

# Botzilla: Detecting the “Phoning Home” of Malicious Software

Paper Identification Number: SEC-135

## ABSTRACT

Hosts infected with malicious software, so called *malware*, are ubiquitous in today’s computer networks. The means whereby malware can infiltrate a network are manifold and range from exploiting of software vulnerabilities to tricking a user into executing malicious code. Monitoring and detection of all possible infection vectors is intractable in practice. Hence, we approach the problem of detecting malicious software at a later point when it initiates contact with its maintainer; a process referred to as “*phoning home*”. In particular, we introduce *Botzilla*, a method for detection of malware communication, which proceeds by repetitively recording network traffic of malware in a controlled environment and generating network signatures from invariant content patterns. Experiments conducted at a large university network demonstrate the ability of Botzilla to accurately identify malware communication in network traffic with very low false-positive rates.

## Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—*Security and protection*; I.5.1 [Pattern Recognition]: Models—*Statistical*

## Keywords

Malicious Software, Network Intrusion Detection

## 1. INTRODUCTION

Malicious software, so called *malware*, is predominantly used for profit rather than fun—this is the verdict of a number of recent studies and industry reviews [e.g., 2, 8, 25]. A crucial role in the underlying business models belongs to harvesting of confidential information such as passwords, credentials, or activity patterns from compromised hosts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC’10 March 22-26, 2010, Sierre, Switzerland.

Copyright 2010 ACM 978-1-60558-638-0/10/03 ...\$10.00.

Another possibility to capitalize on infection is given by illegal remote control of compromised hosts, for example in form of so called *botnets*. Both of these exploitation patterns *require* the existence of a communication channel between compromised systems and the remote attacker. Such communication, known as “*phoning home*”, may include contacting IRC servers, accessing HTTP services, downloading of files via FTP, or communication using proprietary protocols.

Despite a tremendous effort spent on detection and removal of malicious software on the way to its end destination, the success rate of target system infection remains alarmingly high. This is largely due to an enormous variability of malware, as well as an increasing deployment of advanced infection techniques such as drive-by vulnerabilities, typo-squatting, or social engineering. Even if the monitoring of incoming network traffic succeeds in sealing off a targeted system, a compromise may still take place via an alternative transport medium, for example, a USB device.

Consequently, it becomes increasingly important to identify the communication between attackers and their victims. Such detection brings about two benefits: one can easily identify and disinfect compromised computers, and learn about attackers’ infrastructure and potentially take countermeasures against it. The main difficulty of detecting malware communication stems from a versatility of deployed network protocols as well as from a frequent use of obfuscation techniques. Previous approaches to detection of malware communication roughly fall into two categories.

*Vertical correlation* techniques detect infected hosts using hand-crafted communication models [e.g., 3, 5]. *Horizontal correlation* techniques learn network patterns induced by infections of multiple systems [e.g., 4, 6, 21]. Both approaches have significant limitations: manually generated models lack the ability to detect novel and unknown patterns of malware communication, whereas horizontal correlation techniques rely on salient infection features across multiple hosts which are not necessary observable for all malware types, for example keyloggers.

In this paper, we introduce *Botzilla*, a generic method for detecting the “*phoning home*” of malicious software. The underlying idea of Botzilla is the well-established concept of automatic signature generation [e.g., 12, 16]. In contrast to previous work, Botzilla can *automatically* generate accurate models of malware communication even if the malware is observed only on a single infected host. Botzilla proceeds by capturing malware binaries in the wild and monitoring their network activity during repetitive execution of each sample in a controlled environment. Invariant content pat-

terns are extracted from the recorded network traces and assembled into concise signatures suitable for deployment in high-speed networks. The whole process—from capturing of malware to signature generation—is fully automatic, which significantly reduces the time lag between the sighting of a new malware and the availability of a corresponding signature. Moreover, by randomly changing parameters of the controlled environment during repetitive execution, such as the date of the system, our method is hardened against several evasion attacks [e.g., 17, 19].

Experiments conducted in a large university network demonstrate the ability of Botzilla to accurately identify malware communication with low false-positive rates. In an offline experiment our system detected 94.5% of recent malware classes with only a single false alarm in one million network flows. A further live application of Botzilla over a period of 4 days resulted in discovery of 219 true infections of malware in the university network. During this application, Botzilla attained a false-positive rate of 0.00004% (295 false alarms in a total of 840 million network flows), which is remarkably low given that no manual adaptation or refinement of generated signatures was performed.

The rest of this paper is organized as follows. Current techniques used in malware communication are discussed in Section 2. Our method Botzilla is introduced in Section 3 and its experimental evaluation is presented in Section 4. We provide a discussion of related work in Section 5. Concluding remarks are given in Section 6.

## 2. MALWARE COMMUNICATION

A prevalent feature of current malware is the existence of a communication channel from compromised hosts to the remote attacker. Such channel allows the attacker to retrieve information and issue commands that are carried out by infected machines. In contrast to classic Internet worms, for example *Code Red* [15] or *Slammer* [14], modern malware almost always employs such functionality. The remote control over infected machines enables an attacker to abuse compromised systems for financial profit, for instance to distribute spam messages or steal password and credential information. Thus, means for detection of malware communication are crucial to combat current threats on the Internet.

Unfortunately, the “phoning home” traffic of malware can take extremely heterogeneous forms and may include obfuscation, encryption, covert channels, and even steganography. Hence, accurate detection of malware communication in network traffic is, in general, hard if not impossible. Surprisingly, however, strong encryption and covert channel techniques are not widely deployed in current malware communication, and with rare exceptions, such as *Storm* [9] and *Nugache* [24], the majority of current malware employs clear-text protocols. The mechanisms used in such communication can be classified as shown in Figure 1.

The *communication direction* describes the mechanism that is used by the infected machines to receive commands from the attacker. On the one hand, this mechanism can be *push-based* if the attacker sends the command, e.g., to all bots within an IRC channel. On the other hand, the infected machines can periodically *pull* for new commands and send requests to the attacker. Orthogonal to the communication direction is the *communication architecture*, which describes the actual communication structure. This can be a *central-*

Com. architecture Communication direction	Central communication	Peer-to-peer communication
Push mechanism	IRC botnets	Flooding
Pull mechanism	HTTP botnets, Fast-flux service networks	Publish/subscribe-style communication (Storm Worm)

Figure 1: “Phoning home” mechanisms.

*ized* model in which the attacker uses one central server to which infected machines connect. Alternatively, the architecture can also be implemented as a *peer-to-peer* system. Figure 1 provides an example for each of the four classes of “phoning home” mechanisms. No malware that uses a push-based, peer-to-peer communication mechanism exists in the wild and thus we focus in this work on the remaining three communication classes.

## 3. DETECTING THE “PHONING HOME”

The majority of current malware implements communication protocols that exhibit invariant content patterns. For example, several bot networks retrieve instructions using the IRC protocol where the respective traffic is characterized by particular strings of channel and user names. Even encryption does not necessary rule out invariance, for instance as in the case of *Storm* and *Nugache*, whose encrypted communication features distinct byte patterns [see 24]. We exploit such invariance to automatically construct network signatures for malware communication. The resulting method Botzilla operates in three phases:

1. *Malware capturing.* In the first phase, recent malware binaries are collected from the network by means of honeypots, forensic analysis of security incidents, and other automated approaches (Section 3.1).
2. *Repetitive execution.* “Phoning home” is triggered and monitored by repetitive execution of each malware binary in a controlled environment (so called *sandnet*) with varying network and host settings (Section 3.2).
3. *Signature generation.* Using the monitored network traces for each binary and a pool of regular traffic, invariant patterns are extracted and refined to network signatures of malware communication (Section 3.3).

An overview of our approach is shown in Figure 2 which presents the repetitive execution in a sandnet and the signature generation process. In practice, all phases can be realized on a single network system such that the delay between a first appearance of a new malware binary and the deployment of a corresponding network signature can be limited to a few minutes.

### 3.1 Malware Capturing

The effective generation of network signatures for malware communication critically depends on proactive capturing of malware samples. The timely availability of new signatures ensures that a monitored network domain can be protected prior to a potential mass-infection. To this end, we consider several techniques for collecting malware binaries in the wild.

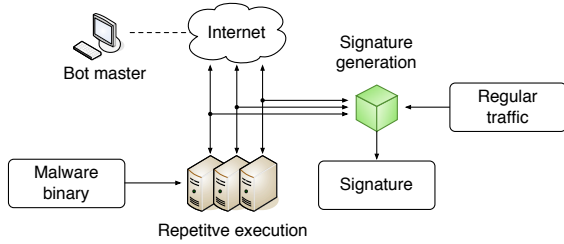


Figure 2: Schematic depiction of Botzilla.

First, we employ *network honeypots* (such as Nepenthes [1] and Amun) which emulate common vulnerabilities in network services and thereby collect malware propagating using corresponding attack vectors. To capture malicious binaries involved in client-side infections, we make use of *honeyclients*, a special type of honeypots designed to capture attacks targeting network clients [20, 27]. The third capturing strategy employs so called *spamtraps* which automatically collect recent spam using dedicated mailboxes. Binaries are then extracted from contained attachments to capture malware that uses e-mail as a propagation vector. Finally, samples identified using forensic analysis of security incidents can be manually contributed to our system.

### 3.2 Repetitive Execution

Once malware is captured from the network, the automatic monitoring of “phoning home” communication is triggered. Each collected malware binary is repetitively executed in a controlled environment where outgoing and incoming traffic is recorded over a period of several minutes. We denote this environment as a *sandnet*. To evoke different patterns of malware communication, the sandnet is altered during each repetition by automatically changing the date, the network environment, the operating system and the time of day. In our experiments each malware binary is executed at least 10 times using different IP addresses and variants of the operating system. This setting ensures that the recorded traffic comprises a large spectrum of communication covering several variations of “phoning home” messages. For example, the majority of malware transfers information regarding the operation system version during the first contact.

The execution in a sandnet may introduce artifacts into the network traffic such as local host and domain names. To limit the impact of possible artifacts on the signature generation, we apply a blacklist filter to the collected network data which replaces occurrences of predefined strings with random byte sequences. In our setting, we filter contents related to the network domain and the setup of the sandnet, such as the domain name and the time of day. While we can not generally rule out the existence of artifacts, the empirical evaluation presented in Section 4 demonstrates the ability of Botzilla to generalize from the particular environment of the considered sandnet. To limit propagation of the malware outside of the sandnet, we block access to certain network services. For example, we redirect all outgoing SMTP traffic to a network sink to avoid spam distribution. Moreover, we block communication targeting services with well-known security vulnerabilities, such as DCOM and SMB services.

### 3.3 Signature Generation

The traffic recorded in the sandnet enables us to infer typical and invariant contents in the malware communication which provides the basis for generation of network signatures. The automatic construction of signatures originates from the field of intrusion detection where corresponding methods have been widely studied [e.g., 10–12, 16, 23, 28].

The signature generation component of Botzilla extends the Bayesian technique originally proposed in Polygraph [16] to the setting of network signatures for malware communication. In comparison to more advanced approaches, such as Nemean [28], that incorporate session-level context, we focus on the contents of individual network flows as basis for signature generation. This setting provides a balance between an expressive representation and sufficient run-time performance of signature matching—a crucial requirement for the application of Botzilla in high-speed networks.

Before presenting this signature generation process in more detail, let us first define the notion of a signature. A *signature*  $S$  is defined as a tuple  $(T, \theta)$  where  $T$  is a set of strings associated with probabilities and  $\theta$  a threshold value. We denote the  $i$ -th string of a signature as a *token*  $t_i$  and refer to the corresponding probability as its *support*  $s_i \in (0, 1)$ . A signature  $S$  matches the payload  $x$  of a network flow if the support for all tokens contained in  $x$  exceeds the threshold value  $\theta$ . The use of tokens and their support values allows for automatic generation of different types of signatures, such as Boolean conjunctions and disjunctions [see 16].

Two data sets are required for automatic generation of signatures: the (malicious) network traffic monitored during multiple executions of a malware binary, denoted as  $X^+$ , and regular traffic recorded at the network site to protect, denoted as  $X^-$ . The network data in both sets is represented in form of reassembled network flows (TCP connections and UDP packets), where the payloads of incoming and outgoing packets are considered jointly. Using these data sets, signatures are inferred in two stages as depicted in Figure 3.

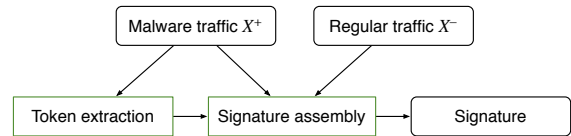


Figure 3: Signature generation process of Botzilla.

**Token extraction.** In the first stage of the signature generation, tokens are extracted from the malicious data  $X^+$ . In particular, every substring contained in at least  $d$  network flows with a minimum length of  $l$  is chosen as a potential token for the signature. By choosing the parameter  $d$  sufficiently high, say 80%, this procedure ensures that the extracted tokens reflect invariant patterns occurring in the majority of communication traces. In our implementation, the extraction is realized using a generalized suffix tree which enables linear time retrieval of arbitrary substrings [7]. The extracted tokens may still overlap with each other. Hence, we apply a constraint on the set of tokens: a token  $t_i$  is discarded, if it is substring of another token  $t_j$ , unless there are  $d$  further occurrences of  $t_i$  independent of  $t_j$ . That is, the token  $t_i$  is only kept if it occurs often enough outside of  $t_j$ .

The size of the original set of tokens is strongly decreased by this constraint and so are the redundancy and the length of the resulting signatures.

**Signature assembly.** Based on the extracted tokens, a signature with support values and a threshold is assembled. Initially, the set of tokens is refined using the regular network flows in  $X^-$ . For each token  $t_i$  two values are computed: the frequency  $f^+$  of  $t_i$  in the malicious communication and the frequency  $f^-$  of  $t_i$  in the pool of normal data. Clearly, tokens with  $f^- > f^+$  must be excluded from a signature as they occur more frequently in normal traffic than in malware communication. Moreover, we remove tokens that are predominant in normal traffic, that is, the ones with  $f^- > l$  for some reasonable limit  $l$ . The final set of tokens  $T$  is now obtained by associating the token  $t_i$  with  $s_i := f^+$ , its support in the set of malicious traffic  $X^+$ . The rigorous selection of tokens significantly reduces a false alarm rate in practice, however, for some malware binaries no signatures can be inferred as simply no suitable tokens exist that fit the stringent criteria above. It remains to select an appropriate threshold  $\theta$ . The threshold calibration process is performed on  $X^+$  and  $X^-$ , where for all possible threshold values detection and false-positive rates are determined. In our implementation, we choose the threshold inducing the best detection accuracy at the lowest possible false-positive rate. Using the same procedure, we also identify optimal configurations for the parameters  $l$  and  $d$  in token extraction. Finally,  $\theta$  and the support values are normalized for better comparability.

**Signature filtering.** The presented procedure assigns a signature to the network traffic of a particular malware binary. In practice, several similar binaries may be captured in a short time frame, which can lead to highly redundant network signatures. To address this problem, we devise a filtering scheme for removal of redundant signatures. We proceed by removing signatures that yield identical detection performance to other signatures on the set of malicious traffic  $X^-$  and regular traffic  $X^+$ . This simple approach performs well in practice, as demonstrated in Section 4.

## 4. EMPIRICAL EVALUATION

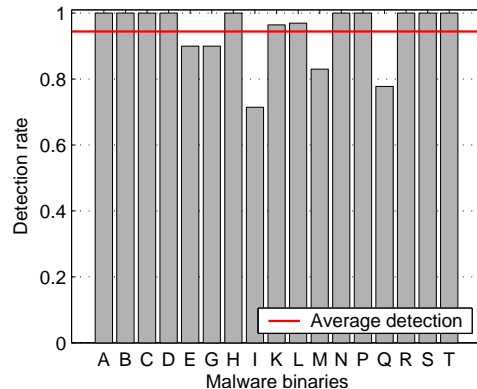
The two essential advantages of Botzilla over previous methods for detection of malware communication are automatic signature inference and high performance during deployment. We verify these features in large-scale experiments carried out on a data collected from a major university network. We first present the results of an offline experiment in which we evaluate the detection accuracy on 20 typical malware classes (Section 4.1). We then present results of a live deployment of Botzilla at a central gateway of the university network (Section 4.3).

### 4.1 Detection Performance

For the offline experiment, we consider a set of 20 typical malware classes listed in Table 1. The different classes cover a broad spectrum of “phoning home” activity, ranging from simple HTTP and IRC communication to proprietary and encrypted protocols. For each class, we capture a malware binary in the wild using techniques detailed in Section 3.1. We consider a sample of two million network flows (TCP connections and UDP packets) recorded over a period of 24 hours at the university network as a pool of normal traffic. To allow for efficient matching of signatures

Malware class	Category	Ports
A <i>Banload</i>	Trojan & Spyware	21
B <i>Hupigon</i>	Trojan & Downloader	80
C <i>Infostealer</i>	Trojan & Keylogger	80 ... 9222
D <i>Ircbot</i>	Botnet	80, 88, 6667
E <i>Koobface</i>	Worm	80, 88
F <i>Ozdok</i>	Trojan & Spambot	53, 80
G <i>Pinch</i>	Trojan & Keylogger	80
H <i>Rbot</i>	Botnet	80, 88, 6668
I <i>Rustock</i>	Backdoor & Spambot	80 ... 65520
J <i>Sality</i>	Virus & Backdoor	80
K <i>Sdbot</i>	Botnet	80, 88, 65146
L <i>Srizbi</i>	Trojan & Spambot	43 ... 65520
M <i>Storm</i>	Botnet	1 ... 63733
N <i>Swizzor</i>	Trojan & Spyware	80, 88
O <i>Tr.Downloader</i>	Trojan & Downloader	80
P <i>Tr.Spy</i>	Trojan & Spyware	80
Q <i>Vanbot</i>	Botnet	80 ... 8080
R <i>Virut</i>	Virus & Backdoor	53 ... 9011
S <i>Zeus</i>	Trojan & Keylogger	80, 88
T <i>Zlob</i>	Trojan & Backdoor	80, 88

**Table 1: Malware classes for empirical evaluation. As the true “phoning home” mechanism is unknown in most cases, all monitored ports are listed.**



**Figure 4: Detection performance of generated signatures. The detection rates have been obtained with one false positive in 1,000,000 benign network flows.**

in high-bandwidth traffic, we limit the length of payloads to a maximum of 256 bytes. This restriction, however, is not inherent to Botzilla, as our signature generation can operate on payloads of arbitrary length.

Each captured malware binary is executed 10 times in the sandnet using different network and host settings, and the resulting network communication is automatically recorded. To ensure a sound evaluation, we split the recorded malicious and regular traffic into two distinct partitions, where the first partition is used for signature generation and threshold computation while the second is only applied for measuring detection and false-positive rates.

Results for the offline application of Botzilla are presented in Figure 4, which shows the detection rate for 17 of the 20 malware classes. On average, 94.5% of the “phoning home” activity is correctly identified by the generated network signatures; a perfect detection is attained for 10 classes. However, for 3 out of 20 malware classes (F, J, O), no signatures have been generated by Botzilla due to the lack of suitable

tokens, hence no detection rate for the respective malware class is shown. While this result may be seen as a weakness of our approach, the rigorous selection of tokens is crucial for enforcing low false-positive rates. The false-positive rate achieved in this experiment is 0.0001%, as only a single network flow out of the 1,000,000 benign flows is reported as a false alarm. Similar results are obtained with a filtered set of signatures. The filtering slightly decreases the detection rate from 94.5% to 92.7%, whereas the false-positive rate remains constant at 0.0001%. The number of generated signatures is reduced by 23.5% from 17 to 13, which confirms the ability of the filtering mechanism to effectively prune redundant signatures. Keeping the signature base small is crucial for maintaining the performance of signature matching.

## 4.2 Generated Signatures

The presented detection performance demonstrates the ability of Botzilla to accurately identify several malware classes. To gain further insights into this process, we study some generated signatures in more detail. In particular, we examine the tokens and support values of three signatures generated for the malware classes *Hupigon*, *Banload* and *Storm* listed in Table 1.

---

```
tokens :
  "GET a2965583/ip.txt HTTP/1.0\n\r
  User-Agent: MYURL[...]" : 1.000
threshold : 1.00
```

---

**Figure 5: Signature for the malware *Hupigon*. For presentation longer tokens have been truncated.**

As the first example, Figure 5 shows the signature generated for the malware class *Hupigon*. The signature reflects the behavior of a typical HTTP bot and comprises a single HTTP request, which is used to inform the attacker of a newly infected host. The accurate selection of tokens in this example yields a very concise signature, comprising the URI “a2965583” and the unusual user-agent “MYURL” in a single token string.

---

```
tokens :
  "SER asaasa510\n\rPASS 3330881\n\rTYPE..." : 0.500
  "220 ProFTPD 1.2.9 Server (ProFTPD) [1..." : 0.500
threshold : 1.00
```

---

**Figure 6: Signature for the malware *Banload*. For presentation longer tokens have been truncated.**

Figure 6 presents the signature generated for the malware class *Banload*. The signature contains two tokens corresponding to malware communication using the FTP protocol. The first token covers the initial FTP request of *Banload* to login into its control server with the username “asaasa510” and the password “3330881”, while the second token corresponds to the respective response of the server. The support values are equally distributed and realize a Boolean conjunction of tokens, which is only triggered if both tokens match.

The third example of a generated signature is presented in Figure 7 for the malware *Storm*. In contrast to the other examples, *Storm* uses a proprietary communication protocol

involving an encryption scheme [see 9, 24]. Although such encryption should generally impede the extraction of invariant patterns, six discriminative strings have been extracted from the network traffic. As a result, *Storm* is identified with over 80% detection rate in Figure 4 *despite the encryption of communication*. We credit the discovered tokens to invariants in the communication which are not truly concealed by a simple encryption scheme, yet we were unable to recover the unencrypted contents.

---

```
tokens :
  ef bf bd 50 00 ef bf bd 0a : 0.920
  ef bf bd ef bf bd ef bf bd ef bf bd ef ... : 0.545
  ef bf bd 0d ef bf bd 0d ef bf bd 0d ef ... : 0.339
  40 ef bf bd 3c ef bf bd 50 00 ef bf bd 0c : 1.000
  40 ef bf bd 3c ef bf bd 50 00 ef bf bd 1b : 0.679
  40 ef bf bd 3c ef bf bd 50 00 ef bf bd ... : 0.581
threshold : 1.00
```

---

**Figure 7: Signature for the malware *Storm*. The tokens are presented using sequences of hexadecimal numbers. For presentation longer tokens have been truncated.**

## 4.3 Live Application

We conclude our evaluation with a live application at a central gateway of a large university network. The application at a high-speed link requires certain prerequisite and technical issues to be addressed. First, we incorporate our approach into the open-source flow monitor Vermont [13] which is able to process raw packets up to 1 Gbit/s. We again restrict the acquisition of payload data to the first 256 bytes of flows, whereby stream reassembly is kept to a minimum. Thorough reassembly is problematic in high-speed network environments, hence our goal is to perform fast aggregation with low memory impact. While the payload aggregation for UDP packets is straightforward, the aggregation of TCP packets is implemented using a lightweight stream reassembly where sequence numbers are matched, but sophisticated checks are omitted for the sake of performance. Moreover, uni-directional flows are aggregated into bi-directional flows so that signatures can be jointly matched on the respective payloads. Finally, to allow for efficient signature matching we employ an implementation of a keyword tree [7] similar to the Aho-Corasick algorithm regularly applied in intrusion detection systems [e.g. 18, 22].

For the live application we consider a second set of 43 malware binaries, which have been captured during a day of May 2009 and correspond to currently active malware strains. For each binary, a network signature is generated as described in Section 4.1 and added to the signature matcher of Vermont. The final system is then deployed to monitor the Internet uplink of a university network, which consists of over 50,000 hosts and features a high variety of different traffic types. During the experiment, Botzilla and the underlying Vermont are running on standard hardware (Intel Core 2 Quad CPU running at 2.8 GHz). Moreover, all monitored network traces are anonymized to comply with privacy regulations.

Table 2 summarizes results obtained during a 4 day deployment of Botzilla. A total of 514 alerts was reported

<i>True positives during 4 days</i>	
167	Infections with spyware <i>HotLog.ru</i>
18	Downloads of malware binaries (zonetech.info)
29	Communication with Russian Business Networks
5	Infections with <i>Tr.Downloader</i>
<i>Presumed false positives during 4 days</i>	
75	False positives with token “rygames”
31	False positives related to HTTP requests
29	False positives related to IRC connections
160	Miscellaneous false positives

**Table 2: Alerts reported by Botzilla during a 4 day application period.**

of which 219 correspond to real infections of malware in the university network. In particular, 167 instances of spyware and 18 downloads of malware binaries were detected in the network traffic. Furthermore, the “phoning home” of malware related to so-called Russian Business Networks was correctly tracked in 29 cases. The other 295 alarms are false positives and were induced by signatures that erroneously flag particular HTTP and IRC traffic as malware communication. Overall, Botzilla yields a false-positive rate of 0.00004%, corresponding to 295 false alarms in 840 million flows, which is still remarkably low given that no manual refinement has been performed. For a comparative evaluation, we also processed the network traffic with BotHunter [5]. As Botzilla is trained to detect the specific 43 malware binaries, absolute detection rates of the two system are not directly comparable. Thus, we restrict our comparison to hosts that triggered true-positive matches using Botzilla. Of these matches, Bothunter flagged only 47% as malicious. This result demonstrates that the signatures generated by Botzilla are *by far superior* to hand-crafted signatures for detection of recent malware communication and provide a valuable alternative to current horizontal and vertical correlation techniques.

Finally, we study the run-time performance of the underlying Vermont infrastructure using the generated network signatures. In particular, we measure the run-time for processing flows with all 43 generated signatures, where the experiment is repeated for 30 times to remove outliers in the measurements. All flows are previously aggregated for the experiment, such that only the run-time performance of signature matching is measured. Vermont is able to process over 36,000 flows per second. Using statistics from our network, this figure relates to 835,000 packets per second and 4 Gbit/s link speed in average. Although we have not considered larger sets of signatures, our results demonstrate the excellent performance of Vermont, which can be easily improved using recent acceleration techniques for signature matching [26].

## 5. DISCUSSION AND RELATED WORK

Detection of infected machines within a network is highly important and, hence, it has attracted a significant amount of research in the past. Closely related to our work are botnet detection approaches like BotHunter [5], BotSniffer [6], BotMiner [4] and TAMD [21]. These tools try to find either horizontal or vertical correlations in network communication. A horizontal correlation implies that several hosts behave similarly at the network level: this indicates infected

machines under a common control infrastructure which respond to a given command. While this is a generic approach to detect compromised hosts, it may fail if only a small number of machines is infected or if the attacker sends different commands to machines within a single network. Furthermore, these approaches need to study communication for a longer time to find correlations, whereas our approach tries to detect the actual “phoning home” mechanism, which provides the advantage that we can detect infected hosts *early* and prior to mass-infections.

The second approach, namely vertical correlation, tries to detect individual machines that behave like infected machines, for example by detecting typical network signatures of botnet communication. We follow this line of research and show how such signatures can be generated from monitored malware traffic. In contrast to previous work in this area, we *automatically generate* these signatures and do not rely on human intervention. Furthermore, our approach does not depend on examining full network payloads. In our experiments, we limit the analysis to the first bytes of network flows and still attain sufficient detection accuracy. This allows for scalable detection that can monitor an order of magnitude more machines for infections in comparison to previous work.

Clearly, the simplest technique to hide communication from automatic signature generation is the encryption of contents. While Botzilla can not overcome this problem in general, two issues related to practical application are noteworthy. First, the absence of invariant patterns to be caused by encryption would result in empty signatures, which would allow to at least identify encrypted communication. The mere presence of encrypted communication may already be indicative of an infection and reveal information about a remote attacker (e.g., at a TCP/IP level). Second, simple or poorly implemented encryption methods may not necessarily eliminate, but rather only obfuscate invariance. In such cases, malware communication can still be detected, for example as in the case of *Storm*.

Methods for automatic signature generation are known to suffer from several evasion techniques [17, 19]. Botzilla is less affected by such threats, as its application generally differs from the regular intrusion detection scenarios. The network traces used for signature generation are obtained from repeated executions of malware in a controlled environment. By randomly changing parameters of this environment, such as the time, it is particularly hard for an attacker to decide when to inject fake invariants into the communication, as the environment and time is controlled by signature generating process. As a result, the attacker is limited to indiscriminate evasion techniques such as inclusion of random chaff when contacting the remote control host. Effectiveness of such attacks has been shown to be significantly lower than that of targeted attacks. Since our malware execution is not carried in a virtualized environment, detection of virtualization is not a promising attack option either.

## 6. CONCLUSIONS

In this contribution, we have presented Botzilla, a method for automatic detection of typical communication patterns used in malicious software. The method provides a simple yet practical tool for automatic network-level analysis and protection. In particular, the use of signatures allows for efficient realizations using common network monitoring

and intrusion detection techniques, such that Botzilla can be readily deployed in high-speed networks. Overall, the automatic generation of signatures for novel malware provides a timely mechanism for keeping abreast of the increasing threat posed by malicious software

Botzilla has been evaluated in a large university network, where it attains a detection rate over 94.5% with a low amount of false alarms. In a practical applications over several days, Botzilla automatically identified various instances of malware infections in a university network, thus providing a valuable tool for security administrators at the network site. While the approach underlying Botzilla addresses the detection of communication as employed by today's malware, our future research will consider "phoning home" techniques likely to occur in next-generation malware, such as covert channel and steganographic remote control.

## References

- [1] P. Bächer, M. Kötter, T. Holz, M. Dornseif, and F. C. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Recent Advances in Intrusion Detection (RAID)*, pages 165–184, 2006.
- [2] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry Into the Nature and Causes of the Wealth of Internet Miscreants. In *Proc. of Conference on Computer and Communications Security (CCS)*, pages 375–388, 2007.
- [3] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Workshop on Hot Topics in Understanding Botnets*, 2007.
- [4] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of USENIX Security Symposium*, 2008.
- [5] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. of USENIX Security Symposium*, 2007.
- [6] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2008.
- [7] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.
- [8] T. Holz, M. Engelberth, and F. Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. Technical Report TR-2008-006, University of Mannheim, Dec. 2008.
- [9] 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 *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.
- [10] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. of USENIX Security Symposium*, 2004.
- [11] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *Proc. of Workshop on Hot Topics in Networks*, 2003.
- [12] Z. Li, M. Sandhi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Proc. of IEEE Symposium on Security and Privacy*, pages 32–47, 2006.
- [13] T. Limmer and F. Dressler. Seamless Dynamic Reconfiguration of Flow Meters: Requirements and Solutions. In *16. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2009)*, pages 179–190, Kassel, Germany, March 2009. Springer.
- [14] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford-Chen, and N. Weaver. Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.
- [15] D. Moore and C. Shannon. Code-Red: A case study on the spread and victims of an Internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, pages 273–284, 2002.
- [16] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–132, 2005.
- [17] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection (RAID)*, pages 81–105, 2006.
- [18] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Elsevier Computer Networks*, 31(23-24):2435–2463, December 1999.
- [19] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proc. of IEEE Symposium on Security and Privacy*, pages 17–31, 2006.
- [20] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *Proc. of USENIX Security Symposium*, 2008.
- [21] M. Reiter and T. Yen. Traffic aggregation for malware detection. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2008.
- [22] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. of USENIX Large Installation System Administration Conference LISA*, pages 229–238, 1999.
- [23] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2004.
- [24] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. *USENIX ;login.*, 32(6):18–27, 2007.
- [25] Symantec. Global Internet Security Threat Report, April 2008. Trends for July – December 07.
- [26] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. Markatos, and S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In *Recent Advances in Intrusion Detection (RAID)*, 2008.
- [27] Y.-M. Wang, D. Beck, C. Verbowski, S. Chen, S. King, X. Jiang, and R. Rouseff. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2006.
- [28] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *Proc. of USENIX Security Symposium*, pages 7–17, 2005.