BotTracer: Execution-based Bot-like Malware Detection

Lei Liu[1], Songqing Chen[1], Guanhua Yan[2], and Zhao Zhang[3]

[1]Dept. of Computer Science
George Mason University
{lliu3, sqchen}@cs.gmu.edu

[2]Information Sciences
Los Alamos National Lab
ghyan@lanl.gov

[3]Dept. of Electrical and Computer Engineering
Iowa State University
zzhang@iastate.edu

Abstract. Bot-like malware has posed an immense threat to computer security. Bot detection is still a challenging task since bot developers are continuously adopting advanced techniques to make bots more stealthy. A typical bot exhibits three invariant features along its onset: (1) the startup of a bot is automatic without requiring any user actions; (2) a bot must establish a command and control channel with its botmaster; and (3) a bot will perform local or remote attacks sooner or later. These invariants indicate three indispensable phases (startup, preparation, and attack) for a bot attack. In this paper, we propose BotTracer to detect these three phases with the assistance of virtual machine techniques. To validate BotTracer, we implement a prototype of BotTracer based on VMware and Windows XP Professional. The results show that BotTracer has successfully detected all the bots in the experiments without any false negatives.
Keywords: Botnet, malware detection, virtual machine

1 Introduction

Bots and botnets have become one of the most serious threats to Internet security in recent years [14][22]. Compared with other malware like virus and worms, bot behavior can be very stealthy, making their detection extremely difficult. For example, a bot can stay inactive without any dramatic activities for a long time. Oftentimes, a bot generates only a small amount of traffic, which is hidden among legitimate traffic. Some of botnet research has focused on the understanding of bots and botnets. For example, Barford et al. have analyzed in-depth bot source code [9] and provided insights from several perspectives, while in [27], through trace collection and analysis, authors observed the real-world botnet behavior. Dagon et al. have studied the botnet propagation using time zones [15]. Some research [25] has studied how to identify non-human behavior characteristics in traffic and build IRC server scanners to identify potential botnets. To counter botnets, honeypots have been used to infiltrate the command and control network of botnets [17].

While researchers are improving the detection and defense schemes, bot developers are also constantly making bots more stealthy. Due to the scrutiny on IRC channels, today's bots are bound to other popular applications (e.g., Web browsers [6]) or protocols (e.g., HTTP [16]). Distributed P2P-based botnets, which are much more difficult to detect and shut down than centralized botnet architectures, have also been developed [19][21][28][30]. Advanced bots like `Spam-Mailbot` [12] have applied encryption to defeat traffic scanning.

Recent improvements on bot techniques call for better bot detection techniques. The distinguishing feature of a typical bot attack is three indispensable phases:

- *Automatic startup:* Different from those virus or worms (e.g., email worms) that rely on user intervention, a bot can be started automatically by modifying

the automatic startup process list or registry entries. This is essential for the bot to actively initialize the command and control channel with the botmaster in order to receive commands.

- *Command and control channel establishment:* In spite of various bot hiding techniques, existing bots all need to build a command and control channel. In a large network environment, it is impractical for a botmaster to actively trace all of its bots. To evade detection, a botmaster normally does not actively contact or scan all bots, particularly when the botnet contains a large number of bots behind firewalls or NAT that would not freely allow incoming traffic.
- *Information dispersion/harvesting:* Sooner or later, a bot will be ordered to take some actions through the established command and control channel[1]. A bot can be asked to collect sensitive information from the local machine (*information harvesting*), or participate in organized attacks, such as spamming and DDoS attacks, against a third party (*information dispersion*)[2].

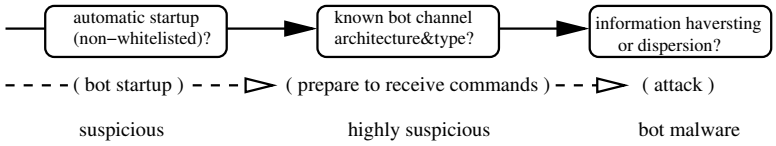

Fig. 1. BotTracer detection logic: startup, preparation, and attack during a bot onset

In this work, we propose BotTracer to detect bots by capturing these three invariant features during their execution. *First*, when a host starts, a virtual machine with the same image is also started. A virtual machine without any human interactions provides an effective playground to identify processes that are automatically started, especially those with networking activities. *Second*, with such a playground with little noise traffic at hand, BotTracer keeps monitoring automatic communications and classifies these communication channels. Since a bot must actively contact a rendezvous point to build a command and control channel with its controller, BotTracer capture these channels and compare them with known characteristics of bot command and control channels. This can significantly narrow down the detection space. *Lastly*, BotTracer constantly monitors system-level activities and traffic patterns of those processes that have been identified as suspicious. Hence, BotTracer is able to capture those bots that are actively performing information harvesting or dispersion. Figure 1 illustrates BotTracer's detection logic. To evaluate the performance of our proposed system, we implement a prototype of BotTracer based on VMware and Windows XP Professional and test a variety of bots. The experimental results demonstrate that BotTracer can detect all these bots without false negatives.

Comparisons with Existing Approaches: Most recently, BotHunter [20] is proposed to detect bots by correlating events from inbound scan to outbound scan. As BotHunter aims to detect bot behavior at the network level, stealthy bots can dodge detection by evading event timing correlation, or conducting local attacks (such as deleting files) without any networking activities. The work in [29] focuses

[1] If a bot simply hibernates, it does not do any harm, although it makes bot detection more difficult.

[2] We do not consider information processing as defined in [19].

on remote control behavior analysis by tracking tainted data from the network. Panorama [31] also relies on taint analysis to analyze malware behavior, although it is done after a suspicious sample has already been detected. A flip side of Panorama is that its effectiveness is contingent on the completeness of the samples that have been collected. SpyProxy [24] also employs behavior analysis through execution to detect malicious Web content. It is, however, based on evidence of malicious side-effects (the attack phase). Compared against these previous schemes, BotTracer leverages virtual machine technology to significantly reduce the detection space, and relies on in-depth bot behavior analysis to detect bots.

BotTracer takes a host-based approach, which complements existing network-based approaches. Efficiency of network-based approaches has already been challenged by various techniques, such as obfuscation [26] and encryption [12]. More importantly, a network-based approach commonly results in the shutdown of the command and control channel server or the change of the DNS entries [18]. Since it leaves infected machines unchanged, they can be easily reclaimed later. In our experiments, we have shown that for traditional centralized botnets, BotTracer is able to locate the centralized server after successfully identifying a bot machine.

The remainder of the paper is organized as follows. We present the design of BotTracer in Section 2. Based on the implemented prototype, we evaluate BotTracer in Section 3. We discuss some limitations in Section 4 and make concluding remarks in Section 5.

2 BotTracer Design

Although bots commonly exhibit the three fundamental features as we identified above, detecting these characteristics is challenging since bot-like malware commonly employ various techniques to conceal themselves. Therefore, in BotTracer, a virtual machine that clones the host image is constructed to provide an ideal detection environment. That is, once the host is started, a virtual machine (VM) that clones the host image is also started automatically. However, the user only operates on the host. The virtual machine thus becomes an environment without human interactions.

On such a playground with significantly reduced noise, BotTracer thus focuses on detecting the three invariant bot behaviors through execution:

- Once the virtual machine in BotTracer starts, automatically started processes will self-expose. After filtering the processes on the whitelist, BotTracer keeps monitoring the remaining ones. As a bot must actively build a command and control channel to the outside before any malicious behavior is conducted, any outgoing traffic from any remaining process on the virtual machine indicates it is suspicious.
- By constantly monitoring its inbound and outbound traffic once a process is flagged as suspicious, BotTracer can identify whether a special command and control channel is established. For example, a traditional IRC-based bot may build a persistent connection to receive commands from the botmaster, while modern bots may periodically contact the botmaster.
- An identified command and control channel indicates a highly suspicious bot-like process. Since a bot will perform information harvesting or dispersion sooner or later, BotTracer, which constantly monitors the highly suspicious

processes, detects information harvesting by tracing relevant system calls and the corresponding parameters that are intercepted through the virtual machine monitor, and detects information dispersion by inferring the traffic patterns.

The detection scheme described above is based on an *ideal* virtual machine environment. In practice, however, there are a number of issues that must be solved to make the detection effective.

2.1 Whitelist and Starting Point Set

In BotTracer, the absence of user interactions on the virtual machine dramatically reduces normal user traffic. To further facilitate the bot detection, it is preferable to eliminate the interference from some legitimate processes that are automatically started on the virtual machine as well. We classify them into three categories as [13]: system daemons, software updates, and network applications automatically started by the OS. The first category covers system daemons like `system.exe`, `svchost.exe`, and `services.exe` in Windows. The second refers to automatic software updates from the well-known Web sites. For the limited number of processes in the first and the second categories, as their networking behaviors are mostly fixed, we can simply whitelist them to allow their network connections.

The third includes user application processes like MSN and ICQ that are configured in advance. For example, if a MSN client is configured to sign in automatically, the MSN client will connect to the MSN server once it is started. In addition, if a bot is bound to a popular application, such as a Web browser, the bot may not automatically start once the virtual machine starts, but starts when a particular application is started. For detection purpose, once they are started by a user on the host, they are started on the virtual machine as well by application synchronizer. For instance, a Web browser typically has a default setting that allows a user to visit a Web site once the Web browser is launched. Thus, once the Web browser is started, it will generate outbound traffic automatically. Since the number of such applications on a host is limited, it is also possible to have their default destinations, which we call *starting points*, whitelisted.

Although being effective most of the time, this approach may unnecessarily burden bot detection: by default a Web browser is set to connect to only one starting point, but the Web page of this site may contain rich information that leads to connections to other sites. If the Web page is frequently updated, this would be more difficult for us to whitelist the traffic based on one starting point. For other applications, the starting point might be a registry server that needs to validate user identification or a name service that provides references to other resources. Thus, allowing connections to the starting point may not only complicate whitelisting, but also make it inaccurate.

More importantly, there is a running copy of the application on the host. If the process on the virtual machine is allowed to send out traffic, it may affect the status of the application process on the host and lead to unexpected results. For example, the automatic sign-in to a service from the virtual machine process may kick out its corresponding host process that signed in before.

Therefore, to guarantee correct semantics of normal applications and ease bot detection, in addition to put them on the whitelist, we also block the connections of user applications in the third categories to their starting points once they are

going through the virtual machine monitor. Blocking these connections can thwart a sequence of actions of the process, and can turn these processes into semi-dormancy or dormancy in most situations according to our experiments. Conservatives can even merge the whitelist into the starting point set. Thus, legitimate traffic and process activities could be minimized, which is very favorable for bot detection.

Generating whitelist and starting point set for system daemons and software updates is relatively easy. We found that nearly all of their network traffic is to well known destinations. BotTracer thus can collect traffic information on a typically configured Windows XP machine. Generating whitelist and the starting point set for network applications is more difficult. BotTracer needs to query starting points of popular applications, such as the default destination of a Web browser, with the intervention of users.

2.2 Command and Control Channel Detection

Now we present how BotTracer detects the initialization of a command and control channel in a controlled virtual machine with minimum noisy traffic. A bot always needs to actively build a command and control channel to communicate with its botmaster. In practice, there are two architectures for operating such channels.

1. **Centralized:** The first is a centralized architecture. Traditionally, IRC-based botnets commonly leverage IRC servers to issue commands to the army of bots. In this centralized mode, there are a number of varieties. For example, the destination could simply be a list of static IP addresses or a list of URLs so that flexible IP addresses could be used. Some bots, such as `Graybird` [6], may use an intermediate point, in which the bot will access a static URL, retrieve the actual centralized server address, and connect to it.
2. **Decentralized:** A more recently emerged architecture that has also been foreseen by many researchers for bot communications is through distributed networks, such as P2P. This decentralized architecture can reduce the risk of being detected. `Nugache` is such a Trojan that uses P2P technology for communication [28]. In addition, not all P2P bots need seed servers. For example, `Sinit` sends special discovery packets to look for peers [28].

Regardless of the architecture, there are two types of command and control channels:

1. **Type 1 – Persistent Channel**: In this approach, a bot process directly starts a connection to the destination and the connection is persistent. The average connection time could be as long as 3.5 hours according to [11]. This type of connections is normally initialized upon the startup of the machine. IRC bots commonly use this approach.
2. **Type 2 – Periodic/Sporadic Channel**: In this approach, the bot process periodically starts connections to a destination. Typically, the destination has not communicated with the host before. An easy variation of this type is to launch aperiodic connections instead of periodic ones. HTTP-based `Bobax` bot [8] falls into this category.

Given the command and control channel architectures and types, we can leverage these known characteristics to construct a bot channel event model. The bot channel initialization event model consists of two levels. The first level represents

the channel type, indicated by low level events, such as a new connection is initialized, an incoming connection is accepted, and a connection is reset. The channel type level generates input to the channel architecture level, which represents whether a centralized channel is built, a decentralized channel is built, etc.

Some IRC-based botnets use persistent channels and the average duration of an IRC bot can be as long as 3.5 hours [11]. By contrast, a typical sporadic channel that uses HTTP may last for only a few seconds. Hence, we use the following heuristics to detect the channel type:

- A *new* connection refers to one whose destination has not been contacted before since the process has started.
- At the beginning, if a new connection is built, the connection is said to connect to an *intermediate* point.
- If an intermediate point is reconnected, the connection is updated to be a *sporadic* one.
- If a connection to an intermediate point or a sporadic connection lasts more than 30 seconds, it is flagged as a *persistent* one.
- When a new connection is accepted, it is flagged as a *sporadic* one.

Based on the above setup, Figure 2 shows the state transitions in our two-level model. Note that in the command and control channel detection, BotTracer focuses on detecting the establishment of a command and control channel without tracking how the channel is used. Therefore, it is expected that we would get some false positives, which BotTracer relies on the next step to further reduce.

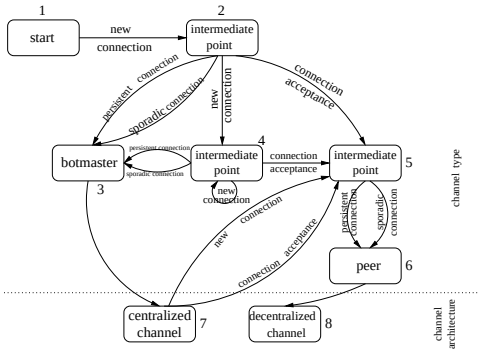

Fig. 2. Command and Control Channel Event Model

2.3 Information Harvesting/Dispersion Behavior Analysis

Information Harvesting Detection For information harvesting, a bot may be instructed to collect the information such as password, game/bank accounts, product keys, some personal information, and report to the botmaster. Some of such information may exist under particular application's directories. Some may be collected from the program's memory space when the application is running. Windows temporary files are also a popular target. For example, malware can search sensitive data in cookies. In addition, some system information is also attractive, such as registry entries. Existing research shows that currently information harvesting is mainly through code injection, keystroke log, and direct memory reading.

While designing strategies to detect and defeat each of these is possible, malware developers may invent new evading approaches. Instead, since information harvesting must involve disk or memory accesses, we rely on the process behavior analysis at the system level to detect information harvesting as follows.

Intuitively, if a bot is commanded to access the disk, no matter what approach it takes, monitoring disk accesses related system calls/APIs could identify any disk accesses. For Windows systems, BotTracer can monitor a limited number of critical system APIs, such as `OpenFile`, `CreateFileMapping`, `CreateFileMappingNuma`, and `OpenFileMapping`. Accessing any of these triggers an alarm.

If a bot harvests information from a process' memory space, no matter which approach (code injection, keystroke log, or direct memory reading) is used, the malware typically starts from querying the information of the process or the window in order to locate the exact victim it wants to peek at. Thus BotTracer can monitor `OpenProcess`, `WriteProcessMemory`, `ReadProcessMemory`, `CreateRemoteThread`, `FindWindow`, `SetWindowsHookEx`, `GetWindowThreadProcessId`, `CreateToolHelp32Snapshot` and their family APIs for potential information harvesting.

Monitoring all these API calls for all processes on the virtual machine is cumbersome since there are a few whitelisted legitimate applications on the virtual machine. Among them, we are particularly concerned about the third category applications, because malware can inject their code into these popular applications. For these applications, they may have disk and memory activities, although their connections to their starting points are cut off (network accesses are not allowed). We have performed extensive experiments and the result shows that most of such processes do not have any further activities without network access, while a few do have disk and memory accesses occasionally. To deal with them, we define their profiles in advance. That is, for a limited number of automatically started processes or popular application processes that may be started by the application synchronizer, we generate their profiles after their connections to their starting points are forcefully cut off without user interactions or network accesses. We call such profiles as *dormant process profile* as most process activities are turned off. The profile includes the resources they can access, the system functions through which they access, etc. In detection, once a process behaves out of its profile, an alarm is raised. We extend the XML language to define the dormant process profile. Figure 3 in Appendix A shows the profile of the `Internet Explorer` that is generated on a clean machine without user interactions or network accesses.

Information Dispersion Detection Besides information harvesting that endangers the infected machine, a bot is commonly commanded to participate in organized attacks, such as DDoS and spam. Many schemes have been proposed to deal with these attacks by leveraging some application level characteristics and are thus application-dependent. From the perspective of an attacking bot, however, all these attacks will show some unique traffic patterns. Furthermore, the traffic destination should not be in the starting point set or on the whitelist. Lastly, it is less likely such traffic is encrypted. Thus, we design our detection and thwarting scheme as follows.

A common feature of information dispersion attacks is that the target of the outgoing traffic is a third party, and often is a destination that the bot has not

Table 1. Command and Control Channel Detection

Name	Alarm Time (s)	Architecture	Type
Agobot	6.532 seconds	Centralized	Persistent
Forbot	34.173 seconds	Centralized	Persistent
Jrbot	1.895 seconds	Centralized	Persistent
Reptilebot	2.719 seconds	Centralized	Persistent
Sdbot	0.953 seconds	Centralized	Persistent
Rxbot	4.409 seconds	Centralized	Persistent
Graybird	2.997 seconds	Centralized	Persistent
Nugache	1.422 seconds	Suspicious	Suspicious

communicated with before. As BotTracer starts to monitor the process behavior from the beginning, it has the record of all the destinations that the bot has communicated with. Before the attack is launched, the communication through the command and control channel is bi-directional. After the bot receives the attack command from the botmaster, the outgoing traffic is likely to go to new destinations. Thus, at a higher level without interpreting any communication content, the destination of the outgoing traffic is different from the previous incoming one, and an asymmetric traffic pattern could be observed.

However, only monitoring outgoing traffic patterns is not sufficient. Recall that on the virtual machine, there is no user interaction and most of the legitimate applications are semi-dormant or dormant. Thus, if there is outgoing traffic from a process on the virtual machine, we can further examine its profile. If the destination is not a starting point specified in the profile, it is highly likely that the process is hijacked by a malware. Moreover, as there is no human interactions on the virtual machine, if there is outgoing email traffic, very likely it is generated due to spam attacks. Lastly, although the bot normally does not generate a large amount of traffic, once it participates in a DDoS attack, its traffic amount would increase remarkably in a short period, which can be leveraged to detect DDoS attacks.

3 BotTracer Evaluation

Based on our design, we have implemented a prototype of BotTracer. In this section, we present the experimental results of BotTracer when a set of representative bots are tested, including the following three classes of bots:

- `IRC bots and their variants` are traditional bots controlled through IRC. We tested a variety of IRC bots including `Agobot4 private`, `Forbot`, `Jrbot`, `Sdbot`, `Reptilebot`, and `Rxbot`.
- `Graybird` has a large number of variants since its first debut in 2001. It is one of the most prolific pieces of Windows malware. We experimented on version 2005. It does not use IRC, but its own communication protocol. To hide itself, it injects itself to `Internet Explorer` (IE). Our testing version can start an IE process and copy itself to IE space and then execute in the context of IE.
- `Nugache` uses encrypted and/or obfuscated P2P traffic for communication. It opens TCP port 8 and has a static list of 22 initial peers to which a peer aims to connect to at TCP port 8. After successful connection, it is going to exchange the list of successfully connected peers. It participates in DDoS attacks once commanded, and it spreads over instant messengers such as American Online Instant Messenger [28]. In Windows, it runs as a `mstc.exe` after infection.

In addition, `Microsoft Outlook Express` and `pcAnywhere` are also experimented to study false positives.

3.1 Prototype Implementation and Experimental Setup

We have implemented the prototype based on Windows XP Professional. Particularly, we use VMware workstation version 5.5.3 for the virtual machine. We use VMware `Converter` [3] to clone the physical machine. The traffic pattern monitor and analyzer are implemented in a *traffic* module, and the process behavior analyzer is implemented as a separate *behavior* module.

The traffic module monitors all ingress and egress traffic after an application starts. As it is necessary to map ports to the owning process for further analysis, we implement our traffic module based on `Enhance Netstat` [4]. It can map a port to its owing process even if the process adopts some approaches to hide itself from `Task Manager`. When the channel architecture and type cannot be detected and there is outgoing traffic that is not going to a starting point, BotTracer reports it as suspicious.

The behavior module is implemented based on Microsoft `Detours 2.1 Express` [1]. It intercepts Win32 function calls. For our experiments, the behavior module is designed to capture all violations of the sensitive data accesses that are not allowed in the dormant process profile. In our current implementation, it monitors a limited number of Win32 functions that are for file and network accesses. In addition, process management functions are monitored.

For experiments, we have set up a controlled network. BotTracer was run on a machine with a 2.79 GHz CPU and 2 GB RAM. The guest OS of VMware is Windows XP Professional that is identical to the host OS.

`Graybird` injects itself to IE at runtime and its dormant profile is shown in Figure 3. For `IRC based bots`, we set up an IRC server on another machine with similar configurations and we modified source code to direct bot samples to our IRC server so that we can issue commands to the bot through a connected IRC client. `Graybird` is configured with its GUI tool. Its botmaster runs on another machine. For `Nugache`, because only binary is available, we can do few configurations.

3.2 Channel Establishment Detection

In the controlled environment, we first test whether BotTracer can successfully detect the channel establishment and the corresponding channel type and architecture. Table 1 shows the detection results for the eight bots. *Alarm Time* is the time between when the bot starts and when its first outgoing traffic is captured.

We found that nearly all bots initialized the command and control channel within 10 seconds after their startup. Furthermore, both `IRC bots` and `Graybird` establish one and only one persistent TCP connection. The entire channel detection time is less than 60 seconds. Note for `Nugache`, as the bot tries to connect to 22 initial peers that are hardcoded in the binary, all these connections failed as expected. BotTracer thus cannot report the architecture and type of the channel. However, since it tries three times for a destination and tries different new destinations in a sequence, BotTracer still reports it as suspicious.

Furthermore, as centralized channels are detected for `IRC bots` and `Graybird`, we check whether or not the host (not the virtual machine) has connections to the same IP and port because on the host there are also identical bot copies. We

found both `IRC bots` and `Graybird` on the host also connect to our IRC server and `Graybird` botmaster, respectively. This confirms that a running copy of the bot process on the virtual machine does not affect its corresponding process running on the host. Furthermore, for bots operated in the centralized mode, it is straightforward to further trace down to the server and shut down the server, and possibly the entire botnet.

3.3 Information Harvesting/Dispersion Detection

As BotTracer alarms for all eight bots, the behavior module is activated as well (note the traffic module is still active in order to thwart potential attacks). Unlike the channel detection which is completed in a short time after a process starts, a bot usually performs information harvesting or dispersion only after it receives commands from the botmaster. Thus, through our experimental setup, we act as the botmaster to start attacks. Particularly, for information harvesting attacks, `Rxbot` was instructed to return keys of products, via a `getcdkeys` command. For information dispersion, we launch an information dispersion attack through `Agobot` by sending it a DDoS command. For `Nugache`, we failed to send any command as we do not have the source code and its behavior is not well understood. In any of the experiments, we keep the logs of traffic and system activities.

Table 2 shows the intercepted APIs and the corresponding parameters for the `Rxbot` process after `getcdkeys` is received.

Table 2. APIs called when `Rxbot` launches attacks

Action	API	Arguments
Access Registry	RegOpenKeyEx RegQueryValueEx	Software\BioWare\NWN\Neverwinter Location
Access Directory	fopen fget	C:\NeverwinterNights\NWN\nwncdkey.ini file handle

Since `Rxbot` has already been reported to be highly suspicious, accessing registry and files under an application directory in the above actions leads BotTracer to report it as a bot and disable its input and output.

Table 3 shows the packet sequences once a `.ddos.httpflood http://www.aaaaa.com 100 www.aaa.com 2000`[3] command is issued to `Agobot`, which requests the `Agobot` to send 100 HTTP requests to `www.aaaaa.com` with a 2000 ms interval. `www.aaa.com` is the HTTP referer. Note that `192.168.88.156` is IP of the IRC server. The IP of the virtual machine is `192.168.88.155`. The attack target uses `192.168.93.52`.

Table 3. Agobot HTTP DDoS Attack Packets

Time (s)	Source	Destination	Type
0	192.168.88.156	192.168.88.155	IRC
0.012	192.168.88.155	192.168.93.52	HTTP
2.608	192.168.88.155	192.168.93.52	HTTP
5.226	192.168.88.155	192.168.93.52	HTTP

In the experiment, `Agobot` sends out each HTTP attack packet for 100 times. Table 3 gives the time BotTracer takes to capture the outgoing attack traffic. In

[3] URL and the public IP addresses are anonymized. The prefix of the public IP address is replaced with 192.168 when necessary.

our implementation, both the traffic pattern monitoring and the starting point (to compare the outgoing traffic destination) in the process profile are leveraged. The default threshold of outgoing packet number is 3, which means it takes 5.2 seconds for BotTracer to detect the attack. Conservative protection can reduce the threshold to 1.

3.4 False Positive Experiments

False positives occur when normal applications are flagged as bot malware. Maintaining and timely updating the dormant profile list for the normal and popular applications on a host can greatly reduce false positive. We first test whether or not a normal application without a profile can be captured in BotTracer. On the host running BotTracer, we install `Microsoft Outlook Express 6`. We set it up to check a `hotmail` mailbox once every minute, and the account and the password are saved before the experiment was run. BotTracer quickly reports this is a bot using a centralized and sporadic channel!

We disable BotTracer, and run it again. We have the following first six packet sequence log as shown in Table 4 without any user interactions. In this table, the first packet is a DNS query for `services.msn.com`. `192.168.68.227` is a domain name server. The application thus obtains the IP address `64.4.60.7`. `Outlook Express` does not keep a persistent TCP connection. Instead, about every one minute it starts a new TCP connection to the Web email server `65.55.154.125` and checks for new emails. This pattern causes BotTracer to report a false positive.

Table 4. Outlook Express 6 Connecting Packets

Time (s)	Source	Destination	Type
0	192.168.88.155	192.168.68.227	DNS
0.095	192.168.88.155	64.4.60.7	HTTP
0.478	192.168.88.155	65.55.154.125	HTTP
1.129	192.168.88.155	65.54.183.193	HTTP
63.908	192.168.88.155	65.55.154.125	HTTP
124.463	192.168.88.155	65.55.154.125	HTTP

To validate whether a user could add its profile to BotTracer to eliminate false alarms, we generate the dormant process profile for `Outlook Express`, and `Outlook Express` is started again in BotTracer. As expected, BotTracer did not raise an alarm. These indicate that it is critical for the user of BotTracer to update the profile list once new applications are installed.

In addition to `Microsoft Outlook Express 6`, `pcAnywhere 12.0.0` is run to see if false positives would be raised. Controlled by a `pcAnywehere` remote, `pcAnywhere` host has similar functions as a `Graybird` bot. Both a `pcAnywehere` remote and a `Graybird` botmaster can manipulate nearly all computer resources under control. We ran both of them on BotTracer. As before, BotTracer successfully detects the command and control channel of `Graybird`, which is flagged as centralized and consistent, while no alarm is raised for `pcAnywhere`. The critical reason for this result is that a `pcAnywhere` remote requires the contact information of a `pcAnywhere` host, while a `Graybird` bot requires the contact information of the `Graybird` botmaster. That is, a `pcAnywhere` host waits to be connected by a `pcAnywhere` remote while a `Graybird` botmaster waits to be connected by `Graybird` bots.

These case studies just show that it is possible to reduce false positives through accumulated process profiles. However, in practice, we believe false positives and

false negatives would be inevitable, particularly when new techniques are continuously adopted by bot developers.

4 BotTracer Limitations

A fundamental assumption of BotTracer is that the virtual machine cannot be detected by the bot. In practice, there are many techniques that can detect virtual machines [7][32]. Thus, if a bot detects whether it is running on a virtual machine based system, our BotTracer will not work properly. This issue can be addressed from two perspectives. *First*, as this is a challenge for all virtual machine based solutions, anti-fingerprinting techniques are still improving [5]. For BotTracer, the detection behavior of a bot could be detected through system level activity monitoring, and thus provides an opportunity to cheat the bot. *Second*, the adoption of virtual machines in practice is quickly increasing [2]. With the continuous performance improvement of the virtual machines, such as VMware and Xen, and the pervasive availability of dual core processors, running applications in virtual machines may slightly degrade the user performance. Therefore, in BotTracer, we can run applications in virtual machines instead. *Lastly*, most bots currently do not detect the virtual machine based honeypots [30].

On the other hand, we prohibit user behavior on the virtual machine in order to make automatically started process self-exposed. If a bot first detects user activities before it launches itself, the current BotTracer would fail to detect such bots. The countermeasure is to synchronize user actions on the host with the corresponding applications on the virtual machine [10].

As always, developing and thwarting bot-like malware is endless arm race. It is foreseeable that some bots with new techniques may evade the detection of BotTracer. For example, if the bot developers use a scheme to identify all bots by labeling each bot with a unique ID when the bot first registers, the botmaster is able to detect the simultaneous arrivals of two bots with the same ID if BotTracer is activated. It is also difficult for BotTracer to detect time-bomb bots. Moreover, as BotTracer relies on known characteristics of bot malware, bots equipped with alternative approaches [23] (e.g., the communication channel is started by user operations when a malicious Web page is accessed) can evade its detection. We are currently trying improve BotTracer for better detection accuracy.

5 Conclusion

Bots and botnets have attracted a lot of attention from both the industry and research communities recently. Detecting bots, however, is still very challenging since bots are very stealthy and bot developers continuously and quickly adopt new techniques to evade detection. In this study, we propose BotTracer to effectively detect bot-like malware on end systems through detecting the bot startup, preparation, and attack behavior during execution. A prototype of BotTracer has been implemented based on VMware and a set of representative bots are tested. The experimental results show that BotTracer is effective for bot detection.

6 Acknowlodges

We thank the anonymous referees for providing constructive comments. The work has been supported in part by the U. S. National Science Foundation under grants CNS-0509061, CNS-0621631, and CNS-0746649.

References

1. http://research.microsoft.com/sn/detours/.
2. http://www.technologynewsdaily.com/node/4859.
3. Convert physical machines to virtual machines. http://www.vmware.com/products/converter/.
4. Enhance netstat - the code project. http://www.codeproject.com/internet/enetstatasp.asp.
5. Malware immunization through deterrence and diversion. http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0650386.
6. One of the most prolific pieces of windows malware has expired. http://news.softpedia.com/news/One-of-the-Most-Prolific-Piece-of-Window%s-Malware-Has-Expired-51466.shtml.
7. Honeyd security advisory 2004-001: Remonte detection via simple probe packet. http://www.honeyd.org/adv.2004-01.asc, 2004.
8. Taxonomy of botnet threats. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibr%ary/botnettaxonomywhitepapernovember2006.pdf, November 2006.
9. P. Barford and V. Yagneswaran. An inside look at botnets, 2006.
10. K. Borders, X. Zhao, and A. Prakash. Siren: Catching evasive malware. In *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, CA, November 2006.
11. Y. Chen. High-performance network anomaly/intrusion detection and mitigation system (hpnaidm). In *ARO-DARPA-DHS Special Workshop on Botnets*, Arlington, VA, June 2006.
12. K. Chiang and L. Lloyd. A case study of the rustock rootkit and spam bot. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
13. W. Cui, R. H. Katz, and W. Tan. Binder: An extrusion-based break-in detector for personal computers. In *Proceedings of USENIX*, 2005.
14. D. Dagon. The network is the infection. http://www.caida.org/projects/oarc/200507/slides/oarc0507-Dagon.pdf, 2005.
15. D. Dagon, C. Zhou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of The 13th Annual Network and Distributed System Security Symposium*, San Diego, CA, Febuary 2006.
16. N. Daswani, M. Stoppelman, the Google Click Quality, and Security Teams. The anatomy of clickbot.a. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
17. F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS)*, September 2005.
18. J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
19. J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
20. G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, Santa Clara, CA, June 2007.
21. A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.

22. D. Kawamoto. Bots slim down to get tough. CNET News.com, November 2005.
23. V. T. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis. Puppetnets: Misusing web browsers as a distributed attack infrastructure. In *Proceedings of ACM CCS*, 2006.
24. A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy. Spyproxy: Execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, August 2007.
25. The Honeynet Project. Know your enemy: Tracking botnets. `http://www.honeynet.org/papers/bots`, March 2005.
26. N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
27. M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of Internet Measurement Conference (IMC)*, Rio de Janeiro, Brazil, October 2006.
28. R. Schoof and R. Koning. Detecting peer-to-peer botnets. `http://staff.science.uva.nl/~delaat/sne-2006-2007/p17/report.pdf`, Feburary 2007.
29. E. Stinson and J. C. Mitchell. Characterizing the remote control behavior of bots. In *Proceedings of DIMVA 2007*, Lucerne, Switzerland, July 2007.
30. P. Wang, S. Sparks, and C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.
31. H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communication Security*, Alexandria, VA, October 2007.
32. C. Zou and R. Cunningham. Honeybot-aware advanced botnet construction and maintenance. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2006.

Appendix A

Figure 3 depicts the profile of the `Internet Explorer` that is generated on a clean machine without user interactions or network accesses.

```
<profile>
    <name>
        Internet Explorer
    </name>
    <description>
        the profile of Microsoft Internet Explorer
    </description>
    <path>
        C:\Program Files\Internet Explorer\iexplore.exe
    </path>
    <starting point>
        www.google.com
    </starting point>
    <registry>
        no
    </registry>
    <file access function>
        getFileSize
    </file access function>
    <file access path>
        C:\Documents and Settings\user\Local Settings
        \Temporary Internet Files\Content.IE5\index.dat
    </file access path>
    <alarm>
        yes
    </alarm>
</profile>
```

Fig. 3. The dormant profile of `Internet Explorer`