16th USENIX Security Symposium

Pp. 167–182 of the *Proceedings*

BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation

Guofei Gu[1], Phillip Porras[2], Vinod Yegneswaran[2], Martin Fong[2], Wenke Lee[1]

[1]College of Computing
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332

[2]Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

We present a new kind of network perimeter monitoring strategy, which focuses on recognizing the infection and coordination dialog that occurs during a successful malware infection. BotHunter is an application designed to track the two-way communication flows between internal assets and external entities, developing an evidence trail of data exchanges that match a state-based infection sequence model. BotHunter consists of a correlation engine that is driven by three malware-focused network packet sensors, each charged with detecting specific stages of the malware infection process, including inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog, and outbound attack propagation. The BotHunter correlator then ties together the dialog trail of inbound intrusion alarms with those outbound communication patterns that are highly indicative of successful local host infection. When a sequence of evidence is found to match BotHunter's infection dialog model, a consolidated report is produced to capture all the relevant events and event sources that played a role during the infection process. We refer to this analytical strategy of matching the dialog flows between internal assets and the broader Internet as *dialog-based correlation*, and contrast this strategy to other intrusion detection and alert correlation methods. We present our experimental results using BotHunter in both virtual and live testing environments, and discuss our Internet release of the BotHunter prototype. BotHunter is made available both for operational use and to help stimulate research in understanding the life cycle of malware infections.

1 Introduction

Over the last decade, malicious software or *malware* has risen to become a primary source of most of the scanning [38], (distributed) denial-of-service (DOS) activities [28], and direct attacks [5], taking place across the Internet. Among the various forms of malicious software, *botnets* in particular have recently distinguished

themselves to be among the premier threats to computing assets [20]. Like the previous generations of computer viruses and worms, a bot is a *self-propagating* application that infects vulnerable hosts through direct exploitation or Trojan insertion. However, all bots distinguish themselves from the other malware forms by their ability to establish a command and control (C&C) channel through which they can be updated and directed. Once collectively under the control of a C&C server, bots form what is referred to as a *botnet*. Botnets are effectively a collection of slave computing and data assets to be sold or traded for a variety of illicit activities, including information and computing resource theft, SPAM production, hosting phishing attacks, or for mounting distributed denial-of-service (DDoS) attacks [12,34,20].

Network-based intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) may come to mind as the most appealing technology for detecting and mitigating botnet threats. Traditional IDSs, whether signature based [30,35] or anomaly based [46,8], typically focus on inbound packets flows for signs of malicious point-to-point intrusion attempts. Network IDSs have the capacity to detect initial incoming intrusion attempts, and the prolific frequency with which they produce such alarms in operational networks is well documented [36]. However, distinguishing a successful local host infection from the daily myriad of scans and intrusion attempts is as critical and challenging a task as any facet of network defense.

Intrusion report correlation enables an analyst to obtain higher-level interpretations of network sensor alert streams, thereby alleviating noise-level issues with traditional network IDSs. Indeed, there is significant research in the area of consolidating network security alarms into coherent incident pictures. One major vein of research in intrusion report correlation is that of alert fusion, *i.e.,* clustering similar events under a single label [42]. The primary goal of fusion is log reduction, and in most systems *similarity* is based upon either attributing multiple events to a single threat agent or providing a consolidated view of a common set of events that target a single victim. The bot infection problem satisfies neither criterion. The bot infection process spans several diverse transactions that occur in multiple directions and potentially involves several active participants. A more applicable area of alert correlation research is multistage attack recognition, in which predefined scenario templates capture multiple state transition sequences that may be initiated by multiple threat agents [40,29]. In Section 3 we discuss why predefined state transition models simply do not work well in bot infection monitoring. While we argue that bot infections do regularly follow a series of specific steps, we find it rare to accurately detect all steps, and find it equally difficult to predict the order and time-window in which these events are recorded.

Our Approach: We introduce an "evidence-trail" approach to recognizing successful bot infections through the communication sequences that occur during the infection process. We refer to this approach as the infection *dialog correlation* strategy. In dialog correlation, bot infections are modeled as a set of loosely ordered communication flows that are exchanged between an internal host and one or more external entities. Specifically, we model all bots as sharing a common set of underlying actions that occur during the infection life cycle: target scanning, infection exploit, binary egg download and execution, command and control channel establishment, and outbound scanning. We neither assume that all these events *are required* by all bots nor that every event *will be detected* by our sensor alert stream. Rather, our dialog correlation system collects an evidence trail of relevant infection events per internal host, looking for a threshold combination of sequences that will satisfy our requirements for bot declaration.

Our System: To demonstrate our methodology, we introduce a passive network monitoring system called `BotHunter`, which embodies our infection dialog correlation strategy. The BotHunter correlator is driven by Snort [35] with a customized malware-focused ruleset, which we further augment with two additional bot-specific anomaly-detection plug-ins for malware analysis: SLADE and SCADE. SLADE implements a lossy n-gram payload analysis of incoming traffic flows, targeting byte-distribution divergences in selected protocols that are indicative of common malware intrusions. SCADE performs several parallel and complementary malware-focused port scan analyses to both incoming and outgoing network traffic. The BotHunter correlator associates inbound scan and intrusion alarms with outbound communication patterns that are highly indicative of successful local host infection. When a sufficient sequence of alerts is found to match BotHunter's infection dialog model, a consolidated report is produced to capture all the relevant events and event participants that contributed to the infection dialog.

Contributions: Our primary contribution in this paper is to introduce a new network perimeter monitoring strategy, which focuses on detecting malware infections (specifically bots/botnets) through IDS-driven dialog correlation. We present an abstraction of the major network packet dialog sequences that occur during a successful bot infection, which we call our *bot infection dialog model.* Based on this model we introduce three bot-specific sensors, and our IDS-independent dialog correlation engine. Ours is the *first* real-time analysis system that can automatically derive a profile of the entire bot detection process, including the identification of the victim, the infection agent, the source of the egg download, and the command and control center.[1] We also present our analysis of BotHunter against more than 2,000 recent bot infection experiences, which we compiled by deploying BotHunter both within a high-interaction honeynet and through a VMware experimentation platform using recently captured bots. We validate our infection sequence model by demonstrating how our correlation engine successfully maps the network traces of a wide variety of recent bot infections into our model.

The remainder of this paper is outlined as follows. In Section 2 we discuss the sequences of communication exchanges that occur during a successful bot and worm infection. Section 3 presents our bot infection dialog

model, and defines the conditions that compose our detection requirements. Section 4 presents the BotHunter architecture, and Section 5 presents our experiments performed to assess BotHunter's detection performance. Section 6 discusses limitations and future work, and Section 7 presents related work. Section 8 discusses our Internet release of the BotHunter system, and in Section 9 we summarize our results.

2 Understanding Bot Infection Sequences

Understanding the full complexity of the bot infection life cycle is an important challenge for future network perimeter defenses. From the vantage point of the network egress position, distinguishing successful bot infections from the continual stream of background exploit attempts requires an analysis of the two-way dialog flow that occurs between a network's internal hosts and the Internet. On a well-administered network, the threat of a direct-connect exploit is limited by the extent to which gateway filtering is enabled. However, contemporary malware families are highly versatile in their ability to attack susceptible hosts through email attachments, infected P2P media, and drive-by-download infections. Furthermore, with the ubiquity of mobile laptops and virtual private networks (VPNs), direct infection of an internal asset need not necessarily take place across an administered perimeter router. Regardless of how malware enters a host, once established inside the network perimeter the challenge remains to identify the infected machine and remove it as quickly as possible.

For this present study, we focus on a rather narrow aspect of bot behavior. Our objective is to understand the sequence of network communications and data exchanges that occur between a victim host and other network entities. To illustrate the stages of a bot infection, we outline an infection trace from one example bot, a variant of the Phatbot (aka Gaobot) family [4]. Figure 1 presents a summary of communication exchanges that were observed during a local host Phatbot infection.

As with many common bots that propagate through remote exploit injection, Phatbot first (step 1) probes an address range in search of exploitable network services or responses from Trojan backdoors that may be used to enter and hijack the infected machine. If Phatbot receives a connection reply to one of the targeted ports on a host, it then launches an exploit or logs in to the host using a backdoor. In our experimental case, a Windows workstation replies to a 135-TCP (MS DCE/RPC) connection request, establishing a connection that leads to an immediate RPC buffer overflow (step 2). Once infected, the victim host is directed by an upload shell script to open a communication channel back to the attacker to download the full Phatbot binary (step 3). The bot inserts itself into the system boot process, turns off security software, probes the local network for additional NetBIOS shares, and secures the host from other malware that may be loaded on the machine. The infected victim next distinguishes itself as a bot by establishing a connection to a botnet C&C server, which in the case of Phatbot is established over an IRC channel (step 4). Finally, the newly infected bot establishes a listen port to accept new binary updates and begins scanning other external victims on behalf of the botnet (step 5).

3 Modeling the Infection Dialog Process

While Figure 1 presents an example of a specific bot, the events enumerated are highly representative of the life cycle phases that we encounter across the various bot families that we have analyzed. Our bot propagation model is primarily driven by an assessment of outward-bound communication flows that are indicative of behavior associated with botnet coordination. Where possible, we seek to associate such outbound communication patterns with observed inbound intrusion activity. However, this latter activity is not a requirement for bot declaration. Neither are incoming scan and exploit alarms sufficient to declare a successful malware infection, as we assume that a constant stream of scan and exploit signals will be observed from the egress monitor.

We model an infection sequence as a composition of participants and a loosely ordered sequence of exchanges: Infection $I = < A, V, C, V', E, \overline{D} >$, where A = Attacker, V = Victim, E = Egg Download Location, C = C&C Server, and V' = the Victim's next propagation target. $\overline{D}$ represents an infection dialog sequence composed of bidirectional flows that cross the egress boundary. Our infection dialog $\overline{D}$ is composed of a set of five potential dialog transactions (*E1, E2, E3, E4, E5*), some subset of which may be observed during an instance of a local host infection:

- E1: External to Internal Inbound Scan
- E2: External to Internal Inbound Exploit
- E3: Internal to External Binary Acquisition
- E4: Internal to External C&C Communication
- E5: Internal to External Outbound Infection Scanning

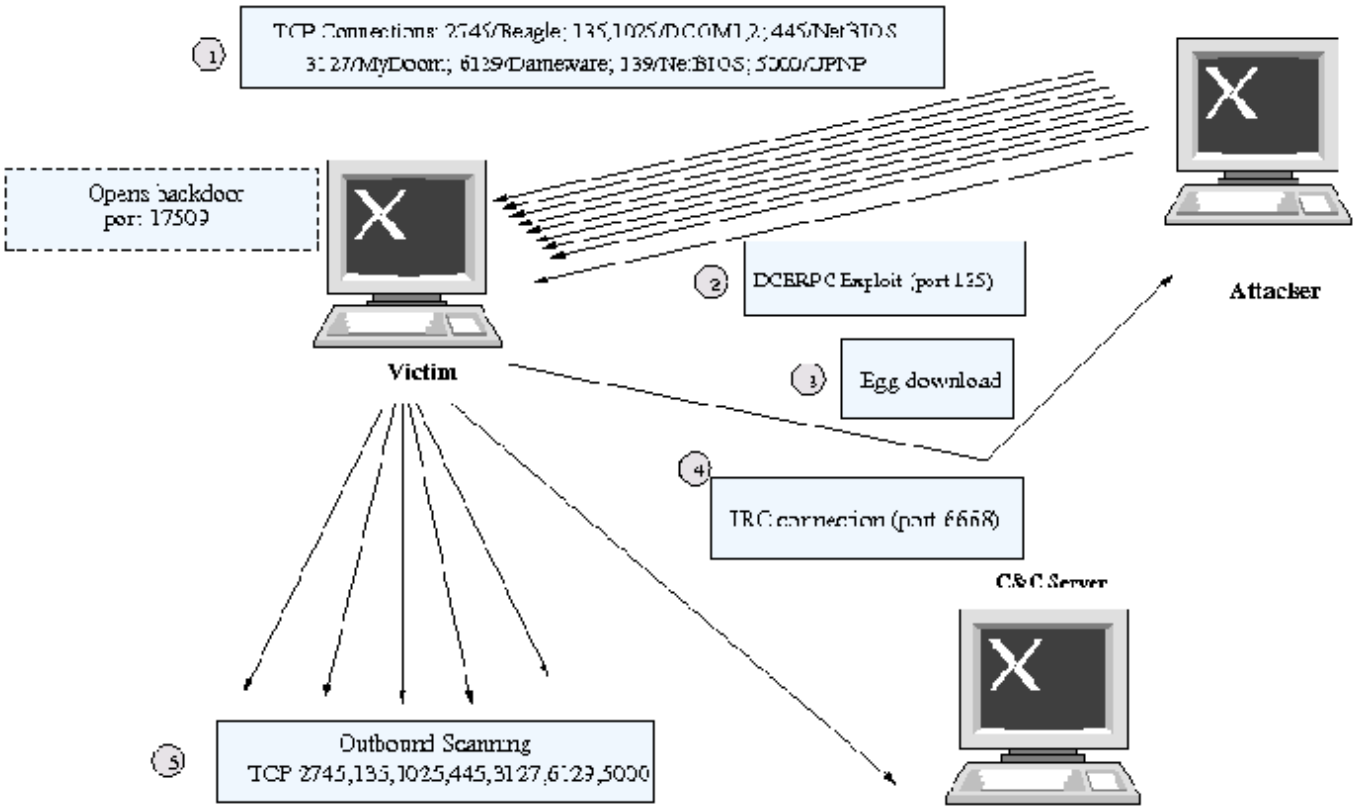

Figure 1: Phatbot Dialog Summary

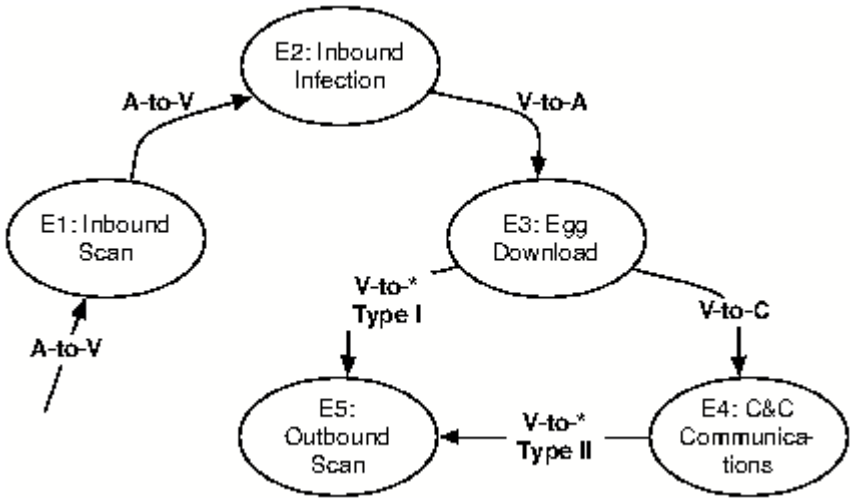

Figure 2: Bot Infection Dialog Model

Figure 2 illustrates our bot infection dialog model used for assessing bidirectional flows across the network boundary. Our dialog model is similar to the model presented by Rajab et al. in their analysis of 192 IRC bot instances [33]. However, the two models differ in ways that arise because of our specific perspective of egress boundary monitoring. For example, we incorporate early initial scanning, which is often a preceding observation that occurs usually in the form of IP sweeps that target a relatively small set of selected vulnerable ports. We also exclude DNS C&C lookups, which Rajab et al. [33] include as a consistent precursor to C&C coordination, because DNS lookups are often locally handled or made through a designated DNS server via internal packet exchanges that should not be assumed visible from the egress position. Further, we exclude local host modifications and internal network propagation because these are also events that are not assumed to be visible from the egress point. Finally, we include internal-to-external attack propagation, which Rajab et al. [33] exclude. While our model is currently targeted for passive network monitoring events, it will be straightforward to include localhost-based or DNS-server-based IDSs that can augment our dialog model. Figure 2 is not intended to provide a strict ordering of events, but rather to capture a typical infection dialog (exceptions to which we discuss below). In the idealized sequence of a direct-exploit bot infection dialog, the bot infection begins with an external-to-internal communication flow that may encompass bot scanning (E1) or a direct inbound exploit (E2). When an internal host has been successfully compromised (we observe that many compromise attempts regularly end with process dumps or system freezes), the newly compromised

host downloads and instantiates a full malicious binary instance of the bot (E3). Once the full binary instance of the bot is retrieved and executed, our model accommodates two potential dialog paths, which Rajab et al. [33] refer to as the bot Type I versus Type II split. Under Type II bots, the infected host proceeds to C&C server coordination (E4) before attempting self-propagation. Under a Type I bot, the infected host immediately moves to outbound scanning and attack propagation (E5), representing a classic worm infection.

We assume that bot dialog sequence analysis must be robust to the absence of some dialog events, must allow for multiple contributing candidates for each of the various dialog phases, and must not require strict sequencing on the order in which outbound dialog is conducted. Furthermore, in practice we have observed that for Type II infections, time delays between the initial infection events (E1 and E2) and subsequent outbound dialog events (E3, E4, and E5) can be significant-on the order of several hours. Furthermore, our model must be robust to failed E1 and E2 detections, possibly due to insufficient IDS fidelity or due to malware infections that occur through avenues other than direct remote exploit.

One approach to addressing the challenges of sequence order and event omission is to use a weighted event threshold system that captures the minimum necessary and sufficient sparse sequences of events under which bot profile declarations can be triggered. For example, one can define a weighting and threshold scheme for the appearance of each event such that a minimum set of event combinations is required before bot detection. In our case, we assert that bot infection declaration requires a minimum of

Condition 1: Evidence of local host infection (E2), AND evidence of outward bot coordination or attack propagation (E3-E5); or

Condition 2: At least two distinct signs of outward bot coordination or attack propagation (E3-E5).

In our description of the BotHunter correlation engine in Section 4, we discuss a weighted event threshold scheme that enforces the above minimum requirement for bot declaration.

4 BotHunter: System Design

We now turn our attention to the design of a passive monitoring system capable of recognizing the bidirectional warning signs of local host infections, and correlating this evidence against our dialog infection model. Our system, referred to as BotHunter, is composed of a trio of IDS components that monitor in- and out-bound traffic flows, coupled with our dialog correlation engine that produces consolidated pictures of successful bot infections. We envision BotHunter to be located at the boundary of a network, providing it a vantage point to observe the network communication flows that occur between the network's internal hosts and the Internet. Figure 3 illustrates the components within the BotHunter package.

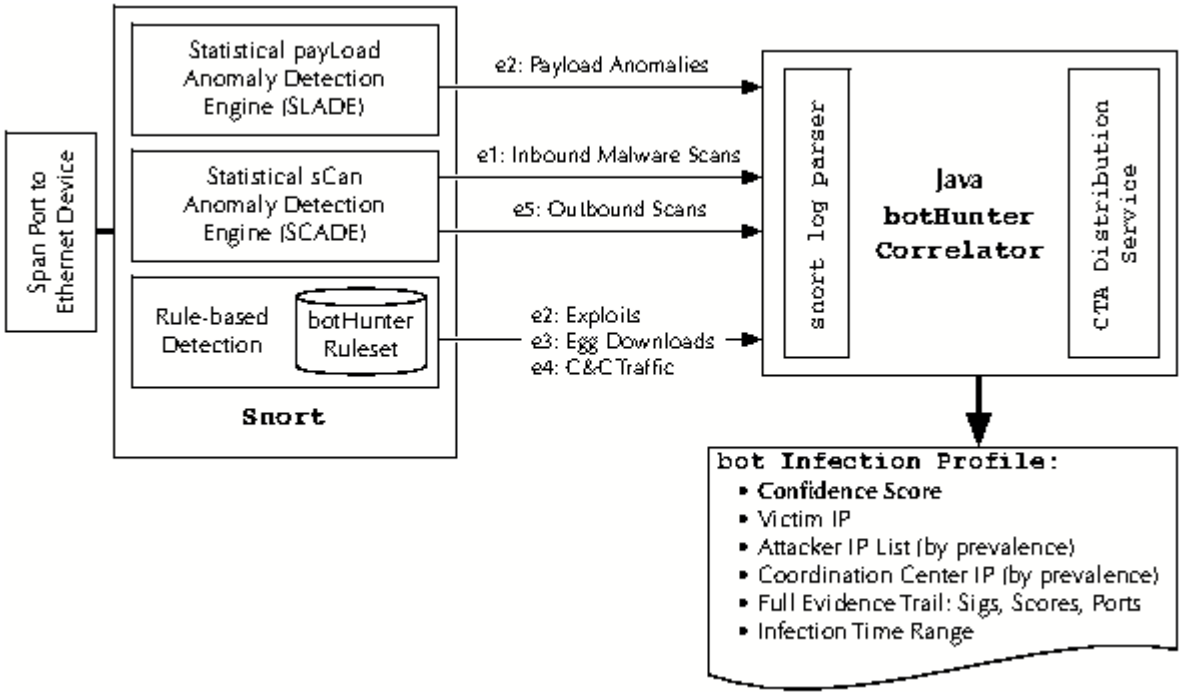

Figure 3: BotHunter System Architecture

Our IDS detection capabilities are composed on top of the open source release of Snort [35]. We take full advantage of Snort's signature engine, incorporating an extensive set of malware-specific signatures that we developed internally or compiled from the highly active Snort community (e.g., [10] among other sources). The signature engine enables us to produce dialog warnings for inbound exploit usage, egg downloading, and C&C patterns, as discussed in Section 4.1.3. In addition, we have developed two custom plugins that complement the Snort signature engine's ability to produce certain dialog warnings. Note that we refer to the various IDS alarms as *dialog warnings* because we do not intend the individual alerts to be processed by administrators in search of bot or worm activity. Rather, we use the alerts produced by our sensors as input to

drive a bot dialog correlation analysis, the results of which are intended to capture and report the actors and evidence trail of a complete bot infection sequence.
Our two custom BotHunter plugins are called SCADE and SLADE. SCADE, discussed in Section 4.1.1, provides inbound and outbound scan detection warnings that are weighted for sensitivity toward malware-specific scanning patterns. SLADE, discussed in Section 4.1.2, conducts a byte-distribution payload anomaly detection of inbound packets, providing a complementary non-signature approach in inbound exploit detection.
Our BotHunter correlator is charged with maintaining an assessment of all dialog exchanges, as seen through our sensor dialog warnings, between all local hosts communicating with external entities across the Internet. The BotHunter correlator manages the state of all dialog warnings produced per local host in a data structure we refer to as the *network dialog correlation matrix* (Figure 4). Evidence of local host infection is evaluated and expired from our correlator until a sufficient combination of dialog warnings (E1-E5) crosses a weighted threshold. When the bot infection threshold is crossed for a given host, we produce a bot infection profile (illustrated in Figure 7).
Finally, our correlator also incorporates a module that allows users to report bot infection profiles to a remote repository for global collection and evaluation of bot activity. For this purpose, we utilize the Cyber-TA privacy-enabled alert delivery infrastructure [32]. Our delivery infrastructure first anonymizes all source-local addresses reported within the bot infection profile, and then delivers the profile to our data repository through a TLS over TOR [15] (onion routing protocol) network connection. These profiles will be made available to the research community, ideally to help in the large-scale assessment of bot dialog behavior, the sources and volume of various bot infections, and for surveying where C&C servers and exploit sources are located.

4.1 A Multiple-Sensor Approach to Gathering Infection Evidence

4.1.1 SCADE: Statistical sCan Anomaly Detection Engine

Recent measurement studies suggest that modern bots are packaged with around 15 exploit vectors on average [33] to improve opportunities for exploitation. Depending on how the attack source scans its target, we are likely to encounter some failed connection attempts prior to a successful infection.
To address this form aspect of malware interaction, we have designed SCADE, a Snort preprocessor plug-in with two modules, one for inbound scan detection (E1 dialog warnings) and another for detecting outbound attack propagations (E5 dialog warnings) once our local system is infected. SCADE E1 alarms provide a potential early bound on the start of an infection, should this scan eventually lead to a successful infection.
Inbound Scan Detection: SCADE is similar in principle to existing scan detection techniques like [35,24]. However, SCADE has been specifically weighted toward the detection of scans involving the ports often used by malware. It is also less vulnerable to DoS attacks because its memory trackers do not maintain per-source-IP state. Similar to the scan detection technique proposed in [48], SCADE tracks only scans that are specifically targeted to internal hosts, bounding its memory usage to the number of inside hosts. SCADE also bases its E1 scan detection on failed connection attempts, further narrowing its processing. We define two types of ports: HS (high-severity) ports representing highly vulnerable and commonly exploited services (e.g., 80/HTTP, 135,1025/DCOM, 445/NetBIOS, 5000/UPNP, 3127/MyDoom) and LS (low-severity) ports.[2] Currently, we define 26 TCP and 4 UDP HS ports and mark all others as LS ports. We set different weights to a failed scan attempt to different types of ports. An E1 dialog warning for a local host is produced based on an anomaly score that is calculated as $s = w_1F_{hs}+w_2F_{ls}$, where F_{hs} and F_{ls} indicate numbers of cumulative failed attempts at high-severity and low-severity ports, respectively.
Outbound Scan Detection: SCADE's outbound scan detection coverage for E5 dialog warnings is based on a voting scheme (AND, OR or MAJORITY) of three parallel anomaly detection models that track all external outbound connections per internal host:

- *Outbound scan rate (s_1):* Detects local hosts that conduct high-rate scans across large sets of external addresses.
- *Outbound connection failure rate (s_2):* Detects abnormally high connection fail rates, with sensitivity to HS port usage. We calculate the anomaly score $s_2=(w_1F_{hs}+w_2F_{ls})/C$, where C is the total number of scans from the host within a time window.
- *Normalized entropy of scan target distribution (s_3):* Calculates a Zipf (power-law) distribution of outbound address connection patterns [3]. A uniformly distributed scan target pattern provides an indication of a potential outbound scan. We use an anomaly scoring technique based on normalized entropy to identify such candidates: $s_3 = H/\ln(m)$, where the entropy of scan target distribution is $H=-\sum_{i=1}^{m}p_i\ln(p_i)$, m is the total number of scan targets, and p_i is the percentage of the scans at target i.

Each anomaly module issues a subalert when $s_i \geq t_i$, where t_i is a threshold. SCADE then uses a user-configurable "voting scheme", *i.e.*, AND, OR, or MAJORITY, to combine the alerts from the three modules. For example, the AND rule dictates that SCADE issues an alert when all three modules issue an alert.

4.1.2 SLADE: Statistical PayLoad Anomaly Detection Engine

SLADE is an anomaly-based engine for payload exploit detection. It examines the payload of every request packet sent to monitored services and outputs an alert if its lossy n-gram frequency deviates from an established normal profile.

SLADE is similar to PAYL [46], which is a payload-based 1-gram byte distribution anomaly detection scheme. PAYL examines the 1-gram byte distribution of the packet payload, *i.e.*, it extracts 256 features each representing the occurrence frequency of one of the 256 possible byte values in the payload. A normal profile for a service/port, e.g., HTTP, is constructed by calculating the average and standard deviation of the feature vector of the normal traffic to the port. PAYL calculates deviation distance of a test payload from the normal profile using a simplified Mahalanobis distance, $d(x,y)=\sum_{i=0}^{255} (|x_i-y_i|)/(s_i+a)$, where y_i is the mean, s_i is the standard deviation, and a is a smoothing factor. A payload is considered as anomalous if this distance exceeds a predetermined threshold. PAYL is effective in detecting worm exploits with a reasonable false positive rate as shown in [46,47]. However, it could be evaded by a polymorphic blending attack (PBA) [18]. As discussed in [47,18,31], a generic n-gram version of PAYL may help to improve accuracy and the hardness of evasion. The n-grams extract n-byte sequence information from the payload, which helps in constructing a more precise model of the normal traffic compared to the single-byte (*i.e.*, 1-gram) frequency-based model. In this case the feature space in use is not 256, but 256^n dimensional. It is impractical to store and compute in a 256^n dimension space for high-n-grams.

SLADE makes the n-gram scheme practical by using a lossy structure while still maintaining approximately the same accuracy as the original full n-gram version. We use a fixed vector counter (with size v) to store a lossy n-gram distribution of the payload. When processing a payload, we sequentially scan n-gram substring str, apply some universal hash function h(), and increment the counter at the vector space indexed by h(str) mod v. We then calculate the distribution of the hashed n-gram indices within this (much) smaller vector space v. We define F as the feature space of n-gram PAYL (with a total of 256^n distinct features), and F' as the feature space of SLADE (with v features).

This hash function provides a mapping from F to F' that we utilize for space efficiency. We require only v (e.g., v=2,000), whereas n-gram PAYL needs 256^n (e.g., even for a small n=3, $256^3=2^{24}\approx$ 16M). The computational complexity in examining each payload is still linear (O(L), where L is the length of payload), and the complexity in calculating distance is O(v) instead of 256^n. Thus, the runtime performance of SLADE is comparable to 1-gram PAYL. Also note that although both use hashing techniques, SLADE is different from Anagram [45], which uses a Bloom filter to store all n-gram substrings from normal payloads. The hash function in SLADE is for feature compression and reduction, however the hash functions in Anagram are to reduce the false positives of string lookup in Bloom filter. In essence, Anagram is like a content matching scheme. It builds a huge knowledge base of all known good n-gram substrings using efficient storage and query optimizations provided by bloom filters, and examines a payload to determine whether the number of its n-gram substrings not in the knowledge base exceeds a threshold.

A natural concern of using such a lossy data structure is the issue of accuracy: how many errors (false positives and false negatives) may be introduced because of the lossy representation? To answer this question, we perform the following simple analysis.[3] Let us first overview the reason why the original n-gram PAYL can detect anomalies. We use g to represent the number of non-zero value features in F for a normal profile used by PAYL. Similarly, g' is the number of non-zero value features in F' for a normal profile used by SLADE. For a normal payload of length=L, there is a total of l=(L-n+1) n-gram substrings. Among these l substrings, 1-b_n percent substrings converge to g distinct features in the normal profile, *i.e.*, these substrings share similar distributions as the normal profile. The remaining (small portion) b_n percent of substrings are considered as noise substrings that do not belong to the g features in the normal profile. For a malicious payload, if it can be detected as an anomaly, it should have a much larger portion of noise substrings b_a ($b_a > b_n$).

We first analyze the false positives when using the lossy structure representation to see how likely SLADE will detect a normal (considered normal by n-gram PAYL) payload as anomalous. For a normal payload, the hashed indices of a 1-b_n portion of substrings (that converge to g distinct features in F for the normal profile of PAYL) should now converge in the new vector space (into g' distinct features in F' for the normal profile of SLADE). Because of the universal hash function, hashed indices of the b_n portion of noise substrings are most likely uniformly distributed into F'. As a result, some of the original noise substrings may actually be hashed to the g' distinct features in the normal profile of SLADE (*i.e.*, they may not be noise in the new feature space now). Thus, the deviation distance (*i.e.*, the anomaly score) can only decrease in SLADE. Hence, we conclude that SLADE may not have a higher false positive rate than n-gram PAYL.

Now let us analyze the false negative rate, *i.e.*, the likelihood that SLADE will treat a malicious payload (as would be detected by n-gram PAYL) as normal. False negatives happen when the hash collisions in the lossy

structure mistakenly map a b_a portion of noise substrings into the g' features (*i.e.*, the normal profile) for SLADE. By using the universal hash function, the probability for a noise substring to fall into g' out of v space is $[(g')/v]$. Thus, the probability for all the lb_a noise substrings to collide into the g' portion is about $([(g')/v])^{lb_a}$. For example, if we assume v=2,000, g'=200, lb_a=100, then this probability is about $(200/2000)^{100}$=1e-100 $\approx$ 0. In practice, the probability of such collisions for partial noise substrings is negligible. Thus, we believe that SLADE does not incur a significant accuracy penalty compared to full n-gram PAYL, while significantly reducing its storage and computation complexity.

We measured the performance of SLADE in comparison to 1-gram PAYL by using the same data set as in [31]. The training and test data sets used were from the first and following four days of HTTP requests from the Georgia Tech campus network, respectively. The attack data consists of 18 HTTP-based buffer overflow attacks, including 11 regular (nonpolymorphic) exploits, 6 mimicry exploits generated by CLET, and 1 polymorphic blending attack used in [18] to evade 2-gram PAYL. In our experiment, we set n=4,v=2,048.[4] Table 1 summarizes our experimental results. Here, DFP is the desired false positive rate, *i.e.*, the rejection rate in the training set. RFP is the "real" false positive rate in our test data set. The detection rate is measured on the attack data set and is defined as the number of attack packets classified as anomalous divided by the total number of packets in the attack instances. We conclude from the results that SLADE performs better with respect to both DFP and RFP than the original PAYL (1-gram) system. Furthermore, we discovered that the minimum RFP for which PAYL is able to detect all attacks, including the polymorphic blending attack, is 4.02%. This is usually considered intolerably high for network intrusion detection. On the other hand, the minimum RFP required for SLADE to detect all attacks is 0.3601%. As shown in [31], 2-gram PAYL does not detect the polymorphic blending attack even if we are willing to tolerate an RFP as high as 11.25%. This is not surprising given that the polymorphic blending attack we used was specifically tailored to evade 2-gram PAYL. We also find that SLADE is comparable to (or even better than) a well-constructed ensemble IDS that combines 11 one-class SVM classifiers [31], and detects all the attacks, including the polymorphic blending attack, for an RFP at around 0.49%. SLADE also has the added advantage of more efficient resource utilization, which results in shorter training and execution times when compared to the ensemble IDS.

Table 1: Performance of 1-gram PAYL and SLADE

	DFP(%)	0.0	0.01	0.1	1.0	2.0	5.0	10.0
PAYL	RFP(%)	0.00022	0.01451	0.15275	0.92694	1.86263	5.69681	11.05049
	Detected Attacks	1	4	17	17	17	18	18
	Detection Rate(%)	0.8	17.5	69.1	72.2	72.2	73.8	78.6
SLADE	RFP(%)	0.0026	0.0189	0.2839	1.9987	3.3335	6.3064	11.0698
	Detected Attacks	3	13	17	18	18	18	18
	Detection Rate(%)	20.6	74.6	92.9	99.2	99.2	99.2	99.2

4.1.3 Signature Engine: Bot-Specific Heuristics

Our final sensor contributor is the Snort signature engine. This module plays a significant role in detecting several of the classes of dialog warnings from our bot infection dialog model. Snort is our second sensor source for direct exploit detection (class E2), and our primary source for binary downloading (E3) and C&C communications (E4). We organize the rules selected for BotHunter into four separate rule files, covering 1046 E2 rules, 71 E3 rules, 246 E4 rules, and a small collection of 20 E5 rules, for total of 1383 heuristics. The rules are primarily derived from the Bleeding-Edge [10] and SourceFire's registered free rulesets.

All the rulesets were selected specifically for their relevance to malware identification. Our rule selections are continually tested and reviewed across operational networks and our live honeynet environment. It is typical for our rule-based heuristics to produce less than 300 dialog warnings per 10-day period monitoring an operational border switch space port of approximately 130 operational hosts (SRI Computer Science Laboratory).

Our E2 ruleset focuses on the full spectrum of external to internal exploit injection attacks, and has been tested and augmented with rules derived from experimentation in our medium and high interactive honeynet environment, where we can observe and validate live malware infection attempts. Our E3 rules focus on (malware) executable download events from external sites to internal networks, covering as many indications of

(malicious) binary executable downloads and download acknowledgment events as are in the publicly available Snort rulesets. Our E4 rules cover internally-initiated bot command and control dialog, and acknowledgment exchanges, with a significant emphasis on IRC and URL-based bot coordination.[5] Also covered are commonly used Trojan backdoor communications, and popular bot commands built by keyword searching across common major bot families and their variants. A small set of E5 rules is also incorporated to detect well-known internal to external backdoor sweeps, while SCADE provides the more in-depth hunt for general outbound port scanning.

4.2 Dialog-Based IDS Correlation Engine

The BotHunter correlator tracks the sequences of IDS dialog warnings that occur between each local host and those external entities involved in these dialog exchanges. Dialog warnings are tracked over a temporal window, where each contributes to an overall infection sequence score that is maintained per local host. We introduce a data structure called the *network dialog correlation matrix*, which is managed, pruned, and evaluated by our correlation engine at each dialog warning insertion point. Our correlator employs a weighted threshold scoring function that aggregates the weighted scores of each dialog warning, declaring a local host infected when a minimum combination of dialog transactions occur within our temporal pruning interval.
Figure 4 illustrates the structure of our *network dialog correlation matrix*. Each dynamically-allocated row corresponds to a summary of the ongoing dialog warnings that are raised between an individual local host and other external entities. The BotHunter correlator manages the five classes of dialog warnings presented in Section 3 (E1 through E5), and each event cell corresponds to one or more (possibly aggregated) sensor alerts that map into one of these five dialog warning classes. This correlation matrix dynamically grows when new activity involving a local host is detected, and shrinks when the observation window reaches an interval expiration.

Int. Host	Timer	E1	E2	E3	E4	E5
192.168.12.1		$A_a \ldots A_b$				
192.168.10.45			$A_c \ldots A_d$		$A_e \ldots A_f$	
192.168.10.66			A_g			
192.168.12.46					$A_h \ldots A_i$	$A_j \ldots A_k$
:						
192.168.11.123		A_l	$A_m \ldots A_n$	A_o		

Figure 4: BotHunter Network Dialog Correlation Matrix

In managing the dialog transaction history we employ an interval-based pruning algorithm to remove old dialog from the matrix. In Figure 4, each dialog may have one or two expiration intervals, corresponding to a *soft prune timer* (the open-faced clocks) and a *hard prune timer* (the filled clocks). The hard prune interval represents a fixed temporal interval over which dialog warnings are allowed to aggregate, and the end of which results in the calculation of our threshold score. The soft prune interval represents a smaller temporal window that allows users to configure tighter pruning interval requirements for high-production dialog warnings (inbound scan warnings are expired more quickly by the soft prune interval), while the others are allowed to accumulate through the hard prune interval. If a dialog warning expires solely because of a soft prune timer, the dialog is summarily discarded for lack of sufficient evidence (an example is row 1 in Figure 4 where only E1 has alarms). However, if a dialog expires because of a hard prune timer, the dialog threshold score is evaluated, leading either to a bot declaration or to the complete removal of the dialog trace should the threshold score be found insufficient.
To declare that a local host is infected, BotHunter must compute a sufficient and minimum threshold of evidence (as defined in Section 3) within its pruning interval. BotHunter employs two potential criteria required for bot declaration: 1) an incoming infection warning (E2) followed by outbound local host coordination or exploit propagation warnings (E3-E5), or 2) a minimum of at least two forms of outbound bot dialog warnings (E3-E5). To translate these requirements into a scoring algorithm we employ a regression model to estimate dialog warning weights and a threshold value, and then test our values against a corpus of malware infection traces. We define an expectation table of predictor variables that match our conditions and apply a regression model where the estimated regression coefficients are the desired weights shown in Table 2. For completeness, the computed expectation table is provided in the project website [1].

	Coefficients	Standard Error
E1	0.09375	0.100518632
E2 rulebase	0.28125	0.075984943
E2 slade	0.09375	0.075984943

E3	0.34375	0.075984943
E4	0.34375	0.075984943
E5	0.34375	0.075984943

Table 2: Initial Weighting

These coefficients provide an approximate weighting system to match the initial expectation table [6]. We apply these values to our expectation table data to establish a threshold between bot and no-bot declaration. Figure 5 illustrates our results, where bot patterns are at X-axis value 1, and non-bot patterns are at X-axis 0. Bot scores are plotted vertically on the Y-axis. We observe that all but one non-bot patterns score below 0.6, and all but 2 bot patterns score above 0.65. Next, we examine our scoring model against a corpus of BotHunter IDS warning sets produced from successful bot and worm infections captured in the SRI honeynet between March and April 2007. Figure 6 plots the actual bot scores produced from these real bot infection traces. All observations produce BotHunter scores of 0.65 or greater.

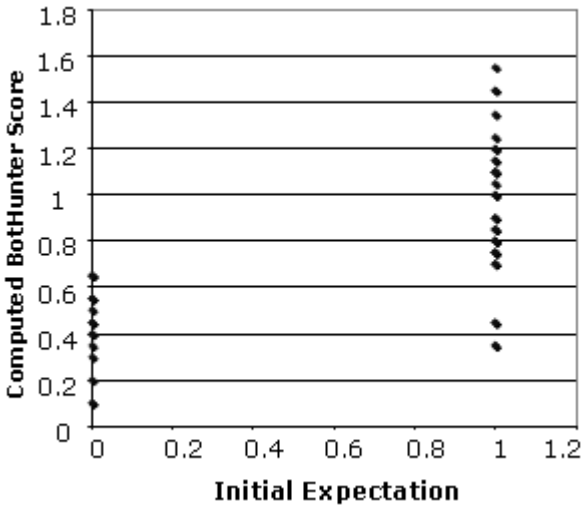

Figure 5: Scoring Plot from Expectation Table

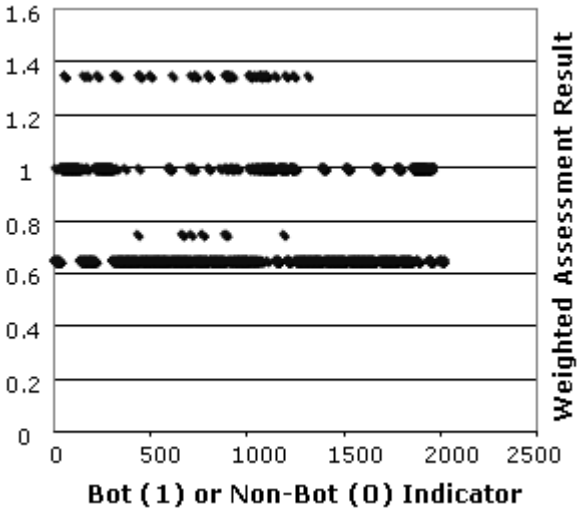

Figure 6: Scoring Plot: 2019 Real Bot Infections

When a dialog sequence is found to cross the threshold for bot declaration, BotHunter produces a *bot profile.* The bot profile represents a full analysis of roles of the dialog participants, summarizes the dialog alarms based on which dialog classes (E1-E5) the alarms map, and computes the infection time interval. Figure 7 (right) provides an example of a bot profile produced by the BotHunter correlation engine. The bot profile begins with an overall dialog anomaly score, followed by the IP address of the infected target (the victim machine), infector list, and possible C&C server. Then it outputs the dialog observation time and reporting time. The raw alerts specific to this dialog are listed in an organized (E1-E5) way and provide some detailed information.

5 Evaluating Detection Performance

To evaluate BotHunter's performance, we conducted several controlled experiments as well as real world deployment evaluations. We begin this section with a discussion of our detection performance while exposing

BotHunter to infections from a wide variety of bot families using *in situ* virtual network experiments. We then discuss a larger set of true positive and false negative results while deploying BotHunter to a live VMWare-based high-interaction honeynet. This recent experiment exposed BotHunter to 2,019 instances of Windows XP and Windows 2000 direct-exploit malware infections from the Internet. We follow these controlled experiments with a brief discussion of an example detection experience using BotHunter during a live operational deployment. Next, we discuss our broader testing experiences in two network environments. Here, our focus is on understanding BotHunter's daily false positive (FP) performance, at least in the context of two significantly different operational environments. A false positive in this context refers to the generation of a *bot profile* in response to a non-infection traffic flow, not to the number of IDS dialog warnings produced by the BotHunter sensors. As stated previously, network administrators are not expected to analyze individual IDS alarms. Indeed, we anticipate external entities to regularly probe and attack our networks, producing a regular flow of dialog warnings. Rather, we assert (and validate) that the dialog combinations necessary to cause a bot detection should be rarely encountered in normal operations.

5.1 Experiments in an *In situ* Virtual Network

Our evaluation setup uses a virtual network environment of three VMware guest systems. The first is a Linux machine with IRC server installed, which is used as the C&C server, and the other two are Windows 2000 instances. We infect one of the Windows instances and wait for it to connect to our C&C server. Upon connection establishment, we instruct the bot to start scanning and infecting neighboring hosts. We then await the infection and IRC C&C channel join by the second Windows instance. By monitoring the network activity of the second victim, we capture the full infection dialog. This methodology provides a useful means to measure the false negative performance of BotHunter.

We collected 10 different bot variants from three of the most well-known IRC-based bot families [20]: Agobot/Gaobot/Phatbot, SDBot/RBot/UrBot/UrXBot, and the mIRC-based GTbot. We then ran BotHunter in this virtual network and limited its correlation focus on the victim machine (essentially we assume the HOMENET is the victim's IP). BotHunter successfully detected all bot infections (and produced bot profiles for all).

We summarize our measurement results for this virtual network infection experiment in Table 3. We use Yes or No to indicate whether a certain dialog warning is reported in the final profile. The two numbers within brackets are the number of generated dialog warnings in the whole virtual network and the number involving our victim, respectively. For example, for Phatbot-rls, 2,834 dialog warnings are generated by E2[rb] ([rb] means Snort rule base, [sl] means SLADE), but only 46 are relevant to our bot infection victim. Observe that although many warnings are generated by the sensors, only one bot profile is generated for this infection. This shows that BotHunter can significantly reduce the amount of information a security administrator needs to analyze. In our experiments almost all sensors worked as we expected. We do not see E1 events for RBot because the RBot family does not provide any commands to trigger a vertical scan for all infection vectors (such as the "scan.startall" command provided by the Agobot/Phatbot family). The bot master must indicate a specific infection vector and port for each scan. We set our initial infection vector to DCOM, and since this was successful the attacking host did not attempt further exploits.

Table 3: Dialog Summary of Virtual Network Infections

	E1	E2[rb]	E2[sl]	E3	E4	E5
agobot3-priv4	Yes(2/2)	Yes(9/8)	Yes(6/6)	Yes(5)	Yes(38/8)	Yes(4/1)
phat-alpha5	Yes(14/4)	Yes(5,785/5,721)	Yes(6/2)	Yes(3/3)	Yes(28/26)	Yes(4/2)
phatbot-rls	Yes(11/3)	Yes(2,834/46)	Yes(6/2)	Yes(8/8)	Yes(69/20)	Yes(6/2)
rbot0.6.6	No(0)	Yes(2/1)	Yes(2/1)	Yes(2/2)	Yes(65/24)	Yes(2/1)
rxbot7.5	No(0)	Yes(2/2)	Yes(2/2)	Yes(2/2)	Yes(70/27)	Yes(2/1)
rx-asn-2-re-workedv2	No(0)	Yes(4/3)	Yes(3/2)	Yes(2/2)	Yes(59/18)	Yes(2/1)
Rxbot-ak-0.7-Modded.by.Uncanny	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(73/26)	Yes(2/1)
sxtbot6.5	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(65/24)	Yes(2/1)
Urx-Special-Ed-UltrA-2005	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(68/22)	Yes(2/1)
gt-with-dcom-profile1	No(1/0)	Yes(5/3)	Yes(6/2)	No(0)	Yes(221/1)	No(4/0)
gt-with-dcom-profile2	No(1/0)	No(5/0)	No(6/0)	No(0)	Yes(221/44)	Yes(4/2)
gt-with-dcom-10min-profile	No(1/0)	Yes(5/3)	Yes(6/3)	No(0)	Yes(221/51)	Yes(4/2)

Note that two profiles are reported in the gt-with-dcom case. In the first profile, only E2[rb],E2[sl] and E4 are observed. In profile 2, E4 and E5 are observed (which is the case where we miss the initial infection periods). Because this infection scenario is very slow and lasts longer than our 4-minute correlation time window. Furthermore, note that we do not have any detected E3 dialog warnings reported for this infection sequence. Regardless, BotHunter successfully generates an infection profile. This demonstrates the utility of BotHunter's evidence-trail-based dialog correlation model. We also reran this experiment with a 10-minute correlation time window, upon which BotHunter also reported a single infection profile.

5.2 SRI Honeynet Experiments

Our experimental honeynet framework has three integral components. The first component *Drone manager* is a software management component that is responsible for keeping track of drone availability and forwarding packets to various VMware instances. The address of one of the interfaces of this Intel Xeon 3 GHz dual core system is set to be the static route for the unused /17 network. The other interface is used for communicating with the high-interaction honeynet. Packet forwarding is accomplished using network address translation. One important requirements for this system is to keep track of infected drone systems and to recycle uninfected systems. Upon detecting a probable infection (outbound connections), we mark the drone as "tainted" to avoid reassigning that host to another source. Tainted drones are saved for manual analysis or automatically reverted back to previous clean snapshots after a fixed timeout. One of the interesting observations during our study was that most infection attempts did not succeed even on completely unpatched Windows 2000 and Windows XP systems. As a result, a surprisingly small number of VM instances was sufficient to monitor the sources contacting the entire /17 network. The second component is the *high-interaction-honeynet* system, which is hosted in a high-performance Intel Xeon 3 GHz dual core, dual CPU system with 8 GB of memory. For the experiments listed in this paper, we typically ran the system with 9 Win-XP instances, 14 Windows 2000 instances (with two different service pack levels), and 3 Linux FC3 instances. The system was moderately utilized in this load. The final component is the *DNS/DHCP server*, which dynamically assigns IP addresses to VMware instances and also answers DNS queries from these hosts.

Over a 3-week period between March and April 2007, we analyzed a total of 2,019 successful WinXP and Win2K remote-exploit bot or worm infections. Each malware infection instance succeeded in causing the honeypot to initiate outbound communications related to the infection. Through our analysis of these traces using BotHunter sensor logs, we were able to very reliably observe the malware communications associated with the remote-to-local network service infection and the malware binary acquisition (egg download). In many instances we also observed the infected honeypot proceed to establish C&C communications and attempt to propagate to other victims in our honeynet. Through some of this experiment, our DNS service operated unreliably and some C&C coordination events were not observed due to DNS lookup failures.

Figure 7 illustrates a sample infection that was detected using the SRI honeynet, and the corresponding BotHunter profile. W32/IRCBot-TO is a very recent (released January 19, 2007) network worm/bot that propagates through open network shares and affects both Windows 2000 and Windows XP systems [37]. The worm uses the IPC share to connect to the `SRVSVC` pipe and leverages the MS06-40 exploit [27], which is a buffer overflow that enables attackers to craft RPC requests that can execute arbitrary code. This mechanism is used to force the victim to fetch and execute a binary named netadp.exe from the system folder. The infected system then connects to the z3net IRC network and joins two channels upon which it is instructed to initiate scans of 203.0.0.0/8 network on several ports. Other bot families successfully detected by BotHunter included variants of W32/Korgo, W32/Virut.A and W32/Padobot.

Overall, BotHunter detected a total of 1,920 of these 2,019 successful bot infections. This represents a **95.1%** true positive rate. All malware instances observed during this period transmitted their exploits through ports TCP-445 or TCP-139. This is very common behavior, as the malware we observe tends to exploit the first vulnerable port that replies to a targeted scans, and ports TCP-445 and TCP-139 are usually among the first ports tested. The infection set analyzed exhibited limited diversity in the infection transmission methods, and overall we observed roughly 40 unique patterns in the dialog warnings produced.

This experiment produced 99 bot infections that *did not* produce bot profiles, *i.e.,* a **4.9%** false negative rate. To explain these occurrences we manually examined each bot infection trace that eluded BotHunter, using tcpdump and ethereal. The reasons for these failed bot detections can be classified into three primary categories: infection failures, honeynet setup or policy failures, or data corruption failures.

- *Infection failures:* We observed infections in which the exploit apparently led to instability and eventual failure in the infected host. More commonly, we observed cases in which the infected victim attempt to

"phone home," but the SYN request received no reply.
- *Honeynet setup and policy failures:* We observed that our NAT mechanism did not correctly translate application-level address requests (e.g., ftp PORT commands). This prevented several FTP egg download connection requests from proceeding, which would have otherwise led to egg download detections. In addition, some traces were incomplete due to errors in our honeypot recycling logic which interfered with our observation of the infection logic.
- *Data corruption failures:* Data corruption was the dominant reason (86% of the failed traces) in preventing our BotHunter sensors from producing dialog warnings. We are still investigating the cause behind these corruptions, but suspect that these likely happened during log rotations by the Drone manager.

Discussion: In addition to the above false negative experiences, we also recognize that others reasons could prevent BotHunter from detecting infections. A natural extension of our *infection failures* is for a bot to purposely lay dormant once it has infected a host to avoid association of the infection transmission with an outbound egg download or coordination event. This strategy could be used successfully to circumvent BotHunter deployed with our default fixed pruning interval. While we found some infected victims failed to phone home, we could also envision the egg download source eventually responding to these requests after the BotHunter pruning interval, causing a similar missed association. Sensor coverage is of course another fundamental concern for any detection mechanism. Finally, while these results are highly encouraging, the honeynet environment provided a low-diversity in bot infections, in which attention was centered on direct exploits of TCP-445 and TCP-139. We did not provide a diversity of honeypots with various OSs, vulnerable services, or Trojan backdoors enabled, to fully examine the behavioral complexities of bots or worms.

5.3 An Example Detection in a Live Deployment

In addition to our laboratory and honeynet experiences, we have also fielded BotHunter to networks within the Georgia Tech campus network and within an SRI laboratory network. In the next sections we will discuss these deployments and our efforts to evaluate the false positive performance of BotHunter. First, we will briefly describe one example host infection that was detected using BotHunter within our Georgia Tech campus network experiments.

In early February 2007, BotHunter detected a bot infection that produced E1, E4 and E5 dialog warnings. Upon inspection of the bot profile, we observed that the bot-infected machine was scanned, joined an IRC channel, and began scanning other machines during the BotHunter time window. One unusual element in this experience was the omission of the actual infection transmission event (E2), which is observed with high-frequency in our live honeynet testing environment. We assert that the bot profile represents an actual infection because during our examination of this infection report, we discovered that the target of the E4 (C&C Server) dialog warning was an address that was blacklisted both by the ShadowServer and the botnet mailing list as a known C&C server during the time of our bot profile.

5.4 Experiments in a University Campus Network

In this experiment, we evaluate the detection and false positive performance of BotHunter in a production campus network (at the College of Computing [CoC] at Georgia Tech). The time period of this evaluation was between October 2006 and February 2007.

The monitored link exhibits typical diurnal behavior and a sustained peak traffic of over 100 Mbps during the day. While we were concerned that such traffic rates might overload typical NIDS rulesets and real-time detection systems, our experience shows that it is possible to run BotHunter live under such high traffic rates using commodity PCs. Our BotHunter instance runs on a Linux server with an Intel Xeon 3.6 GHz CPU and 6 GB of memory. The system runs with average CPU and memory utilization of 28% and 3%, respectively.

To evaluate the representativeness of this traffic, we randomly sampled packets for analysis (about 40 minutes). The packets in our sample, which were almost evenly distributed between TCP and UDP, demonstrated wide diversity in protocols, including popular protocols such as HTTP, SMTP, POP, FTP, SSH, DNS, and SNMP, and collaborative applications such as IM (e.g., ICQ, AIM), P2P (e.g., Gnutella, Edonkey, bittorrent), and IRC, which share similarities with infection dialog (*e.g.*, two-way communication). We believe the high volume of background traffic, involving large numbers of hosts and a diverse application mix, offers an appealing environment to confirm our detection performance, and to examine the false positive question.

First, we evaluated the detection performance of BotHunter in the presence of significant background traffic. We injected bot traffic captured in the virtual network (from the experiments described in Section 5.1) into the

captured Georgia Tech network traffic. Our motivation was to simulate real network infections for which we have the ground truth information. In these experiments, BotHunter correctly detected all 10 injected infections (by the 10 bots described in Section 5.1).

Table 4: Raw Alerts of BotHunter in 4 Month Operation in CoC Network

Event	E1	E2[rb]	E2[sl]	E3	E4	E5
Alert#	550,373	950,112	316,467	1,013	697,374	48,063

Next, we conducted a longer-term (4 months) evaluation of false alarm production. Table 4 summarizes the number of dialog warnings generated by BotHunter for each event type from October 2006 to January 2007. BotHunter sensors generated about 2,563,402 (more than 20,000 per day) raw dialog warnings from all the five event categories. For example, many E3 dialog warnings report on Windows executable downloads, which by themselves do not shed light on the presence of exploitable vulnerabilities. However, our experiments do demonstrate that the alignment of the bot detection conditions outline in Section 3 rarely align within a stream of dialog warnings from normal traffic patterns. In fact, only 98 profiles were generated in 4 months, less than one per day on average.

In further analyzing these 98 profiles, we had the following findings. First, there are no false positives related to any normal usage of collaborative applications such as P2P, IM, or IRC. Almost two-thirds (60) of the bot profiles involved access to an MS-Exchange SMB server (33) and SMTP server (27). In the former case, the bot profiles described a NETBIOS SMB-DS IPC$ unicode share access followed by a windows executable downloading event. Bleeding Edge Snort's IRC rules are sensitive to some IRC commands (e.g., USER) that frequently appear in the SMTP header. These issues could easily be mitigated by additional whitelisting of certain alerts on these servers. The remaining profiles contained mainly two event types and with low overall confidence scores. Additional analysis of these incidents was complicated by the lack of full packet traces in our high-speed network. We can conservatively assume that they are false positives and thus our experiments here provide a reasonable estimate of the upper bound on the number of false alarms (less than one) in a busy campus network.

5.5 Experiments in an Institutional Laboratory

We deployed BotHunter live on a small well-administered production network (a lightly used /17 network that we can say with high confidence is infection free). Here, we describe our results from running BotHunter in this environment. Our motivation for conducting this experiment was to obtain experience with false positive production in an operational environment, where we could also track all network traces and fully evaluate the conditions that may cause the production of any unexpected bot profiles.

BotHunter conducted a 10-day data stream monitoring test from the span port position of an egress border switch. The network consists of roughly 130 active IP addresses, an 85% Linux-based host population, and an active user base of approximately 54 people. During this period, 182 million packets were analyzed, consisting of 152 million TCP packets (83.5%), 15.8 million UDP packets (8.7%), and 14.1 million ICMP packets (7.7%). Our BotHunter sensors produced 5,501 dialog warnings, composed of 1,378 E1 scan events, 20 E2 exploit signature events, 193 E3 egg-download signature events, 7 E4 C&C signature events and 3,904 E5 scan events. From these dialog warnings, the BotHunter correlator produced just one bot profile. Our subsequent analysis of the packets that caused the bot profile found that this was a false alarm. Upon packet inspection, it was found that the session for which the bot declaration occurred consisted of a 1.6 GB multifile FTP transfer, during which a binary image was transferred with content that matched one of our buffer overflow detection patterns. The buffer overflow false alarm was coupled with a second MS Windows binary download, which caused BotHunter to cross our detection threshold and declare a bot infection.

6 <-> <infector-ip> TCP 2971 - <honey-ip> 445 [SYN, SYN,ACK]	**Score:** 1.95 (>= 0.80)
13 -> SMB Negotiate Protocol Request	**Infected Target:** <honey-ip>
14 <- SMB Negotiate Protocol Response	**Infector List:** <infector-ip>
17 -> SMB Session Setup AndX Request, NTLMSSP_AUTH, User: \	**C & C List:** 66.25.XXX.XXX
18 <- SMB Session Setup AndX Response	Observed Start: 01/18/2007 23:46:54.56 PST
19 -> SMB Tree Connect AndX Request, Path: \\<honey-ip>\IPC\$	Gen. Time: 01/18/2007 23:47:13.18 PST
20 <- SMB Tree Connect AndX Response	
21 -> SMB NT Create AndX Request, Path: \browser	**INBOUND SCAN <unobserved>**
22 <- SMB NT Create AndX Response, FID: 0x4000	
23 -> DCERPC Bind: call_id: 0 UUID: SRVSVC	**EXPLOIT**
24 <- SMB Write AndX Response, FID: 0x4000, 72 bytes	
25 -> SMB Read AndX Request, FID: 0x4000, 4292 bytes at offset 0	event=1:2971 {tcp} E2[rb] NETBIOS SMB-DS IPC\$
26 <- DCERPC Bind_ack	unicode share access 445<-2971 (23:46:54.56 PST)

27 -> SRVSVC NetrpPathCanonicalize request 28 <- SMB Write AndX Response, FID: 0x4000, 1152 bytes 29 -> SMB Read AndX Request, FID: 0x4000, 4292 bytes at offset 0	---------- event=1:99913 {tcp} E2[rb] SHELLCODE x86 0x90 unicode NOOP 445<-2971 (23:46:54.90 PST)
Initiating Egg download 30 <-> <honey-ip> TCP 1028 - <infector-ip> 8295 [SYN, SYNACK] 34-170 114572 byte egg download ...	**EXPLOIT (slade)** event=552:5555002 (15) {tcp} E2[sl] Slade detected suspicious payload exploit with anomaly score 1843.680342.
Connecting to IRC server on port 8080 174 <-> <honey-ip> TCP 1030 - 66.25.XXX.XXX 8080 [SYN, SYNACK] 176 <- NICK [2K|USA|P|00|eOpOgkIc]\r\nUSER 2K-USA 177 -> :server016.z3nnet.net NOTICE AUTH :*** Looking up your hostname...\r\n" ... 179 -> ... PING :B203CFB7 180 <- PONG :B203CFB7 182 -> Welcome to the z3net IRC network ...	**EGG DOWNLOAD** event=1:5001683 {tcp} E3[rb] Windows executable sent potential malware egg 1028<-8295 (01:45:56.69 EST) **C&C TRAFFIC** event=1:2002023 {tcp} E4[rb] BLEEDING-EDGE TROJAN IRC USER command 1030->8080 (23:47:01.23 PST) ---------- event=1:2002024 {tcp} E4[rb] BLEEDING-EDGE TROJAN IRC NICK command 1030->8080 (23:47:01.23 PST)
Joining channels and setting mode to hidden 183 -> MODE [2K|USA|P|00|eOpOgkIc] +x\r\nJOIN ##RWN irt3hrwn\r\n	---------- event=1:2002025 {tcp} E4[rb] BLEEDING-EDGE TROJAN IRC JOIN command 1030->8080 (23:47:03.79 PST)
Start scanning 203.0.0.0/8 185 -> ....scan.stop -s; .scan.start NETAPI 40 -b -s; .scan.start NETAPI 203.x.x.x 20 -s; .scan.start NETAPI 20 -a -s;.scan.start SYM 40 -b -s; .scan.start MSSQL 40 -b -s\r\n... 191 -> 203.7.223.231 TCP 1072 > 139 [SYN] 192 -> 203.199.174.117 TCP 1073 > 139 [SYN] scan,scan...	**OUTBOUND SCAN** event=1:2001579 {tcp} E5[rb] BLEEDING-EDGE Behavioral Unusual Port 139 traffic, Potential Scan or Infection 1089->139 (01:46:06 EST) event=555:5555005 {tcp} E5[sc] scade detected scanning of 21 IPs (fail ratio=0:0/21): 0->0 (01:46:06 EST)

Figure 7: Honeynet Interaction Summary (left) and corresponding BotHunter Profile (right) for W32/IRCBot-TO

6 Limitations and Future Work

Several important practical considerations present challenges in extending and adapting BotHunter for arbitrary networks in the future.

Adapting to Emerging Threats and Adversaries: Network defense is a perennial arms race[7] and we anticipate that the threat landscape could evolve in several ways to evade BotHunter. First, bots could use encrypted communication channels for C&C. Second, they could adopt more stealthy scanning techniques. However, the fact remains that hundreds of thousands of systems remain unprotected, attacks still happen in the clear, and adversaries have not been forced to innovate. Open-source systems such as BotHunter would raise the bar for successful infections. Moreover, BotHunter could be extended with anomaly-based "entropy detectors" for identification of encrypted channels. We have preliminary results that are promising and defer deeper investigation to future work. We are also developing new anomaly-based C&C detection schemes (for E4).

It is also conceivable that if BotHunter is widely deployed, adversaries would devise clever means to evade the system, *e.g.,* by using attacks on BotHunter's dialog history timers. One countermeasure is to incorporate an additional random delay to the hard prune interval, thereby introducing uncertainty into how long BotHunter maintains local dialog histories.

Incorporating Additional State Logic: The current set of states in the bot infection model was based on the behavior of contemporary bots. As bots evolve, it is conceivable that this set of states would have to be extended or otherwise modified to reflect the current threat landscape. This could be accomplished with simple configuration changes to the BotHunter correlator. We expect such changes to be fairly infrequent as they reflect fundamental paradigm shifts in bot behavior.

7 Related Work

Recently, there has been a significant thrust in research on botnets. To date, the primary focus of much of this research has been on gaining a basic understanding of the nature and full potential of the botnet threat. Rajab et

al. provided an in-depth study in understanding the dynamics of botnet behavior in the large, employing "longitudinal tracking" of IRC botnets through IRC and DNS tracking techniques [33]. Researchers have also studied the dynamics of botnet C&C protocols [19,50], including global dynamics such as diurnal behavior [14]. Other studies have investigated the internals of bot instances to examine the structural similarities, defense mechanisms, and command and control capabilities of the major bot families [7] and developed techniques to automatically harvest malware samples directly from the Internet [6]. There is also some very recent work on the detection of botnets. Rishi [21] is an IRC botnet detection system that uses n-gram analysis to identify botnet nickname patterns. Binkley and Singh [9] proposed an anomaly based system that combines IRC statistics and TCP work weight for detecting IRC-based botnets. Livadas et al. [26] proposed a machine learning based approach for botnet detection. Karasaridis et al. [25] presented an algorithm for detecting IRC botnet controllers from netflow records. These efforts are complementary in that they could provide additional BotHunter evidence-trails for infection events.

A significant amount of related work has investigated alert correlation techniques for network intrusion detection. An approach to capturing complex and multistep attacks is to explicitly specify the stages, relationships, and ordering among the various constituents of an attack. GriDS [39] aggregates network activity into activity graphs that can be used for analyzing causal structures and identifying policy violations. CARDS is a distributed system for detecting and mitigating coordinated attacks [49]. Abad et al. [2] proposed to correlate data among different sources/logs (e.g., syslog, firewall, netflow) to improve intrusion detection system accuracy. Ellis et al. and Jiang et al. describe two behavioral-based systems for detecting network worms [17,23]. In contrast to the above systems, our work focuses on the problem of bot detection and uses infection dialog correlation as a means to define the probable set of events that indicate a bot infection.

Sommer et al. [36] describe contextual Bro signatures as a means for producing expressive signatures and weeding out false positives. These signatures capture two dialogs and are capable of precisely defining multistep attacks. Our work differs from this in our requirement to simultaneously monitor several flows across many participants (*e.g.*, infection source, bot victim, C&C, propagation targets) and our evidence-trail-based approach to loosely specify bot infections.

JIGSAW is a system that uses notions of concepts and capabilities for modeling complex attacks [40] and [29] provides a formal framework for alert correlation. CAML is a language framework for defining and detecting multistep attack scenarios [11]. Unlike BotHunter, all these systems are based on causal relationships *i.e.,* pre-conditions and post-conditions of attacks. An obvious limitation is that these dependencies, need to be manually specified a priori for all attacks, and yet such dependencies are often unknown.

Alert correlation modules such as CRIM [13] provide the ability to cluster and correlate similar alerts. The system has the capability to extract higher-level correlation rules automatically for the purpose of intention recognition. In [42], Valdes and Skinner propose a two-step probabilistic alert correlation based on attack threads and alert fusion. We consider this line of work to be complementary, *i.e.,* these fusion techniques could be integrated into the BotHunter framework as a preprocessing step in a multisensor environment.

USTAT [22] and NetSTAT [44] are two IDSs based on state transition analysis techniques. They specify computer attacks as sequences of actions that cause transitions in the security state of a system. In [43], multistep attack correlation is performed on attack scenarios specified (a priori) using STATL [16], which is a language for expressing attacks as states and transitions. Our work differs from these systems in that we do not have a strict requirement of temporal sequence, and can tolerate missing events during the infection flow.

8 BotHunter Internet Distribution

We are making BotHunter available as a free Internet distribution for use in testing and facilitating research with the hope that this initiative would stimulate community development of extensions.

A key component of the BotHunter distribution is the Java-based correlator that by default reads alert streams from Snort. We have tested our system with Snort 2.6.* and it can be downloaded from www.cyber-ta.org/botHunter/. A noteworthy feature of the distribution is integrated support for "large-scale privacy-preserving data sharing". Users can enable an option to deliver secure anonymous bot profiles to the Cyber-TA security repository [32], the collection of which we will make available to providers and researchers. The repository is currently operational and in beta release of its first report delivery software.

Our envisioned access model is similar to that of DShield.org [41] with the following important differences. First, our repository is blind to who is submitting the bot report and the system will deliver alerts via TLS over TOR, preventing an association of bot reports to a site via passive sniffing. Second, our anonymization strategy obfuscates all local IP addresses and time intervals in the profile database but preserves C&C, egg download, and attacker addresses that do not match user defined address proximity mask. Users can enable further field

anonymizations as they require. We intend to use contributed bot profiles to learn specific alert signature patterns for specific bots, to track attackers, and to identify C&C sites.

9 Conclusion

We have presented the design and implementation of BotHunter, a perimeter monitoring system for real-time detection of Internet malware infections. The cornerstone of the BotHunter system is a three-sensor dialog correlation engine that performs alert consolidation and evidence trail gathering for investigation of putative infections. We evaluate the system's detection capabilities in an *in situ* virtual network and a live honeynet demonstrating that the system is capable of accurately flagging both well-studied and emergent bots. We also validate low false positive rates by running the system live in two operational production networks. Our experience demonstrates that the system is highly scalable and reliable (very low false positive rates) even with not-so-reliable (weak) raw detectors. BotHunter is also the *first* example of a widely distributed bot infection profile analysis tool. We hope that our Internet release will enable the community to extend and maintain this capability while inspiring new research directions.

10 Acknowledgements

We are thankful to Al Valdes for his help in developing the regression model for computing weights of dialog events. We thank Roberto Perdisci and Junjie Zhang for helpful discussions on the early version of this work. This material is based upon work supported through the U.S. Army Research Office (ARO) under the Cyber-TA Research Grant No.W911NF-06-1-0316 and Grant W911NF0610042, and by the National Science Foundation under Grants CCR-0133629 and CNS-0627477. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of U.S. ARO or the National Science Foundation.

References

[1] Cyber-TA: BotHunter distribution page. https://www.cyber-ta.org/releases/botHunter/index.html, 2007.

[2] C. Abad, J. Taylor, C. Sengul, W. Yurcik, Y. Zhou, and K. Rowe. Log correlation for intrusion detection: A proof of concept. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, page 255, Washington, DC, USA, 2003. IEEE Computer Society.

[3] L. A. Adamic and B. A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3:143-150, 2002.

[4] W32 Agobot IB. https://www.sophos.com/virusinfo/analyses/trojagobotib.html.

[5] K. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. Keromytis. Detecting targeted attacks using shadow honeypots. In *Usenix Security Symposium*, Baltimore, Maryland, August 2005.

[6] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of Recent Advances in Intrusion Detection*, Hamburg, September 2006.

[7] P. Barford and V. Yegneswaran. An inside look at botnets. Special Workshop on Malware Detection, Advances in Information Security, Springer Verlag, 2006.

[8] S. Biles. Detecting the unknown with snort and statistical packet anomaly detection engine (SPADE). https://www.computersecurityonline.com/spade/SPADE.pdf.

[9] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pages 43-48, July 2006.

[10] Bleeding Edge Threats. The Bleeding Edge of Snort. https://www.bleedingsnort.com/, 2007.

[11] S. Cheung, M. Fong, and U. Lindqvist. Modeling multistep cyber attacks for scenario recognition. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[12] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*, 2005.

[13] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of IEEE Symposium on Security and Privacy*, 2002.

[14] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using timezones. In *Proceedings of Network and Distributed Security Symposium(NDSS '06)*, January 2006.

[15] R. Dingledine, N. Mathewson, and P. Syverson. TOR: The second generation onion router. In *Proceedings of the Usenix Security Symposium*, 2004.

[16] S. T. Eckmann, G. Vigna, and R. A. Kemmerer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10, 2002.

[17] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia. A Behavioral Approach to Worm Detection. In *Proceedings of WORM*, 2004.

[18] P. Fogla, M. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee. Polymorphic blending attack. In *Proceedings of the 2006 USENIX Security Symposium*, 2006.

[19] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent denial of service attacks. In *Proceedings of ESORICS*, 2005.

[20]

German Honeynet Project. Tracking botnets. https://www.honeynet.org/papers/bots, 2005.

[21] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[22] K. Iglun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection system. *IEEE Transactions on Software Engineering*, 21, 1995.

[23] X. Jiang and D. Xu. Profiling self-propagating worms via behavioral footprinting. In *Proceedings of CCS WORM* , 2006.

[24] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[25] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[26] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006.

[27] Mitre Corporation. CVE-2006-3439.
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3439.

[28] D. Moore, G. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proceedings of the 10th USENIX Security Symposium*, 2001.

[29] P. Ning, Y. Cui, and D. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of Computer and Communications Security*, 2002.

[30] V. Paxson. BRO: A System for Detecting Network Intruders in Real Time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[31] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'06)*, December 2006.

[32] P. Porras. Privacy-enabled global threat monitoring. In *Proceedings of the IEEE Security and Privacy Magazine*, 2006.

[33] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multi-faceted approach to understanding the botnet phenomenon. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, Brazil, October 2006.

[34] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proceedings of ACM SIGCOMM*, 2006.

[35] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA'99*, 1999.

[36] R. Sommer and V. Paxson. Enhancing byte-level network intrusion detection signatures with context. In *10th ACM Conference on Computer and Communication Security (CCS)*, Washington, DC, October 2003.

[37] Sophos Inc. W32/IRCBot-TO.
https://www.sophos.com/virusinfo/analyses/w32ircbotto.html, 2007.

[38] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. In *Journal of Computer Security*, 2002.

[39] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids-a graph based intrusion detection system for large networks. In *19th National Information Systems Security Conference*, 1996.

[40] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *New Security Paradigms Workshop*, 2000.

[41] J. Ullrich. DSHIELD. https://www.dshield.org, 2007.

[42] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, pages 54-68, 2001.

[43] F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer. Comprehensive approach to intrusion detection alert correlation. In *Proceedings of IEEE Transactions on Dependable and Secure Computing*, 2004.

[44] G. Vigna and R. Kemmerer. NetSTAT: A network-based intrusion detection approach. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC '98)*, 1998.

[45] K. Wang, J. J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006.

[46] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[47] K. Wang and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[48] D. Whyte, P. van Oorschot, and E. Kranakis. Exposure maps: Removing reliance on attribution during scan detection. In *Proceedings of 1st USENIX Workshop on Hot Topics in Security (HotSec'06)*, 2006.

[49] J. Yang, P. Ning, X. S. Wang, and S. Jajodia. Cards: A distributed system for detecting coordinated attacks. In *SEC*, 2000.

[50] V. Yegneswaran, P. Barford, and V. Paxson. Using honeynets for Internet situational awareness. In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets IV)*, College Park, MD, November 2005.

Footnotes:

[1]Our current system implements a classic bot infection dialog model. One can define new models in an XML configuration file and add new detection sensors. Our correlator is IDS-independent, flexible, and extensible to process new models without modification.

[2]Based on data obtained by analyzing vulnerability reports, malware infection vectors and analysis reports of datasets collected at Dshield.org and other honeynets.

[3]We consider our analysis not as an exact mathematical proof, but an analytical description about the intuition behind SLADE.

[4]One can also choose a random v to better defeat evasion attacks like PBA. Also one may use multiple different hash functions and vectors for potential better accuracy and hardness of evasion.

[5]E4 rules are essentially protocol, behavior and payload content signature, instead of a hard-coded known C&C domain list.

[6]In our model, we define E1 scans and the E2 anomaly score (produced by Slade) as increasers to infection confidence, such that our model lowers their weight influence.

[7]In this race, we consider BotHunter to be a substantial technological escalation for the white hats.

Last changed: 6 July 2007 ch