Symptoms-Based Detection of Bot Processes

Jose Andre Morales[1], Erhan Kartaltepe[1], Shouhuai Xu[1,2], and Ravi Sandhu[1]

[1] Institute for Cyber Security, University of Texas at San Antonio
{jose.morales,erhan.kartaltepe,ravi.sandhu}@utsa.edu
[2] Department of Computer Science, University of Texas at San Antonio
shxu@cs.utsa.edu

Abstract. Botnets have become the most powerful tool for attackers to victimize countless users across cyberspace. Previous work on botnet detection has mainly focused on identifying infected bot computers or IP addresses and not on identifying bot processes on a host machine. This paper aims to fill this gap by presenting a bot process detection technique based on process symptoms such as: TCP connection attempts, DNS activities, digital signatures, unauthorized process tampering, and process hiding. We partition symptoms into sets which are input into classifiers generating individual detection models which are later appropriately integrated so as to improve the detection accuracy. The integrated approach correctly identified two bot processes and did not produced any false positives and false negatives.

Keywords: Botnet detection, bot process, process symptom, behavior-based detection, symptom-based detection.

1 Introduction

Botnets are an effective tool in spam distribution, denial of service attacks, illegal content hosting and other malicious acts. By leasing botnets, malware authors have successfully implemented profitable business models. These dynamic structures consist of several infected host machines (bots) running the bot software and responding to the bot master's instructions. Previous work on detection has mainly focused on the identification of infected bot machines or IP addresses, and not the actual bot process executing on the infected machine. This research presents three sets of process-based symptoms drawn from known bot samples — bot network activity behavior, unreliable provenance and stealth mechanisms — that are integrated together to detect bot processes on a host machine. Specifically, we make the following contributions:

- The process-based identification of (1) Bot network activity behavior: failed TCP connection attempts, DNS and reverse DNS queries; (2) Process provenance: using static file image digital signature verification and process/file system tampering; (3) Stealth mechanisms: using the absence of a graphical user interface and no required user input to execute.

I. Kotenko and V. Skormin (Eds.): MMM-ACNS 2010, LNCS 6258, pp. 229–241, 2010.

– A formal detection model based on a non-trivial use of established data mining algorithms (C4.5). We conducted a thorough experiment on generating and evaluating detection models. Results show our methodology leads to better detection accuracy for both centralized and Peer-to-Peer (P2P) bots than a straightforward use of established data mining algorithms.

In both centralized and P2P structures, a bot must establish a connection to participate in the botnet possibly producing several failed connection attempts. Bots use DNS activity to reduce failed connection attempts which may instead produce failed DNS activity. In general, a process will attempt to connect to the input IP address of a successful reverse DNS query and the returned IP address of a successful DNS query, concluding the address is active. Our experiments reveal a counterintuitive approach that some bots attempt connecting to IP addresses regardless of DNS activity results: IP addresses that did not return a reverse DNS record are connected to successfully and IP addresses that did return a reverse DNS record failed to connect. Upon host infection, bot activity may manifest in one or more currently running processes. Bot processes may lack a digital signature, or may have been tampered with by a process lacking a digital signature. Bots typically execute without user knowledge by implementing stealth mechanisms, such as lacking a graphical user interface (GUI), not requiring keyboard and mouse input, removing itself from the list of currently active processes, and so on [16].

The rest of the paper is organized as follows: Section 2 is related work, Section 3 presents our bot detection methodology, Section 4 describes the chosen symptoms, Section 5 details the experimentation, results and limitations, and Section 6 gives our conclusion and future work.

2 Related Work

Network-based research analyzing botnets such as [7,2,12] use different techniques characterizing breadth and depth of centralized and P2P botnets, types of performed malicious activities, botnet structures, intrinsic events in the botnet life cycle and hiding techniques. Botnet detection research such as [5,6,1,4,8] primarily analyze network traffic using destination IP addresses, IRC server names, packet content, sequence of intrinsic bot events, crowd response and spatial-temporal relationships in their detection techniques. This results in the identification of several infected host machines as members of a centralized or P2P botnet. The research presented by Zhu et. al. [18] is a host-based detection technique of bots primarily based on a high rate of failed connection attempts. Connection failure rates of known bots are measured against benign processes and show that bots can be identified and distinguished from benign processes based on this single metric. Only measuring failed connection attempts may be most effective with IP addresses of dead botnets, discovered botnets and partially active botnets. However, a single metric is not enough to detect active bots which can possibly lead to the production of false negatives, especially with bots

designed to limit failed connection attempts. Our research uses the novel approach of analyzing failed connection attempts in relation with DNS activity as the basis of our bot network activity behavior symptoms, along with symptoms for unreliable provenance and stealth mechanisms facilitating the identification of several metrics of suspicious processes producing a more robust detection technique. Establishing relationships between observed network data of a process is novel to this research as most related work considered observed network data in an isolated or sequential form. Relating together different observed network behaviors reveals dependencies bot processes have on various network services. Analyzing these dependencies facilitates deeper understanding of bot behavior which may not be appreciable in isolated or sequential analysis of observed network data. Our approach compliments these two forms of analysis and enhances understanding by adding a new perspective on bot behavior.

3 Bot Detection Methodology

Our model's premise is that bot and benign processes will exhibit different recognizable characteristics that can be utilized via appropriate algorithms. The differences may be characterized by a set of attributes mapped to a set of symptoms. Let us denote by A the universe of bot process attributes and by P a process currently executing on a host machine with symptoms $P_{symp_1} \dots P_{symp_{|A|}}$ with respect to A. The goal is to determine the predicate $Bot(P)$, which determines if P is a bot, `true` means "yes" and `false` means "no". We want to identify a function `f` that computes $Bot(P) = \mathtt{f}(P_{symp_1} \dots P_{symp_{|A|}})$. We can approximate the unknown `f` via a function f. A straightforward construction of f would be output of an established data mining algorithm, denoted as f_0. Such f_0 may not offer the desired detection accuracy, inspiring us to propose the following methodology: We can appropriately (1) partition the attributes A into multiple subsets based on certain domain knowledge, (2) generate a function g_i corresponding to the symptoms with respect to each partition of attributes, and (3) create function f based on composing the individual functions g_i.

Specifically, we propose to partition attributes based on the following "lifecycle" perspective of bot processes: A_1, *bot process network activity behavior*; A_2, *bot process provenance*; A_3, *bot process stealth mechanisms*. With respect to A_1, we hope to approximate the predicate $B(P)$, which indicates if P is exhibiting bot network activity behavior via a function g_1. With respect to A_2, we hope to approximate the predicate $U(P)$, which indicates if P has a unreliable provenance via a function g_2. With respect to A_3, we hope to approximate the predicate $S(P)$, which indicates if P has employed stealth mechanisms via specific known techniques via a function g_3. The desired function f can be constructed using g_1, g_2 and g_3 with flexible use of data mining techniques coupled with expert knowledge. We partition the bot process symptoms into three subsets where a symptom represents an occurred execution event or a property that is present during the life cycle of a bot process.

Approximating the predicate $B(P)$ with function $g_1(P)$. Intuitively, $g_1(P)$ analyzes network activity of a process P with a set of symptoms determining if P

is exhibiting similar network activity of known bots. The set of symptoms B_{sym} consist of $n \geq 1$ symptoms where each b_s describes a symptom of network activity previously observed in a known bot sample. P_{val} is a set of m responses θ from a process P which forms a one-to-one mapping with each b_s in B_{sym} and is used to determine if $B(P)$ is `true` or `false`. The values of θ_r are acquired by analyzing the network activity behavior of a process P during execution. The function $g_1(P)$ returns `true` if and only if there exists a value p_r with $\theta_r = true$ corresponding to a symptom b_s in B_{sym}, thus we have:

$$B_{sym} = \{b_1 \dots b_s \dots b_n\}, P_{val} = \{p_1 : \theta_1 \dots p_r : \theta_r \dots p_m : \theta_m\}$$
$$B(P) = g_1(P) = \texttt{true} \Leftrightarrow \exists b_s, p_r : (s = r \wedge \theta_r = \texttt{true}) \quad (1)$$

When $\theta_r = \texttt{true}$, P exhibited the described network activity of symptom b_s. If all θ_r evaluate to `false`, then P does not exhibit bot behavior and $g_1(P) = \texttt{false}$; but if just one θ_r evaluates to `true` then P has the specific symptom b_s and $g_1(P) = \texttt{true}$.

Approximating the predicate $U(P)$ with function $g_2(P)$. Intuitively, $g_2(P)$ compares origin information of a given process P with a set of symptoms $u_s \in U_{sym}$ deciding if the process's provenance is reliable. This predicate asks the question: has the origin of process P been malevolently tampered or created making it unreliable? A response of `true` indicates it is not reliable; `false` indicates it is. The symptoms are a list $u_1 \dots u_s \dots u_n$, $n \geq 1$ submitted to a process P which returns a set of values P_{val}, $p_1 \dots p_r \dots p_m$, and compared with U_{sym}. Each symptom u_s precisely states a singular scenario of process unreliability previously observed in a known bot sample which tampered or created another process in a malevolent manner. Each result $p_r \in P_{val}$ contains an answer $\xi_r = \texttt{true}$ or `false`, which corresponds to the claim $u_s \in U_{sym}$. The function $g_2(P)$ will return `true` if and only if an answer ξ_r of a result $p_r \in P_{val}$ is `true`, thus we have:

$$U_{sym} = \{u_1 \dots u_s \dots u_n\}, P_{val} = \{p_1 : \xi_1 \dots p_r : \xi_r \dots p_m : \xi_m\}$$
$$U(P) = g_2(P) = \texttt{true} \Leftrightarrow \exists u_s, p_r : (s = r \wedge \xi_r = \texttt{true}) \quad (2)$$

If P_{val} returns all `false` answers then it is reliable and $g_2(P) = \texttt{false}$; if just one $p_r \in P_{val}$ has a value $\xi_r = \texttt{true}$, then P's provenance is not reliable and $g_2(P) = \texttt{true}$.

Approximating the predicate $S(P)$ with function $g_3(P)$. Intuitively, $g_3(P)$ determines if a process P is implementing stealth mechanisms previously observed in a known bot sample. The set of symptoms S_{sym} consist of $n \geq 1$ symptoms $s_1 ... s_s ... s_n$ where each s_s describes a specific stealth mechanism previously observed in a known bot sample. P_{val} is a set of m responses μ_z from a process P which forms a one-to-one mapping with each S_s in S_{sym} and is used to determine if $S(P)$ is `true` or `false`. μ_r values are acquired by analyzing the execution behavior of process P for the possible use of known stealth mechanisms. The function $g_3(P)$

returns `true` if and only if there exists a value p_r with $\mu_r = \texttt{true}$ corresponding to a symptom $s_s \in S_{sym}$, thus we have:

$$S_{sym} = \{s_1 \dots s_s \dots s_n\}, P_{val} = \{p_1 : \mu_1 \dots p_r : \mu_r \dots p_m : \mu_m\}$$
$$S(P) = g_3(P) = \texttt{true} \Leftrightarrow \exists s_s, p_r : (s = r \wedge \mu_r = \texttt{true}) \quad (3)$$

The implication of $\mu_r = \texttt{true}$ is that P exhibited the specific known stealth mechanism described in symptom s_s. If all μ_r evaluate to `false`, then P does not exhibit known stealth mechanisms and $g_3(P) = \texttt{false}$; but if just one μ_r evaluates to `true` then P has the specific symptom s_s and $g_3(P) = \texttt{true}$.

Approximating the predicate `f` with function f based on functions g_1, g_2 and g_3. We approximate `f` via a function f by utilizing g_1, g_2 and g_3. Three example definitions to determine $Bot(P)$ are:

$f_1(P) = g_1(P) \vee (g_2(P) \wedge g_3(P))$. This is the least restrictive, since a process is deemed a bot when it exhibits bot network activity behavior or has both an unreliable provenance and has stealth mechanisms. False positives can be produced by benign processes with an instance of bot network activity behavior such as a process with a successful connection attempt to the input IP address of a failed reverse DNS query.

$f_2(P) = g_1(P) \wedge (g_2(P) \vee g_3(P))$. This is more restrictive, the `and` ($\wedge$) operator requires a process to exhibit bot behavior and either unreliable provenance or stealth mechanisms. This will focus detection more on processes with bot-like behavior. False positives can arise with benign processes lacking a digital signature, thereby giving them unreliable provenance, while having an instance of bot network activity behavior such as a failed connection attempt to the input IP address of a successful reverse DNS query. This definition excludes possible detection of bots that possess unreliable provenance and/or stealth mechanisms but do not show bot network activity behavior.

$f_3(P) = g_1(P) \wedge g_2(P) \wedge g_3(P)$. This is the most restrictive with the `and` ($\wedge$) operators requiring triple analysis with each component returning `true`. A process is deemed a bot when it exhibits bot network activity behavior, unreliable provenance and stealth mechanisms. This detection has the highest probability of identifying malicious bots, and excluding benign processes. A process with an unreliable provenance or exhibiting bot behavior or stealth mechanisms is assumed benign which could produce a false negative.

4 Symptoms of Bot Processes

Bot Behavior Symptoms. Evaluating a process's bot network activity behavior employed three symptoms in set B_{sym}. All the symptoms were based on a process P, on a local machine, interacting with and responding to TCP protocol connection attempts and DNS activity (DNS queries and reverse DNS queries).

b_1: Failed connection attempt to the returned IP address of a successful DNS query. This is considered abnormal behavior; a successful DNS query

suggests the returned IP address is active and can establish connections. Many of these IP addresses failing to connect were also the input IP of a failed reverse DNS query.

b_2**: IP address in a successful DNS activity and connection.** This is considered normal behavior. A DNS activity can be either a DNS query or a reverse DNS query. More precisely, we consider the returned IP address of a successful DNS query or the input IP address of a successful reverse DNS query which is also used in a successful connection. In our analysis several more bots than benign processes connected to such IP addresses. This further implies the dependency bots have on DNS activity when attempting connections to remote hosts.

b_3**: Connection attempt to the input IP address of a failed reverse DNS query.** This is considered abnormal behavior; an IP address failing a reverse DNS query should be presumed inactive and should not be used in a connection attempt. Almost all analyzed bot samples performed reverse DNS queries possibly to harvest new domain names of malware servers or infected hosts. Some bots failed to connect with the input IP addresses of a successful reverse DNS query and other bots successfully connected to input IP addresses of a failed reverse DNS query. This counterintuitive use of input IP addresses used in failed reverse DNS requests implies bots attempt TCP connections with IP addresses regardless of DNS activity results for reasons other than TCP connection attempts. One possible motivation may be the reverse DNS query is used solely to dynamically acquire during execution new IP addresses or domain names of malware servers, redirection servers or newly infected victim machines. This helps bots by having to store fewer IP address/domain name pairs in their static file images prior to initial execution; thereby making it harder for security personnel to predetermine the structure and components of a botnet just through static file image analysis.

Unreliable Provenance Symptoms. Determining the provenance of a process employed three symptoms in set U_{sym}. Selection of these symptoms were based on verifying the existence of a static file image's digital signature for known bot and benign files, files of bot's parent process, and analyzing process memory for unauthorized modification by some other process primarily through dynamic code injection. The absence of a digital signature in a file or the parent file that created it raises suspicion due to its unknown origin. All of our bots and a few benign static file images lacked a digital signature. Most of the static file images of benign software installers were digitally signed. Dynamic code injection, mostly a malevolent technique coercing a process into unauthorized behavior [16,15], is frequently used by our analyzed bots on benign processes which then exhibit bot behavior.

u_1**: Standalone executable's static file image does not have a digital signature.** A standalone executable file is written directly to the file system without an installer. Malware contained in email attachments, website downloads and portable memory infect a system this way [16]. The majority of the analyzed benign standalone executables had digital signatures. All of our standalone bot samples lacked a digital signature.

u_2**: Dynamic code injector's static file image does not have a digital signature.** Dynamic code injection is used by bots to infect legitimate processes

[16,3]. There are benevolent uses for this, such as debugging and detection of suspicious activity where the injector's static file image almost always contains a digital signature. Bot injectors typically do not have digital signatures. A process whose injector lacks a digital signature is identified as having an unreliable provenance since the injector's origin cannot be established.

u_3**: Creator of process's static file image does not have a digital signature.** Bots will self-replicate or install other malware on a system [16]. The newly created malware may or may not have a digital signature but the malware installer will likely lack a digital signature, which is considered unreliable provenance. An installed file lacking a digital signature with its installer having a digital signature is considered to have a reliable provenance.

Stealth Mechanism Symptoms. Evaluating a process's stealth mechanisms employed two symptoms in set S_{sym}, all based on a process P's use of graphical user interfaces (GUI) along with reading keyboard and mouse inputs.

s_1**: Graphical user interface.** A vast amount of benign software interact with the user via a GUI. Bots typically do not use a GUI since it calls attention to their existence and may result in their termination [16]. A process executing without a GUI is considered to have a stealth mechanism.

s_2**: Human computer interface.** A benign program may require user input to execute an operation; this is typical interaction between application and user. Bots tend to execute their nefarious acts without the need of explicit user input. A process executing without reading keyboard or mouse events is considered to have a stealth mechanism.

5 Experiment and Results

Data Collection and Instrumentation. Bot data collection was done using VMWare Workstation running Microsoft Windows XP SP2 with no updates and no antivirus. Four active bots: `virut`, `waledac`, `wopla`, `bobax`, and five inactive bots: `nugache`, `wootbot`, `gobot`, `spybot`, `storm`, were executed for a twelve hour period. These centralized and P2P bots possess different stealth mechanisms, diverse command and control channels, various packet encryption and self updates. Packets were captured using `Windows Network Monitor`. Detecting dynamic code injection and Bot replication was accomplished with a real time monitor implementing known techniques [15,11]. Digital signature verification of static file images was done using `Sigcheck` [14]. An enhanced version of `GlobalHook` [10] was used to collect keyboard and mouse input, GUI presence was recorded using `EasyHook` [9]. Collecting data of known benign processes was performed on two verified malware-free desktops running Windows XP SP2 for twelve hours during which both machines performed several network-based activities including web browsing, FTP, instant messaging, P2P file sharing and software updates. The collection, with 20 bot processes and 62 benign processes (41 different applications with some being tested multiple times) listed in Table 1, produced a diversity of symptom combinations. In Table 1, most of the symptoms have {Yes,No} values

Table 1. Bot and Benign Processes Used in the Training Set

Bot Processes									
Bot	Process	Bot Network			Unreliable			Stealth	
Name	Name	Activity Behavior			Provenance			Behavior	
		b_1	b_2	b_3	u_1	u_2	u_3	s_1	s_2
Nugache	mstc.exe	Yes	0	Yes	No	Yes	Yes	No	No
Virut	Svchost.exe	Yes	0	Yes	No	Yes	No	No	No
	Svchost.exe	Yes	0	Yes	No	Yes	No	No	No
	winlogon.exe	No	2	Yes	No	Yes	No	No	No
	svchost.exe	No	1	Yes	No	Yes	No	No	No
	svchost.exe	No	0	Yes	No	Yes	No	No	No
	svchost.exe	No	0	Yes	No	Yes	No	No	No
	svchost.exe	No	2	Yes	No	Yes	No	No	No
Waledac	Save.exe	Yes	123	Yes	Yes	No	No	No	No
Wopla	Rundll32.exe	Yes	1	Yes	No	Yes	No	No	No
Bobax	Explorer.exe	No	2	No	No	Yes	No	No	No
Wootbot	videosd32.exe	Yes	0	No	No	Yes	Yes	No	No
Gobot	Gobot-o.exe	Yes	0	Yes	Yes	No	No	No	No
Spybot	wuaghqr.exe	Yes	0	No	No	Yes	Yes	No	No
Storm	testdll_f.dll	Yes	0	Yes	Yes	No	No	No	No
Bobax	Explorer.exe	Yes	2	Yes	No	Yes	No	No	No
Wopla	Rundll32.exe	No	4	Yes	No	Yes	No	No	No
Waledac	waledac.exe	Yes	7	Yes	Yes	No	No	No	No
Virut	winlogon.exe	No	2	Yes	No	Yes	No	No	No
	svchost.exe	No	2	Yes	No	Yes	No	No	No

Benign Processes			
360tray	Flock	Mercury	Skype
AOL Explorer	Foxmail	MS Messenger	Snarfer
Avant	Google Chrome	Msfeedssync	stormliv
Bittorrent	googlepinylndaemon	Mstc	Svchost
BlogBridge	Internet Explorer	Opera	ThinReader
Btdna	Jusched	Ppstream	ThunderBird
ccApp	Kaspersky AV	RSS Bandit	WinSCP3
Cuteftp32	K-Meleon	RSS Owl	wlcomm
Explorer	LimeWire	Rundll32	wlmail
FeedReader	Maxthon	SeaMonkey	Xdict
Firefox			

with Yes $\mapsto$ `true` and No $\mapsto$ `false`, except in s_1 and s_2 where Yes $\mapsto$ `false` and No $\mapsto$ `true`. Symptom b_2 is considered normal behavior and presented as a total occurrence amount. Test data was collected using five laptops, with minimal security and no recent malware scans, for eight to twelve hours. A post-test data collection malware scan of all five laptops revealed two bot processes: `servwin.exe` as the *cutwail bot*, which was not part of the training set, and `TMP94.tmp` as the *Virut bot*. The test set, listed in Table 2 consisted of 34 processes including two bot

Table 2. Test Set: Decision Tree and Bot Process Predictions

Process Name	Bot Network Activity Behavior				Unreliable Provenance				Stealth Behavior			Bot Prediction			
	b_1	b_2	b_3	$B(P)$	u_1	u_2	u_3	$U(P)$	s_1	s_2	$S(P)$	f_0	f_1	f_2	f_3
svchost.exe	N	0	N	F	N	N	N	F	N	N	T	F	F	F	F
googletalk.exe	N	2	N	F	N	N	N	F	Y	Y	F	F	F	F	F
firefox.exe	N	5	N	F	N	N	N	F	Y	Y	F	F	F	F	F
cutftp32.exe	Y	1	N	T	Y	N	N	T	N	N	F	F	T	T	F
firefox.exe	N	44	N	F	N	N	N	F	Y	Y	F	F	F	F	F
svchost.exe	N	0	N	F	N	N	N	F	N	N	T	F	F	F	F
servwin.exe	Y	0	Y	T	Y	N	N	T	N	N	T	T	T	T	T
Framework Services.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F
iexplore.exe	N	126	N	F	N	Y	N	T	Y	Y	F	T	F	F	F
firefox.exe	N	49	N	F	N	Y	N	T	Y	Y	F	T	F	F	F
rundll32.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F
firefox.exe	N	67	N	F	N	N	N	F	Y	Y	F	F	F	F	F
firefox.exe	N	7	N	F	N	N	N	F	Y	Y	F	F	F	F	F
iexplore.exe	N	54	N	F	N	N	N	F	Y	Y	F	F	F	F	F
firefox.exe	N	45	N	F	N	N	N	F	Y	Y	F	F	F	F	F
firefox.exe	N	10	N	F	N	N	N	F	Y	Y	F	F	F	F	F
SshClient.exe	N	1	N	F	Y	N	N	T	Y	Y	F	F	F	F	F
BitLord.exe	Y	1	N	T	Y	N	N	T	N	N	F	F	T	T	F
Acrobat.exe	N	1	N	F	N	N	N	F	Y	Y	F	F	F	F	F
Thunder5.exe	Y	13	N	T	N	N	N	F	Y	Y	F	F	T	F	F
Thunder Minisite.exe	N	7	N	F	N	N	N	F	Y	Y	F	F	F	F	F
Thunder5.exe	Y	24	N	T	N	N	N	F	Y	Y	F	F	T	F	F
wmplayer.exe	Y	17	N	T	N	N	N	F	Y	Y	F	F	T	F	F
setup_wm.exe	N	1	N	F	N	N	N	F	Y	Y	F	F	F	F	F
chrome.exe	N	3	N	F	N	N	N	F	Y	Y	F	F	F	F	F
TMP94.tmp	N	3	Y	T	N	Y	N	T	N	N	T	T	T	T	T
Google Update.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F
Google Update.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F
chrome.exe	N	28	N	F	N	N	N	F	Y	Y	F	F	F	F	F
Adobe_ Updater.exe	N	2	N	F	N	N	N	F	Y	Y	F	F F	F F	F F	F F
gup.exe	N	1	N	F	N	N	N	F	Y	Y	F	F	F	F	F
Tvanst.exe	Y	1	N	T	Y	N	N	T	N	N	F	F	T	T	F
msfeeds															
sync.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F
zclientm.exe	N	1	N	F	N	N	N	F	N	N	T	F	F	F	F

processes, the rest were assumed benign. One of the bot processes and several of the benign in the test set were not part of the training set.

J48 classification decision trees. The results presented here are partially based on the J48 decision trees [17] in Figure 1. Running the training sets containing bot behavior, unreliable provenance and stealth mechanisms individually produced the decision trees in Figures 1(b), 1(c) and 1(d). Each leaf node, shown as a rectangle, represents the total number of processes classified as exhibiting (`=yes`) or not exhibiting (`=no`) the symptom of the leaf node's parent. A summation of the numeric values in appropriate leaf nodes gives the total number of processes with a (`=yes`) or (`=no`) answer. The bot network activity behavior decision tree in Figure 1(b) produced eight `true` responses with the test set data. The two bot processes were amongst the eight; six false positives and no false negatives were produced. The unreliable provenance decision tree in Figure 1(c) produced eight `true` responses with the test data. The two bot processes were amongst the eight; six false positives and no false negatives were produced. Five processes exhibited unreliable provenance symptom u_1 and three processes exhibited unreliable provenance symptom u_2. Two of the three processes with symptom u_2 were purposely injected (see paragraph below **The cases of** f_0,f_1,f_2 **&** f_3). The stealth mechanisms decision tree in Figure 1(d) produced ten `true` responses with the test data. The two bot processes were amongst the ten; eight false positives and no false negatives were produced. The high amount is a result of having many system and software update processes in the test set that are known to run without a GUI. The two bot processes had no GUI which is assumed implemented as part of a larger stealth strategy [16].

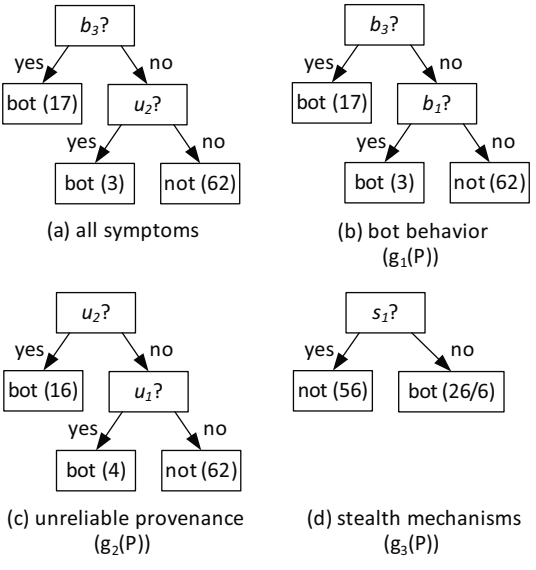

Fig. 1. J48 Decision Trees Used in f_0, f_1, f_2 and f_3

The cases of f_0,f_1,f_2 & f_3. Evaluating the test data results of bot behavior $B(P)$, unreliable provenance $U(P)$ and stealth mechanisms $S(P)$ with f_0, f_1, f_2, and f_3 are listed in Table 2 along with final bot predictions. The case of f_0 is a simplistic use of the J48 classifier. Using all the symptoms to analyze the training set data produced the decision tree in Figure 1(a) with no false positives and no false negatives. Analyzing the test set data with this decision tree produced two false positives and no false negatives, listed in Table 2. `RemoteDLL` [13] is a benevolent utility which lacks a digital signature that loads and removes DLLs from a process. In our test set, processes `iexplore.exe` and `firefox.exe` were purposely DLL injected using RemoteDLL producing two false positives with the decision tree in Figure 1(a) since the injector had no digital signature. In the case of f_1, our test set produced eight `true` responses including the two known bots, leaving six false positives and no false negatives. All eight exhibited bot behavior including the two bots which were also the only ones exhibiting unreliable provenance and stealth mechanisms. Bot behavior was highly prevalent, partly due to benign network active processes executing combinations of DNS activity with connection attempts. In the case of f_2, our test set produced five `true` responses including the two known bots, with three false positives, an improvement over f_1. All five exhibited unreliable provenance but only the two bots exhibited stealth mechanisms as well. Only the bots possessed additional symptoms, hinting more accurate performance can be made with stronger restrictions. In the case of f_3, our test set produced only two `true` responses: our two discovered bot processes. Perfect results were yielded by f_3 suggesting accurate detection with minimal false positives and false negatives may be achieved with high restriction enforcement.

Discussion. Only five of the eight symptoms, b_1, b_3, u_1, u_2, s_1, composed the decision trees and were used in the final bot predictions. Symptom s_1 was the most dominant with thirteen processes in our test set executing without GUI. This is not surprising as several tested benign processes were system services running in the background. Symptom b_1 occurred often due to processes failing to connect with the returned IP address of a successful DNS query. Symptom b_3 only occurred in the two test set bots, suggesting that a well designed benign application will not attempt to connect to IP addresses involved in a failed reverse DNS query while bots attempt connections regardless of DNS activity results. According to Table 2, only the bot processes `servwin.exe` (*cutwail bot*) and `TMP94.tmp` (*Virut bot*) possessed more than one bot behavior symptom. This hints to strong dependencies on DNS activities by bots and higher probability to attempt connections with IP addresses involved in DNS activities. Symptom u_1 occurred often due to our test set processes lacking digital signatures. One can assume that a portion of benign applications and the vast majority of malware will lack a digital signature. Both symptoms s_1 and s_2 precisely matched for each process in the test set, meaning every process executing with a GUI also read user input and every process without a GUI did not read any user input. Ranking from least effective to most effective detection produces: f_1, f_2, f_0, f_3. Even though f_0 was second most effective in bot detection, given a more diverse test set the straightforward construction of f_0 may not be so effective, as shown by our purposeful injection of two processes during

testing. Detecting the most devious of bots may be best achieved with f_3 but f_2 may capture a broader range of bots possessing less symptoms. A combination of the restrictions of f_2 and f_3 may be best suited for bot detection and combining restrictions of f_1 may be the best to detect other non-bot malware.

Limitations. IP addresses of DNS activity not used in a connection attempt by a captured process were not analyzed since we could not reliably map specific DNS activity with a specific process. Only Win32 processes were analyzed while kernel processes were not. We are currently developing utilities eliminating these limitations allowing their inclusion in our evaluations.

6 Conclusion and Future Work

We presented in this research a symptoms-based technique for detecting bot processes using three distinct user defined sets of symptoms drawn from known bot samples: bot network activity behavior, unreliable provenance and stealth mechanisms. Through a non-trivial use of J48 classifier, three distinct evaluations were performed correctly identifying two bot processes. Bot network activity behavior symptoms were based on failed connection attempts and DNS activity; provenance symptoms were based on the existence of digital signatures and process/file system tampering; stealth mechanisms were based on the absence of a GUI and no required reading of user input. Several of the chosen symptoms appeared in both benign and bot processes, but the bot processes showed a much higher quantity and diversity of symptoms. Based on the results, the strongest restrictive analysis requiring symptoms of all three sets was the best singular detection solution producing no false positives and no false negatives. In dealing with future bots and other non-bot malware combining stronger and weaker restrictions may be a desirable detection approach. Future work includes analyzing kernel mode bots and a diverse set of network protocols, as well as a kernel-based real-time monitor detecting presence of bot processes.

Acknowledgments

This work is partially supported by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund.

References

1. Collins, M.P., Shimeall, T.J., Faber, S., Janies, J., Weaver, R., De Shon, M., Kadane, J.: Using uncleanliness to predict future botnet addresses. In: IMC 2007: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp. 93–104. ACM, New York (2007)
2. Dagon, D., Gu, G., Lee, C.P., Lee, W.: A taxonomy of botnet structures. In: Computer Security Applications Conference, Annual, pp. 325–339 (2007)
3. Filiol, E.: Computer Viruses: from Theory to Applications. IRIS International series. Springer, Heidelberg (2005), iSBN 2-287-23939-1

4. Goebel, J., Holz, T.: Rishi: identify bot contaminated hosts by irc nickname evaluation. In: HotBots 2007: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets, p. 8. USENIX Association, Berkeley (2007)
5. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th USENIX Security Symposium, Security 2008 (2008)
6. Gu, G., Zhang, J., Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008) (February 2008)
7. Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: LEET 2008: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, pp. 1–9. USENIX Association, Berkeley (2008)
8. Hu, X., Knysz, M., Shin, K.G.: Rb-seeker: Auto-detection of redirection botnets. In: 16th Annual Network and Distributed System Security Symposium (2009)
9. Husse, C.: Easyhook 2.6, `http://www.codeplex.com/easyhook`
10. Mamaladze, G.: Globalhook, `http://www.codeproject.com/KB/cs/globalhook.aspx`
11. Morales, J.A., Clarke, P.J., Deng, Y., Kibria, B.G.: Identification of file infecting viruses through detection of self-reference replication. Journal in Computer Virology Special EICAR Conference Invited Paper Issue (2008)
12. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: 3rd International Conference on Malicious and Unwanted Software, MALWARE 2008, pp. 24–31 (2008)
13. Remote dll injection application, `http://www.novell.com/coolsolutions/tools/17354.html`
14. Sigcheck 1.6, `http://technet.microsoft.com/en-us/sysinternals/bb897441.aspx`
15. Sun, H.M., Tseng, Y.T., Lin, Y.H., Chiang, T.J.: Detecting the code injection by hooking system calls in windows kernel mode. In: 2006 International Computer Symposium, ICS 2006 (2006)
16. Szor, P.: The Art of Computer Virus Research and Defense. Symantec Press & Addison-Wesley (2005)
17. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
18. Zhu, Z., Yegneswaran, V., Chen, Y.: Using failure information analysis to detect enterprise zombies. In: 5th International ICST Conference on Security and Privacy in Communication Networks, Securecomm 2009 (2009)