IDS Alert Correlation in the Wild with EDGe

Elias Raftopoulos*, Xenofontas Dimitropoulos*
*ETH Zurich
{rilias, fontas}@tik.ee.ethz.ch

***Abstract*—Intrusion detection systems (IDSs) produce a large number of alerts, which overwhelm their operators, e.g., a deployment of the popular Snort IDS in the campus network of ETH Zurich (which includes more than 40 thousand hosts) produces on average 3 million alerts per day. In this paper, we introduce an IDS alert correlator, which we call Extrusion Detection Guard (EDGe), to detect infected hosts within a monitored network from IDS alerts. EDGe detects several malware that exhibit a multi-stage behavior and it can identify the family and even variant of certain malware, which helps to remediate and prioritize incidents. Our validation based on manual real-time analysis of a sample of detected incidents shows that only 15% of the detected infections are false positives. In addition, we compare EDGe with a state-of-the-art previous work and show that EDGe finds 60% more infections and has a lower number of false positives. A large part of this paper focuses on characterizing 4,358 infections (13.4 new infections per day) detected with EDGe from a unique dataset of 832 million IDS alerts collected from an operational network over a period of 9 months. Our characterization shows that infections exhibit spatial correlations and attract many further inbound attacks. Moreover, we investigate attack heavy hitters and show that client infections are significantly more bursty compared to server infections. Finally, we compare the alerts produced by different malware families and highlight key differences in their volume, aliveness, fanout, and severity.**

***Index Terms*—Intrusion Detection, Alert Correlation, Malware, Snort, Malware Measurements**

I. Introduction

As the Internet has become more pervasive during the last decade, malware has also evolved reaching a high level of sophistication. Tailoring effective countermeasures against malware threats has proven to be very hard for a number of reasons. First, the increasing complexity of software components makes it inherently difficult to eradicate all exploitable vulnerabilities. Second, modern malware increasingly target the end-user by leveraging various social engineering techniques that bypass intrusion prevention measures. Third, network compromise is a highly asymmetric threat requiring the exploitation of a single vulnerability, while multiple machines need to be properly patched and hardened to have any kind of security assurances. For these reasons, intrusion prevention on an operational network will always eventually fail. Consequently, security practitioners need effective tools to perform extrusion detection, i.e., methods to detect and monitor hosts within their network that are already infected by malware.

Existing extrusion detection approaches suffer from generating a very large number of false positives. For example, Snort in the academic network of ETH Zurich produces 3 million alerts per day on average. In this work, we introduce a novel extrusion detection system for the popular Snort IDS, which we call Extrusion Detection Guard (EDGe). EDGe uses an information theoretic measure, called J-Measure, to identify statistically significant temporal associations between a selected pool of alerts. In this manner, it detects malware that exhibit a recurring multi-stage behavior. In addition, EDGe can classify the family and variant of detected malware, which helps to prioritize and remediate infections. We evaluate a deployment of EDGe in an operational network and show that EDGe produces only 15% false positives. In addition, compared to a state-of-the-art IDS alert correlator, EDGe detects 60% more infections with fewer false positives.

Our second main contribution is the characterization of 4,358 infected hosts detected with EDGe over a period of 9 months in a large academic site with more than 40 thousand unique hosts. First, we characterize the volume, types and impact of infections we observe. We find for example that out of a total of 40 thousand distinct hosts, approximately 8% were infected at least once during their lifetime. Second, we characterize how infections correlate across time and space. We find that healthy hosts closer in terms of IP address distance to infected hosts are much more likely to become infected. In addition, our time series analysis shows that server infections are almost independent in time, while client infections are consistently more bursty. Finally, we compare key characteristics of different malware families. We find that trojans have the longest lifetime, followed by spyware, backdoors, and finally worms. In addition, we observe that infections have a strong impact on the number of outbound alerts generated by infected hosts, which is more prevalent for backdoors and worms.

In summary, in this work we make the following **contributions**:

1) **EDGe**: We introduce an IDS alert correlator for a deployment of the popular Snort platform in an operational network. In our validation, EDGe finds 60% more incidents and fewer false infections (only 15%) than the most related previous approach. Besides, EDGe can also identify the family and variant of a detected malware.
2) **Malware measurements**: We characterize 4,358 infected hosts detected with EDGe in an academic network and outline several novel insights about infections.

The remainder of this paper is structured as follows. In Section II we describe the IDS alert traces we used in our experiments. We introduce EDGe in Section III and describe our validation and comparison to a previous work in Section IV.

Then, we characterize detected infections in Section V. Finally, we review related work in Section VI, discuss our findings in Section VII and conclude our paper in Section VIII.

II. IDS Data

Our dataset is comprised of raw IDS alerts triggered in the main campus of ETH Zurich by a Snort [1] sensor, which is placed between the edge router of the campus and the network firewall. The sensor monitors all the upstream and downstream traffic of the campus. It uses the official Snort signature ruleset and the Emerging Threats (ET) ruleset [2], which are the two most commonly-used Snort rulesets. As of April 2011 the two rulesets have a total of 37,388 distinct signatures to detect malicious activities.

The collected alerts have the standard full Snort format. The fields we use are the unique rule identification number, the rule description, the timestamp that denotes when the alert was triggered, the IPs and ports of the communicating hosts, the default rule classification, which indicates the type of suspected malicious activity, and the rule priority, which provides a severity rank. The complete raw alerts as generated by Snort are sent every hour to our collection and archiving infrastructure.

The dataset is both large and rich. During the 9 month period we study, spanning from January 1st 2010 to September 22nd 2010, our monitoring ran on a 24/7 basis with only minor interruptions (corresponding to approximately 99% availability), capturing more than 832 million alerts from 91,512 thousand internal IPs. On an hourly basis we record on average more than 130 thousand alerts. The vast majority of these alerts have low priority and usually correspond to policy violations that are not directly related to security incidents. However, a significant number, approximately 50 million, consists of high priority alerts.

To identify unique host infections, we restrict our analysis to hosts with static IP addresses and exclude alerts from dynamic IP address ranges. We distinguish between dynamic and static subnets using a catalog maintained by our network administrators that documents each campus subnet. Additionally, this information enables us to find whether a subnet accommodates server or client machines. The excluded alerts originating from dynamic IP address ranges, correspond to 56% of the total active internal IPs in our data. Focusing on the 40,082 hosts that use static IP addresses is important as it enables us to track and characterize their behavior over time.

III. Methodology

A. Alert Bundling

The first challenge that we deal with is that security events often trigger spurts of very similar alerts. For example, certain types of port scanning targeting a range of destination ports will generate a large number of almost identical alerts that only differ in the destination port and timestamp fields. Besides, malware often change slightly their behavior in order to evade detection. Snort rulesets often include different signatures for each different malware version. When the malicious behavior is manifested, multiple versions of the same signature may be triggered in a very short time window. For example, we observe spurts of the alert *"ET DROP Known Bot C&C Server Traffic group (X)"* that only differ in the version number X. Such spurts of almost identical alerts are not desirable, since they defuse a single event into multiple segments. Alert bundling groups spurts of very similar alerts into a single aggregate alert. Compared to different forms of alerts aggregation, which have been studied in the literature [3], alert bundling aims at aggregating spurts of almost identical alerts instead of creating groups of much more diverse alerts that correspond to the same aggregate multi-stage incident. Alert bundling is useful as it reduces the amount of alerts that need to be processed and facilitates the statistical analysis of different events.

We perform alert bundling over three fields, source/destination ports and alert ID. We generalize the port fields from a numerical value to {*privileged*,*ephemeral*}, based on whether the port number is below or above 1024, respectively. We also group signature IDs that correspond to different flavors of the same malware into a single signature ID by ignoring the version number. We then merge alerts triggered within a short time window into a single aggregate alert. We preserve the timestamp of the first alert of the merged sequence. We select an aggregation window of 5 seconds. Our calibration showed that this is sufficient to substantially reduce the number of alerts, while further increasing this window had a negligible effect on the volume of alerts. Alert bundling reduced the total number of alerts in our data by 19%.

B. Alert Classification

Our dataset includes alerts triggered from several thousands unique rules. Snort rules are mainly community-contributed and follow a loose two-level classification scheme. Each rule is part of a ruleset, which groups related rules. For example, the ruleset `imap.rules` groups rules associated with the IMAP protocol. The second level of classification is based on the class field that is contained within each rule. The class field associates each rule with a unique class that provides information regarding the intended goal of an intrusion.

For our purposes, we find the default two-level classification scheme insufficient to extract alerts that relate to attacks and compromised hosts, which are the types of alerts we are interested in. The first shortcoming is that rules are grouped into rulesets based on different criteria. For example, some rulesets, like `imap.rules` and `voip.rules`, group rules based on the protocol or the application that is targeted, while some other rulesets, like `ddos.rules`, groups rules based on the type of the intrusion. A second problem is that rulesets often contain very diverse rules. For example `sql.rules` contains rules that range from accessing a database, which could correspond to benign behavior, to SQL worm propagation, which could indicate an infected host. Moreover, the classes associated with the classtype field are scarcely documented and in some cases ambiguous. In Table I we list the classes for alerts in the `sql.rules` file and provide

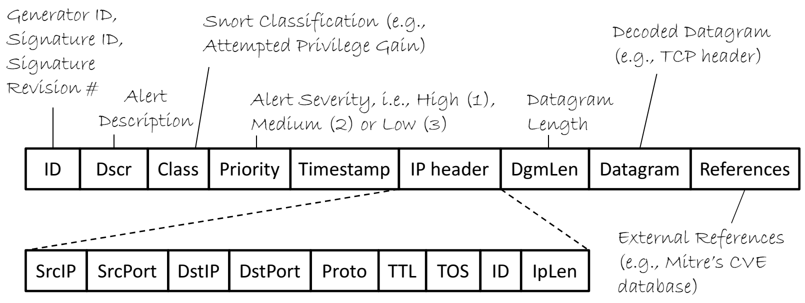

Generator ID, Signature ID, Signature Revision #
Alert Description
Snort Classification (e.g., Attempted Privilege Gain)
Alert Severity, i.e., High (1), Medium (2) or Low (3)
Datagram Length
Decoded Datagram (e.g., TCP header)
ID
Dscr
Class
Priority
Timestamp
IP header
DgmLen
Datagram
References
SrcIP
SrcPort
DstIP
DstPort
Proto
TTL
TOS
ID
IpLen
External References (e.g., Mitre's CVE database)

Fig. 1: Snort Alert Full Format

TABLE I: Classtype frequency of rules in `sql.rules`

#	Classification	Description
691	misc-activity	Miscellaneous activity
293	successful-recon-limited	Information leak
52	attempted-admin	Attempted administrator privilege gain
22	attempted-user	Attempted user privilege gain
4	unsuccessful-user	Unsuccessful user privilege gain
3	shellcode-detect	Executable code was detected
2	suspicious-login	An attempted login using a suspicious username was detected
2	misc-attack	Miscellaneous attack

the official documentation for each class. Some classes are quite intuitive, for example *Attempted administrator privilege gain* denotes that a privilege escalation attack took place. However, some other classes, like *Miscellaneous activity*, are quite cryptic and can result in loose classifications.

TABLE II: Rulesets and classtypes assigned to the *Compromised* class

Rulesets	Description
attack-responses.rules	Privilege escalation attempts
backdoor.rules	Trojan activity operating as Backdoor
ddos.rules	Bot initiating a DDoS attack
virus.rules	Malicious code attempting to propagate
emerging-botcc.rules	Bot-related trojan activity
emerging-compromised.rules	Attacks from blacklisted IPs
emerging-user_agents.rules	Data stealing malware
emerging-virus.rules	Malicious code attempting to propagate
Classtypes	**Description**
trojan-activity	A network Trojan was detected

To address this problem, we use a hierarchical approach to classify the rules included in our data into three classes, namely *Attacks*, *Compromised hosts*, and *Policy violations* (similarly to [4]). In the first step, we manually examined all the rulesets and identified the ones that clearly characterize an attack or a compromised host. With this step we were able to classify 72.5% of the total number of rules. For the remaining set of rules, we used the classtype field and identified 16 classes that can be clearly associated with attacks or compromised host activity. Finally, for the remaining 681 rules, we manually classified them by examining the details of the signature, the assigned default priority level, the exact byte sequence, and when possible we validated our results with information provided in security archives and bulletins [1], [5]. In Table II we summarize the rulesets and classtypes we used for our *Compromised* class.

Finally, the alerts that are not classified as attacks or compromised hosts, mostly occur when a user does not comply with a specific policy. Typically these alerts correspond to P2P, VoIP, and chat related rules. We discard these rules since they do not provide any useful information about infections and work only with alerts of the *Attack* and *Compromised* classes.

C. Malware Detection with EDGe

A naive approach for identifying infected hosts within a monitored network is to rely solely on *Attack* and *Compromised* alerts. However, the excessive amount of false positives, makes it very hard to have any level of confidence on a single alert. EDGe correlates multiple IDS alerts to find multi-stage infection patterns. In particular, it has the following design goals:

- ***Detect recurring multi-stage behavior***: Presently, malware developers bundle a plethora of features and capabilities to make their product more attractive. For example, malware attempt to redirect users to malicious websites and download additional trojans; they update, receive instructions, share confidential data, and participate in (D)DoS attacks or spamming campaigns; they attempt to propagate by scanning for exposed nodes and by exploiting vulnerabilities, etc. This means that most modern malware exhibit a multi-stage network footprint. Additionally, the multi-stage behavior is typically recurring. For example, a host infected with an SQL worm, will scan for vulnerable machines running an unpatched version of the Microsoft SQL server. Every time a target is found, the infected host will initiate a buffer overflow attack in order to exploit the vulnerability and eventually infect the victim. A Zeus trojan will attempt to inject fake HTML code every time the user visits an online bank page, in order to steal confidential data. The collected details will be then delivered to databases residing in a remote site. Based on these observations, EDGe attempts to reduce the number of IDS false positives by searching for malware that exhibit a recurring multistage behavior.

- ***Focus on extrusion detection***: EDGe aims at detecting infected hosts within an organization. It does not try to proactively prevent an infection or to detect other types of intrusions.
- ***Reduce false positives***: The number of false positives is involved in a fundamental trade-off with the sensitivity of the detector. Presently, IDSs suffer from a very large number of false positives. In this trade-off, we opt to make EDGe conservative, i.e., less sensitive, so that the inferences it produces include a small number of false positives. This also means that we may incur some false negatives, which we prefer than triggering a large number of false positives. In order to reduce the number of false positives, we engineer EDGe to combine multiple evidence.
- ***Keep it simple***: We opt to keep EDGe simple as parsimony provides a number of advantages: 1) inferences are interpretable and easier to trace and validate both for a scientist and an IDS operator; and 2) EDGe can efficiently analyze large archives of millions of IDS alerts.

EDGe Detection Heuristic: Our approach aims at detecting a recurring multi-stage footprint generated by infected hosts. A multi-stage footprint resolves into tuples of strongly correlated alerts. Such tuples capture different actions undertaken by an infected host that occur frequently and consistently over time, increasing our certainty that an actual infection has indeed occurred. We use an entropy-based information-theoretic criterion to detect correlated tuples of alerts.

Our input data is a time series of alerts, where each alert is identified by the following five fields: *<ID; SrcIP; DstIP; SrcPort; DstPort>*. We examine each internal host separately, discretize its sequence of alerts into time windows of length T, and mine for tuples of the type: if alert X occurs, then alert Y occurs within the time window T. We denote the above tuple with $X \Rightarrow Y$. Each tuple is associated with a frequency and a confidence, where the frequency is the normalized number of occurrences of the first alert X and the confidence is the fraction of occurrences that alert X is followed by alert Y within T. A well-known measure of tuple significance that combines these two basic metrics and enables to rank tuples is the *J-Measure* [6] (for an overview of tuple ranking methods refer to [7]):

$$\begin{aligned} \textit{J-Measure}(Y;X) = P(X)\big(P(Y|X)\log\frac{P(Y|X)}{P(Y)} + \\ P(\bar{Y}|X)\log\frac{P(\bar{Y}|X)}{P(\bar{Y})}\big), \end{aligned} \tag{1}$$

where $P(X)$ is the probability that alert X occurs; $P(Y)$ is the probability of at least one Y occurring at a randomly chosen window; $P(Y|X)$ is the probability that alert X is followed by at least one alert Y within T; and $\bar{Y}$ denotes the event that Y does not occur. Intuitively, the first term $P(X)$ captures the frequency of X, while the second term is the well-known cross-entropy and captures the average mutual information between the random variables X and Y. In this way, the *J-Measure* ranks tuples in a way that balances the trade-off between frequency and confidence.

The cross-entropy between X and Y drops when the two events tend to occur together. In particular, there are two cases when the corresponding entropy of Y drops. When X happens, Y always happens, or it doesn't ever happen. Clearly, the first case is of interest to us, since it reflects the probability of the two alerts co-occurring in a specific time window T. The second case is irrelevant since there will always be numerous alerts that do not occur when a specific alert happens, resulting in an inflated *J-Measure* value. Therefore, we only keep the left term of the cross-entropy to evaluate the significance of a tuple.

One desirable characteristic of the *J-Measure* is its limiting properties. Its value ranges from 0, when random variables X and Y are independent, to $\frac{1}{P(Y)}$, when they are completely dependent, which facilitates the process of defining a threshold above which tuples are considered strongly correlated. An internal host that produces at least one strongly correlated tuple is detected infected. We fine-tune the threshold to $\frac{0.85}{P(Y)}$ using validated infections from our most reliable source, which is security tickets about infected and remediated systems by our security group. We also evaluate the sensitivity of EDGe to alternative thresholds and find that $\frac{0.85}{P(Y)}$ realizes a good balance. Due to space limitations, we refer readers interested in our sensitivy analysis to our previous work [8]. From the set of correlated tuples we can easily extract the infection timestamps. For each tuple $X \Rightarrow Y$, if there is no other tuple $Z \Rightarrow W$ involving the same internal host within a time window $T_{reinfect}$, then this is a new infection at the timestamp of alert X. Otherwise, this is an ongoing infection and we ignore the corresponding tuple.

In Algorithm 1 we show the pseudo-code of EDGe. Its complexity is $O(n^2)$, where n is the number of unique alerts triggered by an internal node during T. In our experiments n is quite low and on average equal to 3.1. To run EDGe on one day of data takes on average 19.3 minutes on a system running Debian Etch with a 2GHz Quad-Core AMD Opteron.

Parameter Tuning: For the window size T we conservatively select one hour, since most alerts related to the same infection in our data occur within minutes. Selecting a larger window has negligible impact on the results. Moreover, we consider that a host is re-infected if the host is active in our dataset, but for a period of $T_{reinfect}$ it is not detected as infected by EDGe. We set the $T_{reinfect}$ threshold to two weeks. We select this value in a conservative way based on two observations. Incidents identified and investigated in the past in our infrastructure suggest that the worst case delay required by our security group to fix a reported problem is approximately one week. This time frame covers the stages of threat identification, threat assessment, and remediation of

Algorithm 1 Pseudo-code of EDGe for detecting infections

```
Require: Set $L$ of alerts triggered by internal hosts
Ensure: Correlated tuples $S_i$ for internal node $i$
  for all internal nodes $i$ do
    for all hourly timebins $T_k$ do
      for all tuples ($A_i$, $B_i$) in $L$, triggered in $T_k$, where $A_i \neq B_i$ do
        if $A_i \Rightarrow B_i$ in candidate tuple set $R_i$ then
          $R_i$.UpdateTupleStats($A_i \Rightarrow B_i$);
        else
          $R_i$.AddTuple($A_i \Rightarrow B_i$);
        end if
      end for
    end for
    for all tuples $M_i \Rightarrow N_i$ in $R_i$ do
      if J-Measure($M_i \Rightarrow N_i$) > $J_{thresh}$ then
        $S_i$.AddTuple( $M_i \Rightarrow N_i$ );
      end if
    end for
  end for
```

Trojans	
Malware Variant	**Classified Snort Signature IDs**
FakeAV	2012627, 2010627, 2011912, 2012725
Monkif	2010071, 2008411, 2012612
Simbar	2009005
Torpig	2011365, 2010267, 2011894, 16693
Nervos	2802912, 2801671
Koutodoor	2804717
MacShield	2012959, 2012958, 2802929, 2802870
Kryptic	2801962, 2013121
Comotor	2011848

Backdoors	
Malware Variant	**Classified Snort Signature IDs**
Zeus	2010861, 2011827, 2008661, 2011827
Blackenergy	2007668, 2010886
Parabola	2007626, 2002384
Ransky	2002728
Avzhan	2002728
SpyEye	2012491, 2011857, 2010789
Bamital	2802173, 2012299
LibNut	2803032

Spyware	
Malware Variant	**Classified Snort Signature IDs**
AskSearch	2003494, 2012000, 2003492, 2008052
Gator	2003575
SslCrypt	2012862
HotBar	2802896, 2800945, 2801396
Gh0st	2010859
Spylog	2007649, 2008429
Yodao	2011123
QVod	2009785, 2014459
Zango	2003058, 8073
Playtech	2008365
Gamethief	2012736

Worms	
Malware Variant	**Classified Snort Signature IDs**
Storm	2007701
Koobface	2010150, 2014303, 19058, 2009156
Rimecud	2012739
Conficker	2008802, 2009024, 2009205, 2008739
Lizamoon	2010268
Palevo	2010268, 2001689, 2010493

TABLE III: Malware families, malware variants and associated signatures

the host either by completely removing the malware or by rebuilding the entire system. On the other hand it is known, that some malware infections stay dormant for predefined time periods or wait for an external command to trigger their behavior [9]. In this case, the host will be reported as benign by EDGe, since no network trace of malicious activity is being generated. However, after the initial stimulus and assuming that the malicious behavior has been manifested, it is highly unlikely that the malware will fall again into idle mode for a time period longer than $T_{reinfect}$ [10]. Out of the total 4,358 infections we find in our characterization, 7.4% are re-infections.

D. Malware Classification

We classify EDGe-detected malware into a two-level taxonomy, which is based on our manual validation. The classification is useful for prioritizing and identifying malware, which facilitates forensics investigation and remediation. In addition, we use it to compare different classes of malware in Section V. We first classify EDGe malware based on their goal and propagation method into four families, namely *trojans*, *spyware*, *backdoors/bots*, and *worms*:

Spyware are otherwise useful pieces of software that are bundled with hidden fraudulent activity. Typically, they attempt to harvest user confidential data such as passwords, registration details, e-mail contacts, visited domains, cookies, or keystrokes. In some cases the unsolicited activity is stated in the license agreement, and user's consent is required for the installation. Legally spyware fall in a grey zone since users have explicitly accepted the licence terms. However, in reality they exploit user negligence and lack of technical awareness and expertise.

Backdoors/Bots allow an external entity to remotely control an infected machine. Backdoors use an active vulnerability in order to exploit the victim and hook themselves to the OS. A persistent connection to the victim's machine provides full or partial access and control, allowing the attacker to execute arbitrary commands. The compromised machine is typically herded to a botnet that can be used by cybercriminals to perform targeted DoS attacks, instrument large-scale spamming campaigns, or simply be leased to third-parties.

Worms are self-replicating and propagating programs that attach themselves to processes or files making them carriers of a malicious behavior. They employ active scanning to build a set of candidate targets and subsequently attempt to exploit a predefined vulnerability. Worms by default do not provide a control channel for the infected machines. However, their propagation functionality can be added to trojans to built composite malware that are remotely administered and can automatically infect new hosts.

Trojans masquerade as benign programs providing a seemingly useful functionality to the user, but clandestinely perform illegal actions. In contrast to backdoors, the user typically consents to the installation of the malicious software module. Trojans propagate using drive-by-downloads, javascript exploits, and simple social engineering techniques such as attaching malicious code to spam emails. Subsequently they can be used to perform a wide range of illicit actions such as leakage of confidential information, url-redirection to malicious domains typically to raise the hit count of these domains for advertising purposes, and downloading additional malware on the compromised systems. The latter method, called pay-per-install, is a paid service allowing bot herders to install their backdoors on large populations of nodes for a fixed price.

We use these classes because they reflect the four primary behaviors EDGe detected in our infrastructure. We note that in another context, a different taxonomy might be useful. For example, if one cares about spyware, a classification into adware, badware, scareware, and crimeware might be appropriate.

Furthermore, we manually analyzed 409 distinct signatures found in 54,789 tuples produced by EDGe during the 9 month tracing period. From this set we identified 75 signatures that are suitable for detecting specific malware variants. We only use signatures that incorporate payload based criteria that, based on our analysis, can be associated with the respective malware variants. We ignore signatures that attempt to identify blacklisted contacted domains or that detect generic behaviors such as malicious egg downloads, redirections, or scanning that could potentially be triggered by a broad set of malware.

For example Simbar, which is one of the most prominent trojan infections in our infrastructure, attempts to leak sensitive information about the HTTP objects that are susceptible to ActiveX Exploitation attacks by overloading the User-Agent string of the HTTP header. This is done by setting the value of the User-Agent field equal to the string *SIMBAR=value*, where *value* is the descriptor of the target ActiveX objects. The corresponding signature, which we match to the Simbar malware uses the following content rules to detect an infection:

```
content:"User-Agent|3a|"; content:"SIMBAR=";
pcre:"/User-Agent\:[^\n]+\;\sSIMBAR=/H";
```

When the signature of an alert in a tuple reveals the malware variant that triggered it, then we associate the corresponding host with the respective variant. In the example shown above, the detected payload sequence shows that this signature was triggered by a Simbar related activity, therefore, we associate it with Simbar infections. For each family, in Table III we list the corresponding malware variants and the associated set of alerts.

Finally, we exploit the descriptor provided by the signatures. For example, the signature with descriptor *"ET DROP Known Bot C&C Server Traffic UDP"* is associated with the Bot family, whereas *"ET TROJAN Generic Trojan Checkin"* is related to Trojan related infections. In this way, we tagged 94 additional signatures with a malware family label. For these cases, we can determine the malware family although we cannot identify the corresponding variant. This approach can lead to misclassifications in cases where this behavior is manifested by a different type of malware. For, example the alert *"ET TROJAN Generic Trojan Checkin"*, which we associate to the trojan family, could be triggered by backdoors checking whether their controller is alive. However, the classification using alert descriptors affects only 16% of the classified incidents in our analysis in Section V.

For validation, we compare to malware classifications of two well-known publicly available databases [11], [12] and explain differences. In Table IV we list mismatches and the fraction of incidents they affect. Note, that our classification is based on the associated IDS alerts, whereas the classification of [11], [12] is based on the known behavior of malware. The number of infection incidents stemming from ambiguous malware classifications are 164, corresponding to only 4.8% of the total classified incidents presented in our analysis in Section V.

IV. VALIDATION AND COMPARISON TO PREVIOUS WORK

Remotely validating numerous detected infections in an operational network is very challenging. A first challenge is that typically no single tool or information source provides sufficient evidence to validate an incident. A second challenge is that the types of malicious behaviors we examine are diverse, ranging from multi-stage attack and worm propagation events to complex trojan and malware communication patterns. Our validation follows a two step process. Given a detected infection, we first extract related information from independent security sources: 1) we manually examine the signatures of the IDS alerts; 2) we use six independent blacklists; 3) we compute a reputation value based on the visited domains; and 4) we actively scan hosts using a combination of IP sweeps, NIC whois querying, TCP/UDP port scanning, *nmap*-based network reconnaissance and vulnerability scanning (using both *OpenVas* and *Nessus*).

We refer to the collected information as *evidence*. A collection of evidence about suspected infections is then passed in real-time (i.e., within a day of the first time an infection was detected) to a security expert. The expert manually correlates the expected behavior of the malware with the collected evidence. In addition, he exploits contextual information about the victim host (i.e., its server or client role, the installed OS and services, and the location of its subnet) to expedite the investigation. If all the evidence agree with the expected behavior, then a positive assessment is made, otherwise it is concluded that the infection could not be validated. We conservatively consider the latter a false positive. The evidence extraction along with case studies about the manual validation process are presented in detail in [8]. We further characterize the manual validation process in [13] and show how to automate it in [14].

The validation process is very demanding and time consuming for the analyst, therefore, we limit ourselves to a subset of the reported infections. Specifically, we validated 200 consecutive incidents that were reported by EDGe based on the analysis of 37 million raw alerts. The analyzed nodes are diverse spanning from servers to desktop PCs and wireless devices. The overall false positive rate is only 15%. Recall that in our input data, we observe on average 3 million alerts per day, which we believe include a large number of false positives. By reversing our bundling procedure we find that only 0.6% of our input alerts of the class *Attack* and *Compromised* are associated with an infection. EDGe helps focus

Variant (# of Incidents)	EDGe Classification	Threatexpert [11] Classification	Trendmicro [12] Classification	% of Infections Affected
Torpig (28)	Trojan	Trojan	Backdoor	0.82
MacShield (12)	Trojan	Spyware	Spyware	0.35
Zeus (8)	Backdoor	Backdoor	Trojan	0.23
SpyEye (9)	Backdoor	Trojan	Backdoor	0.26
Bamital (3)	Backdoor	Trojan	Trojan	0.08
QVod (93)	Spyware	Trojan	Trojan	2.73
Gh0st (4)	Spyware	Trojan	Spyware	0.11
Koobface (2)	Worm	Trojan	Worm	0.05
Rimecud (3)	Worm	Trojan	Trojan	0.08
Storm (2)	Worm	Trojan	Worm	0.05

TABLE IV: Ambiguous malware variant classifications and percentage of associated incidents

the attention of administrators to a small number of actionable cases that include substantially fewer false positives. The false positive rate for trojans, spyware, worms, and backdoors is 12.3%, 10.6%, 11%, and 35%, respectively.

Moreover, to understand better the strengths and limitations of our heuristic, we investigate the root causes of the observed false positives. The following cases were the source of most false positives.

DNS Requests. First, we find that DNS requests trigger signatures from the *Compromise* class when an IP address is blacklisted. We observe alerts with SIDs in the range [2500000:2500941], which correspond to backdoor activity. These cases increase the false positive rate of backdoors.

Skype Supernodes. Skype supernodes within our network generate alerts with IDs in the ranges [2406000:2406966] and [2500433:2500447]. Skype supernodes connect Skype clients by creating the Skype P2P overlay network. However, if a remote Skype user connecting to a local supernode is blacklisted, then Snort will trigger an alert. This is repeated whenever a Skype client attempts to initiate a communication.

Antivirus. Third, a specific antivirus program generates IDS alerts of the class *Compromise* when updating. The triggered signatures check for known patterns of malicious activity found on the payload of the transmitted packets. The updates of this antivirus contain the actual pattern that it attempts to detect in plain format.

Online Games. Finally, we observe that certain online games generate Snort alerts with IDs in the ranges [2003355:2003626] and [2510000:2510447]. In particular, the triggered signatures of browser-based games suggest an ongoing spyware activity. The reason is that the websites exhibit a behavior similar to clickbots, attempting to redirect the player to 3rd party, potentially malicious, websites for profit. In the case of stand-alone gaming applications, we observe that the client will tend to preserve multiple concurrent connections with several other players. Often a small set of these remote IPs are blacklisted.

It is possible to further increase EDGe's detection accuracy by incorporating contextual information regarding the underlying infrastructure. For, example a network administrator should be able to easily identify that incidents related to the DNS servers do not constitute actual infections, and filter them out. In our evaluation, we did not use such whitelisting information and therefore its performance can become even better in an operational environment.

A. Comparison with BotHunter

Bothunter [15] is the most related previous work to EDGe. It correlates IDS alerts to a predefined malware dialog model. Malware infections are modeled as loosely ordered IDS-alert sequences, triggered during communication of an internal host with several external entities. All malware share a set of underlying actions that occur during their lifetime consisting of reconnaissance, exploitation attempt, binary egg download and execution, command and control establishment, and propagation. This approach is similar to our work, due to the concept of an underlying malware lifecycle that triggers different types of alerts, while the infected host undergoes different infection stages. However, there are some critical differences regarding the methods used to capture the malware lifecycle.

Bothunter considers a more strict sequence of events that need to occur in order to raise an alert. EDGe on the other hand searches for strongly correlated tuples of any two events, rather than a strict sequence of multiple events. This way it is more robust in the absence of evidence for the intermediate stages of the malware's lifetime. Moreover, our sequencing is much more flexible, constructing tuples of alerts that co-occur within a fixed time-window, without specifying the respective order. In this way we allow scenarios, where the inbound exploitation occurs after the communication with the C&C. Although such scenarios may seem unconventional, they do occur in practice, for example in the case of backdoors allowing further exploitation of the victim host by additional badware, such as in the case of W32/Ransky. Finally, the criterion used to mine for significant events in the case of Bothunter does not take into account that frequently occurring events may incorrectly yield correlated sequences. This makes Bothunter more sensitive to alerts that get triggered very often, exhibiting abnormally high frequency, which often can be attributed to network/protocol misconfigurations or badly written signatures. EDGe uses the *J-Measure* to effectively balance the frequency and the confidence of alert tuples.

Bothunter is publicly available. We deployed version 1.7.2 of Bothunter using the latest signature set provided in [16]. Note, that Bothunter uses a custom ruleset, consisting of a selection of VRT and ET rules, bundled with few custom rules, accounting for 3,152 signatures in total. The ET and VRT rules are identical to the original ones, with an additional string appended to the alert description indicating the Bothunter event type. Since these rules are frequently updated, we augmented the current ruleset with 62 additional signatures from [16], which were used during our validation period and were subsequently removed.

We used Bothunter in offline batch processing mode, using as input trace the IDS data we collected during the validation period. We had to adapt the raw alerts, by appending the Bothunter event type to the alert descriptor, whenever the respective Bothunter signature was available. Since, we replay IDS alerts in Bothunter's input stream, the Statistical Payload Anomaly Detection Engine and the Statistical Scan Anomaly Detection Engine, which are custom Snort plugins operating on packet traces, were disabled. This way we can compare on common grounds the two IDS alert correlators.

In Table V we present the results of our comparison. We use as input data for both detectors our validation trace, consisting of 1 month of IDS data. Bothunter detected 60% fewer infections compared to our detector, namely 125 instead of 200. There was a considerable overlap on the reported incidents, since 89 infections were reported by both detectors. Bothunter generated 36 new infections that were not reported by EDGe, which we also validated.

Considering all incidents, our detector generates fewer false positives (15%) compared to Bothunter (18.4%). In addition, we take into account the respective malware family. Our approach performs better for trojans, spyware, and worms, with a false positive rate below 12.3% for all types, whereas Bothunter is consistently worse exhibiting false positive rates of 20.83%, 12.19% and 37.5%, respectively. The results are reversed in the case of backdoors, where Bothunter has a false positive rate of 17.8%, whereas EDGe exhibits a larger false positive rate of 35%. This picture reflects the underlying design assumptions of the two detectors, with Bothunter being more effective for infections that tend to undergo multiple stages and, therefore, generate long sequences of correlated alerts, whereas EDGe exhibits better performance in the case of infections that only generate evidence in few stages. In addition, *J-Measure* enables to detect correlated alerts with higher confidence.

Moreover, based on the validated inferences that were not reported by one detector, we assess the false negatives of the two schemes. Note, that the computed false negatives are lower bounds, since additional infections that are not detected by neither detector might exist. In Table V we see that the false negatives for Bothunter is particularly high for all infection types with the exception of backdoors. Several infections detected by EDGe are completely ignored by Bothunter, such as the trojans Monkif, and Nervos, the spyware Gator and the worm Koobface. On the other hand, Bothunter performs better in the case of backdoors, and manages to detect two additional variants, namely rBot and PhatBot, which are missed by EDGe.

To summarize, our method performs better in the general case taking into account all infection types, generating fewer false positives and finding 60% more infections. Bothunter, on the other hand excels in the case of botnet detection but performs poorly for the other malware families. The two detectors could be used in parallel to improve IDS-based malware detection.

V. Characterizing Infections

A. Volume, Types and Impact of Infections

We first find on average that on a daily basis from 10,124 active[1] hosts, we detect 15.8 new infections. Taking into account the 0.15 false positive rate of EDGe, this corresponds to a lower bound of 13.4 new infections per day. The vast majority of the infected hosts are client systems. Specifically, 91% of the total reported incidents affect clients, whereas we only see on average 1.48 new infections per day on servers. If we normalize these numbers based on the total number of active servers and clients in our infrastructure, we estimate that the lower bound for the probability of infection of an online server during a day is 0.0015, whereas the corresponding value for clients is 0.0031.

The lower probability of a server infection can be attributed to two causes. Firstly, these systems are heavily managed, closely monitored, and often have their OS hardened. This means that unnecessary services are disabled, which reduces their vulnerability "surface". Secondly, as we saw in Section IV most of the malware that we observe propagate using indirect methods (e.g. drive-by-downloads, phishing) that involve the user and exploit his negligence, rather than initiating direct attacks, like scanning or buffer overflow attacks.

Moreover, out of a total of 40 thousand distinct active IP addresses we observed during the 9-month period, approximately 8% exhibited signs of infections at least once during their lifetime, whereas the total number of infections (including nodes that were re-infected) was 4,358. The number of nodes exhibiting re-infections was 239, corresponding to less than 6% of the entire active population. The majority of the re-infected nodes were connected to highly dynamic subnets in our network, corresponding to student labs and recreation areas, which are not heavily monitored. These are mostly private laptops without administrative restrictions on the installed applications and services. Therefore, the attack vector of these machines is broader, which is reflected on the increased probability of reinfection.

Detected Malware Variants: Next, we provide insights into the malware families and the exact malware variants detected by EDGe based on the methodology of Section III-B. Out of the total of 4,358 infections we can classify based solely on

[1]An active host generates at least one IDS alert during the indicated period.

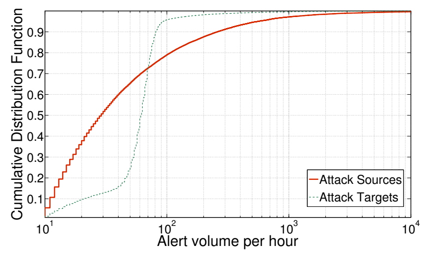
TABLE V: Comparison between EDGe and BotHunter

	Reported Incidents		Validated Incidents		Missed Incidents		False Positive Rate		False Negative Rate	
Detector	EDGe	BotHunter	EDGe	BotHunter	EDGe	BotHunter	EDGe	BotHunter	EDGe	BotHunter
Trojans	97	48	85	38	2	49	12.3	20.8	2.2	56.3
Spyware	66	41	59	36	10	33	10.6	12.2	14.4	47.8
Worms	9	8	8	5	3	6	11.0	37.5	27.2	54.5
Backdoors	28	28	18	23	9	4	35.0	17.8	28.0	14.81
Total	200	125	170	102	24	92	15.0	18.4	12.37	47.42

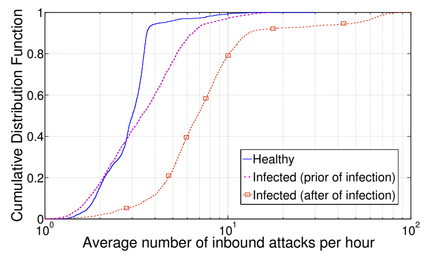
the tuples mined by EDGe 78% into the malware families of Section III-B. In addition, for 62% of the detected malware (2,712 incidents) we can identify the exact malware variant. In Table VI we present the prevalence of different malware families and variants detected by EDGe in our infrastructure. The Trojan family is dominated by the *Simbar* and *FakeAV* variants. In the case of Backdoors, the *Avzhan*, *Ransky* and *Parabola* variants account for the vast majority of the reported infections. Spyware on the other hand is the most popular family with several variants, including *Hotbar* and *AskSearch*, accounting for 311 and 722 incidents, respectively. Worms account for only 64 of the classified incidents and are dominated by the *Palevo* and *Conficker* variants.

Family	Variant	# Infections
Trojans	Simbar	252
	FakeAV	120
	Torpig	28
	Kryptic	15
	Nervos	13
	MacShield	12
	Monkif	7
	Comotor	5
	Koutodoor	3
Backdoors	Avzhan	80
	Ransky	33
	Parabola	21
	SpyEye	9
	Zeus	8
	LibNut	5
	Blackenergy	4
	Bamital	3
Spyware	AskSearch	722
	Zango	441
	HotBar	311
	Playtech	206
	Spylog	139
	Qvod	93
	Yodao	88
	SSLCrypt	19
	Gator	5
	Gh0st	4
	Gamethief	2
Worms	Palevo	25
	Conficker	23
	Lizamoon	9
	Rimecud	3
	Koobface	2
	Storm	2

TABLE VI: Prevalence of malware families and variants detected by EDGe



(a) Distribution of attack sources and targets within the monitored network



(b) Infection impact on the number of inbound attacks

Fig. 2: Heavy Hitters and Prominent Attack Targets

Heavy Hitters: For each internal host within the monitored infrastructure, we count the hourly average number of alerts of type *Attack* in the inbound and outbound directions. In Figure 2a, we illustrate the distributions of the attack sources and targets. We find that the two distributions are dominated by a very small number of heavy hitters. We see that the median number of recorded inbound attacks is equal to 60 per hour. However, this number increases significantly for a small set of internal nodes that are targets of up to 970 attacks per hour. Almost all the servers in our infrastructure are within this highly exposed set. This indicates that **servers are much more preferable attack targets than clients**. We speculate that this is because most malicious pieces of self-spreading software have an initial hit-list of possibly vulnerable hosts. These hit-lists are generated using either scanning or by employing web-spiders and DNS-searches [17]. A highly skewed behavior is also observed in the case of the attack source distribution. Approximately **5% of the internal hosts account for more than 70% of the total outbound attacks originating from the intranet**. Blocking or better defending against these systems can significantly reduce the number of recorded extrusions, safeguarding at the same time exposed internal nodes.

Impact of Infections on Inbound Attacks: Next, we examine the impact of an infection on the number of monitored inbound attacks. We count the average number of alerts of type *Attack* targeting hosts in our intranet in an hourly basis for healthy hosts, for infected hosts prior to their infection, and for infected hosts after their infection. Note, that based on EDGe the infection time is estimated after the actual infection manifests. If a node is infected but the corresponding malware remains dormant, it will not generate a malicious footprint on the network. Therefore, in Figure 2b, nodes of this type are in the pre-infection phase.

In the median case, healthy nodes and nodes in the pre-infection phase are targets of approximately 3 attacks per hour. These are mostly reconnaissance attacks, such as scanning, that could be precursors of a more serious attack. The corresponding number of inbound attacks in the case of infected hosts is 7, i.e., more than double. However, if we observe the tails of the distributions we see a much more sharp change. At the 95th percentile of the distributions, we see on average 5 and 9 inbound attacks per hour for healthy nodes and nodes in the pre-infection phase, respectively. For infected hosts this number rises sharply to 50 inbound attacks per hour.

We learn that **after a host is infected, the number of inbound attacks increases sharply**, in the median case by a factor of 2 and in the tail of the distribution by a factor of 5.5. We speculate that this is because most malware also operate as backdoors, allowing the installation of additional malicious code. In this way they increase the attack vector of the infected host making it a much more attractive target. This is especially true for servers, which dominate the tail of the distributions shown in Figure 2b.

B. Correlations Across Space and Time

Spatial Correlations: The infections we observe exhibit strong spatial correlations. We define *IP distance* as the absolute difference between the integer representation of two IP addresses. For each host that remains healthy throughout the tracing period, we measure its *IP distance* to the closest infected host. For each host that becomes infected, we measure its IP distance to the nearest infected host at the time of infection.

In Figure 3, we plot the Cumulative Distribution Function (CDF) of the IP distance for healthy and infected hosts. Note that in our infrastructure we use two large blocks of IP addresses, which explains the sharp increase we see for IP distance values above 2,600. We observe that **infected hosts are consistently in very close proximity with other infected hosts**. 80% of these hosts have at least one other infected host in an IP distance which is less than 200, meaning that they are likely located in the same subnet. The corresponding

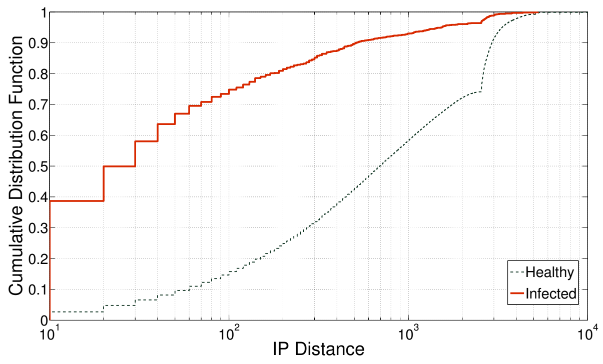


Fig. 3: Infections spatial correlation

percentage for healthy hosts considering the same IP distance is significantly lower, equal to 15%. The presence of strong spatial correlations indicates that certain subnets within a network are "weak links". For this reason, hosts close to existing infections are much more likely to become infected in the future. Observing clusters of infections should guide administrators to review and revise the deployed baseline defenses and security policies.

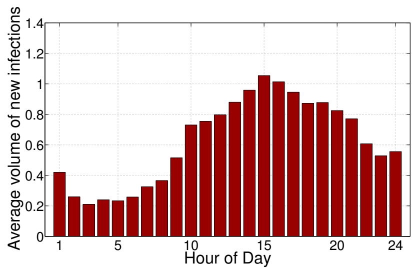
Correlations Across Time: The distribution of the time when infections outbreak exhibits a diurnal pattern as illustrated in Figure 4a, where we see that most infections occur during working hours. This is due to the fact that the activity of client nodes, where most infections outbreak, exhibits strong diurnal patterns.

Another interesting aspect of the extracted infection time series is their burstiness across different time scales. To quantify burstiness, we compute the Allan deviation [18] of the infection time series at different scales. The Allan deviation is given by the following equation:

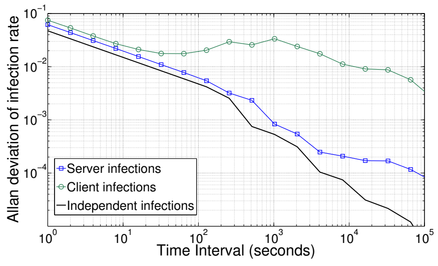
$$\sigma_x^2(\tau) = \frac{1}{2}\langle(\Delta x)^2\rangle \tag{2}$$

The time series is discretized into time intervals of length τ and each interval yields a sample x_i of the number of infections that occurred within it. The equation measures the difference between successive samples x_i for different interval lengths τ.

In Figure 4b, the bold line in the bottom shows the minimum possible deviation which occurs when all infections have independent time arrivals. Intuitively, the Allan deviation should diverge from this reference significantly in time scales where the signal exhibits high burstiness. Figure 4b shows that **server infections in short time-scales are almost independent**, however, this changes if we look at time scales above one hour. This non-burstiness of server infections in short time scales suggests that measuring the infections over hourly intervals can provide a useful long-term average of the expected infections. This observation can be used to build a predictor of near-future infection incidents using simple linear time series models that capture short-range dependences, like ARIMA. On the other hand, we find that **client infections**



(a) Distribution of infections for different hours of day



(b) Infections burstiness for clients and servers

Fig. 4: Infections temporal correlations

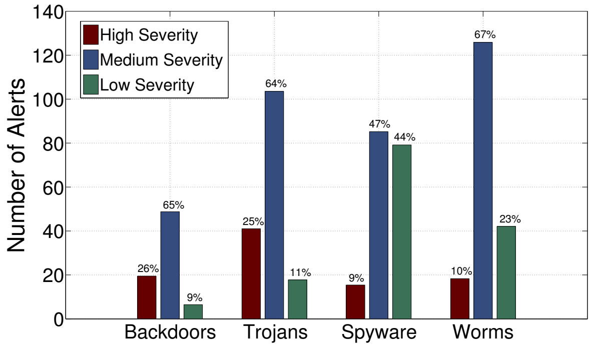


Fig. 5: Daily volume of alerts for different severity levels and malware families

are consistently more bursty and this is more evident for time-scales above two minutes.

C. Differences Among Malware Families

Alert Severity: We next investigate the severity of the alerts produced by different malware families using the classification of Snort into high, medium, and low priority alerts. In Figure 5 we illustrate the daily average number of bundled alerts for different severity levels and malware families. In addition, we mark the percentage of the total alerts of a family to which each bar corresponds. Backdoors exhibit, on the one hand, the highest percentage of high severity alerts and, on the other hand, the lowest absolute volume of alerts among all malware families. The network footprint of backdoor infections is dominated by communication attempts to their C&C, which are not frequent, but are of high severity. In the case of trojans we observe the highest total of high and medium severity alerts. This is due to their rich behavioral profile and their persistence in attempting to perform the respective cycle of malicious actions within short time intervals. Spyware generate a very large number of low severity alerts, mostly due to redirections to malicious domains and phishing attempts. Worms on the other hand exhibit an IDS footprint that is dominated by medium severity alerts that mostly correspond to scanning.

Impact of Infection on Outbound Alerts: We further analyze the impact of infections on the malicious behavior exhibited by internal hosts. Figure 6 highlights the increase in generated alerts in the post infection phase for different malware families. The amount of alerts in the pre-infection phase is relatively low for all families, with a median ranging between 0.7 and 2 alerts in an hourly basis. This means that compromised hosts exhibit very low outbound malicious activity before they become compromised.

In the case of backdoors, for the bulk of the distribution we see only a marginal increase in the outbound malicious activity observed after an infection. Specifically, in the median case the average number of recorded alerts per hour increases from 0.7 to 1.5. In the tail of the distribution we see a much sharper increase. 5% of the infected machines appear to be generating 1 to 10 thousand outbound alerts per hour. These are machines that at some point of their lifetime have been actively used to initiate DoS attacks or to send massive amounts of spam. On the other hand, **the vast majority of backdoor infections remain dormant** and only rarely communicate with their C&C to receive instructions and binary updates.

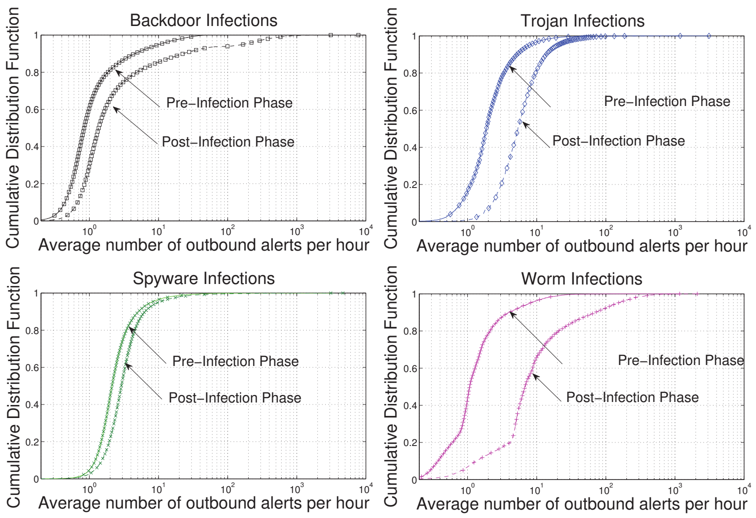


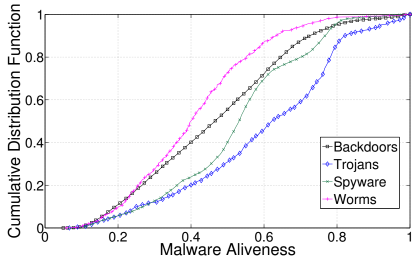
Fig. 6: Infection impact on the number of outbound alerts for different malware families

Trojans exhibit a more prominent increase of malicious behavior after the infection point. In the median case we observe an increase by a factor of 4 in the average number of outbound alerts. We consider that this is due to the fact that trojans frequently attempt to install additional malicious binaries from remote domains. This is consistent with recent studies, e.g., [19], that highlight the commodization of malware distribution and the predominance of pay-per-install services.

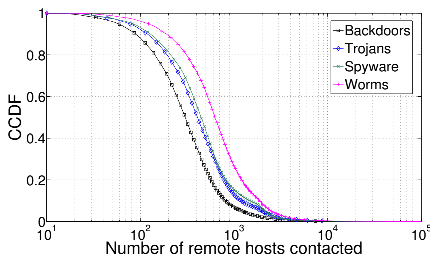
In the case of spyware we see that the infection has only marginal impact on the number of outbound alerts. Note, however, that these nodes exhibit the highest number of outbound alerts in the pre-infection phase. We speculate that this is because users who install spyware are prone to visiting low reputation or malicious domains.

Worms exhibit the most evident increase in the outbound activity after an infection. Specifically, we see that 90% of the infected machines generate on average at least 80 severe alerts per hour throughout their lifetime. This is due to the extensive reconnaissance and scanning activity that is used to build a list of targets for propagation and to perform exploitation attempts. This process is automated and is not based on an external stimulus, thus it often generates a vast number of alerts.

Malware Aliveness: We next analyze the extent to which the malware in our study are *alive*, meaning that they exhibit a visible malicious activity, or whether they are more stealthy, undergoing dormant periods where no detectable network footprint is generated. We define the aliveness of an infection as the ratio of the number of days where we detect the malware as active divided by the total number of days the infected host is active. We tag an infection as alive during a specific day if it has triggered at least one outbound alert of type *Attack* or *Compromised*. On the other hand, an infected host is considered alive during a given day if it has generated any type of alert, including *Policy* related alerts. In Figure 7a we illustrate the CDF of the aliveness of different malware families.



(a) CDF of aliveness, i.e., fraction of days a malware is active, for different malware families



(b) Number of external domains contacted from infected hosts and amount of recorded alerts

Fig. 7: Malware Aliveness and Fanout

We see that trojans are the most alive malware family: 6% of the trojan infections have an aliveness value above 0.9; in the median case the aliveness value is 0.63, suggesting that trojans do not go stealthy by suspending their activity. We speculate that this is because trojans autonomously initiate different actions during their lifetime. They typically attempt to update their binary, contact a remote controller to report, and download additional malicious eggs to install on the compromised hosts. Whenever the infected host is active it is very likely that it will initiate at least one of these activities.

Spyware infections appear to be slightly less alive than trojans. We consider that this is because spyware are dependent to user activity and typically hook to a specific application, such as a browser, a VoIP or FTP client, and, therefore, are triggered when the user utilizes the application. However, when this happens we observe a large number of alerts within a short time window of few minutes.

Backdoors are significantly more stealthy exhibiting aliveness values below 0.5 in the median case. This can be attributed to the fact that these infections will typically undergo two stages, namely the communication with the controller to update their instruction set and the manifestation of the malicious activity, e.g., being a DoS attack or a spam campaign. The former activity has a period in the order of days for most of the backdoors we observed, whereas the latter activity is only seldom observed in our trace.

Worms appear to be the least alive threat in our trace. In the median case worm infected systems exhibit an aliveness value of 0.4. The most predominant worm in our trace, Palevo, spreads through instant messaging applications, therefore it requires the user to use social networking chats or a typical instant messaging client in order to propagate. This user triggered behavior might not be observed for several days of activity.

Malware Fanout: Furthermore, we count the number of distinct remote IP addresses to which malware communicate. In Figure 7b we illustrate the number of remote hosts contacted by internal compromised systems during the entire lifetime of their infection. We consider that we have a communication attempt when we record an outbound alert of type *Attack* or *Compromised* originating from an infected system.

Backdoor infected hosts initiate this communication to contact their C&C. The most prominent backdoors in our trace, i.e., Avzhan, Parabola and Zeus, use a predefined set of rendezvous points to contact their controller and do not use any type of bullet proof hosting. In the median case we see that the infected machines contact at least 300 external domains during their entire lifetime, mostly to receive instructions. In the tail of the distribution we see that 5% of backdoor infections contact at least 1,000 remote hosts and this number reaches a maximum of 3,146 contacts. These are mostly SpyEye infections which use fast-flux to generate the C&C addresses to communicate, resulting in a high number of total contacts.

Trojans have significantly more contacts than bots. Specifically we see that in the median case trojans communicate with 455 external hosts. The vast majority of these domains host malicious content, and the purpose of this communication is to download additional badware. 10% of the most active trojans appear to initiate at least 1,450 communications during their lifetime. Most of these connections can be attributed to Torpig-related infections that use domain-flux in order to generate the domains used to send their harvested user data.

Spyware exhibit similar behavior to trojans. However, the type of activity triggering this behavior is totally different. The vast majority of the observed communication attempts are redirections to third-party web-sites. The most prominent activity of this type is manifested by the HotBar spyware, which is a browser add-on, that will track down user's activity to produce personalized advertisements and will use pop-ups and custom buttons on the browser to force the user to visit relevant sites.

As expected worms communicate with the highest number of remote hosts, generating alerts almost entirely of the class *Attacks* and very few of the type *Compromised*. These are reconnaissance attempts in the form of active scanning. In the median case we see that worm infected machines communicate with at least 620 external hosts whereas a small set of very aggressive worms, accounting for 5% of the total infections, contact at least 2,650 external hosts with a maximum of 10,625. Note that the same malware qualitative patterns also hold at finer time scales.

VI. Related Work

A number of previous studies have focused on IDS alert correlation and aggregation, e.g., [20], [21], [22], [3]. These studies evaluate proposed solutions on a small number of testbed-based benchmarks, like the DARPA dataset [23], and are tailored for general-purpose alert analysis rather than for extrusion detection. In our work, we focus on extrusion detection in an operational network and build an IDS alert correlator to detect infected hosts from alerts produced by Snort in a large site with more than 40 thousand unique hosts. A significant part of our validation relies on non-trivial manual investigation. In [13], we evaluate the complementary utility of different forensics evidence we used in the manual investigation process for different types of malware. Furthermore, in [14] to accelerate the diagnosis of security incidents, we use the C4.5 classification algorithm to capture how to combine low-level evidence from IDSs, reconnaissance and vulnerability reports, blacklists, and a search engine.

Another group of studies, e.g., [24], [25], [26], [27], have focused on novel schemes for detecting botnet-type infections using passive monitoring methods, by comparing observed flow-level features with botnet communication patterns. The assumption made in this line of work is that an infected system will initiate a distinct sequence of connection attempts in order to receive instructions, update its binary, and report back to its controller. This concept is similar to EDGe, which exploits signs of recurring malicious activity to detect multistage malware behavior. However, in our work we leverage IDS data to track host behavior and use entropy-based criteria to identify prominent malicious patterns.

Large traces of intrusion data have also been analyzed in previous studies. In particular, in 2003 Yegneswaran *et al.* [28] characterized the distribution, types, and prevalence of intrusions using anonymized IDS alerts and firewall logs collected by DShield [29]. Besides, in 2005 Kati *et al.* [30] used a large trace of IDS alerts (from DShield and other sources) to characterize correlated attacks and collaborative intrusion detection schemes. Finally, Chen *et al.* [31] in 2011 replayed packet traces to multiple IPSs and used a voting scheme to identify malicious traffic sessions. In contrast, in our work we use a large collection of raw IDS alerts collected in 2011 to characterize a specific type of intrusion, i.e., infected hosts within a monitored site.

Finally, another group of studies analyzed security incidents in the wild using alternative data sources. Most related to our work Sharma *et al.* [32] analyzed 150 security incidents that occurred in a supercomputing center over five years using data from five security sensors. Maier *et al.* [33] tailored custom heuristics to detect scanners, spammers and bot-infected hosts in packet traces from a large number of residential DSL customers. Gu *et al.* [34] performed an extensive passive and active measurement analysis of three predominant botnets and made a number of observations regarding the similarities and differences exhibited in the infection methods. Ho *et al.* [35] analyzed packet traces responsible for the alerts triggered by an IDS in a campus site, identified common sources of FPs/FNs, and tailored a mechanism to improve the detection accuracy.

VII. Discussion

False Negatives: We opt to design EDGe to produce a small number of false positives. This is one of our main goals as the excessive amount of false positives is an important limiting factor for IDSs. Therefore, in the trade-off between false positives and negatives we prefer to incur more false negatives in order to reduce the amount of false positives. Quantifying the false negative rate in a production environment is not possible. However, towards this end one can use synthetic or testbed-based evaluation traces, as discussed in Section VI, where security incidents are known and controlled. Our work is complementary to such approaches and establishes techniques to find and validate security incidents in traces from production environments.

Academic Infrastructure: Our characterization results in Section V are based on data from an academic infrastructure and should only be carefully generalized, when possible, to other types of networks. For example, we expect that similar qualitative findings about the impact of infections and the presence of heavy hitters hold in networks of different type. In contrast, we expect that the volume of infections will be lower in more tightly managed environments. Moreover, our approach assumes static IP addresses, which in the studied network account for 44% of the IP addresses. However, it can be extended to dynamic IP addresses if DHCP lease records are available.

Signature-based IDS: In this work we rely on a signature-based IDS to identify active infections in the monitored infrastructure. A fundamental limitation of an IDS is that it can only detect known attacks with existing signatures.

Besides, it cannot inherently identify social engineering or browser attacks, such as drive-by-downloads and cross-site scripting. To address these issues, security architectures should always consist of multiple layers of defensive mechanisms that complement each other. For example, anomaly detection schemes (for a survey refer to [36]) can be employed to identify patterns corresponding to unknown attacks, whereas host-based AVs can be used to detect attacks targeting the end-user.

VIII. Conclusions

Although network intrusion detection systems (IDSs) have been studied for several years, the security focus has been to detect illicit intrusions on the monitored infrastructure. Traditionally, IDSs inspect inbound traffic for attacks against Internet-exposed systems. However, preventing internal system compromises has proven to be a futile task and the existence of internal infections should be considered a certainty. In this work we focus on the following problem: from a deployment of the popular Snort IDS in an operational network that produces 3 million alerts per day, we want to detect infected hosts within the monitored network with a small number of false positives. To address this problem, we first build a novel IDS alert correlator for Snort, which we call EDGe. EDGe uses a statistical measure to find hosts that exhibit a repeated multi-stage malicious footprint involving specific classes of alerts. In addition, it can identify the malware family and variant of detected infections. A significant part of our work is devoted to the validation of our heuristic. We conduct a complex experiment to assess the security of suspected infected systems in a production environment using data from several independent sources. We find that EDGe produces only 15% false positives. Having validated our heuristic, we apply it to a 9-month long trace of IDS alerts and characterize various important properties of more than 4 thousand infected hosts in total. For example, we find that among the infected hosts, a small number of heavy hitters originate most outbound attacks and that future infections are more likely to occur close to already infected hosts.

IX. Acknowledgements

The authors wish to thank Prof. Bernhard Plattner and Dr. Vincent Lenders for their invaluable help and fruitful discussions. We would also like to thank Stephan Sheridan and Christian Hallqvist at ETH for their help in the collection and archiving of the data used in this paper.

References

[1] "Network intrusion detection system for UNIX and Windows," http://www.snort.org, 1998.

[2] "Emerging Threats Rules," http://www.emergingthreats.net, 2003.

[3] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of the 4th International RAID Symposium*, 2001, pp. 54–68.

[4] L. Etienne and J.-Y. Le Boudec, "Malicious traffic detection in local networks with snort," EPFL, Tech. Rep., 2009.

[5] "Network Security Archive," http://www.networksecurityarchive.org, 2006.

[6] P. Smyth and R. M. Goodman, "An information theoretic approach to rule induction from databases," *IEEE Trans. on Knowl. and Data Eng.*, vol. 4, pp. 301–316, August 1992.

[7] G. Piatetsky-Shapiro, "Discovery, analysis and presentation of strong rules," in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley, Eds. AAAI Press, 1991, pp. 229–248.

[8] E. Raftopoulos and X. Dimitropoulos, "Detecting, validating and characterizing computer infections in the wild," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*.

[9] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin, "Automatically identifying trigger-based behavior in malware," 2008.

[10] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "Is host-based anomaly detection + temporal correlation = worm causality?"

[11] "Advanced automated threat analysis system," www.threatexpert.com.

[12] "TrendMicro Threat Encyclopedia," http://about-threats.trendmicro.com.

[13] E. Raftopoulos and X. Dimitropoulos, "Understanding network forensics analysis in an operational environment," in *IEEE International Workshop on Cyber Crime*, San Francisco, CA, USA, May 2013.

[14] E. Raftopoulos, M. Egli, and X. Dimitropoulos, "Shedding light on log correlation in network forensics analysis," in *Proceedings of the 9th DIMVA Conference*. Crete, Greece: Springer-Verlag, Jul 2012.

[15] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation."

[16] "Malware Threat Center," http://mtc.sri.com/, 2007.

[17] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *USENIX'02*. Berkeley, CA, USA: USENIX Association, 2002, pp. 149–167.

[18] D. W. Allan, "Time and frequency (time domain) characterization, estimation and prediction of precision clocks and oscillators," *IEEE Trans. UFFC*, vol. 34, November 1987.

[19] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring Pay-per-Install: The Commoditization of Malware Distribution," in *USENIX'11*, San Francisco, CA, August 2011.

[20] X. Qin and W. Lee, "Statistical causality analysis of infosec alert data," in *Proceedings of The 6th International RAID Symposium*, 2003.

[21] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," ser. RAID '00.

[22] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM TISSEC*, vol. 6, 2003.

[23] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer Networks*, vol. 34, October 2000.

[24] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proceedings of HotBots'07*. Berkeley, CA, USA: USENIX Association, 2007, pp. 7–7.

[25] J. Goebel and T. Holz, "Rishi: identify bot contaminated hosts by irc nickname evaluation," in *Proceedings of HotBots'07*. Berkeley, CA, USA: USENIX Association, 2007, pp. 8–8.

[26] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *Proceedings of SRUTI'06*. Berkeley, CA, USA: USENIX Association, 2006, pp. 7–7.

[27] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," 2008.

[28] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: global characteristics and prevalence," in *SIGMETRICS '03*.

[29] "Cooperative Network Security Community," www.dshield.org, 2000.

[30] S. Katti, B. Krishnamurthy, and D. Katabi, "Collaborating against common enemies," in *IMC '05*, Berkeley, CA, USA, 2005, pp. 34–34.

[31] I.-W. Chen, P.-C. Lin, T.-H. Cheng, C.-C. Luo, Y.-D. Lin, Y.-C. Lai, and F. C. Lin, "Extracting ambiguous sessions from real traffic with intrusion prevention systems," *I. J. Network Security*, 2012.

[32] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. K. Iyer, "Analysis of security data from a large computing organization," in *DSN*, 2011.

[33] G. Maier, A. Feldmann, V. Paxson, R. Sommer, and M. Vallentin, "An assessment of overt malicious activity manifest in residential networks," in *DIMVA'11*, Berlin, Heidelberg, 2011, pp. 144–163.

[34] S. Shin, R. Lin, and G. Gu, "Cross-analysis of botnet victims: New insights and implications," in *RAID*, 2011.

[35] C. yuan Ho, Y. dar Lin, Y. cheng Lai, I. wei Chen, F. yu Wang, and W. hsuan Tai, "False positives and negatives from real traffic with intrusion detection/prevention systems."

[36] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.