Diss. ETH No. 22022
TIK-Schriftenreihe Nr. 146

Extrusion Detection: Monitoring, Detecting, and Characterizing Internal Infections

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Elias Raftopoulos

Master of Science in Computer Science
born May 22, 1981
citizen of Greece

accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Prof. Dr. Xenofontas Dimitropoulos, co-examiner
Prof. Dr. Evangelos Markatos, co-examiner
Dr. Christian Kreibich, co-examiner

2014

Abstract

It is never impossible for a piece of malware to infect a computer system, only improbable.

The deployment of large networked systems in critical domains such as financial, military, health and government infrastructures necessitates the existence of security assurances in building software and networking components. These systems are often part of an unbounded network, such as the Internet, with no centralized authority, predefined security policies and trust relations. Thus, they allow dynamic memberships where authorization schemes are not enforced, and are based on localized administration. Although, security experts struggle to fend off these infrastructures, it has been proven that even well administered networks exhibit difficulties defending effectively against modern sophisticated malware threats. The existence of compromised machines within the premises of any monitored infrastructure should be considered a certainty.

Therefore, it is no longer sufficient for security practitioners to harden the network perimeter by deploying network defenses, such as firewalls, and limiting the network services exposed to the internet. Internal hosts must be protected and, thus, traffic generated from them must be monitored for signs of compromise. This signals a shift from the traditional fortress concept, that would perceive the network security to be as good as the effectiveness of its perimeter defenses. Security practitioners cannot solely focus on intrusion detection, but need effective tools to perform *extrusion detection*, i.e., methods to detect and monitor hosts within their network that are already infected by malware.

Motivated by this problem, in this work, we introduce a holistic approach for detecting infected machines within the internal monitored network. First, we tailor a novel IDS alert correlator that detects internal infections with a low false positive rate of 15%. Our heuristic uses an information theoretic measure to identify statistically significant temporal associations between a selected pool of alerts. In this way it detects the multi-stage alert signature generated by malicious hosts within the network.

Second, we perform a systematic validation of detected infections on live hosts within the production environment. Based on our experiment, we describe how to leverage heterogeneous security data sources to remotely diagnose live infections in a large production network and evaluate their utility in making an assessment.

Third, we tailor a decision support tool based on the C4.5 algorithm that reflects how low-level evidence from diverse security sources can be combined to diagnose different families of malware. The derived tool encodes a large part of the decisions of the analyst in correlating heterogeneous security logs and is useful for expediting the time-consuming manual security assessment process of security incidents.

Fourth, we perform a thorough characterization of 4,358 infections detected in a production environment and derive novel insights about malware patterns in the wild. We characterize the volume, types and impact of infections, compare key characteristics of different malware families, and illustrate how infections correlate across time and space.

Our findings and developed tools can help security practitioners to fine-tune their baseline defenses, prioritize the collected alerts originating from heterogeneous security sensors, improve the detection accuracy of the deployed IDS systems, and guide the forensics investigation process of identified security incidents.

Kurzfassung

Es ist für eine Schadsoftware niemals unmöglich, in ein Computersystem einzudringen, nur unwahrscheinlich..

Die Verwendung grosser vernetzter Systeme in kritischen Bereichen wie dem Finanzsektor, dem Militär, im Gesundheitswesen und in Infrastruktur, die für das Regierungswesen benötigt werden, macht die Existenz von Sicherheitszusagen beim Entwickeln von Software und von Netzwerkkomponenten nötig. Solche Systeme sind oft Teil eines unbegrenzten Netzwerks wie dem Internet, das keine zentrale Verwaltung und keine vordefinierten Sicherheitsrichtlinien oder Vertrauensbeziehungen hat. Daher erlauben sie dynamische Mitgliedschaften, bei denen Autorisierungsvorschriften nicht durchgesetzt werden und die auf lokaler Administration basieren. Obwohl Sicherheitsexperten sich bemühen, diese Infrastrukturen abzusichern, ist nachgewiesen, dass selbst gut administrierte Netzwerke sich nur schwer gegen moderne Schadsoftware verteidigen können. Es ist daher davon auszugehen, dass es innerhalb jeder überwachten Infrastruktur kompromittierte Maschinen gibt.

Für Sicherheitsfachleute ist es daher nicht mehr ausreichend, den Netzwerkperimeter zu schützen, sei es durch den Einsatz von Technologien wie Firewalls oder durch die Beschränkung derjenigen Netzwerkdienste, die im Internet verfügbar sind. Computer innerhalb des Perimeters müssen ebenfalls geschützt werden und deshalb muss auch Datenverkehr, der von ihnen ausgeht, auf Zeichen von Kompromittierung untersucht werden. Dies zeigt schon, dass das traditionelle Konzept einer Festung ausgedient hat, bei dem Netzwerksicherheit als Perimetersicherheit betrachtet wird. Sicherheitsfachleute können sich nicht mehr nur auf Einbruchserkennung (intrusion detection, IDS) beschränken, sondern benötigen auch effektive Werkzeuge zur Erkennung von Ex-

trusion, also Methoden, die bereits infizierte Rechner im Netzwerk entdecken und überwachen.

In dieser Arbeit präsentieren wir einen holistischen Ansatz, das Problem der Entdeckung infizierter Maschinen zu lösen. Zuerst entwickeln wir einen neuen IDS-Alarm-Korrelator, der interne Infektionen mit einer geringen falsch-positiv-Rate von nur 15% entdeckt. Unsere Heuristik benutzt Messmethoden aus der Informationstheorie, um statistisch signifikante Korrelationen zwischen Alarmen zu entdecken. So entdeckt der Korrelator auch Angriffe, die von infizierten Maschinen in mehreren Stufen durchgeführt werden.

Zweitens validieren wir systematisch die entdeckten Infektionen auf Rechnern in einer Produktionsumgebung. Darauf aufbauend beschreiben wir, wie man verschiedenste Quellen von Sicherheitsinformationen nutzen kann, um aus der Ferne aktuelle Infektionen in einem grossen Produktionsnetzwerk zu diagnostizieren. Wir werten diese Datenquellen ausserdem hinsichtlich ihrer Nützlichkeit für Sicherheitseinschätzungen aus.

Drittens entwickeln wir ein Werkzeug basierend auf dem C4.5 Algorithmus, das bei Sicherheitsentscheidungen unterstützen kann. Dadurch wird klar, wie man hochspezifische Informationen aus verschiedensten Datenquellen zur Diagnose von Schadsoftware nutzen kann. Das Werkzeug beinhaltet dabei bereits einen grossen Teil der Entscheidungen, die ein menschlicher Sicherheitsanalyst trifft, um die verschienenen Datenquellen miteinander zu korellieren und ist daher nützlich, die zeitaufwendige händische Untersuchung von Sicherheitsvorfällen zu beschleuningen.

Viertens untersuchen wir 4.358 Infektionen, die in einem Produktionsnetztwerk entdeckt werden und charakterisieren sie gründlich. Daraus leiten wir neue Einsichten über das Verhalten echter Schadsoftware ab. Wir charakterisieren insbesondere Anzahl, Typ und Auswirkung von Infektionen, vergleichen verschiedene Familien von Schadsoftware und beschreiben, wie Infektionen räumlich und zeitlich korellieren.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Security Landscape

As the Internet has become more pervasive during the last decade, malware (malicious software) has evolved reaching a high level of sophistication. The early forms of malware variants, such as worms, would attempt to propagate, perform predefined actions on the victims, and carry out easily detectable nuisance attacks, typically against high-profile exposed machines, such as web-servers. Today, the activity of malware is becoming increasingly harmful, focusing on confidential information theft, data leakage, user manipulation, targeted service disruption and other illicit activities [15,20]. The attack target is no longer the exposed server offering services, but rather the corporate and private user which is subjected to client side attacks.

Tailoring effective countermeasures against these threats is an extremely challenging task for a number of reasons. First, the complexity of software and networking components used within the premises of the protected network make it inherently difficult to eradicate all exploitable vulnerabilities. Services offered by internal servers, might be extremely safe when used in isolation in terms of vulnerabilities, however, when combined they might introduce exploitable security flaws [27]. Moreover, networked systems are built on top of complex software modules. It has been shown that any piece of software of considerable size involving a relatively high degree of sophistication should be ex-

pected to contain defects. This is partly due to the fact that software complexity correlates with the vulnerability density, i.e. the number of exploitable vulnerabilities found in a code block of predefined size [9]. Also, the effort required to identify and fix software defects grows exponentially with module size [23].

Second, modern malware increasingly involve the user in their propagation by leveraging various social engineering techniques that bypass intrusion prevention measures. With the emergence of the firewall in the early 1990s, as the centerpiece of network security, and the adoption of Request for Comments (RFC) 1918 for private network addressing, internal hosts were rarely the victims of direct attacks in contrast to their server counterparts. Protection of the internal network consisted of hardening internet-facing servers so that the exposure level was minimized, and strengthening access control mechanisms in order to block unauthorized access to private resources. However, the last decade saw this model invert. Since 2005 with the emergence of the new generation of highly sophisticated post-worm malware threats, such as Zeus, Conficker, Koobface and Torpig, end-users have been subjected to an increasing number of client-side attacks [11, 30]. No longer services offered by servers are the sole targets of external attackers. Applications used by the user, such as the web browser, the email client, the chat programs, and the social networking clients are now the direct target of the attacker. This shift from server-side to client-side attacks renders traditional intrusion detection mechanisms ineffective, since the attack involves the end-user, leveraging his regular activity to hide the manifested malicious activity.

Third, and most significantly, the race between malcode writers and security practitioners is heavily unbalanced. The former group seeks to gain unauthorized access to the protected network by exploiting any existing vulnerability, ranging from standard software bugs and access control policy design flaws, to user negligence or lack of technical expertise. On the other hand the latter group needs to patch every single software module against all possible exploits for a large population of protected machines. Moreover, security practitioners have to educate the users in order to avoid simple social engineering client-side attacks, which would allow the attacker to execute code on the victim machine without having to gain access to it.

For these reasons, it is clear that in the presence of sophisticated

malware, building a impenetrable network fortress is a futile struggle and infections in the premises of the internal network should be considered inevitable. Thus, it is no longer sufficient for security practitioners to harden the network perimeter by deploying network defenses, such as firewalls, and limiting the services exposed to the internet in order to harden Internet-facing machines. Internal client workstations must be protected and, therefore, traffic generated from them must be monitored for signs of compromise. This signals a shift from the traditional fortress concept, that would perceive the network security to be as good as the effectiveness of its perimeter defenses. Security practitioners cannot solely focus on intrusion detection, but need effective tools to perform *extrusion detection*, i.e., methods to detect and monitor hosts within their network that are already infected by malware.

1.2 Extrusion Detection

To understand the concept of extrusion detection one should first establish what intrusion detection is and how it is currently used in the context of network monitoring. Intrusion detection can be defined as the act of detecting and responding to computer misuse. The goal of intrusion detection is seemingly simple, to detect intrusions initiated by remote malicious users targeting the premises of a monitored infrastructure. However, in practice this task can become extremely challenging since the detection is performed based on traces generated by the manifestation of the malicious behavior which might lack completeness and correctness. This means that critical information required to make a definite assessment might not be available or that the collected traces can contain untrusted or incorrect data.

Significant effort has been put in building effective Intrusion Detection Systems (IDSs) since the first real time anomaly detection paradigm was proposed by Denning *et al.* [16]. Several survey research papers present the state of the art in anomaly detection approaches [8, 14, 18, 24]. The principle model considered by this long line of work is centered around the inspection and analysis of traffic targeting an exposed server within the monitored network. Signature based IDSs examine the content of the collected traffic traces and try to identify predefined sequences of data that correspond to known malicious behaviors [25, 28]. On the other hand anomaly based IDSs raise an alarm

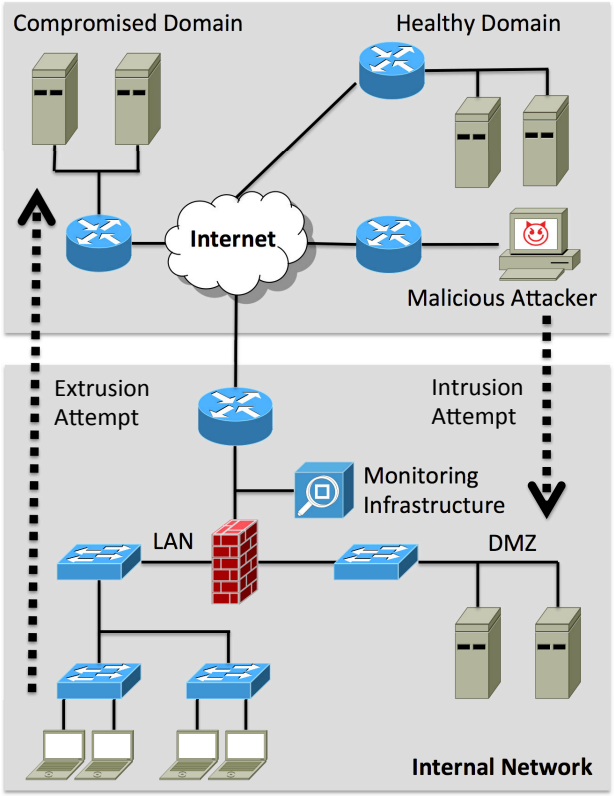


Figure 1.1: *Intrusion and Extrusion Detection*

whenever the observed activity diverges from what is characterized as *normal* behavior [26, 34].

Traditional IDS systems focus on the inspection of inbound traffic, i.e. traffic from the internet to the intranet, for signs of attacks against exposed internal hosts [10]. In this work we reverse this paradigm and shift our interest to outbound traffic, i.e. traffic originating from the local intranet and targeting a remote internet host. In this way we perform *extrusion detection* by monitoring and analyzing suspicious connections initiated by internal systems to remote potentially malicious domains.

In Figure 1.1 we illustrate the difference between the two approaches. On the right side, we see an external attacker initiating attacks against servers in the Demilitarized Zone (DMZ) of the internal network. DMZ is a subnetwork containing and exposing the organization's external-facing services to the Internet. These attacks can be active exploitation attempts against a web server running an Apache HTTP daemon on port 80, a mail server running a Sendmail service on port 25, and a an OpenSSH server running on port 22. The goal of the attacker is to gain unauthorized access on the victim machines, by exploiting a vulnerability on the running services. If successful, then the attacker can escalate the attack by augmenting his privileges and gaining administrator access, at which point he can perform a wide range of malicious actions such as compromise of the integrity or confidentiality of data, denial of service, web-site defacement, planting of malicious code, and data exfiltration.

On the left side of Figure 1.1 we depict the outbound communication attempts initiated by an internal host towards a compromised malicious domain. These attempts might correspond to a user clicking a URL contained in a received email, can get triggered by the activity of an existing trojan infection attempting to exfiltrate stolen user confidential data by using an IRC channel to its controller, or can signify attempts of P2P bots located within the internal network to fetch new instruction sets from their peers. In either case, monitoring outbound traffic provides invaluable indicators of unauthorized activity originating from internal malicious hosts. We define *extrusion detection* as the process of identification of malicious behavior by inspecting and analyzing outbound network traffic.

Extrusion detection can be leveraged to provide evidence of active malware infections within the monitored internal network. As we show in Sections 2 and 4, malware will undergo a series of recurring actions during their lifetime. They will typically attempt to propagate to other vulnerable hosts in proximity, redirect the user to third-party malicious domains for profit, fetch malicious binaries from the internet to update or install additional badware on the victim host, and establish communication channels to receive instructions and share confidential harvested data with their controller. All these activities are triggered by the infected hosts and are visible on the outbound traffic captured at the border of the infrastructure. Therefore, extrusion detection con-

stitutes a indispensable monitoring practice that can provide invaluable pieces of evidence to the security practitioner, regarding active malware infections within the monitored infrastructure.

1.3 The Security Cycle

In this section we give some insights about core concepts that are used throughout this work and provide the respective definitions. These definitions stem from current best practices in network security monitoring and from our own experience in performing security assessment in a large operational environment. Our focus is on extrusion detection and, thus, the presented security terms have been adjusted to reflect this goal.

Network security comprises of the technologies, processes and best practices put into place in order to protect networks, computers, services and data from attack, unauthorized access, and compromise. It consists of four main processes that exhibit a cyclic dependency, as illustrated in Figure 1.2, namely Detection, Assessment, Response and Adaptation.

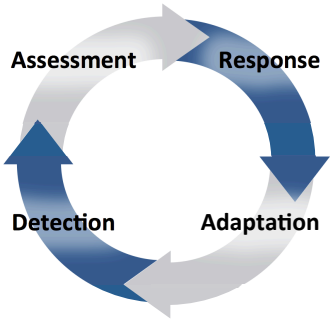


Figure 1.2: *Different stages of the security cycle*

- *Detection* is the process of identifying security incidents that can lead to the compromise of a vulnerable host, or that result from the malicious behavior manifested by an ongoing malware infection.

- *Assessment* is the process of combining multiple sources of security relevant data, captured from diverse sensors strategically placed in the infrastructure, in order to evaluate whether a reported security incident is indeed an active infection.

- *Response* is the process of validating the findings of the Detection and Assessment stages. It involves a thorough forensics investigation process, during which extensive evidence are collected, processed, analyzed and correlated in order to increase the certainty about a reported infection.

- *Adaptation* is the process of incorporating the knowledge distilled from investigating a security incident in the deployed network defenses. It focuses on updating and fine-tuning the deployed security monitoring mechanisms in order to better cope with the identified and analyzed threats.

This work makes significant contributions on all four stages of the security cycle. In the context of detection, in Section 2 we tailor a novel IDS alert correlator for Snort which identifies active infections by extracting the multi-stage malicious footprint generated by different classes of malware. In the context of Assessment, in Section 3 we build a decision support tool that allows the security practitioner to combine evidence from heterogeneous security sources to verify different types of malware infections reported by our IDS correlator. In the context of Response, in Section 4 we systematically investigate 200 detected security incidents about compromised hosts by leveraging the output of four commonly used security sources, namely Snort alerts, reconnaissance and vulnerability scanners, blacklists, and a search engine. We then evaluate the (complementary) utility of the four security data sources and analyze the characteristics of effective IDS signatures. Finally, in the context of Adaptation, in Section 4 we introduce an IDS signature quality metric that can be exploited by security specialists to evaluate the available rulesets, prioritize the generated alerts, and facilitate the forensics analysis processes.

1.4 Network Security Monitoring Data Sources

In Section 1.3 we introduced the different stages of the security monitoring process in the context of extrusion detection. We illustrated how Detection generates the input used in the Assessment stage, and how the forensics investigation carried out in the Response stage provides the means to improve the configuration of existing deployed defenses in Adaptation. However, it is clear that we need credible data collected at the network level of the monitored infrastructure in order to sustain this cyclic monitoring process. In this section, we list the most popular sources of network level monitoring data and highlight the ones used in the context of this work.

1.4.1 Packet traces

Packet traces are the most fruitful source of information available from a security monitoring standpoint. They provide full content data preserving the entire packets exchanged between communicating hosts, including information about the protocols and applications used. They constitute invaluable evidence when investigating a reported incident since every byte sequence exchanged by involved entities is recorded and can be analyzed to detect signs of malicious behavior. However, packet traces are the most expensive network-based data available. It can be extremely difficult from an engineering point to build a robust infrastructure that can capture all traffic generated by a reasonably sized network on a busy day. Commodity monitoring software and hardware solutions have not increased their performance to match today's link speeds [12, 29]. Additionally, the storage required to build a repository of full packet traces can become excessively large. In our analysis we study security incidents that span over several months in a large operational network, therefore packet trace collection was not a viable monitoring solution from an engineering perspective.

1.4.2 Flow records

Flow records or conversations are a summarization of the packets exchanged between two communicating hosts. Full packet content data

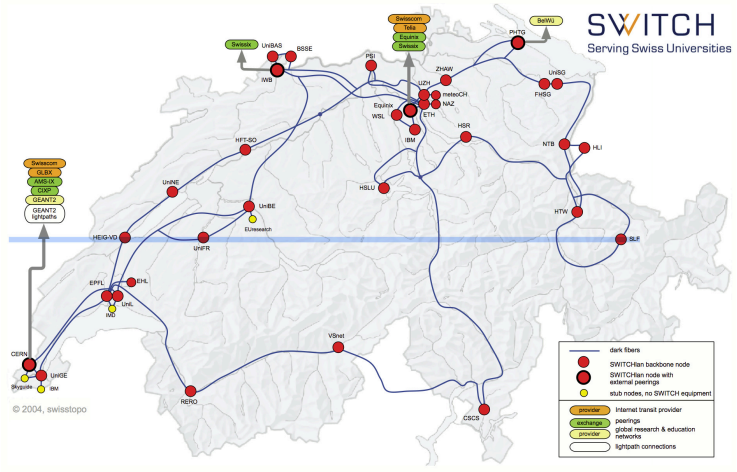


Figure 1.3: *SWITCH backbone network*

are not saved. However, critical elements of the communication are preserved including:

- Source and Destination IPs
- Source and Destination Ports
- Protocol (e.g. TCP, UDP, ICMP)
- IP Type of service and TCP Flags
- Timestamp indicating the beginning of the communication
- Amount of bytes and packets exchanged in the conversation

Although, flow records are not as rich as packet streams in terms of information, they are a reasonable compromise from a network investigation standpoint. They, preserve detailed accounting records of all transiting traffic flows crossing the monitoring point, allowing the analyst to extract all communication attempts involving an investigated host. Moreover, flow records are relatively cheap to collect since most enterprise-grade routers today allow to export active conversations at

real-time with storage requirements at least one order of magnitude lower compared to full packet traces.

In our study, we use NetFlow traces collected at the SWITCH backbone network [31]. The SWITCH Internet backbone (AS559) connects 46 single-homed Swiss universities, e.g., ETH, EPFL, various research labs, and other educational institutions to the Internet. In Figure 1.3 we provide an illustration of the SWITCH network. We have been archiving unsampled flow records since 2003 using hardware based NetFlow meters placed on the border routers of the monitored network. These meters capture all traffic crossing the border destined to or coming from the Internet. In a single peering link, we observe in 2011 on average 108.1 million flows per hour, which corresponds to 3,064 million packets. From each flow record we extract the following fields: IP addresses, port numbers, protocol type, byte/packet counts, and timestamp. We do not export TCP flags since they are not supported by the deployed NetFlow meter for performance reasons.

1.4.3 Alert data

In the previous section we illustrated how the collected Netflow data can effectively summarize the communication triggered by a monitored host. However, NetFlow records lack context, since they do not provide any information regarding the type of activity that triggered the respective flows. Towards this end, we use alert data collected from an IDS sensor, which captures and analyzes all traffic crossing our infrastructure's edge router. Alert data differ from packet traces and flow records in that they provide the judgement made by a software product regarding the nature and severity of an observed network event. Therefore, the accuracy of the generated alerts heavily depends on the effectiveness of the deployed system in analyzing the input packet stream.

Our collected alert trace is comprised of raw IDS alerts triggered in the main campus of ETH Zurich by a Snort [28] sensor, which is placed between the edge router of the campus and the network firewall as shown in Figure 1.4. The sensor monitors all the upstream and downstream traffic crossing the border router of the infrastructure. It is configured to use 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 a wide range of malicious activities, ranging from scanning, exploitation attempts, botnet C&C communication, download of malicious binaries, sensitive data exfiltration, and malware reporting and updating.

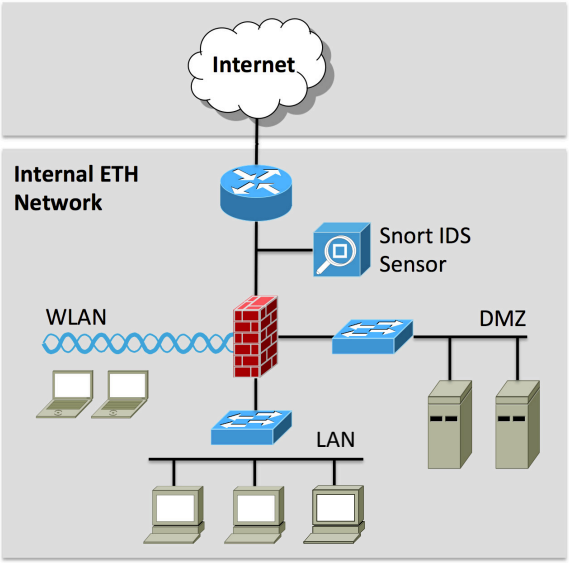


Figure 1.4: *Snort IDS Monitoring Infrastructure in ETH*

The collected alerts have the standard full Snort format [28]. 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 monitoring process has been running on a 24/7 basis with only minor interruptions (corresponding to approximately 99% availability) since October 2009. Our trace spanning over 4.5 years in total accounts for more than 5 billion alerts triggered by 80 thousand internal IPs.

1.5 Network Security Monitoring Tools

The collected Netflow and IDS alert traces constitute a rich and diverse source of monitoring data. However, when performing a forensics investigation to validate a reported security incident, we also rely on a number of monitoring tools to complement the aforementioned sources with additional pieces of forensics evidence. The network security monitoring tools presented in this section are discussed in greater detail in Section 3.

First, we build profiles regarding the type and operation of internal investigated hosts. The context, i.e. the role of a host involved in a security incident, is critical in order to be able to make any type of assessment regarding the validity of a generated alert. For example, consider a monitored host which operates as a mail server running a sendmail daemon and offering SMTP services to clients. If the investigated alerts involving this host are related to an inbound drive-by-download browser attack, then we can safely consider them as false positives. Moreover, the type of OS used and the open services offered by a workstation provide invaluable pieces of evidence that can drive the forensics investigation. If a host is running a flavor of the Debian operating system and the analyzed alerts targeting this host are related to an inbound Microsoft DB SQL-injection attack, then with high certainty we can filter out these alerts as irrelevant.

To build such profiles we leverage a number of open source UNIX tools such as *Hping*, *whois*, *NMap*, and *netcat*. We use these tools to perform reconnaissance scans, such as ICMP ping sweeps, IP fingerprinting, NIC whois querying, and TCP/UDP port-scanning, in order to identify if a host is reachable and exposed to external attacks. We rely heavily on *Nmap* [22] in order to retrieve information regarding the network services running on suspected hosts, determine the type and version of their operating system, and specify the type of probes it responds to. Finally, we employ two well-known vulnerability scanners, namely *Nessus* [17] and *OpenVas* [7], in order to enumerate vulnerable open services on the investigated hosts.

Second, we also build profiles regarding suspicious remote hosts which are contacted by internal investigated machines. We look for known malicious domains which are used by cyber-criminals in order to carry out a number of illicit actions such as hosting malicious bina-

ries, perform drive-by-download attacks, allow trojans to report stolen confidential data, and update the instruction set of bots. To assess whether a remote host is malicious we first rely on five independent public blacklists [1,3–6], which are partly labeled indicating the type of malicious activity exhibited by a blacklisted host, e.g., bot activity, active attack, and spamming. We then leverage the Google search engine in order to fetch information regarding these hosts residing on public Internet resources. In this way we build a comprehensive profile by collecting evidence of the malicious activity the contected host exhibited throughout its lifetime.

1.6 Research Goals and Contributions

In this section we present the main findings and contributions of this thesis. Our work covers four different dimensions of extrusion detection as shown in Figure 1.5. First, in Section 2, we analyze the performance and illustrate the ineffectiveness of a popular IDS system in identifying outbound malicious behaviors in an operational network. Then, we build a novel extrusion detection alert correlator that detects complex malware in our infrastructure with a low number of false positives. Second, in Section 2.4 and 4.3, we perform a systematic validation of suspected infections. We leverage the output of diverse security sensors which we thoroughly analyze in order to provide forensics evidence regarding the presence or absence of a malware infection. Third, in Section 3, towards automating and expediting the forensics investigation process, we build a tool that processes, analyzes, and correlates heterogeneous security relevant traces in order to diagnose different malware families. Fourth, in Section 2.5, we perform an extensive characterization study where we study the behavior and dynamics of 4,358 infected hosts over a period of 9 months in a large academic infrastructure. Finally, in Section 5, we present the characteristics of a large-scale scan targeting our infrastructure, illustrate the subsequent exploitation activity that took place, and assess the aftermath of this orchestrated event in terms of infected internal hosts.

Figure 1.5: *Extrusion Detection Elements studied in this Thesis*

1.6.1 Extrusion Detection of Internal Infections

In recent years corporate and private users have been subjected to an increasing number of client-side attacks. Sophisticated malware no longer target the exposed server but rather attempt to exploit applications users rely on. This signifies a shift on the type of attacks employed in order to compromise the victim. The malware often involves the user in the propagation process by leveraging various social engineering techniques that bypass traditional intrusion prevention measures. For this reason, security administrators need effective tools for detecting hosts within their network, i.e., *extrusion detection*, that are already infected by malware. Detecting internal infections from IDS alerts is an extremely challenging problem due to the high number of false positive alerts, often exceeding 99% [21], IDSs are known to generate.

Motivated by this problem, 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 [33], 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, we compare EDGe against the state-of-the-art IDS correlator Bothunter [19] and show that EDGe detects 60% more infections with fewer false positives.

1.6.2 Validation of Internal Infections

Security analysts often have to cope with an overwhelming amount of collected data traces produced by diverse security sensors strategically placed within the infrastructure. Investigating a suspected security incident is an opaque “art” that involves (1) carefully extracting and

combining evidence from the available security sources, (2) thoroughly understanding how suspected malware operates, and (3) exploiting information about the infrastructure and configuration of the affected network. In this context, security analysts are restricted to using time consuming and often ad hoc forensics analysis processes.

Towards understanding and formalizing the forensic analysis processes we conduct a complex experiment where we systematically monitor the manual forensic analysis of live suspected infections in a large production university network that serves tens of thousands of hosts. In particular, over a period of four weeks, we manually investigate in coordination with the IT department of our university 200 security incidents involving compromised hosts detected by an IDS alert correlator. Based on our experiment, we describe how to leverage four different security data sources to remotely diagnose live infections in a large production network. Second, to delineate the manual investigation process, we evaluate the (complementary) utility of the four data sources. Third, we make available a list of Snort signatures that were effective in detecting validated malware without producing false positives and introduce a novel signature quality metric that can be used by security specialists to evaluate the available rulesets, prioritize the generated alerts, and facilitate the forensic analysis processes. Finally, we apply our metric to the most popular signature rulesets and highlight their differences.

1.6.3 Automation of Forensics Investigation

Computer Security Incident Response Team (CSIRT) experts use a combination of intuition, knowledge of the underlying infrastructure and protocols, and a wide range of security sensors, to perform forensics investigation of suspected incidents. The process of correlating data from multiple sources, in order to assess the security state of a networked system based on low-level logs and events is in most parts manual. Even though, thorough manual investigation is always critical in order to collect all the required evidence and make a definite assessment regarding the severity of an investigated incident, it would be highly beneficial for administrators to have tools that can guide them in the log analysis process, helping them to diagnose and mitigate security incidents.

Towards this end, we build a decision support tool based on the

C4.5 [32] algorithm that reflects how low-level evidence from the four security sources can be combined to diagnose different families of malware, like Torpig, SbBot, and FakeAV. The derived model is useful for expediting the time-consuming manual security assessment of security incidents. It accurately encodes a large part of the decisions of the analyst in correlating diverse security logs and can serve as a decision support tool helping an analyst identify the most critical features that suggest the presence of an infection. In addition, we show that using the decision tree for fully-automated classification correctly identifies infections in 72% of the cases.

1.6.4 Characterization of Internal Infections

Finally, we perform an extensive characterization study of 4,358 infections detected in a production environment and derive several novel insights about malware patterns in the wild. First, we characterize the volume, types and impact of infections. Out of a total of 40 thousand distinct active hosts we observed during the 9-month period, approximately 8% exhibited signs of infections at least once during their lifetime. Second, we compare key characteristics of different malware families. We observe that infections have a strong impact on the number of outbound alerts generated by infected hosts, which increases drastically for backdoors and worms. In addition, we find that trojans have the longest lifetime, followed by spyware, backdoors, and finally worms. Finally, 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. Our time series analysis shows that server infections are almost independent in time, while client infections are consistently more bursty.

Moreover, we study a unique scanning event orchestrated from the Sality botnet that wielded more than 3 million IP addresses to scan the entire IPv4 space. We use a unique combination of unsampled traffic flow measurements and intrusion detection alerts to study how the scan escalated in our monitored infrastructure. We show that the scan was the precursor of a large exploitation campaign that targeted hosts replying to the scan. We then perform a thorough investigation of the internal repliers to assess the impact of the scan. Such analysis is valuable to assess how critical threat the detected inbound scanning really is. Scanning events are regularly recorded with network-based monitor-

ing tools and are often considered by security analysts as baseline noise due to their high frequency and volume. We estimate that the success rate of the Internet-wide scanning attack in a large academic network is 2% in terms of hosts that replied to the scanners. In addition, we conservatively estimate that in the exploitation activity that followed 8% of the hosts that replied were eventually compromised.

1.7 Related Work

A long line of work has utilized extrusion detection techniques to identify botnet related activity. The analyzed activity typically falls into two categories: spam message generation and communication with the botnet controller. By leveraging an infected population of hosts, spammers can launch massive spamming campaigns on short time scales, making detection and blacklisting of offending hosts extremely difficult. Mechanisms to capture this kind of activity are typically based on the analysis of the outbound email trace generated by bots. The AutoRE [35] and Botlab systems [36] executed spambots in a controlled virtualized environment in order to build templates capturing the structural characteristics of generated spam variants. Similarly, Judo [38] and the system developed by Goebel et al. [37] generate signatures of malicious activity by analyzing outbound spam messages sent by compromised hosts end extracting specific static features, such as URLs. Using a complementary approach Chiang et al. [39] and Kreibich et al. [40] reversed engineered the executable of two individual bots, namely Storm and Rustock, in order to infiltrate and study them. This allowed them to record, analyze, and model the outbound spam traffic produced by these botnets.

Besides, researchers have proposed several approaches in order to detect the communication activity generated by bots attempting to report to a controller, update their malicious binary, or propagate a list of instructions to other bots. In [41–43] the outbound network flows generated by compromised hosts are clustered based on IRC-related traffic pattern templates. Similarly, Botsniffer [44] is designed to identify outbound C&C communication towards the servers of centralized botnet architectures. Botminer [45] is tailored to classify network flows to communication related traffic and malicious activity traffic, and then perform cross cluster correlation revealing hosts that exhibit similar

patterns taking into account both criteria. Closer to the work presented in this thesis, BotHunter [19] is designed to detect bot behavior that follows a pre-defined infection alert dialog model, which can be independent of the underlying C&C structure and network protocol used.

Another group of studies analyze security incidents in the wild. Most related to our work Sharma et al. [46] analyze 150 security incidents that occurred in a supercomputing center over five years using data from five security sensors. Their work focuses on the characterization of security incidents based on the traffic generated by infected machines. Maier et al. [47] 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. [30] 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 their triggered activity.

Finally, a number of commercial Security Information and Event Management (SIEM) solutions such as IBM Tivoli Security Compliance Manager [48], Alienvault USM [49], GFI Languard [50], HP ArcSight [51], and BlackStratus LOG Storm [52] unify scattered security sensors within an enterprise and provide a single framework that can be used by security analysts to analyze the generated alert trace by low level security sensors and guide the forensics investigation process. These tools use a set of generic rules, that typically come preconfigured, in order to perform the correlation and fusion of heterogeneous alert data. However, the design and development of advanced rules that can be used to detect active sophisticated malware within the monitored infrastructure is a responsibility left to the security analyst.

Bibliography

[1] Anonymous postmasters early warning system. `www.apews.org`.

[2] Emerging Threats web page. `http://www.emergingthreats.net`.

[3] Projecthoneypot web page. `www.projecthoneypot.org`.

[4] Shadowserver Foundation web page. `www.shadowserver.org`.

[5] The Spamhaus Project. `www.spamhaus.org`.

[6] The Urlblacklist web page. `www.urlblacklist.org`.

[7] Open vulnerability assessment system. http://www.openvas.org/, 2005.

[8] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, 2000.

[9] Victor R. Basili, Lionel C. Briand, and Walclio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[10] Richard Bejtlich. *Extrusion detection: security monitoring for internal intrusions.* Addison-Wesley, 2006.

[11] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr M. Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the zeus botnet crimeware toolkit. In *PST*, pages 31–38. IEEE, 2010.

[12] Lothar Braun, Alexander Didebulidze, Nils Kammenhuber, and Georg Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement.*

[13] Brian Caswell, James C. Foster, Ryan Russell, Jay Beale, and Jeffrey Posluns. *Snort 2.0 Intrusion Detection.* Syngress Publishing, 2003.

[14] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), July 2009.

[15] CSI. Computer crime and security survey, 2010.

[16] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, February 1987.

[17] Renaud Deraison. Nessus: remote security scanner. http://www.nessus.org/, 2004.

[18] Juan M. Estvez-Tapiador, Pedro Garcia-Teodoro, and Jess E. Daz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, (16):1569–1584.

[19] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium* pages 12:1–12:16, Berkeley, CA, USA, 2007.

[20] IC3. Internet crime report, 2007.

[21] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *KDD*, pages 366–375. ACM, 2002.

[22] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, USA, 2009.

[23] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering.* McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[24] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Comput. Netw.*, 51(12):3448–3470, August 2007.

[25] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.

[26] Roberto Perdisci, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Comput. Netw.*, 53(6), 2009.

[27] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 156–, Washington, DC, USA, 2000. IEEE Computer Society.

[28] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, 1999.

[29] Fabian Schneider, Jrg Wallerich, and Anja Feldmann. Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. In *Passive and Active Network Measurement*, Heidelberg, 2007.

[30] Seungwon Shin and Guofei Gu. Conficker and beyond: A large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 151–160, New York, NY, USA, 2010. ACM.

[31] SWITCH. Swiss Tele Communication System for Higher Education. `http://www.switch.ch/`.

[32] Quinlan, J. R. Induction of Decision Trees. In Journal of MAchine Learning, Volume 1, pages 81–106, 1986

[33] P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases In *IEEE Trans. on Knowl. and Data Eng.*, vol. 4, pp. 301316, August 1992.

[34] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. pages 203–222, 2004.

[35] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *Proceedings of SIGCOMM, 2008.*

[36] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *Proceedings of NSDI*, 2009.

[37] Goebel, Jan and Holz, Thorsten and Trinius, Philippi. Towards Proactive Spam Filtering. In *Proceedings of DIMVA*, 2009.

[38] Andreas Pitsillidis and Kirill Levchenko and Christian Kreibich and Chris Kanich and Geoffrey M. Voelker and Vern Paxson and Nicholas Weaver and Stefan Savage Botnet Judo: Fighting Spam with Itself In *Proceedings of NDSS*, 2010.

[39] K. Chiang and L. Lloyd. A Case Study of the Rustock Rootkit and Spam Bot. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets, 2007.*

[40] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In LEET, 2008.

[41] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In Proceedings of USENIX HotBots07, 2007.

[42] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In Proceedings of USENIX HotBots 07, 2007.

[43] W.T.Strayer,R.Walsh,C.Livadas,andD.Lapsley. Detecting botnets with tight command and control. In Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006.

[44] G. Gu, J. Zhang, and W. Lee. BotSniffer:Detecting botnet command and control channels in network traffic. In Proceedings of the Network and Distributed System Security Symposium, 2008.

[45] Guofei Gu and Roberto Perdisci and Junjie Zhang and Wenke Lee BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection

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

[47] G. Maier, A. Feldmann, V. Paxson, R. Sommer, and M. Vallentin, "An assessment of overt malicious activity manifest in residential networks," in *DIMVA*, 2011.

[48] IBM Tivoli Security Compliance Manager. `www.ibm.com/software/tivoli/`.

[49] Alienvault USM. `www.alienvault.com`.

[50] GFI Languard. `languard.gfi.com`.

[51] HP ArcSight. `www.ndm.net/arcsight-siem`.

[52] BlackStratus LOG Storm. `www.blackstratus.com`.

Chapter 2

Tracking Malware in the Wild: Detection, Validation and Characterization

Elias Raftopoulos[1], Xenofontas Dimitropoulos[1]

[1]Computer Engineering and Networks Laboratory,
ETH Zurich, Switzerland

doi: 10.1145/2068816.2068820

Abstract — Intrusion detection systems (IDSs) produce a large number of alerts, which overwhelm their operators. In this paper, we introduce an IDS alert correlator, which we call Extrusion Detection Guard (EDGe), to detect infected hosts from the on average 3 million alerts per day produced by a deployment of the popular Snort platform in a site with more than 40 thousand unique hosts. EDGe detects several different malware that exhibit a multi-stage behavior and in addition identifies the family and variant of a detected malware, which helps to prioritize and remediate incidents. Our validation shows that only 15% of the infections detected with EDGe are false positives. In addition, our scheme finds 60% more infections and has a lower number of false positives than the most related previous work. Besides, the second part of our paper focuses on characterizing 4,358 infections detected by applying EDGe to a dataset of 832 million IDS alerts collected from an operational network over a period of 9 months. We provide a number of unique insights. We see for example 13.4 new infections per day primarily on client hosts and analyze the exact malware families and variants we detect. Our characterization shows that infections exhibit spatial correlations and open a wide attack vector for 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.

2.1 Introduction

Evaluating and improving network defenses necessitates the use of realistic traces, like IDS alerts, from production networks labeled with information about validated security incidents. Although this is a well-known and long-held problem, presently the community is largely lacking both real-world security data and systematic techniques for evaluating network defenses. Given a database of IDS alerts, it is critical to find and validate security incidents in order to build benchmarks for evaluating network defenses. Motivated by this problem, in this work we introduce a heuristic to detect and propose an approach to validate active infections in our infrastructure. An infection is simply a client or a server with malicious software, which, in our context, leaves a network

trace detectable by an IDS sensor. For example, the malware could be a trojan, worm, spyware, backdoor, etc.

The second problem that motivates our work is IDS false-positive reduction in the context of extrusion detection. Modern malware increasingly involve the user in their propagation by leveraging various social engineering techniques that bypass intrusion prevention measures. For this reason, security administrators need tools for detecting hosts within their network, i.e., *extrusion detection*, that are already infected by malware. Detecting extrusions from IDS alerts bears the challenge of reducing the large number of false positives IDSs are known to generate, e.g., figures of 99% false positives have been reported in the literature [1].

Our first contribution is 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.

Then, we apply EDGe to 832 million alerts collected over a period of 9 months and identify 4,358 different infections on 3,230 out of the of the 40,082 distinct hosts that generated IDS alerts. We provide an extensive characterization study of the extracted infections and make a number of novel observations. First, we characterize the volume, types and impact of infections and make the following observations:

- Out of a total of 40 thousand distinct active hosts we observed during the 9-month period, approximately 8% exhibited signs of infections at least once during their lifetime.

- The lower bound on the probability of infection for a server and a client during a specific day is 0.18% and 0.37%, respectively.

- Infections drastically increase the attractiveness of infected hosts to further inbound attacks.

- A small percentage of hosts are popular sources and targets of attacks. In particular, 5% of the internal hosts account for more than 70% of the total recorded attacks originating from the intranet. In addition, servers are much more preferable targets than clients.
- Healthy hosts closer in terms of IP address distance to infected hosts are much more likely to become infected.
- Our time series analysis shows that server infections are almost independent in time, while client infections are consistently more bursty and this is more evident for aggregation time-scales above two minutes.

Second, we compare key characteristics of different malware families and derive the following key results:

- **Aliveness**: Trojans are the most alive malware in our trace, exhibiting a visible network footprint in the median case for 63% of the days their compromised host was active. The corresponding values for spyware, backdoors, and worms are 52%, 47%, and 40%.
- **Fanout**: We observe that worms in the median case attempt to contact 610 distinct remote hosts during their lifetime, whereas the 30% most active worms hit 1,000 to 10,000 remote hosts. In contrast, backdoors are the most stealthy infection we monitored contacting only 20 hosts in the median case. At the 80-th percentile of the fanout distribution, we see only a marginal increase by a factor of 2.6 in the post infection phase.

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 4.2 we describe the IDS alert traces we used in our experiments. We introduce EDGe in Section 4.3 and describe our validation experiments and results in Section 2.4. Then, we characterize a number of interesting properties of the identified infections in Section 2.5. Finally, we review related work in Section 4.7, we discuss our findings in Section 2.7 and conclude our paper in Section 5.6.

2.2 IDS Data

Our dataset is comprised of raw IDS alerts triggered in the main campus of ETH Zurich by a Snort [3] 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 [4], 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.

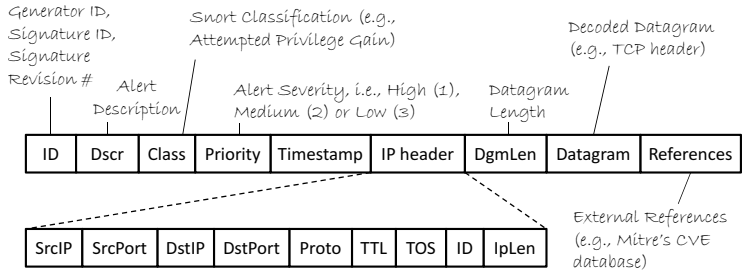


Figure 2.1: *Snort Alert Full Format*

The collected alerts have the standard full Snort format shown in Figure 2.1. For example, the following is an actual high priority alert (with anonymized IP addresses) about a suspected MySQL bot:

```
[**] [1:2001689:7] ET Potential MySQL bot scanning for SQL server [**]
[Classification: A Network Trojan was detected] [Priority: 1]
01/01-22:04:51.319793 aaa.bbb.ccc.ddd:41276 -> xxx.yyy.zzz.hhh:3306
TCP TTL:61 TOS:0x0 ID:14368 IpLen:20 DgmLen:44 DF
******S* Seq: 0xC2A22307  Ack: 0x0  Win: 0x16D0  TcpLen: 24
```

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.

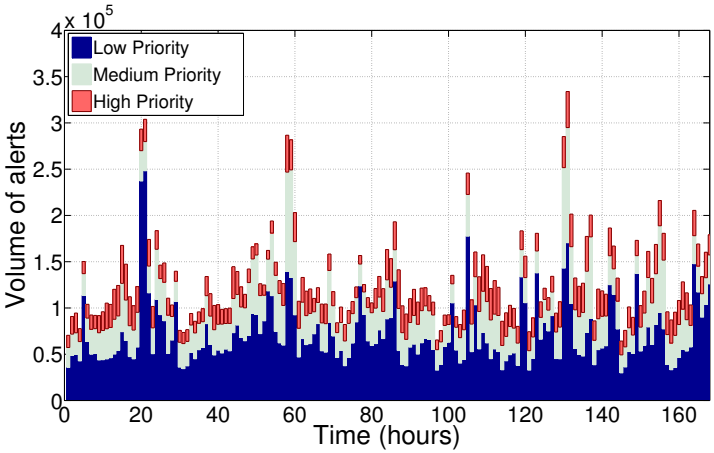


Figure 2.2: *Volume of low, medium, and high priority alerts per hour during a period of a week*

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 81,512 thousand internal IPs. Figure 2.2 illustrates the amount of alerts that we collect during a regular week. 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 portion, approximately 6%, 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.

2.3 Methodology

2.3.1 Alert Bundling

The first challenge that we need to 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 [5], 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 generalize alert IDs that correspond to different flavors of the same malware into a single alert ID by ignoring the version number. We then merge alerts triggered within a short time

window into a single generalized 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%.

2.3.2 Alert Classification

Our dataset includes alerts triggered from 37,388 thousand 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.

Table 2.1: *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

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 2.1 we list the classes for alerts in the `sql.rules` file and provide 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 2.2: *Rulesets and classtypes assigned to the* Compromise *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 [6]). 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 [3,7]. In Table 2.2 we summarize the rulesets and classtypes we used for our *Compromise* 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. For the remaining sections, we only work with alerts of the *Attack* and *Compromise* class.

2.3.3 Malware Detection

A naive approach in identifying infections of internal hosts is to rely on occurrences of *Attack* and *Compromise* related alerts. However, the excessive amount of false positives, makes it very hard to have any level of confidence that we can infer an actual infection using a single alert.

We build EDGe to extract infections based on the following design goals.

- **Keep it simple**: We opt to keep our 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.
- **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.
- **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 hosts within an organization that are already infected. It does not try to proactively prevent an infection.

EDGe: Our approach aims at detecting a recurring multi-stage footprint generated by infected hosts. In the simplest case, 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 significant 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* [8] (for an overview of tuple ranking methods refer to [9]):

$$\textit{J-Measure}(Y;X) = P(X)\big(P(Y|X)\log\frac{P(Y|X)}{P(Y)} + P(\bar{Y}|X)\log\frac{P(\bar{Y}|X)}{P(\bar{Y})}\big), \tag{2.1}$$

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 significant. An internal host that produces at least one significant tuple is considered infected. We fine-tune the threshold to $\frac{0.85}{P(Y)}$ as described in Section 2.4.4 using validated infections from our most reliable source, which is security tickets about infected and remediated systems by our security group. From the set of significant 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_{inf}, 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.

Algorithm 1 Pseudo-code of EDGe for detecting infections

```
Input: Set L of alerts triggered by internal hosts
Result: Significant tuples S_i for internal node i
foreach internal node i do
    foreach hourly timebin T_k do
        foreach tuple (A_i, B_i) in L, triggered in T_k, where A_i ≠ B_i do
            if A_i ⇒ B_i in candidate tuple set R_i then
                R_i.UpdateTupleStats(A_i ⇒ B_i);
            else
                R_i.AddTuple(A_i ⇒ B_i);
            end
        end
    end
    foreach tuple M_i ⇒ N_i in R_i do
        if J-Measure(M_i ⇒ N_i) > J_thresh then
            S_i.AddTuple( M_i ⇒ N_i );
        end
    end
end
```

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_{inf} it is not detected as infected by EDGe. We set the T_{inf} 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 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 [10]. 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_{inf} [11]. Out of the total infections we find in our characterization, 7.4% are re-infections.

2.3.4 Malware Classification

Knowing the malware type of an infection is very useful for prioritizing the reported incidents and facilitating the forensics investigation and remediation, which follows the detection of important malware. In addition, classifying malware detected in the wild is useful for understanding specific patterns of different classes of malware, as we do in Section 2.5.

Building a modern generally-accepted malware taxonomy has not been tackled in the literature beyond textbook taxonomies that date back to the 90s. However, since then malware have undergone significant changes that render their classification challenging. Most malware today exhibit a complex behavior, usually incorporating multiple components allowing them to propagate, communicate with remote hosts to receive commands, automatically update, and initiate the download of additional malicious software. These components are highly modular and extensible. Malware creators use automated tools to incorporate new features to existing malware, improving the provided functionality and making them more resilient and effective in exploiting targeted vulnerabilities. Moreover, the monetization of the malware industry suggests that existing malware components can be customized to meet the needs of cybercriminals. For these reasons, classifying modern malware is an extremely difficult problem to tackle.

Our goal in this section is to make a first step towards solving this problem by introducing a two-level malware taxonomy for Snort. Our taxonomy is largely based on our experience from analyzing a large number of malware in the wild and distinguishes them based on their IDS footprint. Our taxonomy identifies the family and the exact variant of classified malware. We categorize the malicious software we investigate based on its goals and propagation methods into four families, namely *trojans*, *spyware*, *backdoors/bots*, and *worms*, as follows:

- *spyware* are useful pieces of software that are bundled with some hidden fraudulent activity. Typically, they attempt to harvest

Table 2.3: *Trojan infections and associated alerts*

Malware Variant	SID	Signature Description
FakeAV	2012627	ET TROJAN FakeAV Check-in reporting to MSIE with invalid HTTP headers
	2010627	ET TROJAN Likely FakeAVFakeinitFraudLoad Checkin
	2011912	ET CURRENT_EVENTS Possible Fake AV Checkin
	2012725	ET TROJAN Win32/FakeSysdef Rogue AV Checkin
Monkif	2010071	ET TROJAN Hiloti/Mufanom Downloader Checkin
	2008411	ET TROJAN LDPinch SMTP Password Report with mail client The Bat!
	2012612	ET TROJAN Hiloti Style GET to PHP with invalid terse MSIE headers
Simbar	2009005	ET MALWARE Simbar Spyware User-Agent Detected
Torpig	2011365	ET TROJAN Sinowal/sinonet/mebroot infected host Checkin
	2010267	ET TROJAN Sinowal/Torpig Checkin
	2011894	SPYWARE-PUT Torpig bot sinkhole server DNS lookup attempt
	16693	ET TROJAN TDSS/TDL/Alureon MBR rootkit Checkin
	2008660	ET TROJAN Torpig Infection Reporting
Nervos	2802912	ETPRO TROJAN Backdoor.Nervos.A Checkin to Server
	2801671	ETPRO TROJAN BestAntivirus Fake AV Download
Koutodoor	2804717	ETPRO TROJAN Win32/Koutodoor Checkin
MacShield	2012959	ET TROJAN MacShield User-Agent Likely Malware
	2012958	ET TROJAN MacDefender OS X Fake AV Scareware
	2802929	ETPRO TROJAN RogueSoftware.MacOS.MacProtector.A Checkin
	2802870	ETPRO TROJAN RogueSoftware.Win32.MacDefender Buy Screen
Kryptic	2801962	Kryptik/CodecPack.amda/TROJ_RENOS.SM3 Checkin
	2013121	ET TROJAN Win32.VB.OWR Checkin
Comotor	2011848	ET TROJAN Win32/Comotor.A!dll Reporting 2

Table 2.4: *Spyware infections and associated alerts*

Malware Variant	SID	Signature Description
AskSearch	2003494	ET USER_AGENTS AskSearch Toolbar Spyware User-Agent (AskTBar)
	2012000	ET MALWARE ASKTOOLBAR.DLL Reporting
	2003492	ET USER_AGENTS Suspicious Mozilla User-Agent - Likely Fake
	2008052	ET USER_AGENTS Suspicious User-Agent (iexplore)
	2003626	ET USER_AGENTS Suspicious Double User-Agent (User-Agent User-Agent)
Gator	2003575	ET MALWARE Gator/Clarian Spyware Posting Data
SslCrypt	2012862	ET MOBILE_MALWARE SslCrypt Server Communication
HotBar	2802896	ETPRO TROJAN HotbarClickpotato.tv Checkin
	2800945	ETPRO MALWARE Hotbar Spyware Reporting to vic.asp
	2801396	ETPRO MALWARE Hotbar Checkin and Report
Gh0st	2010859	ET TROJAN Gh0st Trojan CnC
Spylog	2007649	ET MALWARE Spylog.ru Related Spyware Checkin
	2008429	ET TROJAN Suspicious User-Agent (HttpDownload)
Yodao	2011123	ET USER_AGENTS Suspicious User-Agent (Yodao Desktop Dict)
QVod	2009785	ET USER_AGENTS QVOD Related Spyware/Malware User-Agent (Qvod)
	2014459	ET P2P QVOD P2P Sharing Traffic detected
Zango	2003058	ET MALWARE Zango Spyware Activity
	8073	SPYWARE-PUT Adware zango toolbar runtime detection
Playtech	2008365	ET USER_AGENTS Suspicious User-Agent (Playtech Downloader)
Gamethief	2012736	ET TROJAN Trojan-GameThief.Win32.OnLineGames.bnye Checkin

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 characterize the four primary and distinct behaviors we observed 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, crimeware, etc. might be appropriate. We found our taxonomy appropriate for characterizing infections detected with a Snort alert correlator.

Table 2.5: *Backdoor infections and associated alerts*

Malware Variant	SID	Signature Description
SDBot	2003494	ETPRO TROJAN Backdoor.Win32.Polybot.A Checkin 1
Zeus	2010861	ET TROJAN Zeus Bot Request to CnC
	2011827	ET TROJAN Zeus related malware dropper reporting in
	2008661	ET TROJAN Zbot/Zeus HTTP POST
	2011827	ET TROJAN Xilcter/Zeus related malware dropper reporting in
Blackenergy	2007668	ET TROJAN Blackenergy Bot Checkin to C&C
	2010886	ET TROJAN BlackEnergy v2.x Plugin Download Request
Parabola	2007626	ET TROJAN Pitbull IRCbotnet Fetch
	2002384	ET TROJAN IRC potential bot commands
Ransky	2002728	ET TROJAN Ransky or variant backdoor communication ping
Avzhan	2002728	ET USER_AGENTS Potential Avzhan DDOS Bot or abnormal User-Agent
SpyEye	2012491	ET TROJAN Spyeye Presto UA Download Request
	2011857	ET TROJAN SpyEye C&C Check-in URI
	2010789	ET TROJAN SpyEye Bot Checkin
Bamital	2802173	ETPRO TROJAN Trojan.Win32.Bamital.F Checkin
	2012299	ET TROJAN W32 Bamital or Backdoor.Win32.Shiz CnC Communication
Tuaye	2008059	ET TROJAN Win32.Inject.ajq Initial Checkin to CnC packet 2 port 443
LibNut	2803032	ETPRO MALWARE Backdoor.Win32.PDFMarca.A Checkin

Table 2.6: *Worm infections and associated alerts*

Malware Variant	SID	Signature Description
Storm	2007701	ET TROJAN Storm Worm Encrypted Variant 1 Traffic
Koobface	2010150	ET TROJAN Koobface HTTP Request
	2014303	ET TROJAN W32/Koobface Variant Checkin Attempt
	19058	SPYWARE-PUT Worm.Win32.Faketube update request attempt
	2009156	ET TROJAN Koobface Checkin via POST
Rimecud	2012739	ET WORM Rimecud Worm checkin
Conficker	2008802	ET TROJAN Possible Downadup/Conficker-A Worm Activity
	2009024	ET TROJAN Downadup/Conficker A or B Worm reporting
	2009205	ET TROJAN Possible Conficker-C P2P encrypted traffic UDP Ping Packet
	2008739	ET TROJAN Conficker Worm Traffic Outbound
Lizamoon	2008802	Potential Lizamoon Client Request
Palevo	2010268	ET TROJAN W32.SillyFDC Checkin
	2001689	ET WORM Potential MySQL bot scanning for SQL server
	2010493	ET SCAN Non-Allowed Host Tried to Connect to MySQL Server

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 tailored to detect traffic patterns which are specific to given malware variants. We only use signatures that incorporate payload based criteria that, based on our analysis,

can be associated with the respective malware variants. 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 are not used in this context.

For example Simbar, which is one of the most prominent trojan infections in our infrastructure, attempts to leak sensitive information about the 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 as shown in Table 2.4, uses the following content rules to detect an infection:

```
content:"User-Agent|3a|"; content:"SIMBAR=";

pcre:"/User-Agent\:[^\n]+\;\sSIMBAR=/H";
```

When an alert is involved in a significant tuple and the respective signature clearly 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 clearly indicates that this signature was triggered by a Simbar specific related activity, therefore, we can safely associate it with Simbar infections. For each family, in Tables 2.3, 2.4, 2.5, and 2.6 we list the set of alerts that we mapped to specific malware variants.

In addition, 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. Using the alert descriptor to associate a detected infection to a malware family is less strict than using payload based evidence. Note, however, that classification using alert descriptors only affects 16% of the classified incidents in our analysis in Section 2.5.

2.4 Validating Infections

In this section we present the process we follow to validate inferred infections and to assess the false positive rate of EDGe. Remotely validating several suspected infections on unmanaged hosts within a production infrastructure is a very challenging problem that to the best of our knowledge has not been previously addressed in the literature. A first challenge is that typically no single tool or information source provides sufficient evidence that an actual security incident has occurred. A second challenge is that the types of malicious behaviors we examine are diverse, ranging from multi-stage attacks and worm propagation events to complex trojan and malware communication patterns.

Our validation follows a three step process. as shown in Figure 2.3. Given a suspected infection, we first extract useful information from six security-related independent information sources about the infected host and the remote hosts it communicates. We refer to this information as *evidence*. A collection of evidence about suspected infections is passed in real-time (e.g., within a day of the first time an infection was detected) to a security expert. The expert correlates the expected behavior of the malware with the collected evidence. If all the evidence agree with the expected behavior, then a positive assessment is made about the suspected infection, otherwise it is concluded that the infection could not be validated, i.e., it is unknown if the suspected host is indeed infected or not. We conservatively consider the latter a false positive.

This process is very demanding and time consuming for the analyst, therefore, we limit ourselves to a subset of the reported infections. Specifically, we analyze 200 consecutive incidents that were reported by EDGe and validate the existence or absence of a variety of malware types. Although, this sample of infections is rather small compared to the total number of infections we report, the analyzed nodes are diverse spanning from servers, to desktop PCs in offices and wireless guests in labs and social areas.

2.4.1 Information Sources

IDS Alerts: For infected nodes we examine the relevant IDS alerts we have collected. We focus on alerts that are triggered in temporal prox-

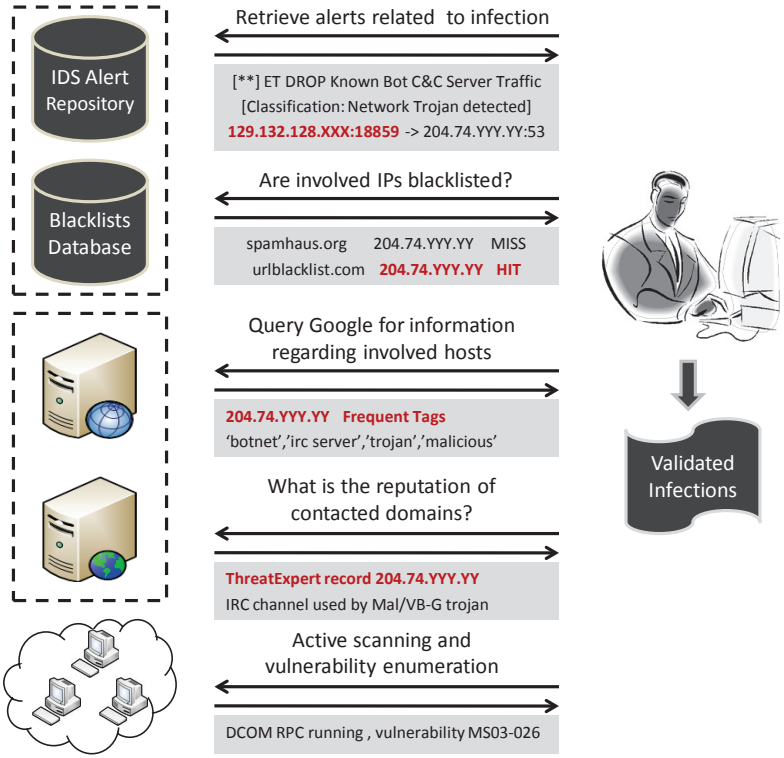


Figure 2.3: *Validation Process*

imity to the infection incident. We evaluate the quality of the alerts based on the following assumption: we consider that oversimplified signatures will tend to generate too many false positives, since it is very likely that they get triggered by benign traffic. On the other hand complex signatures are much more reliable. In order to evaluate the complexity of a signature we check if specific byte sequences within a packet are checked, if the rule specifies the ports, packet size and TCP/IP flags, if previous packets of the same flow are taken into account, and if regular expressions are being used.

Blacklists: We use independent blacklists in order to characterize a suspected host and its communicating remote hosts. We use informa-

tion provided by five publicly available blacklists [12–16] and by one commercial blacklist [17]. We then inspect if an internal node is listed in any of these blacklists within the analyzed tracing period, and if we get a hit we tag the node based on the type of blacklist that generated the hit, e.g., spam or botnet list. Note that due to the rather high percentage of false positives [18] in most reputation-based blacklists, a blacklist hit is insufficient evidence to confirm a suspected infection. It is though a useful indicator that needs to be correlated with additional observations. Moreover, we perform the same blacklist search for external hosts that the analyzed internal machine communicated with. For example communication with hosts within the *Russian Business Network*, a network providing hosting services to cyber-criminals, could signify that the user visits some dangerous websites or that he is redirected to these URLs by an active clickbot [19] or spyware.

Threat Reports: Threat reports are publicly available security logs provided by automated systems [20] or security companies [21, 22] that analyze the behavioral patterns and common actions of a wide range of security threats including worms, trojans, and spyware. They provide a security reputation value for domains based on their observed activity during a specific interval. By investigating threat reports we can identify if a suspected host is contacting URLs that correspond to botnet rendez-vous points or malware landing pages to receive instructions, perform updates or share stolen confidential data.

Web-based Host Profiling: Apart from relying on network traces and threat analysis reports to build a security profile for a suspected host, we also use publicly available data residing on the web, which often provide useful information about the role (type of server, etc.) and involvement of hosts in security incidents [23]. This information originates from several diverse sources such as DNS-lists, website access logs, proxy logs, P2P tracker lists, forums, bulletins, banlists, IRC-lists, etc. In order to retrieve this information we query the Google search engine using as input string the IP of the analyzed host and the respective domain name we get using a reverse-DNS lookup. In an semi-automated fashion we search for tags that reveal possible roles or actions of the host such as 'trojan', 'botnet', 'spam','irc server', 'adserver', 'pop3' and 'webserver'.

Reconnaissance and Vulnerability Reports: Analyzing network based data provides us with rich information regarding the be-

havioral patterns exhibited by a monitored host. However, we do not get any information about the running services, the patching level of critical components, and the existence or absence of vulnerabilities. Naturally, this type of information can be used to validate if an investigated node is susceptible to a specific type of infection or if the set of alerts used to infer the infection correspond to false positives, since they are not relevant to the build and operation of the specific node. Our network security assessment process consists of the following steps:

1. Host Enumeration and Basic Reconnaissance. In this step we use basic reconnaissance techniques such as IP sweeps, NIC whois querying, and TCP/UDP port-scanning in order to identify if a host is reachable and exposed to external attacks. In addition, we determine its role within the infrastructure, such as web, mail, or DNS server.

2. Network Scanning and Probing. In this step we perform targeted network scanning using nmap in order to retrieve detailed information regarding the TCP and UDP network services running on suspicious hosts, details about the OS type and version, and information regarding the types of ICMP messages a host responds to, which reveals its filtering policies and firewall effectiveness.

3. Investigation of Vulnerabilities. After having detected the accessible network services, we investigate the corresponding host for known vulnerabilities. We use publicly available sources [24–26] to identify the existence of exploitable bugs on running services. We augment this assessment with complementary information provided from vulnerability scanners, namely Nessus [27] and OpenVas [28], in order to build a comprehensive profile regarding the vulnerability status of a node.

2.4.2 Security Assessment

To better understand the security assessment process, in the following, we outline a set of frequent infection cases we established during our validation. For each case, we mapped the collected evidence into the behavior that was manifested by a specific malware. The four cases correspond to the four main types of malware we found in our infrastructure, namely backdoors, spyware, worms, and trojans.

Case 1: Backdoor infection. W32/SdBot is a typical family of IRC-controlled trojans with more than 4,000 known variants. It is used by cybercriminals as backdoor in order to gain unauthorized access to a target machine and perform unsolicited actions such as stealing private information or launching active attacks. The typical vulnerabilities we search for when we investigate an SdBot-related infection are the MS-LSASS buffer overflow, the MS-RPC malformed message buffer overflow, and the MS-WebDav vulnerabilities. These are related to the MS network shares services, which are exploited by the trojan to propagate. Regarding its command and control (C&C) activity, an infected host will attempt to use IRC to contact the adversary in order to receive instructions. This communication will trigger alerts with ID within the ranges [2500000:2500500] and [9000077:9000113]. The communicated C&C is typically present in our blacklist or/and profiling data. Additionally, the malware might try to propagate to other subnets. In this case we expect to see extensive scanning activity (mostly on port 445). If a vulnerable host is found and exploited successfully, then the trojan will either attempt to download a version of itself or other additional malware (typically W32/Koobface and Trojan.FakeAV) via ftp.

Case 2: Spyware Infection. The Win32/Hotbar type of malware is one of the most widespread infections in our infrastructure. Most variants appear as a web-browser add-on that provides a seemingly legitimate functionality. However, this malware will clandestinely steal and report user confidential data, like banking information, passwords, browsing habits, etc. For this type of infection, we find IDS alerts with IDs in the range [2003305:2003500]. We trust these alerts as the signatures are quite complex and the malware does not put any effort in disguising. Moreover, the malware operates as clickbot, changing results displayed by search engines and generating pop-ups to redirect the user to potentially malicious websites. These domains are likely to appear in our blacklists or/and web profiling data, usually with tags like ‘malware-hosting’, ‘fraudulent’, and ‘phishing’.

Case 3: Worm Infection. W32/Palevo is the most common malware type found in the rather short list of worm-related infections detected in our infrastructure. It usually spreads automatically using P2P file sharing or Instant Messaging (IM) spam. When investigating this type of infection we expect to see IDS alerts with IDs in the range

[2801347:2801349], which are worm specific, or more generic alerts related to activities complying with a P2P or IM protocol (typically IDs in the ranges [2451:2461], [549:565] and [2008581:2008585]). The worm will attempt to directly contact C&C nodes, without hiding the communication in an IRC channel, using an ephemeral set of port numbers. Commonly, the remote hosts `irc.ekizmedia.com`, `story.dnsentrymx.com`, and `irc.snahosting.net` are contacted. These malicious domains usually appear both in our blacklist data and in our profiling information with tags including 'botnet', 'C&C', 'Rimecud', and 'Mariposa'.

Case 4: Trojan infection. Win32/Monkif is a typical trojan that will attempt to fetch and install malicious software on a victim host. This type of malware is usually bundled with pirated software or is pushed to the victim by using phishing or social engineering attacks. When we investigate this family of infections we expect the host to connect to specific domains (including `www.clicksend.biz` and `stats.woodmedia.biz`) in order to download malicious binaries. These domains are likely to appear in our threat reports as malware hosting and generate tags as 'trojan', 'botnet', 'malware' and 'downloader' in our host profiling results.

The manual security assessment lasted for approximately one month. On a daily basis a security expert was given a list of suspected infections produced by EDGe for the previous day along with a pool of evidence that were extracted in a semi-automated way. The expert thoroughly investigated in total 200 infections. During the first week of the validation process, two experts assessed independently the same suspected infections and then compared, discussed and converged on their assessments.

2.4.3 Validation Results

In Table 2.7 we summarize the number of suspected and verified infections along with the corresponding false positive rate for the four types of infections. We first note that the overall false positive rate is approximately 15%, which is remarkable. 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 *Compromise* are associated with an infection. EDGe helps focus 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.

Table 2.7: *Validated infections for different infection types*

	Reported Incidents	Validated Incidents	False Positive Rate (%)
Trojans	97	85	12.3
Spyware	66	59	10.6
Worms	9	8	11.0
Backdoors	28	18	35.0

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

DNS Servers. First, we find that DNS servers within our infrastructure frequently trigger signatures from the *Compromise* class. The reason is that they often attempt to resolve domains that are considered malicious. These DNS requests trigger signatures that check the destination IP address and compare it against a list of known compromised hosts. An alert will be raised in this case, typically with IDs in the range [2500000:2500941], which corresponds to backdoor related activity. DNS related false positives are mainly responsible for the inflated value regarding backdoors false positive rate shown in Table 2.7.

Skype Supernodes. Second, 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 it happens that a remote Skype user connecting to a local supernode is blacklisted, then Snort will trigger an alert identifying this system as malicious. This communication is persistent and frequent since whenever a Skype client attempts to initiate a communication, it will access a distributed database provided by supernodes in order to get the details of the contacted peer.

Antivirus. Third, a specific antivirus program generates IDS alerts of the class *Compromise* while updating. The triggered signatures check for known patterns of malicious activity found on the payload of the

transmitted packets. It appears that the updates of this antivirus contain the actual pattern that it attempts to detect in plain format.

Online Games. Finally, we have observed that certain types of online games generate Snort alerts with IDs in the ranges [2003355:2003626] and [2510000:2510447]. In the case of browser-based games the triggered signatures suggest that there is an ongoing spyware-related activity. The reason is that the corresponding websites exhibit a behavior that is very similar to clickbots, attempting to redirect the player to 3rd party, potentially malicious, websites for profit. In the case of standalone 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 originate from domains which are blacklisted, and therefore an alert is raised.

It is possible to further increase the detection accuracy of EDGe 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.

2.4.4 Fine-tuning EDGe

As discussed in Section 2.3.3 an important parameter of EDGe is the *J-Measure* threshold that determines if a specific tuple will be tagged as an active infection. In order to adjust this threshold we performed the discussed validation process on an additional small set of nodes in the local subnet of our institute. The duration of this training phase was two weeks and occurred chronologically before the main validation. During this period we run EDGe using variable *J-Measure* threshold values and evaluated its inference results.

For the local subnet of our institute we were able to use very reliable information sources to validate a small number of detected infections. In particular, for a set of nodes we physically visited their owners and verified an infection either by performing an on the spot assessment of a system or by receiving a confirmation from an owner aware that her system was indeed infected. Secondly, our second very reliable information source for our local subnet is security tickets of our IT team. These

are logged events about security incidents that have been detected, assessed, and remediated.

Using information for 28 systems we adjusted the *J-Measure* threshold in a conservative manner, i.e., aiming at keeping the false positives as low as possible, but without increasing the false negatives significantly. Selecting a threshold of $\frac{0.85}{P(Y)}$ achieved a good tradeoff, limiting the false positive rate to below 10% and the false negative rate to below 23%. For threshold values below $\frac{0.80}{P(Y)}$ the corresponding false positive rate increases above 24%, whereas for threshold values above $\frac{0.90}{P(Y)}$ we miss more than 32% of active infections.

2.4.5 Comparison with State-of-the-Art in IDS Alert Correlation

A large number of previous studies, e.g., [29–32], have focused on detecting botnet-type infections using passive monitoring methods, by comparing observed flow-level features and a derived botnet communication model. 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 the one used by 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 within the trace.

Closer to our work, Bothunter [2] uses dialog correlation to associate IDS alerts to a predefined malware infection dialog model. In this approach malware infections are modeled as loosely ordered IDS-alert sequences, triggered during communication of an internal suspicious host with several external contacted 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. Our correlator 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 loose, 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, which might seem unconventional. However, these scenarios 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/SdBot presented in Section 2.4.2. 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 purely badly written signatures. EDGe uses the *J-Measure* to effectively balance the frequency and the confidence of alert tuples.

Bothunter is a closely-related state-of-the-art IDS alert correlator that is publicly available. Therefore, we select to evaluate its performance in our operational network and compare it against our approach. We deployed version 1.7.2 of Bothunter, using the latest signature set provided by [33]. 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 [33], which were used during the 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 Bothunter Snort plugins operating on packet traces, were disabled. This way we can compare on common grounds the two IDS alert correlators.

Table 2.8: *Comparison between proposed detection method and Bothunter*

EDGe	Reported Incidents	Validated Incidents	Missed Incidents	False Positive Rate (%)	False Negative Rate
Trojans	97	85	2	12.3	0.02
Spyware	66	59	10	10.6	0.14
Worms	9	8	3	11.0	0.27
Backdoors	28	18	7	35.0	0.28
Bothunter	Reported Incidents	Validated Incidents	Missed Incidents	False Positive Rate (%)	False Negative Rate
Trojans	48	38	49	20.8	0.56
Spyware	41	36	33	12.2	0.47
Worms	8	5	6	37.5	0.54
Backdoors	28	23	4	17.8	0.14

In Table 2.8 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 significantly 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 incorporated to our validation set. To validate these new infections we used the process described in 2.4.2.

Considering all incidents, our detector generates fewer false positives (15%) compared to Bothunter (18.4%). In addition, we take into account the respective infection type. 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 higher 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 negative rates are a lower bound, since additional infections that are not detected by neither detector might exist. They are useful for comparison purposes. In Table 2.8 we see that the false negative rate 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. However, Bothunter performs much 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 exhibiting a significantly lower false negative rate. 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 combined to improve IDS-based malware detection.

2.5 Characterizing Infections

2.5.1 Volume, Types and Impact of Infections

The first interesting finding, illustrated in Figure 2.4, is that on a daily basis from an average of 10,124 active hosts, we detect on average 15.8 new infections. Taking into account the 15% of false positives 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 0.15%, whereas the corresponding value for clients is 0.31%.

The lower probability of a server infection can be attributed to two

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

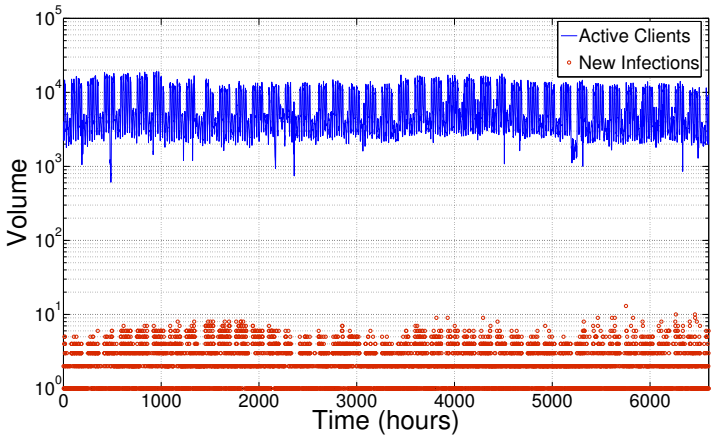


Figure 2.4: *Active clients and new infections time-series*

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 2.4 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

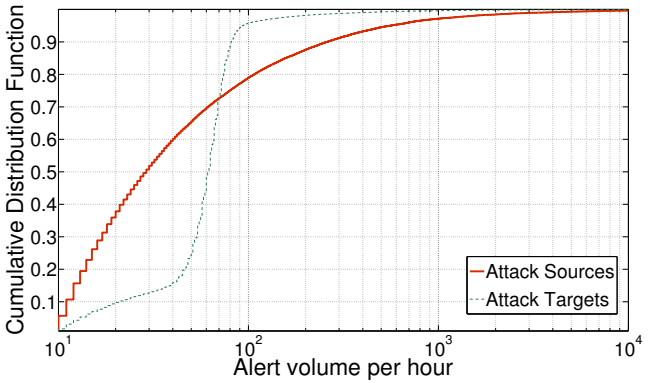
based on the methodology of Section 2.3.2. Out of the total of 4,358 infections we can classify based solely on the tuples mined by EDGe 78% into the malware families of Section 2.3.2. In addition, for 62% of the detected malware (2,712 incidents) we can identify the exact malware variant. In Table VIII 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	FakeAV	120
	Monkif	7
	Simbar	252
	Torpig	28
	Nervos	13
	MacShield	12
	Kryptic	15
	Comotor	5
	Koutodoor	3
Backdoors	Zeus	8
	Blackenergy	4
	Parabola	21
	Ransky	33
	Avzhan	80
	SpyEye	9
	Bamital	3
	LibNut	5

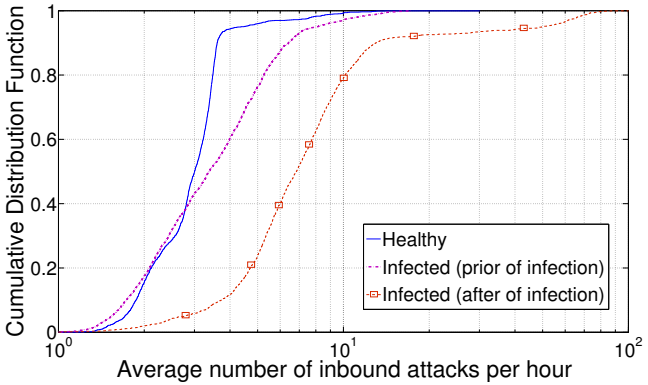
Family	Variant	# Infections
Spyware	AskSearch	722
	Gator	5
	SSLCrypt	19
	HotBar	311
	Gh0st	4
	Spylog	139
	Yodao	88
	Qvod	93
	Zango	441
	Gamethief	2
	Playtech	206
Worms	Koobface	2
	Rimecud	3
	Conficker	23
	Lizamoon	9
	Palevo	25
	Storm	2

Table 2.9: *Prevalence of malware families and variants detected by EDGe*

Heavy Hitters: We count for each internal host within the monitored infrastructure the hourly average number of alerts of type *Attacks* in the inbound and outbound directions. In Figure 2.5a, 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



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



(b) Infection impact on the number of inbound attacks

Figure 2.5: *Heavy Hitters and Prominent Attack Targets*

of possibly vulnerable hosts. These hit-lists are generated using either scanning or by employing web-spiders and DNS-searches [34]. 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**. These are highly suspicious nodes that require additional investigation. 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 *Attacks* 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 2.5b, 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 more than double, i.e., it is equal to 7. 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. However, in the case of 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 significantly**, 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 2.5b.

2.5.2 Differences Among Malware Families

Impact of Infections on Outbound Alerts: We further analyze the impact of infections on the malicious behavior exhibited by internal hosts. Figure 2.6 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 generated 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. However, in the tail of the distribution we see a much more prominent 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 in the infection phase, and only rarely communicate with their C&C to receive instructions and binary updates.

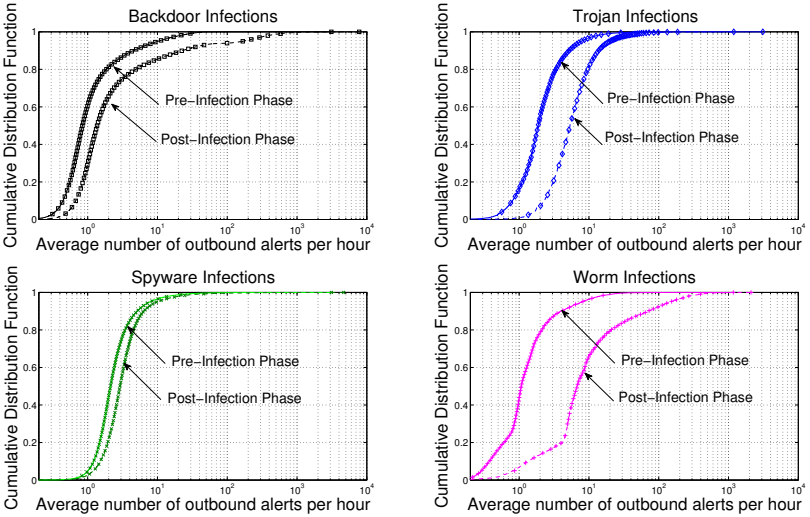


Figure 2.6: *Infection impact on the number of outbound alerts for different infection families*

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 four in the average number of outbound alerts. This is due to

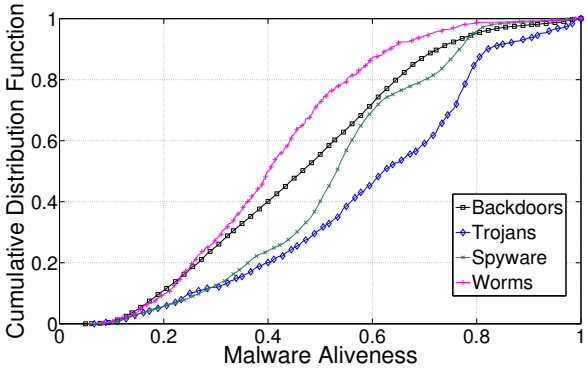
the fact that trojans frequently attempt to install additional malicious binaries from remote domains. This is consistent with recent studies, e.g., [35], that highlight the commodization of malware distribution and the predominance of pay-per-install services.

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. This shows that users who install spyware are prone to visiting low reputation or malicious domains.

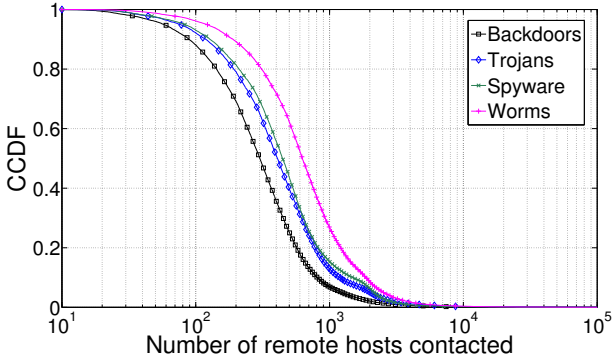
Worms exhibit the most evident increase in the outbound activity after an infection. Specifically, we see that 90% of the infected machines generate on average at least 80 severe alerts per hour throughout their lifetime. This is due to the extensive reconnaissance and scanning activity that is used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

Malware Aliveness: An interesting question is 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 2.7a we illustrate the CDF of the aliveness of different malware families.

We clearly see that trojans are the most alive malware family. 6% of the trojan infections have an aliveness value above 0.9, indicating that whenever the compromised host is online the malware will consistently generate malicious activity. In the median case the aliveness value is still quite high and equal to 0.63, suggesting that trojans rarely attempt to go stealthy by suspending their activity. The reason for this is the diversity in stages undergone by trojan infections during their lifetime. They typically attempt to update their binary, contact a remote con-



(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

Figure 2.7: *Malware Aliveness and Fanout*

troller to report, and download additional malicious eggs to install on the compromised host. 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. The reason is that they 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 actual malicious activity, 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 machines exhibit an aliveness value of 0.4 compared to the 0.63 value for trojans. 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: Further, we investigate the number of distinct remote IP addresses to which malware communicate. In Figure 2.7b we illustrate the number of remote hosts contacted by internal compromised machines 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 the infected system.

Backdoor infected hosts initiate this communication to contact their C&C. The most prominent backdoor infections in our trace such as Avzhan, Parabola and Zeus use a predefined set of rendez-vous 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 1000 remote hosts and this number can reach 3,146 contacts in the maximum case. 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 exhibit significantly higher number of initiated contacts compared to backdoor infections. Specifically we see that in the median case trojan infections initiate communication 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 *Compromized*. These are reconnaissance attempts in the form of active scanning in order to detect vulnerable hosts in the wild to propagate and active exploitation attacks attempting to compromise potential victims. 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 equal to 10,625. Note that we have seen that the same comparative qualitative characteristics among malware types also hold in finer time scales.

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 2.8 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 aforementioned cycle of malicious actions within short time

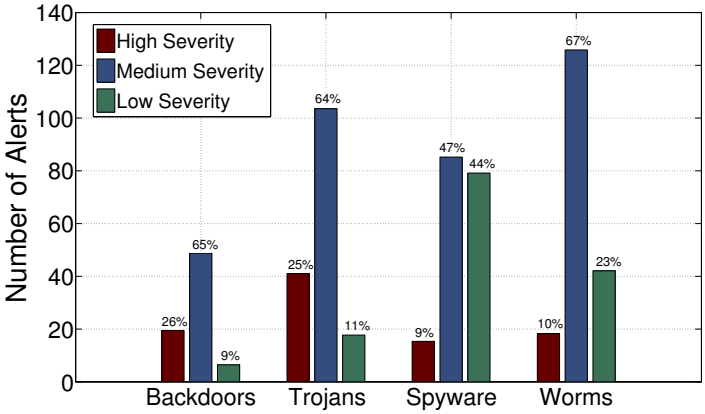


Figure 2.8: *Daily volume of alerts for different severity levels and malware families*

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 activity.

2.5.3 Correlations Across Space and Time

Spatial Correlations: The infections we observe exhibit strong spatial correlation. 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 2.9, 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

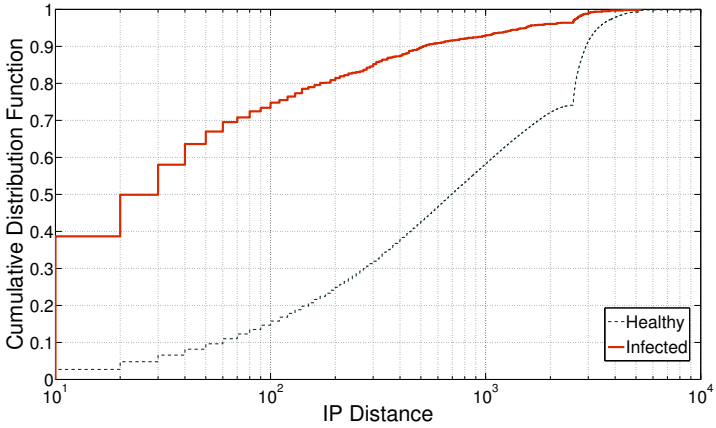


Figure 2.9: *Infections spatial correlation*

are likely located in the same subnet. The corresponding 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.

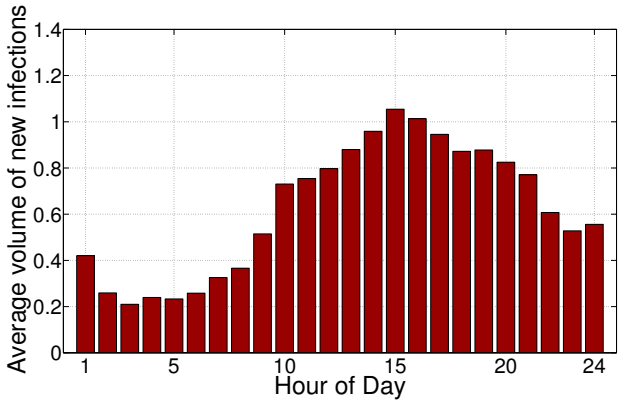
Correlations Across Time: The distribution of the time when infections outbreak exhibits a diurnal pattern as illustrated in Figure 2.10a, where we see that most infections occur during working hours. This is due to the fact that the activity of client nodes, where most infections outbreak, exhibits 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 [36] of the infection time series at different scales. The Allan deviation is given by the following equation:

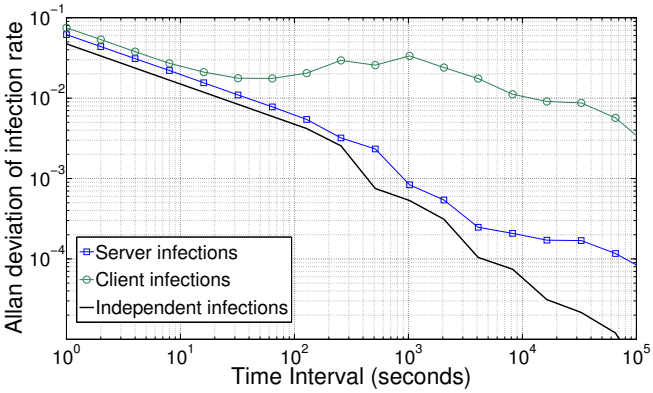
$$\sigma_x^2(\tau) = \frac{1}{2}\langle(\Delta x)^2\rangle \tag{2.2}$$

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.



(a) Distribution of infections for different hours of day



(b) Infections burstiness for clients and servers

Figure 2.10: *Infections temporal correlations*

In Figure 2.10b, 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 2.10b 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 are consistently more bursty and this is more evident for time-scales above two minutes**.

2.6 Related Work

IDS Evaluation: The DARPA dataset [37] remains today one of the best options, although it dates back to 1999 and has several well-known shortcomings [38, 39]. Another well-known dataset, the DARPA Correlation Technology Validation [40] was created in 2002 but unfortunately is not anymore available. These datasets were created in testbeds under controlled experiments. A lot of research has focused on generating synthetic traffic. More related to our work, MACE [41] is an environment for generating malicious packets for evaluating IDSs in testbeds. In addition to testbed experiments, in this work we stress the need to use and label traces collected in the wild.

Intrusion Measurements: Large traces of intrusion data, like IDS alerts and firewall logs collected by DShield [16], have been analyzed in previous studies. In particular, Yegneswaran *et al.* [42] made a number of observations regarding the distribution, types, and prevalence of intrusions. In addition, they projected the global volume of intrusions and estimated the potential of collaboration in intrusion detection. Kati *et al.* [43] analyzed a large trace of IDS alerts, reported characteristics of correlated attacks, and investigated how to effectively collaborate in intrusion detection. In this work we provide a number of further insights about intrusions focusing specifically on infections, which have not been studied as a separate class in the past.

Another group of studies analyze security incidents in the wild using alternative data sources. Most related to our work Sharma *et al.* [44] analyzed 150 security incidents that occurred in a supercomputing center over five years using data from five security sensors. Maier *et al.* [45] 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.* [46] 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.

Besides, a number of previous studies have focused on IDS alert correlation and aggregation. These studies evaluate proposed solutions on a small number of testbed-based benchmarks, like the DARPA dataset, and are tailored for general-purpose alert analysis rather than for extrusion detection. In our work, we highlight the need for using and analyzing measurements from real networks in addition to testbed-based evaluation methods. In particular, related work on alert correlation and aggregation can be classified in three categories.

Statistical/temporal Alert Correlation: A group of studies explores statistical [47, 48] or temporal [49] alert correlations to identify causality relationships between alerts. Statistical correlation methods estimate the association between different alerts by measuring the co-occurrence and frequency of alert pairs within a specific time frame. Qin [47] introduced a Bayesian network to model the causality relationship between alert pairs, while Ren *et al.* [50] proposed an online system to construct attack scenarios based on historic alert information. Temporal-based correlation approaches perform time series analysis on the alert stream to compute the dependence between different alerts. Qun and Lee [49] generate time series variables on the number of recorded alerts per time unit and use the Granger causality test to identify causal relationships. We also use a statistical alert correlation test in our heuristic and show how alert correlation can be useful specifically for extrusion detection.

Scenario- and Rule-based Alert Correlation: On the other hand, a number of studies hardcode details about attack steps into full scenarios [51, 52] or into rules [53–55], which are used to identify, summarize, and annotate alert groups. Scenario-based correlation approaches try to identify causal relationships between alerts in order to reconstruct high-level multi-stage events. Most approaches rely on attack scenarios specified by human analysts using an attack language [56,57]. Deriving attack scenarios for all observed attacks requires significant technical expertise, prior knowledge, and time. In order to automate the scenario derivation process, machine learning techniques have been used [58]. Rule-based approaches are based on the observa-

tion that attacks can be usually subdivided into stages. These methods attempt to match specific alerts to the prerequisites and consequences of an active attack stage. The idea is to correlate alerts if the precondition of the current alert stream is satisfied by the postcondition of alerts that were analyzed earlier in time. A main limitation of scenario- and rule-based approaches is that the detection effectiveness is limited to attacks known a priori to the analyst or learned during the training phase. Therefore, they are not useful against novel attacks that have not been encountered in the past.

Alert Aggregation: The goal of alert aggregation is to group alerts into meaningful groups based on similarity criteria, making them manageable for a human analyst and amenable for subsequent statistical analysis. Each new alert is compared against all active alert sequences and is associated with the most relevant one. The set of attributes used to compute the corresponding similarity measure are diverse spanning from inherent alert features such as IP addresses, port numbers, and alert signatures, to topological and contextual information about the monitored network, such as the role and operation of nodes that triggered the alert or the deployed operating system and services. The work by Valdes *et al.* [5] represents each alert as a vector of attributes and groups alerts based on the weighted sum of the similarity of different attributes. The weight of an attribute is heuristically computed. Dain *et al.* [59,60] propose a system that associates incoming alerts to groups in an online fashion. Julisch [61] proposes a clustering technique that aims at grouping alerts sharing the same root-causes, based on attribute-oriented induction. To address the problem that prior knowledge regarding how the alerts should be aggregated might not be available, machine learning techniques have been used. Zhu *et al.* [62] propose a supervised learning approach based on neural networks. Compared to these studies, we also use a simple form of alert aggregation in our heuristic, which we call alert bundling, that groups spurts of almost identical alerts for further statistical analysis rather than potentially diverse alerts based on complex similarity functions.

2.7 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. This means that 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, to assess false-negative rates one can use synthetic or testbed-based evaluation traces, as discussed in Section 4.7, 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 2.5 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.

2.8 Conclusions

In this paper, we present a novel approach to identify active infections in a large population of hosts, using IDS logs. We tailor our heuristic based on the observation that alerts with high mutual information are very likely to be correlated. Correlated alerts of specific types reveal that an actual infection has occurred. By applying this heuristic to a large dataset of collected alerts, we find infections for a population of more than 91 thousand unique hosts. We perform an extensive validation study in order to assess the effectiveness of our method, and show that it manages to reduce the false-positive rate of the raw IDS alerts to only 15%. Our characterization suggests that infections exhibit high spatial correlations, and that the existing infections open a wide attack vector for 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 types of malware and highlight several key differences in the volume, aliveness, fanout, and severity of the alerts. We believe that our results are useful in several diverse fields, such as evaluating network defenses, extrusion detection, IDS false positive reduction, and network forensics.

2.9 Acknowledgements

The authors wish to thank Prof. Bernhard Plattner and Dr. Vincent Lenders for their invaluable help and fruitful discussions. Furthermore, we would like to thank Matthias Eggli for helping with the validation of the infection incidents. 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.

Bibliography

[1] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," in *KDD'02*, 2002.

[2] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation," 2007.

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

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

[5] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, 2001, pp. 54–68.

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

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

[8] 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.

[9] 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.

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

[11] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "Is host-based anomaly detection + temporal correlation = worm causality?" 2007.

[12] "Projecthoneypot web page," www.projecthoneypot.org, 2013.

[13] "Foundation web page," www.shadowserver.org, 2006.

[14] "The Urlblacklist web page," www.urlblacklist.org, 2000.

[15] "Anonymous postmasters early warning system," www.apews.org, 2006.

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

[17] "The Spamhaus Project," www.spamhaus.org, 1998.

[18] S. Sinha, M. Bailey, and F. Jahanian, "Shades of grey: On the effectiveness of reputation-based blacklists," in *MALWARE '08*, 2008.

[19] N. Daswani and G. Inc, "The anatomy of clickbot.a," in *USENIX Hotbots'07*, 2007.

[20] "Advanced automated threat analysis system," www.threatexpert.com, 2003.

[21] "TrustedSource Internet Reputation System," www.trustedsource.org, 2006.

[22] "Botnet and Advanced Malware Detection and Protection," www.damballa.com, 2006.

[23] I. Trestian, S. Ranjan, A. Kuzmanovi, and A. Nucci, "Unconstrained endpoint profiling (googling the internet)," ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 279–290.

[24] "Common Vulnerabilities and Exposures dictionary of information security vulnerabilities," 2007.

[25] "SecurityFocus technical community," www.securityfocus.com, 2002.

[26] "Packet Storm Full Disclosure Information Security," packetstormsecurity.org, 1999.

[27] "The Nessus vulnerability scanner," www.tenable.com/products/nessus, 2003.

[28] "The Open Vulnerability Assessment System," www.openvas.org, 2005.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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.

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

[37] 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.

[38] M. John, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations," *ACM Trans. Inf. Syst. Secur.*, vol. 3, pp. 262–294, November 2000.

[39] C. Brown, A. Cowperthwaite, A. Hijazi, and A. Somayaji, "Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict," ser. CISDA'09, NJ, USA, 2009, pp. 67–73.

[40] J. Haines, D. K. Ryder, L. Tinnel, and S. Taylor, "Validation of sensor alert correlators," *IEEE Security and Privacy*, vol. 1, pp. 46–56, January 2003.

[41] J. Sommers, V. Yegneswaran, and P. Barford, "A framework for malicious workload generation," ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 82–87.

[42] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: global characteristics and prevalence," ser. SIGMETRICS '03. New York, NY, USA: ACM, 2003, pp. 138–147.

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

[44] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. K. Iyer, "Analysis of security data from a large computing organization," in *DSN*, 2011, pp. 506–517.

[45] 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.

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

[47] X. Qin, "A probabilistic-based framework for infosec alert correlation," Ph.D. dissertation, 2005.

[48] F. Maggi and S. Zanero, "On the use of different statistical tests for alert correlation: short paper," in *RAID'07*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 167–177.

[49] X. Qin and W. Lee, "Statistical causality analysis of infosec alert data," in *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection*, 2003, pp. 73–93.

[50] H. Ren, N. Stakhanova, and A. A. Ghorbani, "An online adaptive approach to alert correlation," ser. DIMVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 153–172.

[51] B. Morin and H. Debar, "Correlation of intrusion symptoms: an application of chronicles," in *RAID03*, 2003, pp. 94–112.

[52] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," ser. RAID '00. London, UK: Springer-Verlag, 2001, pp. 85–103.

[53] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *CCS*, 2002, pp. 245–254.

[54] F. Cuppens and A. Miège, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, USA, 2002.

[55] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multistep cyber attacks for scenario recognition," 2003.

[56] F. Cuppens and R. Ortalo, "Lambda: A language to model a database for detection of attacks," ser. RAID '00. London, UK: Springer-Verlag, 2000, pp. 197–216.

[57] S. Eckmann, G. Vigna, and R. A. Kemmerer, "Statl: An attack language for state-based intrusion detection," 2002.

[58] D. Curry and H. Debar, "Intrusion detection message exchange format: Extensible markup language document type definition," 2003.

[59] O. Dain and R. Cunningham, "Building scenarios from a heterogeneous alert stream," 2002.

[60] Oliver Dain, "Fusing a heterogeneous alert stream into scenarios," in *In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, 2001, pp. 1–13.

[61] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Transactions on Information and System Security*, vol. 6, pp. 443–471, 2003.

[62] B. Zhu and A. A. Ghorbani, "Alert correlation for extracting attack strategies," *I. J. Network Security*, vol. 3, no. 3, pp. 244–258, 2006.

Chapter 3

Shedding Light on Log Correlation in Network Forensics Analysis

Elias Raftopoulos[1], Matthias Egli[1], Xenofontas Dimitropoulos[1]

[1]Computer Engineering and Networks Laboratory,
ETH Zurich, Switzerland

doi: 10.1007/978-3-642-37300-8_14

Abstract —

Presently, forensics analyses of security incidents rely largely on manual, ad-hoc, and very time-consuming processes. A security analyst needs to manually correlate evidence from diverse security logs with expertise on suspected malware and background on the configuration of an infrastructure to diagnose if, when, and how an incident happened. To improve our understanding of forensics analysis processes, in this work we analyze the diagnosis of 200 infections detected within a large operational network. Based on the analyzed incidents, we build a decision support tool that shows how to correlate evidence from different sources of security data to expedite manual forensics analysis of compromised systems. Our tool is based on the C4.5 decision tree classifier and shows how to combine four commonly-used data sources, namely IDS alerts, reconnaissance and vulnerability reports, blacklists, and a search engine, to verify different types of malware, like Torpig, SbBot, and FakeAV. Our evaluation confirms that the derived decision tree helps to accurately diagnose infections, while it exhibits comparable performance with a more sophisticated SVM classifier, which however is much less interpretable for non statisticians.

3.1 Introduction

Computer Security Incident Response Team (CSIRT) experts use a combination of intuition, knowledge of the underlying infrastructure and protocols, and a wide range of security sensors, to analyze incidents. Although, the low-level sensors used provide a source of fine-grained information, often a single source is not sufficient to reliably decide if an actual security incident did occur. The process of correlating data from multiple sources, in order to assess the security state of a networked system based on low-level logs and events is complex, extremely time consuming and in most parts manual. Although, thorough manual investigation is critical in order to collect all the required evidence for a detected breach and to make a definite assessment regarding the severity of an investigated incident, it would be highly beneficial for administrators to have tools that can guide them in the log-analysis process, helping them to diagnose and mitigate security incidents.

A large number of previous studies have analyzed how to aggre-

gate, correlate, and prioritize IDS alerts. A survey can be found in [1]. However, aggregated IDS alerts are then passed to a security analyst for manual diagnosis, which is a complex, typically ad-hoc process that leverages multiple security sources, like blacklists, scanning logs, etc. In this work we focus on this process with the goal of understanding how to correlate *multiple* security data sources and how to expedite manual investigation.

For this purpose, we conduct a complex experiment turning a human security analyst into the subject of our analysis. We systematically monitor the used evidence and the decisions of an analyst during the diagnosis of 200 security incidents over a period of four weeks in a large academic network. Based on the analyzed incidents, we build a decision tree using the C4.5 algorithm that reflects how low-level evidence from the four security sources can be combined to diagnose different families of malware, like Torpig, SbBot, and FakeAV. The derived model is useful for expediting the time-consuming manual security assessment of security incidents. It accurately encodes a large part of the decisions of the analyst in correlating diverse security logs and can serve as a decision support tool helping an analyst identify the most critical features that suggest the presence of an infection. In addition, we show that using the decision tree for fully-automated classification correctly identifies infections in 72% of the cases.

Finally, we ask the question if other state-of-the-art classifiers exhibit better performance than a C4.5 decision tree, which is highly interpretable and therefore useful as a decision support tool. We compare its detection accuracy with a support vector machine (SVM), a Bayesian tree classifier (BTC), and a tree-augmented naive Bayes (TAN). We find that a C4.5 decision tree is better than BTCs and TANs and only slightly worse than the more sophisticated SVM, which however is much less interpretable.

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

- We outline a number of features useful for security assessment that can be extracted from four commonly-used data sources.
- We build a decision support tool that clearly depicts how evidence from four sources should be correlated to diagnose different types of malware. We show that our decision tree is 72% accurate in automatically classifying suspected infections.

- We compare our decision tree with other classifiers and show that state-of-the-art SVMs, which are more sophisticated but much less understandable, have only slightly better performance.

In the next section we describe in detail the data and features we used. In Section 4.3 we provide a brief summary of our experiment. Next, in Section 3.3.1 we present our decision support tool and in Section 3.3.2 we compare its performance to other alternatives. Finally, in Section 5.5 we discuss related work and we conclude in Section 5.6.

3.2 Data Collection and Feature Extraction

In this section, we review in detail the four data sources we used for security investigation and the features we extracted from each source. We start with a brief description of the monitored infrastructure from which we collect data.

We conducted all our experiments in and collected data from the network of the main campus of the Swiss Federal Institute of Technology at Zurich (ETH Zurich). During our experiments, which spanned four weeks, we observed in total 28,665 unique internal hosts. The host population is diverse including standard desktop PCs in labs and offices, laptops, handheld devices, and more critical systems such as mail and web servers. The application mix of the monitored network is also diverse since users can freely install software on their systems.

We evaluate four commonly-used security data sources. First, we collect IDS alerts generated by a Snort IDS sensor, which is located between the primary border router and the network firewall of the monitored network. The IDS sensor monitors all upstream and downstream traffic and generates an alert when a malicious signature is triggered. Snort gives us a view of malicious activities from the gateway of the network. To collect information related to services and applications running on internal hosts, we perform reconnaissance and active scanning using NIC *whois* querying and NMap. After mapping the accessible network services of a host, we investigate for known vulnerabilities using the Nessus [5] and OpenVas [6] vulnerability scanners. These four tools, to which we collectively refer as reconnaissance and vulnerability scanners, give us a more detailed view of internal end-hosts. The last

two data sources help us extract information about remote hosts by leveraging publicly available data. In particular, we use blacklists from five blacklist providers covering different types of malicious hosts, from active attackers to spammers, and we query the Google search engine for suspected hosts and domains. The search engine indexes a variety of sources, including public forums, bulletins, and banlists, which we exploit in an automated way by searching for security-related tags in the query output that reveal possible roles or actions.

3.2.1 IDS Alerts

Our Snort data is comprised of raw IDS alerts triggered by a Snort sensor [7] that monitors all the upstream and downstream traffic through the main border link of the network of the main campus of ETH Zurich. The sensor is configured with the official Snort signature ruleset and the Emerging Threats (ET) ruleset [8], which are the two most commonly-used Snort rulesets. As of April 2011 the two rulesets have in total 37,388 distinct signatures. We analyzed in total 37 million alerts generated during the four week period.

We use IDS alerts in two ways. First, we detect infected hosts within the monitored network based on the Snort alert correlation heuristic we developed in our recent work [2]. Our heuristic first aggregates similar raw Snort alerts produced in close temporal proximity, then it classifies aggregated alerts into three classes, namely *Attacks*, *Compromised hosts*, and *Policy violations*, and finally it infers infections within a monitored network by detecting hosts that produce a recurring multi-stage pattern involving alerts of the first two classes. The heuristic returns the IP address of a detected infection and an associated timestamp. We apply our heuristic on our IDS alerts and use the detected infections as the input to the manual security assessment process we focus on in this work.

In principle, any detector that finds a suspicious host could form the input to our methodology. For this reason, our findings clearly do not hold for all possible malware, but for a variety of important malware detected with our heuristic.

Secondly, we use the IDS alerts as a data source that can be further exploited for manual security assessment of a suspected host. Given an IP address and a timestamp of a detected infection, we retrieve

the aggregate IDS alerts of the classes *Attack* and *Compromised host* that have been observed within 24 hours before or after the timestamp. From the aggregate IDS alerts we extract the following features we found useful for manual diagnosis:

- *Suspicious remote hosts*: If the communication between a suspected local host and a remote host triggers a large number of aggregate alerts we consider the corresponding remote host suspicious and mark its IP address for further investigation. We consider that the number of generated alerts is significant if its fraction compared to the total number of aggregate alerts generated within the examined interval is above 10%. This feature is useful to identify common malicious domains used by infected machines in order to receive instructions, share data, or update their malicious binary.

- *Suspicious remote services*: If the contribution of aggregate alerts generated within the analyzed interval that target a specific remote port is above 10%, then we consider this service suspicious. We aggregate the activity of all non-privileged ports (port numbers above 1024) into a single port with label *High*. This feature is useful to identify internal infections that attempt to attack multiple external nodes, such as in the case of worms performing a distributed scan for vulnerable services or in the case of spamming bots.

- *Suspicious local services*: If a local service port is involved in more than 10% of the aggregate IDS alerts, then we consider it suspicious. Again, we aggregate the activity of all non-privileged ports into a single port with label *High*. This feature is useful to detect malware that attach to popular software such as IE, Firefox, Skype, or CuteFTP.

- *Count of severe alerts*: We count the absolute number of aggregate alerts of the classes *Attack* and *Compromised* that were triggered within the investigated time interval. This feature is important in order to detect malware that generate spurts of high severity alerts. It helps to distinguish high activity malware from more stealthy ones.

- *Infection duration*: We compute the time in hours that elapsed between the first and the last triggered alert within the observed daily interval. We only take into account hourly slots where at least one alert from any class, including policy alerts, was triggered. This feature enables to normalize the volume of alerts of a suspected host over time.

- *Common severe alerts*: If a specific alert accounts for more than 5% of total number of aggregate alerts, then we build a new feature for that alert ID. This is a boolean feature, with value 1 if the alert count for the corresponding ID is above the 5% threshold and 0 otherwise. This feature targets malware that have a very consistent network footprint triggering always the same set of IDS alerts.

On Table 3.1 we illustrate the IDS alert features extracted for a host that has been infected with the Torpig trojan. Note that IP addresses have been anonymized.

Table 3.1: *Example features extracted from Snort alerts for an investigated internal host.*

Feature	Value
Suspicious remote hosts	{a.b.c.d , k.l.m.n}
Suspicious remote services	{80}
Suspicious local services	{80,135,443}
Count of severe alerts	271
Infection Duration (hours)	23
Common severe alerts (IDs)	{2801953, 2012642, 2912939, 240001}

3.2.2 Reconnaissance and Vulnerability Reports

We next actively probe suspicious local hosts to collect more information about the running services, the patching level of critical components, and the existence or absence of vulnerabilities. This type of information can be used to evaluate if an investigated node is susceptible to a specific type of infection or if the observed IDS alerts are inconsistent with the host information and therefore likely false positives.

We use a combination of network scanning and vulnerability assessment techniques from which we extract a number of useful features.

In particular, we first use basic reconnaissance techniques such as IP sweeps, NIC whois querying, and TCP/UDP port-scanning in order to identify if a host is reachable and exposed to external attacks. In addition, we determine its role within the infrastructure, such as web, mail, or DNS server. Secondly, we perform targeted network scanning using NMap in order to retrieve detailed information regarding the TCP and UDP network services running on suspicious hosts, details about the OS type and version, and information regarding the types of ICMP messages a host responds to, which reveals its filtering policies and firewall effectiveness.

In Figure 3.1 we give an example of the output of NMap for a scanned suspicious host. We highlight the information we are interested in, namely the type and version of open services and the OS running on the scanned system. After having detected the accessible network services, we investigate the corresponding host for known vulnerabilities. In this respect we use publicly available vulnerability scanners, namely Nessus [5] and OpenVas [6], in order to build a comprehensive profile regarding the vulnerability status of a node.

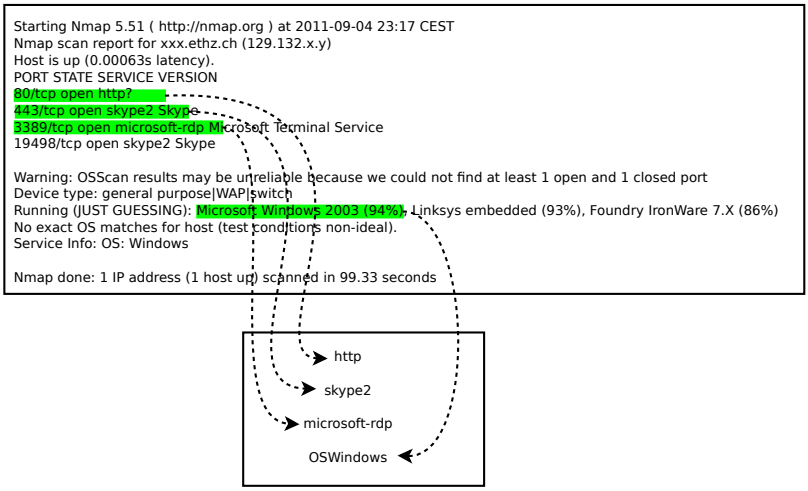


Figure 3.1: *NMap feature extraction*

In summary, the full set of extracted features from reconnaissance and vulnerability reports is the following:

1. *Host Reachability:* This is a boolean feature that determines whether the host is exposed or not. Nodes behind a firewall or a NAT will typically be unreachable.

2. *Host Role:* We exploit the hostnames assigned to machines within the ETH internal network, such as `proxy-XX.ethz.ch` or `guest-docking-nat-YY.ethz.ch`, to determine the function of a host. For regular user machines this feature takes the value *client*, whereas in the case of servers it takes one of the following values *dns-server, web-server, ftp-server, unknown-server*.

3. *OS and active services:* For each open service we detect, we introduce a new feature that takes the value 1 if it is open for the respective host or 0 otherwise. The same process is followed for information we collect for the underlying OS. Each new OS keyword is considered a new feature. We only take into account the suggested OS match with the highest likelihood provided by NMap OS fingerprinting output.

4. *Vulnerability data:* we collect the vulnerability logs we get from Nessus and OpenVas for each investigated host. We use this source of information in the manual security assessment performed in Section 4.3. However, we exclude the vulnerability data from the feature space used in the classification scheme discussed in Section 3.3.1, since it introduces significant amount of noise. For example, we have seen that a host running an unpatched version of Windows 7 will typically have more than 60 active vulnerabilities.

3.2.3 Blacklists

The third security source we exploit is blacklists. Blacklists are commonly-used to identify IP addresses and domains that have been reported to exhibit malicious activity. In our context, we use them to investigate suspicious remote hosts extracted from IDS data, i.e., remote hosts that have a persistent communication with an investigated local host and that generate several IDS alerts of the type *Attacks* or *Compromised hosts*.

We leverage five public blacklist providers [9–13]. The blacklists are partly labeled providing information about the reason a host was enlisted, including the type of malicious activity it was involved in, e.g.,

Table 3.2: *Blaklist data labels.*

	ads	attack	bot	chat	drugs	generic	malware	porn	rbn	religion	spam
Apews											✓
Dshield		✓				✓					
Emerging Threats		✓	✓			✓			✓		
Shadowserver			✓								
Urlblacklist	✓			✓	✓		✓	✓		✓	

bot activity, active attack, and spamming. For each available blacklist label we generate a new feature and we assign to it a value equal to the total hits we measured, taking into account all blacklist sources. If a blacklist source does not provide labels, we use a generic label to count the total hits originating from this source. In Table 3.2 we show the different blacklist providers we used and the available labels per source.

3.2.4 Search Engine

Network traces and security logs collected within the premises of our infrastructure provide a local view of the malicious activity manifested by external hosts contacting internal infected machines. However, endpoint profiling information is publicly available on different forms on the web [14]. This information originates from several diverse sources such as DNS-lists, website access logs, proxy logs, P2P tracker lists, forums, bulletins, banlists, IRC-lists, etc. In order to retrieve this information we query the Google search engine using as input string the IP address of an analyzed host and the respective domain name we get using a reverse-DNS lookup. This process is illustrated in Figure 3.2.

For each investigated internal host we query for the contacted IP addresses in the *Suspicious remote hosts* list. Then in an automated fashion we parse the output and extract tags associated with the corresponding IP address. In Table 3.3 we show a list of tags we look for and the corresponding extracted features. These features take a value equal to the total number of occurrences of the respective tag.

3.3 Forensics Analysis Experiment

In this section we briefly describe our forensics analysis experiment. More information on the validated infections and the diagnosis process

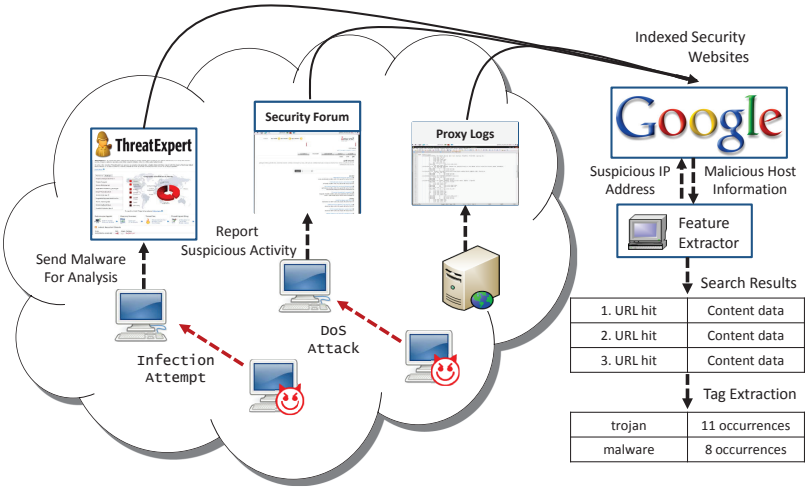


Figure 3.2: *Extracting features from Google*

Table 3.3: *Google profiling tags and extracted features.*

Tags	Profile	Feature
ftp, webmail, email, mysql, pop3, mms, netbios	Benign Host	host
dhcp, proxy	Benign Server	server
malware, spybot, spam, bot, trojan, worm	Malicious host	malicious
blacklist, banlist, blocklist, ban	Blacklisted hosts	blacklisted
adaware	Adaware	adaware
irc, undernet, innernet	IRC Servers	irc
torrent, emule, kazaa,edonkey, announce, tracker, xunlei, limewire, bitcomet, uusee, qqlive, pplive	P2P clients	p2p

along with four example malware cases can be found in [2].

We used the Snort alert correlator we developed in our previous work [2] to detect infected hosts within the monitored infrastructure. During our experiment, we ran our correlator on the latest Snort alerts and we passed on a daily basis newly detected infections to an analyst for manual inspection and validation. Our experiment lasted for approximately four weeks between 01.04.2011 and 28.04.2011 during which we thoroughly investigated 200 consecutive infections. We limited our study to nodes with static IP addresses, which correspond to

the majority of the active nodes within the monitored network. Besides, we built automated tools to extract the features discussed in Section 4.2 and to present them on a dashboard in order to facilitate the investigation process. The analyst would typically have to check the quality of collected signatures, examine the network footprint generated by the studied host, gather information about the expected behavior of the investigated malware, and most importantly cross-correlate this information. We validated the activity of a specific malware type for 170 out of the 200 infections. We use this set of validated infections to build our decision support tool in Section 3.3.1.

3.3.1 Decision Support Tool

In this section we introduce a decision tree model that captures the key evidence from the four security sources that were used to identify different types of malware. Our model is useful for expediting the complex and time-consuming manual correlation of multiple data sources for the diagnosis of infected hosts.

We use the C4.5 decision-tree induction algorithm, which is a state-of-the-art tree-based classifier [16]. Studies have shown that its performance is better compared to that of BTCs and TANs in terms of classification efficiency [17], whereas it is comparable to SVMs. Moreover, it is computationally efficient and an open-source implementation is publicly available. For our purposes, the most important aspect of C4.5 is the interpretability of its results. It is important that a security analyst can understand which feature contributed in every step of the process in making a decision, without requiring expert statistical knowledge such as in the case of SVNs. The classification is performed using a tree structure, where each internal node corresponds to a decision based on one or more features and the leafs correspond to the decision outcome.

We use the J48 implemetation of the C4.5 algorithm [18]. It takes as input a vector of training samples, that correspond to the manually classified hosts. Each sample is a vector that captures the values of different security features of a specific host. We use the 131 security features we described in Section 4.2. Secondly, C4.5 takes as input a vector that indicates the class of each host.

The algorithm decides how to split the data based on an optimal

cut that separates the given samples into two subsets. The criterion used to define this cut is the normalized information gain. At each step C4.5 selects the feature that maximizes the normalized information gain that results from making the cut. Then, it recursively iterates for each subtree till all samples in the remaining data belong to the same class, in which case it generates a leaf node for this class.

The algorithm will attempt to iterate over all data and incorporate all features in the decision tree. There are two problems related to this approach. Firstly, the generated tree will be extremely large sucrificing interpretability for resolution. This is not desirable in our context since our key goal is to generate a model that is intuitive and useful for a security specialist. Secondly, over-fitting data from a training dataset that contains noise will reduce the classification accuracy. In order to to obtain manageable trees and to avoid over-fitting we use a prunning approach called subtree raising. Subtree raising will replace a node with low information gain with the node's subtree that exhibits the highest information gain value.

In the implementation of the C4.5 that we use, the pruning level is tuned by setting a confidence threshold. We use a heuristical approach to get the optimal value for this threshold. We exhaustively explore all possible values while trying to maximize the classification accuracy. Note that since our dataset is rather small such exhaustive search is possible. In our experiments it took less than a minute in a Quad-core 3GHz PC to complete the search.

To ensure the robustness of the threshold we compute, we perform a stratified ten-fold cross-validation. In this process the dataset is randomly divided into ten folds, nine of which are used for training and one for testing. The folds are stratified so that each fold has on average the same mix of different classes. We generate a decision tree from the training set given a value for the confidence threshold, and then we compute the classification score, which is simply the percentage of correctly classified hosts. The final score is the average of ten consecutive trials. This process reduces the bias or over-fitting effects that result from noisy data. The confidence value that we used in our experiments is 0.81. Intuitively, a value close to 1 means that we are confident that there are not many misclassified items in our training set, whereas a value close to 0 means that the training dataset is very noisy.

In Figure 3.3 we show the derived decision tree. This model accu-

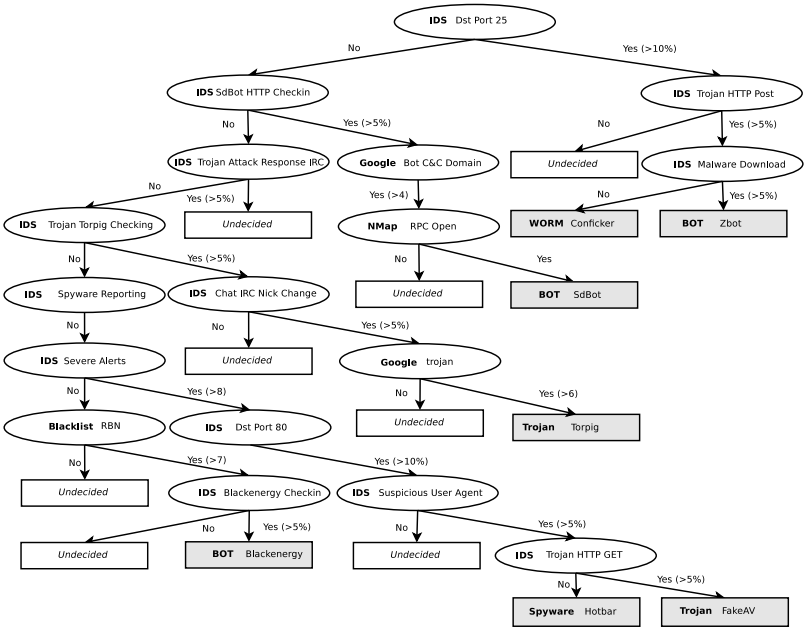


Figure 3.3: *Decision tree generated from the C4.5 algorithm using training data from 200 manually examined incidents*

rately depicts the main decisions of the manual process followed in the investigation of security incidents highlighting the most important signs of infection that can drive forensics analysis. Using the decision tree we can easily identify critical signs of malicious behavior originating from infected hosts. In the following paragraphs, we provide examples based on the derived model on how to combine evidence to detect specific types of malware.

Zbot-infected hosts are prominent spammers. We see that they generate a high percentage of high severity alerts that are related to destination port 25. These correspond to spamming attempts for which Snort raises an alert. Moreover, we see that they typically attempt to share stolen confidential data by performing an HTTP POST on a malicious domain. This typically triggers the IDS alert 2013976:"*ET TROJAN Zeus POST Request to CnC*". Periodically, the bot will at-

tempt to upgrade its binary triggering alerts with ID 2010448:"*ET MALWARE Potential Malware Download, trojan zbot*".

Besides, *SdBot*-infected hosts exhibit frequent communication with their C&C. The bot will attempt to identify a valid communication channel from a set of predefined rendez-vous domains with its C&C in order to update its instruction set and potentially post valuable client data it has intercepted. These attempts will trigger alerts with ID 2007914:"*ET WORM SDBot HTTP Checkin*". Moreover, the malware uses MS network shares to propagate and therefore we see that on most of the infected machines port 135 is open, which corresponds to the RPC service.

The *Torpig* trojan will periodically attempt to HTTP POST the data it has stolen from a victim triggering the snort alert "*ET TROJAN Sinowal/Torpig Checkin*" with ID 2010267. Also Torpig will also typically use IRC in order to receive updates resulting in frequent IRC nickname changes, detected by the Snort rule 542: "*CHAT IRC nick change*". The domains used to upload harvested user data, `vgnyarm.com`, `rajjunj.com` and `Ycqgunj.com`, were typically tagged as *trojan* hosting by our Google tagging method.

Finally, *FakeAV* is a Trojan that intentionally misinterprets the security state of the victim and generates pop-ups attempting to redirect the user to domains where the victim can purchase software to remediate the non-existing malware. This activity generates a very high number of alerts with ID 2002400:"*ET USER_AGENTS Suspicious*" that are related to port 80 (HTTP) activity.

Note that due to the pruning performed only the most prominent infections appear in the tree. However, if a higher resolution is required one can fine tune the confidence threshold accordingly. To summarize, from the analysis for the construction of the decision tree we learned the following main lessons:

- Combinations of features yield more accurate security assessment results. Moreover, different data sources are required if we are interested in detecting a wide range of malware types exhibiting complex behavioral patterns. On the other hand, simple rules of thumb such as 'if the IDS triggers at least N alerts of type X then there is an infection' cannot be used to effectively solve the problem.

- A small number of features is usually sufficient in order to make an assessment with a high degree of confidence. In most studied scenarios this feature set is representative of different types of activity exhibited by infected hosts. C4.5-generated decision trees retain a high level of interpretability assisting the analyst in making an assessment using this manageable and intuitive set of features.
- C4.5 can be used in an adaptive fashion providing a feedback loop to the analyst, who can update the set of classified infections in order to enrich the derived trees and also to improve the classification results. Model induction is very efficient even for datasets involving a large number of features.

3.3.2 Automated Diagnosis

In this section we analyze the effectiveness of our model in automatically diagnosing host infections using the stratified ten-fold cross-validation technique described in Section 3.3.1. We also compare how C4.5 performs against other popular classification methods. Specifically, we evaluate the classification accuracy of BTCs and of TANs, which are popular choices when it comes to tree-based decision making. We use the WEKA implementation for these two classifiers with the default parameters. We also evaluate the performance of SVMs, which are commonly considered a state-of-the-art classification method. To configure the SVM parameters we use the sequential minimal optimization method [19].

Table 3.4: *Performance of different classification algorithms.*

Malware Type	C4.5		Bayesian Tree		TAN		SVM	
(#incidents)	TP (%)	FN (%)	TP (%)	FN (%)	TP (%)	FN (%)	TP (%)	FN (%)
Trojans (85)	83	10	80	12	82	10	89	6
Spyware (59)	85	4	85	5	85	4	88	4
Backdoors (18)	55	8	53	7	56	7	63	5
Worms (8)	75	1	75	1	75	1	77	1
Undecided (30)	60	10	48	14	51	13	63	9

In Table 3.4 we summarize our findings. C4.5 exhibits on average a true positive rate of 72% whereas the false negative rate does not exceed on average 7%. Bayesian networks and TANs are consistently worse exhibiting a true positive rate of 68% and 70% and a false negative rate

of 8% and 7%, respectively. On the other hand SVMs achieve slightly better classification results with a true positive rate of 76% and a false negative rate that does not exceed on average 5%.

3.4 Related Work

Previous works have extensively studied the aggregation and correlation of IDS alerts with the goal of generating high-level inferences from a large number of low-level alerts. A group of studies exploit statistical correlation to perform causality inference and root cause analysis of detected incidents [20–22]. On the other hand, another group of studies hardcode expert knowledge by introducing scenarios [23, 24] or sets of rules [2, 25–27] that capture observed malicious behavior. These studies mine solely IDS alerts without taking into account complementary sources of security logs that are often available. They produce and prioritize inferences that at the end are passed on to a security analyst for manual inspection. Our work is complementary and focuses on the manual verification of aggregated IDS alerts by correlating data from multiple instead of a single source.

A number of commercial solutions such as IBM Tivoli Security Compliance Manager [28], Alienvault [29], and GFI Languard [30] unify scattered security sensors within an enterprise and provide a single framework that can be used by security analysts to configure the available security components and to visualize logs. However, log correlation in these systems is based on simple rules. Few generic rules come typically pre-configured, while most rules need to be determined by the administrator. In our work, we build a number of classification rules in the form of a C4.5 decision tree that can constitute the input to such systems.

Another group of studies analyze security incidents in the wild. Most related to our work Sharma *et al.* [4] recently analyze 150 security incidents that occurred in a supercomputing center over five years using data from five security sensors. Their work focuses on the characterization of security incidents, including their severity, detection latency, and the popularity of different alerts. We also analyze a larger number of less severe incidents that occurred within a shorter period in a larger operational infrastructure. However, we focus on the evaluation of the complementary utility of different data sources and based on our analy-

sis build a decision support tool to expedite manual forensics analysis. Besides, Maier *et al.* [3] tailor custom heuristics to detect scanners, spammers and bot-infected hosts in packet traces from a large number of residential DSL customers. They find that systems with more risky behavior are more likely to become infected. We also analyze compromised hosts in the wild, but focus on correlating multiple security sources. In our previous work we developed a heuristic to correlate Snort alerts [2] and we characterized a number of aspects of approximately 9,000 infections we detected over a period of nine months in a large academic infrastructure. We also validated our heuristic as part of the results of our manual analysis experiment. In this paper we focus on the lessons learned from the complex manual analysis process and in particular we evaluate the complementary utility of different security data sources and we induce a model to facilitate the assessement of future incidents. In [31] Gu *et al.* perform an extensive passive and active measurement analysis of three predominant botnets and make some interesting observations regarding the similarities and differences exhibited in the infection methods. They provide some insights for building better defenses without incorporating them, though, into a statistical model or framework.

Finally, a number of studies use historical records of healthy and unhealthy system states and attempt to diagnose in real time the root cause of failures and misconfigurations using decision trees [32–35]. Our approach is similar in that we also correlate information from diverse sources and represent complex system states using decision trees. However, our focus is on diagnosing security incidents rather than system and network failures.

3.5 Conclusions

Manual security assessment can be likely better described as art rather than science. It relies on a security expert combining his reasoning with background knowledge about malware, domain specific knowledge about the target environment, and available evidence or hints from a number of diverse security sensors, like IDS systems, vulnerability scanners, and other. We believe that in the future processes for handling and diagnosing security incidents should be largely automated diminishing (but not completely removing) the involvement of humans.

A major challenge towards this goal is that manual security diagnosis is a very complex process and requires further understanding. Towards improving the present ad-hoc manual investigation methods, in this work we conduct a complex experiment: we systematically monitor the decisions of a security analyst during the diagnosis of 200 incidents over a period of four weeks. From this process we learn about the complementary utility of four different security data sources and we build a decision support tool to expedite manual investigation. From our experiment, we report a number of interesting findings. Firstly, we observe that a search engine, a less well-established security data source, was much more useful in the diagnoses of security incidents than other more traditional security sources, like blacklists, reconnaissance and vulnerability reports. Secondly, we show that a large part of the decisions of the analyst of our experiments can be encoded into a decision tree, which can be derived in an automated fashion from training data. The decision tree can help as a decision support tool for future incident handling and provides comparable performance in fully automated classification with a more advanced classifier, an SVM, which however is much less interpretable. Furthermore, from our experiments we learn that a single source is typically not sufficient for manual security diagnosis and that multiple sources should be combined. In more than 10% of the cases, no single alert or evidence, but the overall behavior of the host helped to verify an infection. These results highlight the importance of a holistic approach to malware detection. Multiple sensors are needed in more than 70.5% of all cases when the cases of easier to detect spyware are excluded.

Our findings are based on incidents detected in a large academic network using the heuristic we developed in our previous work [2]. Clearly, they are not indicative of all possible malware. They provide however useful insights for a large diversity of malware we present in detail in Table 4.1. In addition, as the malware landscape is very dynamic it is possible that the identified evidence will not hold for future malware. An open difficult challenge for future research is how to make decision support tools robust to changes in malware.

Bibliography

[1] R. Sadoddin and A. Ghorbani, "Alert correlation survey: framework and techniques," in *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, ser. PST '06. New York, NY, USA: ACM, 2006, pp. 37:1–37:10. [Online]. Available: http://doi.acm.org/10.1145/1501434.1501479

[2] 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*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 29–44. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068820

[3] G. Maier, A. Feldmann, V. Paxson, R. Sommer, and M. Vallentin, "An assessment of overt malicious activity manifest in residential networks," in *Proceedings of the 8th international conference on Detection of intrusions and malware, and vulnerability assessment*, ser. DIMVA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 144–163. [Online]. Available: http://dl.acm.org/citation.cfm?id=2026647.2026660

[4] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. K. Iyer, "Analysis of security data from a large computing organization," in *DSN*, 2011, pp. 506–517.

[5] "The Nessus vulnerability scanner," http://www.tenable.com/products/nessus.

[6] "The Open Vulnerability Assessment System," http://www.openvas.org.

[7] "A free lightweight network intrusion detection system for UNIX and Windows," http://www.snort.org.

[8] "Emerging Threats web page," http://www.emergingthreats.net.

[9] "Anonymous postmasters early warning system," http://www.apews.org.

[10] "The Urlblacklist web page," http://www.urlblacklist.org.

[11] "Shadowserver Foundation web page," http://www.shadowserver.org.

[12] "Cooperative Network Security Community - Internet Security," http://www.dshield.org.

[13] "Advanced automated threat analysis system," http://www.threatexpert.com.

[14] I. Trestian, S. Ranjan, A. Kuzmanovi, and A. Nucci, "Unconstrained endpoint profiling (googling the internet)," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 279–290. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402991

[15] S. Sinha, M. Bailey, and F. Jahanian, "Improving spam blacklisting through dynamic thresholding and speculative aggregation."

[16] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, March 1986. [Online]. Available: http://dl.acm.org/citation.cfm?id=637962.637969

[17] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Mach. Learn.*, vol. 29, pp. 131–163, November 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=274158.274161

[18] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, 2005.

[19] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[20] X. Qin and W. Lee, "Statistical causality analysis of infosec alert data," in *RAID 2003*, 2003, pp. 73–93.

[21] H. Ren, N. Stakhanova, and A. A. Ghorbani, "An online adaptive approach to alert correlation," in *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment*, ser. DIMVA'10. Berlin, Heidelberg:

Springer-Verlag, 2010, pp. 153–172. [Online]. Available: http://portal.acm.org/citation.cfm?id=1884848.1884861

[22] X. Qin, "A probabilistic-based framework for infosec alert correlation," Ph.D. dissertation, Atlanta, GA, USA, 2005, aAI3183248.

[23] B. Morin and H. Debar, "Correlation of intrusion symptoms: an application of chronicles," in *RAID03*, 2003, pp. 94–112.

[24] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '00. London, UK: Springer-Verlag, 2001, pp. 85–103. [Online]. Available: http://portal.acm.org/citation.cfm?id=645839.670735

[25] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *In Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 245–254.

[26] F. Cuppens and A. Miège, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 202–. [Online]. Available: http://portal.acm.org/citation.cfm?id=829514.830542

[27] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multistep cyber attacks for scenario recognition," 2003.

[28] "IBM Tivoli Security Compliance Manager," http://www-01.ibm.com/software/tivoli/products/security-compliance-mgr.

[29] "AlienVault," http://www.alienvault.com/.

[30] "GFI Languard," http://www.gfi.com/network-security-vulnerability-scanner.

[31] S. Shin, R. Lin, and G. Gu, "Cross-analysis of botnet victims: New insights and implications," in *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID'11))*, September 2011.

[32] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *In Proceedings of the International Conference on Autonomic Computing (ICAC)*, 2004, pp. 36–43.

[33] I. Cohen, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *In OSDI*, 2004, pp. 231–244.

[34] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, "Netprints: diagnosing home network misconfigurations using shared knowledge," in *Proceedings of the 6th USENIX symposium*, ser. NSDI'09, Berkeley, CA, USA, 2009, pp. 349–364. [Online]. Available: http://dl.acm.org/citation.cfm?id=1558977.1559001

[35] S. Kandula, D. Katabi, and J.-P. Vasseur, "Shrink: A Tool for Failure Diagnosis in IP Networks," in *ACM SIGCOMM Workshop on mining network data (MineNet-05)*, Philadelphia, PA, August 2005.

Chapter 4

Understanding Network Forensics Analysis in an Operational Environment

Elias Raftopoulos[1], Xenofontas Dimitropoulos[1]

[1]Computer Engineering and Networks Laboratory,
ETH Zurich, Switzerland

doi: 10.1109/SPW.2013.12

Abstract —

The manual forensics investigation of security incidents is an opaque process that involves the collection and correlation of diverse evidence. In this work we conduct a complex experiment to expand our understanding of forensics analysis processes. During a period of four weeks we systematically investigated 200 detected security incidents about compromised hosts within a large operational network. We used data from four commonly-used security sources, namely Snort alerts, reconnaissance and vulnerability scanners, blacklists, and a search engine, to manually investigate these incidents. Based on our experiment, we first evaluate the (complementary) utility of the four security data sources and surprisingly find that the search engine provided useful evidence for diagnosing many more incidents than more traditional security sources, i.e., blacklists, reconnaissance and vulnerability reports. Based on our validation, we then identify and make available a list of 165 good Snort signatures, i.e., signatures that were effective in identifying validated malware without producing false positives. In addition, we compare the characteristics of good and regular signatures and highlight a number of differences. For example, we observe that good signatures check on average 2.1 times more bytes and 2 times more fields than regular signatures. Our analysis of Snort signatures is essential not only for configuring Snort, but also for establishing best practices and for teaching how to write new IDS signatures.

4.1 Introduction

Security analysts are overwhelmed by massive data produced by different security sources. Investigating security incidents is an opaque "art" that involves 1) carefully extracting and combining evidence from the available security sources; 2) thoroughly understanding how suspected malware operate; and 3) exploiting information about the infrastructure and configuration of the affected network. In this arena, security analysts are restricted to using slow manual and often ad-hoc forensics analysis processes.

Towards understanding and improving forensics analysis processes, in this work we conduct a complex experiment in which we systematically monitor the manual forensics analysis of live suspected infections

in a large production university network that serves tens of thousands of hosts. In particular, over a period of four weeks, we manually investigated in coordination with the IT department of our university 200 security incidents about compromised hosts detected by an IDS alert correlator. The security investigation combined data from four security sources: 1) Snort alerts, 2) reports from four scanning and vulnerability assessment tools, 3) five independent blacklists, and 4) a search engine (Google).

Based on our experiment, we describe a number of lessons we learned from the validation of security incidents. In particular, we make three contributions. First, we describe how to leverage four different security data sources to remotely diagnose live infections in a large production network. Second, to delineate the manual investigation process, we evaluate the (complementary) utility of the four data sources. Surprisingly, we find that a search engine was one of the most useful sources in deciding if a suspicious host was infected, providing useful evidence that led to a positive diagnosis in 54.5% of the cases. Reconnaissance and vulnerability reports were useful in fewer cases, but helped diagnose more sophisticated malware, whereas blacklists were useful only for 10.5% of the incidents. In addition, we report which combinations of sources helped diagnose different types of malware.

Third, we make available a list of 165 Snort signatures that were effective in detecting validated malware without producing false positives. We analyze the differences between good and regular Snort signatures and find, for example, that good signatures check on average 23.2 Bytes in 2.4 different fields, while regular signatures check on average only 11.2 Bytes in 1.2 fields. In addition, we observe that good signatures tend to use offsets, regular expressions, fixed packet sizes, and specific destination ports much more often than regular signatures. Based on these observations, we highlight good signature characteristics useful for configuring Snort and for establishing signature writing best practices.

The remaining of our paper is structured as follows. In the next section we describe the data we used. Then, in Section 4.3 we describe the investigation of four representative malware cases. We present our findings regarding the utility of the data sources and the effective signatures in Section 4.4 and 4.5, respectively. Finally, in Sections 4.7 and 5.6 we outline related work and conclude our paper.

4.2 Data Collection

Our monitored infrastructure is the main campus network of the Swiss Federal Institute of Technology at Zurich (ETH Zurich). The campus is a "zoo" of diverse systems, like critical servers, desktops, laptops, and other lab and office devices. The traffic mix we observe is rich since there are almost no restrictions on the software users can install on their devices and on the services they can use. Our monitoring period lasted for approximately four weeks between the 1st and the 28th of April 2011, during which we observed in total 28,665 unique active internal hosts.

We select four data sources that provide complementary views into the security status of a host from different vantage points covering aspects, like its traffic, the running services, and their vulnerabilities. These sources are commonly used for security assessment. First, we collect IDS alerts generated by an IDS, which is located between the primary border router and the network firewall of the monitored network. The IDS monitors all upstream and downstream traffic and generates an alert when a malicious signature is triggered. The IDS gives a view of malicious activities from the gateway of the network.

To collect information related to services and applications running on internal hosts, we perform reconnaissance and active scanning using NIC *whois* querying and Nmap. After mapping the accessible network services of a host, we investigate for known vulnerabilities using the Nessus [14] and OpenVas [15] vulnerability scanners. These four tools, to which we collectively refer as reconnaissance and vulnerability scanners, give a more detailed view of the internal hosts. The last two data sources help extract information about remote hosts by leveraging publicly available data. In particular, we use blacklists from five blacklist providers covering different types of malicious hosts, like active attackers and spammers, and we query the Google search engine for suspected remote hosts and domains. The search engine indexes a variety of sources, including public forums, bulletins, and banlists, which we exploit in an automated way to find the roles, i.e., server type, or actions of remote hosts. The detailed description of each feature can be found in the accompanying technical report [28].

4.2.1 IDS Alerts

The IDS deployed in our infrastructure is Snort [32], which is commonly considered the open source IDS solution with the largest number of registered users and the most active community [8,13]. Snort is configured to use signatures from the two most widely-used publicly available rulesets, namely the Vulnerability Research Team (VRT) ruleset and the Emerging Threats (ET) ruleset [11]. As of April 2011 the two rulesets have in total 37,388 distinct signatures.

We use IDS alerts in two ways. First, we apply an effective IDS alert correlator we have introduced in our previous work [27] to derive a small set of incidents, which we comprehensively analyze to evaluate their validity. Second, during the forensics analysis we manually examine the alerts of an investigated host and extract a number of features regarding the type and severity of the alerts, the size and duration of an incident, and the involved services and hosts. These features are presented to the analyst along with additional information extracted from the other three data sources and are used for the manual diagnosis of incidents. In addition, in many cases the analyst manually examined the specific signatures of Snort rules to assess their trustworthiness.

We had to address two challenges when analyzing Snort alerts. First, the amount of alerts is overwhelming making the analysis very time-consuming. In total we collected 37 million alerts over a period of four weeks, the majority of which are policy alerts that are not directly related to security incidents of interest. Second, the alerts are dominated by false positives, which makes it very hard to have any certainty about the existence of a malware based solely on a single alert. Our Snort alert correlator distills a small number of events that exhibit a recurring multi-stage malicious pattern involving specific types of alerts. It first aggregates similar alerts that exhibit temporal proximity, it then classifies each aggregate alert as *Direct attack*, *Compromised host*, or *Policy violation*, and finally it infers active infections by identifying internal hosts that exhibit a recurring multi-stage network footprint involving alerts of the first two classes. Applying our correlator on raw Snort alerts results into a small number of highly suspicious events. In particular, during the four weeks of our experiment, the correlator distilled 200 aggregate events from 37 million raw alerts.

4.2.2 Reconnaissance and Vulnerability Reports

In order to measure the exposure of a studied host to external attacks, we additionally collect host-based information about the running services, the patch level of deployed software, and the presence of vulnerabilities. This information is complementary to the network behavior patterns we get from Snort alerts, since it helps identify the threats that a host is susceptible to.

We use a combination of network scanning and vulnerability assessment techniques. In particular, we first use basic reconnaissance techniques, like IP sweeps, NIC whois querying, and TCP/UDP portscanning, in order to identify if a host is reachable and exposed to external attacks. In addition, these techniques help determine the role of host, e.g., web, mail, or DNS server, within the infrastructure.

Secondly, we perform targeted network scanning using Nmap in order to retrieve information regarding the TCP and UDP network services running on suspected hosts, details about the type and version of their operating system, and information regarding the types of ICMP messages a host responds to, which reveal its filtering policies and firewall effectiveness. After having detected the accessible network services, we investigate a host for known vulnerabilities. For this purpose, we use two well-known vulnerability scanners, namely Nessus [14] and OpenVas [15], in order to build a comprehensive profile of the vulnerability status of a host.

4.2.3 Blacklists

In order to examine if an examined host within our network frequently initiates connections to known malicious domains, we use a set of independent blacklists. We leverage five public blacklists [6, 7, 10, 12, 16], which are partly labeled indicating the type of malicious activity exhibited by a blacklisted host, e.g., bot activity, active attack, and spamming.

We then investigate whether the remote hosts contacted by an internal host are listed in a blacklist. If there is a hit then we tag the investigated host with the label of the corresponding blacklist. This method is useful to identify if suspicious internal hosts frequently establish communication with known malicious domains. Such communication typically occurs when a user visits malicious websites or when

a piece of malware installed on the infected host attempts to perform unsolicited actions, e.g., redirecting to third-party domains, updating its binary, sharing stolen confidential data, or getting instructions from a remote controller.

4.2.4 Search Engine

Apart from using intrusion detection alerts, active scans, and blacklists in order to characterize remote hosts exhibiting frequent communication with local investigated hosts, we also exploit security-related information residing on the web. When a host exhibits a malicious activity it will leave traces in different forms in publicly available sources such as DNS lists, website access logs, proxy logs, P2P tracker lists, forums, bulletins, banlists, and IRC lists. To retrieve this information we query the Google search engine using as input the IP address and the respective domain name of the local and remote hosts. In an automated manner we parse the output looking for a set of pre-selected tags such as *malware*, *spam*, *bot*, *trojan*, *worm*, *pop3*, *netbios*, *banlist*, *adaware*, *irc*, *undernet*, *innernet*, *torrent*. A similar approach has also been used in [33]. These tags reveal specific actions a host has taken, e.g., receiving instructions from a known botnet C&C, or roles it had for extended periods of time, e.g., operating as a mail server.

4.3 Evidence Correlation Studies

In this section, we first describe the manual malware validation experiment we conducted and then we outline example evidence correlation cases for four different types of malware.

Our experiment used Snort alerts that were pushed in an hourly basis to an alert archival infrastructure we are operating since 2009. We configured our alert correlator with the default configuration parameters [27]. We restricted our analysis to static IP addresses, which are widely-used in the monitored network. Detected incidents about infected hosts were annotated with information about the IP address of the host, the involved alerts, and a timestamp. A security analyst on a daily basis (during week days) manually investigated the latest detected incidents. Our experiment was conducted in cooperation with the IT department of ETH Zurich. To expedite the manual analysis process

we developed a number of tools that took as input the IP address of an investigated host, collected the data described in the previous section, extracted a large number of associated features, and displayed all the relevant information on a dashboard. The manual analysis process was complex and very time-consuming as it often required processing and analyzing a huge amount of IDS alerts, checking the quality of their signatures, collecting information about malware, and most importantly cross-correlating all this information. We investigated in total 200 consecutive incidents over approximately four weeks from the 1st until the 28th of April 2011. In the following paragraphs we present representative cases of the manual investigation performed for four different types of malware.

Case 1: Torpig infection. Torpig is one of the most sophisticated trojans in the wild. The typical method of infection is Drive-by-Downloads. The victim visits a vulnerable legitimate web site that requests Javascript code from a malicious webserver. The code is then executed and the trojan attaches itself to popular applications on the victim's system. Torpig is a typical data harvesting trojan using both passive monitoring and active phishing techniques to get valuable confidential data from an infected machine. Our active scanning for Torpig-infected hosts shows that Windows is the operating system of the host, whereas the services HTTP, Skype, and FTP are typically open. HTTP is the protocol used by the malware to contact the C&C servers and also to redirect the user to malicious domains, whereas Skype and CuteFTP are standard applications Torpig injects itself into. Torpig periodically contacts the C&C servers to get instructions and to update its binary triggering IDS alerts with IDs 2002762 and 2003066. Also, frequently it will attempt to report stolen user data using a POST command on known malicious domains triggering alerts with IDs in the range [2404000:2404300]. The domains we saw that were used for the POST operations were `vgnyarm.com`, `rajjunj.com` and `Ycqgunj.com`. The dominant tags produced by our search engine profiling method were *trojan* and *bot*.

Case 2: SdBot infection. W32/SdBot is a backdoor used by cyper-criminals to gain unauthorized access to victim machines. It can be used to launch active attacks or to harvest sensitive user data from an infected host. We observe that W32/SdBot-infected hosts are typically running MS network services such as WebDav, RPC, and LSASS.

The malware exploits the MS-LSASS buffer overflow, the MS-RPC malformed message buffer overflow, and the MS-WebDav vulnerability to compromise a victim. W32/SdBot uses a typical IRC communication method to contact its C&C servers to update its configuration triggering Snort alerts with IDs in the range [2500000:2500500]. Also, the malware will often attempt to propagate using extensive port scanning to detect vulnerable hosts, mostly on port 445, triggering alerts with IDs in the range [2011088:2011089]. Finally, the W32/SdBot backdoor will attempt to fetch additional badware from remote web sites to install them on the local host. The infected hosts we analyzed attempted to connect via FTP to the domains `joher.com.tw` and `service.incall.ru`, which are typically used to download W32/Koobface and Trojan.FakeAV, respectively. The tags we extracted from our search engine query results for these domains indicated that they are involved in *malware* and *bot* related activities.

Case 3: Hotbar infection. Win32/Hotbar is the most prominent infection detected in our infrastructure. Typically it comes as a web-browser add-on, like the Asksearch or Mysearch toolbar, providing seemingly legitimate functionality. However, in the background it tries to harvest user activity patterns including browsing habits and application usage. This spyware does not attempt to hide and it will often identify itself using the User-Agent field of an HTTP request using the string "AskTB". Snort uses alerts with IDs in the range [2003305:2003500] to detect this behavior. Moreover, the spyware will regularly attempt to redirect a user to the domains `bestdealshost.biz` and `zangocash.biz` by generating clickable pop-ups. These domains are typically tagged by our search engine analysis for hosting malware.

Case 4: Koobface infection. W32.Koobface is a worm using social networking sites such as Facebook and MySpace to propagate. It typically links to popular videos, in order to convince a user to install an executable that appears to be a necessary codec update but is in fact the Koobface executable. After successful infection it will attempt to propagate by sending messages to the victim's contact list. This is done by issuing HTTP POST requests, which are detected by the Snort signatures with IDs 2009156, 2010335, and 2010700. Typical landing pages used for the redirection in order to fetch the Koobface executable are `prospect-m.ru` and `pari270809.com`, which are

tagged as suspicious by our Google profiling method generating the tags *bot* and *worm*.

4.4 Complementary Utility and Ranking of Security Sources

Table 4.1: *Prevalence of different malware types and variants in the 200 investigated incidents. The last four columns mark the data sources that provided useful evidence for diagnosis.*

Malware Type (#incidents)	Variant (#incidents)	IDS Logs	Search Engine	Blacklist Data	Active Scans
Trojans(85)	FakeAV(27)	✓	✓		
	Simbar(26)	✓	✓		
	Monkif(18)	✓	✓		
	Torpig(10)	✓	✓	✓	✓
	Nervos(4)	✓	✓		
Spyware(59)	AskSearch(50)	✓			
	MySearch(9)	✓			
Backdoors(18)	SdBot(5)	✓	✓	✓	✓
	ZBot(5)	✓	✓		✓
	Blackenergy(4)	✓	✓	✓	✓
	Parabola(2)	✓	✓		✓
	Ramsky(2)	✓			✓
Worms(8)	Koobface(6)	✓	✓		
	Conficker(2)	✓	✓		✓

In this section we present our results on the complementary utility of the four security data sources for validating different types of malware. In Table 4.1 we list the malware variants that were identified. We classify malware into four categories, namely backdoors, spyware, worms, and trojans, and for each malware we indicate in the second column the relevant category. Note that the behavior of modern malware often combines multiple characteristics, which seriously perplexes the process of putting real-world malware into a taxonomy. For this reason, in few cases malware could also be assigned to a different category.

For 30 out of the 200 analyzed incidents, our investigation did not lead to a definite assessment even when combining evidence from all

sources. The remaining 170 validated incidents include 85 trojans, 59 spyware, 18 backdoors, and 8 worms. The most popular malware family was AskSearch followed by FakeAV and Simbar. In the last four columns of Table 4.1 we identify the combination of sources that were useful for identifying the corresponding type of malware. In 41.5% of the cases two sources and in 14% of the cases at least 3 sources had to be combined to have high certainty about the diagnosis of an investigated host. The correlation of multiple sources was particularly useful for the diagnosis of more sophisticated malware, like Torpig, SdBot, and Polybot. On the other hand, a single source was useful to identify AskSearch and MySearch.

In our previous short paper [26], which complements this work, we additionally mapped the forensics analysis process into a decision tree, and used the results of the manual investigation to train the C4.5 tree classifier. In this way we encoded the knowledge we derived from the manual assessment of security incidents presented here, to a decision support tool, which can be a significant aid in the diagnosis of future incidents.

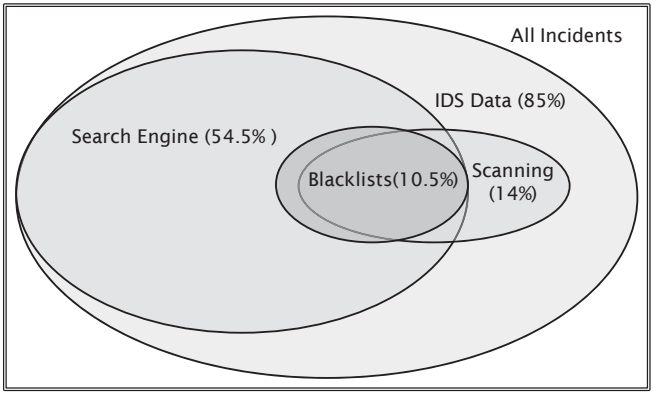


Figure 4.1: *Complementary utility of security data sources for the diagnosis of 200 incidents*

In Figure 4.3 we depict how often a source or a combination of sources were useful for diagnosing an infection, which illustrates the complementary utility of the four data sources. Based on our experi-

ments, we rank the four sources as follows:

1. **IDS alerts:** IDS alerts are the most fruitful source of information since they were the basis for our assessment. In 170 out of the 200 investigated incidents, the further manual examination of IDS alerts observed near the time of a detected incident provided useful evidence for validation.

2. **Search Engine:** The second most useful data source in our arsenal was the output of our automated search engine queries. In 54.5% of the cases, query results were a valuable resource providing critical information for the analysis. This information was in most cases complementary to the content of IDS alerts, providing knowledge about active C&C hosts, botnet communication channels, malware landing pages, redirections to malicious domains, malware download pages, and phishing forms.

3. **Reconnaissance and Vulnerability Reports:** On the other hand, the information we collected by scanning suspicious hosts helped us to reach a definite assessment in approximately 14% of the cases. However, these were the most demanding inferences involving sophisticated malware that exhibit a very complex behavior like *Torpig* or *SdBot*.

4. **Blacklists:** Finally, blacklists were the least valuable source of security information. Their output partially overlapped with information we already extracted from IDS alerts or from Google. Moreover, they contain a very large number of false positives, we speculate due to slow responsiveness in enlisting new malicious hosts [31].

Below we summarize the *main lessons* we learn from our evaluation of the utility of different security sources:

Insight 1. No single sensor provides conclusive evidence about an investigated incident. Relying on a single defensive mechanism might be sufficient to detect automated threats with deterministic and predictable activity. However, *most modern threats exhibit complex behaviors that require a multi-vantage point security architecture for effective detection.*

Insight 2. IDS sensors have been heavily criticized for producing a large number of false positives, rendering them unreliable if used in isolation for identifying active infections. In this work we highlight that by *exploiting statistical and temporal correlations of the alert stream, our IDS data became very useful, since they helped identify a few actionable cases that exhibited a high likelihood of infection.*

Insight 3. Alternative security sources, such as the output we get from the profiling method using the Google search engine, proved to be more helpful for making an assessment than traditional security sources, such as vulnerability reports and blacklists. This highlights the *importance of endpoint profiling for domains visited by infected hosts. The reputation and typical activity of these remote hosts provides invaluable pieces of evidence that can drive the investigation process.* Moreover, *blacklists should only be used as a low quality indicator of domain reputation. Search engine content seems to be more reactive and up-to-date regarding changes in a host's activity compared to the information contained in blacklists.*

4.5 What a Good IDS Signature Looks Like?

Next, we provide a list of Snort signatures that were found by our alert correlator useful for detecting confirmed malware without generating false positives. These signatures are useful for configuring the widely-used Snort IDS and for teaching good signature writing practices. They are based on the 170 validated incidents and are effective in detecting the malware types listed in Table 4.1.

Our alert correlator finds tuples of aggregated alerts that occur frequently together. For each tuple involved in a validated incident, we extracted the corresponding Snort signatures. We found in total 165 Snort Signature IDs (SID) that can be summarized into 49 aggregate signatures as several affiliate SIDs detect small variations of the same pattern. In Table 4.7 we provide the 165 SIDs and a short description of the 49 aggregate signatures classified in five classes based on the behavior they detect.

A signature is labeled as good if the following conditions are met: it is contained in a tuple generated by our correlator, it is used as

evidence in the subsequent manual assessment process, and the respective incident is validated by a security expert as an infection. Note that signatures which generate a large number of false positives and might coincidentally get triggered during a security incident, will be in most cases filtered out by our correlator [27]. Otherwise they will be discarded as irrelevant during the manual assessment process.

The malware we studied typically undergo a sequence of stages after the initial infection. In many cases, they attempt to contact their C&C to receive new instructions and/or to report stolen data. Periodically they may attempt to update their binary or to fetch additional malware, which are installed on infected hosts. Some types of malware (e.g. clickbots) often attempt to redirect a user to known malicious pages by changing search engine results, by generating pop-ups, or by redirecting HTTP requests. Finally, most trojans and worms in our study also attempt to propagate by scanning for vulnerabilities. Based on these, we classify signatures in one of the following five categories (see Table 4.7): C&C communication, reporting, egg download, redirection, and propagation.

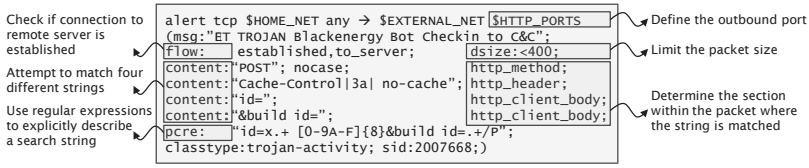


Figure 4.2: *Example of good Snort signature used to detect a beacon frame sent by a Blackenergy bot to its controller*

In Figure 4.2 we show an example of a good signature that looks for a beacon sent by a Blackenergy bot using an HTTP POST command to its controller. The signature checks for specific byte sequences in four different fields of the HTTP packet. In addition, it attempts to match the identification string sent by the infected host providing information about the build of the malicious binary. Additional conditions regarding the maximum packet size, the target ports, the state of the connection, and the exact format of the identification string are introduced to increase its specificity.

We next compare key characteristics of our good signatures to the average of the signatures that are triggered in our infrastructure, which

Table 4.2: *Comparison of good and regular Snort signatures. Statistics are computed over 165 good and 1371 regular signatures.*

	Regular Signatures	Good Signatures	Increase
Bytes Checked	11.2	23.2	2.1 ×
Fields Checked	1.2	2.4	2 ×
Byte Offset is Set	8%	28%	3.5 ×
Regular Expression is Set	15%	51%	3.4 ×
Regular Expression Size	4.2	11.3	2.7 ×
Destination Port is Set	17%	22%	1.3 ×
Packet Size is Set	6.9%	14.5%	2.1 ×
Flow Options are Set	30.25%	38%	1.8 ×
IP Options are Set	62.1%	68.4%	1.1 ×

uses the default VRT and ET rulesets. We created two groups of signatures. A group with the extracted 165 good signatures and its complement group of 1,371 regular signatures that are triggered during our experiment. In Table 4.2 we compare different features of the two groups. We observe that good signatures are much more complex requiring the satisfaction of multiple conditions. A good signature attempts to match on average 23.2 Bytes in 2.4 different fields, while a regular signature checks only 11.2 Bytes in 1.2 fields. In addition, 28% of the good signatures set the offset of a sought byte sequence; 50% provide a regular expression to further specify a search pattern; 22% fix the destination port(s); and 14.5% give a specific packet size. In sharp contrast, the corresponding numbers for regular signatures are only 8%, 15%, 17%, and 7%, respectively.

The regular expression size in good signatures is 11.3, compared to 4.2 in the case of regular signatures. To compute the regular expression size we use the alphabetic width which is, to the best of our knowledge, the most commonly used metric [5]. The alphabetic width counts the total number of alphabetic symbols in the regular expression, taking into account multiplicity. However, we have adapted this metric to reduce the effect of the union operator, which attempts to match any of a list of input variables to a target string, by considering only the alphabetically widest variable of the union. For example, we iteratively reduce the regular expression $R(V_1|V_2)$ to $R(V_1)$ if $alphabetic_{width}(V_1) \geq alphabetic_{width}(V_2)$.

Flow options are used to provide flow level control. For example,

signatures can state whether the processed packet belongs to an *established* connection, and further explicitely define the direction (e.g. *to_server*, *to_client*). IP options are typically used to detect IP-level attacks, reconnaissance and network mapping attempts, and protocol based DoS attacks. For example, IP options check for the usage of the *Fragment* bit, the *Type of Service* option, and the *Time to Live* value.

In the case of *HTTP*, we observe that good signatures tend to define the section of an HTTP packet, e.g., with the Snort key-words *header*, *method*, *uri*, and *revision*, where pattern matching is performed. Overall, we observe that the computed statistics exhibit a consistent increase for good signatures by a factor that takes values between 1.1 and 3.5. We conclude that good Snort signatures tend to incorporate the entire arsenal of available features. Complexity in the case of signature writing is a virtue. Note, however, that this complexity typically comes at the cost of higher processing overhead for matching signatures.

Insight 4. IDS signature writing best practices often suggest that signatures should be kept short and simple. The primary reason for this is performance, since signature length and the usage of additional features handled by software modules, such as regular expressions, have a negative impact on the packet processing delay. Secondly, malware will often slightly change their behavior and communication patterns in order to evade detection. This results in a large number of similar network footprints generated from different flavors of the same malware. IDS signature writers cope with this problem by generalizing existing signatures, so that they effectively detect all different variants of a malware family. However, our work suggests that *reducing signature complexity will also reduce its effectiveness, since more generic signatures will often get triggered by benign traffic. Highly specific signatures exhibit higher true positive rate and generate in most cases a low number of alerts that can be easily handled by a security analyst.*

4.6 Building a Signature Effectiveness Metric

In Section 4.5 we identified a number of complexity features for the triggered signatures, and highlighted that *good* signatures will tend to exhibit much higher values compared to the *regular* ones. We next

ask the following question: is it possible to build a metric that would predict the effectiveness of a new signature based on its structural characteristics? Such a metric would be extremely useful to authors of new signatures, allowing them to evaluate the quality of the tailored rules before release, and also to security practitioners that wish to rank and prioritize signatures in the deployed IDS rulesets. To answer this question we first evaluate the correlation between the collected complexity features and the signature effectiveness. We explore whether a single complexity metric is sufficient or whether a group of complexity features are needed to capture the signature effectiveness. Then, we tailor a metric based on the most prominent set of complexity features and evaluate its performance. Finally, we apply this metric to evaluate the signatures contained in the most popular publicly available rulesets.

4.6.1 Evaluating Complexity Features

To investigate the predictive strength of the collected complexity features we first evaluate whether they are correlated to the signature effectiveness. Signature effectiveness for *good* signatures, as defined in Section 4.5, is equal to the number of validated infection incidents involving the signature, whereas it is equal to zero for *regular* signatures.

Table 4.3: *Complexity features and their correlations with signature effectiveness for 165* good *signatures.*

Complexity Feature	**Signature Class**					
	C&C Communication	*Reporting*	*Egg Download*	*Redirection*	*Propagation*	*All Signatures*
Bytes Checked	**0.95**	**0.52**	0.30	**0.88**	**0.65**	**0.84**
Fields Checked	**0.94**	**0.60**	**0.55**	**0.87**	**0.61**	**0.83**
Byte Offset is Set	**0.48**	0.28	**0.54**	**0.62**	0.34	**0.51**
Regular Expression is Set	**0.48**	0.27	**0.59**	**0.73**	0.35	**0.52**
Regular Expression size	**0.73**	0.24	**0.51**	**0.63**	0.30	0.37
Destination Port is Set	**0.49**	0.19	0.13	0.35	0.29	0.24
Packet Size is Set	0.19	0.12	0.19	0.16	**0.40**	0.12
Flow Options are Set	**0.91**	**0.78**	**0.60**	**0.41**	**0.48**	**0.68**
IP Options are Set	**0.53**	**0.42**	**0.48**	**0.53**	**0.73**	**0.56**

We use the Spearman correlation that measures the monotonic relationship between two input random variables. It is considered as a robust correlation technique [20], since it does not consider a linear relationship between the examined variables, such as in the case of Pearson correlation, and it is not sensitive to outliers. In Table 4.3 we present

the correlation results for different signature classes. We highlight the significant correlation values, at the 0.05 level, in bold.

First, we see that there are significant correlations among all signature classes for several complexity features. These correlations exceed in some cases 0.90 at the 0.05 significance level, indicating that there is a very strong association between the respective complexity features and the measured signature effectiveness.

Second, we observe that there is no single set of features that correlates with signature effectiveness for all signature classes. *C&C Communication* signatures detect the communication attempts between a victim host and its controller. Typically the victim machine will include several pieces of information regarding its state and id, the version of the rootkit used, and the type of request. This information is dispersed within different sections of the payload. Therefore, we see increased correlation values 0.94 and 0.95, for the *Bytes Checked* and *Fields Checked* features.

Reporting signatures are triggered whenever a victim host is attempting to leak sensitive data it has harvested from the victim host. Typically, this information is split into sections, which are preceded by a short string indicating the beginning of the section. Therefore, we see significant correlation for the *Bytes Checked* and *Fields Checked* features. Moreover, a communication flow needs to be established before the reporting takes place in most of the cases, which explains the 0.78 for the *Flow Options are Set* feature.

Egg Download signatures detect attempts made by the malware to update itself or to download additional badware. Typically, they try to match the string following the *GET* command of an HTTP request, corresponding to the target malicious binary. This request can slightly vary since the sender might include id or timestamp information, and the requested binary might come in different versions with slightly different filenames. Therefore, regular expressions are used to cope with this issue, explaining the increased correlation of 0.59 we observe for the *Regular Expression is Set* feature.

Redirection signatures detect requests towards potentially malicious domains. These signatures describe the target domain using either a string descriptor or a more flexible regular expression field. As a result *Bytes Checked*, *Fields Checked* and regular expression related features exhibit an increased correlation value. Finally, *Propagation* signatures

involve scanning and network enumeration detection, which often uses IP specific flags, such as the fragmentation bit, therefore, we see that the correlation value for the *IP Flags are Set* feature is high and equal to 0.73.

On the last column we see the respective correlation values between the complexity features and the signature effectiveness, considering all *good* signatures. We see that *Bytes Checked* and *Fields Checked* are the most prominent features, with correlation values of 0.84 and 0.83, followed by the *Flow/IP options are Set* and the *Regular Expression is Set* feature, with correlation values of 0.68, 0.56, and 0.52, respectively. However, these aggregate correlation results exhibit significant difference from the per-class correlation values, indicating that there is no single feature set that captures the signature effectiveness across all classes.

Insight 5. For each class of signatures there exists a set of complexity features that strongly correlate with signature effectiveness. However, there is no single set of features that fits all classes.

4.6.2 Building a Metric that Captures Signature Effectiveness

Our goal is to exploit complexity features that correlate with signature effectiveness to build predictor variables. This can be done by using a regression model that takes as input the complexity features as the explanatory variables and the signature effectiveness as the the response variable. However, an inherent problem that we need to deal with is the fact that the feature variables might exhibit inter-correlations. This is the known problem of feature multicollinearity, which means that one feature can be predicted from other input features with a non-trivial degree of accuracy [4]. Such inter-correlations among predictor variables can lead to unreliable and unstable estimates of regression coefficients, and therefore an inflated variance of the response variable, which in our model is the signature effectiveness.

To overcome this problem of inter-correlation among input features we employ *Principal Component Analysis* (PCA). PCA takes as input the original features and generates a new set of variables, which are called *principal components* (PCs). PCs are linear combinations of the original features and form an orthogonal basis, meaning that they are

linearly uncorrelated. Moreover, PCs are ordered so that the first few components retain most of the variation present in the original features. Therefore, we can reduce the dimensions of the feature space by only retaining the first few PCs. Fewer parameters in the regression modelling translate in lower variance of the regression coefficient estimator, or in other words an "easier" and more effective regression.

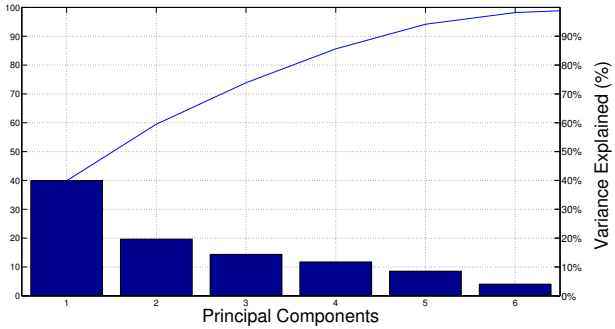


Figure 4.3: *Variance explained by different PCs for all* good *signatures*

In Figure 4.3 we illustrate the extracted PCs for the feature matrix consisting of all *good* signatures. We see that 6 PCs account for approximately 98% of the total variance of the dataset. This means that we can safely drop the remaining 3 PCs and, thus, reduce the feature space to 6 dimensions. Additionally, we perform the same transformation for five feature matrices derived from the remaining signature classes and one that corresponds to all *good* signatures. In all cases, 6 PCs account for at least 95% of the total variance and, therefore, suffice.

We then apply multiple linear regression using the PCs derived in the previous step as exploratory variables, and the signature effectiveness as responce variable. The goal of this process is to build a model that best describes the relationship between the exploratory and the response variables and can be used to perform prediction. In Table 4.4 we show the regression results for the derived models.

For each signature class, we list the R^2 value which is a statistic that measures the goodness of fit. It shows how closely the values generated by the model match the response variable. A value of 0.8 would mean that 80% of the variability in the original data is explained by

Table 4.4: *Regression models.*

Signature Class	R^2	Adjusted R^2	F-test
C&C Communication	0.939	0.848	91.8, $p < 0.001$
Reporting	0.737	0.614	9.9, $p < 0.001$
Egg Download	0.957	0.864	12.9, $p < 0.0005$
Redirection	0.961	0.910	151.9,$p < 0.001$
Propagation	0.917	0.856	6.9, $p < 0.005$
All Signatures	0.933	0.802	31.9, $p < 0.001$

the model, whereas an R^2 value of 1.0 suggests that this is a perfect fit. However, R^2 improves as exploratory variables are being added to the model, which might lead to artificially inflated R^2 values. Adjusted R^2 fixes this problem by penalizing the number of terms in the model, and therefore is more appropriate when comparing how different models fit the original data. The F-statistic tests the hypothesis that the coefficients that associate the exploratory variables to the response variable are zero. In essence, it tests whether the selected exploratory variables are useful in predicting the response variable. Values close to 1 suggest that there is no association whereas values greater than 5 at statistically significant levels signify that the selected exploratory variables are good predictors of the response variable.

In Table 4.4, the R^2 value indicates that the PCs extracted from the complexity features explain in all cases between 73% to 96% of the original variance, meaning that the derived regression model fits well the data. The adjusted R^2 ranging from 0.6 to 0.9 denotes that these results are robust and are not subject to bias introduced by the number of predictors used in the model.

To assess the predictive strength of our models we perform a stratified three-fold cross-validation. In this process the complete feature matrix, consisting of all *good* signatures, is split into three folds, two of which are used for training and one for testing. The folds are stratified so that each fold has on average the same mix of different signature classes. Based on the training data we build the predictors described in Table 4.4. We then compute the Spearman correlation between the predicted values and the signature effectiveness. A high correlation indicates that the predictor sensitivity is high, signifying that it can be used to estimate the signature effectiveness given the respective fea-

tures.

Table 4.5: *Regression model validation using 3-fold stratified validation.*

Signature Class	Split A	Split B	Split C	Class Average
C&C Communication	0.8628	0.8691	0.6652	0.7990
Reporting	0.6752	0.7269	0.9517	0.7846
Egg Download	0.9127	0.8282	0.8845	0.8751
Redirection	0.6940	0.1309	0.6713	0.4115
Propagation	0.8473	0.9443	0.5387	0.7768
All Signatures	0.7680	0.6211	0.6783	0.6893

In Table 4.5 we show the results of the validation. In total we evaluate 18 models derived from the 6 listed classes and the three dataset splits introduced by the stratified validation. We see that we get high correlations on average, above 0.41 in all cases, at the 0.05 significance level. If we exclude the *Redirection* signature class, which appears to produce the worst prediction results, the remaining classes exhibit correlations ranging from 0.68 to 0.87, suggesting that the generated models have a very high prediction accuracy. Moreover, we see that the generic model that uses all *good* signature complexity features, without taking into account class information, performs reasonably well with an average correlation of 0.6893.

However, how can one use these models in a realistic setting? Assuming that we have a new set of developed signatures, for which we have extracted the respective complexity features. If we apply our regression model on this data we will get a list of response values, corresponding to the estimated signature effectiveness. These values should be interpreted in a comparative fashion. Our metric is essentially a method to rank the evaluated signatures and assign a score of effectiveness that only makes sense when compared to the same score computed for other signatures.

To illustrate this process we perform the following experiment. We use the predictors generated based on the training folds of the stratified validation illustrated in Table 4.5. We apply these predictors to the respective testing fold, and additionaly apply the generic *All Signatures* predictor to the PCs that we have extracted from the complexity features of *regular* signatures. The predictor is useful in practice if the *good* signatures of the testing fold rank above the *regular* signatures in

terms of estimated effectiveness. We present the respective results in Table 4.6.

Table 4.6: *Predicted ranking of* good *signatures taking into account all 1423 signatures.*

Top N% (#number of signatures)	Percentage(%) and number(#) of *good* signatures
N=5 (#71)	47% (#24)
N=10 (#142)	79% (#40)
N=50 (#711)	94% (#48)

Table 4.6 shows that in the top 5% of the predicted most effective signatures are included 24 (47%) of the *good* signatures in the dataset, whereas 94% of the *good* signatures are in the upper half of the ranking. This result practically tells us that if a security specialist would have a new ruleset to deploy consisting of these 1423 signatures, then by using our predictor he could derive an estimate about which are the most prominent and reliable ones that he should focus on. He could then use this information to prioritize alerts generated by the deployed IDS and facilitate the forensics analysis.

Insight 6. Complexity features can be used to build regression models that allow the estimation of signature effectiveness. These prediction models can constitute a signature quality metric that can be exploited by security specialists to evaluate the available rulesets, prioritize the generated alerts, and facilitate the forensics analysis processes.

4.7 Related Work

Several studies have detected security incidents in traffic traces from production networks, e.g., [21, 29, 34], without providing though a systematic validation of detected incidents. Closer to our work, Sharma *et al.* [30] analyzed security incidents in a supercomputing center. The incidents were verified based on forensics analysis that exploited data from five security sources. The authors highlighted a number of best practices for the perimeter security of an organization. However, they do not provide insights about the effectiveness of the different security sources. In our recent short paper [26] we built a decision support

Table 4.7: *Effective Snort signatures in identifying malware infections for the 200 investigated incidents.*

SID	Signature Description
[**C&C Communication**] Update malicious binary instruction set.	
2007668	ET TROJAN Blackenergy Bot Checkin to C&C
2010861	ET TROJAN Zeus Bot Request to CnC
2404138:2404156,2404242: 2404247,2404335:240434	ET DROP Known Bot C&C Server Traffic TCP/UDP
16693	SPYWARE-PUT Torpig bot sinkhole server DNS lookup attempt
2011857	ET TROJAN SpyEye C&C Check-in URI
2013076	ET TROJAN Zeus Bot GET to Google checking Internet connectivity
2013348	ET TROJAN Zeus Bot Request to CnC 2
2013911	ET TROJAN P2P Zeus or ZeroAccess Request To CnC
2000348	ET ATTACK_RESPONSE IRC - Channel JOIN on non-std port
2014107	ET TROJAN Zeus POST Request to CnC - cookie variation
2015813	ET CURRENT_EVENTS DNS Query Torpig Sinkhole Domain
16140	BACKDOOR torpig-mebroot command and control checkin
[**Reporting**] Share stolen user confidential data with controller.	
2008660	ET TROJAN Torpig Infection Reporting
2011827	ET TROJAN Xilcter/Zeus related malware dropper reporting in
2009024	ET TROJAN Downadup/Conficker A or B Worm reporting
2802912	ETPRO TROJAN Backdoor.Nervos.A Checkin to Server
2002728	ET TROJAN Ransky or variant backdoor communication ping
2010150	ET TROJAN Koobface HTTP Request
2010885	ET TROJAN BlackEnergy v2.x HTTP Request with Encrypted Variable
2012279	ET CURRENT_EVENTS SpyEye HTTP Library Checkin
2002762	ET TROJAN Torpig Reporting User Activity
2008660	ET TROJAN Torpig Infection Reporting
2000347	ET ATTACK_RESPONSE IRC - Private message on non-std port
[**Egg download**] Update malicious binary/ Download additional malware.	
2010886	ET TROJAN BlackEnergy v2.x Plugin Download Request
2802975	ETPRO TROJAN Linezing.com Checkin
1012686	ET TROJAN SpyEye Checkin version 1.3.25 or later
2010071	ET TROJAN Hiloti/Mufanom Downloader Checkin
2011388	ET TROJAN Bredolab/Hiloti/Mufanom Downloader Checkin 2
2014435	ET TROJAN Infostealer.Banprox Proxy.pac Download
2007577	ET TROJAN General Downloader Checkin URL
2016347	ET CURRENT_EVENTS Styx Exploit Kit Secondary Landing
2011365, 2010267	ET TROJAN Sinowal/sinonet/mebroot/Torpig infected host checkin
[**Redirection**] Redirect user to malicious domain.	
2011912	ET CURRENT_EVENTS Possible Fake AV Checkin
2003494:2003496	ET USER_AGENTS AskSearch Toolbar Spyware User-Agent
2003626,2007854	ET USER_AGENTS Suspicious User Agent (agent)
2009005	ET MALWARE Simbar Spyware User-Agent Detected
2406001:2406012,2406147:2406167, 2406361:2406383,2406635:2406649	ET RBN Known Russian Business Network IP TCP/UDP
2016583	ET CURRENT_EVENTS SUSPICIOUS Java Request to DNSDynamic DNS
[**Propagation**] Detect and infect vulnerable hosts.	
2008802	ET TROJAN Possible Downadup/Conficker-A Worm Activity
2003068	ET SCAN Potential SSH Scan OUTBOUND
2001569	ET SCAN Behavioral Unusual Port 445 traffic
2003292	ET WORM Allaple ICMP Sweep Ping Outbound
2011104	ET TROJAN Exploit kit attack activity likely hostile
2010087	ET SCAN Suspicious User-Agent Containing SQL Inject/ion, SQL Scanner
2006546	ET SCAN LibSSH Based Frequent SSH Connections BruteForce Attack!
2001219	ET SCAN Potential SSH Scan
2003	SQL Worm propagation attempt
3817	TFTP GET transfer mode overflow attempt
12798:12802	SHELLCODE base64 x86 NOOP

tool that correlated evidence from different security sensors to expedite manual forensics analysis of compromised systems. [26] focused on the automation of the security assessment process. In contrast, in this paper we study in detail how the manual investigation of a wide range of incidents was carried out. In addition, we evaluate the complementary utility of the available security sources and highlight good practices for writing IDS signatures.

Besides, a group of studies on automating digital forensics analysis have focused on discovering and correlating evidence primarily from end hosts [3, 17, 35]. This line of work is orthogonal to ours, since their goal is to detect unauthorized user activity by combining the available host-level sources. Substantial part of these studies are centered around optimizing data representation so that evidence integrity and chain of custody is ensured [18,22]. Our work, on the other hand is to the best of our knowledge the first that systematically documents network forensics analysis practices in an operational environment.

Prediction methods have been the subject of extensive study in the field of vulnerability and software defect prediction. The goal in this line of work is to detect vulnerabilities and defective software modules at design time, based on the structural or behavioral characteristics of the respective code. Ohlsson et al. used graph based complexity metrics to detect software prone components in telecom systems [25]. Gyimothy et al. [19] applied logical and linear regression methods to assess the applicability of object-oriented metrics to estimate the fault-proneness of Mozilla. Neuhaus et al. used diverse code features, including imports, function call structure, and bug history, to predict the vulnerabilities of future Mozilla components [24]. Nagappan et al. used a long list of module, function, and class structural features to build a predictor of five Microsoft systems, and found that predictors built for individual projects only work well for similar projects [23]. Zimmermann et al. exploited historical bug data and code complexity features to predict vulnerable components [36]. Our work uses a similar modelling approach to solve a different problem. Instead of predicting software vulnerabilities we focus on building a metric that captures the effectiveness of IDS signatures.

4.8 Conclusions

In this paper we conducted a complex manual investigation of 200 suspected malware in a live operational infrastructure. We exploited four commonly-used security data sources and a number of derived features to validate suspected infections. Based on our experiment, we analyze the complementary utility of the different data sources and the characteristics of the IDS signatures that were associated with confirmed incidents. Notably, we observe that a search engine, which is a less well-established security data source, was much more useful in the diagnosis of security incidents than other more traditional security sources, namely blacklists, active scanners, and vulnerability analysis tools.

Furthermore, we learn that a single data source is typically not sufficient to validate an incident and that multiple sources should be combined. In more than 10% of the cases, no single source, but the overall behavior of a host as seen from multiple sources helped to validate an infection. In addition, multiple sensors are needed in 70.5% of all cases when the easier to detect spyware are excluded. These results highlight the importance of a holistic approach in security assessment that combines multiple data sources. In order to detect elaborate pieces of malware, as the ones shown in Table 4.1, we need to combine local information about the exposure level and the behavioral patterns of studied hosts with public knowledge about the maliciousness of contacted domains. In this context future work could also use a variety of tools to complement some of the sensors used in our study. For example, security inspectors (e.g. SecuniaPSI, QualysGuard), IDSs (e.g. BRO, Surricata), and NetFlow anomaly detectors can also be used to detect malicious activity, whereas spamtraps, blocklists and DNS-based reputation engines can be exploited to build profiles of contacted domains.

Finally, we extracted and made available a list of 165 Snort signatures that were triggered on hosts with validated malware. We compare the characteristics of good and regular signatures and report a number of interesting statistics that provide essential guidelines for configuring Snort and for teaching good signature writing practices. In addition, we introduce a novel signature quality metric that can be exploited by security specialists to evaluate the available rulesets, prioritize the generated alerts, and facilitate the forensics analysis processes. Based on our metric, we compare the most popular signature rulesets and

highlight differences.

Bibliography

[1] Sourcefire Vulnerability Research Team. http://www.snort.org/vrt.

[2] SRI Malware Threat Center. http://www.snort.org/vrt.

[3] Andrew Case, Andrew Cristina, Lodovico Marziale, Golden G. Richard, and Vassil Roussev. Face: Automated digital evidence discovery and correlation. *Digit. Investig.*, 5:S65–S75, September 2008.

[4] Samprit Chatterjee and Bertram Price. A Wiley-Interscience publication. Wiley, New York [u.a.], 2. ed edition.

[5] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-wei Wang. Regular expressions: new results and open problems. *J. Autom. Lang. Comb.*, 9(2-3):233–256, September 2004.

[6] Advanced automated threat analysis system. http://www.threatexpert.com.

[7] Anonymous postmasters early warning system. http://www.apews.org.

[8] Best IDS/IPS solution. http://www.scmagazine.com/best-idsips-solution/article/130871/.

[9] Bleeding Edge Snort Rules. http://www.bleedingsnort.com.

[10] Cooperative Network Security Community - Internet Security. http://www.dshield.org.

[11] Emerging Threats. http://www.emergingthreats.net.

[12] Shadowserver Foundation. http://www.shadowserver.org.

[13] The Case for Open Source IDS. http://www.itsecurity.com/features/the-case-for-open-source-ids-022607/.

[14] The Nessus vulnerability scanner. http://www.tenable.com/products/nessus.

[15] The Open Vulnerability Assessment System. http://www.openvas.org.

[16] The Urlblacklist web page. http://www.urlblacklist.org.

[17] Simson L. Garfinkel. Forensic feature extraction and cross-drive analysis. *Digit. Investig.*, 3:71–81, September 2006.

[18] S.L. Garfinkel. Automating disk forensic processing with sleuthkit, xml and python. In *IEEE SADFE '09*.

[19] Tibor Gyimthy, Rudolf Ferenc, and Istvn Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31:897–910, 2005.

[20] A. Lehman, N. O'Rourke, and L. Hatcher. *Jmp for Basic Univariate and Multivariate Statistics: Methods for Researchers and Social Scientists, Second Edition*. SAS Institute, 2013.

[21] Gregor Maier, Anja Feldmann, Vern Paxson, Robin Sommer, and Matthias Vallentin. An assessment of overt malicious activity manifest in residential networks. In *DIMVA*, 2011.

[22] J. Mena. *Investigative Data Mining for Security and Criminal Detection*. Butterworth-Heinemann Limited, 2003.

[23] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 452–461, New York, NY, USA, 2006. ACM.

[24] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components, 2007.

[25] Niclas Ohlsson and Hans Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Trans. Softw. Eng.*, 22(12):886–894, December 1996.

[26] Elias Raftopoulos and Xenofontas Dimitropoulos. Shedding light on log correlation in network forensics analysis. In *DIMVA'12*.

[27] Elias Raftopoulos and Xenofontas Dimitropoulos. Detecting, validating and characterizing computer infections in the wild. In *ACM SIGCOMM IMC*, Berlin, Germany, November 2011.

[28] Elias Raftopoulos and Xenofontas Dimitropoulos. Technical report : Shedding light on data correlation during network forensics analysis. Technical Report 346, 2012.

[29] Stefan Saroiu, Steven D. Gribble, and Henry M. Levy. Measurement and analysis of spyware in a university environment. In *NSDI*, pages 141–153, 2004.

[30] Aashish Sharma, Zbigniew Kalbarczyk, James Barlow, and Ravishankar K. Iyer. Analysis of security data from a large computing organization. In *DSN*, 2011.

[31] Sushant Sinha, Michael Bailey, and Farnam Jahanian. Shades of grey: On the effectiveness of reputation-based blacklists. In *MALWARE'08*, pages 57–64, Fairfax, Virginia, USA.

[32] A free lightweight network intrusion detection system for UNIX and Windows. `http://www.snort.org`.

[33] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovi, and Antonio Nucci. Unconstrained endpoint profiling (googling the internet). In *ACM SIGCOMM'08*, NY, USA.

[34] Ting-Fang Yen and Michael K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the 5th international conference on*, DIMVA, Berlin, Heidelberg, 2008.

[35] Yuanyuan Zeng, Xin Hu, and K.G. Shin. Detection of botnets using combined host- and network-level information. In *DSN'10*, pages 291 –300.

[36] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, Washington, DC, USA, 2007.

Chapter 5

The Day After: Studying the Aftermath of a "/0" scan from the Sality botnet

Elias Raftopoulos[1], Eduard Glatz[1], Alberto Dainotti[1], Xenofontas Dimitropoulos[1]

[1]Computer Engineering and Networks Laboratory,
ETH Zurich, Switzerland
[2]CAIDA, University of California San Diego, USA
[3]Foundation for Research and Technology, Greece

Abstract —
Although Internet scanning is one of the most popular malware propagation methods, sound measurements about its success rate are not generally available. In this work, we assess the success rate of an Internet-wide scanning event that was orchestrated by the Sality botnet during February 2011 using data from a university network. We first use unsampled NetFlow records from the border router of the network to find how many targetted hosts replied to the scanners. Second, we correlate the replies with IDS alerts triggered in the same network and uncover significant exploitation activity that followed toward the scan repliers. In our data, 2% of the scanned hosts replied and at least 8% of the repliers we believe were eventually compromised. Furthermore, we characterize the exploitation activity and find surprisingly that scanners and exploiters came from different geographical locations. Our analysis provides a novel look into the success rate of Internet scanning in the wild based on two unique data-sets.

5.1 Introduction

Botnets of up to millions of compromised computers are presently the most widely-used cyberweapon for executing criminal activities, such as fraud, sensitive data leakage, distributed denial-of-service attacks, and spam. Botnets engage into large-scale scanning to enumerate vulnerable hosts for targeted criminal activities or simply propagation [1,2]. A recent study showed that scanning accounts for 34-67% of all connections (successful and unsuccessful) in an ISP [3]. Besides, recent advances in scanning software make possible to scan the entire IPv4 address space in less than 45 minutes [4], simplifying further the execution of aggresive scanning attacks. In spite of the prevalence of scanning, measurements about its success rate in the Internet are not generally available. Network administrators often ignore scanning as a baseline noise that does not pose a significant threat.

In this work, we use unsampled Netflow records and IDS alerts collected from a university network to assess the success rate of an Internet-wide scanning event. In particular, we focus on the "sipscan", an Internet-wide scanning event orchestrated from the Sality botnet over 12 days in February 2011 that was uncovered by Dainotti *et al.* [5].

This event had several unique characteristics over previously known scanning attacks: 1) it used a well-orchestrated stealth scanning strategy; 2) it originated from 3 million IP addresses; 3) it is believed that it scanned the entire Internet address space; and 4) it was targeted towards Session Initiation Protocol (SIP) [6] servers.

We discovered that the scanning event escalated into persistent exploitation attempts towards the hosts that replied to the sipscan. We use our data to assess the effectiveness of scanning in terms of scan repliers and hosts that were eventually compromised. We find that 2% of the scanned IP addresses replied and at least 8% of the repliers were eventually compromised. Besides, our analysis shows that scanners originated primarily from Eastern countries, while the subsequent exploitation attempts originated from Western countries. This suggests that information about scan repliers was communicated to the subsequent attackers likely through underground channels. Moreover, we observe 352,350 new scanner IP addresses and show that the sipscan was largely undetected by the IDS, which only raised alerts for 4% of the scan probes.

In summary, our work makes the following **contributions**:

- We assess the effectiveness of an Internet-wide scanning event. To the best of our knowledge, this is a first measurement study that focuses on Internet scan repliers.

- We uncover and characterize how an interesting /0 scan escalated to exploitation activity. Among other observations, our analysis shows that the subsequent exploitation attempts came from different geographical locations and that the sipscan originated from 352,350 more IP addresses than previously seen.

The rest of the paper is structured as follows. We first discuss related research in Section 5.5. In Section 5.2 we describe the used data-sets. Then, Section 5.3 presents how unsampled NetFlow records were used to detect the sipscan and measure scan repliers. Then, in Section 5.4 we characterize the exploitation activity that followed based on our IDS data. Finally, Section 5.5 discusses the impact of false-positive IDS alerts on our analysis and Section 5.6 concludes our paper.

5.2 Monitoring Infrastructure and Data Collection

In this section, we describe the monitored network and the data we use in this study. We collected our measurements from the network of the main campus of the Swiss Federal Institute of Technology at Zurich (ETH Zurich). The ETH Zurich network is large and diverse. During our data collection period, which spanned 5 months (between the 1st January and the 31th of May 2011), we observed in total 79,821 internal hosts. On these hosts, the IT policy grants full freedom to users regarding the software and services they can use.

We select two data sources that provide complementary views into the studied event. Firstly, we collect unsampled NetFlow data from the only upstream provider of ETH Zurich. Netflow produces summary records for *all* flows crossing the monitoring point. However, Netflow lacks context, since it does not provide information regarding the type of activity that triggered a flow. To fill this gap, we use IDS data collected from a Snort sensor, which captures and analyzes all traffic crossing our infrastructure's border router. Snort uses signature-based payload matching to perform protocol analysis, revealing more information about the type of activity that triggered an observed packet sequence. The two passive monitoring tools complement each other, since they capture flow summaries for all traffic and finer (payload/-header) details for packets that trigger IDS signatures.

5.2.1 Netflow Data

We use unsampled NetFlow records collected at SWITCH, a regional academic backbone network that serves 46 single-homed universities and research institutes in Switzerland [21]. The hardware-based NetFlow meters capture all the traffic that crosses the border destined to or coming from the Internet. In a single peering link, we observe in 2011 on average 108.1 million flows per hour, which correspond to 3,064 million packets. From each flow record we extract the following fields: IP addresses, port numbers, protocol number, byte/packet counts, and timestamp. We do not use TCP flags because they are not supported by the fast hardware-based NetFlow. We dissect NetFlow records into one- and two-way flows using a number of preprocessing steps, which

are described in detail in [3]. We first assign each flow to a time interval by its start time, then defragment flows which have been fragmented into multiple NetFlow records, and finally pair two-way flows. For TCP and UDP, a two-way flow is the aggregate of two defragmented flows that have the same 5-tuple with reverse values in the source and destination IP address and port fields. Accordingly, a one-way flow is a flow that does not have a matching reverse flow. We search for matching one-way flows in the same and adjacent time intervals. In [3] we also describe how we eliminate double-counting of flows and handle the problem of asymmetric routing. For this study, we focus on those flows that involve the IP address range allocated to ETH Zurich. We analyze flow data for the first 400 hours of February 2011 which is equivalent to 16.7 days.

5.2.2 IDS Alerts

We collect alerts generated by the popular Snort IDS in the border router of the monitored network. The IDS monitors both downstream and upstream traffic and triggers an alert when a packet matches a signature. Snort is configured with the two most widely-used signature rulesets, namely the Vulnerability Research Team (VRT) and the Emerging Threats (ET) rulesets. The former is the default Snort ruleset, which accounts for 5,559 rules, developed and maintained by Sourcefire. The latter is an open-source ruleset, which is maintained by contributors and accounts in total for 11,344 rules. By using both rulesets, our Snort sensor is sensitive to a wide range of possibly malicious events, including reconnaissance scanning, active attacks, malware spreading and communication, data leakage and host compromise.

The collected alerts have the standard full Snort format. For example, the following is a medium priority scan alert, that was triggered by SIPVicious inbound traffic. The SIPVicious suite is a set of tools that can be used to enumerate and audit SIP-based VoIP systems. The IP addresses in this example have been anonymized:

```
[**] [1:2011766:4] ET SCAN Sipvicious User-Agent Detected [**]
[Classification: Attempted Information Leak] [Priority: 2]
01/01-01:57:00.272686 a.b.c.d:12312 -> m.n.p.r:5060
UDP TTL:48 TOS:0x0 ID:0 IpLen:20 DgmLen:439 DF Len: 411
```

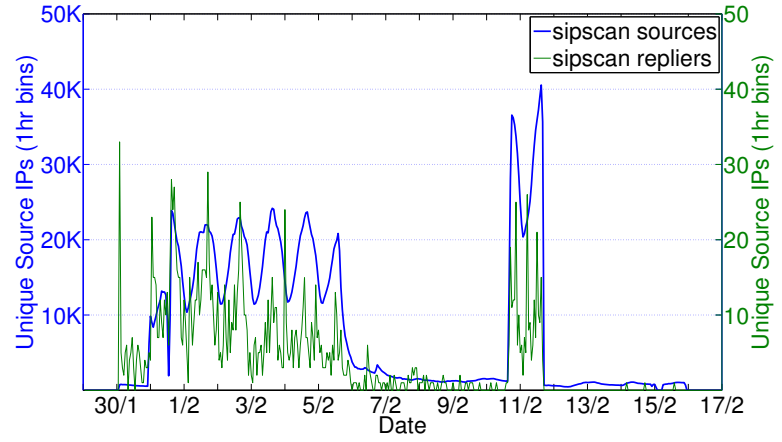
We have tailored a Perl module that parses raw IDS alerts and extracts a set of features relevant to our analysis. Specifically, the fields

that we use are the rule identification number (2011766), the rule description (ET SCAN Sipvicious User-Agent Detected) and classification (Attempted Information Leak) that provide the context of the security event, the rule priority which is a severity measure, the event timestamp (01/01-01:57:00.272686) , and the involved IP addresses and port numbers (a.b.c.d, 12312, m.n.p.r, 5060). In parentheses we illustrate the respective extracted values for the Sipvicious scan example shown above.

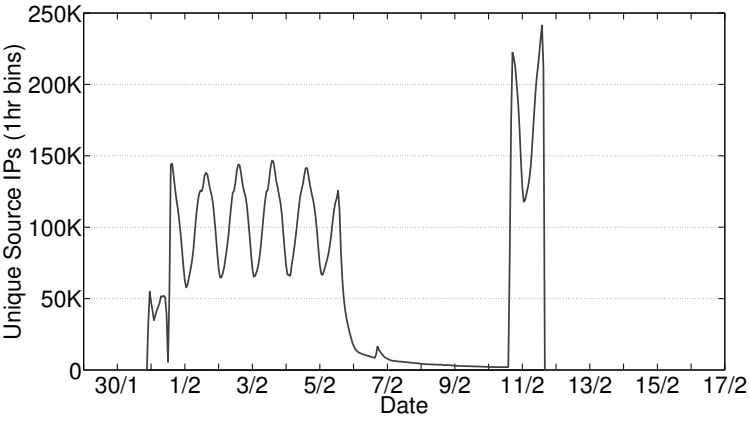
5.3 Sipscan Detection

To extract sipscan traffic from NetFlow data, we rely on heuristics introduced by Dainotti *et al.* [5], which are based on the analysis of the payload of sipscan packets. However, because flow data do not include packet payload contents, we adapted the extraction rules. We focus on the UDP part of sipscan traffic, which is sufficient to detect sipscan activity and identify sipscan sources. Specifically, we identify a sipscan flow as a single-packet one-way flow towards port 5060/udp having a size in the range of 382 to 451 bytes.

In Figure 5.1a, we highlight how the host population sourcing attacks towards the SIP service port evolved over 16.7 days (from 31/01/2011 to 16/02/2011). In Figure 5.1b, we illustrate how the same event was captured by the UCSD network telescope. Note that Dainotti *et al.* [5] used full packet traces collected at the network telescope in order to estimate the scanning population. The similarity in these two patterns, indicates that our heuristics adapted to Netflow records are able to capture the same phenomenon as seen on our network. We observe two major sipscan outbreaks in terms of participating attackers along with a minor fraction of hosts engaged continuously in SIP scanning. The first outbreak starts at 2011.01.31 21:30 UTC and lasts until approximately 2011.02.06 22:40, while the second outbreak starts at 2011.02.11 14:10 and lasts until 2011.02.12 15:00 UTC. In total, 952,652 scanners participated in the scan. A significant number (352,350) of hosts targeting our infrastructure were not observed in the population of Sality scanners detected by the UCSD network telescope, which were 2,954,108 [5]. This finding indicates that the size (expressed in terms of source IP addresses) of the botnet was at least 11.9% larger than the lower bound estimated in the previous work.



(a) ETH Zurich (NetFlow trace)



(b) UCSD network telescope (full packets trace)

Figure 5.1: *Number of IP addresses per hour sourcing or replying to scan flows in ETH Zurich and in the UCSD network telescope.*

At the victim side, 77,158 hosts within ETHZ were scanned at least once during the 16.7 days period, meaning that the coverage of the scan in our infrastructure was 96.6%. The scan was largely stealthy, in terms of generated alerts from the IDS, since only 4% of the respective

probing flows triggered a scan-related IDS signature.

In contrast to [5], our data set allows us to identify those target hosts that reply to the sender of a sipscan flow. For this purpose, we search for two-way flows matching a relaxed filter (i.e., requiring port 5060/UDP only). Additionally, we look at the number of attacker-victim host pairs where a sipscan flow is answered with an ICMP flow. For this answer type, we see a weak correlation of ICMP flow counts with the two sipscan outbreaks. On the other hand, when looking at host pairs where we have biflows, we observe a strong correlation of biflow counts with the sipscan outbreaks indicating that sipscan attacks significantly result in bidirectional communication between attacker and victim. In Figure 5.1a we present the number of unique internal IP source addresses responding to the sipscan. In total, we identify 1,748 sipscan repliers, whereas during the scan we find 3.8 new unique internal IPs responding to the scan every hour. For 80.2% of the repliers we detected a TCP reply originating from the respective host, whereas for 8.3% of the repliers, the sipscan was answered with an ICMP flow. 0.2% of the replies involved both a TCP and an ICMP flow, while the remaining 11.5% used neither TCP or ICMP.

5.4 Aftermath of the Sipscan

5.4.1 Inbound exploitation attempts

In this section, we study the impact of the sipscan on the target host population within ETH Zurich. We first investigate if scanning was a precursor of subsequent exploitation attempts targeting hosts that replied to the scanners. Recall that our IDS data cover 5 months, including one month before the beginning of the sipscan (31/01/2011) and approximately 3.5 months after its end (16/02/2011).

In Figure 5.2, we show how the daily number of exploitation alerts per target host triggered by inbound traffic changed after the sipscan. We consider alerts of the VRT rulesets *exploit.rules*, *exploit-kit.rules*, and *inidicator-shellcode.rules* and of the ET ruleset *emerging-exploit.rules*. These rulesets are tailored to detect exploitation activity, including buffer overflow attacks, remote command execution, brute force authorization and privilege escalation attempts. In Figure 5.2, we also show the daily number of exploitation alerts per target host for the

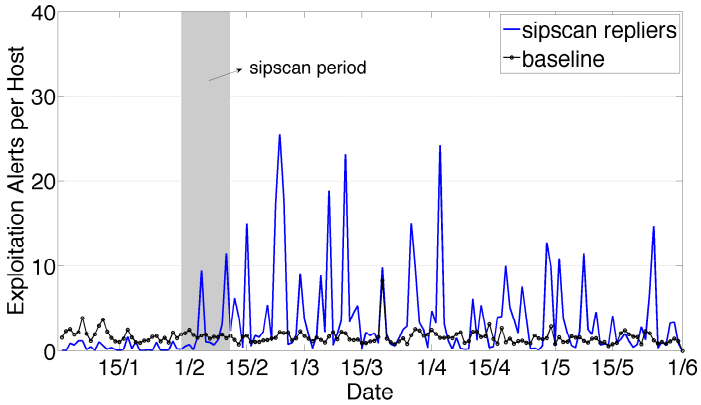


Figure 5.2: *Daily number of inbound exploitation alerts per target host over a period of 5 months. The two lines mark hosts that replied and that did not reply (baseline) to the sipscan. The shaded region marks the duration of the sipscan.*

baseline, i.e., the ETH Zurich hosts that did not reply to the scanners according to our data. The *baseline* accounts for 78,073 hosts, whereas the number of sipscan repliers is 1,748. In the pre-sipscan period sipscan repliers were involved on average in 122 exploitation alerts per day. During the sipscan period we see that this number increases to 842 alerts per day, whereas after the sipscan it remains high at 931 alerts per day. In sharp contrast, the inbound exploitation activity associated with the *baseline* remains low after the sipscan. On average, each host is a target in 1.2 alerts per day, which is a baseline noise caused by automated threats attempting to propagate and false alerts. The respective noise level for the sipscan repliers in the pre-sipscan period is 0.4 alerts per day. After the sipscan, this number increases to 3.7 alerts per day. The high number of exploitation alerts towards sipscan repliers persists even 4 months after the end of the sipscan, although it is more intense during the first two months (from 31/1 to 28/2), when 68% of the total exploitation alerts are triggered. Out of the 1,748 sipscan repliers, we observe that 852 were involved in inbound exploitation alerts.

In addition, we explore how persistent the attacking hosts are in

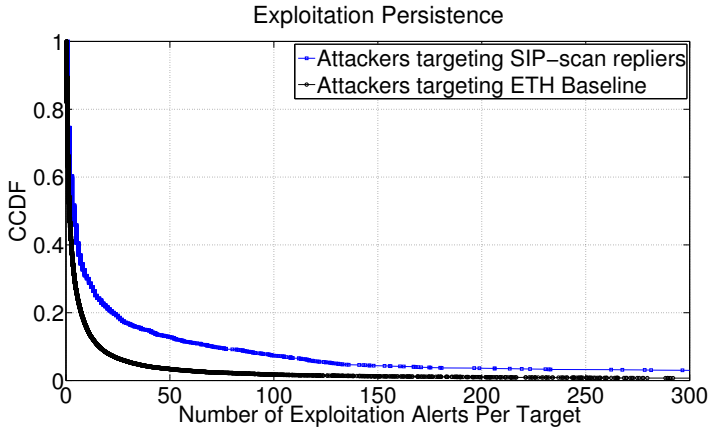


Figure 5.3: *Persistence of exploitation attackers for sipscan repliers and ETH-Baseline.*

terms of generated exploitation alerts, and examine whether the attackers targeting the sipscan repliers are more persistent compared to the ones targeting the baseline. In Figure 5.3, we compare the average number of exploitation alerts per target for sipscan repliers and baseline attackers, respectively. We see that the former group tends to be more persistent triggering in the median case 4 exploitation alerts per target, whereas the same number for the latter group is 2 alerts. The increased persistence towards sipscan repliers is more prominent in the tails of the distributions. We see that the top 10% most active attackers towards sipscan repliers launch up to 73 alerts on average per target, whereas the respective number for the baseline is only 21 alerts.

Next, we study whether the observed increase in exploitation activity comes from new offenders. Figure 5.4 illustrates the daily number of new offending IP addresses per target host for sipscan repliers and for the baseline. We report IP addresses that appear in exploitation alerts, however we consider an address new only when it has not previously appeared in the entire alert trace. A baseline host records a new external attacker approximately every four days consistently throughout the 5-month period. However, this number increases sharply for sipscan repliers during the sipscan, when each victim is attacked on average by 1.4 new IP addresses per day. Moreover, we investigate whether

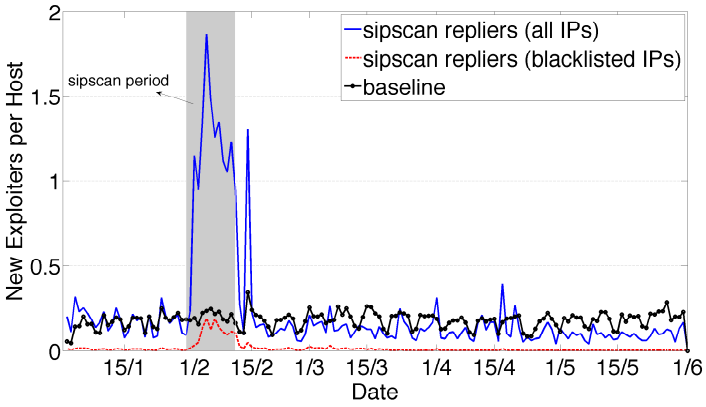


Figure 5.4: *Daily number of new offending IP addresses per target host for sipscan repliers and the baseline.*

these IP addresses are known blacklisted hosts using four public blacklists [22–25]. Figure 5.4 shows that only 7% of the new offenders were already blacklisted, while this number drops to almost 0 before and after the sipscan period.

Table 5.1: *Top 10 countries used by the sipscanners compared to the respective countries for exploitation attack originators. Geolocation data for sipscan sources and exploitation attack originators was obtained using the MaxMind GeoIP Lite Database [26].*

	sipscanners	*CAIDA*	*sipscanners*	*ETH*	*Exploiters*	*ETH*
Rank	%	Country	%	Country	%	Country
1	12.55	Turkey	10.06	Indonesia	27.11	United States
2	12.54	India	9.72	Turkey	12.70	Canada
3	8.64	Brazil	7.32	China	9.90	China
4	7.23	Egypt	6.86	Brasil	7.01	Switzerland
5	5.77	Indonesia	6.52	Egypt	4.98	Germany
6	5.59	Romania	5.94	India	4.78	Taiwan
7	5.58	Russian Federation	4.80	Thailand	4.31	Japan
8	5.36	Vietnam	4.06	Philippines	3.31	India
9	5.10	Thailand	3.71	Russian Federation	2.95	Russian Federation
10	3.01	Ukraine	3.20	Romania	2.88	Brazil

We also investigate the similarity between the IP addresses of scanners (extracted from NetFlow) and of exploiters (extracted from Snort

alerts towards sipscan repliers). Surprisingly, we observe that out of 6,676 exploiter and 1.3 million scanner IP addresses, only 17 are in common. This suggests that there is a clear separation between scanners and bots wielded to exploit target hosts. In Table 5.1, we compare the geographical distribution of the scanners detected in our infrastructure and in the UCSD network telescope [5] with the exploiters targeting the ETH Zurich sipscan repliers. The geographical distribution of scanners seen in the UCSD network telescope and in ETH Zurich is very similar with the exception of China. In our dataset China is a significant source of SIP scanning accounting for 9.90% of the total scanners population. On the UCSD dataset China is ranked 27th. More importantly, the geographical distribution of exploiters is particularly interesting, since it is dominated by Western countries and United States in particular, which is the most strongly represented country with 27.11% of the exploiters. In contrast, the geographical distribution of scanners is dominated by Eastern countries. US is not sourcing sipscanning, which is remarkable since the analysis of the botnet has shown a strong presence in the United States [27]. This observation shows that information about scan repliers was communicated from scanning to attacking bots through unknown channels.

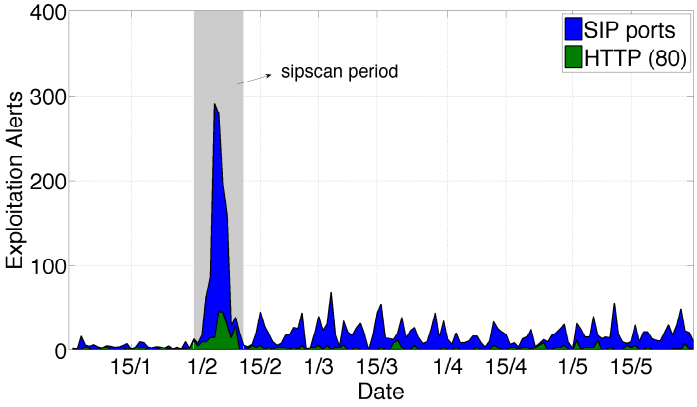


Figure 5.5: *Alert volume for exploitation attempts targeting SIP related ports.*

Next, we examine the exploitation activity on port numbers related

to SIP. Figure 5.5 shows the number of exploitation alerts targeting sipscan repliers on ports 5060, 5061, 5070 and 80. Ports 5060, 5061 and 5070 are used by SIP for control and data traffic. Moreover, the sipscan binary attempts to open a connection and gain administration privileges on port 80, where an HTTP server may provide remote administration to SIP servers [28]. Figure 5.5 shows a sharp increase of exploitation activity targeting SIP ports during and after the sipscan. Before, the sipscan we observe on a daily basis less than 12 exploitation alerts targeting SIP ports and 3 alerts targeting port 80. During the sipscan period, these numbers jump to 135 and 27, respectively, exhibiting approximately a ten-fold increase. Moreover, during the sipscan period 22% of all inbound exploitation alerts are on SIP ports. In the post-scan period we observe that these values drop, but still remain significant compared to the pre-sipscan period. Specifically, the daily number of exploitation alerts targeting SIP ports and port 80 are 5 and 21, respectively.

Finally, we inspect whether the inbound exploitation attacks against SIP ports occur in closer temporal proximity to the sipscan than attacks targeting non-SIP ports. In Figure 5.6 we compute the time between the first scan flow and the first exploitation alert towards a victim. We differentiate between exploitation alerts that target SIP ports and all other ports. In the former case, we see that in 98% of the incidents the attack occurs within one week after the sipscan event, whereas the median is 2.7 days. In the latter case 95% of the incidents occur within one month after the sipscan event, whereas the median is 4.2 days.

To summarize the key findings of this section, we first observe a steep increase in exploitation alerts against sipscan repliers right after the sipscan, which is associated only with sipscan repliers and not with other hosts in the monitored infrastructure. Second, we observe that the attackers associated with the increase appear for the first time during the sipscan and were not active before. Third, we observe a sharp increase in exploitation alerts towards SIP ports and show that these exploitation attempts happen in close temporal proximity to the sipscan. We believe these findings constitute sufficient evidence that the sipscan was the precursor of a subsequent large-scale exploitation activity targeting sipscan repliers.

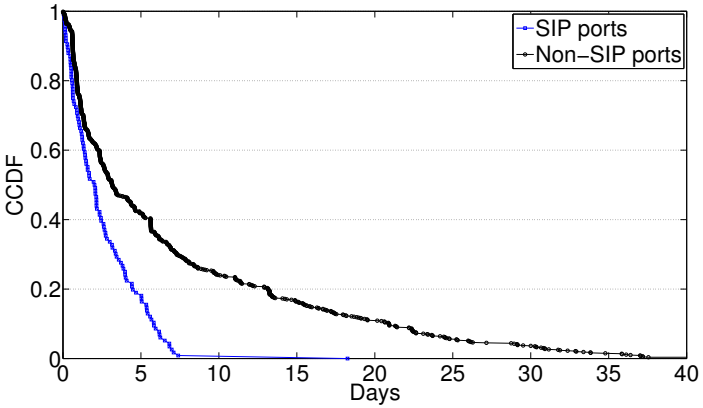


Figure 5.6: *CCDF of time between scanning and inbound exploitation attempt of a host for alerts on SIP and all other ports.*

5.4.2 Sality alert classification and outbound exploitation activity

In Sections 5.3 and 5.4, we analyzed the inbound scanning and exploitation activity towards the monitored network. In this section, we shift our attention to the outbound IDS alerts generated by sipscan repliers and analyze the new behavioral patterns that emerge.

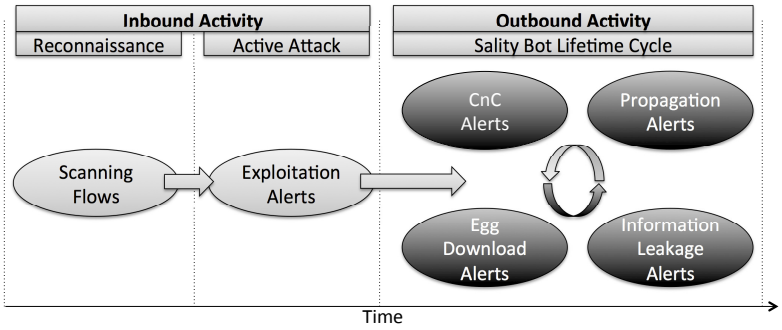


Figure 5.7: *Sality bot lifecycle*

Sality is a very sophisticated and resilient malware. It consists of three main components: an infector, a communicator, and a downloader. The infector employs advanced propagation techniques to spread by infecting executable files and replicating on removable drives and network shares. The malware is polymorphic and encrypted, which makes its detection and remediation using traditional signature-based anti-viruses very challenging. The communicator uses a C&C architecture to pull instructions from the botmaster and to push harvested confidential data from the victim. Older versions of Sality utilized a centralized HTTP-based C&C channel, whereas recent versions form an unstructured P2P network, where each peer can be used to disseminate instructions within the botnet and to receive and forward information from other bots. The communicator is responsible for fetching a list of URLs hosting malware, which are subsequently fed to the downloader. In this way, the Sality botnet can be leveraged to push additional malware to the infected population, operating as a pay-per-install infrastructure [29].

Figure 5.7 illustrates the different stages that a Sality bot undergoes during its lifetime. The first two stages correspond to the enumeration and active exploitation of the victim, which occur during the pre-infection phase. After the victim has been compromised, the Sality bot will typically undergo through a cycle of recurring actions. First, it will frequently contact its C&C servers to receive instructions and update its list of bootstrap nodes. Second, it will attempt to fetch malicious eggs to either update itself or to install additional malware on the victim. Third, it will try to leak sensitive information harvested from the victim, either by directly sending this information to a C&C server or by exploiting popular services and social networks, such as Dropbox and Facebook, to exfiltrate the data. Finally, it will attempt to propagate to vulnerable hosts by exploiting vulnerabilities related to the remote desktop and network shares services.

In Table 5.2, we list the Snort identifiers (SIDs) and their official short description for relevant signatures that are triggered in our data. To compile the list, we manually analyzed the outbound alerts generated by sipscan repliers. We found the new types of alerts that emerged in the post-scan period and inspected their signatures in order to identify specific behaviors. We group signatures into four categories shown in Table 5.2.

Table 5.2: *Snort signatures related to Sality bot lifecycle.*

SID	Signature Description
[**C&C Communication**] Communication with botnet controller.	
2404138:2404156	ET DROP Known Bot C&C Server Traffic TCP/UDP
2000348	ET ATTACK_RESPONSE IRC - Channel JOIN on non-std port
2000334	ET P2P BitTorrent peer sync
2009971	ET P2P eMule KAD Network Hello Request
2008581	ET P2p BitTorrent DHT ping Outbound
2010142	ET P2P Vuze BT UDP Connection Outbound
2008584	ET P2P BitTorrent DHT announce_peers request
2181	P2P BitTorrent transfer
[**Exfiltration**] Possible leakage of sensitive user data.	
5	SENSITIVE-DATA Email Addresses Outbound
2006380	ET Policy Outgoing Basic Auth Base64 HTTP Password detected unencrypted
2010784	ET CHAT Facebook Chat POST Outbound
2000347	ET ATTACK_RESPONSE IRC - Private message on non-std port
1463	CHAT IRC message Outbound
[**Propagation**] Attempted infection of vulnerable hosts.	
2007695,2008070	ET User-Agent Malware overflow attempt
4060	POLICY RDP attempted administrator connection request
2006546	ET SCAN LibSSH Based SSH Connection - BruteForce Attack
2002383	ET SCAN Potential FTP Brute-Force attempt
3817	TFTP GET transfer mode overflow attempt
2010643	ET SCAN Multiple FTP Administrator Login Attempts- Brute Force Attempt
2001972	ET SCAN Behavioral Unusually fast Terminal Server Traffic, Potential Scan or Infection
2001569	ET SCAN Behavioral Unusual Port 445 traffic
[**Egg Download**] Possible download of malicious executable.	
2009897	ET MALWARE Possible Windows Executable sent when remote host claims to send a Text File
19270	POLICY attempted download of a PDF with embedded Javascript
15306	WEB-CLIENT Portable Executable binary file transfer
2003546	ET USER Agents Suspicious User agent Downloader
2007577	ET TROJAN General Downloader Checkin URL
2012648	ET Policy Dropbox Client Downloading Executable
2009301	ET Policy Megaupload file download service access

Signatures in the group *C&C Communication* detect the activity triggered by a bot when calling its controller for instructions. In the case of the HTTP version of the Sality bot, the signatures in the SID range *(2404138:2404156)* are triggered when a set of known blacklisted C&C servers are contacted, whereas the signature *(2000348)* detects the setup of an IRC channel, which is used by the bot and the controller to communicate. The remaining alerts are related to the P2P version of the bot and are triggered when the bot is either attempting to join the P2P network, instantiating a new P2P connection, or fetching the latest peers list.

Signatures in the group *Exfiltration* are tailored to detect the exfiltration of confidential data. The SIDs *(5,2006380)* are triggered when passwords or email addresses are sent from the intranet unencrypted. The signature *(2010784)* is triggered when the bot is attempting to leak sensitive information using Facebook's POST mechanism. This alert should be expected to generate a significant amount of false posi-

tives, since it is also triggered when a user sends a legitimate Facebook message. However, a sudden sharp increase in the amount of Facebook POST operations could signify a malicious activity. The signatures with SIDs *(2000347,1463)* are triggered when information is exfiltrated using an IRC channel.

Signatures in the group *Propagation* are generated when the bot is attempting to infect exposed vulnerable hosts. The main targeted vulnerabilities are the MS-LSASS buffer overflow and the MS-WebDav vulnerability related to services used for accessing remote network shares. The set of signatures shown in Table 5.2 are fine-tuned to detect brute force privilege escalation attacks (*4060,2006546,2002383,2010643*), buffer overflow exploitation attempts (*2007695, 2008070,3817*), and targeted scanning on these services (*2001972 ,2001569*).

Finally, signatures in the group *Egg Download* correspond to attempts made by the bot to fetch a malicious binary from a remote domain. The downloaded executable can be either an update of Sality's own code or can correspond to a new malware pushed to the infected population. Signatures with SIDs (*15306,2003546,2007577*) detect the activity of Sality's downloader module when attempting to check a suspicious URL or when a binary download is initiated. Sality tries to obfuscate the downloaded binary by hiding it in seemingly legitimate files, such as Text and PDF documents. This activity is detected by signatures with SIDs (*2009897,19270*). The obfuscation is used to evade detection by cloud providers, such as Dropbox and Megaupload, which are exploited in order to host the malicious content. Signatures with SIDs (*2012648,2009301*) detect the download of executables from these sites.

Figure 5.8 shows the average number of C&C alerts triggered by sipscan repliers and baseline hosts. For sipscan repliers, we differentiate between IRC and P2P C&C alerts, whereas for the baseline we include both types of alerts. After the sipscan, we see a sharp increase in the IRC C&C alerts, which indicates that hosts are attempting to contact known malicious IRC servers operating as controllers. This behavior continues for approximately two months, during which we see daily on average 2.4 C&C alerts per sipscan replier. However, on April 11 (day 111) there is a clear shift in the pattern of triggered signatures: the volume of IRC alerts suddenly drops, while the volume of P2P alerts rises. This signifies a likely upgrade in the mode of C&C communication

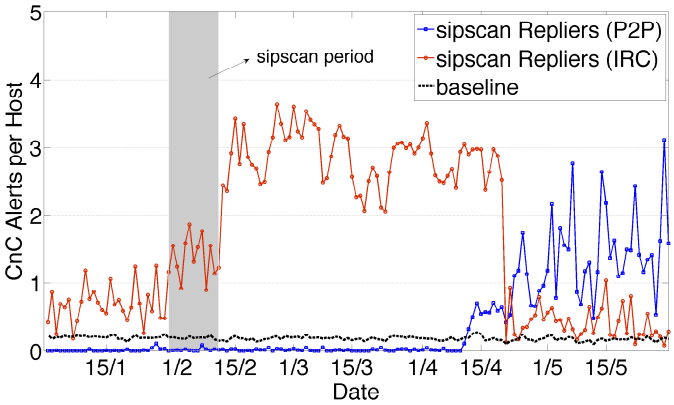


Figure 5.8: *Daily number of outbound C&C alerts per host for sipscan repliers and for baseline hosts over a period of 5 months. We show IRC and P2P C&C alerts in different lines.*

of the Sality botnet.

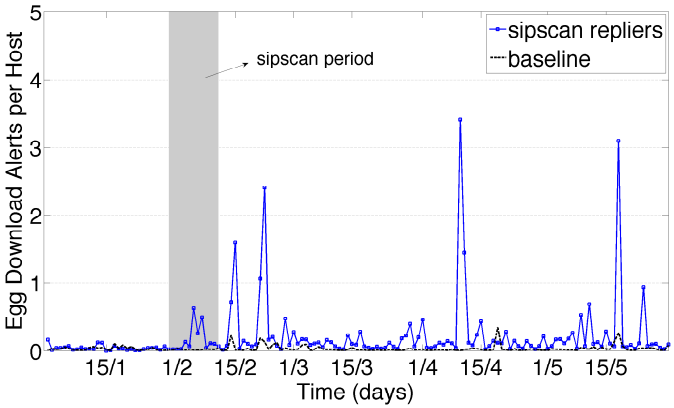


Figure 5.9: *Daily number of outbound Egg Download alerts per host for sipscan repliers and for baseline hosts over a period of 5 months.*

Figure 5.9 illustrates the daily number of *Egg Download* alerts per

sipscan replier and baseline host. After the sipscan, we observe 4 malware downloading spikes, during which the daily alert count ranges from 1.6 to 3.4 per sipscan replier. The spike that occurs on April 11 (day 111), seems to be associated with the shift in the communication method used to contact the controller shown in Figure 5.8. We believe that during that event the Sality botnet pushed a major update to the infected population, upgrading itself from the centralized HTTP to the fully decentralized P2P version.

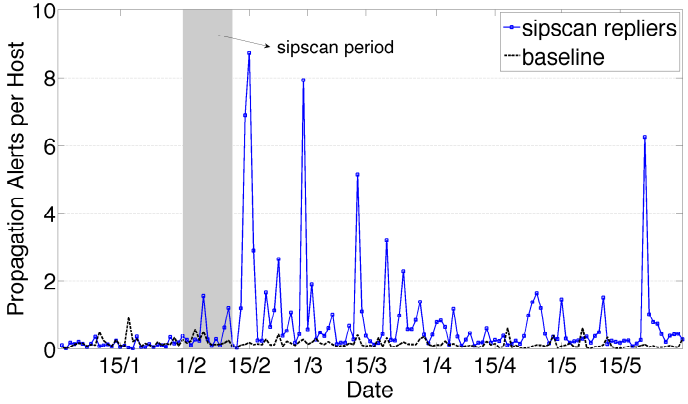


Figure 5.10: *Daily number of outbound Propagation alerts per host for sipscan repliers and for baseline hosts over a period of 5 months.*

In Figure 5.10, we show the daily number of *Propagation* alerts per local host for sipscan repliers and baseline hosts. We see that after the sipscan the number of outbound exploitation attempts originating from the sipscan repliers increases drastically, exhibiting an average daily value of 1.2 alerts per host compared to only 0.21 alerts per baseline host. The most dominant alerts of this group are the privilege escalation attempts with SIDs *(4060,2006546,2002383,2010643)* accounting for 72% of the observed activity.

Finally, Figure 5.11 illustrates the daily number of information leakage alerts per local host for sipscan repliers and baseline hosts. Again we see a sharp increase in the number of exfiltration alerts for sipscan repliers in the post-sipscan period, where the daily average increases from 4.7 to 18.2 alerts per host. The triggered alerts are dominated

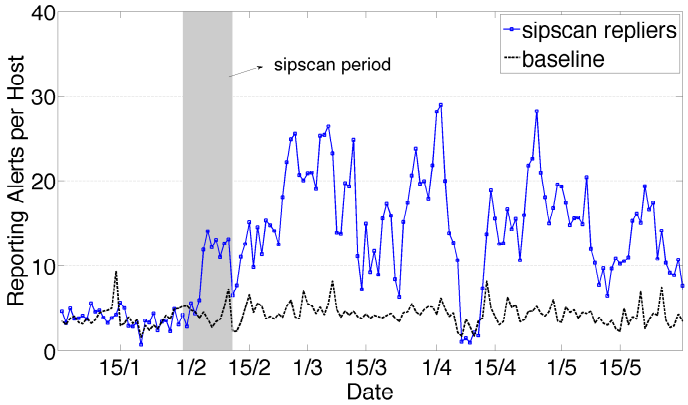


Figure 5.11: *Daily number of outbound Exfiltration alerts per host for sipscan repliers and for baseline hosts over a period of 5 months.*

by the signature *ET CHAT Facebook Chat POST Outbound*, which accounts for 83% of all alerts. However, this signature is also triggered by legitimate user activity and may introduce a significant number of false positives. This is reflected in the high baseline and in the pre-sipscan period, when it accounts on average for 4.7 alerts per host. Although the baseline for this alert group is high, we can still see a clear increase in the post-sipscan period when its alert volume quadruples.

5.4.3 Sality-bot infections

In Section 5.4.2 we discovered major changes in the alert patterns of sipscan repliers that correlate with the behavior of the Sality bot. In this section, we build a heuristic to identify this behavioral shift and extract likely Sality infections. Our goal is not to build a general purpose detector, but rather a systematic way to identify infected sipscan repliers in the monitored network. We use our heuristic to conservatively estimate a lower bound on the success rate of the sipscan in terms of infected hosts.

Our heuristic is summarized in Algorithm 2. We focus on sipscan repliers that were subsequently attacked. Then we find repliers that

Algorithm 2 Pseudo-code for identifying Sality-bot infections

Input:
B_T^S : mean count of S type alerts generated by Baseline hosts on day T
I_T^S : IQR of S type alerts generated by Baseline hosts on day T
R_T^S : mean count of S type alerts generated by sipscan repliers on day T
S={CnC Communication, Reporting, Propagation, Egg Download}
Output: Returns *true* if the examined host is infected, *false* otherwise

```
foreach alert type S do
    BelowThreshCount = 0;
    for T_i = 1:T_max do
        if isHostActiveAt(T_i) eq false then next;
        SignificanceThresh = B^S_{T_i} + 1.5 * I^S_{T_i};
        if T_i ≤ T_scan then
            if R^S_{T_i} > SignificanceThresh then
                return false;
            end
        else
            if R^S_{T_i} ≤ SignificanceThresh then
                BelowThreshCount += 1;
            end
        end
        if BelowThreshCount/(T_max − T_scan) > 0.05 then
            return false;
        end
    end
end
return true;
```

exhibit a persistent increase in outbound exploitation activity for the four signature classes listed in Table 5.2, while their respective activity in the pre-sipscan period is low. In particular, for the four classes in Table 5.2, we first compute the number of alerts per day each internal host generates. Our heuristic then finds and keeps hosts that trigger in the pre-sipscan period fewer alerts per day than the corresponding baseline of that day plus a tolerance of 1.5× the inter-quartile range of the baseline. If a host has more alerts per day even for a single day, then it is discarded from further consideration because it is either already

infected or it generates a large number of false positives. Second, our heuristic makes the same comparison in the post-sipscan period. If the daily alert count is consistently above the tolerance threshold, then it constitutes an indication of compromise activity. To assess whether this increase persists, we count the number of daily bins where it is above the threshold and tolerate only 5% of the post-sipscan bins where this condition is not met. We consider only the bins in which a host has generated at least one alert of any type.

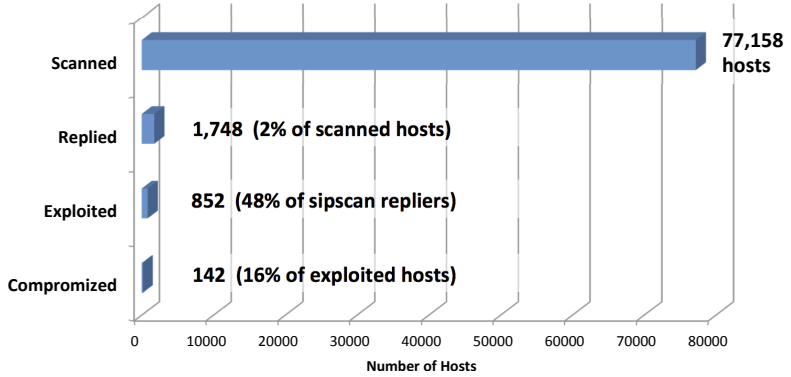


Figure 5.12: *Sality sipscan ETH turnover*

Our heuristic takes a conservative approach by introducing several conditions to make a Sality infection assessment. It is possible, however, that a Sality bot exhibits some of the post-sipscan behaviors illustrated in Figure 5.7 but not all. For example, some examined hosts show persistent signs of C&C communication and attempts to propagate, but do not attempt to leak data. Others attempt to exfiltrate data, but do not frequently visit malicious domains to fetch malware. By tailoring our heuristic to only make an assessment if all alert types in the post-sipscan period exhibit a persistent increase, we attempt to minimize possible false positives even if we introduce a number of false negatives. This way, we believe we can estimate a lower bound of the Sality infections that occurred in our infrastructure.

Our heuristic identified a total of 142 Sality infections in our IDS dataset. Figure 5.12 summarizes the estimated success rate of the Sality botnet in the monitored network. In the first stage of reconnaissance,

77,158 exposed ETH Zurich IPs were scanned. Out of these only 1,748 (2%) hosts replied to the scanners. Almost half of the sipscan repliers, specifically 48%, were subsequently the targets of inbound exploitation attacks. Based on our heuristic we identified that 142 hosts showed persistent signs of infection during the post-sipscan period. Therefore, the sipscan turnover, i.e. the percentage of hosts that were infected out of the sipscan repliers, was 8%.

5.5 Related Work

A long line of measurement studies has analyzed botnets over the last years, following their evolution from centralized IRC-based [7,8] to fully decentralized C&C architectures [9, 10]. The goal of these efforts has been to characterize botnet activities [11], analyze C&C communication methods [7], and estimate the respective botnet size and geographical properties [12]. Their observations have been used to fine tune network defences [13] and tailor novel detection mechanisms [14–16].

One of the most integral aspects of botnet activity is scanning. Since scanning is widespread [3] and regularly captured by monitoring infrastructures [8, 17], it is imperative for security analysts to have a measure regarding its severity and impact on the victim population. However, few studies have focused on the probing characteristics of botnets. In [18] Paxson *et al.* analyzed traffic captured at honeynets in order to study the statistical properties of 22 large-scale scanning events. In a followup study, Li *et al.* [19, 20] extracted botnet scan traffic from honeynet data and used it to infer general properties of botnets, such as population characteristics, blacklisting effectiveness, dynamics of new bot arrivals and scanning strategies. Finally, Yegneswaran *et al.* [8] analyzed the source code of a widely-used botnet malware, revealing the scanning capabilities of basic IRC bots.

Most related to our work, Dainotti *et al* [5] discovered an interesting stealthy scan of the entire IPv4 address space that was carried out by the Sality botnet and analyzed the different phases of the event. However, this study was based solely on packet traces collected at the UCSD network telescope and does not provide insights regarding the effectiveness of scanning and its followup activity. In our work, we detect the sipscan in a large ISP with live hosts, identify the set of hosts that replied to scanners, and analyze the targeted exploitation activity

that followed. This way we provide new insights about the escalation of this event and the effectiveness of scanning.

5.6 Conclusions

In this work, we provide new insights about Internet scanning, focusing on an interesting Internet-wide scanning event orchestrated by a large botnet. Using a unique dataset of both unsampled Netflow records and IDS alerts collected from a large academic network, we assess the effectiveness of scanning in terms of targeted hosts that replied and hosts that were eventually compromised. We find that 2% of the scanned hosts replied to the scanners and at least 8% of the repliers were eventually compromised by subsequent exploitation attempts. In addition, our work provides new insights about the "/0" sipscan orchestrated from the Sality botnet [5]. We find the sipscan was only a pre-cursor of subsequent exploitation attacks towards sipscan repliers. The attack escalated, leading to at least 142 infected hosts in the monitored network. Furthermore, we observe a segregation of roles between scanners and exploiters, which originate from different geographical locations. Finally, we observe that the sipscan originated from 352,350 additional IP addresses. Our study provides a novel view into Internet scanning from the side of scan repliers.

5.7 Acknowledgements

The authors wish to thank Prof. Bernhard Plattner for his invaluable help. Furthermore, we would also like to thank Stephan Sheridan and Christian Hallqvist at ETH for their help in the collection and archival of the IDS data used in this paper. This work was supported by the ZISC Secmetrics project.

Bibliography

[1] S. Shin, R. Lin, and G. Gu, "Cross-analysis of botnet victims: New insights and implications." in *RAID*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier,

Eds., vol. 6961. Springer, 2011, pp. 242–261. [Online]. Available: http://dblp.uni-trier.de/db/conf/raid/raid2011.html#ShinLG11

[2] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses."

[3] E. Glatz and X. Dimitropoulos, "Classifying internet one-way traffic," in *Proc. of the 2012 ACM conf. on Internet measurement*. NY, USA: ACM, 2012.

[4] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proceedings of the 22nd USENIX Security Symposium*, Aug. 2013.

[5] A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescap, "Analysis of a "/0" Stealth Scan from a Botnet," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, Nov 2012, pp. 1–14.

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," U.S., 2002.

[7] E. Cooke, F. Jahanian, and D. Mcpherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," 2005, pp. 39–44.

[8] P. Barford and V. Yegneswaran, "An inside look at botnets." in *Malware Detection*, ser. Advances in Information Security, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. Springer, 2007, vol. 27, pp. 171–191.

[9] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets," in *IEEE Symposium on Security and Privacy*, 2013, pp. 97–111.

[10] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proceedings of the 1st Usenix LEET Workshop*, Berkeley, CA, USA, 2008, pp. 9:1–9:9.

[11] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06, NY, USA, 2006.

[12] B. Stone-gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover."

[13] F. C. Freiling, T. Holz, and G. Wicherski, *Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks*, 2005.

[14] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale Botnet Detection and Characterization."

[15] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol and Structure-Independent Botnet Detection," in *USENIX Security Symposium*, 2008, pp. 139–154.

[16] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," in *NSDI*, 2008.

[17] P. Bacher, T. Holz, M. Kotter, and G. Wicherski, "Know your enemy: Tracking botnets," http://www.honeynet.org/papers/bots, 2008.

[18] V. Yegneswaran, P. Barford, and V. Paxson, "Using honeynets for internet situational awareness," in *Fourth Workshop on Hot Topics in Networks (HotNets IV)*, 2005.

[19] Z. Li, A. Goyal, and Y. Chen, "Honeynet-based botnet scan traffic analysis." in *Botnet Detection*, ser. Advances in Information Security, 2008, vol. 36, pp. 25–44.

[20] Z. Li, A. Goyal, Y. Chen, and V. Paxson, "Towards situational awareness of large-scale botnet probing events." *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 175–188, 2011.

[21] SWITCH, "Swiss Tele Communication System for Higher Education," http://www.switch.ch/.

[22] "Anonymous postmasters early warning system," http://www.apews.org, [Online; accessed 13-03-2014].

[23] "Shadowserver foundation," https://www.shadowserver.org/, 2014, [Online; accessed 13-03-2014].

[24] "Dshield: Internet storm center," http://www.dshield.org/, 2014, [Online; accessed 13-03-2014].

[25] "Threatexpert - automated threat analysis," http://www.threatexpert.com/, 2014, [Online; accessed 13-03-2014].

[26] M. Lite, "GeoIP Database," http://dev.maxmind.com/geoip/legacy/geolite/, 2014, [Online; accessed 03-03-2014].

[27] Falliere, "Sality: Story of a peer-to-peer viral network," 2011.

[28] N. Falliere, "A distributed cracker for voip," http://www.symantec.com/connect/blogs/distributed-cracker-voip, 2011.

[29] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: The commoditization of malware distribution," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11, Berkeley, CA, USA, 2011, pp. 13–13.

[30] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," in *In The 8th ACM International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 366–375.

[31] C. Kruegel and W. Robertson, "Alert verification - determining the success of intrusion attempts," in *Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004*, 2004, pp. 1–14.

[32] 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*. ACM, 2011, pp. 29–44.

[33] E. Raftopoulos and X. Dimitropoulos, "A quality metric for ids signatures: In the wild the size matters," *EURASIP Journal on Information Security*, Dec 2013.

Chapter 6

Conclusions

The internet has become a central pillar of today's economy and needs to be protected by modern malware. Security practitioners traditionally use multiple layers of defensive mechanisms, consisting of firewalls, antivirus scanners, access control policies, intrusion detection and prevention systems in order to shield the premises of the enterprise network. In this fortress concept the security of the organization is considered as good as the effectiveness of its individual layers.

However, relying on strengthening perimeter defenses in order to safeguard the monitored network from sophisticated professionalized threats has been proven to be ineffective. Commonly employed network security solutions are getting routinely bypassed by malware and external attackers. A combination of technical, financial, and organizational forces contribute in rendering perimeter security futile.

First, there are technical challenges in building secure networking equipment and applications. Vendors are under pressure to release new products and versions, but do not have the same incentives when it comes to properly testing these solutions for security flaws. Moreover, all these insecure components generate a stream of patches that need to be applied in a regular basis. Each patch can potentially disrupt the smooth operation of the running services, and thus it has to be tested extensively before being rolled out. A mid-size operational network should be expected to have hundreds of known exploitable vulnerabilities at any given time [1, 2].

Second, there are several financial aspects that affect the significance assigned to security at the organizational level. Security has no return on investment in the classic sense, since there is no quantifiable proof that can be used in order to support the budget needed to build a secure network. The resources required to identify all the possible threats an infrastructure is exposed to, tailor effective countermeasures, roll-out the necessary patches, and harden critical components is often prohibitively high. Therefore, security specialists have to often make a compromise accepting a level of risk given the financial constraints.

Third, enterprise networks today are inherently unbounded. There is no strict concept of a perimeter, since the arrival of smartphones, cloud based servers, and mobile services has blurred the lines of enterprise network boundaries [4]. Mobilization has made it extremely difficult to define a perimeter, since users are allowed to connect to the network from the location and the device of their choosing.

Therefore, the cynical assertion that has to be made is that, in the presence of sophisticated malware, a breach of the deployed network defenses should be considered a certainty. Does this reality render perimeter security obsolete? In this work, we stress the fact that data collected at the network level can be extremely useful in performing egress traffic monitoring, instead of the classic ingress analysis. Perimeter security can provide invaluable information in the context of extrusion detection, i.e. the identification of compromised hosts within the internal network. This work focuses on tailoring new tools to facilitate extrusion detection, expediting and automating the forensic analysis processes, and characterizing a large number of active infections in the wild.

In Section 2 of this work, we presented a novel approach to identify active infections using IDS logs. We tailored our detector EDGe, based on the observation that alerts with high mutual information are very likely to be correlated. Correlated alerts of specific types reveal that an actual infection has occurred. By applying EDGe to a large dataset of collected alerts, we found infections for a population of more than 81 thousand unique hosts. We performed an extensive characterization study of the 4,358 identified infections. Our study suggested that infections exhibit high spatial correlations, and that the existing infections open a wide attack vector for inbound attacks. Moreover, we compared the alerts produced by different malware families and highlighted key differences in the volume, aliveness, fanout, and severity of the alerts.

In Section 3, we conducted a complex experiment: we systematically monitored the decisions of a security analyst during the diagnosis of 200 incidents over a period of four weeks. Towards improving the ad-hoc manual and time-consuming investigation process we showed that a large part of the decisions of the analyst of our experiments can be encoded into a decision tree, which can be derived in an automated fashion from training data. The decision tree can help as a decision support tool for future incident handling and provides reasonable accuracy, identifying infections in 72% of the cases, without sacrificing interpretability.

In Section 4, we analyzed the complementary utility of the different data sources and the characteristics of the IDS signatures that were associated with confirmed incidents. From our experiments we learned that a single source is typically not sufficient for manual security diagnosis and that multiple sources should be combined. Multiple sensors were needed in more than 70.5% of all cases. These results highlight the importance of a holistic approach to malware detection that incorporates multiple security vantage points. Moreover, we extracted and made available a list of 165 Snort signatures that were triggered on hosts with validated malware. We compared the characteristics of good and regular signatures and reported a number of interesting statistics that provide essential guidelines for configuring Snort and for teaching good signature writing practices. In addition, we introduced a novel signature quality metric that can be exploited by security specialists to evaluate the available rulesets, prioritize the generated alerts, and facilitate the forensics analysis processes.

In Section 5, we studied a large scale security incident that targeted our infrastructure and investigated its evolution. Specifically, we detected a well orchestrated scan originating from the Sality botnet, that probed 96% of the active IPs in our infrastructure. We then identified hosts that replied to the scan and investigated the follow-up activity, revealing persistent inbound attacks attempting to compromise the responding hosts. We estimated that the success rate of the scanning attack in our network network was 2% in terms of hosts that replied to the scanners. In addition, we conservatively estimated that in the exploitation activity that followed 8% of the hosts that replied were eventually compromised. Finally, we provided new insights about the volume, scale and stealthiness of the scan.

6.1 Critical Assessment

The extrusion detection methods designed and evaluated in this thesis are by nature dependent on the security sensors used to generate the raw monitoring data. First and foremost, we rely heavily on the Snort IDS. Since 1998 when the first windows version of Snort was released, it has become the de-facto standard for IDS and in recent years also IPS. Although numerous commercial IDSs are available in the market, including Cisco Net Ranger, IBM ISSS, Symantec Intruder Alert, Network ICE, Cybersafe Centrax, Snort remains the most popular IDS solution among security practitioners for a number of reasons. Snort is free to use under the GPLv2 licence, is extremely versatile in terms of traffic analysis and attack detection, is easy to configure and deploy, has a large community support contributing new rulesets on a daily basis, and provides plenty of administrative front-ends. Snort is an IDS with a long product life, it seems to enjoy a prolonged maturity and exhibits no signs of going away.

In this work we exploit the IDS trace produced by a Snort sensor to identify the alert footprint generated by different malware infections. We leverage the collected alerts in two ways. First, we identify signatures generated in different stages of the malware's lifetime, corresponding to different actions undertaken by the malware in order to communicate, propagate, and update. Second, we manually analyze a set of signatures, and assign them to different classes based on the malware family and variant that is responsible for triggering them. If a different IDS system was to be used instead of Snort, with entirely different signature rulesets, then these parts of the methodology would have to be adapted to the new alert trace. Assigning signatures to the stages of the malware lifetime is rather straight forward, since usually signatures implicitly contain this information. In our work, as shown in Section 2, we relied on the *classtype* field and the *ruleset* type to perform this mapping. The classification of signatures to malware families and specific variants is more demanding in terms of manual-intensive work. Again we can leverage specific fields, such as the signature description, to get hints about the respective malware that is more likely to generate the respective alert. However, based on our experience this classification is mostly based on the security analyst's experience in investigating real life infections in an operational network.

Moreover, we utilize the output of several network monitoring tools,

presented in Sections 3 and 4, in order to strengthen our assessment regarding a suspected malware infection. The line of reconnaissance, service enumeration, and vulnerability scanners we use, such as NMap, Hping, and OpenVAS, are standard cross-platform tools available freely under the GPL licence. We believe that these are essential tools for network monitoring which are already part of the arsenal of most security practitioners. However, if a different set of tools was to be used providing information of the same nature, then we consider that our findings regarding the complementary utility of the security sources would still hold and could help analysts guide and expedite the forensics analysis processes. Additionally, the output of these monitoring sensors combined with two profiling methods, presented in Section 3, are used as input by our decision support tool. In order to deploy this tool in a different environment and use it to perform automated diagnosis, one would first have to systematically collect available blacklists in order to build a comprehensive database of malicious hosts in the wild. Second, and most importantly, our Google-based profiling process, which based on our study provided useful forensics evidence for 54.5% of the investigated incidents, would have to be revised. Collecting these two types of profiling data does not bear any challenges from an engineering perspective.

Finally, in Sections 2 and 5 of this thesis we perform a comprehensive characterization of modern malware in the wild. We perform all our measurements in the university campus network of ETH Zurich. The monitored network is both large and rich in terms of diversity in types of active hosts and deployed services. The relatively loose security policies in terms of restrictions enforced to the users regarding the applications they are allowed to install and use suggest that the infrastructure is exposed to a wide range of security threats. Therefore, we can safely assume that the malware samples we detect and analyze are representative of the types of threats a mid-size enterprise operational network with loose security policies is exposed to.

However, a very valid criticism is that malware constantly evolve and that the type of network footprint we detect today might not be valid for future malware. This argument is indeed true. The first generation of high profile malware were the disruptive worms that emerged in 2001-2005 like Code Red, Nimda, Blaster and Sasser. Their identification at the network level was based on simple anomaly detection

schemes that would attempt to identify extensive activity on a specific port or IP-range, triggered by the worm's automated propagation mechanism. In the years 2006-2012 we saw the rise of large botnets like SDbot, Zeus, and Sality and the emergence of highly sophisticated and professionalized malware like Torpig, Conficker, and FakeAV. Malcode writers shifted their attention in building a persistent platform that could sustain a systematic and dynamic attack, instead of building malware that would simply perform a predefined set of actions, such as propagate and send spam. More importantly this platform could be used to perform numerous illicit actions on the victim hosts and was paramount in monetizing the infected population.

The last generation of malware are the Advance Persistent Threats (APTs) [3] that emerged in late 2011. These are sophisticated and usually professionally developed malware incorporating cutting-edge technology. APTs are not carried out only by the traditional hacker, but rather are employed by governments, agencies and large corporations to target a specific infrastructure and perform a stealthy, long-term, sustained infiltration. Our work, focusing on detecting the multi-stage footprint of malware at the network level, targets the second group of sophisticated malware which are without doubt the most dominant threat in today's Internet. While worms have a very simple network signature and can be detected using basic anomaly detection techniques, APTs are extremely stealthy so we don't expect to see their activity generating any type of detectable trace, rendering our approach inadequate. However, analysis of APTs has shown that during their lifetime they also undergo multiple stages of malicious activity in order to infiltrate the victim infrastructure, escalate privileges, exfiltrate data, persist and update. So, if the available baseline defenses, such as IDS systems, succeed in introducing effective signatures capable of generating partial evidence of APT activity, then our work can be extremely useful in combining this evidence and performing a high confidence assessment.

6.2 Future Work

This work is a first systematic approach in studying security monitoring based on extrusion detection. There are many directions for future work including the analysis of different IDS systems and monitoring

tools that were not considered in this thesis, the generalization of our results by performing our analysis in different network types, and the extension of the detection and validation methods to take into account additional malware variants. However, we consider that the most interesting and beneficial next steps would be to provide the ability to fine-tune and parameterize EDGe, extend the decision support tool to enable reporting of intermediate assessments regarding investigated incidents, and develop a visualization dashboard illustrating the available security sources and the respective correlated meta-information to the analyst.

6.2.1 EDGe Tuning and Sensitivity Analysis

In Section 2.3.3 where we introduce EDGe, we state that a basic design goal of our detector is the reduction of false positives at the expense of introducing a reasonable amount of false negatives. This design principle is motivated by our observation that IDS sensors inherently suffer from a excessive number of false positives. However, this design goal might not be desirable in the case of a network with strict security policies, such as in the case of a financial or government institution, where false negatives are a prime concern since the might correspond to an undetected breach or an exfiltration of sensitive data. In this case EDGe should allow the user to set the sensitivity of the detector based on his own requirements. To incorporate such a functionality one should study the degree to which an input parameter, in this case the *J-Measure* threshold, affects the detector's output, i.e. the amount of reported infections. Such sensitivity analysis, would reveal what is the impact of the *J-Measure* threshold on the tradeoff between the false positives and false negatives generated by the detector.

Another, important parameter is the time interval over which we run the detector. Since EDGe is based on the detection of recurring actions manifested by the suspected infected hosts, it will perform better if alert traces corresponding to sufficiently large time intervals are used. However, in several scenarios it might be desirable to use EDGe using a few hours of alert trace for a "real-time" inference. Therefore, performing a sensitivity analysis regarding the impact of the duration of the alert trace on the accuracy of EDGe is also critical in order to assess whether EDGe can be used in an online mode, instead of the fully offline approach we used in our experiments.

6.2.2 Security Monitoring Visualization

An important aspect of security monitoring is the ability to visualize the collected data. Data visualization provides insights in sparse and complex datasets by distilling key aspects of the analyzed event in an intuitive and interpretable way. In our work we deal with large, heterogeneous datasets, collected from diverse security sensors. The inferences made by EDGe and our decision support tool can help the human analyst prioritize the analyzed incidents and focus on the forensics investigation of hosts that exhibit a high likelihood of infection. However, we can only expedite and facilitate and not completely eliminate the manual investigation process. Building tools that would allow the visualization of the traces can help not only to disambiguate suspected infections, but also to perform root cause analysis by highlighting the key pieces of evidence supporting a reported inference.

6.2.3 Security Monitoring Dashboard

Data collection, preprocessing, and archiving scripts combined with the software modules developed to perform the EDGe inference, implement the decision support tool, and carry out the forensics investigation account for a significantly large code base. These software modules are written in different languages including Perl, Python, Bash, and C++, utilize a wide range of independent system libraries, and operate on custom formats used to represent and store intermediate data traces. Deploying and running our system can be quite challenging for the unseasoned operator. Therefore, building a dashboard that can operate as a control panel, hiding all the low-level implementation details from the security analyst, would be highly beneficial. Such a front-end should allow the user to easily run the analysis on traces corresponding to a selected time-span, retrieve raw data for a specific event if manual investigation is required, and fine-tune the detection modules based on custom requirements.

6.2.4 Security Assurance Quantification

Currently, EDGe and our decision support tool produce at any given point a boolean inference, i.e. an inspected host is either benign or infected based on its generated activity trace. However, it might be

the case that although a specific host has been compromised it has only generated few signs of infection which are not sufficient to make a positive assessment. This is a common scenario encountered in our study which can be attributed to multiple reasons, such as short infection durations, stealthy malware behavior, and ineffective detection signatures on the security sensors. However, it might be valuable for a security administrator, especially in the case of critical components of the infrastructure such as DNS, Web and Mail servers, to have an early warning about the probability of compromise of these systems based on partial evidence. Our tools could be augmented, to provide not only an inference about the existence or absence of an infection but rather a score about the (in)security state of the inspected hosts. In this way we can build security metrics that capture the current state of security hygiene of the entire monitored infrastructure. Such high-level security quantification metrics are highly desirable since they encode a complex state of the system into a single value that can be easily communicated to non-specialists.

Bibliography

[1] Stefan Frei and Dominik Schatzmann and Bernhard Plattner and Brian Trammell. Modelling the Security Ecosystem- The Dynamics of (In)Security. WEIS, 2009.

[2] Neuhaus, Stephan and Zimmermann, Thomas and Holler, Christian and Zeller, Andreas. Predicting Vulnerable Software Components. Proceedings of the 14th ACM Conference on Computer and Communications Security, New York, USA, 2007.

[3] Li, Frankie and Lai, Anthony and Ddl, Ddl Evidence of Advanced Persistent Threat: A Case Study of Malware for Political Espionage. Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software, Washington, DC, USA.

[4] Chen, Y. and Paxson, V. and Katz, R. What's New About Cloud Computing Security. University of California, Berkeley Report No. UCB/EECS-2010-5

Appendix A

Full List of Publications

During this thesis, the following journal, conference and workshop papers were authored.

P1 Elias Raftopoulos, and Xenofontas Dimitropoulos. **Detecting, Validating and Characterizing Computer Infections in the Wild**, in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (IMC 2011). Berlin, Germany, November 2011.

P2 Elias Raftopoulos, Matthias Egli, and Xenofontas Dimitropoulos. **Shedding Light on Log Correlation in Network Forensics Analysis**, in *Proceedings of the 9th international conference on Detection of intrusions and malware, and vulnerability assessment* (DIMVA 2012). Crete, Greece, July 2012.

P3 Elias Raftopoulos, and Xenofontas Dimitropoulos. **Understanding Network Forensics Analysis in an Operational Environment**, in *IEEE Symposium on Security and Privacy International Workshop on Cyber Crime* (IWCC 2013). San Francisco, CA, USA, May 2013. **Best Paper Award.**

P4 Elias Raftopoulos, and Xenofontas Dimitropoulos. **A Quality Metric for IDS Signatures: In the Wild the Size Matters**, in *Journal on Information Security, Cyber Crime: New Trends, Challenges, and Detection* (EURASIP 2013).

P5 Elias Raftopoulos, and Xenofontas Dimitropoulos. **Tracking Malware in the Wild with EDGe**, in *IEEE Journal on Selected Areas in Communications, Special Issue on Deep Packet Inspection: Algorithms, Hardware, and Applications* (JSAC 2014).

P6 Elias Raftopoulos, Eduard Glatz, Alberto Dainotti, and Xenofontas Dimitropoulos. **The Day After: Studying the Aftermath of a "/0" scan from the Sality botnet**, in *International Workshop on Traffic Monitoring and Analysis* (TMA 2015).

Acknowledgments

First and foremost, I thank Prof. Bernhard Plattner for enabling this thesis and for his support throughout the project. By providing the freedom of exploring own ideas, he creates an environment fostering the creativity of his PhD students. I am particularly grateful for the support and guidance from my co-advisor, Prof. Xenofontas Dimitropoulos. Having such an exceptionally capable teacher was a great experience and made the work on this project very enjoyable, even during stressful times. I would also like to thank Dr. Vincent Lenders (ArmaSuisse) for his invaluable help in formulating the research topic of this thesis. I also want to thank Prof. Markatos and Dr. Kreibich for being co-examiners and providing valuable feedback on this dissertation.

Moreover, I would like to thank my colleagues at ETH Zurich, in particular my office mates Bernhard Distl, Ehud Ben-Porat, and Sascha Trifunovic and (in alphabetical order) Abdullah Alhussainy, Andreea Hossmann, Ariane Trammell, Bernhard Ager, Bernhard Tellenbach, Brian Trammell, Daniela Brauckhoff, David Gugelmann, Domenico Giustiniano, Dominik Schatzmann, Eduard Glatz, Franck Legendre, Karin Anna Hummel, Mahdi Asadpour, Markus Happe, Martin Burkhart, Merkourios Karaliopoulos, Matthias Egli, Panagiotis Georgopoulos, Panayotis Antoniadis, Stefan Frei, Simon Heimlicher, Stephan Neuhaus, Theus Hossmann, Thomas Dübendorfer, Thrasyvoulos Spyropoulos, Vasileios Kotronis, Wolfgang Mühlbauer, for countless interesting discussions and enjoyable hours in the coffee corner. I further thank Dr. Hans Barz for the fruitful teaching collaboration, and Beat Futterknecht, Caterina Sposato, and Thomas Steingruber, for providing frictionless administrative and technical support.

I would also like to thank Stephan Sheridan and Christian Hallqvist

at ETH for their help in the collection and archiving of the IDS data, which were of key importance for this work. Moreover, I would like to thank SWITCH for providing access to their NetFlow records and thank Bernhard Tellenbach, Brian Trammell, and Dominik Schatzmann for maintaining the local CSG NetFlow repository. I am also grateful to the Zurich Information and Privacy Center (ZISC) for the funding based on which this research was made possible.

Last but not least, I would like to thank my parents for their unconditional love and especially my father for being a source of inspiration throughout my life. I would also like to thank my partner for her immense support and all others that influenced my research and that were not explicitly mentioned here.