SEC*, 2013.
[12] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitras. The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics. In *ACM CCS*, 2015.
[13] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. Blade: An attack-agnostic approach for preventing drive-by malware infections. In *ACM CSS*, 2010.
[14] Hesham Mekky, Ruben Torres, Zhi-Li Zhang, Sabyasachi Saha, and Antonio Nucci. Detecting malicious HTTP Redirections Using Trees of User Browsing Activity. In *IEEE INFOCOM*, 2014.
[15] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. BotGrep: Finding P2P Bots with Structured Graph Analysis. In *USENIX SEC*, 2010.
[16] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths. In *USENIX SEC*, 2015.
[17] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Towards Measuring and Mitigating Social Engineering Software Download Attacks. In *USENIX SEC*, 2016.
[18] Phillip M. Hallam-Baker and Dan Connolly. Session identification uri. https://www.w3.org/TR/WD-session-id/.
[19] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iframes point to us. In *USENIX SEC*, 2008.
[20] M. Zubair Rafique, Tom van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *ISOC NDSS*, 2016.
[21] Moheeb Abu Rajab, Lucas Ballard, Noe Lutz, Panayiotis Mavrommatis, and Niels Provos. CAMP: Content-Agnostic Malware Protection. In *ISOC NDSS*, 2013.
[22] Jerome Segura. Fileless infections from exploit kit: An overview. https://blog.malwarebytes.org/exploits-2/2014/09/fileless-infections-from-exploit-kit-an-overview/, 09 2014.
[23] Jerome Segura. Exploit Kits: A Fast Growing Threat. https://blog.malwarebytes.org/101/2015/01/exploit-kits-a-fast-growing-threat/, 01 2016.
[24] Ben Stock, Benjamin Livshits, and Benjamin Zorn. Kizzle: A Signature Compiler for Exploit Kits. Technical report, Microsoft Research, 02 2015.
[25] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging Surfing Crowds to Detect Malicious Web Pages. In *ACM CCS*, 2013.
[26] Teryl Taylor, Xin Hu, Ting Wang, Jiyong Jang, Marc Ph Stoecklin, Fabian Monrose, and Reiner Sailer. Detecting Malicious Exploit Kits Using Tree-based Similarity Searches. In *ACM CODASPY*, 2016.
[27] Teryl Taylor, Kevin Z. Snow, Nathan Otterness, and Fabian Monrose. Cache, Trigger, Impersonate: Enabling Context-Sensitive Honeyclient Analysis On-the-Wire. In *ISOC NDSS*, 2016.
[28] Phani Vadrevu, Babak Rahbarinia, Roberto Perdisci, Kang Li, and Manos Antonakakis. Measuring and detecting malware downloads in live network traffic. In *ESORICS*, 2013.