

Fall 2011

ActiBot: A Botnet to Evade Active Detection

Xinjun Zhang
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects



Part of the [Computer Sciences Commons](#)

Recommended Citation

Zhang, Xinjun, "ActiBot: A Botnet to Evade Active Detection" (2011). *Master's Projects*. 205.
DOI: <https://doi.org/10.31979/etd.8xu4-mqag>
https://scholarworks.sjsu.edu/etd_projects/205

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

ActiBot: A Botnet to Evade Active Detection

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Submitted By:

Xinjun Zhang

December 2011

© 2011

Xinjun Zhang

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Mark Stamp

Dr. Sami Khuri

Dr. Richard Low

APPROVED FOR THE UNIVERSITY

ACKNOWLEDGEMENTS

I am indebted to my advisor, Dr. Mark Stamp, for helping me to complete this paper and giving me a lot of useful and valuable advice, he is so nice and generous to provide his kind help whenever I turn to him. Dr. Mark Stamp always supported me when I working in my master program. And thanks to Dr. Sami Khuri and Dr. Richard Low for serving on my defense committee.

Thanks to my family and friends, especially my mom and dad, for supporting my studies and my life since I was born.

Abstract

In recent years, botnets have emerged as a serious threat on the Internet. Botnets are commonly used for exploits such as distributed denial of service (DDoS) attacks, identity theft, spam, and click fraud. The immense size of botnets, some consisting of hundreds of thousands of compromised computers, increases the speed and severity of attacks.

Unlike passive behavior anomaly detection techniques, active botnet detection aims to collect evidence actively, in order to reduce detection time and increase accuracy. In this project, we develop and analyze a botnet that we call ActiBot, which can evade some types of active detection mechanisms. Future research will focus on using ActiBot to strengthen existing detection techniques.

Contents

Abstract.....	5
Contents	6
1. Introduction.....	11
1.1 What is a Bot?	11
1.2 Classification of Bots	11
1.3 What is Botnet?.....	13
1.4 Classification of Botnets	16
1.4.1 Command and Control botnet.....	16
1.4.2 P2P botnets	17
1.4.4 What's our target	18
2. Botnet Detection	19
2.1 Behavior-based botnet detection	20
2.2 Signature-based botnet detection	22
3. Active botnet detection - BotProbe.....	24
3.1 Key observation.....	25
3.2 Active probing techniques.....	25
3.3 Limitations and possible evasions.....	28

3.3.1 Strong encryption evasion	28
3.3.2 Timer-based evasion.....	29
3.3.3 Stateful C&C protocols	29
4. Possible evasion	30
4.1 Encrypted communication.....	30
4.1.1 Blowfish cipher.....	31
4.2 A timer to slow down response	34
4.3 Multiple channel communication.....	35
4.4 Stateful communication.....	36
5 ActiBot.....	37
5.1 Blowfish cipher	38
5.2 Stateful communication.....	39
5.3 Slow down response.....	39
5.4 Multiple channel communication.....	40
5.5 User authentication.....	41
6. Test scenarios.....	42
6.1 User authentication.....	42
6.1.1 Scenario #1	43
6.1.2 Scenario #2	43

6.2 Blowfish cipher	45
6.3 Stateful communication.....	46
6.4 High tolerance	47
6.5 Multi-channel with timer-based communication	49
7. Conclusion and future work.....	50
Reference	52

List of Figures

Figure 1: How a botnet is used to send email spam.....	14
Figure 2: Distribution of C&C botnets	19
Figure 3: An example of the false alarm rate of a behavior-based detection system	21
Figure 4: Blowfish cipher	31
Figure 5: Blowfish cipher encryption	32
Figure 6: Unauthenticated user login fail.....	43
Figure 7: Authenticated user login.....	44
Figure 8: Login succeed.....	44
Figure 9: Bot receive an encrypted command	45
Figure 10: Encrypted command.....	46
Figure 11: Bot decrypt the encrypted command.....	46
Figure 12: Being detected by P1	47
Figure 13: Being detected by P2.....	48
Figure 14: Receive and response to a correct command.....	48
Figure 15: Tolerance to a typo in command	48

Figure 16: To join a new channel 50

Figure 17: Bot appears in the new channel..... 50

1. Introduction

1.1 What is a Bot?

The word bot is short for “robot.” In software, a bot is an automated program that can execute certain commands. Generally, bots are remotely controlled by a master via one or more controller hosts. The controller is often an Internet Relay Chat (IRC) server. IRC is normally used to relay messages among client terminals [13]. A first-generation bot is a program that interacts with a chat service to automate tasks for a human, such as creating chat logs. These chat bots were designed to help operate chat rooms, or to entertain chat users [1].

Today, the main use of bots is sending chat spam. Modern chat bots deliver spam URLs via either links in chat messages or user profile links. A single bot operator, controlling a few hundred bots, can distribute spam links to thousands of users, making chat bots very profitable to the bot operator, who is paid per generated click. Other potential abuses of bots include spreading malware, phishing, booting and similar malicious activities [1].

Compromised computers are controlled by malware, which is malicious software designed to break security. Compromised hosts are usually referred to as “zombies.”

1.2 Classification of Bots

Bots can be classified as benign, gray-area, or malicious. Here, we briefly discuss each of these categories.

1.2.1 Benign bots

Benign bots include search bots, shopping bots, and telephony bots. These bots are used to automate basic and innocent tasks on Internet relay chat (IRC). For example, a searchbot is user's own personal search robot that continuously searches the Internet trying to find all the best websites it can on user's behalf. User can even ask a searchbot a question and it will talk to other searchbots to find an answer [17]. Such a benign bot provide us a more creative and convenient life. Some other countless examples can be found at bot knowledge website [14].

1.2.2 Gray-area bots

This category refers to bots that are neither clearly benign nor clearly malicious. Some examples of gray-area bots include Blogbots, xdcc, fserve bots and Trainer bots (MMORPGs) [5]. Blogbots is open-sourced software for Microsoft Outlook and Internet Explorer that allows users to subscribe to RSS or ATOM feeds. Right now, just about every web-site, blog, wiki or newsletter is supporting this format [15]. Using such bots is a new way to put the internet to work for users.

1.2.3 Malicious bots

Malicious bots are designed to coordinate and conduct attacks on networked computers. Examples of such attacks include denial-of-service [2], identity theft and sending mass spam. Spambot is one of the popular malicious bots, which is designed to collect, or

harvest, e-mail addresses from the Internet. A spambot starts out on a web page. It scans the page for two things: hyperlinks and email addresses. It stores the email addresses to use as targets for spam, and follows each hyperlink to a new page, starting the process all over [18]. Another example is the zombie computer (often shortened as a zombie), it is a computer connected to the Internet that has been compromised by a cracker, computer virus or trojan horse can be used to perform malicious task or launch DoS attack [16]. Most owners of zombie computers are unaware that their system is being used in this way. There are a lot more examples we can find out in the world. All of these bots have some key characteristics including process forking with network and file access, and propagation potential [16].

1.3 What is Botnet?



Figure 1: How a botnet is used to send email spam

The word botnet is a combination of the words robot and network. A botnet is a collection of zombie computers that are controlled by a “botmaster” [3]. Such a collection of compromised computers is usually built up over a long period of time. A botnet can be employed to unleash a DDoS (distributed denial of service) attack or used to send very large quantities of spam. DDoS attack relies primarily on brute force attack, flooding the target with an oversaturating flux of packets, using up the resources of targeted system or depleting its connection bandwidth. In February 2007, more than 10,000 online game servers in games such as Halo, Return to Castle Wolfenstein, Counter-Strike and many others were attacked by the hacker group RUS. The DDoS attack was made from more than a thousand computer units located in the republics of the former Soviet Union, mostly from Russia, Uzbekistan and Belarus. Minor attacks are still continuing to be made today [2].

Botnets’ size could vary from tens to hundreds of thousands of systems, and one single botnet may cross plenty of homes, corporate and educational networks [19]. Botnets are usually used as zombies for executing a variety of malicious tasks [4]. The size of botnets and their potential power to attack afforded by the combination their processing potency and bandwidth have resulted to a general understanding that botnets are major threat to the network security, due to the low cost of conducting a botnet attack. Kaspersky Laboratories has researched price of botnet attacks in chat rooms and clandestine websites and found the following [20]:

- Hiring a botnet for DDoS attacks costs from \$50 to thousands of dollars for a continuous 24-hour attack.
- Stolen bank account details vary from \$1 to \$1,500 depending on the level of detail and account balance.
- Personal data capable of allowing the criminals to open accounts in stolen names costs \$5 to \$8 for US citizens.
- A list of one million emails addresses costs between \$20 and \$100; spammers charge \$150 to \$200 extra for doing the mailshot.
- Targeted spam mailshots can cost from \$70 for a few thousand names to \$1,000 of tens of millions of names.
- User accounts for paid online services and games stores such as Steam go for \$7 to \$15 per account.
- Phishers pay \$1,000 to \$2,000 a month for access to fast flux botnets, which is a DNS technique used by botnets to hide phishing and malware delivery sites behind an ever-changing network of compromised hosts acting as proxies.
- Spam to optimize a search engine ranking is about \$300 per month.

From this research data, we can see that a low-cost botnet attack could result in significant harm and losses.

1.4 Classification of Botnets

Many papers try to provide taxonomy of botnets [22], using properties such as the propagation mechanism, the topology of Command and Control (C&C) infrastructure used the exploitation strategy, or the set of commands available to the perpetrator. It is known that botnets have thousands of different implementations, which can be classified into two major categories based on their topologies. The most common type is a C&C botnet. C&C botnets have a centralized architecture, which has enabled researchers to develop some feasible countermeasures to detect and destroy such botnets [23, 24].

Hence, newer and more sophisticated botnets often use Peer to Peer (P2P) technologies.

1.4.1 Command and Control botnet

A botmaster uses command and control (C&C) channel to command his/her bot army to carry out a variety of tasks. Though developers of botnet own the capability to create new C&C protocols, communications of the most contemporary C&C botnet are built on top of some popular protocols like IRC (Internet Relay Chat) and HTTP.

As HTTP botnets use HTML to communicate, naturally they try to blend into normal HTTP traffic, but HTTP botnets are also hosted on legitimate (hacked) websites and they use specially registered domain names for their purpose. One major advantage of HTTP based botnets over traditional C&C botnet is the fact that more information can be easily presented to the botmaster [6]. But as the most popular botnets, IRC-based botnet use IRC protocol for text-based instant messaging over the Internet. It is based on client/Server (C/S) model but suited for distributed environment as well [25].

The attacker's operations on IRC-based botnet have four stages.

1. The creation stage is where the attacker may add malicious code or just modify an existing one out of numerous highly configurable bots over the Internet [26].
2. The configuration stage occurs when the IRC server and channel information is collected. As long as the bot is installed on the victim, it will automatically connect to the selected host [26]. Then, the attacker may restrict the access and secure the channel to the bots for business or some other purpose
3. The third stage is the infection stage, where bots are propagated by various direct and indirect means [26]. As the name implies, direct techniques exploit vulnerabilities of the services or operating systems and are usually associated with the use of viruses.
4. The final stage is the control stage, where the attacker can send the instructions to a group of bots via IRC channel to perform some malicious tasks.

1.4.2 P2P botnets

P2P botnets is a peer-to-peer network of bots that perform malicious task as both a client and a server. Newer botnets are almost entirely P2P, with command-and-control embedded into the botnet itself. By being dynamically updateable and variable they can evade having any single point of failure. Commanders can be identified solely through secure keys and all data except the binary itself can be encrypted. For example, a spyware program may encrypt all suspected passwords with a public key hard coded or distributed

into the bot software. Only with the private key, which only the botmaster has, can the data that the bot has captured be read [7].

1.4.4 What's our target

Although recently HTTP-based botnets and P2P botnets have drawn considerable attention, we must keep in mind that one consistent and major threat to network security is still from the IRC-based botnets. It's still one of common form of botnet all over the world, as shown in Figure 2 [21]. The simplicity and flexibility fulfilled by the IRC's text-based protocol could attribute to the consistence communication channels of IRC-based botnet.

Moreover, nowadays IRC botnets have been developed to a new state in which the content of C&C IRC messages is obfuscated by exploiting a foreign language, a custom sign, or some simple obfuscation techniques like hashing, XOR, or simple substitution. There is no difficulty for these botnets to evade most of the existing and popular botnet detection method like behavior-based approaches and signature-based detection by using obfuscated IRC messages (e.g., "hi" instead of "print").

This is a truth that we've found that obscure C&C communications have been utilized in a number of existing IRC-based botnets. More and more attackers/botmasters are using IRC bots to channel Trojans containing viruses and worms onto unsuspecting Internet users' computers. IRC is one of the earliest versions of Internet "chatting," and is still widely used by groups of people with common interests. So bots were originally used by IRC members to manage access lists, run quizzes, or serve files, which make an IRC bot

more difficult to be recognized on internet by pretending an “IRC” human user. Recognizing an IRC bot infected machine is not always easy by passively collecting evidence. It has already become a hot topic for most of the information security scientist.

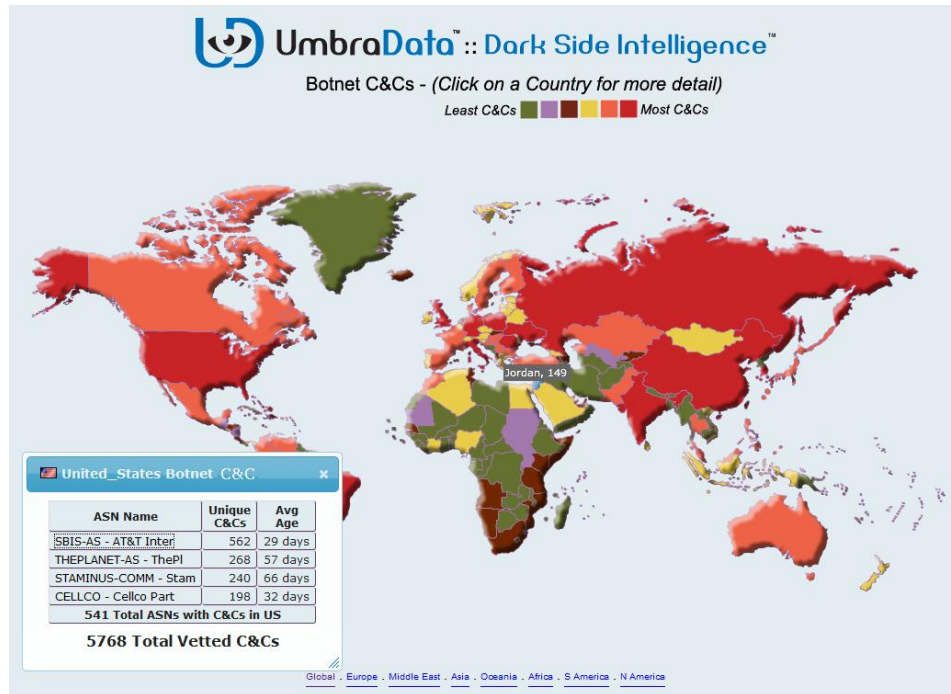


Figure 2: Distribution of C&C botnets

2. Botnet Detection

Botnet detection is a detection technology aiming to detect any bot-infected host in network by collecting evidence. Those detection methods could be classified into two classifications: passive detection and active detection. Active botnet detection method is an innovated and unique technology, which was raised by a study group in SRI International, Texas A&M University, and Georgia Institute of Technology [8].

Nevertheless most of the existing botnet detection is passive detection, which means

collecting evidence passively after any damage happens. Like the intrusion detection, passive botnet detection has two subcategories: Behavior-based botnet detection and signature-based botnet detection.

2.1 Behavior-based botnet detection

By monitoring the behavioral anomalies, behavior-based botnet detection approaches are proposed and utilized to detect botnets. Many such systems have been developed in the past few years [27] [28] [29]. Behavior-based botnet detection techniques assume that an intrusion could be detected by observing a deviation from normal or expected behavior of the system or the users [30]. The model which defines what normal and valid behavior is extracted from reference information collected by various means. The botnet detection system later compares this model with the current activity. When a deviation is observed, an alarm is generated. In other words, anything that does not correspond to a previously learned behavior is considered infection. Therefore, the botnet detection system might be complete (i.e. all infections should be caught), but its accuracy is a difficult issue, which means you will get considerable false alarms [8]. From the below figure, you can easily find the false alarm rate of one behavior-based detection system. The average of false positive over negative is about 1.4%, which would be a real concern to the users.

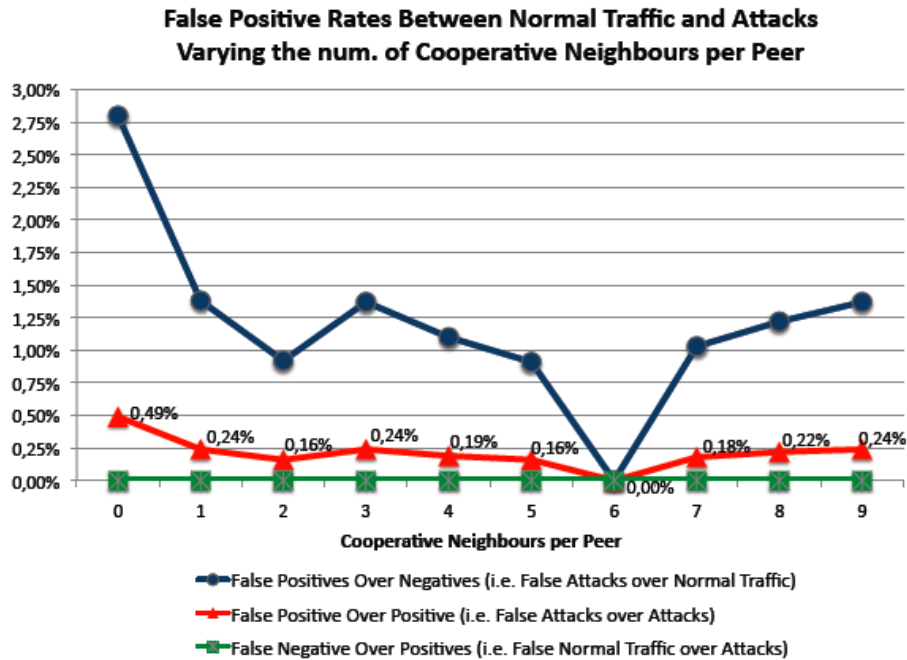


Figure 3: An example of the false alarm rate of a behavior-based detection system

Advantages of behavior-based approaches are that they can detect attempts to exploit new and unforeseen infection. They can even contribute to the (partially) automatic discovery of these new attacks [30]. They are less dependent on operating system-specific mechanisms.

The high false alarm rate is generally cited as the main drawback of behavior-based techniques because the entire scope of the behavior of an information system may not be covered during the learning phase. Also, behavior can change over time, introducing the need for periodic online retraining of the behavior profile, result to either in unavailability of the botnet detection system or in additional false alarms [31].

2.2 Signature-based botnet detection

Majority of the commercial products are based on signatures which examine the traffic looking for well known patterns of attack [32]. Signature-based detection system monitors packets sending back and forth in the network and compares with predefined and preconfigured attack patterns which are known as signatures. But the problem is that there will be lag time between signature being applied for detecting a new threat and the new threat discovered. During this lag time the detection system will be not able to identify the threat [33].

Key advantage of using signature based detection method is that signatures are easy to develop and understand if you know what network behavior you're trying to identify (that is, if you already have a copy of the exploit). However, with tools like Metasploit and new obfuscation techniques it provides this task is becoming difficult but still achievable to some extent [33].

The events generated by a signature-based botnet detection system can very precisely inform you about what caused the alert. Some of the modern logging systems also allow you to attach packet-captures with every event triggered making it easy to research on the issue

However, all these techniques above have its own limitations. Group analysis is used by some techniques to detect botnet. Nevertheless, those techniques request the occurrence of certain amount of bots in the network being monitored. What's more, they could not really work while in the network there is just one infected host. Moreover, some other detection techniques may request more time to gather enough evidence to improve it

self's confidence to identify a botnet [8]. For example, an offline correlation engine that executes daily group analysis on C-plane data is running in BotMiner [27]. Although BotSniffer is one more agile technique, which in its spatio-temporal correlation still request to observe some rounds of communication to collect enough evidence [29]. To claim an infection, BotHunter apply a primary signature-based approach to detect bot-infected host, which also requires monitoring the communication or performance from several stages of the bot lifecycle as well in order to help the detection system with enough confidence. 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 interaction is low enough, botnets could possibly evade the detection of these systems [8]. Indeed, stealthy 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, a new botnet detection system explores a new botnet detection technique which is able to collect evidence actively. The following questions are answered by active botnet detection: Can we still claim a bot-infection with a high confidence by only observing one round of obscure chat-like botnet C&C communication? How about zero round? Active botnet detect is the solution [8] which is able to achieve the goal to detect a lot of existing IRC botnets and complement some current detection approaches in the world.

3. Active botnet detection - BotProbe

Some authors believe that it is almost impossible to identify obscure chat-like botnet command and control (C&C) communications with traditional signature-based techniques, because it looks like similar to human-to-human communication, which means it is indistinguishable for computer system[8]. As mentioned above, the passive behavior-based anomaly detection techniques which exist nowadays either needs to keep monitoring several bot-infected hosts that belong to one botnet or require certain time period in collecting evidence, which makes their effect limited. In [8], the potential of active botnet probing techniques is exploited by the author as a middlebox in the network, which aims to complement and augment the existing passive botnet C&C detection approaches, it is able to easily identify infrequent C&C communication and obfuscated content in C&C message in small botnets. What's more, an algorithmic framework is introduced in [8]. To distinguish botnet C&C communications from human-human conversations, it implements a prototype system named BotProbe by using hypothesis testing with considerable accuracy. They also conducted server experiment on multiple real-world IRC bots, the result of which in this paper prove that obscure and obfuscated botnet communications can be successfully identified by their proposed active botnet detection techniques. They also conduct a study with real-world user on hundreds of participants, which indicates that when it comes to human-to-human communication the active botnet detection techniques have a very low false positive rate. A discussion on BotProbe's limitations is presented as well. New directions and ideas are expected to be

inspired in the feasibility study by using active botnet detection techniques in the malware research and botnet research communities.

3.1 Key observation

Active botnet detection posits that we should apply active execution of selecting suspicious interaction rather than passively monitoring two-way network traffic to better identify the communication of botnet. This strategy of detection named active botnet probing is based on the following two observations [8].

1. A clear command-response pattern could be found in a typical botnet C&C interaction. Thus to a certain dialog replays, a stateless bot is supposed to act deterministically, while it will be a nondeterministic interaction with a human-controlled end point.
2. Second, unlike humans, bots has limited tolerance for typographical errors during communication, because of their being preprogrammed to respond to the set of commands they receive.

3.2 Active probing techniques

Based on both key observations above, active botnet detection is used in BotProbe system, using the following active probing techniques.

P0: Explicit-Challenge-Response

There is one example explicit validation mechanism where educated users knowingly participate in the BotProbe scheme. A reverse Turing test would be prompted for users to perform in the BotProbe system, when BotProbe first detect a new IRC session between

two IP addresses. The system might also redirect the internal human IRC users to visit a secure web site where users are required to parse a CAPTCHA. In computing, this is a typical challenge-response test widely used to distinguish a response is not made from a computer [9]. A small puzzle could also be added by BotProbe for the user to solve as an alternative way. With the help of this approach, BotProbe maybe able to identify a botnet channel before capturing actual C&C interaction, in another word, BotProbe doesn't require observing any round of communication. But this is a simple and effective technique which requires user tolerance, awareness, and compliance [8].

P1: Session-Replay-Probing

The addresses of the servers are spoofed by the BotProbe monitor. And other TCP packets would be appended by BotProbe replaying the same control command to the remote end for a number of times. If a bot is the client, it will probably perform responses that are deterministic (in regarding to both timing and content) [8].

P2: Session-Byte-Probing

With this probing approach, a few bytes of the control command would be randomly permutes by the BotProbe monitor. Assume a bot is the client, and then it is expected to be highly sensitive to the changed commands by BotProbe in order to react quite differently and the whole packet may even be dropped. Nevertheless, a higher tolerance is expected for a human user chatting in an IRC channel for the typo errors of an IRC message. By interleaving strategies P1 and P2, this test might be repeated by BotProbe for several times as necessary until BotProbe has sufficient confidence and evidence to make a conclusion. The authors also discussed this algorithm (Interleaved-Binary-

Response-Hypothesis) used in BotProbe in more detail in their paper [8]. Generally, this algorithm assume that the probability of a human user responses to tampered packets (interleaved P1 and P2 probing for example) with repeatable content is pretty low, which is near to $\frac{1}{4}$ [8]. But this probability of a bot with the response is observed as 1. The main benefit of this algorithm is that they could find out a general method to detect the third-party access response which doesn't need to depend on the signatures of content.

We also note that in the same TCP session if subsequent C&C interactions occur, by inserting new TCP packets, the existing connections may be broken by approached P1 and P2. To prevent this, the numbers of TCP packet (sequence and acknowledge) would be altered by their in-line botnet probing system along with checksums to specify a new TCP packets that are produced by the P1 and P2 probes. Plus, at the application level, certain amount of interference will also be introduced into current sessions by these two probing approached. Nevertheless, the authors find out that there is another probing technique (P3) that will not impact current sessions for their targeted IRC protocols [8].

P3: Client-Replay-Probing

Users are allowed by IRC, like the other chat protocols, to send messages to each other directly. In this case, BotProbe create a new user which then logs in to the channel being monitored and sends the captured command(s) to the suspicious client(s), acting as the botmaster. By using this approach, BotProbe will not break current connections, but achieve the same goal like P1 and P2 [8].

P4: Man-In-The-Middle-Probing

None observed command packet is intercepted by any of the above techniques directly. But, in some situations, such as the simple replay-probing may not have effect in highly stateful C&Cs, BotProbe intercepts the new command, and performs message injection like a man-in-the-middle chat [8].

P5: Multi-client-Probing

However, there is an assumption for all the above techniques, which is to probe a session from one single client. In the network, when multiple likely infected clients being monitored are interacting with the same C&C server, BotProbe reduces the number of probing rounds needed to validate its hypothesis and distributes these probing approaches among multiple clients [8].

3.3 Limitations and possible evasions

According to above, BotProbe only works on a number of assumptions. It is limited to a specific category of botnets which is using chatting-like C&C. Therefore, we can easily find out some limitations in BotProbe, based on which we will discuss some possible evasions as show below.

3.3.1 Strong encryption evasion

Those botnet C&C channels which adopt strong encryption obfuscation schemes, making them resilient to replay attacks, could not be identified by active probing techniques.

Although, the existing passive botnet detection strategies are not able to detect such channels and most of the existing IRC botnets just use some or even don't use such weak encryption schemes. So in order to avoid active botnet detection, all botnets should use

strongly encrypted communication. Arguably, using such strong encrypted communication in botnet sometimes may possibly expose the bots as suspicious and therefore not expected by botmasters in some cases. So in our case, we can use a symmetric key cipher to encrypt our communication and adopt timestamp to evade the replay probing technique.

3.3.2 Timer-based evasion

Some knowledgeable attackers would design bots to use programmed timers which could randomly and delay the response time (the time between the sending out a command and observing a response) greatly, or set a range of the number of one command which could be received from the botmaster in a certain time period [8].

BotProbe replay one suspicious command in each detecting round using the P1 approach, then it records whether or not the response is expected, which is named Single-Binary-Response-Hypothesis algorithm. A bot is able to evade BotProbe's Single-Binary-Response-Hypothesis algorithm potentially, with the help of such a timer described above. Nevertheless, this approach will do harm to the efficiency of the botnet since the botmaster is not able to send C&C to the botnet in a timely manner or perform some identical task(s) repeatedly in a certain time window. A possible implementation is to use a timer to delay the bots' response time to the command from botmaster.

3.3.3 Stateful C&C protocols

The P1 and P2 probing techniques both have an assumption that the C&C protocol is stateless, i.e., BotProbe replays the captured command for certain times, and bots are

expected to always respond similarly to the replayed command. So in this case, a processor to process stateful command could be created by the botmaster in order to detect identical commands, e.g., by using sequence number or a timestamp within every command sent to make the simple replay probing technique ineffective. In our case, we would rather use timestamp to create an application-leveled stateful C&C protocol to detect duplicate commands.

4. Possible evasion

Since active botnet detection technology is still not perfect with such limitations discussed above. We're trying to attack or evade the detection of BotProbe system by implementing an advanced obscure command and control channels botnet. In this botnet, we propose the following four main features:

1. Using strong cipher to encrypt the communication.
2. Adopt a timer to slow down the bot's response.
3. Apply mutli-channel communication between botmaster and bot.
4. Using stateful communication

4.1 Encrypted communication

In our implementation, we adopt blowfish cipher, a strong symmetric block cipher, to encrypt the communication between botmaster and bot, since it is easy to be implemented and not suspicious, since symmetric block cipher is widely used in IRC communication.

4.1.1 Blowfish cipher

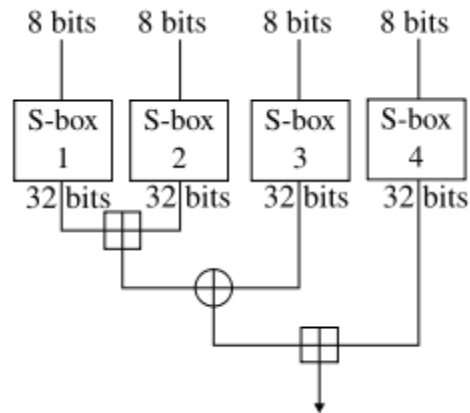


Figure 4: Blowfish cipher

Blowfish comes within a large scale encryption products and cipher suites. Its good encryption rate in software is quite impressive and there is not any effective cryptanalysis found to date [34].

Blowfish's block size is 64-bit long. Length of the variable key is from 1 bit up to 448 bits [10]. A 16-round Feistel cipher is used and large key-dependent S-boxes are adopted as well. It is almost the same as CAST-128 in structure that uses fixed S-boxes.

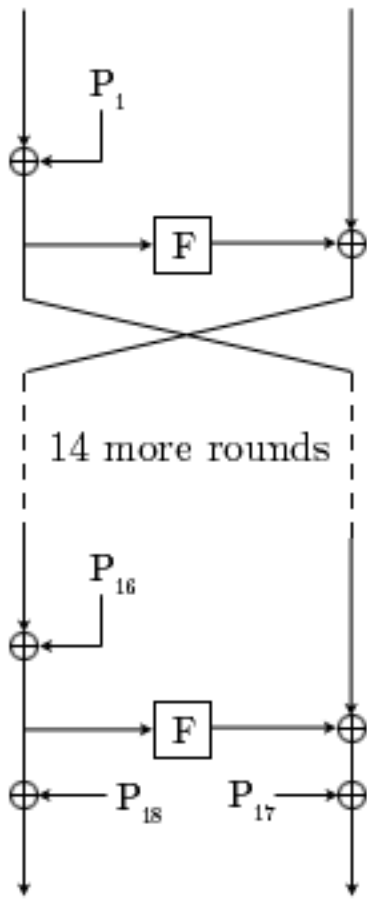


Figure 5: Blowfish cipher encryption

This diagram above shows the process of Blowfish cipher. Each line stands for 32 bits.

There are two subkey arrays kept by this algorithm: the P-array containing 18 entries and S-boxes which have four 256-entries. The S-boxes take an eight-bits-long input and generate a 32-bits-long output for the P-array. Every round there is one entry of the P-array would be used, and then after the final round, one of the two remaining unused P-entries is xored with each half of the data block [10].

The diagram shows Blowfish's F-function, this function divides the 32-bit input into four

quarters in the length of eight-bit each, which then are introduced to the S-boxes as input. At the end, the outputs of S-boxes are added modulo 232, and then a final 32-bit output is produced by xoring the result.

Similar to the encryption process, P1, P2... P18 are applied to the decryption process in the reverse order. It is quite unobvious since xor is associative and commutative. It is one of the common mistakes to adopt reverse order of encryption as decryption algorithm (for example: the first cipher text block is produced by xoring P17 and P18, then the P-entries are used in reverse order).

By initializing the S-array and P-boxes with values produced from the hexadecimal digits of pi without any apparent pattern, Blowfish's key schedule begins. Then the secret key starts to cycle the key byte to byte as needed, xored with every input from P-array one by one. A 64-bits-long block with all zero is then encrypted by the algorithm as it function, whose output as cipher text take place of P1 and P2. The identical cipher text is then used to encrypt the new subkeys, and then P3 and P4 are both replaced by the new cipher text. This procedure repeats to replace the entire P-array and all input of the S-box. After all, the Blowfish encryption algorithm needs to be executed 521 times to produce all these subkeys, which means about 4KB of data is computed [10].

As a result of the P-array is 576-bits-long, all these 576 bits are xored through the key bytes during the initialization, most of the existing implementations support key sizes up to 576 bits. But this is absolutely possible. The limit of 448 bits is used to make sure that each bit in each subkey relies on each bit in the key [10], since the last four values of the

P-array cannot impact any bit of the cipher text. This point is considered as a requirement in implementations with variable rounds, moreover this also improves the security against the exhaustive search attack, and however it weakens the security that this algorithm promises. As a result of the slow initialization of the cipher with every tiny difference of the key, Blowfish was borned with a natural defense against the brute-force attacks, which really cannot justify any key whose size is over 448 bits long [10].

In 1996, Serge Vaudenay found a known-plaintext attack needing 2^{8r+1} known plaintexts to break, where r is the number of rounds [10]. Nevertheless, there is no effective cryptanalysis found up to now against the full-round version of Blowfish [34].

We adopt Blowfish cipher to encrypt every command sent to the bot, we can efficiently prevent being intercepted and the replay probing by using a powerful but not so doubtful encryption as SSL/SSH.

4.2 A timer to slow down response

A timer implemented to delay the response time, the time between sending out the command and observing the response, is one of the effective evasion techniques. By using such a timer to delay the response time, a bot can evade one of BotProbe's assumptions which assume the bot would perform the same action after receiving a replayed command in a certain time window. So we can implement a random timer to generate some random time interval after which the bot will response to a preprogrammed command. This approach will easily confuse the BotProbe. If the response occurs out of BotProbe's time window for one round of active probing, this respond evade the detection successfully. In another case, if a respond is caught by

BotProbe, BotProbe will replay the command and expect the same response after the same time interval, which is unlikely to happen due to the randomly generated number from our timer. A random time interval will make it hard for the BotProbe to confirm an activity is the same response to a replayed command sent by the system itself. This unpredictable response would undermine BotProbe's confidence to confirm a bot infection.

In a word, the random-timer make our advanced bot act more like a human, whose activities are unique and natural.

4.3 Multiple channel communication

The basic means of communicating to a group of users in an established IRC session is through a channel. Channels are the communication medium in an IRC session. Channels on a network can be displayed using the IRC command LIST that lists all currently available channels on that particular network. Users can join to a channel using the JOIN command, in most clients available as /join #channel name. Messages sent to the joined channels are then relayed to all other users.

BotProbe's P1 and P2 probing techniques assume all the communication happens in one single channel, i.e., BotProbe replays the observed command several times to the bot in one channel, and check to see if the bot always responds similarly to the same command. So if the bot receive a command but response in another new channel, BotProbe could not detect anything abnormal, what's more, if the new channel is randomly picked by the Botmaster and Bot, it will make it harder for BotProbe's P1 and P2 probing techniques to

detect any response. BotProbe cannot even detect any identical response. So when a bot receive a command from the botmaster, our bot would randomly create and join a new channel, based on botmaster's command. Our bot would leave the current channel after notifying the botmaster with the new channel, where the bot would response to the command received in current channel. With the randomly-generated channel, it's another possible evasion to make the communication undetectable.

4.4 Stateful communication

Moreover, BotProbe's P1 and P2 probing techniques both assume a stateless C&C protocol, i.e., it will keep sending one observed command several times to the clients, and the bots are expected always responds similarly to this replayed command. So in our implementation, we could create a stateful communication that provide our bots with the capability to detect duplicate or replayed commands by including a timestamp or sequence number in every command sent. This approach could easily make simple replay probing technique ineffective. For instance, when our bot receive a command, it will first compare the timestamp received in the command with the current timestamp. Then it will determine whether this command is a replayed command or not, based on the difference of these two timestamp. If this command is considered as a replayed command, the bot will then respond to it with some random normal content, which makes our bot behave non-deterministically as a normal human user.

5 ActiBot

We implement a botnet called ActiBot, which is based on the Eggdrop botnet [11].

Eggdrop is one of the most popular and best supported, open source IRC bot, designed for flexibility and ease of use, and is freely distributable under the GNU General Public License (GPL). All features we designed are implemented over Eggdrop.

An IRC bot is a program that sits in an IRC channel around the clock, keeping it open 24 hours a day. It looks just like a normal user on the channel, but is usually idle until it's called upon to perform a particular function.

And all our implemented commands are DCC commands. Direct Client-to-Client (DCC) is an IRC-related sub-protocol enabling peers to interconnect using an IRC server for handshaking in order to exchange files or perform non-relayed chats. Once established, a typical DCC session runs independently from the IRC server. The DCC chat service enables users to chat with each other over a DCC connection. The traffic will go directly between the users, and not over the IRC network. When compared to sending messages normally, this reduces IRC network load, allows sending of larger amounts of text at once, due to the lack of flood control, and makes the communication more secure by not exposing the message to the IRC servers.

To realize those four features listed in above section, ActiBot adopts the following concrete implementations in two DCC commands (*.enc* and *.mul*):

1. Implement blowfish cipher to encrypt and decrypt the *.enc* command sent from the botmaster to a bot in ActiBot.
2. Implement a stateful communication, using timestamp as the time/state indicator in the *.enc* command. So the bot could decide how to respond to a specific command after comparing the receive time and current time.
3. By using a timer, to slow down the response to *.mul* command.
4. Create a multiple channel communication, allowing the bot response to a command sent from botmaster in a randomly created new channel after negotiation.
5. Adopt user authentication to identify the botmaster, only receive and respond to the command sent out by the authenticated user, who is recognized as the botmaster.

5.1 Blowfish cipher

Blowfish cipher is a build-in feature in Eggdrop Botnet, which is well implemented with friendly API. We use tcl script to adopt Blowfish cipher to encrypt a specific DCC command named *.enc*. The symmetric key for encryption and decryption is predefined by the user/botmaster, and is stored in a file along with the bot instance. After our bot receives *.enc* this encrypted command, it will use the key to decrypt the command. If the content of this command is validated, a preprogrammed response will be launched. Otherwise the bot will respond with some random content.

5.2 Stateful communication

Timestamp is always a good indicator of state. A timestamp is a sequence of digits, denoting the date and/or time at which a certain event occurred. A timestamp is the time at which an event is recorded by a computer. Comparing to sequence number, timestamp is more efficiency without the need to remember the sequence number in a communication session.

The UNIX epoch (or UNIX time or POSIX time or UNIX timestamp) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). So in ActiBot, we treat this number of seconds as an integer, which is easy for the use of comparison and encryption. We implement this feature in *.enc* command for a stateful communication, making the bot only response deterministically at the first time when it receives a new *.enc* command. After the first time receiving this command, the bot will turn into the second state which means if the bot receive this command again, which will then be considered as a replayed command, it will just response with some random text data as a normal human user.

5.3 Slow down response

The response time is sensitive information to whatever detection system. If a command is detected, the detection system will then keep monitoring the future traffic for the related response. If the client does not take any malicious or suspicious action in a certain time interval, the detection system would easily be seduced to consider this is a normal activity. In this way, as time goes by, the suspicion of detection system is getting close to

clear. Then after the suspicion is purged, any malicious action will not be very easy to be considered as a sign of bot infection. In ActiBot, we try to use a timer to randomly determine the response time to a command, which is between 0 seconds to 100 seconds. For instance, after the bot received a command, there is no rush to response to it immediately. Our bot will then enable a timer with a random timer interval. And it will only response after the random timer interval has elapsed.

5.4 Multiple channel communication

In ActiBot, we implemented a DCC command *.mul*, following the format: *.mul <prefix of the new channel name>*, in order for the botmaster to create a new channel for further communication with the bots. With this command, botmaster could specify the prefix of the name of a new channel. And the bot would create a new channel based on the prefix received in the command. After creating a new channel, the bot would notify the botmaster with the new channel name, displaying it in the current channel's chat console. Then the bot will leave this channel, and join to the new channel it just created then response there.

We propose this schema to create multi-channel communication.

1. Botmaster connect to a bot in a DCC channel.
2. Botmaster send out a DCC command *.mul* to the bot in such format: *.mul <#prefix of the new channel>*.

3. Bot receive the *.mul* command, and start to decide the rest of the new channel name by using a random number generator.
4. After deciding the name of the new channel by combining the prefix from botmaster and the random number generated, the bot notify the botmaster with the full name in the chat console, and then leave the current channel.
5. After 5 seconds, a time interval for the bot and Botmaster to join the new channel, the bot will provide the real or malicious response to the command *.mul* receive in previous channel.

5.5 User authentication

We use a user file to store the user information. In ActiBot's userfile, the botmaster could control who have the access to bots and what access level each of these users could have. Ban lists and ignore lists are also contained in the userfile. One of the fundamental things is userfile management which we could exploit to effectively use our advanced botnet.

When the new users first come to Eggdrop, they needs to use the *hello* command to introduce themselves to the application, by which the new users are added to the user list.

Then the users must set a password by using the *pass* command once they have been added. They need to follow the following command format */msg <botnick> pass <password>* to set their password. The *.chpass* command could be used to reset their password in the future.

Once the userfile is setup, each DCC communication could be set up successfully after a valid user is identified by Eggdrop Botnet. The scenario is once a connection to the bot is established. The user will be prompted for the password, and then placed on the party line

(the main chat area) automatically. This feature would prevent some unknown or unauthenticated user to send any command to the bot on the test or detection purpose, which is the P2 probing technique.

6. Test scenarios

Nothing could be better than to test our botnet with the BotProbe. But up to now BotProbe is only a prototype system and an experimental article, not a real product in the market. The authors of BotProbe are still investigating a more practical, robust, and less controversial extensions of active techniques [8]. So we don't have the luck to test our botnet with real active botnet detection system. Hence, we just test our botnet in our environment, and try to simulate the test scenarios as genuine as possible.

6.1 User authentication

In eggdrop, the first user is recognize as the administrator of a bot and has the privilege to give other user access to the bot. And all these setting are store in a userfile protected by MD5 cryptographic hash.

So in our case, we set up our own administrator's account and use it to control our bot as a botmaster. No other user could talk to our bot using DCC chat without the permission from botmaster.

The purpose of this test is to evade the detection probe P3 (Client-Replay-Probing) and P5 (Multi-Client-Probing). With both of these two probing, BotProbe will instantiate one or several new user(s) that log(s) into the channel and sends some observed commands to

the selected clients acting as the botmaster. Hence, with the user authentication, any unauthenticated user created by BotProbe will not be allowed to access and detect the bot. Only the botmaster and authenticated user could send command to the bots and control the whole botnet. P3 and P5 probing technique would fail.

6.1.1 Scenario #1

- a) Try to access it with an unauthenticated account.

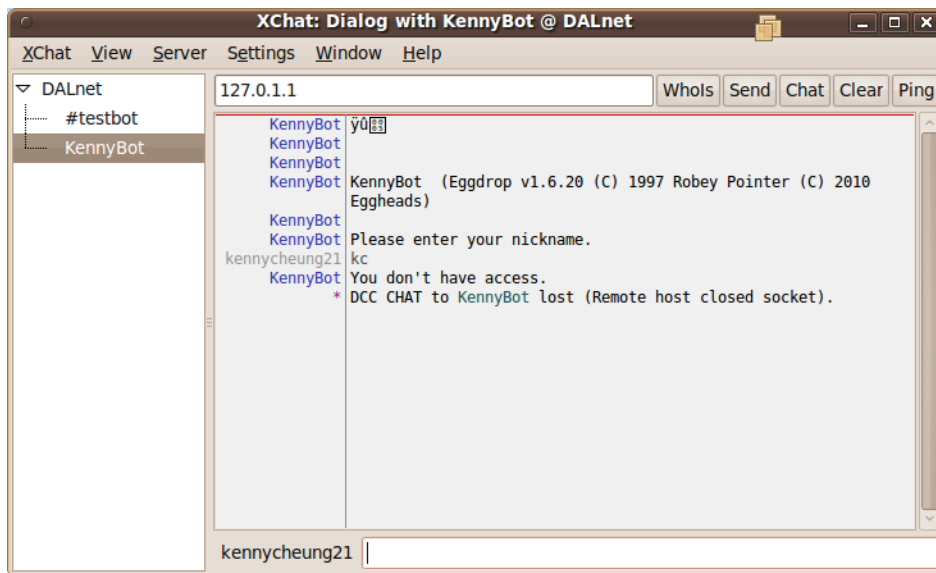


Figure 6: Unauthenticated user login fail

- b) Bot reply with error, denying the access from a strange user.

6.1.2 Scenario #2

- a) Try to access the bot with an authenticated administrator account.

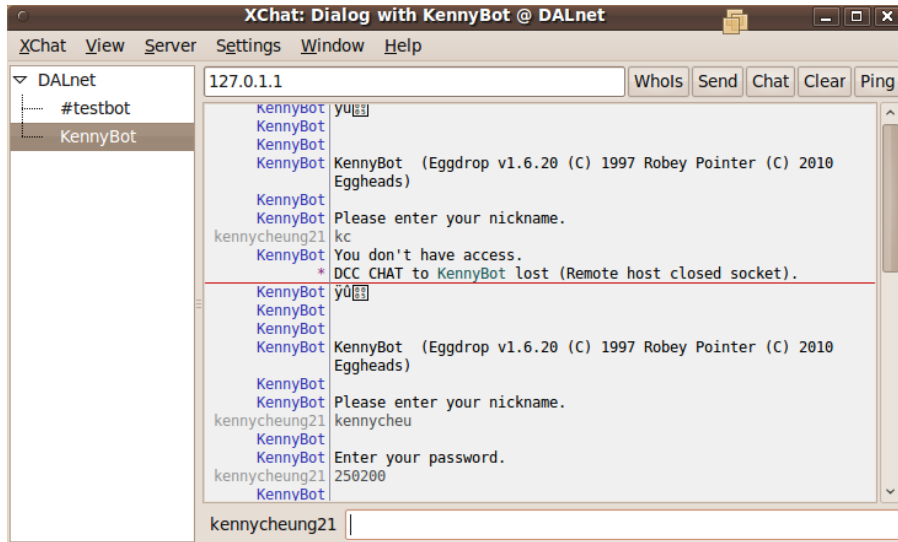


Figure 7: Authenticated user login

b) Access accepted, login successfully as a botmaster.

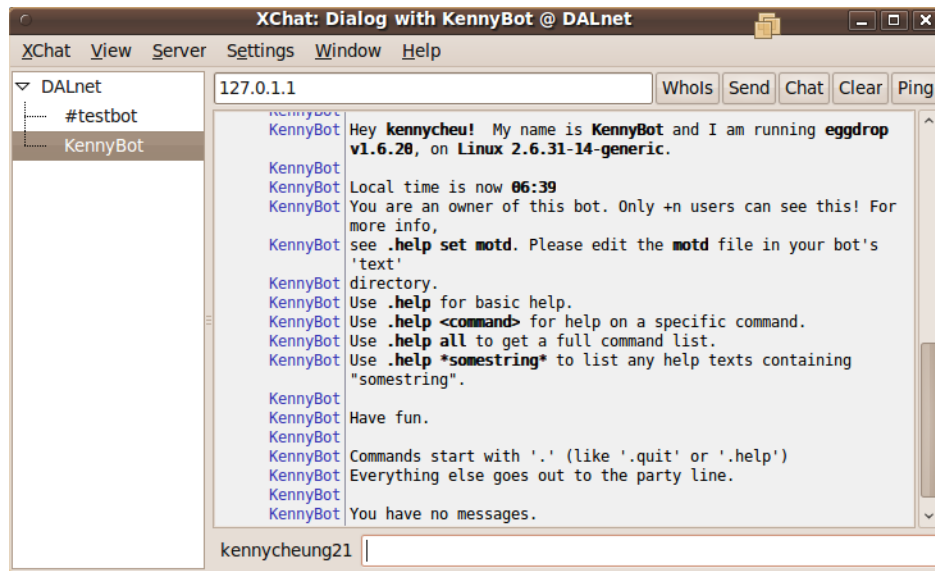


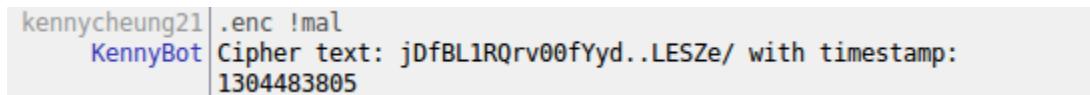
Figure 8: Login succeed

6.2 Blowfish cipher

Encrypting the communication with blowfish cipher can make ActiBot resilient to P4 (Man-In-The-Middle-Probing). Botmaster could create a symmetric key stored in the user file while setting up the whole botnet, so that this key is shared between the botmaster and bots. Botmaster then could use this shared key to encrypt the command sending to a bot. And bot can also decrypt and read the command with this shared key. In this way, ActiBot makes Botprobe hard to read and change any content of the command sending from Botmaster to Bot.

To test the implementation of Blowfish cipher, we use wireshark to capture our packet and try to read its content.

- 1) Enter a implemented DCC command: `.enc !mal`
- 2) The bot receive the command which is encrypted by blowfish cipher.



```
kennycheung21 | .enc !mal  
KennyBot | Cipher text: jDfBL1RQrv00fYyd..LESZe/ with timestamp:  
1304483805
```

Figure 9: Bot receive an encrypted command

In the above figure, we can clear see the cipher text of the encrypted command.

- 3) In the wireshark, we can intercept the package and see the workload.

```

▷ Frame 1 (95 bytes on wire, 95 bytes captured)
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol, Src: 127.0.1.1 (127.0.1.1), Dst: 127.0.1.1 (127.0.1.1)
▷ Transmission Control Protocol, Src Port: 45092 (45092), Dst Port: 40005 (40005), Seq: 1, Ack: 1, Len: 29
▼ Data (29 bytes)
  Data: 2E656E63206A4466424C31525172763030665979642E2E4C...
  [Length: 29]

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 51 7b 98 40 00 40 06 bf 0c 7f 00 01 01 7f 00  .Q{.@.@. ....
0020  01 01 b0 24 9c 45 86 0f a8 7d 86 86 aa c0 80 18  ...$.E.. }. ....
0030  04 23 00 46 00 00 01 01 08 0a 00 07 5b fc 00 06  .#.F.... ...[...
0040  f2 d6 2e 65 6e 63 20 6a 44 66 42 4c 31 52 51 72  ..enc j DfBL1RQr
0050  76 30 30 66 59 79 64 2e 2e 4c 45 53 5a 65 2f    v00fYyd. .LESZe/

```

Figure 10: Encrypted command

- 4) The bot receive it and decrypt the command correctly.

```

kennycheung21 .enc !mal
KennyBot Ciper text: jDfBL1RQrv00fYyd..LESZe/ with timestamp:
1304483805
KennyBot Plain text: !mal with time difference 1

```

Figure 11: Bot decrypt the encrypted command

6.3 Stateful communication

In order to evade the P1 (Session-Replay-Probing), we use timestamp as the state indicator to establish a stateful communication. We set the threshold to 2 seconds, allowing the latency of network delay.

In order to simulate the P1 replay probing, we set up our botnet to send out an *.enc* DCC command 3 second after the botmaster initiate sending it out. This encrypted command also contains a timestamp denoting the date and time when the botmaster initiate this action. After comparing the current timestamp from system with the timestamp within the

command, which shows this command is generated at 3 or more seconds before, the bot would consider this package as a replay package and response with some random content in order to evade this detection. The following is the test scenario.

- 1) Send the command `.enc !mal` 3 seconds after the botmaster initiate it.
- 2) The bot received such a package and compare the timestamp and find out it exceeds the threshold (2 seconds). So the bot consider this is a replay command. And it will not try to decrypt this command, then just response to the command with some random content.

```
kennycheung21 | .enc !mal
KennyBot | Cipher text: chlQX1UTnbb1PPgOV0JNoIx. with timestamp:
           | 1303457290
KennyBot | Maybe I'm being detected by P1! Randomly response with a
           | number 42
```

Figure 12: Being detected by P1

6.4 High tolerance

By using P2 (Session-Byte-Probing) detection techniques, the BotProbe monitor will permute certain bytes in the observed application command randomly, which is challenging the tolerance to typographical error.

So in order to evade P2 detection, we use the wildcard to increase our bot's tolerance to typographical error. In this way, while BotProbe permutes some bytes of our command, the bots in ActiBot will still recognize this as an acceptable command and response to it. To test this, we try to input three “.enc” commands with different content, simulating the P2 probing.

- 1) Input the wrong format with “.enc kenny”, which is not follow the format requirement starting with the exclamation mark “!”.

```
kennycheung21 .enc kenny
KennyBot Cipher text: hSRIV.z3fWi0k2j/y1kJFre. with timestamp:
1303457940
KennyBot Maybe I'm being detected by P2! Randomly response with a
number 11
```

Figure 13: Being detected by P2

The bot didn't recognize this command, and response with an alert for P2 probing and some random content.

- 2) Input a correct command with “.enc !mal”.

```
number 11
kennycheung21 .enc !mal
KennyBot Cipher text: jwfz/.FJ3901xgz0c0d9y/H0 with timestamp:
1303458082
KennyBot Plain text: !mal with time difference 1
```

Figure 14: Receive and response to a correct command

The bot consider it with a validated command and decrypt it successfully.

- 3) Input another command with a slight modification “.enc !laml”.

```
kennycheung21 .enc !laml
KennyBot Cipher text: d7yAa1cLwGJlwx05Z.2p0az. with timestamp:
1303458185
KennyBot Plain text: !laml with time difference 1
```

Figure 15: Tolerance to a typo in command

The bot still consider it as a validated command and decrypt it successfully.

6.5 Multi-channel with timer-based communication

Both purpose of slowing down the response time and creating multi-channel communication are make it more difficult for BotProbe to capture the response from the bots in an ActiBot.

We integrate the multi-channel communication with timer-based communication in a DCC command *.mul*. With this command, the botmaster requires the bot to response in a new channel which is different from the current channel. Since the new channel is decided by both the botmaster and the bot, it's very difficult for a detection system to associate this response with the correct command captured in a different channel. Moreover, the response is also hard to be considered as a response to a DCC command, due to the response time is longer than usual. Even the response is a malicious action, which would draw the BotProbe's attention; BotProbe has to take a lot of effort to find out the exact corresponding command before performing any active detection. For example to replay the command and wait for the expected same response from the bot, if the replayed command is not the correct command, the bot in ActiBot will give out a different response, which decrease the confidence of Botprobe to recognize it as a bot.

- 1) Our bot join in the channel #testBot. Botmaster is talking to the bot in this channel.
- 2) Input the command: “.mul #testMul”. “#testMul” is the prefix of the new channel name decided by Botmaster.

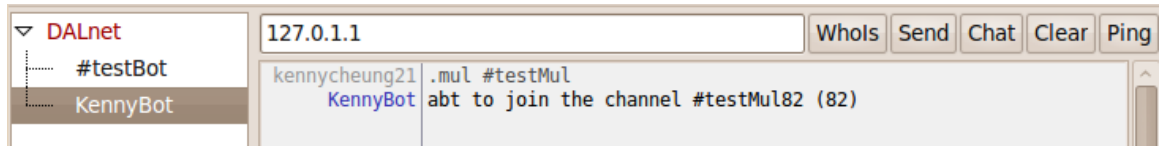


Figure 16: To join a new channel

The bot receive this command, and decide the rest of the name of new channel with a random number, which in our case above is eighty-six. Then notify the Botmaster before it leave.

- 3) The bot appear in the new channel “#testMul82” and leave the old channel. After 5 seconds, in the new channel #testMul82, the bot response to the *.mul* received in the old channel #testBot.

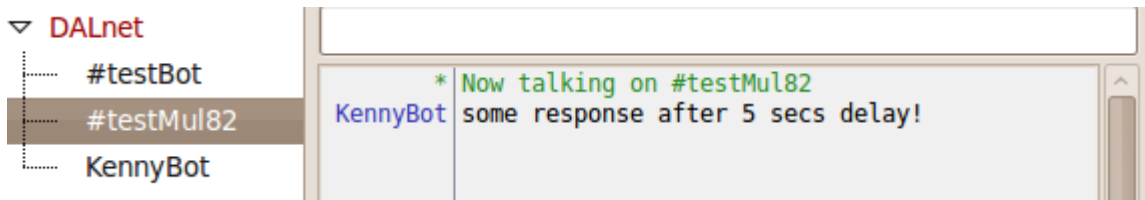


Figure 17: Bot appears in the new channel

7. Conclusion and future work

In this paper, we discussed and analyzed ActiBot, a botnet designed to evade active detection. Specifically ActiBot was designed to evade the BotProbe active detection system [8]. We considered the active botnet detection techniques employed by BotProbe and discussed their limitations. The design of ActiBot was then discussed, and we analyzed ActiBot from the perspective of active detection.

It is important to note that the purpose of this paper is not to encourage malicious

activities. Instead, we hope to lay the foundation for further research into botnet detection. We plan pursue such research in the future.

We believe that ActiBot would easily evade all of the active detection techniques employed by BotProbe, with the exception of the explicit-challenge-response.

Overcoming this limitation would become a focus of further work of ActiBot. Of course, we also plan to pursue improved detection mechanism that can be used to detect ActiBot and similar botnets.

There is no doubt that active botnet detection is promising in the realm of botnet detection research and market. But Active botnet detection definitely could be improved and augmented by learning from some passive botnet detection approaches. For example, BotProbe could utilize the database of abnormal behavior defined by some other botnet detection system, so that BotProbe could improve its efficiency in detecting some known botnets even without actively probing the bots' communication.

Other than this, botnet detection research should utilize the advantages of both active botnet detection and passive botnet detection in improving the defect of both detection approaches.

Reference

1. **Gianvecchio, S.; Xie, M.; Wu, Z.; Wang, H.** Measurement and classification of humans and bots in internet chat. 2008. *Proceedings of the 17th conference on Security symposium*. pp. 155-169.
2. Denial-of-service attack. *Wikipedia*. [Online] http://en.wikipedia.org/wiki/Denial-of-service_attack.
3. What you need to Know about Botnets! *Michigan Department of Technology*. [Online] 2007.
http://www.michigan.gov/documents/cybersecurity/CSNewsletter_September2007_207450_7.pdf.
4. **Ramachandran, A.; Feamster, N.** Understanding the network-level behavior of spammers. *ACM SIGCOMM Computer Communication Review*. 2006, Vol. 36, 0146-4833.
5. **Daswani, N.; Stoppelman, M.** The anatomy of Clickbots.I. : USENIX Association, 2007.
6. **Cymru, Team.** A Taste of HTTP Botnets. 2008.
7. Botnet. *wikipedia*. [Online] <http://en.wikipedia.org/wiki/Botnet>.
8. **Gu, G., et al.** Active Botnet Probing to Identify Obscure Command and Control Channels. s.l. : IEEE, 2009. ISSN:1063-9527.

9. CAPTCHA. *Wikipedia*. [Online] <http://en.wikipedia.org/wiki/CAPTCHA..>
10. **Schneier, B.** Description of a new variable-length key, 64-bit block cipher (Blowfish). *Fast Software Encryption*. s.l. : Springer, 1994, pp. 191--204.
11. EggHeads.ORG *eggdrop development*. [Online] Eggheads Organization.
<http://www.eggheads.org/>.
12. **Ramachandran, A.; Feamster, N.** Understanding the network-level behavior of spammers. *ACM SIGCOMM Computer Communication Review*. 2006, Vol. 36, 0146-4833.
13. **Karasaridis, A. and Rexroad, B. and Hoeflin, D.** Wide-scale Botnet Detection and Characterization. 2007. *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. pp. 7
14. Bot Knowledge. [Online] <http://www.botknowledge.com/>.
15. Blogbot. [Online] <http://blogbot.com/>.
16. **Holz, T.** A short visit to the bot zoo [malicious bots software]. : IEEE, 2005.
ISSN:1540-7993.
17. SearchBots.Net. [Online] <http://www.searchbots.net/about/>.
18. SpamBot Beware. [Online] <http://www.turnstep.com/Spambot/info.html>.
19. ITU Botnet Mitigation Toolkit project. [Online] <http://www.itu.int/ITU-D/cyb/cybersecurity/projects/botnet.html>.

20. Computer Weekly. Kaspersky reveals price list for botnet attacks. [Online]
<http://www.computerweekly.com/Articles/2009/07/23/237015/Kaspersky-reveals-price-list-for-botnet-attacks.htm>.
- 21 World Umbra data Dark Side Intelligence. [Online]
<http://www.umbradata.com/fc/world.html>.
22. **R. Puri**. Bots and botnets: an overview. Tech. Rep. SANS Institute, 2003.
23. **C. Mazzariello**. IRC traffic analysis for botnet detection. *Proceedings of the 4th International Symposium on Information Assurance and Security (IAS '08)*. pp. 318–323, Napoli, Italy. September 2008.
24. **B. McCarty**. Botnets: big and bigger. *IEEE Security and Privacy*, vol. 1, no. 4, pp. 87–90, 2003.
25. **R. Puri**. Bots and botnets: an overview. Tech. Rep., SANS Institute, 2003.
26. **J. Govil**. Examining the criminology of bot zoo. *Proceedings of the 6th International Conference on Information, Communications and Signal Processing (ICICS '07)*. pp. 1–6, Singapore, December 2007.
27. **G. Gu, R. Perdisci, J. Zhang, and W. Lee**. BotMiner: Clustering analysis of network traffic for protocol- and structureindependent botnet detection. *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

28. **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.
29. **G. Gu, J. Zhang, and W. Lee.** BotSniffer: Detecting botnet command and control channels in network traffic. *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.
30. **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.
31. **Herve Debar.** What is behavior-based intrusion detection. [Online]
http://www.sans.org/security-resources/idfaq/behavior_based.php.
32. **Garcia-Teodoro, P. and Diaz-Verdejo, J. and Macia-Fernandez, G. and Vazquez, E.** Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*. Vol 28. Pages 18—28, ISSN 0167-4048. 2009
33. **Mattord, verma.** Principles of Information Security. Course Technology. pp. 290–301. ISBN 9781423901778. 2008.
34. Blowfish Cipher - Cryptanalysis of Blowfish. [Online]
http://www.experiencefestival.com/blowfish_cipher_-_cryptanalysis_of_blowfish