SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2015; **8:**952–969
Published online 3 July 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1052

RESEARCH ARTICLE

BotCatch: leveraging signature and behavior for bot detection

Yuede Ji[1,2], Qiang Li[1,2]*, Yukun He[1,2] and Dong Guo[1,2]

[1] College of Computer Science and Technology, Jilin University, Changchun, China
[2] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun, China

ABSTRACT

The goal of bot detection is to discover malicious bot processes by signature comparison or behavior analysis. Existing approaches have several drawbacks, such as requiring a lot of prior knowledge, low detection accuracy, and high false alarm rate. In this paper, we propose a multi-feedback approach, BotCatch, to detect bots effectively and efficiently on a host by leverage of a combination of signature and behavior. First, BotCatch assigns suspicious files to signature-analysis and behavior-analysis modules, which generate each detection result. Second, BotCatch correlates signature and behavior results to generate the final detection result through correlation engine. Third, BotCatch feeds back signature, behavior, and correlation results to dynamically adjust detecting modules through multi-feedback engine. We evaluated the performance of BotCatch with 636 bot and 150 benign samples. Our results indicate that BotCatch achieves an accuracy of 97.1% and an F-measure value of 0.982 simultaneously, which is better than existing approaches without feedbacks. BotCatch, due to the multi-feedback mechanism, has the ability to gradually get more robust and accurate as the number of samples increases. The final stage even reaches an accuracy of 98.5% and F-measure value of 0.991.

KEYWORDS
botnet; bot detection; feedback; correlation

***Correspondence**
Qiang Li, College of Computer Science and Technology, Jilin University, Changchun, China.
E-mail: li_qiang@jlu.edu.cn

1. INTRODUCTION

A bot is a host that has been compromised by malware under the control of a botmaster through command and control (C&C) channel (i.e., Internet Relay Chat [IRC], Hypertext Transfer Protocol [HTTP], and P2P). When many bots work together, they form a botnet. Botnets have become one of the most serious threats to Internet security [1]. The botmaster can utilize botnets to conduct various cyber crimes, such as spreading malware, conducting distributed denial-of-service (DDoS) attacks, spamming, and phishing. Recently, botnets have become the major platform for most online criminal activities [2].

A lot of efforts have been conducted on how to detect bots and botnets. Botnets can be detected on hosts or in networks. Network-based detection measures mainly observe data traffic in the network and look for suspicious communications that may be from bots or C&C servers [3]. Network-based approaches primarily target bot hosts and botmaster hosts, while host-based approaches mainly target bot processes and malicious files on infected hosts. This research mainly focuses on detecting bots on hosts.

Host-based bot detection approaches are usually either signature-based or behavior-based [3]. Signature approaches mainly extract the feature information of suspicious programs and match them against a signature database, such as Rishi [4]. Behavior detection approaches monitor the abnormal behavior of hosts to determine whether any one host is infected. For example, the approaches might involve checking the status of the operating system, the running status of suspicious programs, access to suspicious registries or files or system call sequences, and so on. [5–7].

Signature-based bot detection approaches are low-risk and result in few positives and low overhead. However, they are unable to detect unknown bots or overcome obfuscation techniques. They also require a lot of prior knowledge. Behavior bot detection approaches can deal with unknown bots and some obfuscation techniques. However, they are high risk, provide only low detection accuracy, and incur high overhead.

Bots and botnets are evolving to become more and more difficult to detect. Their evolution tendencies can be divided into three categories: (i) More hidden mechanisms. Existing bots utilize advanced hidden techniques to evade detection, such as dividing one process into several [8], using covert channels to communicate [9]. (ii) More obfuscation. Existing bots are acting more and more like benign software. They also use other techniques, such as metamorphism mechanism that changes the internal structure of software while maintaining its functionality [10,11]. (iii) Novel C&C channels. Bots used to utilize IRC, HTTP, and P2P as C&C channels. However, new bots are using some channels that are difficult to detect, such as online social network [12,13] and the Tor network [14,15].

With the evolution of these new botnet behaviors, pure signature-based or behavior-based detection approaches are less effective and efficient. Combining signature-based and behavior-based bot detection can maintain some advantages and overcome some critical disadvantages, such as the ineffectiveness of signature-based detection against unknown bots and obfuscation techniques, and the high risk and low detection accuracy of behavior-based detection. However, many challenges must be overcome in order to combine the two detection methods, such as how to assign weight to each method [16]. Although there are many techniques for combining scores (linear discriminant analysis, quadratic discriminant analysis, machine learning, etc.), they need to be trained on large samples labeled by a human oracle. The accuracy and coverage of this training sample are essential to the performance of this kind of correlation approaches.

To solve these problems, we propose a multi-feedback approach, BotCatch, to detect bots effectively and efficiently on a host by leverage of a combination of signature and behavior. BotCatch includes five modules: an analysis engine, a signature-analysis module, a behavior-analysis module, a correlation engine, and a multi-feedback module. The analysis engine assigns the suspicious file to either the signature or behavior-analysis module. These modules analyze the file and generate detection results for the correlation engine. Then, correlation engine correlates signature and behavior detection results to generate the final detection result. The multi-feedback module uses the signature, behavior, and correlation results to dynamically adjust BotCatch. It optimizes the signature-analysis module by maintaining the signature database. It optimizes the behavior-analysis module by maintaining the sample set and guiding the module's learning procedure. It optimizes the correlation engine by modifying the parameters. Our evaluation results show the following: (i) The correlation algorithm in BotCatch is efficient and effective in combining signature and behavior detection results. (ii) The multi-feedback mechanism makes BotCatch adaptive to samples and gradually becomes more robust and accurate. (iii) Other correlation algorithms, such as those of support vector machine (SVM) models, are also effective; however, our correlation algorithm with its multi-feedback mechanism provides better detection results.

Our work makes the following contributions:

(1) We propose an end-host approach, BotCatch, to detect bots using a combination of behavior-based and signature-based bot detection. By combining signature-based and behavior-based detection, our approach maintains their advantages and overcome some of their critical disadvantages. BotCatch is composed of five modules: an analysis engine, a signature-analysis module, a behavior-analysis module, a correlation engine, and a multi-feedback module.
(2) We propose a multi-feedback mechanism that takes signature, behavior, and correlation results to dynamically optimize BotCatch. It optimizes the signature-analysis module by maintaining the signature database. It optimizes the behavior-analysis module by maintaining the sample set and guiding its learning procedure. It optimizes the correlation engine by modifying the parameters.
(3) We evaluated the performance of BotCatch with 636 bot and 150 benign samples. Our results indicate that BotCatch achieves an accuracy of 97.1% and F-measure value of 0.982, which is better than existing works without feedbacks. More importantly, as the number of samples increases, BotCatch becomes more robust and accurate. BotCatch even reaches an accuracy of 98.5% and F-measure value of 0.991 at the final stage.

The paper is organized as follows. Section 2 compares the proposed approach to previous approaches. Section 3 presents a system overview, including system architecture and methodology. Section 4 presents the multi-feedback mechanism. Section 5 presents experiment details and results. Section 6 presents discussions. Section 7 summarizes this paper.

2. RELATED WORK

2.1. Signature-based bot detection

Signature-based detection approaches mainly extract feature information about suspicious programs and match that information to a knowledge base of existing bots [4,17]. Rishi is a signature-based IRC botnet detection system that detects IRC bots using well-known IRC bot nickname patterns as signatures. Rishi mainly relies on the monitoring of passive network traffic for unusual or suspicious IRC nicknames and IRC servers, as well as uncommon server ports. It employs n-gram analysis and a scoring system to detect bots that use uncommon communication channels, and these channels cannot be detected by classical intrusion detection systems. Rishi is able to automatically generate warning emails to report infected machines to an administrator. The disadvantage of this approach is that it cannot detect encrypted communication or non-IRC bots.

2.2. Behavior-based bot detection

Many behavior-based bot detection approaches have been proposed [5,6,18–21]. Seungwon *et al*. proposed EFFORT [7], which includes five modules: a correlation engine and modules, analyzing human-process-network correlation, process reputation, the exposure of system resources, and the trading of network information. The correlation engine correlates the detection results of the other modules to generate the final detection result. Zeng proposed a host-level and network-level information correlation approach for detecting bots [22]. They used process monitor to monitor information on the host, including the registry, file system, and network stack. The suspicion level generator uses the extracted feature vector to generate the suspicion level. The router passes the collected Netflow data for each time window to the flow analyzer. Then, flow analyzer analyzes the Netflow data, extracts the feature vector, and passes the feature vector to the cluster analyzer. The cluster analyzer clusters the hosts in the local area network based on the network feature vector of each time window and the preprocessed information of host distance and then passes the results to the correlation engine. By sending requests to all hosts in each cluster, the correlation engine combines host information with network information to calculate the final detection results and determine whether a host has been infected.

2.3. Combined signature-based and behavior-based bot detection

Ammar *et al*. proposed a system for combining signature-based and behavior-based techniques using an application programming interface (API) graph system [23]. There are three procedures in their framework—preprocessing, graph construction, and graph matching. The preprocessing procedure executes the Portable Executable (PE) file and collects the API call after unpacking. The graph construction procedure constructs the call graph based on the API call and operating system resources and then decreases the constructed API graph. The graph matching procedure matches the graph with the API call graph database. This framework combines behavior and signature information, but it is only an ideal framework without further implementation or experiments. Guo *et al*. proposed a novel malware detection framework based on binary translation, called HERO [24]. The framework exploits static and dynamic binary translation features to detect a broad spectrum of malware and prevent it from being executed. They first use an analyzer based on a static binary translator to analyze the binary file. If the analyzer cannot complete the analysis, they use an analyzer based on a dynamic binary translator to analyze the binary file. Their work still faces the problems of low detection accuracy and a high false alarm rate. Hsiao *et al*. proposed an approach that combines dynamic passive analysis and active fingerprinting for bot detection in virtualized environments [16]. A passive detection agent on a virtual machine monitors its host for profiles of bot behavior and checks monitored behavior with behavior on other hosts. The active detection agent sends a specific stimulus to a host and examines if the expected behavior is triggered. However, the active agent needs to know a lot of specific bot commands.

Unlike this earlier work, our work leverages multi-feedback mechanisms and combines signature-based and behavior-based bot detection. In our prototype, the signature-analysis module uses the signature databases of antivirus tools to compensate for the weakness of needing to build a large signature database. The behavior-analysis module executes the suspicious file in an isolated environment and monitors its host behaviors, including in the registry, file system, and network. The correlation engine uses a correlation algorithm and dynamically updates the weight factors to determine whether files are malicious or benign. The multi-feedback mechanism dynamically optimizes the signature-analysis module, behavior-analysis module, and correlation engine to make the whole system more accurate.

3. SYSTEM OVERVIEW

In this section, we present an overview of BotCatch, including system architecture and methodology.

3.1. System architecture

BotCatch primarily consists of five modules: an analysis engine, a signature-analysis module, a behavior-analysis module, a correlation engine, and a multi-feedback module. A system diagram is shown in Figure 1.

The analysis engine is the entrance to BotCatch, and it provides the submission interface. When a suspicious file is submitted to our approach, the analysis engine assigns it to the signature-analysis and behavior-analysis modules separately.

The signature-analysis engine in this module queries the signature detection results using the well-known network

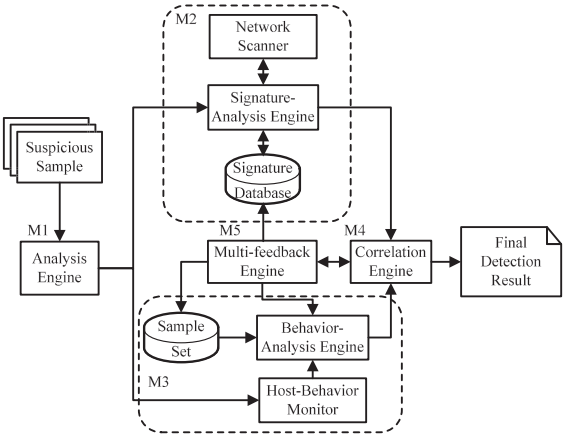

Figure 1. Architecture of BotCatch.

DOI: 10.1002/sec

scanner VirusTotal [25] and our signature database. The network scanner returns the detection results of up to 47 different antivirus tools and scan engines. Our signature database stores the signature of suspicious samples as aggregated by the multi-feedback module. The signature-analysis engine generates the final signature detection result and delivers it to the correlation engine.

In the behavior-analysis module, the host behavior monitor executes the suspicious sample in an isolated environment and monitors its host behaviors, including the registry, file system, and network. Then, it delivers the host behaviors to the behavior-analysis engine. The behavior-analysis engine generates the final behavior detection result using a machine learning algorithm and delivers it to the correlation engine.

The correlation engine finally generates a detection result between 0 and 1. If the detection result is greater than a threshold, it will be detected as malicious. Otherwise, it will be detected as benign. If the detection result is close to 1, it will be regarded as a high detection value. Otherwise, if it is close to 0, it will be regarded as a low value.

The multi-feedback module takes signature, behavior, and correlation results to dynamically optimize BotCatch. The multi-feedback module optimizes the signature-analysis module by maintaining the signature database and optimizes the behavior-analysis module by maintaining the sample set and guiding the module's learning procedure. The multi-feedback module also optimizes the correlation engine by modifying the parameters.

3.2. Methodology

In this section, we present the methodology of our approach.

3.2.1. M1: analysis engine.

The analysis engine provides two functions: a submission interface and task assignment. It provides three different submission methods: Web utility, submission utility, and Python functions. The analysis engine provides a basic Web utility that users can use to submit samples. The submission utility is a command-line utility that facilitates sample submission. The Python functions directly use the interface of the database to add samples, which is the most efficient method. The analysis engine also manages task assignment. If the signature-analysis and behavior-analysis modules are in a waiting state, the sample will be directly assigned to them. If they are analyzing another sample, the new sample is stored in our database. When they finish their analysis, the analysis engine assigns new samples to them in order by submission time.

3.2.2. M2: signature-analysis module.

The signature-analysis module is composed of a network scanner, the signature-analysis engine, and the signature database. Given a new file, the signature-analysis engine first accesses the network scanner for the antivirus signature of the analyzed file. If a signature is found, the network scanner returns the result to the signature-analysis engine. Then, it accesses the signature database to check whether this file has been flagged as malicious. The signature-analysis engine generates the final signature-analysis result based on the network scanner and signature database.

3.2.2.1. Network scanner. Most signature-based bot detection approaches first extract signatures of known bots to build a large signature database [3]. When detecting a suspicious file, they generate its signature and compare the signature to those in the database. We lack a large-scale collection of bot samples, and even if we have enough, they may not include every known sample. Antivirus scanners are malware detectors, which attempt to identify malware using signatures and other heuristics techniques [23].

Taking the preceding text into consideration, we can use the knowledge databases of antivirus scanners to generate signature detection results. The network scanner accesses the detection results of many famous antivirus engines. They are deployed in different countries and regions, and some bot samples may be specific to certain regions. Thus, we use the ratio of positives with the total number to denote the suspiciousness value of the signature. Suppose d antivirus engines detect it as malicious and the total number of antivirus engines is t. Then, the signature-analysis result will be d/t. We use s_{net} to denote the network scanner detection value and $s_{net} = d/t$.

3.2.2.2. Signature-analysis engine. The signature-analysis engine first accesses the network scanner to obtain network scanner detection value. Then, it accesses the local signature database. This database only stores the signatures of the samples added by the multi-feedback module. The multi-feedback engine adds two special kinds of samples. The first kind has a high correlation result but a low signature result. These may be bot variants or novel bots. The second kind has a low correlation result but a high signature result. That means that the samples may have similar signatures as malicious samples but have not exhibited enough malicious behaviors. The multi-feedback engine extracts the signatures of these two types of samples and adds them in the local signature database. We flag the first type as malicious and the second as benign. The multi-feedback mechanism is described in more detail in Section 4.

The local signature database returns a result of 0 or 1, and we use s_{local} to denote it. The final signature-analysis result is shown in Equation (1). If s is greater than 1, then s is set to 1. Then, the signature-analysis engine delivers the signature-analysis result to the correlation engine.

$$s = s_{net} + s_{local}, s \in [0, 1] \quad (1)$$

3.2.3. M3: behavior-analysis module.

In the behavior-analysis module, the host behavior monitor executes the suspicious file in an isolated environment and monitors its host behaviors, including the

registry, file system, and network. Then, it delivers the host behaviors to the behavior-analysis engine. The behavior-analysis engine generates the final behavior detection result using a machine learning algorithm and delivers it to the correlation engine.

3.2.3.1. Host behavior monitor. Before deploying host behavior monitors, we needed to decide which behavior features to capture. We analyzed the behaviors of existing bots and observed that they share certain behaviors that are different from benign programs. As Silva *et al*. summarized, bots have five phases in their life cycle: initial injection, secondary injection, connection or rally, malicious activities, and maintenance and upgrading [3]. We classified these behaviors into three categories: registry, file system, and network.

We present the execution procedure of a typical bot as an example. A bot first creates an exe or dll file in the system directory and registers an autorun key in the registry to make itself run automatically. It also injects its binary file into other processes to hide itself. Then, it opens one or more ports to establish connections with C&C server. Finally, the botmaster controls the bot to perform malicious activities, such as information theft, DDoS attacks, and spamming. The behavior features that we present can cover these typical activities. We should note that a single behavior such as those mentioned earlier may not be malicious, but a combination of these behaviors indicates a high possibility of malicious activity. So our behavior-analysis engine uses all features of behavior to make precise decisions.

To facilitate further analysis, the host behaviors of each suspicious file are transformed into a uniform format known as a behavior vector. Each behavior vector consists of 12 behavior features, as shown in Table I. The first three features are registry features, the next four are file features, and the last five are network features. Each vector is represented in this way: <1 2 3 4 5 6 7 8 9 10 11 12>, from behavior 1 to 12. As mentioned earlier, these features are intrinsic to bot processes.

Table I. Behavior feature vector.

Index	Feature description
1	Creation or modification of autorun key in registry
2	Creation or modification of process injection key in registry
3	Creation or modification of other critical registry keys
4	DLL Creation in the system directory
5	EXE Creation in the system directory
6	Modification of files in the system directory
7	Creation of other files in the system directory
8	Number of ports opened
9	Number of suspicious ports
10	Number of unique IPs contacted
11	Number of suspicious IPs
12	Number of unique domains queried

IPs, Internet protocols.

As an example, consider the Zeus bot [26]. Zeus mainly performs the following behaviors:

(1) Delete older versions: The Zeus bot searches for any existing copies of previous Zeus infection files (*sdra64.exe* files) and erases them. This procedure usually takes place when the Zeus binary file is updated with a newer version.
(2) Create a local copy: The Zeus bot makes an exact copy of itself and saves it to *C:/Windows/System32/sdra64.exe*. In order to hide itself, Zeus modifies its modification, access, and creation times using information from *Ntdll.dll*. It also sets *sdra64.exe* file attributes to "system" and "hidden."
(3) Set itself to run automatically using Autorun: The Zeus bot appends the path *C:/Windows/System32/sdra64.exe* to the registry key *HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft-/WindowsNT/CurrentVersion/Winlogon/Userinit*. This entry enables the Zeus bot to start automatically.
(4) Inject a process : The Zeus bot injects the binary file into the virtual memory of the *winlogon.exe* process and passes control to this process by creating a new user thread.
(5) Establish C&C connection: The Zeus bot injects itself into another process, *svchost.exe*. This process establishes a connection with the C&C server. In this way, Zeus can receive and execute commands, such as to steal information, launch DDos attacks, and send spam.

By consulting Table I, we can see that when the Zeus bot deletes older versions of itself, it exhibits behavior 6. When it creates a local copy, it exhibits behaviors 5 and 6. Using autorun is an example of behavior 1, injecting a process is an example of behavior 2, and establishing a C&C connection is an example of behaviors 8 and 10. Other malicious behaviors besides these can also be captured by these behavior features.

3.2.3.2. Host behavior analysis. Given the feature vector, the behavior-analysis engine uses machine learning algorithms to generate a behavior detection value b. It learns from benign and bot feature vectors to predict unlabeled behavior vectors. Much previous research has focused on the binary (malicious or benign) classification, and the results are likely to be inaccurate, because of the learning procedure [22]. In order to improve the learning model, we calibrate the distance score to a posterior classification probability indicating how likely it is that a test feature vector belongs to a particular class [27].

$$Pr(y = 1|x) \approx P_{A,B}(f) \equiv \frac{1}{1 + exp(Af + B)}, where f = f(x) \quad (2)$$

DOI: 10.1002/sec

Equation (2) presents the calculation formula of posterior classification probability. In order to calculate it, we need to have A, B, and f. We can have A, B through the training dataset. $f(x)$ is the decision value of the sample vector; we can obtain it from the prediction procedure. From Equation (2), we can conclude that the posterior classification probability is a value between 0 and 1. We use b to represent the behavior detection value.

The sample set in the behavior-analysis module stores the training dataset, which is updated by the multi-feedback module. We will present more details in Section 4.

3.2.4. M4: correlation engine.

There are many techniques for combining scores (linear discriminant analysis, quadratic discriminant analysis, machine learning, etc.). However, they need to be trained on large samples labeled by a human oracle. The coverage and accuracy of this training sample are essential to the performance of this kind of correlation approaches. Different from them, our correlation engine makes full use of signature detection result and needs no prior knowledge. With samples increase, BotCatch can gradually get robust and accurate.

The input values for our correlation engine are signature and behavior detection value. Among these values, s is the signature detection value, b is the behavior detection value, and w is the final detection value. The correlation engine uses the correlation algorithm shown in Algorithm 1 to generate the final detection value. Because $s \in [0, 1]$, $b \in [0, 1]$, and $\alpha, \beta \in [0, 1]$, w is still a value between 0 and 1. Suppose θ is the final threshold, and if $w < \theta$, the suspicious file is considered benign; otherwise, it is considered malicious.

Algorithm 1 Correlation Algorithm

Input:
Signature detection value s
Behavior detection value b
Output:
Correlation result w
if behavior_on == false **then**
  $w = s$ {Behavior analysis is not ready}
else
  if $max(s, b) \geq \gamma$ **then**
    $w = max(s, b)$
  else
    Get the value of coefficients α and β
    $w = \alpha s + \beta b$
  end if
end if
if $w \geq \theta$ **then**
  This file is malicious
else
  This file is benign
end if
Return w

In the correlation algorithm, we first check whether behavior analysis is ready. It is ready when it has trained at least once. If it is not ready, BotCatch is in the initial stage and the final detection result is determined by the signature detection value. If behavior analysis is ready, we compare the signature detection value s with the behavior detection value b and obtain the maximum $max(s, b)$. If $max(s, b) \geq \gamma$, then $w = max(s, b)$. Note that γ is greater than θ, thus w is sure to be greater than θ, and the file is malicious.

Under this condition, we only use the signature or behavior detection result. The file will be considered malicious in three different situations. In the first situation, $s \geq \gamma$ but $b < \gamma$, which means this file has a high signature detection value and low behavior detection value. The high signature detection value shows that many antivirus engines find its signature in their malicious signature databases. That means this file is a well-known malicious file, and the correlation engine detects it as malicious. It may have a low behavior value either because it has not exhibited many malicious behaviors yet or because the behavior-analysis engine is not robust enough yet. In the latter case, BotCatch uses multi-feedback module to improve the behavior-analysis engine. The second situation in which a file will be considered malicious is when $s < \gamma$, but $b \geq \gamma$, which means this file has a low signature detection value but a high behavior detection value. The low signature detection value may indicate that obfuscation techniques are being used or that this is either a variant or a novel bot. However, the high behavior detection value shows that it performs similar behaviors as other bots. Thus, the correlation engine detects it as malicious. The third case is $s, b \geq \gamma$, which means that this file has high signature detection and behavior detection values and is certainly malicious.

When $s, b < \gamma$, the correlation engine uses $\alpha s + \beta b$ to calculate the final detection result. Parameters α and β are weight factors, which are dynamically updated by the multi-feedback module. The values of these parameters are evaluated in our experiment.

4. MULTI-FEEDBACK MECHANISM

Because bots and botnets keep evolving, we should focus on not only existing bot behaviors but also their possible evolutionary descendants. With this motivation, we sought an approach that can adapt to bot evolution. An incremental learning algorithm is a possible solution [28] [29]. Learning from new data without forgetting prior knowledge is known as incremental learning. We utilize a similar mechanism—a multi-feedback mechanism—to gradually optimize BotCatch. This mechanism not only allows the behavior-analysis module to learn from old and new data but also optimizes the signature-analysis module and correlation engine. The multi-feedback mechanism allows our

approach to gradually adapt to the evolution of bots and botnets.

The multi-feedback engine uses the signature, behavior, and correlation results to dynamically optimize BotCatch. It optimizes the signature-analysis module by maintaining the signature database. It optimizes the behavior-analysis module by maintaining the sample set and guiding its learning procedure. And it optimizes the correlation algorithm by maintaining parameters. In this way, the multi-feedback engine acts like a brain, adjusting the approach to make it more robust and accurate.

Algorithm 2 Multi-feedback Algorithm

Input:
Signature detection value s
Behavior detection value b
Correlation result w
Output:
Update signature database, sample set, and correlation weight factors

```
1:  if is_stable == false then
2:    if n < init_limit then
3:      ++n
4:      Add it to sample set
5:    else if n == init_limit then
6:      behavior_retrain()
7:      behavior_on = true
8:      ++n
9:    else
10:     Add it to sample set and flag it as malicious or
        benign
11:   end if
12: else
13:   if w ≥ θ && s < sig_lower then
14:     Add it to signature database as malicious
15:   else if w < θ && s ≥ sig_upper then
16:     Add it to signature database as benign
17:   else if w ≥ θ && b < behav_lower then
18:     ++m
19:     Add it to sample set as malicious
20:   else if w < θ && b ≥ behav_upper then
21:     ++m
22:     Add it to sample set as benign
23:   end if
24: end if
25: if m ≥ μn && behavior_on == true then
26:   behavior_retrain()
27:   Update correlation parameters
28:   n+ = m
29:   m = 0
30: end if
```

4.1. Optimizing signature analysis

The multi-feedback engine maintains the signature database to improve signature analysis when BotCatch reaches a stable and robust state. According to the three input values, the multi-feedback engine filters out two kinds of samples: those with low signature results and high correlation results and those with high signature results and low correlation results. The first kind of samples is not detected by signature detection but is detected by BotCatch. They are thus not uploaded to VirusTotal or recorded as variants or even novel bots. Samples of the second kind are detected as malicious by signature detection but as benign by BotCatch. These samples may have similar signatures as malicious samples but have not exhibited enough malicious behaviors. We extract the signatures of both of these kinds of samples and add them in our local signature database. We flag the first kind of samples as malicious and the second as benign.

If a sample similar to one of the first kind is analyzed again, even if the network scanner still returns a low detection value, the local signature database returns a match. Thus, the signature detection result will be high, as described in Section 3.2.2. To be more specific, if the sample is detected as malicious and the signature is below a certain threshold *sig_lower*, its signature is added to our local signature database as malicious. In Algorithm 2, lines 13 to 16 are the signature optimization procedure.

4.2. Optimizing behavior analysis

The multi-feedback module optimizes the behavior-analysis module in two ways: it maintains the behavior sample set and triggers the learning procedure.

4.2.1. Maintaining the behavior sample set.

To maintain the behavior sample set, the multi-feedback module uses different mechanisms based on the states of BotCatch. Before BotCatch reaches stable state, the multi-feedback module uses the first mechanism, which is on lines 1 to 11 of the Algorithm 2. Because BotCatch is not stable and robust, every sample is added to the sample set for behavior training. When the total sample number n reaches the initial threshold *init_limit*, it triggers first time behavior training.

After BotCatch reaches a stable state, the multi-feedback module uses the second mechanism. With this mechanism, it does not add all samples to the sample set—it extracts only two kinds of special samples. Samples of the first kind have low behavior detection results and high correlation results. The reason for the low behavior detection result may be that the sample did not exhibit much malicious behaviors during monitoring. This kind of sample is nonetheless added to the sample set as malicious. Samples of the second kind have high behavior detection results and low correlation results. Samples like these are added as benign.

4.2.2. Triggering the learning procedure.

The multi-feedback module triggers the learning procedure to make behavior analysis more stable and robust. With more samples in the training set, the learning model

DOI: 10.1002/sec

is more accurate, especially with different kinds of samples. By gradually adding more samples and triggering the learning procedure, the multi-feedback module makes the behavior-analysis module more efficient and effective. Behavior training is triggered for the first time when sample number n reaches an initial threshold *init_limit*. After training, the behavior-analysis module is ready, and the variable *behavior_on* is set to *true*. The multi-feedback engine records the total sample number n of last training and new sample number m. When the ratio of n and m reaches a certain level μ, the feedback engine triggers the retrain procedure of the behavior-analysis engine. It also triggers the update procedure of the correlation parameters, as shown on lines 25 to 30 of Algorithm 2. There are two important thresholds: *init_limit* and μ. If they are too small, then the learning procedure is inefficient. If they are too large, the triggering of feedback will take a long time. Thus, these two values must be chosen carefully. We present the impact of them in an experiment.

4.3. Optimizing correlation engine

The multi-feedback engine also optimizes the parameters of the correlation engine. In the correlation algorithm, α and β denote the weight of signature and behavior detection, respectively. In order to improve the adaptivity of our approach, we use a dynamic value instead of static value. To begin with, the behavior-analysis module is not robust, but the signature-analysis module is, so we set a high detection signature weight and low behavior weight. In this circumstance, detection results rely more on signature analysis. When the training procedure of behavior-analysis module is triggered, the update of correlation parameters is synchronous. The behavior weight gradually increases and the signature weight factor gradually decreases, concurrently. After they reach certain thresholds, the two weight factors remain stable, causing BotCatch to reach its stable state. There are three important parameters involved in this process: α, β, and the increase step δ. For example, we can set the initial value of α to 0.9, β to 0.1, and the increase step δ to 0.05. These parameters are configurable. For example, if you want to stress behavior detection, you can increase β to a higher value. We evaluated these parameters in our experiment.

5. EXPERIMENT

This section presents experiment details, including system implementation, data collection, the impact of different parameters, experiment results, and performance overhead.

5.1. System implementation

In the signature-analysis module, we use VirusTotal [25] as our network scanner. VirusTotal is a free online service that analyzes suspicious files and URLs and facilitates the quick detection of viruses, worms, Trojans, and other kinds of malicious content that is detected by antivirus engines and network scanners. VirusTotal can return the detection results of about 47 different antivirus engines with the most updated signature databases. Our signature-analysis engine looks up the suspicious analysis results of the file on VirusTotal and uses it for further analysis. VirusTotal draws from 47 antivirus engines at the time of this writing, and every engine returns a result of 0 or 1. A result of 0 denotes that the file is benign, and a result of 1 denotes that the file is malicious. Note that the file is not actually uploaded on VirusTotal, and the signature result is from former analysis. Thus, if the file was not previously uploaded on the website, no results will be retrieved.

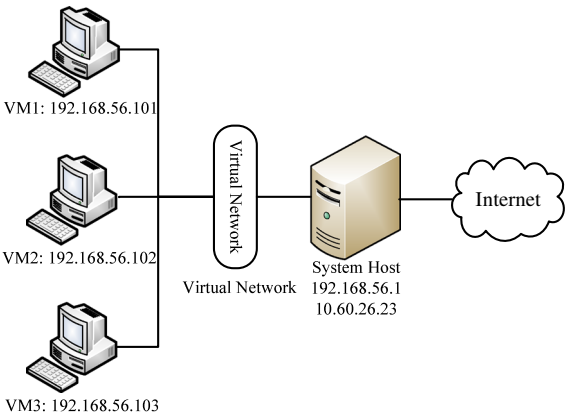

Figure 2. Topology of analysis host.

In the behavior-analysis module, host behavior monitor executes the suspicious file in an isolated environment. It injects suspicious processes using QueueUserAPC() to record registry-related and file system related system calls, parameters, and status information. It uses the PCAP library to extract network information, including Internet protocols, HTTP requests, Simple Mail Transfer Protocol (SMTP) traffic, and domains. We use Cuckoo [30]—the leading open-source automated malware analysis system—as our analysis system.

The topology of our analysis host is shown in Figure 2. The analysis host has three analysis virtual machines. They form an isolated local network. In order to capture accurate network information, we configured the analysis machines to connect to the Internet. The analysis host has the following configurations: Intel Q6600 quad-core processor, 2.40 GHz, 2 GB RAM, and Ubuntu 12.04 operating system. We use VirtualBox 4.2.6 as our virtual machine with Windows XP SP3.

5.2. Data collection

We collected bot binaries from Open Malware [31], which has more than 5 million pieces of malware on its website. We collected a total of 636 bot samples consisting of 47 different kinds of bots. We summarized their names

Table II. Bots for evaluation.

Name	Numbers	Name	Numbers	Name	Numbers
abot	6	cone	19	Lethic	20
agent	20	ebot	4	mbot	1
agobot	20	fbot	9	Nugache	7
Asprox	9	Forbot	20	pbot	20
BiFrost	20	gbot	20	Phatbot	1
Bagle	20	GTbot	4	Pushdo	2
bbot	2	ibot	1	qbot	16
Bobax	20	IRCBot	20	rbot	20
cbot	4	kbot	20	Rustock	20
Conficker	20	koobface	20	Rxbot	20
dbot	4	Kraken	8	sbot	8
Donbot	16	lbot	2	sdbot	19
Sinit	20	SpyBot	20	Srizbi	20
Storm	20	Trojan	20	ubot	2
vbot	2	xbot	20	Zeus	20
Flux	20	Lizard	15		

and numbers in Table II. For benign samples, we tested Google Chrome, Mozilla Firefox, Internet Explorer, bitcomet, uTorrent, Eudora, eMule, and mIRC. We also collected some free tools on the Internet as benign samples, such as Sysinternals Live [32] tools. We collected 150 benign samples, for a total of 786 samples.

5.3. Evaluation metrics

We used several evaluation metrics to comprehensively evaluate our prototype, including receiver operating characteristic (ROC), F-measure, and accuracy. The confusion matrix in Table III presents the relationship between true positive (TP), true negative (TN), false positive (FP), and false negative (FN). ROC curves illustrate the performance of a classifier system as the discrimination threshold is varied. ROC curves show the relationship between the FP rate and TP rate, as presented by Equation (3). Based on ROC, we calculated the area under the curve (AUC) values using AUC Calculator to illustrate their detailed performance [33]. The F-measure (FM) combines precision (P) and recall (R) to present a balanced result [34]. Accuracy (A) presents the proximity of measurement results to the true value [35]. Equation (4) is used to calculate precision, recall, F-measure, and accuracy.

$$\begin{aligned} TP_{rate} &= \frac{TP}{TP+FN}, TN_{rate} = \frac{TN}{TN+FP} \\ FP_{rate} &= \frac{FP}{TN+FP}, FN_{rate} = \frac{FN}{TP+FN} \end{aligned} \tag{3}$$

$$\begin{aligned} P &= \frac{TP}{TP+FP}, R = \frac{TP}{TP+FN} \\ FM &= 2\frac{PR}{P+R}, A = \frac{TP+TN}{TP+TN+FP+FN} \end{aligned} \tag{4}$$

Table III. Confusion matrix.

		Predicted	
		Malicious	Benign
Actual	Malicious	True positive	False negative
	Benign	False positive	True negative

5.4. Impact of different parameters

This section presents how to set the parameter values of BotCatch. The correlation and multi-feedback algorithms contain several parameters, including α, β, γ, *init_limit*, μ, and the final threshold θ. These parameters are vital to our prototype, and well-selected values can significantly improve the detection approach.

We used four different experiments to find the best values for each parameter. In each experiment, we supposed that other parameters are appropriate and have no influence on the final detection result. Because our approach uses feedback mechanisms, the initial submission order will have an impact on the approach. We generated 10 different file submission orders using randomized algorithm. Benign files and bots were submitted randomly to BotCatch.

5.4.1. Parameter 1: *init_limit*.

Parameter *init_limit* in the multi-feedback algorithm is the maximum number of parameters for first time training. It affects the detection rate of the initial stage and the speed at which stability is achieved. In this experiment, we set other parameters to appropriate values, such as α = 0.8, $\beta = 1-\alpha$, $\mu = 0.2$, $\gamma = 0.7$, and $\theta = 0.5$. We evaluated *init_limit* for values between 10 and 100 with an incremental step of 10. For each *init_limit* value, we evaluated 10 different submission orders and used the maximum, mean, and minimum values.

DOI: 10.1002/sec

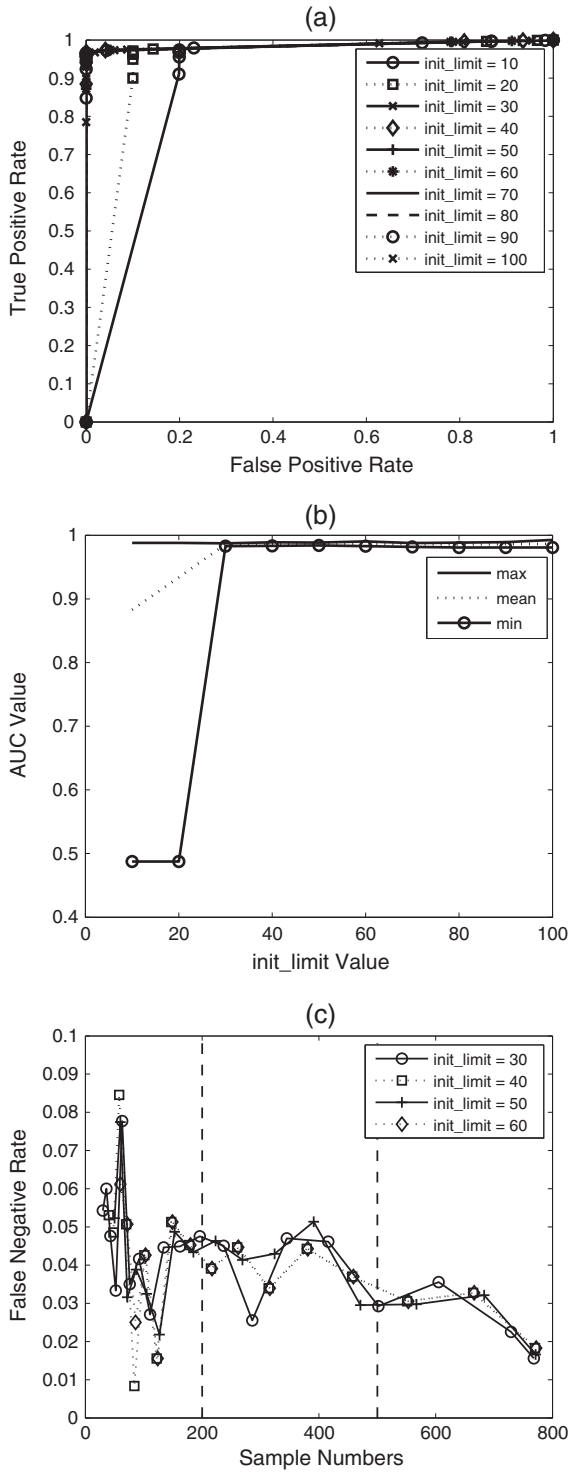

Figure 3. Detection rate with different *init_limit* value.

Table IV. Area under the curve values and retrain times of different *init_limit*.

init_limit	*Max*	*Mean*	*Min*	Times
10	0.988	0.883	0.487	23
20	0.988	0.934	0.487	20
30	0.987	0.986	0.983	18
40	0.989	0.987	0.983	16
50	0.988	0.986	0.984	15
60	0.990	0.987	0.983	14
70	0.988	0.986	0.982	13
80	0.988	0.986	0.981	13
90	0.989	0.986	0.981	12
100	0.993	0.987	0.981	12

Figure 3(a) presents ROCs with different *init_limit* values. Based on ROC, we also calculated maximum, mean, and minimum AUC values, as shown in Figure 3(b). These figures show that *init_limit* = 10, 20 has the lowest AUC value, while others all have almost the same curve. Note also that AUC values for *init_limit* values above 60 drop very little. The specific AUC values are shown in Table IV. For *init_limit* values above 30, mean AUC values are almost the same. We are more concerned about minimum values than maximum values. Above *init_limit* values of 60, the minimum AUC values drop a little. In this way, we first eliminate the values below 30 and above 60.

Because FP rates are almost 0%, we present a more detailed run-time FN rate for *init_limit* values from 30 to 60 in Figure 3(c). We divide the figure into three stages: initial stage, middle stage, and final stage. The initial stage includes samples 0 to 200, in which BotCatch relies more on signature detection than behavior detection. In the middle stage, from samples 201 to 500, BotCatch is set to improve stability and accuracy. In the final stage, BotCatch gradually becomes both stable and accurate. We set two metrics to evaluate them: (i) stability in the initial stage and (ii) increases in stability and accuracy. In the initial stage, an *init_limit* value of 40 resulted in high fluctuation, but values 50 and 60 provide better performance. In the middle stage, values of 40 and 60 are more stable than others and become more stable faster. In the final stage, all *init_limit* values reached a stable state with a low detection rate.

Taking all these into consideration, we selected 60 as the value of *init_limit*. Note that higher values may provide better performance, but 60 has more retrain procedure than them as shown in Table IV. An *init_limit* value of 60 presents more obvious evolution details with a sufficient detection rate in all stages.

5.4.2. Parameter 2: μ.

Parameter μ is a trigger threshold between the previous total sample number n and added sample number m. When $m \geq \mu n$, it triggers the behavior-retraining procedure and modification of the correlation parameters. In this experiment, we set *init_limit* to 60, and the other variables the same as in the last experiment. We evaluated μ from 0.1 to

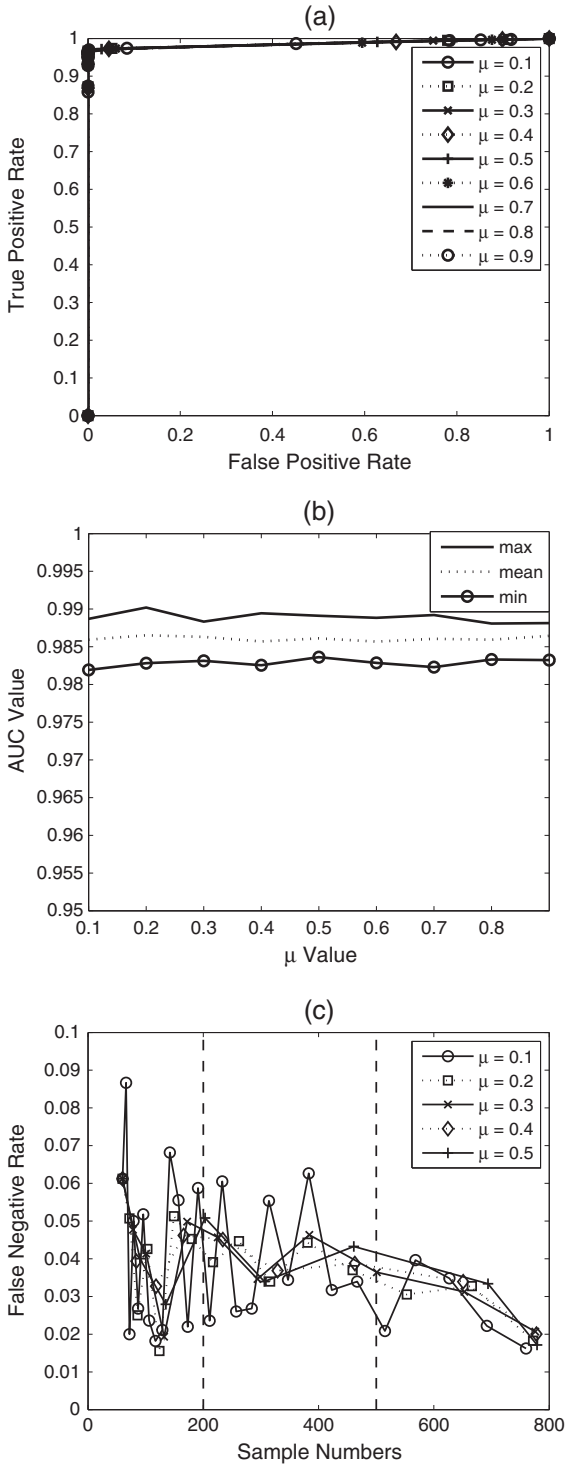

Figure 4. Detection rate with different μ value.

Table V. Area under the curve values and retrain times of different μ.

μ	*Max*	*Mean*	*Min*	Times
0.1	0.989	0.986	0.982	26
0.2	0.990	0.987	0.983	14
0.3	0.988	0.986	0.983	10
0.4	0.989	0.986	0.983	8
0.5	0.989	0.986	0.984	7
0.6	0.989	0.986	0.983	6
0.7	0.989	0.986	0.982	5
0.8	0.988	0.986	0.983	5
0.9	0.988	0.986	0.983	4

0.9, with an increment step of 0.1. For every value, we also evaluated 10 different submission orders.

Figure 4(a) presents ROCs with different μ values—they are almost the same. Based on ROCs, we also calculated accurate maximum, mean, and minimum AUC values, as shown in 4(b). Table V presents the accurate AUC values. The values differ somewhat, 0.2 has a higher maximum value, and 0.5 has a higher minimum value. However, from these figures, we are not able to affirm which one is better.

We thus present the more detailed run-time FN rates in Figure 4(c). We also divide it into three stages: initial, middle, and final. Obviously, 0.1 fluctuates too much, and others all perform well enough in all three stages. They all reach a stable and accurate final stage. Table V presents the retrained times of different μ values. A μ value of 0.1 has up to 26 retraining times, which consumes too many resources, while 0.4 to 0.9 have fewer than 8 and so cannot present a detailed evolutionary procedure. Taking all of these into consideration, we select 0.2 as the μ value.

5.4.3. Parameter 3: α and β.

Parameters α and β significantly impact signature and behavior detection by the correlation engine. They are dynamic values and change at increments of 0.05. In this experiment, we evaluated the initial values of these two parameters. Note that $\beta = 1 - \alpha$, thus we evaluate α values from 0.5 to 1.0. We set μ to 0.2 and set the others as in the last experiment. For each α value, we also evaluate 10 different submission orders.

Figure 5(a) presents the ROCs with different α values—they are almost the same. Based on ROC, we also calculate their accurate maximum, mean, and minimum AUC values, as shown in Figure 5(b). Table VI presents the accurate AUC values. Their values differ little from these figures, so we could not identify which one is better.

We then analyzed the detailed run-time FN rates as shown in Figure 5(c). However, they also differed very little. Recall that in the correlation algorithm, before first time training, the final detection result is equal to the signature detection result. We set *init_limit* to 60. After training, the correlation engine combines the two detection results to generate the final result. At this time, behavior detection

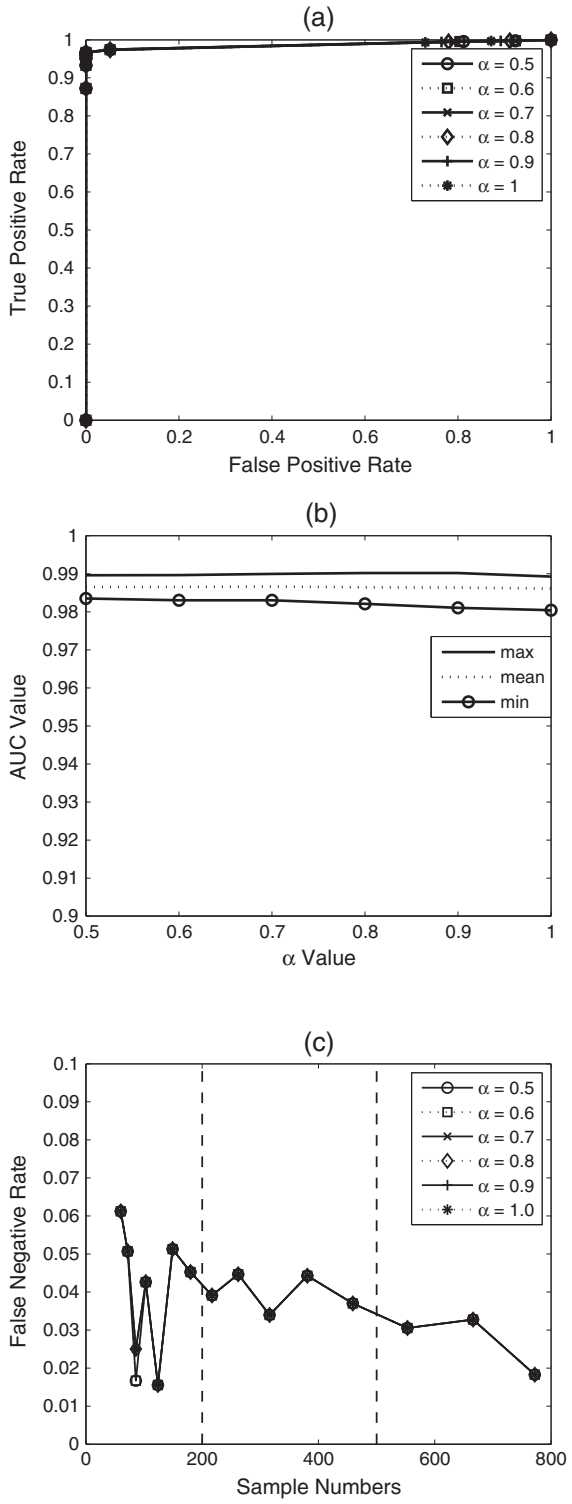

Figure 5. Detection rate with different α value.

Table VI. Area under the curve values of different α.

α	*Max*	*Mean*	*Min*
0.5	0.990	0.987	0.983
0.6	0.990	0.987	0.983
0.7	0.990	0.987	0.983
0.8	0.990	0.986	0.982
0.9	0.990	0.986	0.981
1	0.989	0.986	0.980

provides sufficiently detailed results. Thus, the correlated results for different α values are almost the same. There may be another reason that the samples are so similar, that after first time training, the behavior detection engine is able to classify most samples with high accuracy. In order to provide more different samples as feedback, we select an α value of 0.8 and β of 0.2. Although the difference is not obvious in this experiment, we still select a high value for α to make our approach adapt to more samples.

5.4.4. Parameter 4: θ and γ.

Parameter θ is the final threshold for determining whether a suspicious file is malicious or benign. Parameter γ is another threshold when $max(s, b) \geq \gamma$ and thus $w = max(s, b)$. We only evaluate θ and set γ greater than θ. In the correlation engine, γ goes when either the signature or behavior detection value, but not both, is very high. For example, if a novel bot is submitted to BotCatch, signature detection returns 0 or a very low value, because its signature has not yet been stored. However, the behavior detection engine may return a rather high value, and the correlation result may be lower than θ. In this situation, BotCatch generates a FN. However, if γ is considered, the novel bot will be detected as malicious. Thus, we believe a value in $[\theta, 1]$ is appropriate. In evaluating θ, we set other parameters as described earlier. We evaluate θ from 0.4 to 0.7 with a step of 0.05.

Figure 6(a) presents the FN and FP rates of BotCatch with different θ values. FP rates are all 0 except for a θ of 0.55, which also has a rather low value. FN rates also vary a little, remaining between 0.03 and 0.04. Figure 6(b) presents more detailed run-time FN rates, which are also very similar. The similarities between them are also rather high. The two diagrams do not identify a clearly superior value—all θ values yielded precise detection results. We selected 0.5 as the θ value and 0.7 as the γ value.

5.5. Experiment results

This section provides our experiment results when using well-defined parameters. We first establish the benchmark that we used for comparison, then provide the detection results.

5.5.1. Benchmark establishment.

Because our prototype leverages a multi-feedback mechanism to correlate signature-based and behavior-

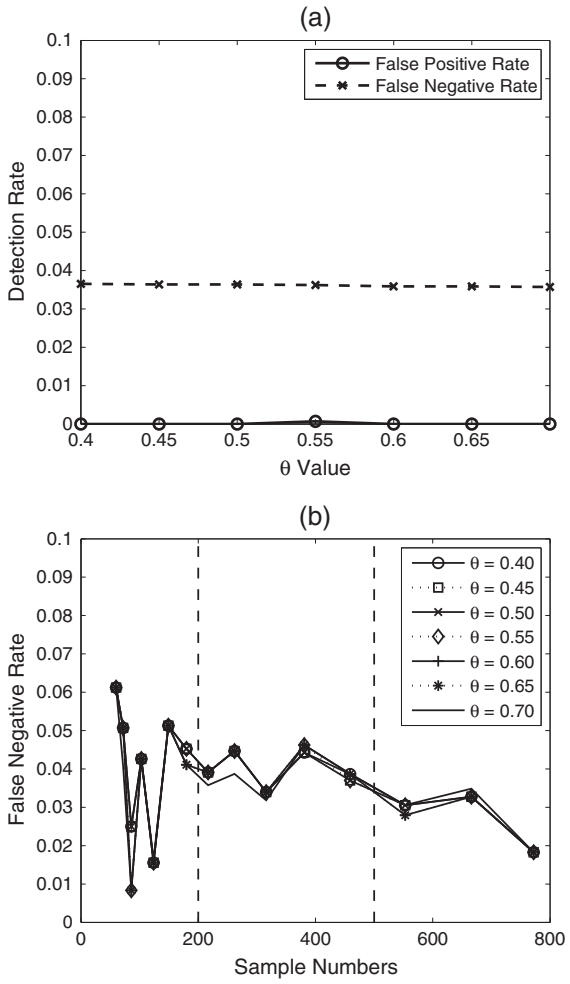

Figure 6. Detection rate with different θ value.

Table VII. Detection results of benchmark approaches.

	TP	TN	FP	FN
Signature	586	150	0	50
Behavior	626	125	25	10
SVM correlation	623	137	13	13

TP, true positive; TN, true negative; FP, false positive; FN, false negative; SVM, support vector machine.

based bot detection, we use signature detection results, behavior detection results, and SVM [36] correlation results as benchmarks.

In evaluating signature detection results, we use the detection results of the signature-analysis module described in Section 3.2.2. We submit all 786 samples to signature-analysis module and compare the results with the same threshold θ to determine whether each sample was malicious or benign. After that, we compared signature detection results with the real results to generate the final signature detection results. Table VII presents the final signature detection results.

Table VIII. Detection results of different order.

Order	TP	TN	FP	FN
1	611	150	0	25
2	613	150	0	23
3	614	150	0	22
4	615	150	0	21
5	612	150	0	24
6	611	150	0	25
7	612	150	0	24
8	615	150	0	21
9	614	150	0	22
10	613	150	0	23
Mean	613	150	0	23

TP, true positive; TN, true negative; FP, false positive; FN, false negative.

In evaluating behavior detection results, we employed machine learning classifiers using LIBSVM [36]. We use a fivefold cross-validation model to evaluate it. We used cross-validation (a technique for protecting against overfitting of predictive models) to mitigate possible biases in the data. As a side benefit, cross-validation yields far more statistical data from a given dataset. In fivefold cross-validation specifically, the whole dataset is randomly divided into five groups and the classifier is re-estimated five times, holding back a different group each time. The overall detection result is the mean result of the five classifiers. We randomly divide all 786 samples (including 636 bot and 150 benign samples) into five groups. In each group, there were 30 benign samples. The first four groups had 127 bot samples and the last had 128 samples. Table VII presents the final behavior detection results.

In evaluating SVM correlation, the signature and behavior detection results are the same as those of BotCatch with no feedback mechanism and the SVM correlation algorithm. We first use fivefold cross-validation to generate posterior classification probability and then use fivefold cross-validation to correlate signature detection results and posterior classification probabilities. Table VII presents the final SVM correlation detection results.

5.5.2. Detection results.

After setting appropriate parameter values and establishing a benchmark, we also randomly generated 10 different file submission orders and submitted them to BotCatch. We utilized several metrics to evaluate them.

Table VIII summaries TP, TN, FP, and FN values of different orders. They all have 0 FPs, which means that BotCatch is able to detect all benign samples. In order to compare our approach with benchmark approaches, we used different metrics to evaluate them and present the results in Table IX. From the perspective of FP rate, the signature approach and BotCatch have a 0% FP rate, while behavior and SVM correlation approaches have 16.67% and 8.67% FP rate, respectively. In contrast, the signature approach and BotCatch have higher FN rates. We were

Table IX. Detection results of different evaluation metrics.

	FP rate (%)	FN rate (%)	F-measure	Accuracy (%)
Signature	0	7.86	0.959	93.6
Behavior	16.67	1.57	0.973	95.5
SVM correlation	8.67	2.04	0.98	96.7
BotCatch	0	3.62	0.982	97.1

TP, true positive; TN, true negative; FP, false positive; FN, false negative; SVM, support vector machine.

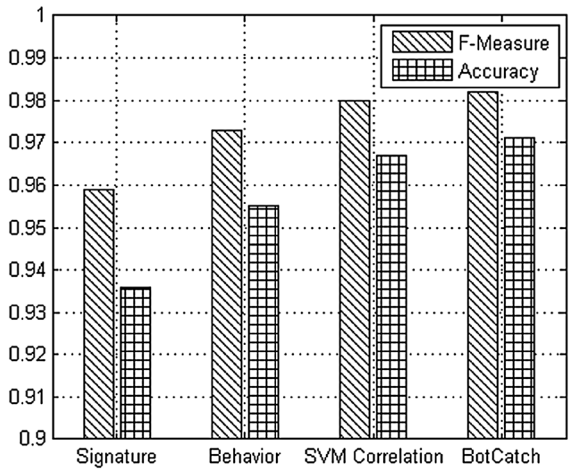

Figure 7. F-measure and accuracy.

not able to draw conclusions from FP and FN rate, so we gathered comprehensive F-measure and accuracy evaluation metrics. Figure 7 presents a comparison of F-measure and accuracy values. By comparing Table IX and Figure 7, we can identify that SVM correlation and BotCatch provide better detection results than the signature and behavior approaches, with BotCatch being a little better than SVM correlation.

Note that our approach is adaptive to samples, which means that it can gradually improve as the number of samples increases. To prove this, we present the more detailed run-time F-measure and accuracy results in Figure 8. In these figures, we compare BotCatch to the signature, behavior, and SVM correlation approaches. We also present mean BotCatch detection results to clearly show the evolution as the number of samples increases. We divide the picture into three stages: the initial stage is from the first sample to sample 200. The middle stage is from the sample 201 to 500, and the remaining samples are in the final stage.

F-measure is presented in Figure 8(a). In the initial stage, BotCatch is not stable; its F-measure values fluctuate around the values of the behavior and SVM correlation approaches. In this stage, BotCatch relies more on signature detection results, while still providing better result than pure signature detection. During the middle stage, the BotCatch curve fluctuates between the behavior and SVM correlation curves. As the number of samples increases,

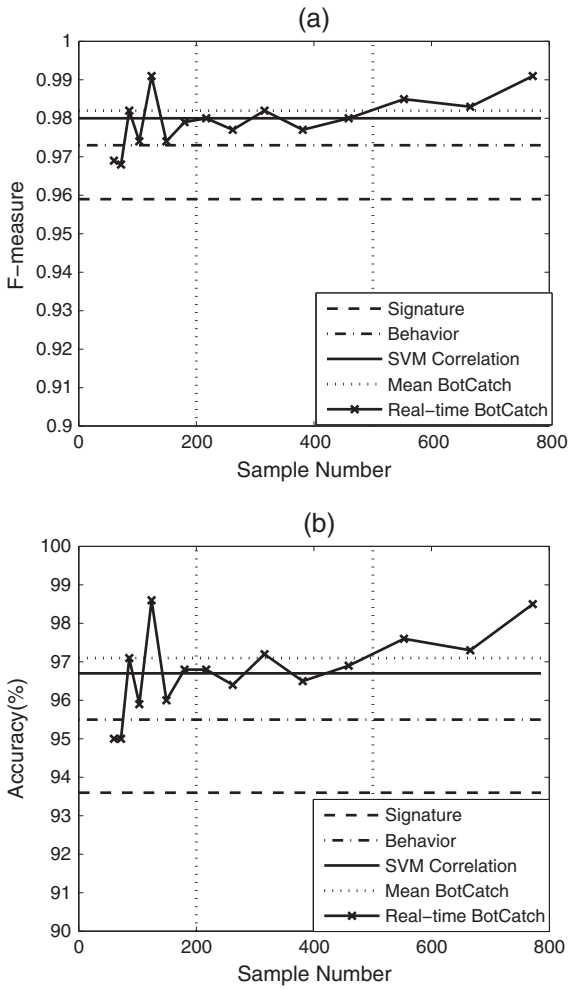

Figure 8. Real-time F-measure and accuracy.

behavior detection is given greater weight than in the initial stage. BotCatch starts to provide more stable and better results than behavior detection, although it still performs worse than SVM correlation. At the end of middle stage, BotCatch has triggered several cycles of retraining, and it has reached a stable and robust state. In the final stage, the curve of BotCatch is substantially above that of SVM correlation. In this stage, BotCatch reaches a 0.991 F-measure value. The accuracy curve in Figure 8(b) resembles the F-measure curve.

Taking the aforementioned results into consideration, we can draw several conclusions: (i) The correlation algorithm in our approach is efficient and effective at combining signature and behavior detection results. (ii) The multi-feedback mechanism makes our approach adaptive to samples, allowing it to gradually become more robust and accurate. (iii) Other correlation algorithms, such as SVM, are also effective; however, our correlation algorithm with its multi-feedback mechanism provides better detection results.

5.6. Performance overhead

We use the *top* command to monitor CPU and memory usage every other second. We first record about 2 h during which only the operating system is running without any other software. Then, we record about another 2 h, during which our prototype is being used normally. Figure 9 shows a summary of the CPU and memory usage.

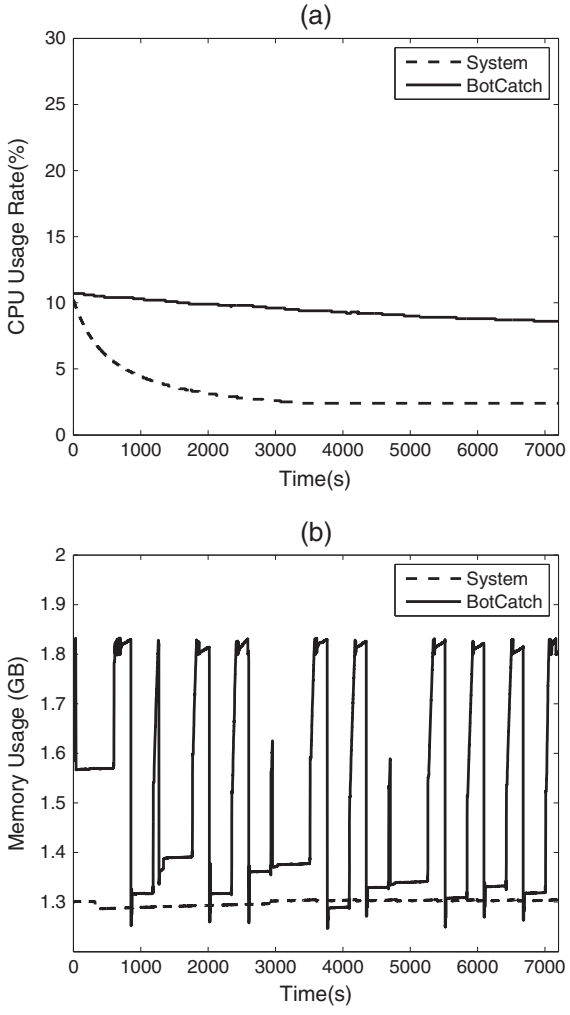

Figure 9. Performance overhead.

At the beginning of the diagram in Figure 9(a), BotCatch and the operating system both provide a CPU usage rate of about 10%. This is because that human operation and the starting of the program consume few resources. They both then reduce their usage at different rates. After 2000 s, system CPU usage stabilizes at 2.4%. However, BotCatch usage declines only from 10.7% to 8.6% in 7200 s. CPU usage by BotCatch remains fairly stable.

Unlike CPU usage, memory usage is unstable. The memory of the operating system remains at about 1.3 GB. Every peak in the BotCatch curve occurs when BotCatch analyzes a suspicious sample. Note that we distribute 512 MB for every virtual machine, thus the apex reaches above 1.8 GB. When BotCatch is quiet, the memory usage is between 1.3 and 1.4 GB, which consumes only a bit more resources than the pure operating system.

Taking the preceding text into consideration, when BotCatch is quietly running, it consumes few resources. However, when BotCatch is analyzing suspicious samples, the overhead that it incurs increases rapidly. This is a common challenge for virtual machines and emulation methods. Although the overhead of our approach is high, there is no risk in a virtual machine, and our prototype can achieve a high detection accuracy. We believe that the sacrifice of incurring a slightly greater overhead is a reasonable price for high detection accuracy.

6. DISCUSSION

No detection approach is perfect. Our BotCatch detection approach is no exception. For example, an advanced bot may detect whether it is running in a virtual machine before performing malicious activities. This is a common challenge for all virtual machine based detection approaches. There is also an agent running in the virtual host, which cleverly designed bots can detect. BotCatch can address this issue, because our approach combines signature and behavior detection. Although cleverly designed bots can evade behavior detection, signature detection is still effective. Thus, BotCatch can still detect these kinds of bots with a well-designed correlation engine.

As the first evolving tendency of bot and botnet illustrates, existing bots utilize advanced hidden techniques to evade detection, such as dividing one process into several, using covert channels to communicate. These hidden techniques are effective to evade some behavior-based bot detection approaches. For example, by dividing one process into several can effectively evade approaches based on single process. However, BotCatch monitors all the behaviors to provide a comprehensive detection result. In this way, BotCatch can steal some of these advanced hidden techniques.

Another limitation of virtual machine based detection is that suspicious files run on the virtual machine for only several minutes. During this period, many bot samples can take a significant number of malicious actions. But some cleverly designed bots may use delay mechanisms to evade

DOI: 10.1002/sec

detection. For example, bots can sleep for random numbers of seconds between continuous malicious behaviors. Some bot programs may be terminated before they finish the life cycle of the botnet. This can confound our behavior detection approach, although it has no influence on signature analysis. Thus, our approach is still able to detect them.

Although our approach mainly focuses on bot detection, it is able to detect other malwares with specific behaviors and parameters. However, existing malware also utilize different mechanisms to evade detection. Metamorphism is a representative mechanism that changes the internal structure of software while maintaining its functionality [37,38]. Metamorphism is not generally applied to botnets, it still represents a great challenge to signature detection. Although a cleverly designed bot can evade signature detection, it still performs intrinsically malicious behaviors that can be detected by the behavior detection module. By combining signature-based and behavior-based detection, BotCatch can still detect them.

The 0% FP rates of both the signature approach and BotCatch are remarkable. However, the benign samples that we used may slightly bias the results. For example, most of our benign samples are pieces of well-known software, so their signatures are definitely not in the malicious signature database. For another reason, the benign samples all quite similar to each other but different from the malicious samples. In our future work, we would like to collect benign samples from less well-known software, in order to evaluate our approach more accurately.

In regard to system overhead, besides the CPU and memory usage of BotCatch, there is another important evaluation metric—the extra time required to execute the code. This metric reflects the efficiency of the monitor or hook mechanism. However, this evaluation metric is not easy to quantify in our experiment. We will try to quantify this metric in our future work.

Our approach uses a multi-feedback mechanism and combines signature and behavior detection results. However, the feature vector of behavior analysis is constant. These features may not work well with some new generation bots, such as social bots [39,40], and Tor-based botnets [14,15]. These new generation bots hide their malicious behaviors in ways that make it difficult to detect. Our approach may also be unable to detect such novel bots. Creating an approach that identifies these kinds of bots is the primary goal of our major future work. We would like to perform this research from several perspectives: (i) We want to deeply analyze the specific malicious behaviors of new generation bots, especially their host behaviors. In addition to classifying novel bots, we can also classify conventional bots into different types with different host behaviors. (ii) We want to update the feature vector with these new behaviors and to provide more detailed detection results that contain more than a classification of samples as bot or benign. For example, we could have results that include detailed bot classification information. (iii) We also want to optimize the multi-feedback engine with well-designed parameters. We can also try to utilize a genetic algorithm to adapt our approach to different bots.

7. CONCLUSION

We propose a multi-feedback approach, BotCatch, to detect bots effectively and efficiently on a host by leverage of a combination of signature and behavior. BotCatch primarily consists of five modules: an analysis engine, a signature-analysis module, a behavior-analysis module, a correlation engine, and a multi-feedback module. The analysis engine assigns each suspicious file to the signature-analysis and behavior-analysis engines. The two analysis engines analyze the file and generate signature and behavior detection results. Then, the correlation engine correlates the two detection results to generate the final detection result. The multi-feedback module then optimizes the signature-analysis module, behavior-analysis module, and correlation engine using a multi-feedback algorithm. In order to evaluate our approach, we collected 636 bot and 150 benign samples to test. Besides providing the final detection result, we also analyzed how different parameters affected those results. After defining parameters effectively, we compared BotCatch results to those of signature detection, behavior detection, and SVM correlation. The results indicate the following: (i) The correlation algorithm in our approach effectively combines signature and behavior detection results. (ii) The multi-feedback mechanism adapts our approach to samples and gradually makes BotCatch more robust and accurate. (iii) Other correlation algorithms, such as SVM, are also effective; however, our correlation algorithm with its multi-feedback mechanism provided the best detection results.

ACKNOWLEDGEMENTS

We gratefully acknowledge the funding from National Natural Science Foundation of China under grant no. 61170265, Fundamental Research Fund of Jilin University under grant no. 201103253, and our anonymous reviewers for their helpful comments.

REFERENCES

1. Abdullah RS, Abdollah MF, Noh ZA, Mas' ud MZ, Selamat SR, Yusof R. Revealing the criterion on Botnet detection technique. *IJCSI International Journal of Computer Science Issues* 2013; **10**(2): 208–215.
2. Rodríguez-Gómez RA, Maciá-Fernández G, García-Teodoro P. Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys (CSUR)* 2013; **45**(4): 45.
3. Silva SSC, Silva RMP, Pinto RCG, Salles RM. Botnets: a survey. *Computer Networks* 2012; **57**(2): 378–403.

4. Goebel J, Rishi TH. Identify bot contaminated hosts by IRC nickname evaluation, *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, 2007; 8–8.
5. Park Y, Reeves DS. Identification of bot commands by run-time execution monitoring, *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, Honolulu, Hawaii, 2009; 321–330. IEEE.
6. Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X-Y, Wang XF. Effective and efficient malware detection at the end host, *USENIX Security Symposium*, Montreal, Canada, 2009; 351–366.
7. Shin S, Xu Z, Gu G. EFFORT: efficient and effective bot malware detection, *Proceedings IEEE INFOCOM, 2012*, Orlando, Florida USA, 2012; 2846–2850.
8. Ma W, Duan P, Liu S, Gu G, Liu J-C. Shadow attacks: automatically evading system-call-behavior based malware detection. *Journal in Computer Virology* 2012; **8**(1-2): 1–13.
9. Zander S, Armitage GJ, Branch P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials* 2007; **9**(1-4): 44–57.
10. Baysa D, Low RM, Stamp M. Structural entropy and metamorphic malware. *Journal of Computer Virology and Hacking Techniques* 2013; **9**(4): 179–192.
11. Shanmugam G, Low RM, Stamp M. Simple substitution distance and metamorphic detection. *Journal of Computer Virology and Hacking Techniques* 2013; **9**(3): 159–170.
12. Boshmaf Y, Muslukhov I, Beznosov K, Ripeanu M. Design and analysis of a social botnet. *Computer Networks* 2013; **57**(2): 556–578.
13. Brito F, Petiz I, Salvador P, Nogueira A, Rocha E. Detecting social-network bots based on multiscale behavioral analysis, *SECURWARE 2013, The Seventh International Conference on Emerging Security Information, Systems and Technologies*, Barcelona, Spain, 2013; 81–85.
14. Johnson A, Wacek C, Jansen R, Sherr M, Syverson P. Users get routed: traffic correlation on Tor by realistic adversaries, *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security*, Berlin, Germany, 2013; 337–348. ACM.
15. Biryukov A, Pustogarov I, Weinmann R. Trawling for tor hidden services: detection, measurement, deanonymization, *2013 IEEE Symposium on Security and Privacy (SP)*, San Francisco, California, 2013; 80–94. IEEE.
16. Hsiao S-W, Chen Y-N, Sun YS, Chen MC. Combining dynamic passive analysis and active fingerprinting for effective bot malware detection in virtualized environments, *The 7th International Conference on Network and System Security,* Madrid, Spain, Springer, Berlin Heidelberg, 2013; 699–706.
17. Kugisaki Y, Kasahara Y, Hori Y, Sakurai K. Bot detection based on traffic analysis, *The 2007 International Conference on Intelligent Pervasive Computing, 2007. IPC*, Jeju Island, Korea, 2007; 303–306. IEEE.
18. Stinson E, Mitchell JC. Characterizing bots' remote control behavior, *The 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Lucerne, Switzerland, 2007*, Springer Berlin Heidelberg, 2007; 89–108.
19. Liu L, Chen S, Yan G, Zhang Z. Bottracer: execution-based bot-like malware detection, *Information Security, 11th International Conference,* Taipei, Taiwan, Springer, Berlin Heidelberg, 2008; 97–113.
20. Martignoni L, Stinson E, Fredrikson M, Jha S, Mitchell JC. A layered architecture for detecting malicious behaviors, *Recent Advances in Intrusion Detection, 11th International Symposium,* Cambridge, MA, USA, Springer, Berlin Heidelberg, 2008; 78–97.
21. Jacob G, Hund R, Kruegel C, Holz T. Jackstraws: picking command and control connections from bot traffic, *USENIX Security Symposium*, San Francisco, CA, USA, 2011.
22. Zeng Y. On detection of current and next-generation botnets. *PhD thesis*, The University of Michigan, 2012.
23. Elhadi AAE, Maarof MA, Osman AH. Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences* 2012; **9**(3): 283.
24. Guo H, Pang J, Zhang Y, Yue F, Zhao R. Hero: a novel malware detection framework based on binary translation, *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, Xiamen, China, 2010; 411–415. IEEE.
25. *VirusTotal*. https://www.virustotal.com/. accessd Feburary 2014.
26. Binsalleeh H, Ormerod T, Boukhtouta A, Sinha P, Youssef A, Debbabi M, Wang L. On the analysis of the zeus botnet crimeware toolkit, *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)*, Ottawa, Ontario, Canada, 2010; 31–38. IEEE.
27. Lin H-T, Lin C-J, Weng RC. A note on Platts probabilistic outputs for support vector machines. *Machine Learning* 2007; **68**(3): 267–276.
28. Chen F, Ranjan S, Tan PN. Detecting bots via incremental LS-SVM learning with dynamic feature adaptation, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge discovery and*

Data Mining, San Diego, CA, USA, 2011; 386–394. ACM.
29. Kim TK, Stenger B, Kittler J, Cipolla R. Incremental linear discriminant analysis using sufficient spanning sets and its applications. *International Journal of Computer Vision* 2011; **91**(2): 216–232.
30. *Automated malware analysis—Cuckoo Sandbox*. http://www.cuckoosandbox.org/. accessed February 2014.
31. *Open malware—community malicious code research and analysis*. http://www.offensivecomputing.net/. accessed February 2014.
32. *Sysinternals live*. http://live.sysinternals.com/, accessed February 2014.
33. Davis J, Goadrich M. The relationship between precision-recall and ROC curves, *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, Pennsylvania, USA, 2006; 233–240. ACM.
34. Wang D, Navathe SB, Liu L, Irani D, Tamersoy A, Pu C. Click traffic analysis of short URL spam on twitter, *2013 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)*, Austin, TX, USA, 2013; 250–259. IEEE.
35. *Accuracy and precision - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Accuracy_and_precision [Accessed February 2014].
36. *LIBSVM—a library for support vector machines*. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. accessed February 2014.
37. Lin D, Stamp M. Hunting for undetectable metamorphic viruses. *Journal in computer virology* 2011; **7**(3): 201–214.
38. Wong W, Stamp M. Hunting for metamorphic engines. *Journal in Computer Virology* 2006; **2**(3): 211–229.
39. Burghouwt P, Spruit M, Sips H. Towards detection of botnet communication through social media by monitoring user activity, *Information Systems Security, 7th International Conference,* Kolkata, India, Springer, Berlin Heidelberg, 2011; 131–143.
40. Singh A, Toderici AH, Ross K, Stamp M. Social networking for botnet command and control. *International Journal of Computer Network & Information Security* 2013; **5**(6).