A Mulitiprocess Mechanism of Evading Behavior-Based Bot Detection Approaches

Yuede Ji, Yukun He, Dewei Zhu, Qiang Li*, and Dong Guo

College of Computer Science and Technology, Jilin University, Changchun China 130012
{jiyd12,heyk12,zhudw5509}@mails.jlu.edu.cn, {li_qiang,guodong}@jlu.edu.cn

Abstract. Botnet has become one of the most serious threats to Internet security. According to detection location, existing approaches can be classified into two categories: host-based, and network-based. Among host-based approaches, behavior-based are more practical and effective because they can detect the specific malicious process. However, most of these approaches target on conventional single process bot. If a bot is separated into two or more processes, they will be less effective. In this paper, we propose a new evasion mechanism of bot, multiprocess mechanism. We first identify two specific features of multiprocess bot: separating C&C connection from malicious behaviors, and assigning malicious behaviors to several processes. Then we further theoretically analyze why behavior-based bot detection approaches are less effective with multiprocess bot. After that, we present two critical challenges of implementing multiprocess bot. Then we implement a single process and multiprocess bot, and use signature and behavior detection approaches to evaluate them. The results indicate that multiprocess bot can effectively decrease the detection probability compared with single process bot. Finally we propose the possible multiprocess bot architectures and extension rules, and expect they can cover most situations.

1 Introduction

Botnet has become one of the most serious threats to Internet security. A bot is a host compromised by malwares under the control of the botmaster through Command and Control (C&C) channel. A large scale of bots form a botnet. The botmaster can utilize botnets to conduct various cyber crimes such as spreading malwares, DDoS attacks, spamming, and phishing. Bots always try to hide themselves from detection tools to accomplish malicious behaviors.

According to detection location, existing approaches can be divided into two categories: host-based and network-based. (1) Host-based approaches mainly include signature- and behavior-based approaches [1]. Signature-based approaches mainly extract the feature information of the suspicious program to match with

* Corresponding Author.

a knowledge database [2]. Behavior-based detection approaches monitor the abnormal behaviors on hosts to detect bot [3–6]. (2) Network-based approaches mainly analyze network traffic to filter out bot host [7–9].

Among host-based detection approaches, behavior-based approaches are more practical and effective because they can detect the specific malicious process. However, most behavior-based approaches are based on single process or related family processes. If a bot is separated into two or more processes, these approaches will be less effective. Multiprocess bot has two specific features as we proposed in this paper: (1) It can separate C&C connection from malicious behaviors; (2) It can assign malicious behaviors to several processes. As we know, the biggest difference between bot and other malwares is the C&C infrastructure. If the C&C connection is separated from malicious behaviors, the detection approaches correlating network behaviors with malicious behaviors will be less effective. Similarly, if malicious behaviors are assigned to several processes and each process only performs a part of malicious behaviors, the suspicion level may drop to the same with benign process. Thus, malicious behaviors detection approaches will be less effective. If bot can successfully evade existing behavior-based detection approaches, it will cause more threats to Internet security.

Multiprocess malware has been analyzed by some researchers. Ramilli M *et al.* propose the idea of multiprocess malware and prove that the malware divided into several processes will effectively evade the detection of most anti-virus engines [10]. Lejun Fan *et al.* define three important architectures of multiprocess malwares, relay trace, master slave, and dual active mode [11]. Weiqin Ma *et al.* present a new attack, namely "shadow attacks", which divides a malware into multiple "shadow processes" [12]. Experiments indicate that multiprocess malwares can effectively evade the detection of behavior-based detection approaches. Multiprocess bots have been discovered [13], while they have not been studied in detail. If multiprocess bots really explodes, we know nothing about their architectures, communication mechanisms, and other critical knowledge, then they will cause great threats. Thus analyzing them will be very significant.

Our work makes the following contributions:

(1) We identify two specific features of multiprocess bot: separating C&C connection from malicious behaviors, and assigning malicious behaviors to several processes. Then we theoretically analyze why existing behavior-based bot detection approaches are less effective with multiprocess bot according to four categories of behavior-based approaches.

(2) We present two critical challenges of implementing multiprocess bot, and implement a single process and multiprocess bot from a simplified version of Zeus. We use signature and behavior based detection approaches to evaluate them. The results indicate that multiprocess bot can effectively decrease the detection probability. Then we propose other multiprocess bot architectures and extension rules, and expect they can cover most situations.

2 Evasion Mechanism of Multiprocess Bot

We first present the two specific features of multiprocess bots, then analyze why they are able to evade behavior bot detection approaches.

2.1 Specific Features of Multiprocess Bot

There are two specific differences between multiprocess and conventional bot: separating C&C connection from malicious behaviors, and assigning malicious behaviors to several processes. Lejun Fan *et al.* propose a typical multiprocess architecture *master slave mode* [11]. We rename it as *star architecture* using the terminology of network topology. In star network topology, each host is connected to a central hub with a point-to-point connection. Similarly, as shown in Figure 1, process P_1 acts as the central hub, connects with C&C server S and other malicious processes P_2, P_3, and P_4. In Figure 1, the circle denotes benign process, the hexagon denotes malicious process, S denotes C&C server, and P_i denotes different processes. We will analyze the two features using star architecture.

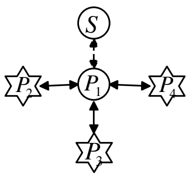


Fig. 1. Star Architecture of Multiprocess Bot

Feature 1: Separating C&C Connection from Malicious Behaviors. The first feature of multiprocess bot is separating C&C connection from malicious behaviors. It means that the process communicating with C&C server has no other malicious behaviors, and the malicious processes do not communicate with C&C server directly. We regard this specific process as server process. We will analyze this feature using the star architecture as shown in Figure 1.

In star architecture, P_1 is the server process that establishes C&C channel with server S. P_2, P_3, and P_4 are the malicious processes. Suppose the botmaster sends a command to the multiprocess bot and we will explore the whole execution procedure. C&C server S sends a command to P_1. In order to evade track techniques like taint analysis [14], the server process can transform data flow dependence into control flow dependence or other obfuscation techniques. After the transformation, the server process sends the command to a certain process using process communication mechanisms. This process performs malicious behaviors in accordance with the command. After execution, the malicious process sends the result data to the server process. The malicious process can also use obfuscation techniques to better evade track techniques. After receiving the result data, the server process sends them to the C&C server.

In this feature, the server process P_1 only has network behaviors, and the malicious processes P_1, P_2, and P_3 only perform a part of all malicious behaviors. Some behavior-based bot detection approaches detect the process which only has network behaviors as benign, and the process without network behaviors will be neglected. Through transforming data flow dependence into control flow dependence or other obfuscation mechanisms, the server process is separated from other malicious processes. Thus this feature is able to evade the detection approaches correlating network behaviors with malicious behaviors. However it may not be able to evade approaches which only detect host malicious behaviors.

Feature 2: Assigning Malicious Behaviors to Several Processes. The second feature is that malicious behaviors are assigned to several processes. Thus, each process only performs a small part of whole malicious behaviors. This feature can effectively evade malicious behavior detection approaches with well designed number of malicious behaviors each process has. Thus, the number becomes a critical challenge. We will use three phases to explain how to define the number and prove that multiprocess bot is able to evade behavior detection.

We utilize the notations in Table 1 to explain this feature. In Table 1, C denotes the critical system call set, a_i denotes the ith system call, and there are k system calls in total. f_i denotes the ith behavior which has $num(f_i)$ system calls, and each one of them is denoted as a^i_j. $num1_i$ denotes the number of critical system calls of each behavior f_i, and $num2_i$ denotes the number of behaviors that critical system call a_i is in.

Table 1. Notations of system calls and behaviors

Description	Set
critical system call set	$C = \{a_1, \dots, a_i, \dots, a_k\}$
system call set of each behavior	$f_i = \{a^i_1, \dots, a^i_j, \dots, a^i_{num(f_i)}\}$
number of critical system calls of each behavior f_i	$Num1 = \{num1_1, \dots, num1_i, \dots, num1_n\}$
number of behaviors that critical system call a_i is in	$Num2 = \{num2_1, \dots, num2_i, \dots, num2_k\}$

Phase 1: Suppose we extract the system calls of known malicious behaviors to build the system call set f_i of each behavior. We build the critical system call set C using the similar methods of building Common API in [15]. The system calls in the critical set are frequently called by these malicious behaviors.

Phase 2: We match every system call a^i_j of each set f_i with critical set C to generate $num1_i$, and after matching all we can get set $Num1$. It denotes the number of critical system calls of each behavior. We match every system call a_i of critical set C with each set f_i to generate $num2_i$, and then $Num2$. It denotes the number of behaviors that call a specific critical system call.

Phase 3: Based on set $Num1$ and $Num2$ we can get two assignment mechanisms: behavior level and system call level assignment mechanism. In behavior level assignment, we sort the behaviors in descending order in set $Num1$. The top behaviors represent the most frequent behaviors. We can separate the top behaviors with each other to average the critical behavior numbers of each process. In this way, we can decrease the malicious grain size of each process. We can assign the sorted set to the processes in S shape, and other assignment mechanisms like arithmetic, random walk, etc. can also be used. Set $Num2$ is about system call level assignment mechanism. We also sort the system calls in descending order. We assign top system calls to different processes or even a process only performing one critical system call. This assignment mechanism is more complicated than behavior level.

We summarized the whole procedure in Algorithm 1. A multiprocess bot using either of them will significantly improve the evasion probability. If a multiprocess bot uses both of them, it will be very difficult to detect. Thus this feature can effectively evade behaviors detection approaches.

Algorithm 1. Process Assignment Algorithm

```
 1. build the system call set $f_i$ for each malicious behavior
 2. build the critical system call set $C$
 3. match each system call set $f_i$ with $C$ to generate $Num1$
 4. match critical set $C$ with each system call set $f_i$ to generate $Num2$
 5. $t$ = number of processes
 6. sort $Num1$ in descending order
 7. for $i$ from 1 to $t$ do
 8.    assign behavior $i, 2*t+1-i, 2*t+i, \dots$ to process $i$
 9. end for
10. sort $Num2$ in descending order
11. for $i$ from 1 to $t$ do
12.    assign system call $i, 2*t+1-i, 2*t+i, \dots$ to process $i$
13. end for
```

2.2 Evading Behavior-Based Bot Detection Approaches

According to detection targets, we classify existing behavior-based bot detection approaches into 4 categories: detecting C&C connections, detecting malicious behaviors, detecting bot commands, and detecting bots (correlating C&C connection with malicious behaviors). Based on the two specific features, we utilize an example approach of each category to analyse why behavior-based bot detection approaches are less effective with multiprocess bot.

Detecting C&C Connections. In this category, detection approaches detect bots based on C&C connections on host and JACKSTRAWS [16] is a typical one. It associates with each network connection a behavior graph that captures the system calls that lead to the connection and operate on returned data.

We use star architecture in Figure 1 to present the evasion procedure. The server process P_1 establishes connection with C&C server S and it will be captured by JACKSTRAWS. P_1 can transform data flow dependence into control flow and distributes the corresponding data to appropriate processes using process communication mechanisms. The malicious processes P_2, P_3, and P_4 perform fine-grained malicious behaviors which have nothing to do with network connections. After finishing the malicious behaviors, they will send the result data to P_1. Then P_1 will upload them to C&C server. In this way, multiprocess bot can separate network connection from malicious behaviors.

According to JACKSTRAWS, the captured network connection alone is not enough for being detected as malicious C&C. What's more, if the connection is encrypted, the detection will be more difficult. They mention three failed detection cases and the first is that the bot process did not finish its malicious behaviors after receiving commands. In this way, multiprocess bot can evade this detection approach.

Detecting Malicious Behaviors. Approaches in this category detect bots based on host malicious behaviors. Martignoni *et al.* propose an typical approach using hierarchical behavior graphs to detect malicious behaviors.

This approach is less effective with multiprocess bot. First, it monitors the execution of one single process, while in multiprocess bot, there are several processes performing malicious behaviors. Specifically, multiprocess bot can evade taint analysis from transforming data flow dependence into control flow, thus this approach is not able to detect any relationship between processes.

Second, the first feature of multiprocess bot is separating C&C connection from malicious processes. As shown in star architecture, S only communicates with the server process P_1. Based on this feature, P_1 only has network behaviors, thus in behavior graphs it is similar with benign network processes. The other malicious processes perform a part of malicious behaviors without C&C connection, thus in behavior graphs they may not be the same with malicious behavior graphs. However, if a process still performs critical malicious behaviors, it can also be detected.

Third, the second feature is assigning malicious behaviors to several processes. As we discussed before, there are two separation mechanisms: behavior and system call level. A multiprocess bot using these two mechanisms can make the event sequence of each process different from any malicious behavior graph. Thus this approach is less effective with multiprocess bot.

Detecting Bot Commands. In this category, detection approaches detect bots based on bot commands. BotTee [15] is a typical approach of identifying bot commands by run time execution monitoring.

BotTee can effectively detect conventional bot commands. However, it has two obvious drawbacks: it monitors the execution of single process; it highly relies on network related system calls. The first drawback is opposite with the second feature of multiprocess bot which assigns malicious behaviors to several processes. BotTee detects bot commands in system call level thus behavior level assignment mechanism is ineffective, while system call level assignment is still effective. The second drawback is opposite with the first feature which separates C&C connection from malicious behaviors. This is a fatal blow to BotTee because they suppose bot command begins with recv or other network reception system calls and ends with send or other network sending system calls.

We use an example to present the evasion procedure. In star architecture, suppose S sends a command to P_1. After P_1 received the command, Deviare API [17] captures the recv command and begins to monitor process P_1. P_1 assigns the command to appropriate process, for example P_2. Then P_2 performs malicious behaviors and sends the result to P_1. Then P_1 sends the result to S, while Deviare API captures the command and triggers Bot Command Identifier. Then Bot Command Identifier analyses the system call sequences between recv and send. And then the sequence will be sent to other components. Thus the sequence of P_1 does not includes malicious behaviors and it will be detected as benign. In this way, multiprocess bot can evade BotTee.

Detecting Bots. Detecting bots means the detection approach correlates malicious behaviors with C&C connection. BotTracer detects bots through three phases: automatic startup, establishment of C&C channel, and information harvesting/dispersion [18].

BotTracer highly relies on the behaviors of a bot process, and multiprocess bot can effectively evade them. We will present the evasion mechanism using star architecture. After the bootstrap phase, P_1, P_2, P_3 and P_4 are flagged as suspicious processes. All the processes of multiprocess bot have to be started automatically, thus they are not able to evade this phase. In the C&C establishment phase, only the server process P_1 establishes C&C channel and other processes communicate with P_1. Thus only P_1 is regarded as suspicious and others can effectively evade this phase. In the last phase, the server process P_1 only communicates with other processes and the C&C server. Thus it can evade this phase because it does not perform malicious activities. In summary, the server process P_1 can evade BotTracer in the last phase, other malicious processes can evade in the C&C establishment phase. Thus multiprocess bots can effectively evade this kind of detection approaches.

3 Critical Challenges of Multiprocess Bot

Although multiprocess bot is able to evade behavior-based bot detection approaches, it still has many critical challenges. We will present two of them: bootstrap mechanism, and process communication mechanism.

Bootstrap Mechanisms. Conventional bots can be started automatically by modifying the bootstrap process list or Registry entries [18]. This is essential for bot to actively initialize C&C channel.Conventional bot which has one process only needs to start itself, while multiprocess bot need to start all the processes. Multiprocess bot may run in the hosts stealthily, while the bootstrap of all the processes is not easy to accomplish stealthily. If the bootstrap mechanism is not well designed, multiprocess bot may be detected at the startup stage. Thus the design of bootstrap mechanisms becomes a critical challenge of multiprocess bot.

Process Communication Mechanisms. Each process of multiprocess bot has to communicate with others to accomplish malicious behaviors together. The communication methods mainly include Interprocess Communications (IPC) and covert channel communication.

IPC mechanisms are common and mainly include clipboard, Component Object Model (COM), data copy, Dynamic Data Exchange (DDE), file mapping, mailslots, pipes, Remote Procedure Call (RPC), and Windows sockets. Covert channel is a computer security attack that can transfer information between processes that are illegal to communicate by the computer security policy. Covert channels are classified into storage and timing channels [19]. A variety of covert channels have been proposed. Aciiçmez *et al.* propose an attack named Simple Branch Prediction Analysis (SBPA) [20], which analyzes the CPU's Branch Predictor states through spying on a single quasi-parallel computation process. Percival demonstrates that shared access to memory caches provides not only an easily used high bandwidth covert channel between threads, but also permits a malicious thread to monitor the execution of another thread [21].

IPC data may be easy to capture, while it may not be easy to identify the suspicious data from the variety benign IPC data. Covert channels are difficult to detect and changeable. Thus the process communication mechanisms make the detection more difficult.

4 Experiments

In order to evaluate the above analyses, we develop a prototype of multiprocess bot from Zeus bot[22]. First, we develop a single process bot, named Mini_Zeus, which is a simplified version of Zeus. Then, we develop a multiprocess version of Mini_Zeus. We use signature and behavior analysis to evaluate them.

4.1 Prototype Architecture

Mini_Zeus is a simplified version of Zeus bot. It has 4 major behaviors: (1) It uses bootstrap mechanisms to make the bot process automatically started. (2) It establishes C&C channel. Thus it can receive commands, execute commands, and send information. (3) It captures http requests of Internet Explorer and sends them to C&C server. (4) It will copy itself to the directory of *system32*, and replace its time stamp with the time stamp of *ntdll.dll*.

The single process version of Mini_Zeus is shown in Figure 2(a). *Mini_Zeus.exe* is the bot infection process. Once started, it will modify Registry to make it automatically start. Then it will use remote thread injection to make *Explorer.exe*

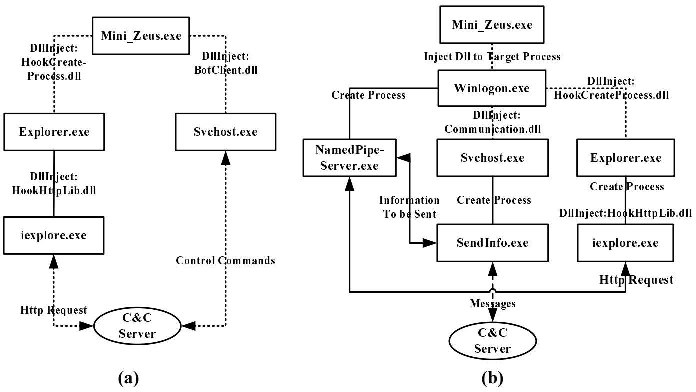


Fig. 2. Architecture of Mini_Zeus

load *HookCreateProcess.dll*, and *Svchost.exe* load *BotClient.dll*. When *HookCreateProcess.dll* is loaded, it will hook function *CreateProcess*. When users try to launch IE browser, *HookHttpLib.dll* will be injected to *iexplorer.exe*. This dll will inject function *HttpSendRequest*, thus the information of post operation will be injected and sent to C&C server. When *BotClient.dll* is loaded, *Svchost.exe* will create a thread to communicate with C&C server. In summary, *Mini_Zeus.exe* is the initial process and *Svchost.exe* is the running bot process.

Multiprocess version of Mini_Zeus is shown in Figure 2(b). *Mini_Zeus.exe* is the bot infection process. Once started, it will use remote thread injection to make *Winlogon.exe* load *Schedule.dll*. *Schedule.dll* firstly modify the Registry to make *Mini_Zeus.exe* automatically started. It has three other major behaviors: (1) It will create *NamedPipeServer.exe*. (2) It will inject *Communication.dll* into *Svchost.exe*. (3) It will inject *HookCreateProcess.dll* into *Explorer.exe*. After these three steps, *NamedPipeServer.exe* will establish a named pipe server and is responsible for receiving and sending information. *Svhost.exe* will create process *SendInfo.exe*. *SendInfo.exe* will establish a named pipe to connect with *NamedPipeServer.exe*. It also establishes C&C channel with C&C server. When *HookCreateProcess.dll* is loaded by *Explorer.exe*, it will hook function *CreateProcess*. Once users try to launch IE browser, *HookHttpLib.dll* will be injected to *iexplorer.exe*. This dll will inject function *HttpSendRequest* and establish a named pipe, thus the information of post operation will be injected and sent to *NamedPipeServer.exe*. In summary, *Mini_Zeus.exe* is the initial bot process, and *NamedPipeServer.exe*, *SendInfo.exe*, and the controlled *iexplore.exe* are the running bot processes.

4.2 Signature Analysis

We use VirusTotal to take a signature analysis of single process Mini_Zeus and multiprocess Mini_Zeus. The results are shown in Table 2, the URL in the table is the ID number and the real url is `https://www.virustotal.com/en/file/URL/`

Table 2. Signature analysis results

File / URL	Detection ratio
Single_Mini_Zeus.exe 71e82907ae2a45fc51071910b7db39a62675b190f26e444b796eb81dbdfad77f	28 / 47
SendInfo.exe 5d86d1fabefb094034e192039a5d75d5f982205b1149f5684bf3e74dc6e63224	6 / 33
NamedPipeServer.exe edb8897344e40b237c7d99ed6f5177f39c0b99f693a7b619e168c667058f0d55	2 / 47
Mini_Zeus.exe 0647dcd190af0e7519f2a4f003a6502e6186776be609c5494fe23cd6335fada5	13 / 47

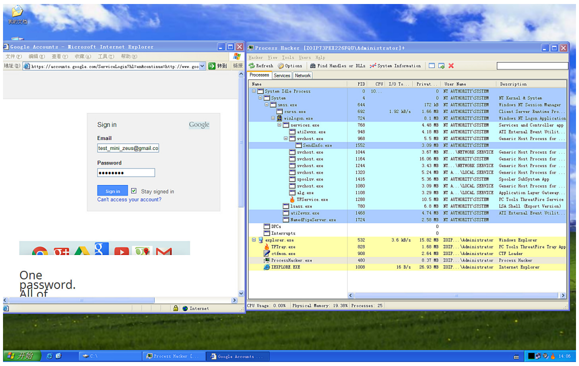
`analysis`. For example, the result of the first url is `https://www.virustotal.com/en/file/71e82907ae2a45fc51071910b7db39a62675b190f26e444b796eb81dbdfad77f/analysis/`.

The single process Mini_Zeus is detected as malicious by 28 of 47 antivirus engines, benign by 19 antivirus engines. Mini_Zeus is detected as benign because it has different signatures with Zeus bot, it is a simplified version and only has the basic functions, and we distribute some malicious behaviors into dll files. In Multiprocess Mini_Zeus, the main process *Mini_Zeus.exe* is detected as malicious by 13 of 47 antivirus engines, as benign by 34 antivirus engines. The other two processes are detected as malicious by 6 and 2. The main process is detected as malicious because it uses remote thread injection. This injection mechanism is a little obvious for antivirus engines, and we believe the number can further decrease if we use different injection mechanisms. These two analysis reports are able to indicate that multiprocess bot can effectively decrease the detection probability compared with single process bot.

4.3 Behavior Analysis

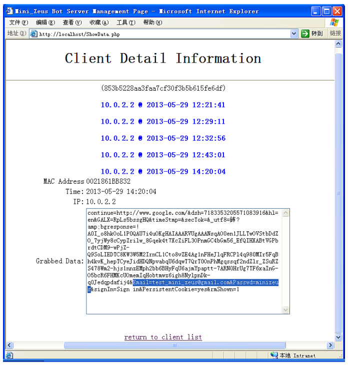
In behavior analysis, we use host-based behavior analysis tool ThreatFire.

ThreatFire is a host-based behavior detection tool, we use it to comparatively analyze our single process and multiprocess version of Mini_Zeus bot. The bot


Google Accounts - Microsoft Internet Explorer
https://accounts.google.com/ServiceLogin?hl=en&continue=http://www.goo
Sign in
Google
Email
test_mini_zeus@gmail.co
Password
•••••••••
Sign in
Stay signed in
Can't access your account?
One password. All of
Process Hacker [ZOIP73PEX226FQU\Administrator]+
Refresh
Options
Find Handles or DLLs
System Information
Processes
Services
Network

```
Name                     PID   CPU  I/O To...  Privat...  User Name                    Description
System Idle Process        0   10...               0      NT AUTHORITY\SYSTEM
 System                    4                       0      NT AUTHORITY\SYSTEM          NT Kernel & System
  smss.exe               644                  172 kB      NT AUTHORITY\SYSTEM          Windows NT Session Manager
   csrss.exe             692       1.92 kB/s  1.66 MB     NT AUTHORITY\SYSTEM          Client Server Runtime Pro...
   winlogon.exe          724                  8.1 MB      NT AUTHORITY\SYSTEM          Windows NT Logon Application
    services.exe         768                  4.48 MB     NT AUTHORITY\SYSTEM          Services and Controller app
     ati2evxx.exe        948                  4.18 MB     NT AUTHORITY\SYSTEM          ATI External Event Utilit...
     svchost.exe         968                  5.5 MB      NT AUTHORITY\SYSTEM          Generic Host Process for ...
      SendInfo.exe      1552                  3.09 MB     NT AUTHORITY\SYSTEM
     svchost.exe        1044                  3.67 MB     NT...\NETWORK SERVICE        Generic Host Process for ...
     svchost.exe        1164                  16.06 MB    NT AUTHORITY\SYSTEM          Generic Host Process for ...
     svchost.exe        1244                  3.43 MB     NT...\NETWORK SERVICE        Generic Host Process for ...
     svchost.exe        1320                  5.24 MB     NT A...\LOCAL SERVICE        Generic Host Process for ...
     spoolsv.exe        1416                  5.36 MB     NT AUTHORITY\SYSTEM          Spooler SubSystem App
     svchost.exe        1080                  3.09 MB     NT A...\LOCAL SERVICE        Generic Host Process for ...
     alg.exe            1108                  3.29 MB     NT A...\LOCAL SERVICE        Application Layer Gateway...
     TFService.exe      1288                  10.5 MB     NT AUTHORITY\SYSTEM          PC Tools ThreatFire Service
    lsass.exe            780                  6.8 MB      NT AUTHORITY\SYSTEM          LSA Shell (Export Version)
    ati2evxx.exe        1468                  4.74 MB     NT AUTHORITY\SYSTEM          ATI External Event Utilit...
    NamedPipeServer.exe 1724                  2.58 MB     NT AUTHORITY\SYSTEM
 DPCs                                          0
 Interrupts                                    0
explorer.exe             532       3.6 kB/s   15.82 MB    ZOIP...\Administrator        Windows Explorer
 TFTray.exe              828                  1.68 MB     ZOIP...\Administrator        PC Tools ThreatFire Tray App
 ctfmon.exe              908                  2.64 MB     ZOIP...\Administrator        CTF Loader
 ProcessHacker.exe       480                  8.37 MB     ZOIP...\Administrator        Process Hacker
 IEXPLORE.EXE           1008       16 B/s     26.93 MB    ZOIP...\Administrator        Internet Explorer

CPU Usage: 0.00%   Physical Memory: 19.38%   Processes: 25
```


Mini_Zeus Bot Server Management Page - Microsoft Internet Explorer
http://localhost/ShowData.php
return to client list
本地 Intranet

Client Detail Information

(853b5228aa3faa7cf30f3b5b615fe6df)

10.0.2.2 @ 2013-05-29 12:21:41

10.0.2.2 @ 2013-05-29 12:29:11

10.0.2.2 @ 2013-05-29 12:32:56

10.0.2.2 @ 2013-05-29 12:43:01

10.0.2.2 @ 2013-05-29 14:20:04

MAC Address	0021861BB832
Time:	2013-05-29 14:20:04
IP:	10.0.2.2
Grabbed Data:	continue=http://www.google.com/&dsh=7183353205571083916&hl=en&GALX=RpLs5bssgHQ&timeStmp=&secTok=&_utf8=鉨?amp;bgresponse=!A0I_o8hkOoL1P0QAU7i4uOKgHAIAAARVUgAAANsqAOOen1JLLTwOVStbDdZO_7yjWy8cCypZrilw_8Gqek4t7XcZiFL30PnmGC4bGm56_EfQIHXABtVGPbrdtCDM9-wPjZ-Q9SoLIEDTC8KW3W5M2IrnCL1Cto8vZE4Ag1nFHeJlqFRCP14q980MIr5FqBh4kvK_hepTCyeJidHDQNpvabqU6dqwT7QrT0OnPhMgqssqf2ndZlr_ZSuRZS478Wm2-hjslnnzEMph2bb6BHyFqU6ajmYpaptt-7ARN0HrUg7TF6xaInG-05bcR6FHMKcUOmemIqHobtmwz6igh8NylpnDk-q0Jedqpdafij4&Email=test_mini_zeus@gmail.com&Passwd=minizeus&signIn=Sign in&PersistentCookie=yes&rmShown=1

Fig. 3. Behavior analysis results

host has the following configurations: Intel Q6600 quad-core processor, 2.40GHz, 2GB RAM, and Windows XP SP3 operating system. ThreatFire and Process Hacker are installed. The server host has the same configurations, with XAMPP and Google Chrome installed. We perform the following experiments:

1. We ran ThreatFire and adjusted its sensitive level to 5 (highest) in the bot host. We ran our bot server program in the server host.

2. For each bot, we ran it to evaluate the kinds of alerts. We all click "Allow this process to continue" to make the bot started.

3. After the bot successfully started, we ran IE browser to test bot behaviors.

The results of our experiments are as follows:

1. In single process Mini_Zeus, there are 5 alerts, one for registering itself in "Windows System Startup" list. The other four are for remote thread injection. Remote thread injection is not able to evade the detection because of its high risk. C&C connection behavior is not detected because we create a thread to connect with bot server.

2. In multiprocess Mini_Zeus, there are 2 alerts for *Mini_Zeus.exe* and they are both remote thread injection. There are no alerts for other processes. There are no alerts for register because we hook winlogon to make it register Mini_Zeus.

3. Both of these two bots can work well without alerts. Figure 3 shows multiprocess Mini_Zeus can successfully capture the post information.

The results indicate that multiprocess Mini_Zeus performs better than single process Mini_Zeus. It can successfully reduce the number of alerts and the risk level, however, there still exists some alerts. In summary, the experiments indicate that multiprocess bot can effectively decrease the detection probability compared with single process bot.

5 Extended Architectures of Multiprocess Bot

Besides star architecture, Lejun Fan *et al.* also present the relay race mode and dual active mode. The relay race mode is the same with the ring in network topology, thus we rename it as ring architecture. Since these two architectures are well suited with network topology, we analyse other network topology architectures and find that multiprocess bot can also adopt these architectures. Thus we present 6 more architectures and 4 extension rules as shown in Figure 4. We hope these architectures with the extension rules can cover most situations.

Architecture 1: Bus Architecture. In bus network, all nodes are connected to a single cable. Similarly, in the bus architecture of multiprocess bot all malicious processes are connected to C&C server. As shown in Figure 4(a), malicious processes P_1, P_2, and P_3 connect to C&C server S. Malicious behaviors are assigned to several processes and each one only performs a part of them.

Architecture 2: Ring Architecture. A ring network is set up in a circular architecture in which data travels around in one direction. Similarly, in the ring architecture of multiprocess bot as shown in Figure 4(b), all the processes form a one direction ring. The data travels along the ring and every process identifies whether the data is for it.

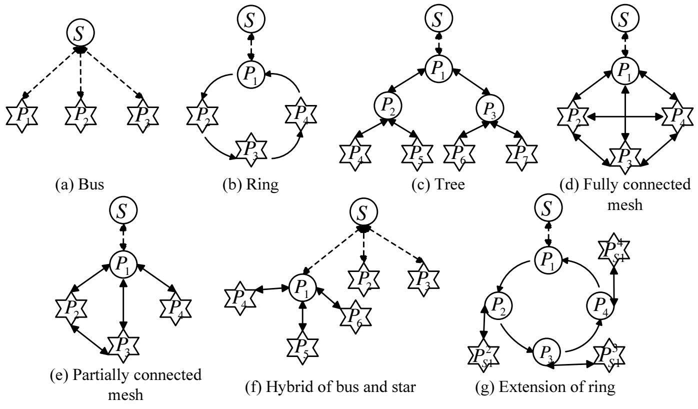


Fig. 4. Architectures of multiprocess bot

Architecture 3: Tree Architecture. Tree architecture is a hierarchical architecture as shown in Figure 4(c). The highest level of this tree is the root process P_1. It communicates with C&C server S. In this architecture, only the leaf processes perform malicious behaviors and other processes are the controller of their child nodes. The data are passed along the tree.

Architecture 4: Fully Connected Mesh Architecture. There are two mesh architectures, fully connected and partially connected. In fully connected mesh architecture, the processes can communicate with each other as shown in Figure 4(d). P_1 communicates with C&C server S and other processes. P_2, P_3, and P_4 perform malicious behaviors and they can communicate with each other.

Architecture 5: Partially Connected Mesh Architecture. In partially connected mesh architecture, some nodes connect with more than one. As shown in Figure 4(e), P_1, P_2, and P_3 connect with each other and P_4 only connects with P_1. This architecture is a subset of fully connected mesh with one specific condition that P_1 should connect with all other processes directly or indirectly.

Architecture 6: Hybrid Architecture. The above 5 architectures and star architecture are the basic architectures, while multiprocess bot can generate more complicated architectures through combining them. For example, we can combine bus with star architecture to generate a new architecture as shown in Figure 4(f). S, P_1, P_2, and P_3 forms the standard bus architecture, S, P_1, P_4, P_5, and P_6 forms the standard star architecture.

Extension Rules. Besides these architectures, we define four extension rules. The server process and malicious processes can create a child process, and we can get the following rules.

(1) Rule 1: The server process communicates with C&C server and its child process communicates with others. (2) Rule 2: The malicious process communicates with others and its child process communicates with C&C server. (3) Rule 3: The malicious process communicates with others and its child process performs malicious behaviors. (4) Rule 4: The malicious process performs malicious behaviors and its child process communicates with others.

Figure 4(g) is an extension architecture of ring using Rule 3. Each of the original malicious processes creates a child process to perform malicious behaviors. Through these rules, the behaviors of all the processes can be minimized and may cause great confusions to behavior-based bot detection approaches.

6 Related Work

Ramilli M *et at.* propose an attack mode named multiprocess malware [10]. If a malware is divided into multiple coordinated processes, no sequence of system calls executed by one process will match the behavioral signatures. Thus this attack mode can evade anti-virus detection tools. However, it also faces many problems, such as the division of malware, the communication of multiple processes, the bootstrp of multiple processes, and the execution sequence of multiple processes. Lejun Fan *et al.* use dynamic analysis approaches to detect privacy theft malware [11]. They also monitor the related processes of suspicious process to discover the collaborative behavior of multiprocess privacy theft malwares. They propose three important architectures of multiprocess malwares, relay trace mode, master slave mode, and dual active mode. Weiqin Ma *et al.* present a new generation of attacks, namely "shadow attacks", to evade current behavior-based malware detections by dividing a malware into multiple "shadow processes" [12]. They analyze the communication between different processes, and the division of a malware into multiple processes. They also develop a compiler-level prototype, AutoShadow, to automatically transform a malware to several shadow processes.

These works all target on multiprocess malwares, however, we target on the attack of multiprocess bot. Although bot is one category of malwares, it has different architectures and features and can cause more serious threat. The architectures are more complicated than others, especially the C&C infrastructure. We propose some specific features of multiprocess bot, and deeply analyze why existing behavior-based approaches are less effective with multiprocess bot.

Virtual machine based malware detection approaches, Holography, Anubis, and CWSandbox , *etc.* can track multiprocess malwares, while these approaches run malwares in an isolated environment. Many novel bots can detect whether they are running in a virtual machine before they perform malicious behaviors. Also, these approaches are not practical for protecting hosts of normal users.

7 Limitations and Future Work

There are several limitations in our work. (1) We theoretically analyzed behavior-based bot detection approaches and did not implement them. There are many challenges when we try to implement them, such as the large-scale data, and the unclear implementation details. If we can implement these approaches to evaluate multiprocess bot we can get a more convincing result. (2) Mini_Zeus is a simplified version of Zeus, and many malicious behaviors are not implemented.

However, the primary behaviors of Zeus are included and more than half anti-virus engines detect it as malicious. The experiment results are still clear. (3) In our experiment about behavior detection, we only use one detection engine to analyze. We will try to use more behavior-based detection approaches to evaluate multiprocess bot.

We are very interested in multiprocess bot and this is a primary work. We will perform the following further works: (1) We will try to implement some behavior-based bot detection approaches, and perform some systematic tests about the concrete reasons why existing behavior-based approaches are less effective with multiprocess bot. (2) We will try to find or implement more instances of multi-process bots to perform a large-scale experiments. (3) We will deeply analyze the advantages and disadvantages of multiprocess bot, and try to find the effective detection approaches about multiprocess bot.

8 Conclusion

In this paper we analyze multiprocess bot in detail. First, we identify two specific features of multiprocess bot, separating C&C connection from malicious behaviors and assigning malicious behaviors to several processes. Based on the two features, we theoretically analyze why existing behavior-based bot detection approaches are less effective with multiprocess bot. After that we present two critical challenges of implementing multiprocess bot. Then we implement a single process and a multiprocess bot. We use signature and behavior based detection approaches to evaluate them. The results indicate that multiprocess bot can effectively decrease the detection probability compared with single process bot. Finally we propose the possible multiprocess architectures and extension rules, and hope they can cover most situations of multiprocess bot.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61170265, Fundamental Research Fund of Jilin University under Grant No. 201103253.

References

1. Silva, S.S.C., Silva, R.M.P., Pinto, R.C.G., Salles, R.M.: Botnets: A survey. Computer Networks (2012)
2. Goebel, J., Holz, T.: Rishi: Identify bot contaminated hosts by irc nickname evaluation. In: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA, p. 8 (2007)
3. Stinson, E., Mitchell, J.C.: Characterizing bots remote control behavior. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 89–108. Springer, Heidelberg (2007)
4. Kolbitsch, C., Comparetti, P.M., Kruegel, C., Kirda, E., Zhou, X., Wang, X.: Effective and efficient malware detection at the end host. In: Proceedings of the 18th Conference on USENIX Security Symposium, pp. 351–366. USENIX Association (2009)

5. Shin, S., Xu, Z., Gu, G.: Effort: Efficient and effective bot malware detection. In: 2012 Proceedings of the IEEE INFOCOM, pp. 2846–2850 (2012)
6. Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., Mitchell, J.C.: A layered architecture for detecting malicious behaviors. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 78–97. Springer, Heidelberg (2008)
7. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, p. 12. USENIX Association (2007)
8. Gu, G., Perdisci, R., Zhang, J., Lee, W., et al.: Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In: Proceedings of the 17th Conference on Security Symposium, pp. 139–154 (2008)
9. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic (2008)
10. Ramilli, M., Bishop, M., Sun, S.: Multiprocess malware. In: 2011 6th International Conference on Malicious and Unwanted Software (MALWARE), pp. 8–13. IEEE (2011)
11. Fan, L., Wang, Y., Cheng, X., Li, J., Jin, S.: Privacy theft malware multi-process collaboration analysis. In: Security and Communication Networks (2013)
12. 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 8(1-2), 1–13 (2012)
13. Microsoft security intelligence report, `http://www.microsoft.com/security/sir/story/default.aspx#!zbot` (accessed November 2013)
14. Schwartz, E.J., Avgerinos, T., Brumley, D.: All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 317–331. IEEE (2010)
15. Park, Y., Reeves, D.S.: Identification of bot commands by run-time execution monitoring. In: Annual Computer Security Applications Conference, ACSAC 2009, pp. 321–330. IEEE (2009)
16. Jacob, G., Hund, R., Kruegel, C., Holz, T.: Jackstraws: Picking command and control connections from bot traffic. In: USENIX Security Symposium (2011)
17. `http://www.nektra.com/products/deviare-api-hook-windows/` (accessed November 2013)
18. Liu, L., Chen, S., Yan, G., Zhang, Z.: Bottracer: Execution-based bot-like malware detection. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 97–113. Springer, Heidelberg (2008)
19. Zander, S., Armitage, G., Branch, P.: A survey of covert channels and countermeasures in computer network protocols. IEEE Communications Surveys and Tutorials 9(3), 44–57 (2007)
20. Aciiçmez, O., Koç, Ç.K., Seifert, J.-P.: On the power of simple branch prediction analysis. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, pp. 312–320. ACM (2007)
21. Percival, C.: Cache missing for fun and profit (2005)
22. Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., Wang, L.: On the analysis of the zeus botnet crimeware toolkit. In: 2010 Eighth Annual International Conference on Privacy Security and Trust (PST), pp. 31–38. IEEE (2010)