Active Botnet Probing to Identify Obscure Command and Control Channels

Guofei Gu[1], Vinod Yegneswaran[2], Phillip Porras[2], Jennifer Stoll[3], and Wenke Lee[3]
[1]Texas A&M University [2]SRI International [3]Georgia Institute of Technology
guofei@cse.tamu.edu, {vinod, porras}@csl.sri.com, {jstoll@,wenke@cc.}gatech.edu

***Abstract*—We consider the problem of identifying obscure *chat*-like botnet command and control (C&C) communications, which are indistinguishable from human-human communication using traditional signature-based techniques. Existing passive-behavior-based anomaly detection techniques are limited because they either require monitoring multiple bot-infected machines that belong to the same botnet or require extended monitoring times. In this paper, we explore the potential use of *active botnet probing* techniques in a network middlebox as a means to augment and *complement* existing passive botnet C&C detection strategies, especially for small botnets with obfuscated C&C content and infrequent C&C interactions. We present an algorithmic framework that uses hypothesis testing to separate botnet C&C dialogs from human-human conversations with desired accuracy and implement a prototype system called BotProbe. Experimental results on multiple real-world IRC bots demonstrate that our proposed active methods can successfully identify obscure and obfuscated botnet communications. A real-world user study on about one hundred participants also shows that the technique has a low false positive rate on human-human conversations. We discuss the limitations of BotProbe and hope this preliminary feasibility study on the use of active techniques in botnet research can inspire new thoughts and directions within the malware research community.**

I. Introduction

Botnets refer to large collections of compromised machines infected with a specific malware instance (i.e., a bot), which enable them to be commandeered by an individual often referred to as a "botmaster". Botnets may range in size from tens to hundreds of thousands of systems, often spanning a multitude of home, educational and corporate networks, and are typically exploited as platforms for conducting a wide range of criminal activities, including spam campaigns, identity theft, and denial-of-service (DoS) attacks. The magnitude of these collections and the potency of attacks afforded by their combined bandwidth and processing power have led to a recognition of botnets as one of the premier threats to Internet security.

A unique property of a botnet that separates it from other malware families is the command and control (C&C) channel, which the botmaster uses to command the bot army to perform different tasks. Although botnet developers have the option of devising novel protocols for C&C, most contemporary botnet C&C communications are overlaid onto existing protocols such as IRC (Internet Relay Chat) and HTTP. This prevailing tendency to overlay botnet C&Cs on *existing* protocols may have several plausible explanations: (a) existing protocols provide greater flexibility in using available server software and installations; (b) existing protocols invoke less suspicion than neoteric protocols; (c) existing protocols work so well so that there is no sufficient incentive for botnets to innovate.

Although HTTP-based botnets (e.g., Bobax [26] and its new variant Kraken) and P2P botnets (e.g., Nugache [19], Storm [11], and Conficker [22]) have recently garnered considerable attention, we should note that IRC-based botnets remain a lingering and significant threat [2], [6]. The persistence of IRC-based botnet communication channels could be attributed to the simplicity and flexibility afforded by IRC's text-based protocol. In addition, contemporary IRC botnets have evolved from simple dialects to a new era where C&C content in IRC messages are obfuscated (e.g., using a custom dialect, a foreign language, or a naive obfuscation technique such as simple XOR, substitution, or hashing). By using obfuscated IRC messages (e.g., "hello" instead of "scan"), these botnets can evade signature-based detection [10], [21], [25] and honeypot-based tracking approaches [23]. Indeed, we have observed a substantial collection of current IRC botnets utilizing obscure C&C communications [2], [6].

Behavior-based botnet detection approaches can detect botnets through behavioral anomalies. Many such systems have been proposed recently [12]–[14], [17], [27], [34]. However, they all have limitations. Some techniques (e.g., [12], [14], [17], [27], [34]) use *group analysis* for detection. However, these techniques require the presence of multiple bots in the monitored network and cannot help much when there is only one infection in the network. Furthermore, techniques such as (e.g., [12], [14]) may require a longer time in collecting sufficient evidence for detection. For example, BotMiner has an offline correlation engine that performs daily group analysis on C-plane data. BotSniffer is more agile than BotMiner, but still requires observing several rounds of messages to gather sufficient confidence in its spatio-temporal correlation. BotHunter uses a (mainly) signature-based approach to track bot infection dialogs and requires observing activity from multiple stages of the bot lifecycle to declare an infection. In contrast, real-world IRC-based botnet C&C communications can be quiet, i.e., some have infrequent C&C interactions because the botmaster is not always online to command the bot army. If the frequency of C&C interactions is low enough, botnets could potentially

evade detection by these systems. Indeed, steathy botnets with small sizes, obfuscated C&C dialogs, and infrequent C&C interactions pose an ongoing challenge to the malware research community.

To address this challenge, we explore new botnet detection techniques that *actively* collect evidence. We intend to answer the following questions: Assume there is only one round of (obscure) chat-like botnet C&C interaction from one bot,[1] can we still detect the bot with a high probability? What if we observe *zero* rounds of interaction? We will show that our solution can achieve the detection goal for many real-world botnets that use chat-like C&C protocols such as IRC, and complement existing techniques in many cases.

Key Observations: We posit that instead of passively inspecting two-way network flows, one can engage in the active manipulation of selected suspicious sessions to better identify botnet dialogs. Our detection strategy, which we call *active botnet probing*, is based on two observations. First, a typical botnet C&C interaction has a clear *command-response* pattern, and therefore a stateless bot will tend to behave deterministically[2] to dialog replays, whereas interaction with a human-controlled end point will be nondeterministic. Second, bots are *preprogrammed* to respond to the set of commands they receive and, unlike humans, bots have limited tolerance for typographical errors in conversations (aka the Turing test [28]).

New Approach and System: Based on the above observations, we develop a set of active probing techniques to detect stateless chat-like botnet communications, regardless of whether or not the botnet communications are protected using simple obfuscation. At first glance, these active techniques may be aggressive and controversial because of the interference they may introduce to normal benign communications/chatting. While a legitimate concern, we propose to ameliorate this interference in multiple ways. First, we provide a set of candidate filters that use heuristics to filter out a large class of well-behaved connections. Second, we provide a hypothesis testing framework that enables network administrators to tune the level of expected interference with detection rates. In addition, a whitelist approach to avoid disturbing known critical/legitimate programs/sessions can also be used to reduce potential interference. Finally, we argue that limited interference might be acceptable in pure IRC-like chatting channels on which no critical applications are built, and certain deployments such as military scenarios, particularly if users are educated about the presence of such probing monitors. We develop the BotProbe prototype system to demonstrate this active technique. By actively probing botnets, we can accumulate enough evidence (without passively waiting) of *cause-effect* correlation that exploits the command-response patterns of botnet C&Cs. We need to observe only one or even zero rounds of actual C&C interaction before probing. Thus, we can greatly shorten the detection time compared to a passive approach.

[1]One round of C&C interaction is defined as a typical command-then-response interaction. We further clarify this command-response pattern of botnet C&C and various types of responses in Section II.

[2]Examination of popular bot source code and binaries reveals that most bot communications are stateless.

Contributions of this paper:

- We propose active botnet probing based on cause-effect correlation as a novel approach to complement existing passive botnet C&C detection. We envision the detection of future botnets will require a combination of different detection features and techniques because botnets are very complex and dynamic. We believe our proposed new active probing technique, although has limitations, can provide a unique perspective and inspire new research directions.
- We present a hypothesis testing framework for detecting deterministic communication patterns. We develop BotProbe, a prototype implementation of the framework that validates our approach with contemporary IRC-based bots such as Sdbot, Phatbot, Rbot, RxBot, Agobot, Wargbot, and IRCBot. In addition, we show with a real-world example that BotProbe can also assist with automating a chosen-ciphertext attack to break the encryption of some botnet C&C.
- We conduct a real user study on around 100 users to evaluate false positive rates.

II. Problem Statement and Assumptions

Our goal is to evaluate the feasibility of identifying chat-like botnet C&C channels using *active* network traffic inspection strategies, while observing only a limited number of C&C interactions (one or even zero) on a single bot, thereby to complement existing passive approaches. By active, we mean that we assess traffic for suspicious traffic sessions, which may lead us to dynamically inject packets that will probe the internal client to determine whether that side of the communicating/chatting session is being managed by a human or a bot. To achieve the goal, first we need to examine the *invariant* that can be used to differentiate a bot from human chatting. We observe that bots are *preprogrammed to respond* to certain *predefined* commands, and these responses are consistent across command repetition. Different from normal human chatting, the above command-response pattern has a strong cause-effect correlation, i.e., the command causes the response in a deterministic way. This is the key intuition we use in designing our algorithm. In addition, we observe that bots are different from humans in tolerating typographical errors, i.e., if the command is altered by even one character, bots are not likely to process the command

properly. This auxiliary intuition helps us design one of our detection algorithms. Before introducing our algorithms and system in detail, we present the adversary model, i.e., the detailed communication patterns that we seek to identify when adversaries communicate with compromised machines inside our network perimeter.

We now discuss our adversary and detection assumptions below. We will discuss limitations and policy/risk concern further in Section V.

Adversary Assumption: Botnet C&C communications are well-structured duplex flows, similar to a command-response protocol, i.e., a bot should respond when it receives a predefined command in a reasonable time. The network-level response of a bot to an (obfuscated) command might be either *message response* or *activity response*, or both [14]. A typical example of message response is an IRC `PRIVMSG` message. For example, when the botmaster issues a "`.sysinfo`" command,[3] each bot replies with a `PRIVMSG` message telling its host system information, such as CPU, memory, and software version. There are three most common activity responses: scan response (bots perform network scan or DoS attack), third-party access (e.g., bots connect to a certain address to download/update their binary), and spam response (bots send spams). For instance, when the botmaster issues a scanning command (e.g., "`.scan.startall`"), bots usually perform network scanning and reply with the scanning progress and/or any new victims they have infected. This involves both an activity response (scan) and a message response. One may define other possible responses, but from our observation of live bot infections, these aforementioned types of responses are highly representative and regularly encountered.

Fortunately, the assumption of command-response pattern holds in almost all existing botnets, because the botmaster needs the bots to perform some (malicious) activity, and usually requires feedback to track the bot-infected machine information and execution progress/result from its bot army. Thus, we can observe message/activity responses corresponding to most botnet commands. According to a honeynet technical report [36], about 53% of botnet commands observed in thousands of real-world IRC-based botnets are scan related (for propagation or reconnaissance) and about 14.4% are related to binary download (for malware update). Also, many HTTP-based botnets are primarily used for sending spam [26]. Thus, for most infections, we can expect to observe (malicious) activity responses with a high probability [5].

Detection Assumption: We now discuss the design assumptions used in defining our architecture for actively probing and detecting botnet C&C channels:

- *Input Perspective.* Our assumed solution will reside at the network egress point (as a middlebox), where it can observe all flows that cross the network perimeter. Furthermore, the system is in-line with the communication, and has the authority to inject or modify *inbound* packets, as necessary.
- *Chat Protocol Awareness.* Our solution incorporates knowledge of the standard (chat) protocols that botnets use to overlay their C&C communications. For example, in the case of IRC-based bots, we can comprehend IRC keywords and `PRIVMSG` message exchanges.

[3]We assume the botmaster could obfuscate the C&C channel using simple encryption or substitution, e.g., say "`hello`" instead of "`.sysinfo`."

III. Active Botnet Probing: Architecture and Algorithms

A. Architecture Design

Our botnet C&C detection architecture has two integral components, as shown in Figure 1.

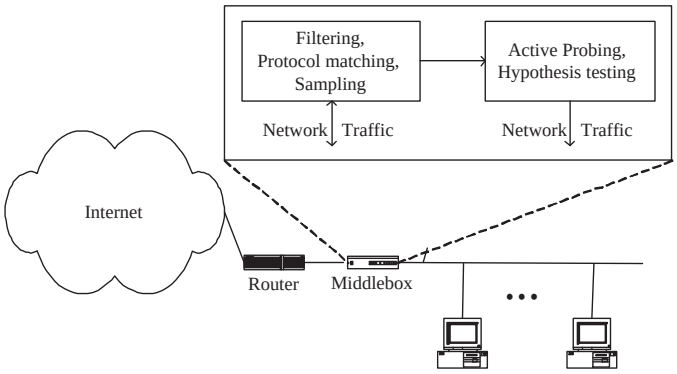


Figure 1. Two-layer architecture using active techniques for identifying botnet C&Cs.

The first component performs benign traffic filtering, protocol matching (selects protocols often exploited for C&C transmissions, e.g., IRC), and flow sampling. Thus, it leaves only a small portion of highly suspicious candidates worthy of deeper examination.[4] Benign (chat-like) traffic filtering modules are implemented using a general traffic feature vector (e.g., duration of the flow, average bytes per packet, average bytes per second) similar to [17], [20], [27]. Finally, in the case of IRC-based C&C, we use the following protocol matching policies to perform detection in a port-independent fashion:

1) A traffic filter removes non-TCP flows.
2) Port-independent IRC protocols are keyword matched, e.g., "`NICK`," "`USER`," "`PRIVMSG`." This analysis occurs on the first few packets of established TCP flows (which indicate the beginning of an IRC session [1]).
3) A volume filter mainly focuses on infrequent IRC chat channels (because overly chatty IRC channels are unlikely to be used for botnet C&C).

[4]As always, this is a trade-off between performance and accuracy.

4) A message filter finds a candidate list of command-like packets (IRC `PRIVMSG` and IRC `TOPIC`) that can cause client responses.

Once we have completed the above down-selection to our candidate flows, we then focus our analyses on the `TOPIC` and `PRIVMSG` message packets, where the overlay C&C commands/responses typically reside. In addition, one can incorporate any other behavior-based logic into these filters.

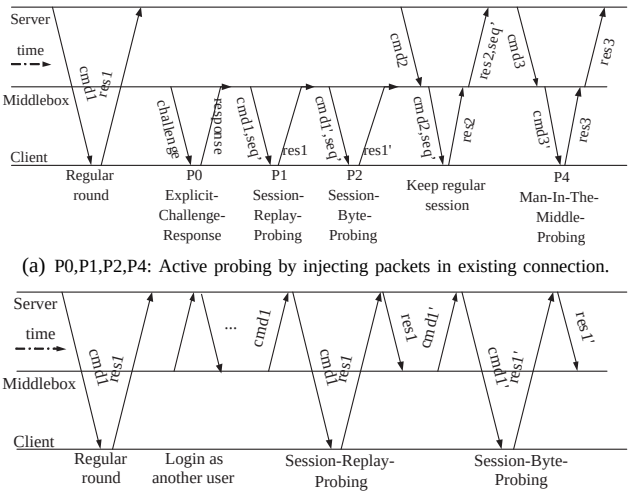
The second component implements what we refer to as our *BotProbe analysis* scheme. To illustrate the scheme, let us suppose that we have a candidate suspicious IRC session and we need to further identify whether there is another layer of overlay C&C-like protocol. We observe a command-then-response-like packet pair (P_c, P_r) where P_c is a short packet from the server, and P_r is a response from the client immediately after the receiving of P_c.[5] We hypothesize that this command-response pattern is from a bot instead of a human. However, observing only this likely command-response pair is not enough to make the claim, because it could be caused by chance.[6] We want to make sure whether there is truly a cause-effect correlation between the command and the response, which is a distinguishing feature between botnet C&C and human chatting. To achieve the detection goal with high accuracy, we perform several rounds of active probing and use a sequential hypothesis testing technique to obtain enough confidence. The next section will detail the design space of active probing techniques.

B. Design Choices of Active Probing Techniques

We investigate the design space of active probing strategies and illustrate in Figure 2 several probing techniques that were considered in our BotProbe system.

P0 (Explicit-Challenge-Response). An example explicit-validation mechanism is one in which educated users knowingly participate in the BotProbe scheme. The BotProbe system may prompt users to perform a reverse Turing test, when a new IRC session among two IP addresses is first encountered by BotProbe. The in-line monitor could request that the internal human IRC participant visit a website to read and translate a CAPTCHA [29]. Alternatively, BotProbe can inject a simple puzzle for the internal participant to solve. Using this technique, one may detect botnet channels before observing actual C&Cs, i.e., observing *zero* rounds of interaction. Although simple and effective, such a technique requires user awareness, compliance, and tolerance to be successful. We further discuss our experience of this technique in an actual user study in Section IV-C.

P1 (Session-Replay-Probing). The BotProbe monitor spoofs the address of the server and inserts additional TCP packets that replay the same application command P_c to the client several times. If the remote end point is a bot, it will likely provide responses that are deterministic (with respect to both content and timing).



(a) P0,P1,P2,P4: Active probing by injecting packets in existing connection.

(b) P3: Active probing by establishing new connection.

Figure 2. Example active probing techniques. Here cmd' means a modified command packet, seq' means modification is needed on the sequence/acknowledge number to keep the TCP session.

P2 (Session-Byte-Probing). The BotProbe monitor randomly permutes certain bytes of the application command.[7] If the client is a bot, then we expect it to be highly sensitive to modifications of commands and hence to respond differently or drop the modified packet. However, a human user in an IRC chat channel would have a higher tolerance for typographical mistakes in an IRC message. We may repeat our test as many times as necessary by interleaving strategies P1 and P2, until we have sufficient evidence to validate our hypothesis. We describe the algorithm (Interleaved-Binary-Response-Hypothesis) in more detail in Section III-C.

Note that strategies P1 and P2 may break existing connections (by injecting new packets) if subsequent C&C communications occur in the same TCP connection. To recover from this, our in-line botnet probing system should adjust the TCP sequence/acknowledge numbers and checksums to account for the new packets that were introduced because of the probes. Also, the above two probing strategies introduce some amount of interference into existing sessions at the application level. Fortunately, we find that, for our targeted chat-like protocols, we have an alternate probing technique (P3), which does not disturb existing sessions.

P3 (Client-Replay-Probing). Chat protocols like IRC and IM allow users to directly message each other. In such instances, we instantiate a new user that logs into the channel

[5] Sometimes there is no such a message response packet P_r, but rather an activity response. We still use P_r to stand for this activity response.

[6] The false positive rate can be higher, particularly if P_r is only a message response packet because it could be just a normal prompt chat message from a human.

[7] Since many common botnet command names (e.g., `.dos`, `.scan`) are embedded in the initial bytes of IRC `PRIVMSG` or `TOPIC` message packets, we recommend biasing the byte modification algorithm to choose the early bytes with higher probability.

and sends the observed command(s) P_c to the selected client (pretending to be the botmaster). By doing this, we do not break existing connections, but achieve an effect similar to that above. Figure 2(b) illustrates this scenario.

P4 (Man-In-The-Middle-Probing). The above techniques do not directly intercept a new command packet. However, in some cases (as discussed in Section V) such as highly stateful C&Cs where simple replaying may not work, we intercept the *new* command, and launch a man-in-the-middle-like chat message injection.

P5 (Multiclient-Probing). The above techniques discuss probing sessions from a single client. However, when multiple likely infected clients in the monitored network are communicating with the same C&C server, we distribute the probes among multiple clients and reduce the number of probing rounds needed to test our hypothesis.

C. Algorithm Design for Botnet Detection Using Active Probing

Based on the active probe techniques, we now describe several simple detection algorithms for isolating deterministic botnet communication patterns from human chat dialogs with controlled accuracy (i.e., to achieve a desired false positive/negative rate). We use a sequential probability ratio testing (SPRT [30]) technique, which has been successfully applied in several other scenarios [14], [16]. To illustrate the algorithm, we start with a basic description of how to apply a hypothesis testing framework using botnet probing strategies.

Let us assume that we are given a (suspicious) IRC session and we want to differentiate whether it is more likely a botnet C&C channel or a human chat session. We perform one or more rounds of P0 probing (i.e., inject a challenge to the client, ask the local participant (within our network boundary) to solve a puzzle). We denote H_1 as the hypothesis "botnet C&C," H_0 as the hypothesis "normal chat." Let a binary random variable D denote whether or not we observe a *wrong* reply for a challenge from the client (that is, $D = 1$ means an incorrect reply). We also denote $\theta_1 = Pr(D = 1|H_1), \theta_0 = Pr(D = 1|H_0)$. If the client is a bot, we presume $\theta_1 \approx 1$, assuming that bots are unable to reliably solve arbitrary puzzles on demand. For a human, such a puzzle is easy to answer, i.e., $\theta_0 \approx 0$. If we want to have very high accuracy for the hypothesis (let us denote α, β as the false positive rate and false negative rate we want to achieve), we can perform several rounds of probing. Then, after observing n rounds, we get a likelihood ratio $\Lambda_n = \frac{Pr(D_1,\ldots,D_n|H_1)}{Pr(D_1,\ldots,D_n|H_0)}$. D_i represents our independent identical distribution (i.i.d.) observation result from our client probe test. We define $\Lambda_n = \ln \frac{\prod_i Pr(D_i|H_1)}{\prod_i Pr(D_i|H_0)} = \sum_i \ln \frac{Pr(D_i|H_1)}{Pr(D_i|H_0)}$. To calculate this likelihood Λ_n, we are essentially performing a threshold random walk (TRW). The walk starts from origin (0), goes up with step length $ln\frac{\theta_1}{\theta_0}$ when $D_i = 1$, and goes down with step length $\ln \frac{1-\theta_1}{1-\theta_0}$ when $D_i = 0$. If Λ_n is greater than a threshold $t_1 = \ln \frac{1-\beta}{\alpha}$ we declare H_1 to be true, *i.e.*, it is a botnet C&C. If Λ_n is less than another threshold $t_2 = \ln \frac{\beta}{1-\alpha}$, this indicates a normal IRC dialog. If Λ_n is in between t_1 and t_2 we proceed with additional rounds of testing. A nice property of this SPRT/TRW algorithm is that it can achieve bounded false positive and false negative rates as desired, and it usually needs only a few rounds to reach a decision [30]. We call our first extension of the algorithm **Turing-Test-Hypothesis** because it uses explicit challenge response. This algorithm even does not require observing any actual botnet C&C interaction.

Similarly, we can adapt the algorithm to use the P1 technique in every round. Let P_c be a suspicious command packet from the server to the client. We replay P_c in each round and we denote D to indicate whether or not a response from the client is observed. We call this **Single-Binary-Response-Hypothesis** algorithm because this test considers the probe response as a binary outcome. Depending on the response we observe (IRC `PRIVMSG` message, scanning, spamming, or third-party access), we iterate the TRW process at different scales, because θ_0, θ_1 (the corresponding probability associated with a bot or human) is different for different responses. For example, a human-driven IRC session is very unlikely to perform scanning when receiving a chat message. Thus, we improve our confidence when we observe a scanning response corresponding to the replayed (command) message. If we receive multiple different types of responses corresponding to the same command, we choose the one that provides highest confidence (walks a largest step). The exact number of rounds we need in this case is discussed in the next section. In general, Single-Binary-Response-Hypothesis is very effective if the replayed command packet is scan, spam, or binary download related. As shown in Section III-D, we may need only *one* extra replay in addition to the original command, i.e., two rounds to detect a botnet.

In addition to performing binary response testing, we can further evaluate whether the response is *similar* to the previous response observed, because bot responses may not be perfectly identical across multiple command replays. We hypothesize that for bot C&C communication, responses to the same command are similar in structure and content. We can design a new hypothesis algorithm that inspects whether a response is correlated to previous responses using a simple edit distance metric or a DICE metric as in [14]. We call this extension the **Correlation-Response-Hypothesis** algorithm.

Finally, we introduce the **Interleaved-Binary-Response-Hypothesis** algorithm. In each round, we perform interleaved P1 and P2 probing, i.e., replaying the original P_c packet, and then replaying a modified P_c packet. $D = 1$ denotes the observation of a response from the replayed

P_c, and $D = 0$ denotes no response from modified P_c. The assertion is that bots reliably respond to P_c, but do not recognize the modified command. This occurrence is then observed as $D = 1$. To a human user, these two are similar (a modified P_c is just like a typographical error (typo), and while chatting, a typo is normal and generally not a problem). It is hard to predict how normal users may respond when they receive these two replayed IRC `PRIVMSG` messages, but the probability of obtaining repeatable responses from replayed P_c and no responses from modified P_c should diminish with rounds. A naive assumption is that the human responses to tampered packets are uniformly random, $\theta_0 = Pr(D = 1|H_0) = 1/4$. In reality, normal users would quickly lose patience upon receiving multiple similar IRC messages, and hence this probability θ_0 should be lower than the uniformly random case. Our later user study (in Section IV-C) also confirms that θ_0 is very low.

One benefit of the Interleaved-Binary-Response-Hypothesis algorithm is that we can have a *general* way to detect a *third-party access* response and do not rely on content signatures, e.g., PE header (Microsoft Windows executable) signature as used in BotHunter [13] to detect egg downloading. This has the advantage when we do not have signatures for detecting these third-party accesses, e.g., the access is not for a Windows executable, or the access connection does not yield a successful download of a Windows executable. We begin by building a suspicious access set containing addresses (most likely, HTTP addresses) that appear after the P_c but not after the modified P_c. Then for each subsequent round, we assign $D = 1$ if we see an address from the suspicious set still appear upon replay of P_c, but not upon the sending of the modified P_c.

We have introduced several different detection algorithms. Now we discuss the typical selection of proper algorithms in practice when facing a different type of response or a different combination of responses. We think that for a normal chatting host, the probability of performing a certain (malicious) activity response (e.g., scan, spam) is lower than that of performing a message response. The general principle we need to follow here is to choose the algorithm that favors the response with the lowest probability and thereby makes the fewest probes and the largest walk in the threshold random walk. In the following analysis we assume $Prob(scan) \approx Prob(spam) < Prob(3rd-party-access) < Prob(message)$ in the case of a normal chatting client.

If we observe a scan/spam response associated with a command (there might be other responses such as an IRC `PRIVMSG` message), we choose the *Single-Binary-Response-Hypothesis* algorithm on the scan/spam response, and ignore other responses. Usually, we only need another active probing (using P1) to declare a botnet as shown in Sections III-D and IV-B. It is possible that these scan/spam responses are long-lasting, i.e., we still observe the response to the original command after we perform P1 (a replayed command). However, we do not consider this as a problem, because we still detect the bot. Here our detection performance is at least no worse than the approaches that issue alerts when observing the combination of IRC events and scan events such as [4] and BotHunter [13].

If we observe a third-party access (by matching a PE signature) associated with a command (there might be some message response, but no scan/spam responses), we choose the *Single-Binary-Response-Hypothesis* algorithm on the third-party access response.

For the remaining combination of responses (e.g., a message response and a third-party access response without PE signature capturing) or only a message response, we can choose *Interleaved-Binary-Response-Hypothesis* algorithm. If there are both a message response and a third-party access observed, to make a walk in the algorithm, we always pick the type of response that makes a larger step (third-party access in this case) in the threshold random walk.

D. Evaluating User Disturbance and Detection Accuracy Tradeoff

We now describe how the above algorithms can be adapted to trade off user disturbance with system performance. For benign IRC chat sessions, replaying or modifying some byte is essentially equivalent to receiving a duplicate message or receiving a message with a typo: humans have natural resilience to at least limited occurrences of these events. The Client-Replay-Probing technique, which establishes a new session, is even less harmful. Nevertheless, we acknowledge that active modifications to user IRC sessions may impose some degree of cost to human users. We present a more detailed discussion on the legal concerns of using active techniques in Section V.

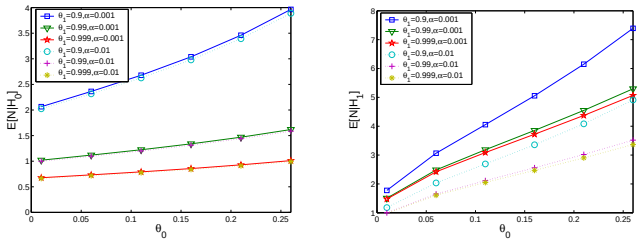
As discussed earlier, to have a high confidence of hypothesis testing, we may need N rounds of probing. If we are concerned about the disturbance/interference to normal users, we could use the number of rounds (packets modified/replayed) by active probing as a means to quantify the degree of disturbance. Clearly, less disturbance means fewer rounds, smaller N, which on the other hand, may affect the performance of detection. Fortunately, because of the use of SPRT, the average number of N to make a decision is quite small. To produce a botnet C&C declaration, the expected number of rounds we need is [30]

$$E[N|H_1] = \frac{\beta \ln \frac{\beta}{1-\alpha} + (1-\beta) \ln \frac{1-\beta}{\alpha}}{\theta_1 \ln \frac{\theta_1}{\theta_0} + (1-\theta_1) \ln \frac{1-\theta_1}{1-\theta_0}}$$

Similarly, to produce a human user IRC channel declara-

tion, the expected number of rounds is

$$E[N|H_0] = \frac{(1-\alpha)\ln\frac{\beta}{1-\alpha} + \alpha\ln\frac{1-\beta}{\alpha}}{\theta_0\ln\frac{\theta_1}{\theta_0} + (1-\theta_0)\ln\frac{1-\theta_1}{1-\theta_0}}$$



(a) Average number of rounds to detect normal IRC user.

(b) Average number of rounds to detect botnet C&C.

Figure 3. Disturbance to normal user and the effect on detection.

Figure 3 shows the average number of rounds we need to declare a normal user (a) or bot (b). For example, if we set parameters $\theta_1 = 0.99$, $\theta_0 = 0.15$, and our desired false positive/false negative rates are $\alpha = 0.001, \beta = 0.01$, then the average number of rounds to declare a botnet is about $N_1 = 3.7$. Likewise, the average number of rounds to declare a human user is less than two for IRC (approximately $N_0 = 1.3$). If we observe some scan response, we use a lower probability of θ_0, e.g., $\theta_0^{scan} = 0.01$; then it takes less than two rounds (e.g., one extra replay) to detect bots on average.

Our system is bolstered by an IRC channel whitelist to minimize user disturbance (i.e., once an IRC server/channel is validated, we will not disturb other users for a certain time window, and the time window could be randomized). Finally, the BotProbe strategy should be viewed as one input among a broader set of threat indicators that can be applied for detecting internal botnet infections. For example, the results produced by the BotProbe hypothesis testing framework could be incorporated into systems such as BotHunter [13], which considers the full set of potential botnet-related infection indicators, such as exploit usage, egg download events, and inbound and outbound attack behavior.

IV. Experiments with BotProbe

A. BotProbe: a Prototype Active Botnet Probing System

We have implemented a prototype middlebox system called BotProbe for the purpose of evaluating our active probing techniques. BotProbe is implemented as a collection of Click routing elements [18]. Click provides a C++ software framework for packet processing, with impressive scaling performance and a flexible configuration language, which makes it ideal for building software routers and middleboxes.

The BotProbe architecture is shown in Figure 4. The key elements that we developed are WatchList, IRCMatcher, and

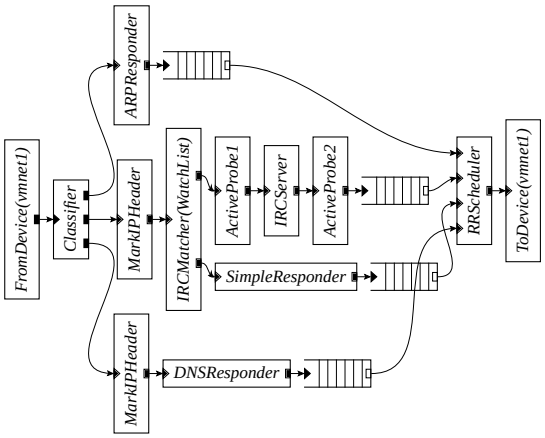


Figure 4. Click configuration for BotProbe. The figure shows a configuration for black-box testing on existing bot binaries. If BotProbe is deployed as a middlebox into a real network, we can remove the IRC Server, SimpleResponder, and DNSResponder elements.

ActiveProbe. WatchList is a Click "information" element that keeps track of live TCP flows and IRC records. The IRCMatcher uses a WatchList to maintain flow records and examines incoming packets to identify IRC flows. The ActiveProbe element monitors all IRC flows, performs active probing if the IRC channel is deemed suspicious, and modifies TCP sequence/acknowledge numbers and checksums when necessary.

To simplify black-box testing on existing bot binaries, we also implemented the following elements: (i) an IRCServer element, which plays the role of a simple IRC server, (ii) a SimpleResponder that handles all non-IRC connections by acknowledging every packet it receives, and (iii) a DNSResponder that answers DNS queries with a local address. If BotProbe is deployed in-line as a middlebox into a real network, we can simply remove these three elements.

B. In Situ Experimental Evaluation

We evaluate the detection performance in a virtual network environment with several malicious IRC bots including Sdbot, Phatbot, Rbot, RxBot, Agobot, Wargbot, and IRCBot that we obtained from our bot source code repository and honeynet capture in the wild. The purpose is to test the false negative rate, i.e., how many bot C&Cs are missed by BotProbe? We answer this question using *in situ* VMware testing of real-world bot binaries described below. For false positive evaluation, we will report our user study in Section IV-C.

1) Detection Performance and Analysis: We begin our analysis by conducting a series of *in situ* experiments to evaluate the false negative rate. We proceed by executing the bot in a Windows XP (VMware guest) instance and monitoring with BotProbe running on the Linux host machine. Initially, BotProbe essentially acts as a faithful NAT middlebox interposing all communications between the

infected host and the Internet. If the IRCMatcher element identifies an IRC session, the flow will be forwarded to the IRCServer element that handles and responds to all IRC requests. The ActiveProbe element resides between the bot client and the IRCServer element, monitoring chatter and introducing active probes at appropriate times (e.g., when the channel is idle on a suspicious session). While the IRCServer element has the actual botnet commands, we do not assume that the ActiveProbe element knows the commands, as BotProbe runs in the realistic scenario.

Note, in real-world IRC based botnets, we observe that most of the commands are in IRC `TOPIC` messages, because the botmasters are not online all the time. To instruct bots even when they are not online, botmasters usually put the commands in the `TOPIC` of the channel. Thus, whenever a bot joins the channel, it will understand the commands in `TOPIC` and execute (without authentication). In such cases where there is no `PRIVMSG` message from the server but client responses are still produced, we can presume that the `TOPIC` is the command and play the probing game by manipulating observed `TOPIC` messages (332). We use this trick in our experiments, in order to faithfully replicate real-world scenario. In addition, as discussed in Section III-C, BotProbe performs a Single-Binary-Response-Hypothesis or Interleaved-Binary-Response-Hypothesis algorithm in our experiments depending on what kind of (combination of) response(s) it observes.

We evaluate BotProbe on several real-world IRC bots that can be grouped into three classes.

1. Open-source bots with obfuscated communication.: Our first case study is an "open source" (as described in the bot documentation) IRC bot called Spybot, which was released in 2003. Being open source, many variants of this bot are on the Internet, making it one of the more popular botnet families [3], [7]. Spybot is also equipped with a command encryption option that obfuscates C&C communication. The encryption method implemented is a simple byte shift scheme. We recompiled the Spybot source with the encrypt option enabled and tested the binary using BotProbe.

In evaluation, we configured the IRCServer to issue a set of commonly used commands listed in Table I (one command in each test). We set the parameters of the hypothesis testing algorithm to be $\theta_1 = 0.99, \theta_0 = 0.15$ giving expected false positive (FP) and false negative (FN) rates of 0.001 and 0.01, respectively.[8] We set $\theta_0^{scan} = 0.01$, because the probability that a normal chatting client has scan response is low (much lower than an IRC message response). Similarly, for a third-party access response, we set $\theta_0^{3rd-party-access} = 0.02$. We used this parameter setting across all experiments. In the test on Spybot, BotProbe took two probes when the command was "scan" (Single-Binary-Response-Hypothesis algorithm was automatically performed), two probes when the command was "download" (Interleaved-Binary-Response-Hypothesis algorithm was automatically performed because we do not use any PE signature to identify access response), and four probes when using commands such as "info" and "passwords" (Interleaved-Binary-Response-Hypothesis algorithm was automatically performed).

[8]This θ_0 is for the case of Interleaved-Binary-Response-Hypothesis on message response only.

2. Bot binaries with cleartext communication.: We tested a few other bots, e.g., Phatbot, Rbot, Rxbot, Sdbot [3], in our controlled network. In these experiments, C&C exchanges are in cleartext by default. However, we noticed that the source code for these bots includes encryption and decryption functions, shell code encodings, and support for polymorphism. It is straightforward to enable the use of these encryption routines for command obfuscation. The performance of BotProbe on these bots was identical to Spybot, i.e., it took two or four rounds of probes, depending on the command.

3. Bot binaries with obfuscated communication.: Next, we tested on a recent bot binary (W32.Wargbot as labeled by Symantec) captured in the wild. The botmaster put an encrypted command (shown below) in the IRC `TOPIC` message for bots to execute upon joining the channel. Subsequently, BotProbe automatically performed the Single-Binary-Response-Hypothesis algorithm, and it took only one extra probe to declare the bot because the bot had background scanning behavior.

```
!Q ;\\|!Q <W:Z<Z=B=B=>;P;E;E<[=;<Y=>=:<S<U<W;D<U===;;E<V<[=@<W<U=B===@=G;E<V
<[=@;I;O;N;O;E;G;G;N;G;J;H;L;G;O;H<Q;M;J;F;K;D<\\=><Y
```

We then enabled the Interleaved-Binary-Response-Hypothesis algorithm on the same bot. Again BotProbe took two total rounds to declare the bot, and this time it reported observing a third-party access response (the bot initiated an HTTP GET request when it received the `TOPIC` command), which was suppressed in the previous test because BotProbe automatically chose Single-Binary-Response-Hypothesis once it observed scanning behavior (which yielded a larger walk in TRW than the case observing a third-party access response, as discussed in Section III-C).

This third-party access response is interesting because it established some mapping between the obfuscated command and the corresponding visited URL. By intentionally changing a different portion of the obfuscated command and watching the corresponding URL, we can perform a chosen-ciphertext attack to crack the obfuscation scheme. BotProbe again demonstrated its extra utility in automating the informed active probing and collecting the mapping for our analysis, in addition to detecting the bot. By interrogating the bot with single byte modifications using BotProbe we were able to reverse engineer the encoding scheme used by the bot. The actual command after decoding is

Table I
BOTPROBE TEST ON SPYBOT AND RBOT.

Bot	Original cmd	Obfuscated cmd	IRC response	Activity response	No. rounds
Spybot	info	otlu	Version:... cpu:...	-	4
	passwords	vgyy}uxjy	Error operation failed	-	4
	scan 192...	yigt&7?847<>477<4<&79?&7	Portscanner startip:...	scan	2
	download http:...	ju}trugj&nzzv@55oxi...	download http:...	3rd-party access	2
Rbot	.id	-	[MAIN]: Bot ID: rx01.	-	4
	.sysinfo	-	[SYSINFO]: [CPU]: ...	-	4
	.scan 192...	-	[SCAN]: Port scan started...	scan	2

```
" F |" e http://img2.freeimagehosting.net/uploads/03bd27490b.jpg
```

Here the original obfuscated `!Q␣` (`"␣`) is likely to be a command prefix and `|` is a separator between commands. We are unsure about the meaning of the translated "`F`" command, but suspect that "`e`" is a download command followed by a URL. Breaking the encoding/decoding scheme is interesting because it enables us to decode other commands we observe for different variants of this botnet. In our honeynet, we have observed at least two other commands issued by the same botnet (with different bot binaries).[9] The changes in commands reflected relocation of binary hosting websites and file names. Apparently, the original hosting site (media.pixpond.com) was no longer available, so the botmaster switched to two other websites (imgplace.com and img2.freeimagehosting.net).

Although we successfully launched a chosen-ciphertext attack to break the encryption scheme of some botnets with the assistance of BotProbe, there are still some cases where we could not break the scheme. However, these instances were all successfully detected as botnet C&C by BotProbe. We captured a contemporary bot in our honeynet, which is labeled as Trojan.Dropper.Sramler.C by several AV tools. This bot uses C&C obfuscation and it makes several DNS requests, which all translated to the same IP address, demonstrating multiple degrees of stealthiness. The command we observed (shown below)[10] is apparently an update/download command because BotProbe successfully identified a third-party access response (using Interleaved-Binary-Response-Hypothesis algorithm and probing only two rounds), i.e., a download from http://220.196.X.107/packed_7711.exe.

```
=xAgVMf81RvN+xBBhG+xXwttpTsaSBfWeekvMkmkVNcbo20jZvmkCo7CUUbRsdRPzz6wiS1O
Y8pcXg3d9ucVufq2bgQ1mvh+9OBJDwIuw1kOamPaw+2jw/CTaWVQRjrX8Xl2Iph
```

After a few months, we captured a new variant of this bot, which is labeled as Backdoor.Win32.IRCBot.aby by several AV tools. We verified that this is essentially the same botnet as the aforementioned botnet, as they both contacted the same IRC server 220.196.X.226, The bot observed in June contacted port 3938 while the later bot contacted the server on port 2234 with the following command:[11]

```
=YXCdm8MDxhmOoBo3aSrxyp83pM5yZRnQVt8O+mVxm9bwLd77Ahc6KWKVn/DWu+ACn4mrpT
j6U5+yXie37WfPaymQmLtbkxPUVB2JaMwddAVokDxqsbjxmPlqpjeQIh
```

It turns out that this is actually an access to 220.196.X. 107/kk.exe, and BotProbe took only two rounds to flag this as a botnet C&C communication. To conclude, BotProbe has a 100% detection rate in recognizing IRC-based botnet C&Cs, despite the presence of obfuscated communication.

C. User Study on Normal Chat Probing

Now we need to test the false positive rate, i.e., how frequently could normal chatting sessions be mislabeled as botnet C&C using BotProbe techniques. We explore this issue through a user study on about one hundred users.

Study design and ethical guidelines: Since we are not allowed to *directly* alter live network flows on campus, we recruited human users to go online and chat with real users at diverse channels on multiple networks. During the chat sessions, our human users periodically sent crafted messages that simulate the effect of botnet probing. Our goal was to confirm our hypothesis about human response to tampered messages and evaluate the degree to which simulated BotProbe techniques affect normal users, e.g., how many actual rounds would we need on average to detect a normal user? While our current study is limited to two different chat platforms, IRC and meebo.com (a website providing instant messaging and chat room capabilities), we believe that our results hold across chat platforms because they simply capture basic human responses.

Our study protocol was reviewed and approved by the institutional review board (IRB). To alleviate any privacy concerns, we anonymized usernames and IP addresses and recorded only the following necessary information: messages exchanged and timestamps. Furthermore, although we introduced additional network flows, our methodology caused no interference to existing IRC network flows.

Participant selection: We logged into different IRC/meebo sites/channels and randomly selected active chat users who were exchanging chat messages in the channel when we logged in. We started engaging them in conversations just like normal users. The users we contacted were not aware

[9] We know that it is the same botnet because the binaries use the same C&C channel.

[10] At first glance, this looks like a BASE64 encoded string. However, we verified that this is not the case, at least not a pure BASE64 scheme.

[11] The fact that the same IP address remained as the C&C server for over 3 months suggests that obfuscated botnets might be more resilient to detection.

of the study or the active probing techniques/algorithms that we employed. This was necessary to ensure the fairness of our testing procedure.

Study procedure: We designed six different question sets to test on 123 different users. Our question set includes simple messages like "what's up," "nice weather," "you like red?" "how may I help you?" "English only! I play nice fun" and Turing test messages such as "what's 3+6=?" As we conversed with a user on a chatting channel/room using a random question set, we deliberately introduced probing at certain predefined points. We then measured the user's responses to these tampered messages. The conversations we recorded could be broken down into two classes.

First, although we randomly chose a user who seemed to be active in the chat room/channel, there is always a chance that the user does not respond to our overtures. Such cases occurred 26 times (no active replies to our messages). We discount these cases from subsequent analysis. Second, if the user was willing to pursue a conversation, by responding to our first question, we followed by sending two or three rounds of repeated questions that interleave original and slightly tampered messages (by introducing a typo in the first few bytes of the message). Some examples of tampered messages include "waat's up," "noce weather," "aou like red?" "Bow may I help you?" "Eaglish only! I play nice fun." This simulates the behavior of BotProbe performing P1/P2 probing. We recorded the exchanged messages for evaluating the Interleaved-Binary-Response-Hypothesis algorithm. In addition to P1/P2 probing, we subjected the user to P0 probing using the Turing-Test-Hypothesis algorithm described above.

Table II
USER STUDY OF PERFORMING P1 AND P2 PROBING, USING THE INTERLEAVED-BINARY-RESPONSE-HYPOTHESIS ALGORITHM. MOST USERS ARE DETECTED AS NORMAL IN TWO OR THREE ROUNDS.

	meebo chats	**IRC chats**	**Total**
Detected in 2 rounds	63 (75%)	10 (77%)	73 (75.3%)
Detected in 3 rounds	8 (9.5%)	1 (7.7%)	9 (9.3%)
Pending after 3 rounds	13 (15.5%)	2 (15.3%)	15 (15.4%)
Total	84	13	97

User study of Interleaved-Binary-Response-Hypothesis: In total, we tested 97 different users, 84 on meebo and 13 on IRC. A simulated BotProbe can detect most of the normal users (75.3%) in just two rounds and 9.3% in three rounds. The rest (about 15%) are marked still pending. We provide a summary of our results with respective to breakdowns for meebo and IRC in Table II. We set our probing to be three rounds to limit annoyance/interference to chat users. We further believe that most of the pending sessions can be easily declared as normal users by sending additional probes (we selectively verified this on a few cases). Finally, we did not encounter any false positives (misclassifying a normal user as a bot) in our limited testing.

User study of Turing-Test-Hypothesis: In addition to P1 and P2 probing, we tested P0, i.e., injecting Turing test messages (but without user education). We performed tests on 30 different users in meebo. The basic question/puzzle we sent is "what's 3+6=?" Although all users provided the correct answer upon repeated interrogation, we found it difficult to get a direct answer the first time the question is posed. These users tend not to answer this in a correct way, possibly because they thought it might be unnatural to receive such Turing questions in the chatting channels (they perceive this to be some sort of a joke). We conclude that if users are not educated to be familiar with such Turing tests or have an unusually strong desire to be in a channel, it is difficult to perform generic Turing tests on IRC or meebo networks. This also illustrates that although P0 probing seems simple and effective (if users are educated), there is still a need for alternative and transparent techniques that require no explicit user education (like our P1-P5 techniques).

V. POLICY IMPLICATIONS AND LIMITATIONS

A. Policy Concerns

It is likely that in some cases there are legal "bots," e.g., some client-side legitimate programs or automatic scripts that build their application logic over the chat protocols such as IRC. For instance, some chat bots [8] can also be detected by BotProbe. A possible solution is to whitelist these legitimate applications if they are very important and critical, and do not want to be disturbed (we expect such applications to be very few). However, we think probing a pure chat bot is not very critical, and arguably, the detection of such a chat bot is not considered as a false positive. Furthermore, there are several heuristics that can help differentiate these chat bots from real malicious bots. For example, unlike malicious bots, chat bots are unlikely to generate activity responses (e.g., scan response). In addition, we can consider a group property (similar to the group analysis in BotSniffer [14] and BotMiner [12]) to differentiate a malicious botnet, where clients in the same channel are mostly bots, from a normal human chat channel with mostly human and very few chat bots.

Our active probe techniques might be deemed controversial because they alter network flows to human users, and may replay malicious commands. In Section III, we have discussed the tradeoff between detection accuracy and disturbance to human users and various means to mitigate interference with legitimate chat sessions. We now consider potential policy implications of replaying a "potentially malicious command packet." First, we argue that to minimize liability issues, the only packets that should be tampered with by BotProbe are packets that are inbound to the local network. Second, this potentially malicious command has already been transmitted into our network and executed on

the local host prior to our probing. Third, if the purpose of the command is information gathering (e.g., `.sysinfo`), then we argue that the first command-response already leaks enough information, and our further replay most likely does not leak more information or perform more harm. In short, although controversial, we believe that the benefits of actively probing suspicious sessions could outweigh the potential disturbance in many cases. We believe a more formal study of cost-benefit analysis and risk assessment is needed in this area. We leave this as our future work.

B. Limitations and Potential Solutions

As stated in Section II, BotProbe has clear assumptions to limit its application to a certain class of botnets that use chatting-like C&C. Next, we describe some possible evasions,[12] although we have not observed real examples yet, and discuss some potential solutions.

Strong encryption: Active probing techniques cannot identify botnet C&C channels that use strong encryption schemes (e.g., SSH, SSL) making them resilient to replay attacks. Note, however, that existing passive perimeter monitoring strategies cannot detect such channels either, and most contemporary IRC bots avoid or use weak encryption/obfuscation schemes. At a minimum, BotProbe raises the bar and forces all botnets to adopt strongly encrypted communications. Arguably, using such strong encryption channels sometimes may actually expose them as suspicious and thus not desired by botmasters in some cases.[13] We envision that combining both networ- and host-based monitoring could be helpful in recognizing botnets using strong encryption, and leave it as our future work.

Timer-based evasions: Knowledgeable adversaries could design bots to have programmed timers that greatly delay the response time (the time between the command and response), or limit the number of commands of the same type that could be issued to the bot in a certain time window. By using such timers, a bot can potentially evade our Single-Binary-Response-Hypothesis algorithm. Note, however, this would also reduce the efficiency of the botnet because the botmaster cannot command the botnet promptly, or repeat the same task for a certain time. Our potential solution against such an attack is to randomize the delay in command replays.

Stateful C&C protocols: Our P1 and P2 probing techniques assume a stateless C&C protocol, *i.e.*, we can replay the observed command several times, and the bot always responds similarly to the same command. In the future, botmasters may create a stateful command processor that can detect duplicate commands, e.g., by using a timestamp or sequence number with every command sent, making simple replay ineffective. Note, most contemporary IRC botnet command-response protocols are stateless and deterministic. In addition, our P0 probing can still work even in this evasion. Moreover, to counter this possible future evasion, we describe a potential solution if there are multiple command-response rounds and multiple clients in the monitored network. Instead of replaying packets, we could intercept and modify chatting packets sent to subsequent clients by using P4 and P5 probing techniques. By intentionally modifying the command sent to some clients while leaving commands to other clients untouched, we could measure the difference in response between messages, which would be analogous to replaying the command to the same client several times in an Interleaved-Binary-Response-Hypothesis test.[14]

Finally we envision that given the complex nature of botnets, a combination of different techniques (network- and host-based, passive and active) is probably necessary for future botnet detection. Although BotProbe is imperfect and limited, it has its unique detection merit to contribute to multi-perspective botnet detection.

VI. Related Work

Several recent papers propose different approaches to the botnet detection problem. Livadas *et al.* [20], [27] proposed a machine-learning-based approach for botnet detection using some general network-level traffic features of chat traffic protocols such as IRC. Karasaridis *et al.* [17] studied network flow-level detection of IRC botnet controllers for backbone networks. Ramachandran *et al.* [24] proposed using DNSBL counter-intelligence to find botnet members who generate spam. Rishi [10] is a signature-based IRC botnet detection system that tracks IRC bot nickname patterns. BotGraph [35] is a tool to detect botnet spamming attacks targeting major Web email providers. Binkley and Singh [4] proposed combining IRC statistics and TCP work weight for detection of IRC-based botnets. Giroire *et al.* [9] proposed to track the persistence of new connection destination that is not already whitelisted to identify suspicious C&C destinations. Wurzinger *et al.* [32] proposed an automatic method to generate network-level botnet signature/model of a given bot binary based on the botnet command-response pattern. Yen and Reiter proposed TAMD [34], a system that detects centralized botnets by aggregating traffic that shares the same external destination, similar payload, and that involves internal hosts with similar operating systems. BotHunter [13] is a passive bot detection system that uses IDS dialog correlation to associate IDS events to a bot infection dialog model. BotSniffer [14] and BotMiner [12] are two botnet detection systems that utilize

[12] Most of these evasions are against Single-Binary-Response-Hypothesis and Interleaved-Binary-Response-Hypothesis algorithms. That is, the Turing-Test-Hypothesis algorithm could still work.

[13] In many cases, a simple obfuscation scheme such as substituting "scan" with "hello" is much less suspicious than using strong encryption protection.

[14] Here we assume the C&C is one-to-many, *i.e.*, one command to many clients in the network.

horizontal correlation to perform a spatio-temporal group analysis *across multiple hosts*. In [8], entropy and machine-learning-based approaches are proposed to detect chat bots (not botnet C&C). This work has a similar limitation in that it requires observing many chat messages before making a decision and thus is not suitable for detecting infrequent botnet C&C interactions. In a short summary, the aforementioned systems are all *passive*, and this paper describes several *active* botnet-probing techniques, which have unique advantages and can complement existing detection schemes.

Several other papers discuss various means to modify network traffic for security purposes. Protocol-scrubbing [31] techniques modify network flows transparently to remove ambiguities from flows that can reveal implementation-specific details of a host's operating system. Traffic normalization [15] is a technique to limit evasion opportunities by eliminating potential ambiguities before the traffic is seen by the IDS monitor. Kaleidoscope is an in-line system that protects honeynets by dynamically shuffling network address blocks [33]. These techniques need to process and then forward *all* packets sent to the network. In comparison, BotProbe needs to inject packets very rarely and affects only *a very small number of flows*.

VII. Conclusion and Future Work

We proposed the idea of using active probing techniques to detect botnet C&C communications that use chat-like protocols. By requiring the observation of *at most one* round of actual C&C interaction and then applying active probing, this approach, unlike existing passive approaches, can actively collect evidence and shorten the detection time. We have developed a hypothesis testing framework and a prototype system implementation that effectively separates deterministic botnet communication from human conversations, while providing control over false positives and detection rates. We validated our system on several contemporary malicious IRC bots and conducted an actual user study on around 100 users. Our experimental results, while preliminary, are encouraging. BotProbe is not intended to replace existing passive detection approaches, but to complement them from a new perspective.

This work represents the first feasibility study of the use of active techniques in botnet detection; thus, we hope to inspire new thoughts and directions in the research community. While controversial and clearly limited, BotProbe has demonstrated its effectiveness in detecting a large portion of contemporary real-world botnets. In future work, we will study robust, practical, and less controversial extensions of active techniques and apply to a more general class of botnet C&C detection (e.g., applicable to HTTP- and P2P-based botnets). In addition to detection, active techniques can be used for other purposes, e.g., server-side probing and injecting watermarks to trace the location of botmasters. We plan to investigate these new potential utilities of active techniques in the future.

Acknowledgment

The authors would like to thank Jon Giffin, Nick Feamster, Roberto Perdisci, and Junjie Zhang for comments on an early version of this paper, and thank Mike Hunter for the help in user study. This material is based upon work supported in part by the National Science Foundation under grants no. 0716570 and 0831300, the Army Research Office under Cyber-TA Grant no. W911NF-06-1-0316, the Department of Homeland Security under contract no. FA8750-08-2-0141, and the Office of Naval Research under grant no. N00014-09-1-1042 and N00014-09-1-0776. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Army Research Offic, the Department of Homeland Security, or the Office of Naval Research.

References

[1] Hi-performance protocol identification engine. http://hippie.oofle.com/.

[2] Shadowserver. http://shadowserver.org.

[3] P. Barford and V. Yegneswaran. An inside look at botnets. Special Workshop on Malware Detection.

[4] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, 2006.

[5] M. Collins, T. Shimeall, S. Faber, J. Janies, R. Weaver, M. D. Shon, and J. Kadane. Using uncleanliness to predict future botnet addresses. In *Proceedings of the 2007 Internet MeasurementConference (IMC'07)*, 2007.

[6] Cyber-TA. Multi-perspective malware analysis. http://www.cyber-ta.org/releases/malware-analysis/public/.

[7] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent denial of service attacks. In *Proceedings of ESORICS*, 2005.

[8] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang. Measurement and classification of humans and bots in internet chat. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[9] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and K. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *12th International Symposium on Recent Advances in Intrusion Detection (RAID'09)*, 2009.

[10] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[11] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[12] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[13] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *16th USENIX Security Symposium (Security'07)*, 2007.

[14] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[15] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th Conference on USENIX Security Symposium*, Berkeley, CA, USA, 2001. USENIX Association.

[16] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[17] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *USENIX Hotbots'07*, 2007.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[19] R. Lemos. Bot software looks to improve peerage. http://www.securityfocus.com/news/11390.

[20] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006.

[21] V. Paxson. BRO: A System for Detecting Network Intruders in Real Time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[22] P. Porras, H. Saidi, and V. Yegneswaran. A foray into conficker's logic and rendezvous points. In *2nd Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[23] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, Brazil, October 2006.

[24] A. Ramachandran, N. Framster, and D. Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *2nd USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[25] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA'99*, 1999.

[26] SecureWorks. Bobax trojan analysis. http://www.secureworks.com/research/threats/bobax/.

[27] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *31st IEEE Conference on Local Computer Networks (LCN'06)*, 2006.

[28] A. Turing. Computing machinery and intelligence. In *Mind Vol.59*, 1950.

[29] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[30] A. Wald. *Sequential Analysis*. Dover Publications, 2004.

[31] D. Watson, M. Smart, G. R. Malan, and F. Jahanian. Protocol scrubbing: Network security through transparent flow modification. *IEEE/ACM Trans. Networking*, 12(2):261–273, 2004.

[32] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, 2009.

[33] V. Yegneswaran, C. Alfeld, P. Barford, and J.-Y. Cai. Camouflaging honeynets. In *Proceedings of IEEE Global Internet Symposium*, 2007.

[34] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the Fifth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*, 2008.

[35] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. Botgraph: large scale spamming botnet detection. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 321–334, Berkeley, CA, USA, 2009. USENIX Association.

[36] J. Zhuge, X. Han, J. Guo, W. Zou, T. Holz, and Y. Zhou. Characterizing the IRC-based botnet phenomenon. China Honeynet Technical Report, 2007.