6-8-2008

An Analysis of Botnet Vulnerabilities

Sean W. Hudson

AN ANALYSIS OF BOTNET VULNERABILITIES

THESIS

Sean W. Hudson, Captain, USAF

AFIT/GCE/ENG/07-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCE/ENG/07-05

AN ANALYSIS OF BOTNET VULNERABILITIES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Sean W. Hudson, BS

Captain, USAF

June 2007

AFIT/GCE/ENG/07-05

AN ANALYSIS OF BOTNET VULNERABILITIES

Sean W. Hudson, BS

Captain, USAF

Approved:

_______/signed/_____________________ 06/08/2007
Rusty O. Baldwin, Ph.D. (Chairman) Date

____/signed/________________________ 06/08/2007
Barry E. Mullins, Ph.D. (Member) Date

____/signed/__________________________ 06/08/2007
Robert F. Mills, Ph.D. (Member) Date

AFIT/GCE/ENG/07-05

Abstract

Botnets are a significant threat to computer networks and data stored on networked computers. The ability to inhibit communication between servers controlling the botnet and individual hosts would be an effective countermeasure. The objective of this research was to find vulnerabilities in Unreal IRCd that could be used to shut down the server. Analysis revealed that Unreal IRCd is a very mature and stable IRC server and no significant vulnerabilities were found. While this research does not eliminate the possibility that a critical vulnerability is present in the Unreal IRCd software, none were identified during this effort.

For my wife, for her caring and support throughout the course of this thesis effort

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Rusty O. Baldwin, for his unwavering support and encouragement throughout the course of this thesis effort.

Sean W. Hudson

Table of Contents

List of Figures

Page

List of Tables

Page

An Analysis of Botnet Vulnerabilities

I. Introduction

1.1 Background

Bots are software agents designed to automatically perform tasks. Examples include web-spiders that catalog the Internet and bots found in popular online games. Hackers use bots to infect and control large numbers of computers called a botnet [BSA07]. An expert recently called botnets the “No.1 emerging Internet threat [Sie07].” Botnets are responsible for 80% of e-mail spam and can be used for much more devious and destructive purposes [Sie07]. Botmasters are able to harness the computing power of thousands of geographically separated machines to launch distributed denial of service (DDOS) attacks against critical computer systems and to harvest data from unsuspecting victims.

1.2 Problem Statement

Currently, the Air Force and Department of Defense do not have an effective defense against a botnet attack. Current prevention techniques focus on vulnerability patching while commercial antivirus products and network sensors are used for detection. Furthermore, the logistics of vulnerability patching leave a window of opportunity for bots to gain a foothold on government networks while infected systems on non-government networks can be used to launch DDOS attacks and paralyze vital network assets. Remediation of infected computers on privately owned computers is impractical as a defense measure. Therefore, an attack on the botnet controller itself may be the most expedient means to thwart a botet attack.

1.3 Research Objectives

The objective of this research is to discover methods that can be used to counter the botnet threat without having to identify and clean infected computers. Botmasters are able to infect new computers at a much faster rate than administrators can remediate them, rendering cleaning methods ineffective. This research focuses on shutting down the command and control portion of botnets. Since bots must connect to an Internet Relay Chat server to receive instructions and updates or to transfer data mined from the host, a methodology that can interrupt communication between the controller and bots would be an effective countermeasure against attacks and could be used to prevent data exchange.

1.4 About this Document

This document is divided into five main chapters. This chapter provides an introduction to the research subject and a brief discussion of the problem. Chapter 2 provides background information and an overview of current research related to the subject area. Chapter 3 covers the methodology used for research and data collection. The results of the experiments and an analysis of the data is contained in Chapter 4 followed by conclusions drawn from the research and future research topics in Chapter 5.

II. Literature Review

2.1 Introduction

This section contains a brief overview of bots, internet relay chat, Unreal IRCd, dynamic domain name system, and current bot detection methods. It is followed by a detailed discussion of software testing and analysis as a means of vulnerability identification. Common code vulnerabilities and exploits are also discussed. The last section covers current research in the area of botnet countermeasures.

2.2 Bots

A bot is a script or executable file designed to perform certain functions automatically [Pur03]. Legitimate bots help maintain Internet Relay Chat (IRC) channels and catalog the web for search engines. Illegitimate bots, the focus of this research, are designed for malicious purposes. Bots are distinguished from other malicious code like viruses and worms by a communication channel linking them with a controller [CJM05]. Multiple bots directed by a single controller are called a botnet. Easy to use bot "kits" are available on the Internet which can be configured by users with limited programming skills [Pur03]. Furthermore, bots do not have a particular infection mechanism, but use the same methods as other malware. Typically, they take advantage of well-known unpatched vulnerabilities to spread [Hon05]. Virtually all bots contain mechanisms to scan networks for vulnerable hosts and infect new victims. After a successful exploitation, a bot uses a file transfer protocol to upload itself to the compromised host, starts itself, and tries to connect to its server [Hon05]. Bots are popular because their flexibility means they can be used for many different purposes. One of the most common malicious uses of botnets is to launch Distributed Denial of Service (DDOS) attacks. DDOS attacks are much more effective and difficult to counter because multiple hosts are involved. Bots may also use infected computers as proxies for spam or phishing emails. Some bots harvest email addresses from hosts. One of the advantages of bots is they can download updates or place other software such as keyloggers or sniffers onto infected systems. Another characteristic of bots is self-propagation. The most vulnerable and desired target hosts are lightly monitored, high bandwidth, home or university computers or servers [Pur03]. Home users moving from low bandwidth dial-up connections to higher bandwidth cable and DSL connections have dramatically increased the number of high value targets available to bot masters [Pur03].

2.3 Internet Relay Chat

Internet Relay Chat (IRC) is the protocol of choice for communication between a bot controller and its agents. “IRC provides a simple, low-latency, widely available, and anonymous command and control channel for botnet communication” [CJM05]. The default port for IRC is 6667 and IRC servers normally listen on ports 6000-7000. The IRC protocol (RFC 1459) was designed so users of bulletin boards could communicate during the early days of the Internet. The IRC protocol uses TCP for communication between clients and servers because of its reliability [OiR93]. Furthermore, the wide ranging popularity of IRC networks and services help attackers to obscure their activities and evade detection [Pur03]. IRC servers are freely available, easy to set up, and many attackers have years of IRC experience [Hon05]. IRChelp.org is a help site for users unfamiliar with IRC, and the following summarizes IRC operation [Van91]. Users connect to an IRC server via a client program and join a channel. A “channel” is similar to a chat room and divides the users into groups based on discussion topics. Messages on a channel may be designated public or private. IRC channel names begin with a pound sign (#) if they are shared among servers or an ampersand (&) if they are confined to a single server. The /Server irc.ais.net command connects to the server irc.ais.net. An important capability of IRC leveraged by bots is the ability to send and receive files. DCC Send and DCC Get transfer files between clients. Once a bot is installed on a victim machine, it automatically joins the channel specified by the bot master and waits for commands.

2.4 Unreal IRC

Most bot masters use either Unreal IRCd or Conference Room. [Hon05]. Unreal IRCd is a freely available open source IRC server that is available in Windows and Linux versions. Unreal IRCd is popular among bot masters because it is easily customized and very flexible. Bot masters may alter the server used by the bots in their network, and commands can be customized so that some are not supported or even follow the RFC specification. Common modifications include not using "JOIN", "PART" and "QUIT" messages to cut down on traffic and removing support for the commands "LUSERS" and "RPL_ISUPPORT" to limit information gathering attempts [Hon05]. Since Unreal IRCd is open source, the code can be examined for vulnerabilities that may be exploited to stop bot attacks.

Unreal IRCd is a robust IRC daemon that provides users with many features. It is compatible with Secure Sockets Layer (SSL) so communications can be encrypted. It includes defenses against Trojans, a spam filter, and flood protection. One of the most valuable features of Unreal IRCd is the ability to add modules. Around 50 modules are available to enhance the security and/or functionality of the server. Since the server is open source, third party modules are also available. One module is for cloaking. Cloaking allows the user hostnames of a client to be hidden from other users [Var06]. This required prevents users from flooding each other since they cannot see the real hosts/IPs [Var06].

A review of open source vulnerability lists reveals the types of vulnerabilities that have previously been discovered in Unreal IRCd. Secunia reported a vulnerability that would allow a denial of service due to improperly handling temporary kill line (TKL) commands [Sec06]. Kill lines (k-lines) are used by operators to ban users from a server. This list is checked before any connections are allowed. Unreal IRCd does not filter input from trusted, or "linked," servers which must be specified in the configuration file. This vulnerability, fixed in release 3.2.4, required that the server exploited be linked to the attacking server. While this vulnerability required the servers to be linked, which implies trust between the operators of the respective servers; this suggests that all command inputs should be tested to prevent similar attacks. Improper checking of user input is common software vulnerability. Securiteam reported a vulnerability similar to the TKL vulnerability with the JOIN command that could cause a server to crash, but again required the server to be linked to the attacker's server [Sky06]. In this same bulletin, the svsnick command did not check for illegal characters [Sky06]. These vulnerabilities emphasize the need for proper input checking and provide an early focus for testing and analysis. The other vulnerability widely reported was in the cloaking algorithm [Sec06]. A weakness in the algorithm could allow an attacker to discover other users IP addresses. This vulnerability poses a risk to users connecting to an Unreal IRCd server, but does not suggest possible ways to exploit the server.

2.5 Dynamic DNS

A bot master may use Dynamic Domain Name Service (DDNS) or multiple channels to maintain access to their botnet [Pur03]. DDNS allows users who do not have a static IP to maintain DNS records for their servers [Smi02]. A service provider maintains a database of current IP addresses and sends updates whenever the server address changes. Botmasters run these servers on their home computer or a victim computer to avoid public IRC servers. If they do use a public IRC server, they can update the DDNS records if they are banned from one server and change to another. Server operators ban any users or channels suspected of malicious bot activity. Since the DNS records are updated frequently, records are cached for a very short time. This forces all bot clients to perform DNS queries frequently which may be an avenue for detection [CWD05].

2.6 Detection Methods

Current bot detection and removal methods are not very effective. Anti-virus products are currently the best defense against bots, but are always playing catch up because they rely on signatures to identify malware. One of the difficulties is that bots are easily modified and once modified often evade detection by anti-virus signatures. Symantec documented 6,542 variants of Spybot during the last half of 2005 [Sym06]. Intrusion Detection Systems (IDS) often use common IRC commands for signatures to detect bot traffic on a network. Thus, networks that allow IRC traffic will produce a high false positive rate. Bots using non-standard ports have been documented [CJM05]. Encrypting command and control traffic or using nonstandard ports could hinder detection. “Attackers can make small modifications that make detection nearly impossible. For example, encrypting traffic, masking flow behavior with random noise, and even switching to different communication topologies can make detection immensely more challenging (e.g., versions of AgoBot with SSL encryption have been reported). In the end, any approach that relies on directly detecting command and control traffic can be defeated by changing the mode or behavior of the communication” [CJM05]. Since botnet detection is ineffective in preventing attacks, other means of countering the threat are needed.

2.7 Infiltration

To infiltrate a bot network, key pieces of information are needed. The IP address of the botnet server, or its IRC channel name and password [Hon05]. This information can be captured when the bot tries to connect to its master, or by analyzing the source code of the bot. With this information, an IRC client that is infiltrating a bot network can connect to the botnet. However, since the client will not respond to commands from the botmaster, it is possible the botmaster will detect this. Larger botnets increase the likelihood of remaining anonymous. Disabling auto response messages is one way to avoid detection [Hon05]. Once connected, a client can log all message traffic between the controller and clients and thus provide intelligence on the workings and intent of botnets. Still, this does not provide a means of stopping a threat.

2.8 Vulnerability Identification

Once a botnet has been detected and infiltrated, a method is needed to shut it down. Botnets depend on the controller to relay instructions and coordinate attacks, so shutting down the controller can prevent attacks. As part of this research, the source code for Unreal IRCd is examined to find vulnerabilities that can be used to stop attacks.

Static analysis tools are used to review source code. Any tools that examine the source code without running it are considered “static” analysis tools. Automated tools are available that provide a security audit on source code. Even so, manual methods are still needed to verify any vulnerability found by code checkers.

One of the most well-known and most exploited vulnerabilities is buffer overflows. A buffer overflow occurs when a program attempts to read or write data outside of the bounds allocated for the data. For Unreal IRCd all input vectors need to be verified to ensure that proper bounds checking occurs prior to memory access. Any calls using C “string” functions should also be checked to see if improper formatting can cause an overflow. Assuming that a string is null terminated when, in fact, it is not could result in an overflow. Also, using multi-byte width encoding formats could bypass rudimentary bounds checks and cause an overflow.

Format string vulnerabilities are a common issue in programs written in C. In C, strings are printed using a format string such as %d or %f that substitutes a variable value for the %d and %f. There are two problems with this. First, C does not check for missing variable arguments. If no arguments are passed to the printf function, it will pull values off the stack. This could allow an attacker to access stack data. The second problem occurs when a programmer does not check input data for correct formatting before using printf or another string function. This can lead to serious consequences. By including formatting strings in the input data, an attacker could direct printf to write code to memory and execute it which could allow an attacker to gain complete control of the vulnerable program [New00]. Unreal IRCd source code is checked for calls to string functions that use data supplied by the clients that an attacker could manipulate.

A problem similar to format string vulnerabilities is input validation vulnerabilities and code injection that takes advantage of these vulnerabilities. A program that receives input from untrusted sources may be vulnerable if the inputs are not thoroughly checked to ensure the inputs are formatted correctly, are of the correct length, and do not contain metacharacters that may be interpreted as code rather than data.

Resource exhaustion and memory management vulnerabilities are common in software. Common targets include memory and CPU resources. If legitimate users are unable to communicate with the server, a denial of service is successful. Can an attacker posing as a client send a flood of JOIN requests to the Unreal IRC server and prevent bot clients from connecting? If clients are using SSH to communicate with the server, will large encrypted requests overwhelm the server? Memory management holes may also be exploited. Can resources allocated for logging be overwhelmed by sending a flood of malformed packets? Will this prevent the administrator from identifying the real attack?

Trust relationships often provide an attack vector. Previously discovered vulnerabilities in Unreal were exploitable because of the trust relationship between linked servers. Can an attacker use Unreal server to link to the target Unreal server and take advantage of the trust relationship to crash or take over the server? Can a client take advantage of privileges gained by authenticating to the server to exploit it? Authentication data is hardcoded into bots, so login information may be obtained if needed to exploit a server.

2.9 Related Research

Current research on botnets is focused on improving techniques to detect them before they can be used for malicious purposes. In addition to focusing on detection methods, some researchers are using infiltration to monitor emerging techniques.

To better understand Botnets, their capabilities were studied using static analysis of the source code so defense strategies could be developed [BaY05]. The overall architecture of each bot was determined along with the commands used to control bots. All of the evaluated bots used IRC for command and control. Mechanisms used to propogate and alter a victim were also detailed. Some bots attempt to patch the system to defend against other malicious attacks, kill anti-virus software, or harvest information [BaY05].   Finally, the exploits and attack capabilities of each type of bot was documented. Commonalities among the bots could be used to develop defense strategies including the “predominant remote control mechanism for botnets IRC.” “Disruption of specific channels on IRC servers should continue to be an effective defensive strategy. [BaY05]” “The IRC-based command and control systems remain an area that the network security community can potentially exploit for defensive purposes. [BaY05].”

The analysis of IRC traffic using Cisco Netflow data is a method to detect botnets [Rac04]. One characteristic of botnets is a large population of inactive clients. A typical bot client connects to the server and awaits instructions from the controller. The server periodically sends a ping to each client and the client responds with a pong. Netflows from a botnet channel consist exclusively of ping/pong traffic until the controller sends a command. There were several technical hurdles encountered when trying to identify botnets through Netflow analysis. Since data is collected at gateway routers, the shear volume of traffic collected makes analysis difficult. Not all traffic flows were collected; in some cases 30% of the ping/pong traffic was not captured. The biggest concern was the high false positive rate. The proposed algorithm incorrectly classified non-pong traffic as pongs. Without refinement of the algorithm to more accurately identify traffic of inactive clients, this technique is not practical. The experiment tried to identify inactive clients, but did not attempt to group them by channel membership. To be useful, filters or methods would have to be developed based on channel membership to exclude traffic on channels that exhibit normal IRC characteristics and identifies channels that contain a preponderance of inactive clients. Netflow data has been used successfully to get an overview of malware activity and to identify large scale attacks in progress, but has not successfully been used to obtain information about emerging threats or detailed command and control data [Wic06].

One promising detection technique uses the Domain Name System (DNS). Many bots use Dynamic DNS (DDNS) which are usually third level domains. By examining DNS requests for third level domains, bots using DDNS can be identified [Dag05]. Current DNS detection methods use blacklists of canonical names associated with botnets [Kri05]. This method is very labor intensive and requires extensive storage on large networks. A comprehensive study of DNS analysis to determine its effectiveness for identifying botnets found the false positive rate to be high and the computations to be very CPU intensive [Sch06]. Combining DNS analysis with other monitoring devices was recommended.

Extending the capabilities of network intrusion devices (NIDS) to identify protocols used on non-standard ports has proved to be successful at identifying bot infected hosts [DFM06]. Traditional NIDS use port numbers to correlate network traffic to specific protocols and are often ineffective in identifying botnet related IRC traffic on non-standard ports or distinguishing regular IRC traffic from botnet traffic. Signatures to match the protocol regardless of port number were used to identify IRC traffic. A bot detector was added to the IRC protocol analyzer and used three different types of checks to flag bot traffic. It checks for “nicks” that match common botnet nickname patterns, looks for typical botnet commands, and flags connections to known bot servers. This successfully identified bot infected hosts on the networks being monitored. However, this approach relies on signatures of known bots and will not identify bots that do not conform to known signatures or whose command and control traffic is encrypted.

The goal of infiltrating a botnet is to analyze the command and control structure so it can be disabled [FrW05]. Tracking down every infected host of a large botnet is impossible, while removing the ability to control the clients is an achievable goal. This can be done by shutting down the control server or severing the ability of the controller to communicate with its clients. Botnets that rely on DDNS can be shut down if the DNS provider can be convinced to "blackhole" the DNS name. DNS queries for blackholed addresses are dropped, preventing bots from connecting to their controller. An alternative is to locate the IP address of the command and control server and get it removed from the network. This requires the cooperation of the Internet Service Provider or local law enforcement. Researchers have shown that they can infiltrate and track botnets [FrW05]. Unfortunately, both these methods of shutting down a botnet require third party cooperation. However, the ability to determine critical information about the entire botnet is much more effective than trying to locate individual infected hosts.

2.10 Summary

Botnets threaten critical infrastructures and understanding the techniques used by bot controllers to avoid detection is essential to defeating the threat. Continued analysis of emerging trends is needed to develop timely countermeasures.

III. Methodology

3.1 Problem Definition

Currently, the primary defense against botnets is prompt patching of vulnerable systems and antivirus software. Network monitoring can identify infected hosts attempting to propagate to other systems. However, identifying infected hosts is only the first step in stopping a botnet. To stop the botnet, either all the infected hosts must be located and cleaned, or the botnet controller must be identified and eliminated. Locating and cleaning all the infected hosts is impossible for a large botnet, since the infected hosts may be located worldwide. This leaves locating and eliminating the controller as the best option for limiting botnet effectiveness. Botnet researchers have had success eliminating botnets by working with Internet Service Providers (ISPs) and Dynamic Domain Name Service (DDNS) server operators to shut down controllers once they are found. While effective, this modus operandi takes considerable time and relies on cooperation between internet security professionals and commercial entities. Previous research has provided the tools needed to effectively locate botnet controllers. Still needed is a way to quickly shut down bot controllers before they can use their networks for malicious purposes. The primary control server used by botnet operators is an open source IRC Server called Unreal IRCd [Hon05]. The goal of this research is to find vulnerabilities in Unreal IRCd that can be used to disrupt the botnet controller's ability to communicate with its infected hosts. This research will determine whether vulnerabilities in Unreal IRCd can be exploited to sever communication and effectively kill a botnet. If a DDOS attack using a botnet can be thwarted by attacking the control server, this would provide an effective defense against a significant network warfare threat.

Static and dynamic analysis are used to examine the source code of Unreal IRCd. The code will be profiled to identify key components and document data flow. Software vulnerability scanning tools are used to evaluate the source code for potential vulnerabilities, which are evaluated to determine the effectiveness of attacks using the vulnerability to shut down the botnet server.

3.2 Static Analysis

Since Unreal IRCd is an open source project, the source code is readily available. The code is profiled and key components that perform critical functions are identified by adding debugging statements to an otherwise default Unreal IRCd server. This provides a target list of code functions and focuses the vulnerability search on areas of the software most likely to cause a critical fault. IRC servers only perform a few functions: adding and deleting clients, channel setup and maintenance, and message passing among clients. Operations that require operator privileges are not examined since that privilege level allows the server to be controlled without exploiting a vulnerability. A search of open source vulnerability databases is conducted to look for known vulnerabilities in Unreal IRCd. Known vulnerabilities in Unreal IRCd provide insight into the types of vulnerabilities that are likely to be found in other code segments. Buffer overflow vulnerabilities found and corrected in one module may be present and unpatched in other code sections. These initial steps help to focus the search on critical areas of the code and provide a starting point for performing additional tests on the software.

To facilitate code profiling, debugging statements added to each function of the source code provide a sequential trace of functions called when Unreal is started and when critical operations are invoked. One of the most important functions in Unreal is the read_message function. It checks all file descriptors (sockets) for incoming messages and new connections. For a new connection, the accept function is called. The accept function creates a socket and assigns a file descriptor to it. The add_connection function allocates memory for client information and adds the client to its client list. Once the client is added, its file descriptor is added to the read_set of file descriptors with messages in the queue to be read. Messages are only added to the queue if their length is less than 2048 bytes to protect against buffer overflows.

All operations in Unreal are use messages. Before a client can complete the connection process it sends several messages including authentication information if required and the client's nickname. All messages are handled by the same functions before being passed to specific modules for each message type. Read_packet is called to get the contents of messages from the read_set.   Read_packet enforces message rules based on client type. Servers are allowed to send messages without restriction, while regular users are limited. Read_message enforces a limit of 2048 bytes per message, and read_packet ensures users do not send a large number of small messages to flood the server. If a user's queues contain multiple messages that exceed the maximum limit (6090 bytes), the user receives an attempted flood message and is disconnected. Another buffer overflow mechanism found in read_packet checks for the presence of a carriage return or line feed. If read_packet processes 512 bytes without finding one, it adds one and flushes the rest of the buffer.

Once read_packet has transferred the message to the read buffer, do_packet collects statistical data and calls the parsing function for the message. The functions to add a client and process messages were examined carefully to look for buffer overflow opportunities. Since these functions process any data sent to the client it is critical that they are secure. Unreal uses multiple checks to ensure buffers are not overrun with data. In addition to the checks mentioned earlier, Unreal also limits the number of connections from a single IP address and the number of connection attempts allowed per unit of time is also limited.

After examining how Unreal handles incoming connections and monitors traffic for buffer overflow attempts and flooding, the routines to handle individual messages and commands was analyzed.   After read_message has placed the incoming message in a buffer and do_packet collects the statistics, parse is called. Parse receives as input pointers to the client structure that sent the message and to the beginning and end of the message. The first thing parse does is extract the command from the message. The command is the first word in every message. When entering text into an IRC client, users must use a slash (/) before entering a command into the chat window. If no slash is detected by the client, it prefixes the command PRIVMSG onto the text entered to indicate that it is a text message. The parse function looks for the first space in the message and takes everything prior to the space to be the command. The command is matched against a hash table that stores all allowed Unreal commands. If the command is not found an error message is sent and the buffer is cleared. If the command is found a pointer to the command structure is returned. The command structure indicates how many parameters the command takes and the type of user that can invoke the command. Some commands may only be invoked by an operator or a server. Parse checks to see if the sender is authorized to use the command and then divides the message into parameters with spaces again being the delimiters. If the total number of parameters is reached the rest of the message is stuck in the last parameter. The command structure also specifies the module that processes the command since each Unreal command is handled by a separate module.

The parse function and the modules that handle commands are examined to determine if there were format string vulnerabilities or input validation weaknesses that could be exploited. Unreal is not vulnerable to format string vulnerabilities because it does not do any processing on the strings that are supplied. Inserting escape characters or using hex encoding has no effect on the way the messages are handled. Input validation vulnerabilities and code injection attempts are also thwarted by the fact that Unreal does not do any pre-processing of the input. Attempts to inject code are interpreted as the text for a command or a command parameter.

After determining how Unreal handled attempts to insert format strings, code, and encoded input each module was examined to see how they handled missing parameters, encoded input, and extra data. Examples of common IRC commands include NICK, USER, JOIN, PASS, and PRIVMSG. The NICK command supplies the name to be displayed to others on an IRC server. The module that processes NICK commands ensures that exactly two parameters are supplied unless the connection is from a linked server specified in the configuration file. The length of the nickname is checked against a specified maximum and is rejected if it exceeds it. The supplied name is also checked against a list of disallowed (blacklist) and allowed (whitelist) characters. This is a very effective method for eliminating malicious input.

The USER command is used in conjunction with the NICK command to provide additional information about the connecting client. Again, the number of parameters is checked to ensure that the correct number has been supplied and if any are NULL. Usernames are limited to numbers (0-9) and characters from allowed language sets along with the dash and period. Any other characters including control characters cause an error and the user is disconnected. A special function checks the IP address supplied and verifies it with socket data.

The JOIN function specifies a channel or channels that a user wants to join. The function ensures that there are enough parameters and the channel name is not NULL. A pound (#) sign is required and an error is returned if it is not present. Optional parameters are also verified.

The PASS command is used if the server you are connecting to requires a password and only takes two parameters. The function checks the password length against a specified maximum and stores it to use later by a separate authorization function. If the password does not match the client is disconnected.

PRIVMSG sends text messages to a channel or individual clients. The function checks for enough parameters, both a message and a receiver must be specified. Hash tables are used to look up users and/or channels. No interpretation is done on the message, so any attempts to encode the message have no effect.

Every command that users may invoke was checked to ensure that it handled missing parameters or extra data and that control characters or command injection would not result in malicious code being run on the system. Operator commands were not verified since users with operator privileges are able to control the server.

Software vulnerability scanning tools identify more potential software vulnerabilities. Splint is a tool used to statically check C programs for security vulnerabilities and syntax errors. When used to evaluate the Unreal IRCd source code, Splint identified over 8,000 potential vulnerabilities. This high number is the result of Splint checking for coding errors as well as security vulnerabilities. Many of the items flagged by Splint might concern a programmer without opening a security hole. Examples of common errors that Splint caught included initializing a variable to NULL (885 errors) and using a non-boolean operand with a boolean operator (1455 errors).

While Splint caught many trivial code issues, it also caught some potentially serious ones. Splint identified 22 instances where unsafe functions were used. Many early C functions are vulnerable to buffer overflows if used improperly and have been replaced by safer versions. Splint identified instances in the Unreal code where the unsafe versions of functions such as Sprintf were being used. Sprintf is susceptible to buffer overflows if format specifiers are present or users are able to manipulate the input to the function. A review of the source code containing the vulnerable functions showed that care was taken to use them properly. Input to the functions came from the system not the user. Format strings were only used for data that had a known range of values. There was no opportunity to inject malicious values into the functions and cause the system to crash.

Another prevalent problem was suspected memory leaks. Splint flagged over one thousand potential memory leaks where “allocated storage was not released before return” or “incomplete deallocation of a structure or deep pointer is suspected.” Dynamic analysis is used to check for memory leaks during Unreal IRCd operation. Memory leaks could potentially lead to memory exhaustion and be used to shut down the server.

All Splint errors that identified potential security holes were evaluated manually to determine if they could be exploited to gain unauthorized access or crash the system. None of the potential vulnerabilities found could be used to crash the server or interrupt communication.

3.3 Dynamic Analysis

The dynamic portion of the analysis is checks for memory leaks and attempts to exhaust the system memory, causing Unreal IRCd to stop. To test memory performance, an automated method of rapidly connecting to the Unreal server using unique Internet addresses is required unless a large botnet can be used to initiate connections. Unreal limits the total number of connections per address and the number of connection attempts per address per unit of time. Hping is a network security testing tool that can assemble and capture packets. Using Hping, source addresses can be spoofed and hundreds of client connections to Unreal can be initiated without having access to hundreds of computer systems. During testing, Hping generates large numbers of SYN packets, initiating three way handshakes with the target server. Responses from the server are captured and used to generate the proper sequence of packets to create a full client connection to Unreal. Each connection, in addition to having different addresses, must also provide a unique nickname and user data. After a connection has been established, it may be terminated or left open depending on the test that is being conducted. Hping's scripting capability provides the ability to build and send packets based on data extracted from packets received.

3.4 System Boundaries

The system under test is the suite of tools and techniques used to discover potential vulnerabilities in the Unreal IRCd software (Figure 1). This system is called the Unreal IRCd Evaluation System. Critical code segments identified during code profiling are examined via manual and automated methods for potential avenues of exploitation. Software vulnerability scanning is used to identify other possible code vulnerabilities.

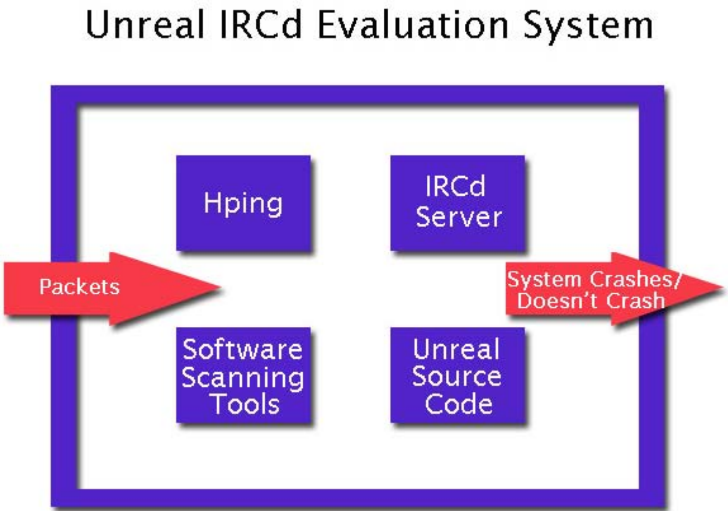

Figure 1: System Under Test

Potential vulnerabilities are examined and exploits to target the vulnerability are developed if the vulnerability appears to be exploitable. The component under test is Unreal IRCd. Potential exploits and vulnerabilities are tested by generating packets that target the vulnerability and sending them to the Unreal IRCd server.

3.5 System Services

The Unreal IRCd Evaluation System identifies potential vulnerabilities in Unreal IRCd and tests them using a packet generator. The system provides multiple ways of evaluating Unreal IRCd when attempting to find vulnerabilities. Code profiling identifies critical portions of the software that are evaluated using manual methods and software scanning tools to find coding errors and security vulnerabilities.

3.6 Workload

Packets generated using Hping comprise the workload for the system. Hping sends crafted packets identical to packets generated by a normal IRC chat client. Hping's packet capturing function is used to analyze response packets from the Unreal IRCd server. The only modification to the Unreal IRCd server was to create a static address resolution protocol (ARP) table, so all server responses would be sent to the attack system running Hping. Using spoofed source addresses, one computer is used to generate connections that would normally require thousands of clients.

3.7 Performance Metrics

The only performance metric tracked is memory utilization. Static analysis of the source code indicated that memory management might be a significant vulnerability in

Unreal IRCd. Using dynamic analysis, memory utilization is measured using operating system queries while connection attempts are sent to Unreal IRCd.

3.8 Parameters

Parameters of this experiment include the hardware used and operating system (OS) choice. An early decision was made to use Linux as the operating system. A precompiled version of Unreal IRCd for Windows is available, but using the Linux version provided access to the source code. Altering the source code to insert debugging statements aided code profiling. An unaltered version is used for dynamic analysis experiments. Linux limits the number of open files per process to 1024. This affected any tests requiring more than 1024 connections. To increase the number of files allowed, a kernel variable must be changed and the kernel recompiled which is unlikely in fielded versions of Unreal IRCd on Linux.

The attack system and the Unreal IRCd server are standalone computers on a private network. Only traffic generated by the two systems is on the 100Mbps network. The hardware specifications for the Unreal IRCd server are provided in Table 1.

Table 1: Unreal IRCd Hardware Specifications

CPU	Intel Pentium 4 (3 GHz, 800MHz FSB)
Memory	512 MB SDRAM
OS	Fedora Core 6

3.9 Factors

There are two factors used in the experiments run during dynamic analysis, the number of connections and the connection type. Two different types of connections were used. For one set of experiments, clients connected and immediately closed the connection. This experiment checks for the potential memory leaks indicated by the static analysis. The other type of connection was persistent. Unreal IRCd was bombarded with client connections which were not terminated. This set of experiments was designed to check memory utilization. The factors chosen and the levels for the experiments are summarized in Table 2.

Table 2: Factors and their levels

Factor	Levels
Connections	1,000/2,500/5,000/10,000 10,000/20,000/30,000/40,000/50,000 10,000/25,000/50,000/75,000/100,000
Connection Type	Transient/ Persistent

3.10 Evaluation Technique

Performance is evaluated by direct measurement. More accurate results can be obtained using actual servers rather than a simulator or analytic model. The attack system generates connection attempts and responds appropriately to Unreal IRCd server responses. Wireshark, a network sniffer, monitors communication between the clients and server to validate that the attack system and server are interacting properly. At prescribed intervals based on the number of connection attempts, the operating system of the Unreal IRCd server is polled using a “ps aux” command to obtain memory utilization statistics. These statistics are recorded for later analysis.

3.11 Experimentation Design

Memory utilization for each type of connection is measured for each connection level. If memory utilization continues to increase without bound, additional experiments are conducted until utilization peaks, the server crashes, or stops responding to client requests. The experiments are replicated three times to account for any variation due to network and system anomalies. Since the experiments are conducted in a controlled lab environment, variance is expected to be very low.

3.12 Summary

Static analysis of Unreal IRCd is conducted using a variety of tools to evaluate vulnerabilities. Dynamic analysis is used to evaluate the significance of the vulnerabilities and the feasibility of exploiting them to interrupt communication between a botnet and the control server.

IV. Analysis and Results

4.1 Results and Analysis of Individual Measures

The first series of experiments check for memory leaks in Unreal IRCd. During static analysis using Splint, memory management issues triggered a high number of alerts. Using Hping, connections were established to the server, a channel was joined, and then the client closes the connection. If memory leaks are present, the memory footprint of Unreal should increase as the total number of connections increases. Unreal IRCd was restarted before every run of the experiments. Before any connections were made, Unreal IRCd used either 3800 Kilobytes (K) or 3804K of memory. This indicates that there is a memory leak issue since 4K of memory is not always released. The first factor level tested was 10,000 connections, and the results of this experiment are shown in Figure 2.

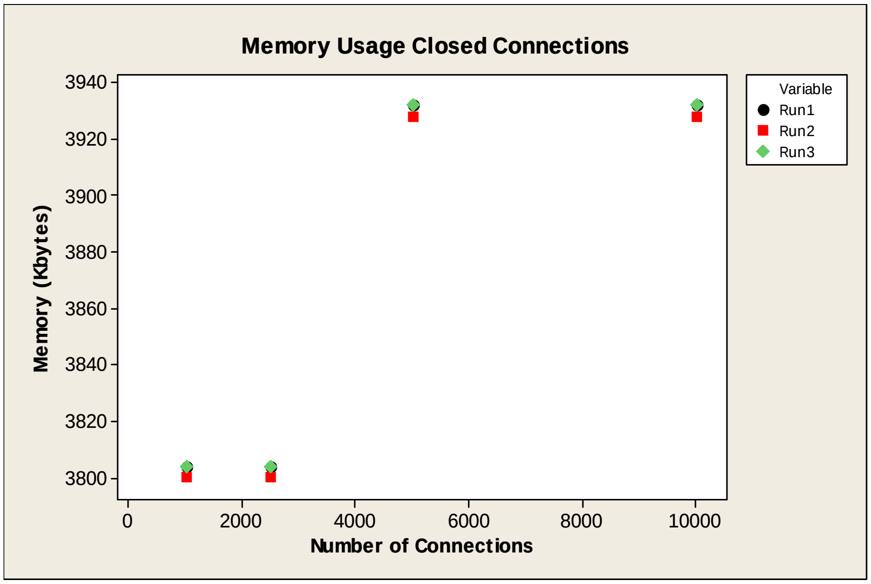

Figure 2: Memory Utilization with 10,000 Closed Connections

At 2,500 connections and under, there is no increase in memory footprint. Between 2,500 and 5,000 memory usage increases 28K. The only difference seen in the memory usage between the three runs was the initial 4K difference present after restart. The 28K increase in memory utilization should not occur and indicates that there is a memory issue with Unreal IRCd. Since every connection is closed, the structures used to hold client information should be deleted and the memory freed. If Unreal IRCd continues to use memory without freeing it all, it will eventually consume all the system memory resources and cause the system to shut down or lock up. For the next experiment the connections were increased to 50,000 to see if additional memory would be allocated. The results of this experiment are shown in Figure 3.

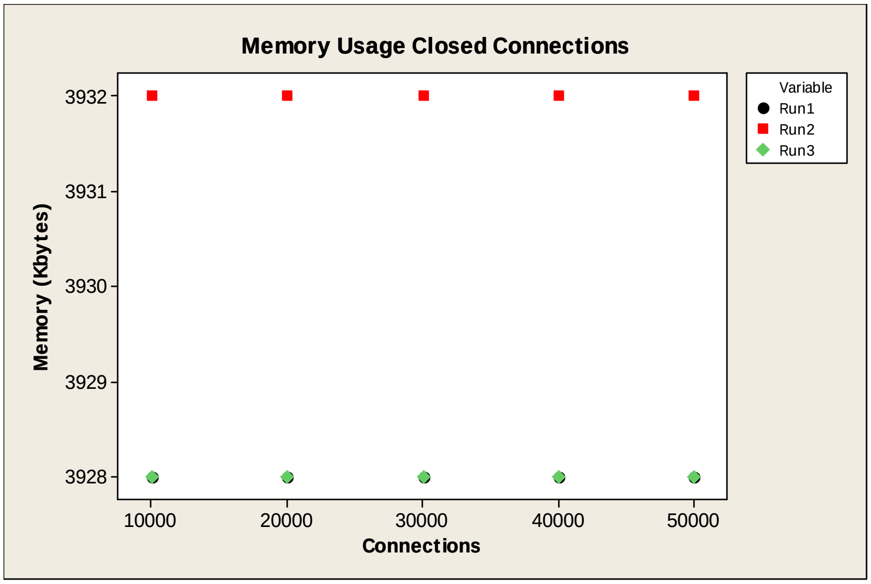

Figure 3: Memory Utilization with 50,000 Closed Connections

As seen in the graph, memory usage did not increase. The amount of memory used between 10,000 and 50,000 connections remained the same. The next series of runs increased the connections to 100,000 with the results presented in Figure 4.

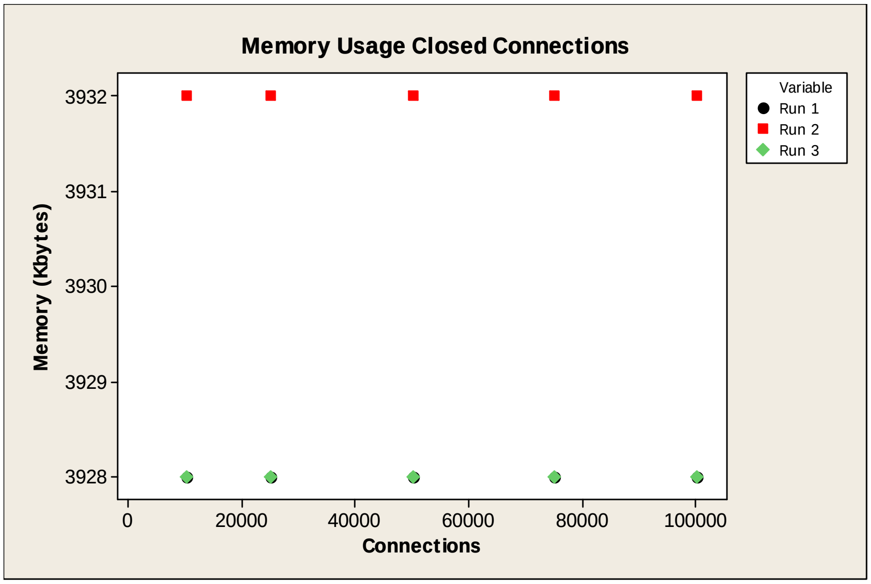

Figure 4: Memory Utilization with 100,000 Closed Connections

Again, there was no additional increase in memory footprint. One additional experiment was run with 500,000 connections to verify that memory usage remained at this level, and the results did not indicate any increased usage at that level either.

The first group of experiments shows that there were minor memory management issues with Unreal IRCd. When Unreal IRCd is stopped, some memory may not be released. This amount of memory is always 4K, so it is not a significant amount. Additionally, Unreal IRCd uses an additional 28K after approximately 2,500 connections. This is additional memory that should be freed, however, since Unreal IRCd never consumes additional resources it is not a serious issue.

The next series of experiments test another aspect of memory management in Unreal IRCd. For these experiments, connections are established with the Unreal IRCd server, but were not closed. These experiments should result in a larger memory footprint for Unreal IRCd since memory will be allocated to store client data and for the queues used to receive data from the clients. The first experiment involved 10,000 connection attempts with the results recorded in Figure 5.

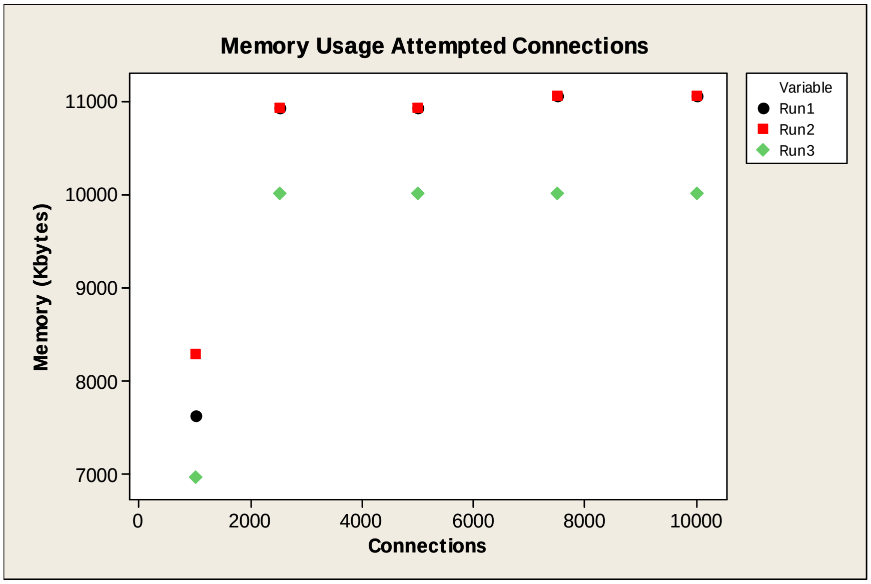

Figure 5: Memory Utilization with 10,000 Attempted Connections

As shown in the graph, memory usage has peaked at 2,500 connections and remains the same through 10,000 connections. An analysis of the network traffic captures provided the explanation. After 1,023 connections Unreal IRCd stopped allowing new connections until previous connections timed out and were closed. The Unreal IRCd configuration allows the total number of connections to be specified by the administrator, but Linux only allowed 1,024 open files per process. Unreal IRCd continues to accept connection requests, but instead of allocating memory it replies with an error message.

There is a large increase in memory usage between 1,000 and 2,500 connection attempts that cannot be attributed to the 24 additional connections allowed after the data was gathered at 1,000 connections. After the maximum number of connections is reached, Unreal IRCd is only handling connections that it eventually denies. Additional memory usage must be caused by overhead associated with replying to new connection attempts. Two additional experiments were conducted with 50,000 and 100,000 attempts to see if Unreal IRCd could be overwhelmed with new connection attempts. These results are presented in Figures 6 and 7.

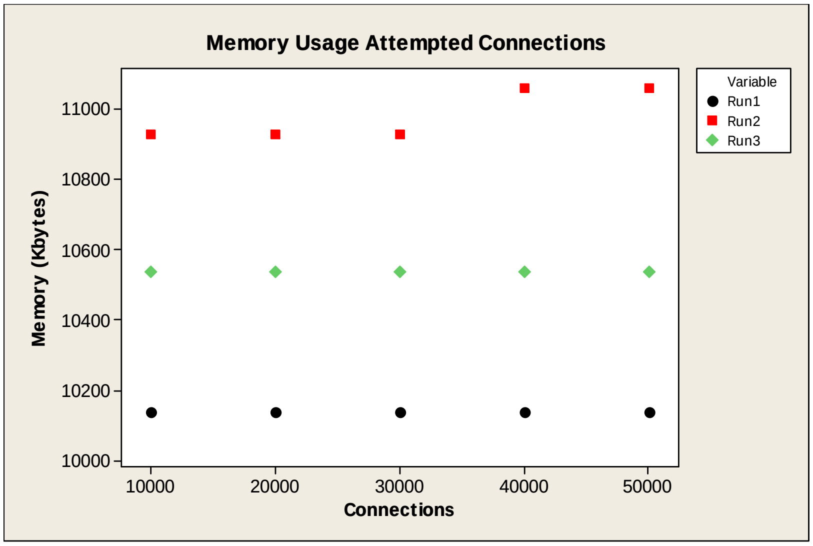

Figure 6: Memory Utilization with 50,000 Attempted Connections

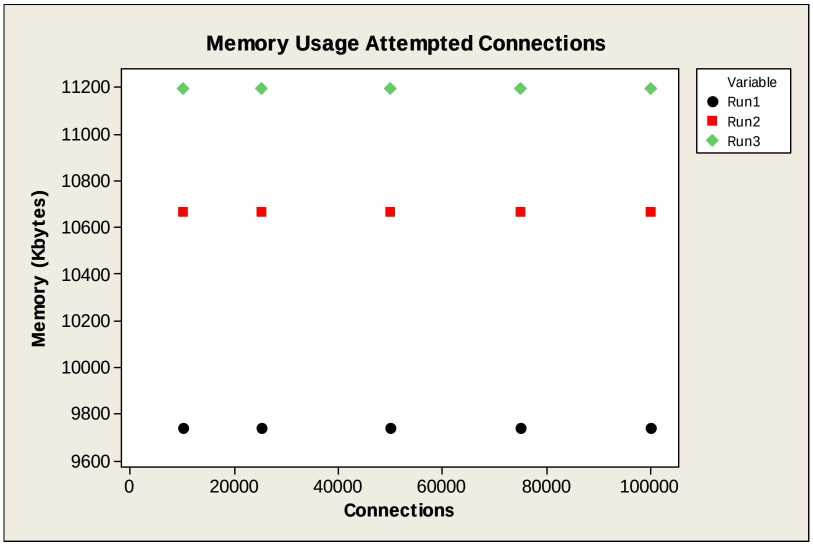

Figure 7: Memory Utilization with 100,000 Attempted Connections

As seen in Figure 6 and 7, memory usage does not increase no matter how many new connection attempts are sent to Unreal IRCd.

This series of experiments demonstrates that an attack designed to exhaust the server's memory is unlikely to be effective unless the server administrator alters the kernel to allow a larger number of connections than the operating system can handle, or use an operating system that does not limit open files. The number of allowed connections would have to be large enough to allow the server to exhaust all available resources.

4.2 Summary

The experiments in this section test potential memory management vulnerabilities identified during static analysis by Splint. While the experiments show that there were minor issues with memory management by Unreal IRCd, the vulnerability is not significant enough interrupt communication between the server and its clients.

V. Conclusions and Recommendations

5.1 Overview and Conclusions

The source code and operation of Unreal IRCd was studied using static and dynamic analysis to identify vulnerabilities. A critical vulnerability in Unreal IRCd could be used to shut down a server directing an attack, providing an effective defense against a growing botnet threat. Analysis revealed that Unreal IRCd is a very mature and stable IRC server. The lack of documented vulnerabilities in the software is indicative of the obvious effort to provide a secure and robust platform. Comments in the source code indicate that security was an important consideration of the programmers. While this research does not eliminate the possibility that a critical vulnerability is present in the Unreal IRCd software, none were identified during this effort.

5.2 Recommendations for Future Research

Analysis of the Unreal IRCd server did not identify a critical vulnerability that could be exploited to shut down the controller. However, the possibility exists to hamper communication through DDOS attacks. DDOS attacks are one of the most devastating uses of botnets, but also might be used to interrupt or stop an attack. Can a methodology be developed to attack the control server with bogus connections or IRC requests that prevent the server from issuing commands or receiving data from its clients? All IRCd servers have limits on the number of bots they can support. It may be feasible to clone harmless bots that emulate the botmaster's clients and reduce the effectiveness of an attack. Bots might also be engineered to provide false or misleading data to their controllers as part of a counterintelligence measure.

Botnets are a significant and growing threat to our networks and data. Relying on software vendors to develop patches to fix vulnerabilities will never eliminate the threat, software developers and anti-virus vendors are always playing catch-up with hackers. Developing effective botnet countermeasures is critical to protect our networked assets and ensure their availability.

Bibliography

[BaY05] Barford, Paul and Vinod Yegneswaran, “An Inside Look at Botnets”, 2005. http://www.cs.wisc.edu/~pb/botnets_final.pdf, Accessed: Oct 02, 2006.

[BSA07] Business Software Alliance “Cyber Safety Glossary,” 2007. http://www.bsacybersafety.com/threat/bots.cfm, Accessed: May 30, 2007.

[CJM05] Cooke, Evan, Farnam Jahanian, and Danny McPherson, "The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets," Electrical Engineering and Computer Science Department, University of Michigan, 2005.

[CWD05] Cliff, C. Zou, Gong Weibo, Don Towsley, and Gao Lixin, "The monitoring and early detection of Internet worms," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 961-974, 2005.

[Dag06] Dagon, David, "Botnet Detection and Response: The Network is the Infection," 2005. http://www.caida.org/funding/oarc/200507/slides/oarc0507-Dagon.pdf, Accessed: May 21, 2006.

[DFM06] Dreger, Holger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer, "Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection," 2006, http://www.icir.org/robin/papers/usenix06/, Accessed: May 21, 2006.

[FHW05] Freiling, Felix C., Thorsten Holz, and Georg Wicherski, *Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks*, 2005.

[Hon05] Honeynet Project "Tracking Botnets," 2005, http://www.honeynet.org/papers/bots/ Accessed: May 7, 2006.

[Kri05] Kristoff, John, "Botnets, Detection and Mitigation: DNS Based Techniques," 2005, www.it.northwestern.edu/ bin/docs/bots_kristoff_jul05.ppt, Accessed: May 13, 2006.

[OiR93] Oikarinen, Jarkko and Darren Reed, "RFC 1459 Internet Relay Chat," 1993, http://rfc.net/rfc1459.html, Accessed: May 7, 2006.

[New00] Newsham, Tim, “Format String Attacks,” 2000, http://muse.linuxmafia.org/lost+found/format-string-attacks.pdf, Accessed: September 19, 2006.

[Pro04] Provos, Niels, "Honeypot Background," 2004, http://www.honeyd.org/background.php, Accessed: May 21, 2006.

[Pur03] Puri, Ramneek, "Bots & Botnet: An Overview," 2003, http://www.sans.org/rr/whitepapers/malicious/1299.php, Accessed: April 8, 2006.

[Rac04] Racine, Stephane, "Analysis of Internet Relay Chat Usage by DDOS Zombies," Swiss Federal Institute of Technology, 2004, MA-2004-01.

[Sch06] Schonewille, Antoine and Dirk-Jan van Helmond, "The Domain Name Service as an IDS," University of Amsterdam, 2006.

[Sec06] Secunia, Secunia Vulnerability Report: UnrealIRCd 3.x, http://secunia.com/product/3663/?task=advisories, Accessed: September 14, 2006.

[Sie07] Sieberg, Daniel "Expert: Botnets No. 1 Emerging Internet Threat," 2007 http://www.cnn.com/2006/TECH/internet/01/31/furst/, Accessed:May 23, 2007.

[Sky06] Skyrim msh, "Serious Flaw in Unreal IRCd (Server Linking, Svsnick)" http://www.securiteam.com/unixfocus/5GP0E2A7PC.html, Accessed: September 14, 2006.

[Smi02] Smith, David, "Dynamic DNS," 2002, http://www.technopagan.org/dynamic/, Accessed: May 21, 2006.

[Sym06] Symantec, "Symantec Internet Security Threat Report," 2006, https://enterprise.symantec.com/enterprise/whitepaper.cfm, Accessed: May 21, 2006.

[Van91] Van Loon, Ronald, "An IRC Tutorial," 1991, http://www.irchelp.org/irchelp/irctutorial.html, Accessed: May 7, 2006.

[Var06] Various, "UnrealIRCd-3.2-Official Documentation,"2006, http://www.vulnscan.org/UnrealIRCd/unreal32docs.html, Accessed: Sep 13, 2006.

[Wic06] Wicherski, Georg, "Medium Interaction Honeypots," 2006, www.pixel-house.net, Accessed: May 21, 2006.

Vita

Sean Hudson was born in Asheville, OH into a military family and after several moves graduated from Jacksonville High School, Jacksonville, AR in 1987. He enlisted in the Air Force in 1990 and served 11 years as an enlisted member achieving the rank of SSgt. In 2002, Capt Hudson was selected to attend Officer Training School at Maxwell AFB where he earned his commission. Capt Hudson's first assignment as an officer was monitoring the Air Force's computer networks at the 33rd Information Operations Squadron. Capt Hudson was selected to attend the Air Force Institute of Technology (AFIT) in August 2003 and will be assigned to Los Angeles AFB, CA working on the Transformational Satellite Communications System (TSAT) Program after graduation.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 14-06-2007	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* March 2006 – June 2007

4. TITLE AND SUBTITLE **AN ANALYSIS OF BOTNET VULNERABILITIES**	**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Hudson, Sean W., Captain, USAF	**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/07-05
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)** **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Botnets are a significant threat to computer networks and data stored on networked computers. The ability to inhibit communication between servers controlling the botnet and individual hosts would be an effective countermeasure. The objective of this research was to find vulnerabilities in Unreal IRCd that could be used to shut down the server. Analysis revealed that Unreal IRCd is a very mature and stable IRC server and no significant vulnerabilities were found. While this research does not eliminate the possibility that a critical vulnerability is present in the Unreal IRCd software, none were identified during this effort.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Dr. Rusty O. Baldwin
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	50	937-255-3636 x 4445; rusty.baldwin@afit.edu (emailname@afit.edu)