Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods

Elizabeth Stinson
Stanford University

John C. Mitchell
Stanford University

Abstract

Automated bot/botnet detection is a difficult problem given the high level of attacker power. We propose a systematic approach for evaluating the evadability of detection methods. An evasion tactic has two associated costs: *implementation complexity* and *effect on botnet utility.* An evasion tactic's implementation complexity is based on the ease with which bot writers can incrementally modify current bots to evade detection. Modifying a bot in order to evade a detection method may result in a less useful botnet; to explore this, we identify aspects of botnets that impact their revenue-generating capability. For concreteness, we survey some leading automated bot/botnet detection methods, identify evasion tactics for each, and assess the costs of these tactics. We also reconsider assumptions about botnet control that underly many botnet detection methods.

1 Introduction

Since malicious botnets are a relatively new security threat as compared to viruses [2] and worms [1], it is an opportune time to establish an extensible framework that would enable comparisons across current and future bot/botnet detection methods.

Comparing Detection Methods: Comparisons can be made on the basis of an automated detection method's false negatives against current bots, false positives against a representative sample of benign programs, and practicality, including performance impact. Also of interest is the time required to detect a malware instance as this identifies the method's opportunity cost. Since we presume an adaptive adversary, we would also like to evaluate evadability.

Our Contributions: We propose a systematic framework for evaluating a key consideration in assessing the fitness of a detection method: its evadability. Our estimation of the amount and difficulty of work required to apply an evasion tactic is based on the ease with which current bots can be incrementally modified. We also identify characteristics of botnets that impact their revenue-generating capability; this enables us to evaluate the extent to which evasive bot modifications result in a less useful botnet. We assume that the attacker has precise details of the detection algorithm and its implementation. For brevity, we do not currently consider the reconnaissance effort required to identify the values of a detection algorithm's parameters (e.g., time thresholds), which may be tuned per installation.

Organization: Section 2 describes bots and their implementations, control, and attacks. We propose a framework for quantifying evasion tactics in sect. 3. Some leading botnet detection methods and the characteristics on which they depend are presented in sect. 4. Some evasive tactics and the detection methods they defeat are described in sects. 5 and 6, respectively. We reconsider some assumptions underlying current botnet detection methods in sect. 7.

2 Bots

2.1 Definition of a Bot

Since we explore ways that a bot can be changed in order to defeat detection, we need some baseline notion of what constitutes a bot. A bot (I) participates in a command-and-control (C&C) network, through which the bot receives commands (II) which cause the bot to carry out attacks. We do not impose temporal constraints on when the attack must be carried out relative to command receipt nor do we constrain command format. Our bot definition is more general than the one proposed in [13], which required (explicitly or implicitly) that botnet attacks be performed in a coordinated fashion and be network-detectable.

2.2 Bot Implementations

A typical bot implementation consists of two independent engines: a C&C-communication-protocol processor and a *command interpreter*, which interprets and executes bot commands, i.e., implements

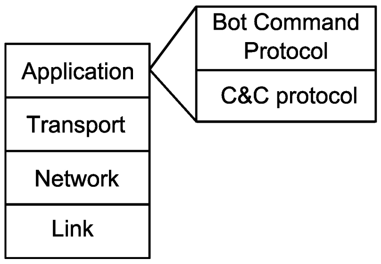

Figure 1: Historically, bots live at the application layer of the TCP/IP stack, which can be further subdivided into the C&C and bot command layers.

the bot's protocol. Figure 1 provides an abstraction of the structure of current malicious bots, which live at the application layer of the TCP/IP stack. A bot protocol message (i.e., command) is generally encapsulated as the payload of a C&C communications protocol message. A bot's *command syntax* encodes the actions the bot can perform as well as the ways in which each can be invoked (i.e., the parameters).

2.3 Botnet Control

Botnet control is achieved through a *C&C network*, which consists of the: *C&C protocol*, which defines communication format, *network topology*, which identifies who talks to whom, and *rendezvous point* (RP), the location to which commands are delivered. The historical view of botnets is that they are *tightly controlled*: the bot master sends a command that is received and executed by all listening bots more or less immediately [4], e.g., IRC botnets which have latency on the order of seconds. Peacomm [6]'s use of P2P for C&C demonstrates a looser control model which has higher latency since commands percolate through a distributed network as bots poll for them.

2.4 Botnet Attacks

Botnet attacks include click fraud, distributed denial-of-service (DDoS), identity theft (harvesting the infected host's file system or registry and/or keylogging), malware distribution (e.g., pay-per-install spyware), phishing, piracy (product or license key theft), proxying, scanning, server hosting (e.g., SMTP, HTTP, C&C), spamming, and spreading.

3 The Cost of Evasion

The cost of evading a detection method has two components: implementation effort and effect on botnet utility. Presumably, evasive tactics that are easy to implement and have little negative effect on botnet utility are more attractive (to the bot master) than those which require more work to achieve or reduce a botnet's usefulness. Understanding and quantifying these costs enables us to systematically evaluate the strength of automated detection methods.

Implementation Complexity: Our rubric is based on the ease with which bot writers can incrementally modify current bots to evade detection. This approach to measuring implementation effort tracks with the manner in which bot variants are generated by their installers. *Low*: Tactic can be applied without source modification; e.g., via command selection or simple binary changes, such as packing. *Medium*: Tactic requires source modification which can be achieved via bot-development kits, such as agobot's `configgui.exe` which provides a point-and-click interface for specifying bot configuration details and builds the requested bot. *High*: Tactic requires minimal direct source modification, e.g., modifying a bot to append junk data to its communications. *Very High*: Tactic requires extensive or complicated direct source modification, e.g., changing the C&C protocol or implementing use of covert channels.

Botnet Utility: Quantifying botnet utility is a substantial research question. We identify botnet characteristics that affect its market value. Calculating overall utility would ideally combine these characteristics in a manner which reflected the price at which a botnet can be rented to perform attacks.

- *Diversity of attacks*: The number of different attack types in which a botnet can be used. Attacks could be weighted differently so as to reflect their relative lucrativeness.
- *Lead time required to launch an attack*: This captures whether bots are available in real-time as well as latency through the C&C network.
- *Botnet size*: Given the required lead time, the number of bots expected to participate in an attack. May be sensitive to diurnal effects.
- *Attack rate*: The maximum number of attacks per hour in which a botnet can be used. May be affected by required lead time.
- *Synchronization level*: Identifies the upper bound on the difference in time between the actions of the first and last bots participating in an attack. Is relevant for types of attacks that require synchronized bot execution (e.g., DDoS).

Calculating Overall Evasion Cost: If evading a detection method requires simultaneously applying multiple evasion tactics $\{E_1, E_2, ..., E_n\}$ then the resulting *evasion strategy*'s implementation complexity is obtained by combining each E_i's complexity using standard principles from complexity theory. The strategy's effects on botnet utility could be obtained by summing the effects of each E_i. An interesting question is, how to combine the two components of a strategy's cost? In particular, how to weight implementation complexity relative to effects on botnet utility. Moreover, some effects on utility may be considered more detrimental than others. Finally, selecting an optimal evasion strategy for a method could be made with an eye toward simultaneous evasion of multiple methods — so as to amortize implementation effort. We describe our weights in sect. 6.

4 Automated Detection Methods

To consider the problem of evasion, one must have an understanding of the detection landscape. In Table 1, we identify some botnet characteristics on which various detection methods rely in whole or in part. Table 2 identifies some leading detection methods and their dependencies — the first step in devising a plan for evasion. Some methods OR-wise combine characteristics; in which case, suppressing a single characteristic would not suffice for evasion. Additional method details can be found in App. A.

Related Terminology: *Information flow tracking* or *tainting* entails marking data from certain sources as tainted and propagating taint across select operations. A *communication flow* is a four-tuple consisting of the source and destination IP addresses and port numbers. Flow characteristics include: packets per flow (ppf) and average: bytes per packet (bpp), bytes per second (bps), and packets per second (pps). For aggregated flows, characteristics include: flows per address (fpa) and flows per hour (fph).

5 Evasive Techniques

Each surveyed method can be evaded by applying one of the following four tactics. We identify other evasive techniques for these methods to inform our discussion on ranking evasion strategies in sect. 6. We provide each tactic's expected implementation complexity and effects on botnet utility. For simplicity, we only consider first-order or direct effects.

5.1 Tactic #1: Encrypt Traffic

Methods which rely on *Syntax* are vulnerable to evasion via communications encryption which can be applied at any of the layers in the stack as in fig. 1. Because a bot which performs encryption can be created using a bot-development kit, the implementation complexity of this tactic is Medium. Encrypting traffic does not negatively affect a botnet's utility.

5.2 Tactic #2: Threshold Attacks

For methods which rely on *Time*, malicious activities (which correspond to NIDS events) can be spread out in time so as to fly beneath the radar. Since most of the events correspond to bot actions, simple source modifications can be made to the bot which cause it to inject a bit of delay between events; e.g. after finding a vulnerable host but before infecting that host or after receiving but before executing a command. The implementation complexity is High. Applying this tactic reduces a botnet's attack rate.

5.3 Tactic #3: Launder Tainted Data

For methods which rely on *Taint*, a bot can defeat detection via "laundering" tainted data using at least three techniques. First, if there are channels across which taintedness is not propagated (e.g., persistent storage), the bot can write the tainted data to such channels then immediately read the data back from the same channel. Secondly, if only explicit information flow is tracked, laundering can be performed across control flow operations. In particular, the bot could invoke the following method on all data received over the network and subsequently use the `dst` buffer which will not be considered tainted (and yet whose contents are identical to those of `src`):

```
void launder(char* src, char* dst, int n) {
  for (int i = 0; i < n; i++) {
    char c = 0;
    switch( src[i] ) {
      case 'a': c = 'a'; break;
      case 'b': c = 'b'; break; ...
    } // end of laundering char
    dst[i] = c; // write laundered char
  } }
```

An approach to implicit information flow analysis as in [9] entails statically determining all instructions whose execution depends on a tainted branch condition and marking the destination operands of those instructions as tainted. A bot, however, could devise a command syntax that used tainted data in a man-

Table 1: Description of some botnet characteristics upon which automated detection methods rely.

Characteristic	Description
Basis	Type of method as in host- or network-based
Hub	Relies on network topology where single server has multiple clients
IRC	Relies on specific IRC port number or model of communications patterns
Flow-Chars	Uses flow characteristics to correlate C&C communications and/or attacks
Time	Correlates events or network traffic that occur within a time window
Net-Det	Relies on automated, network-based detection of botnet attacks such as scanning
Syntax	Relies on bots' use of a particular nickname, command, or protocol syntax
Taint	Requires that bots' execution of commands demonstrates explicit information flow

Table 2: Automated botnet detection methods (in chronological order) and some characteristics on which each depends partially or wholly.

Method	Basis	Hub	IRC	Flow-Chars	Time	Net-Det	Syntax	Taint
Strayer [4]	Net-	No	Yes	bpp, bps, pps, etc.	Yes	No	No	No
Rishi [5]	Net-	No	Yes	No	No	No	Yes	No
Karasaridis [7]	Net-	Yes	Yes	bpp, fpa, ppf	No	Yes	No	No
BotSwat [8]	Host-	No	No	No	No	No	No	Yes
BotHunter [10]	Net-	No	Yes	No	Yes	Yes	Yes	No
BotMiner [13]	Net-	No	No	bpp, bps, fph, ppf	Yes	Yes	No	No

ner that was indistinguishable from benign programs' use of tainted data. In particular, rather than a command being a string that is sent as a packet payload, each letter of the command could be encoded by the packet's length (or by other malleable fields of lower-layer protocol packets). Upon invocation of `recv`, the bot would check `recv`'s return value which identifies the number of bytes received; from this the bot would recover the corresponding letter. The rub is that normal use of `recv` entails branching on its return value to select the appropriate next steps.

Thirdly, bots can use covert channels to obscure their responsiveness to network-derived data. The implementation complexity of these techniques ranges from High to Very High. The effects on botnet utility vary from None to increasing the required lead time for techniques that entail sending multiple packets to accomplish what was formerly achieved by one.

5.4 Tactic #4: Perturb Flows

For methods which rely on *Flow-Chars* to filter, classify, and/or correlate traffic, perturbing flows provides a rich avenue for evasion. Regardless of botnet topology, we can obscure similarities between flows by modifying sender and/or receiver behavior. Most flow correlation considers flows during a particular time window, which implicitly defines the set of flows to target for perturbation. Also, flow correlation operates with respect to a threshold; e.g., the minimum number of flows which must be correlated or the closeness required for correlation. Our assumption is that the command channel is idle most of the time; hence, increased channel utilization does not adversely affect the botnet's ability to launch attacks.

We can defeat correlations on the basis of bytes per packet, bytes per second, packets per flow, and packets per second by inserting a random number of junk bytes into each packet and/or junk packets into each flow. By striping a command across packets, we can increase the range of packet sizes. We can defeat correlations based on flows per address or flows per hour by having bots stay connected, disconnect, and reconnect at random intervals rather than using a long-lived persistent connection model. Noise injection does not affect botnet utility whereas perturbing flow durations may affect botnet size since bots are not consistently connected to the C&C network.

5.5 Other Tactics

Below we identify other techniques that could be used against the surveyed methods.

- *Only perform a subset of attacks*: For methods which rely on *Net-Det* to distinguish botnets from other tightly-controlled overlay networks, evasion can be achieved by confining the botnet's attacks to those which have marginal network-visible effects. In practice, this means that the botnet can perform all attacks except DDoS, scanning, spamming, and spreading. Since the attacks in which a botnet is used are determined by commands sent to that botnet, the implementation complexity of this approach is Low. Applying this tactic results in a botnet with decreased attack diversity.

- *Restrict attack targets*: For methods which rely on observing traffic at the network boundary (as inbound or outbound flows), e.g., [10, 13], one can restrict botnet communications or attacks to target hosts on the same internal network. Since attack targets are specified as command parameters, this has Low complexity and results in decreased botnet size for any given attack.
- *Induce IP churn*: For methods [4, 7, 10, 13] that correlate a host's activities over time using the host's IP, a bot that can obtain a different IP address on demand will defeat such correlations. The implementation complexity of this is unknown and it would not affect botnet utility.
- *Defeat cross-host clustering*: Cross-host clustering entails identifying when some threshold of hosts demonstrates similar behavior. Such correlations can be defeated by having bots participate in different attacks. Moreover, which bots participate in which attack at which time can be biased so that bots in the same administrative domain are less likely to participate in the same attack. The complexity is High and this would reduce the number of bots involved in an attack.
- *Coordinate bots out-of-band* (OOB): There are indirect ways to achieve coordinated bot response, including synchronization on a source other than command receipt (High to Very High) and fast-flux (Low), which entails frequently re-mapping a host name's DNS record to different IP addresses. For example, Storm bots synchronize via the Network Time Protocol [12] and fast flux is used in phishing attacks. These tactics do not affect botnet utility.
- *Attack the process monitor*: For systems that monitor process behavior [8], the bot can halt execution, escape the sandbox, or attack the monitor. The implementation complexity of this is Very High; it does not affect botnet utility.

6 Choosing an Evasion Approach

We weighted effects on botnet utility more highly than implementation complexity and sought to apply the fewest tactics. For this reason, even though Karasaridis and BotMiner can be evaded by restricting a botnet's diversity of attacks (which has Low implementation complexity), we favored evasion tactics that required more work to implement but which did not result in this hit to botnet utility. Each effect on botnet utility was weighted equally. Table 3 identifies the resulting optimal evasive strategy for each method as discussed below.

Defeating Strayer: The method in [4] consists of three stages, each of which uses flow characteristics: filtering traffic unlikely to be bot C&C (# of packets, bps, packet size, duration), classifying traffic as likely to be IRC or not (duration, role, bpp, bps, pps), and clustering related flows (characteristics relating to inter-arrival time and packet size). The output of each stage becomes the input for the next. Flow perturbation could be used to defeat each stage; the simplest approach targets the filtering of high bitrate flows via injecting packet- and flow-level noise.

Defeating Rishi: The cheapest approach to defeating the method in [5], which identifies IRC packets with client nicknames that match pre-specified templates, entails encrypting communications traffic. Other approaches include changing the C&C protocol or the syntax of nicknames or IRC keywords.

Defeating Karasaridis: The method in [7] identifies "suspected bots" as hosts involved in spamming, scanning, DDoS, or sending viruses in email. All flow records for suspected bots are fetched then pruned — keeping flows: whose server port is one of the standard IRC ports or which involve a hub server. Of these, the flow records for <server_ip, server_port> tuples which have the most suspected bots are aggregated. Those which sufficiently resemble the model for IRC in terms of average fpa, bpp, and ppf undergo heuristics analysis which entails identifying the number of peers, idle clients, etc. By injecting packet- and flow-level noise, we can perturb bpp and ppf. Since all three flow characteristics are equally weighted, this likely suffices to break similarities between C&C traffic and the model for IRC. Employing a connect-disconnect-reconnect model would perturb fpa. Other approaches include limiting the botnet's attack types, inducing IP churn, using OOB coordination, using C&C protocols other than IRC, and thwarting identification of hub servers.

Defeating BotSwat: The implementation in [8] could be evaded by certain uses of communications encryption and attacking the process monitor. Since

Table 3: The surveyed methods and an optimal evasive tactic which could be used to defeat each as well as the tactic's implementation complexity and effects on botnet utility, as applicable.

Method	Evasive Tactic	Implementation Complexity	Effects on Utility
Strayer [4]	Inject packet- or flow-level noise	High	None
Rishi [5]	Encrypt C&C traffic	Medium	None
Karasaridis [7]	Inject packet- or flow-level noise	High	None
BotSwat [8]	Launder tainted data	High	None
BotHunter [10]	Attack time thresholds	High	$\downarrow$Attack rate
BotMiner [13]	Inject packet- or flow-level noise	High	None

the method relies on explicit information flow to identify execution of bot commands, it is generally susceptible to data laundering.

Defeating BotHunter: The method in [10] uses five different NIDS event types; certain combinations of these events seen within a fixed time window (four minutes in their evaluation) cause alarm generation. The most straight-forward approach to evasion entails spacing events out in time such that the necessary combinations do not occur sufficiently closely together. There are myriad other approaches, however, which entail targeting NIDS detection of the various events. For example, the Snort rulesets used to detect C&C communications rely largely on content-based matching and hence could be evaded via encrypting or changing the syntax of messages. Scan detection can be foiled by restricting attack targets or rate-limiting. Inducing IP churn is another tack.

Defeating BotMiner: The method in [13] identifies hosts with similar communications activity; i.e., hosts whose flows are similar in terms of bpp, bps, ppf, and fph. In parallel, they identify hosts with similar attack traffic; i.e., that are scanning the same ports, spamming, or downloading similar files. They perform cross-planar correlation to identify hosts with both similar communications and attack traffic. An optimal approach to evasion is to inject packet- and flow-level noise such that communications-plane clustering will come up empty-handed. Other approaches include defeating scan detection, foiling binary-download clustering, and restricting communications and attack targets.

7 Discussion

We believe that much research into automated network-based botnet detection has operated from an assumption that may have been historically valid but is less true of botnets today. In particular, the *tight-control myth* is that (I) for a botnet attack to be effective, bots must simultaneously participate (coordinate) in attack execution and (II) that this coordination must come through the C&C network. We addressed (II) in 5.5, which identified ways to achieve bot coordination outside of the C&C network. Below, we consider (I), i.e., the extent to which each botnet attack type relies upon the simultaneous, coordinated (synchronized) execution of the bots.

To determine the necessity of bot synchronization for each type of botnet attack, we may ask: is the contribution of each bot to the attack greater than x/N where x is the amount of time this bot spent participating in the attack and N is the total amount of time spent by all bots in this attack? That is, is there a *synergy* in having multiple bots simultaneously participating in an attack? Of the eleven botnet attacks listed in sect. 2.4, only a few obviously require some bot synchronization: DDoS, phishing, and spamming. Of these, only DDoS requires a synchronization level on the order of seconds. For phishing attacks where the botnet both sends the luring spam email and hosts the web site, it is necessary to coordinate the bot hosting the website with the bots sending the spam, which can be achieved via fast-flux migration of the web site from bot to bot. Beyond that, there may be value in having bots simultaneously participating in a spam campaign but the necessary synchronization level here is likely to be on the order of minutes or tens of minutes.

8 Conclusion

In this paper, we establish a broad framework for evaluating the evadability of automated bot/botnet detection methods. In so doing, we identify myriad areas which warrant further study, including: quantifying the reconnaissance effort necessary to evade a method, improving the precision of implementation complexity estimations, enhancing understanding of botnet utility, and identifying how to best combine the components of an evasive tactic's cost so as to reflect market realities, the ultimate drivers in bot development. The exploration of evadability also yields interesting insights relating to future directions for detection research, including: identifying the fundamental characteristics of botnets and overall approaches to detection method design.

The most resilient detection methods are those which identify fundamental characteristics of bots/botnets rather than incidental traits of current bots. Hence, there is value in assessing the extent to which various characteristics used in detection are necessary, i.e., affect botnet utility. We identified the limitations of detection methods which require: synchronized bot execution (only a very small number of botnet attacks require tight coordination), that bot synchronization be achieved via the C&C network (synchronization can be obtained via external sources), or network-based botnet attack detection (many botnet attacks have negligible network-visible effects).

Considering evadability also yields alternative approaches to designing detection methods. First, we note that many of the leading detection methods can be evaded using the same tactic. Hence, there may be value in a novel detection method merely if evading it requires applying a new (non-trivial) evasive tactic. Additionally, while the traditional approach to devising detection methods has been to consider current bot/botnet structure and behavior, an alternative approach would focus on botnet utility and attempt to devise methods whose evasion would negatively affect this utility. For example, we know that a botnet's utility includes the diversity of attacks in which that botnet can be used. Consequently, we might design a detection method that used this diversity to identify bots/botnets. Then, evading that hypothetical method would entail reducing this diversity and hence sacrificing some botnet utility. Superior detection methods force the bot writer to choose between detection and reduced utility.

9 Acknowledgments

We are grateful to Dan Boneh, who gave us the paper idea and suggested venue, and to Phil Porras, who promptly provided additional BotHunter implementation details. Thanks also to Adam Barth, Matt Fredrikson, Guofei Gu, Collin Jackson, and Lorenzo Martignoni for reading earlier versions and providing helpful feedback. Finally, we thank the reviewers for their thoughtful questions and commentary.

References

[1] B. Page. A Report on the Internet Worm. Risks Digest, Volume 7, Issue 76, Nov. 1988.

[2] R. M. Slade. History of Computer Viruses. 1992.

[3] W. Cui, R. H. Katz, W. Tan. Design and Implementation of an Extrusion-based Break-In Detector for Personal Computers. In Annual Computer Security Applications Conf., Dec. 2005.

[4] W. T. Strayer, R. Walsh, C. Livadas, D. Lapsley. Detecting Botnets with Tight Command and Control. In IEEE Conference on Local Computer Networks, Nov. 2006.

[5] J. Goebel, T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In Workshop on Hot Topics in Understanding Botnets, April 2007.

[6] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, D. Dagon. Peer-to-Peer Botnets: Overview and Case Study. In Workshop on Hot Topics in Understanding Botnets, April 2007.

[7] A. Karasaridis, B. Rexroad, D. Hoeflin. Wide-scale Botnet Detection and Characterization. In Workshop on Hot Topics in Understanding Botnets, April 2007.

[8] E. Stinson, J. C. Mitchell. Characterizing Bots' Remote Control Behavior. In Detection of Intrusions & Malware, and Vulnerability Assessment, July 2007.

[9] H. Yin, D. Song, M. Egele, C. Kruegel, E. Kirda. Dynamic Spyware Analysis. In USENIX Security Symposium, Aug. 2007.

[10] G. Gu, P. Porras, V. Yegneswaran, M. Fong, W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In USENIX Security Symposium, Aug. 2007.

[11] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, S. Savage. On the Spam Campaign Trail. In USENIX Workshop on Large-Scale Exploits and Emergent Threats, April 2008.

[12] T. Holz, M. Steiner, F. Dahl, E. Biersacky, F. Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm-Worm. In USENIX Workshop on Large-Scale Exploits and Emergent Threats, April 2008.

[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In USENIX Security Symposium, July 2008.

A Current Automated Bot Detection

The aspects of the surveyed methods which are relevant to the consideration of evasion are presented in sect. 4. Below, we provide additional details pertaining to the surveyed detection methods.

A.1 Network-Based

A.1.1 Strayer

The goal of Strayer *et al.* in [4] is to identify tight C&C as exhibited by IRC networks. There are several stages in their analysis, where the output of each stage becomes the input for the next. First, eliminate flows that are unlikely to be botnet C&C (e.g., bulk data transfers, port scans, non-TCP traffic, short-lived connections, flows with average packet size greater than 300 bytes). Secondly, use machine learning algorithms to further prune flows keeping only those which are likely to be IRC (using flow characteristics, such as duration, role, bytes per packet (bpp), bytes per second (bps), packets per second (pps)). Thirdly, only keep flows from a period of time during which the botnet was active. Fourthly, correlate flows by having each represent a point in a five-dimensional space, where the dimensions relate to packet inter-arrival time and size. The distance between two points in this 5D space represents the likelihood that two flows are correlated. Fifthly, obtain a cluster of flows that are highly correlated (small distance between them). Then perform topological analysis on these to identify the RP and possibly manual analysis to identify the bot master's IP.

A.1.2 Rishi

Rishi [5] identifies hosts that are likely to be infected with a bot by passively monitoring network traffic and identifying certain IRC protocol packets to/from users which specify suspicious (bot-like) nicknames as defined by pre-constructed templates.

A.1.3 Karasaridis

The goal of Karasaridis *et al.* in [7] is to identify botnet controllers (e.g., IRC servers) given transport layer data. First, they identify "suspected bots" as those hosts which are spamming, sending viruses in email, port scanning, or participating in DDoS attacks. All flow records to/from suspected bots are fetched. Of these, they identify flow records that may be connections to controllers (referred to as *candidate control flows*) using three criteria: the server port is a standard IRC port, the <server_ip, server_port> has incoming connections from multiple suspected bots (i.e., is a "hub server"), or flow records whose characteristics are "within the bounds of a flow model for IRC traffic". The candidate control flows for each distinct <client_ip, server_ip, server_port> are summarized into a *candidate control conversation* (CCC).

CCCs are pruned via multi-stage correlation wherein the output of one stage determines the input for the next. First, calculate the number of unique suspected bots for each the <server_ip, server_port> tuple. Of these, consider the most popular servers. For each such <server_ip, server_port>, combine all of that server's CCCs and calculate the average flows per (client) address, packets per flow, and bytes per packet. Then obtain the distance of this traffic from the model for IRC by giving equal weight to each flow characteristic. Servers whose distance is below a threshold will be considered in the final stage, which entails applying heuristics to order the remaining candidate controllers, such as number of idle clients and whether the server uses both TCP and UDP on the suspected port and has many peers.

A.1.4 BotHunter

BotHunter [10] presents a method for botnet detection which entails correlating alarms from different network intrusion detection system (NIDS) elements which reside at the egress boundary. They introduce a model of a bot infection sequence which entails certain combinations of the following events: inbound port scan (E1), inbound exploit (E2), internal-to-external binary download (E3), internal-to-external C&C communications (E4), and outbound port scan (E5). A port scan detection engine identifies E1 and E5, Snort signatures and a payload-anomaly detection engine identify E2, and Snort signatures detect E3 and E4. Different combinations of alarms — which occur within a particular time and which all reference the same internal host (IP) — can satisfy the threshold for declaring a bot infection. In particular, E2 followed by E3, E4, or E5 results in an alarm as does any two of {E3, E4, E5}.

A.1.5 BotMiner

In BotMiner [13] the authors present a botnet detection method which clusters: communications traffic (C-Plane), which identifies which hosts are talking to which other hosts, and activity traffic (A-Plane), which identifies which host is doing what. A C-Plane flow (C-flow) contains all of the flows over a given epoch between a particular internal IP and destination IP and port which use the same transport layer protocol. Some flows are excluded from consideration: internal-to-internal, external-to-internal, and those to “legitimate servers”, such as Google. Certain C-flow characteristics are extracted: flows per hour (fph), packets per flow (ppf), bytes per packet (bpp), and bytes per second (bps). The A-Plane identifies hosts which are scanning (i.e. demonstrate an abnormally high: scan rate or weighted failed connection rate), spamming (high number of: DNS queries for MX records or connections to external mail servers), and downloading any Portable Executable binary. Clustering algorithms are applied to group hosts with similar: communication patterns (C-Planes) and activities patterns (A-Planes). They then perform cross-plane correlation to identify hosts with similar communications and activities patterns.

A.2 Host-Based

The relevant work in this area includes systems that target malware generally as well as those that focus specifically on bots.

A.2.1 Binder

In [3], Cui *et al.* present a method for identifying *extrusions*, which are user-unintended malicious outbound connections. User-driven input is used as a proxy for user intent, and network connections made within some time window following receipt of user-driven input are considered user-intended. Hence, their system identifies as malicious: network connections which were not preceded in time by a user input event. For typical standalone bots, detection could occur upon the first connection of the bot to its C&C network since this event is not preceded by receipt of a user input event to the bot program.

A.2.2 BotSwat

Stinson *et al.* characterize the remote control behavior of bots via identifying when selected system call arguments contain data received over the network, such as occurs when a bot executes a command received from its bot master [8]. This approach performs explicit information flow tracking on network data. To distinguish remotely-initiated from locally-initiated system call invocations, the method identifies data which is dependent upon local user input and sanitizes such data.