STICKY WORMS OR:

RESPONDING TO A DISTRIBUTED DENIAL OF SERVICE ATTACK WITH A SELECTIVE TARPIT

JORDAN K. WIENS

A SANS GCIA CERTIFIED INCIDENT HANDLER
PRACTICAL ASSIGNMENT v4.0
SUBMITTED SEPTEMBER 20, 2004

Acknowledgements

Many thanks to my wife who not only challenges me and loves me, but more specifically is probably the only reason I even finished this paper. I will always admire your diligence and intelligence. I love you more each day, and don't mind sounding sappy in public professing it.

Kathy, you and the rest of the CNS staff have made my work experience these last four years an ethical hacker's dream. I tell anyone who asks that you do all the things a boss should do, and none of the things they shouldn't. Well, none besides the occasional overzealous edit. (I wonder if that will survive the edits)

Thanks to Chuck Logan for all your work on Sticky. This whole concept probably could not have been made at all, and certainly could not have been adapted as quickly without your effort.

Thanks to Tom Liston for LaBrea as well as your willingness to help throughout the process, and Joe Stewart for the idea of using the tarpit to counteract the DDOS, as well as the invaluable analysis on this and many other pieces of malware.

Thanks to the SANS Malware list which has been a great resource for advice, techniques, and samples.

Abstract

Some variants of the Netsky worm targeted a distributed denial of service (DDOS) attack at a web server on the University of Florida (UF) network. The University of Florida Incident Response Team (UFIRT) was able to prepare a unique defensive measure against the attack that may prove useful in defending other networks from similar attacks. This paper will detail the Netsky worm in general, the specifics of the three variants that attacked the University of Florida, and will explain how a selective tarpit could drastically reduce the impact of some denial of service attacks.

TABLE OF CONTENTS

Introduction

The University of Florida (UF) Incident Response Team (UFIRT) is familiar with the impact of viruses and worms on a large network. The Netsky worm however, impacted UF in a strikingly different way. For a still unknown reason (see Virus Wars for speculation), Netsky.x, Netsky.y, and Netsky.z performed a distributed denial of service (DDOS) attack on three educational web servers. One of them was www.medinfo.ufl.edu, a web server maintained by the College of Medicine at UF.

Fortunately, the DDOS attack was programmed to begin a week after the variants were released, giving the worm time to amass a larger set of infected hosts, but also giving UFIRT more time to prepare.

During the preparation stage of incident handling, UFIRT queried various mailing lists for more information. On one such list, Joe Stewart from LURHQ proposed an intriguing defensive mechanism. He suggested a selective tarpit (or stickypit, as UFIRT named it) to trap the DDOS attacks of the worms. This would not only help protect the UF targeted server by decreasing the frequency of attacks, but would also slow down the rate of attack against other servers that were simultaneously targeted.

During the attack problems were discovered with the stickypit design. However, these problems only impacted a small population of legitimate visitors to the www.medinfo.ufl.edu web site, and only for short periods of time.

There are many other potential uses for a stickypit and hopefully the concept will be adapted and utilized by others to protect their networks as well.

Note: While most GCIH assignments are sanitized to hide the organizations involved in the incident, it is already public knowledge that the Netsky worm attacked a server in the University network space. Thus, this paper will attempt to discuss already public information in detail; however, some internal details (such as specific incident response procedures and network topology details) must be obfuscated or omitted since the organization cannot help but be identified.

The Worm

Terminology

There are some distinctions in terminology that are important to clarify. Virus is usually defined to mean a piece of code that infects other programs or files and requires the user to transmit the infected content to other machines where it repeats the process. A worm is different from a virus in that it actively spreads itself. This may be by sending email, attempting exploits, or using other methods.

The terms worm and virus are often conflated. For example, Netsky is typically referred to as a virus since most variants require some user action. It is categorized as a virus despite the fact that it emails itself instead of infecting an existing file as traditional viruses do. For that reason, while worm is usually used in this paper, virus and worm are often used interchangeably. Categorization is more difficult because for many years now such programs have not been easily classified as either a virus or a worm. For example, the Nimda virus exhibits behaviors of both a virus and a worm. It use mass mailing techniques more commonly ascribed to viruses and IIS exploits more commonly ascribed to worms. Most recent worms and viruses also include backdoors which are another example of malicious program functionality converging. The generic term malware is used to describe the entire category of malicious software such as worms, viruses, adware, spyware, trojans, backdoors, bots (short for robots), and others.

History

The original Netsky worm was discovered on February 16th, 2004, and was considered by most Anti Virus (AV) vendors to be a low priority threat. It did not exhibit any particularly new or powerful features to help it spread. The primary distribution method was via email, and the initial infection count was estimated to be very small. The second distribution method was to copy itself to various directories and mapped drives using a variety of appealing names. It could then be executed when discovered on both Peer-to-Peer (P2P) networks and over file shares.

Multiple variants followed in rapid succession. From February 16th until at least May 2nd, more than 20 different variants were released; averaging a release every three or four days.

The distribution method using file shares and P2P networks was apparently not as successful as the author would have liked and this propagation method was dropped in the D variant, although it reappeared in later variants. Other features appeared and disappeared randomly. For example, some variants played a random series of tones on the speaker of the infected machine during certain dates. There seemed to be a general amount of experimenting as features were tested and discarded.

Messages were transmitted in strings embedded in the worms to the AV community, as well as to the authors of other worms with whom the Netsky author was feuding. The initial message claimed authorship under the name “Team Skynet” (likely a reference to the Terminator series of movies in which a computer network called Skynet supposedly takes over the world and engages in a battle with mankind).

There are definite trends and styles to the variants. There appear to be multiple strains that are likely built upon the same source code, yet are fairly different in terms of features, packing methods, types of comments, etc. In fact, in one variant, strings embedded in the worm thank the original Netsky team for the source code and announce that a new group called NetDy would create new variants. Additionally, while the original Netsky strain was quite disparaging in its comments about the Bagle worm for having a backdoor, later variants of Netsky would have backdoors of their own.

The Q variant was the first to exhibit denial of service capabilities. A variety of P2P websites were attacked, and though specific targets changed, the P2P trend continued until the X, Y, and Z variants that attacked educational web servers. Shortly after these three variants were released, the Sasser worm was discovered, and a later Netsky variant contained a message that claimed credit for the Sasser worm as well. Expert analysis of the source code [1,2] supports the claim.

Over the next few weeks, additional Sasser and Netsky variants were released, however, they dwindled off after the arrest by German authorities of a man they believed to be the author [3,4]. Whether he was the author of all Netsky and Sasser variants, or whether other writers simply moved on after his arrest is unknown.

Variants

Table 1: Variants includes highlights of each of the Netsky variants, as well as links to references for each variant in major anti-virus databases. The following vendor databases were used as references: McAfee, Symantec, Trend Micro, Panda Software, and F-Secure. Due to nomenclature differences, the Kaspersky virus database was not used as a reference, despite their technical and concise analyses. At some point, the variant names of most other AV vendors and Kaspersky's became out of sync, and for that reason only the above databases were used.

Virus Wars

There are a number of common trends and attitudes in the world of malware writers, as well as some recent differences. For many years, the sole motivation (that most would admit) for a virus writer was the pure challenge of technical learning and pushing the limits of systems. These idealists often claimed to be espousing the ideal aspects of hacking [5], despite the obvious negative impact malware has when released.

In recent years, however, there has been a growing trend of malware for profit. From adware that is targeted at infecting user machines and forcing them to view advertisements, to backdoors that monitor and steal credit card information and

other sensitive data, malware has very serious financial considerations that go beyond the well known impact of cleanup alone.

There have even been suggestions that many malware for profit operations are being funded or run by the Russian mafia [6,7], and other organized crime syndicates [8]. The commercialization of malware writing has resulted in a schism between the newer profiteers and idealists who disdain those with less 'pure' motivations.

The Bagle and Mydoom worms are believed to be used in Spam rings and other profiteering groups. Backdoors built into these worms are used to control infected hosts for profit.

The Netsky worm appears to have originated from one of the idealistic malware writers. Not only are vitriolic comments made back and forth between the Netsky author and the authors of Bagle and Mydoom, but many Netsky variants actively attack other worms installed on infected hosts. Typically, this is done by removing known registry keys used to start the rival worm so that, after the next reboot, only the Netsky worm will run on the infected host.

It's from that perspective that many of the comments in the Netsky program code should be taken into context. There are references to Skynet AV as if it was a legitimate anti-virus product, and the Skynet team apparently feels as if they are doing less harm than authors of other worms.

Despite the supposedly wholesome intentions, Netsky causes as many problems as it fixes. A Netsky infection is still an infection, and must be handled accordingly.

While the motivation for Netsky to target www.medinfo.ufl.edu is still unknown, UFIRT speculated that it might have been as a result of this feud. If the medinfo server was involved (or perceived to be involved) somehow, it might have been targeted for that reason. This was taken into account during the incident handling process.

Operating Systems

Since Netsky does not exploit any technical flaws besides user willingness to open attachments that shouldn't be opened, there are no specific Windows operating systems which are more or less vulnerable to Netsky. The worm was written in Visual C++ and compiled for Windows operating systems and is not portable to other operating systems without some other emulation layer. Any Windows operating system from Windows 95 on is therefore potentially vulnerable.

It should be noted that the Windows XP Service Pack 2 [9] has some mitigating effects on Netsky. None of the memory protections inhibit infection since Netsky

is essentially a normal user program not taking advantage of any exploits or buffer overflows. Netsky does, however, generate large amounts of network activity in a short period of time. Service Pack 2 includes limits to the rate at which network connections can be opened [10]. Thus, while a host running Service Pack 2 could become infected, it would neither spread as rapidly, nor attack as effectively.

Protocols

SMTP

As an email virus, Netsky makes heavy use of SMTP not only by using its own built in SMTP client to send email directly to mail servers, but also by taking advantage of the MIME encoding extensions to encode the program as an attachment in a portable fashion.

SMTP servers function by listening on TCP port 25 for connections from clients to receive mail and are based on RFC821 [11]. A basic SMTP transaction consists of a few simple commands. In fact, mail can be sent using SMTP with nothing more than a telnet session since the protocol is human readable. A short SMTP transaction would occur on port 25/TCP and would appear as follows (server responses in orange, 'client' in blue – the SMTP client may be another server):

```
220 server-name SMTP Mail-server-program; Date
HELO client-domain-name
250-server-name Hello client-domain-name [IP]
MAIL FROM:<from@client-domain-name>
250 2.1.0 <from@client-domain-name>… Sender ok
RCPT TO:<destination@server-name>…
250 2.1.6 <destination@server-name>… Recipient ok
DATA
354 Enter mail, end with "." On a line by itself
From: from@client-domain-name
To: destination@server-name
Subject: Hi

Testing SMTP!
.
250 2.0.0 msg-id Message accepted for delivery
QUIT
221 2.0.0 server-name closing connection
```

Additionally, there are extensions to the SMTP protocol that allow for a slightly different transaction style. These are derived from RFC2821 [12], and include the EHLO syntax used in the Netsky sample communication documented later.

The MIME specification was created to allow the transport of binary files within the SMTP protocol which only allows for printable ASCII characters. The MIME format is used by Netsky to encode a copy of itself in text that can be sent via SMTP and decoded into a binary on the other end. More information is available on MIME in RFC2045 [13], and some ancillary specifications are included in RFC2112 [2112].

HTTP

HTTP is the transport protocol of the World Wide Web and is built on TCP. Interestingly, as firewalls and other security devices restrict and lock down previously open ports, more and more applications are moving inside of HTTP. This means that more exploits are occurring higher in the protocol stack and that security devices must learn another set of protocols built on top of HTTP to detect attacks. This can be seen in the emergence of new protocols using HTTP tunneling such as SOAP. Netsky targets web servers with a flood of HTTP requests to affect a DDOS. Fortunately, Netsky doesn't use anything as complex as SOAP for the DDOS. In fact, Netsky does not even construct a properly formatted HTTP request according to RFC2616 [15]. This makes it rather easy to identify the HTTP requests of the worm as will be demonstrated later.

DNS

Since the domain names that Netsky attacks and the email addresses that it emails are not IP addresses, they must first be converted via a DNS query. Netsky uses the built in resolver libraries of Windows to try to resolve domain names and MX records (records that point to the appropriate mail server for a domain) first, however it does have its own built in DNS routines in the event that the local DNS resolver is misconfigured but the host still has network access. DNS packets are UDP packets sent to port 53 on the server. The DNS specifications (RFC3658 [16]) do include TCP, however that is only for zone transfers. Normal client interactions are all via UDP.

Relevant DNS queries mentioned in this paper include CNAME, A, and MX. 'A' records are the most basic DNS record, translating a domain name into an IP address. CNAMES are like shortcuts to 'A' records. They allow one domain name to be pointed to another domain name. To resolve the first domain name, an A record query is sent, a CNAME response is received, and an A record query is sent for the second domain name, and so on until an A record is received. MX records are used for mail. An MX query for a domain name returns the domain name of the host that handles mail for the original request.

Successful Infection

A successful infection from one host to another occurs in the following process. First an email address is found on the hard drive of the infected machine inside files containing the following extensions: adb, asp, cfg, cgi, dbx, dhtm, doc, eml, htm, html, jsp, mbx, mdx, mht, mmf, msg, nch, ods, oft, php, pl, ppt, rtf, sht, shtm, stm, tbb, txt, uin, vbs, wab, wsh, xls and xml.

The appropriate mail server for that address is found via an MX query to the configured DNS server. Since Netsky first tries to use the local system resolver, it's possible to use the Windows 'hosts' file (%systemroot%\system32\drivers\etc\hosts on Windows XP and 2000, and %systemroot%\hosts on Windows 98 and ME) to force the worm to resolve the

addresses to be attacked. It's unfortunately not possible to use the hosts file to force the worm to send emails elsewhere as MX queries do not use the local hosts file as a reference.

Once email addresses have been harvested from those files, the virus uses its own built in SMTP engine to send them an infected email (Sample 1: Sample SMTP transaction). In the X variant, the email is targeted to specific languages (German, Finnish, French, Italian, Norwegian, Polish, Portuguese, Swedish, and Turkish). The virus attempts to match the top level domain extension on the email address and if it matches the domains of one of the above countries, it attempts to send a language specific message. In the case of the Turkish message, the email is actually going to a domain name in the Caribbean (tc) instead of Turkey (tr). The Y and Z variants don't try to guess languages and are in English only.

The payload is a very simple note about a "document", "notice", or similar terms in the X and Z variants, while the Y variant uses a slightly more clever approach of trying to appear to be a bounce request from a mail server.

None of these variants use the P2P or local file share spreading mechanisms that were used in some of the other Netsky variants. The executable file name of the X and Y is "FirewallSvr.exe". The file is created in the %systemroot% directory, and registry entry is created named FirewallSvr that starts the worm when Windows loads. The Z variant uses the executable file name "Jammer2nd.exe" and Jammer2nd as the registry entry name.

From there a new host is infected when a user checks their email [17] and opens the attachment and the cycle repeats.

Backdoor

The Y and Z variants also have backdoors that can be used to upload and run additional code. TCP port 82 is opened by the Y variant, and Z listens on TCP port 665. Data sent to that port is saved into a file and executed. While it is not legal, advisable, or even within the realm of common sense, some individuals have suggested automated hack-back systems that could 'clean' worms via these backdoors. In fact, that is similar to what the Netsky authors claim to be attempting. The system would detect that an attacking host (whether the attack was an infected email or a DDOS event) was infected with Netsky, then connect back to the attacking host and attempt to upload cleansing code.

It is possible to use the backdoor methods on machines which are under the legal authority of an administrator. For example, if a Netsky outbreak were to occur on a large scale network which had a CIRT legally able to remotely manage those machines, but lacking the technical resources (each department has different password sets, de-centralized day-to-day administration of individual hosts), they could develop a tool to scan for machines with the

backdoor ports open on their network, and attempt to upload various cleaning tools to remove the worm. Unfortunately, even that approach is probably ill-advised. There is no substitute for proper management of hosts via legitimate methods, and being able to push out virus updates and other patches and fixes through normal channels. In an emergency situation, however, such an action could be considered as a possible response.

The Attack

Generic DDOS

A denial of service attack is an attack in which service is denied. The addition of distributed does exactly what it should to the definition: denies service in an attack originating from a variety of distributed locations. There are a variety of non-distributed denial of service attacks, such as carefully formed packets that result in resources being exhausted through clever manipulation of data. However, most DDOS attacks come in two flavors. The first, like Netsky, is the result of a worm that is pre-seeded with an attack destination and time schedule. The second is a collection of infected machines (bots, worms, or otherwise) remote controlled to initiate a DDOS against a host.

The mechanism used by the individual hosts participating in a DDOS attack can vary, however they typically operate on the same principle. Overwhelm the target with so much information that it cannot respond to legitimate queries. More efficient DDOS attacks require the target to do more work in response to the attackers. An HTTP request, for example, can be one small packet, requiring the target web server to return a much larger web page in response. With enough bandwidth, even the simplest attack is debilitating. For example, a SYN flood is a stream of TCP SYN packets to the target. There are a variety of tweaks [18] to allow targeted hosts to not track SYN packets in their network stack and thus save on computation necessary to maintain state on SYN packets. However, even with such protections, if the bandwidth taken up by the SYN flood can exceed some large percentage of the available bandwidth to the target, then legitimate packets will be unable to reach the target, and a denial of service will still be affected.

Specifics of Netsky HTTP DDOS

The DDOS behavior of the Netsky variants activates during a specific date range (Figure 1: Timeline). Much of this analysis was from Joe Stewart in correspondence to a closed security list. His research is summarized here with his permission. Additionally, each of the three variants was analyzed in a VMWare sandbox for verification. A FreeBSD server was configured to fake DNS responses to the worm, and Ethereal was used to sniff the traffic. While the VMWare results did not correspond to the description below, the description as follows is the designed behavior of the DDOS code in Netsky. The DDOS code first decrypts the stored target server domain names:

```
ooo.nqcuk.uf          (www.educa.ch)
ooo.anqbmvw.cvh.nqc   (www.medinfo.ufl.edu)
ooo.mbebz.qn          (www.nibis.de)
```

Then each domain name is resolved into an IP address. Fifty threads are spawned, each thread choosing one target at random. An HTTP connection (TCP to port 80) is made to the target server and the following text (without the parenthetical remark) is sent:

```
GET / HTTP/1.1
Host: www.medinfo.ufl.edu

(note the extra blank line at the end)
```

The thread then sleeps for 250 milliseconds, repeats the HTTP request, and cycles through the last two steps until the worm exits.

While this would nominally result in only 200 requests per second, when the attack is multiplied that by hundreds of thousands of infected machines, it becomes quite powerful indeed.

Network Infrastructure

See Figure 2: Network Topology for a graphical overview of the network. An infected user begins to DDOS and follows the steps described in the specifics of the Netsky DDOS above. While the attack never reached the level where it disrupted service to the UF campus as a whole, if the DDOS target had a smaller network connection, it have been disrupted in a path from the inbound connection to UF to the host receiving the attack. Ignore the server called Sticky for now. It is introduced in the incident handling section.

Attack Signatures

Infected Hosts

The Netsky worms are identifiable by a number of methods. The first line of defense for most viruses is AV software. If current updates were installed on attacked hosts or mail servers, the virus could have been detected and stopped that way. For specific version and update information, see the AV database in Appendix A.

Since Y and Z had backdoors on infrequently used low-port numbers, it would be useful to use flow-analysis tools to detect TCP traffic to port 665 and 82 for possible abuse of the backdoors. Also, it would be easy to use a port scanning tool such as Nmap to port scan for machines with either 665 or 82 open, and further investigate those hosts as potential infections.

While it is unlikely that many infected clients on networks would have used the hard coded DNS servers instead of their default local DNS servers, DNS traffic to the following IPs could be used as an additional detection method:

```
145.253.2.171
193.141.40.42
193.189.244.205
193.193.144.12
193.193.158.10
194.25.2.129
194.25.2.130
194.25.2.131
194.25.2.132
194.25.2.133
194.25.2.134
195.185.185.195
195.20.224.234
212.185.252.136
212.185.252.73
212.185.253.70
212.44.160.8
212.7.128.162
212.7.128.165
213.191.74.19
217.5.97.137
151.189.13.35
```

DDOS Signatures

If a host was a DDOS target of the Netsky virus and had not received prior warning, the first alert would likely be from high traffic patterns to a website that was not normally as loaded. Examining the web server logs would reveal a large number of log entries such as:

```
aa.bb.cc.dd - - [03/May/2004:21:02:21 -0400] "GET / HTTP/1.1" 200 3333 "-" "-"
aa.bb.cc.dd - - [03/May/2004:21:02:21 -0400] "GET / HTTP/1.1" 200 3333 "-" "-"
aa.bb.cc.dd - - [03/May/2004:21:02:21 -0400] "GET / HTTP/1.1" 200 3333 "-" "-"
```

In this case, aa.bb.cc.dd is the IP address of one of the attacking hosts. Many repeated lines would be indicative of a DDOS, or at least a poorly configured client.

Of particular note are the last two fields of the requests above. Those "-" fields serve in the place of the HTTP Referer and the User-agent field. In this case, both of those fields being empty is highly suspicious. Requests may come to a web server with no referer field specified either because the link was typed, came from a bookmark, or some software or proxy stripped the field. It's unlikely to have a blank User-agent field and such behavior is very indicative of some sort of untoward activity.

The Netsky HTTP request (Sample 2: Netsky HTTP DDOS Requests) contains the bare minimum of information necessary to retrieve the default webpage for the domain name. The lack of any other HTTP fields is a telltale sign that the activity is not legitimate.

Depending on the security architecture of the targeted site, other systems may notice the attack before the web server administrator. It is possible that various Intrusion Prevention Products, QOS or bandwidth monitoring systems, or even user complaints could be the first indication that the attack is under way. UF was certainly lucky in the amount of warning they had for this incident.

DDOS Protections

The following are some suggested responses to typical DDOS attacks:

- ⇒ Upstream block: The worst DDOS attacks threaten much more than a single host. Whatever they are targeted at, any time a DDOS

generates enough traffic to fill up available bandwidth to an organization, any traffic to or from that site could potentially be impeded. In such cases, it is extremely important to have good communication with upstream bandwidth providers. Many DDOS attacks can only be stopped by proper communication with the upstream provider and asking them to place blocks that will prevent the malicious traffic from entering the connection that is currently flooded with traffic.

⇒ DNS black hole: When a DDOS is coordinated in advance via domain names, the domain name can be changed so its IP address is either a loop back IP address (and thus the host attacks itself), or some other non-routable address. This does deny legitimate service to the same domain name as well and is typically a last resort.

⇒ Move IPs: In cases where a prearranged DDOS is targeted at a specific IP instead of a domain name such as in the Code Red worm [19] attacking the White House, the domain name can be changed to a new IP address, and the original IP address traffic can be blocked at the upstream provider without any effects to the legitimate user population.

⇒ Security Products: A number of firewalls and Intrusion Prevention Systems (IPS) include the capability to either automatically or manually block certain types of traffic. The specific abilities depend on the product, but some are powerful enough to detect malformed HTTP packets and disallow only those packets; so even legitimate traffic from worm infected hosts would still be able to reach the target organization even though the attacks from the same hosts were not.

⇒ Black Hole Routing: Black Hole Routing is a technique [20] used to send packets to a null interface (essentially through them away) and can be used in conjunction with router ACLs to block hosts within routers on the own organizations network. The connection to the upstream is still susceptible, but for DDOS attacks in which that bandwidth is not overwhelmed, it is likely easier to make changes via Black Hole Routing or ACLs.

Incident Handling

There are two distinct perspectives of incident handling in the case of Netsky. There is incident handling from the perspective of those infected with the virus, and incident handling from the perspective of the DDOS targets. UFIRT was fortunate to only have to worry about one of these perspectives as no infected hosts were found on the UF network. IDS signatures were created to detect the http traffic (Sample 2: Netsky HTTP DDOS Requests) used in the denial of

service attack coming from internal UF networks. No such attacks were detected.

Incident handling for infections is fairly straightforward as Netsky behaves similarly to most modern email worms. For more information on incident handling of worm and virus outbreaks, see the following references: [21, 22, 23]

Summary

Most organizations will not be the target of a worm initiated DDOS such as that experienced by UF during the Netsky attack. Those that do, however, will face a rare challenge. The likelihood of such attacks is increasing as worms are becoming more common. Increased code sharing between virus and worm authors has resulted in more individuals capable of creating and abusing groups of compromised hosts used in such attacks.

The six steps of Incident Handling (Preparation, Identification, Containment, Eradication, Recovery, and Lessons Learned) [24] must be modified slightly when dealing with an incident such as a received DDOS. For example, while Identification of an incident is normally a more subtle task, when targeted by a DDOS, it's a much more obvious process. This is due to the very nature of a DDOS; it denies services that would otherwise be available.

For an overview of the incident, Figure 1: Timeline outlines the timeline of the incident through the various phases, as well as the DDOS attacks time frame.

Identification

Identification is usually the second step of Incident Handling. However, initial identification that a threat existed was received by UFIRT in the form of an email forwarded by the administrator of the www.medinfo.ufl.edu server around noon, Tuesday April 20th. The administrator was notified of the pending attack by the administrator at one of the other two targeted servers. UFIRT uses abuse@ufl.edu as the central point of contact for abuse handling on campus, per RFC2142 [25]. Thus, the administrator of the server in question was able to easily contact UFIRT and begin the Incident Handling Process.

Notification should be considered an ongoing process throughout incident handling. UFIRT gathered a list of email addresses for all the relevant parties who would be involved in the incident (the server administrators, network administrators, appropriate management contacts in multiple departments, the DNS administrator) and used email and phone calls as the primary methods for communication.

After the initial information was received and while the appropriate parties were gathered, the investigation began. First, independent verification was found through research on various anti-virus vendor databases. Second, additional information was sought on the specifics of the DDOS method on a closed

security mailing list. The information that was gathered indicated that the threat was real and growing. By Wednesday the 21st, two additional variants had been released and analyzed.

Preparation

After the initial information was processed, the response team gathered and response options were considered. One fortunate aspect of the situation helped UFIRT in planning a response to the DDOS. Two names were listed in the DNS configuration for the UF target: www.medinfo.ufl.edu and medinfo.ufl.edu. The A record was medinfo.ufl.edu and www.medinfo.ufl.edu was a CNAME or alias. Since the worm was programmed to attack www.medinfo.ufl.edu, removing that domain name from DNS would prevent the worm from resolving an IP address for the target, while legitimate users could simply use medinfo.ufl.edu without any interruption of service.

Additionally, most of the primary users for the domain were on campus, and could be instructed to change their bookmarks to point to the new domain in a relatively short period of time. Thus, the impact to the primary user population could be minimized.

There were unfortunately two faults in this plan. The first was that it was known that the virus had a hard-coded list of name servers in its configuration, and it was unknown if those could be used by the attacker to force the old DNS data and still attack the host. The second problem was some traffic to the target server originated from search engine requests, which often are pointing to the targeted domain name. Another plan would have to be implemented to maintain service to that population.

To resolve that issue, the plan was modified to include a hardened Linux server in the central networking machine room running a modified web server whose only function would be to redirect client requests to the new domain. The idea was that even if the DDOS were able to overcome the forwarding server, the real server was still protected.

Unfortunately, the largest flaw with this plan was that not only could the forwarding server still be successfully attacked (though a separate machine built to be attacked could theoretically handle attacks much more efficiently), but also the UF upstream internet connection could conceivably become saturated as well. Upstream saturation is an even more serious threat than the direct threat against the specific web server because it impacts network services to the entire campus.

While UFIRT was researching the mechanisms and behavior of the Netsky virus on an internet security mailing list [26] Joe Stewart made the suggestion of using a tarpit to trap the worm traffic.

The LaBrea Tarpit is the famous archeological find that had ensnared dozens of dinosaurs in sticky tar, trapping the bones and preserving them until they were dug up in modern times. Likewise, the LaBrea Tarpit [27] for networks traps network worms and was developed by Tom Liston in response to the Code Red worm. LaBrea tied together two clever ideas. The first was that it would automatically respond to requests to unused IP address spaces. This was accomplished by monitoring ARP requests on the local network and when an ARP request went unanswered for a specified period of time, the host running LaBrea would respond, temporarily assuming that IP address. While that idea was not unheard of, the second was rather unusual, and was the most important aspect. LaBrea would shrink the TCP window size and not send return ACK (acknowledge) packets to the host communicating with it.

From a practical standpoint this means that hosts communicating with LaBrea keep trying to communicate, fail, and keep trying again until the TCP connection times out about 12 minutes later for Windows hosts. The Netsky DDOS code, for example, spawns 50 threads that each attempt to connect to the target web server, send a simple HTTP request (Sample 1: Sample SMTP transaction), sleeps 250 milliseconds, and sends again. This allows one copy of the worm to optimally attack about 200 times per second. That's 144,000 attacks in 12 minutes. If the worm were to attack a tarpit, each thread would take 12 minutes to time out, so there would only be 50 attacks in 12 minutes. Being able to reduce the attack flow by nearly 3000 times might be the difference between turning a crippling DDOS into a small traffic spike.

Stewart's tarpit suggestion helped address the additional problem of saturating the UF upstream connection, and it fit perfectly into the current plan. The forwarding web server would be modified to not only forward the legitimate traffic, but detect the worms and tarpit their DDOS attempts.

Given that the tarpit would not actually be a true tarpit (trapping everything that connected to it), but would be selective, it was given the name Sticky, and the concept called a stickypit.

The plan was approved by all involved parties, and UFIRT began to build the stickypit.

Two possible ways to implement the stickypit were considered:

- ⇒ Use the original Tarpit source code [27] and modify it to not use ARP redirection. Additional code would have to be added for the initial handling of an HTTP connection and worm request detection, and use the existing source for tarpitting of subsequent connections using either a tracking system of previous worm requests (implementing any fast search system to check the list of previously detected hosts), or by changing the sequence number in a manner that would enable

subsequent packets from worms to be tarpitted without having to maintain a state table of which attacking hosts were infected.

⇒ Use the IPTables Tarpit module which is an extension of the Linux Netfilter kernel 'packet mangling' code [28]. IPTables has a simple and easy to use interface, and would only require some relatively simple TCP listening code to simulate an HTTP server, send the client a response, and if the client request matched a worm request, generate an IPTables command-line entry to tarpit the host on any subsequent requests.

Given the relatively short time frame, the second option was chosen. Mandrake Linux [29] has the Tarpit module compiled into the distribution. After some initial, unsuccessful, effort was put into compiling the modules on Redhat Linux, Mandrake was eventually used as the easiest out of the box solution for the tarpitting modules.

While the system was built and the code was being written to do the detection and HTTP forwarding, simultaneous research was being done on the Netsky variants (all three were out by this time). In VMWare testing (and even a 'real' system that could afford a rebuild), the worms all suffered a fatal crash whenever the date on the system was set to the trigger date of the worm and the DDOS code was activated. It seemed entirely possible that there would be no DDOS at all.

UFIRT decided that regardless of lab testing, the stickypit development would continue.

UFIRT coordinated with DNS administrators to redirect www.medinfo.ufl.edu, the CNAME for medinfo.ufl.edu, to Sticky.

Containment

Sticky was prepared for the DDOS attack and the original server was moved to a domain name that was not targeted by the worms. Unfortunately, like most real-world scenarios, once the attack began, many changes and tweaks were required to maintain performance of the stickypit.

While some small amount of attacks came in earlier (likely hosts with incorrect time settings), the true DDOS began near the time predicted by the malware analysis (Figure 3: Tarpit Bandwidth). Bandwidth to Sticky was very sporadic however, and though there were still definite trends, there was very little steady growth or decay, but rather many spikes and valleys. Further analysis of the lab tests on Netsky, as well as monitoring of the traffic revealed the reason.

The X, Y, and Z variants of the Netsky worms do have a fatal flaw. There is an error that occurs in the DDOS thread (Figure 4: Netsky Exception) of the worm; however, it occurs at different times on different machines and under different

loads. It appears that an infected host under low load during the DDOS phase will tend to exit out immediately. However, as load increases on an infected host, it is able to complete more attacks before it crashes. This is likely why the behavior did not show up in initial testing; since the infected lab hosts were only being used to analyze the worms, their load was minimal. Sometimes the code crashes before any attacks are sent to the target, sometimes many complete attacks are sent before crashing, and sometimes the HTTP request becomes a SYN flood [30].

The crashing behavior was both good news and bad news. The good news was that Sticky would receive much less load than if the attack code worked correctly. For the bad news, it meant that the server now had to deal with a variety of behaviors instead of one well-defined behavior with one pre-programmed response.

Specifically, SYN flooding is a DDOS attack by itself that can be quite effective at exhausting resources as a target's network stack must track each SYN packet that is sent out until it times out and respond with a SYN ACK packet. Changes were made to the syncookies option in sysctl [18] to help prevent Sticky from wasting resources on SYN attacks.

Xinetd [31] was used as the socket listener on port 80 to direct open sessions to the tard (Sample 3: Tard.c) executable. Some minor modifications were made to allow xinetd to run as desired. The NOLIBWRAP option was used to force xinetd to hand the process to tard instead of the normal http server.

Another change to xinetd was the backlog value was adjusted to prevent queued connections from being dropped. The compiled value of backlog in xinetd is 7. The backlog value [32] is passed to the listen function that listens on a socket. A backlog value of seven means that if xinetd is handling a connection, the kernel will only allow 7 connections to be queued before additional connections are dropped. This resulted in unacceptable performance from the xinetd daemon in conjunction with Tard. Xinetd was recompiled with a backlog value of 32000 to prevent queued connections from being dropped.

The biggest change that was made to Sticky was with IPTables. The IPTables kernel module was used instead of manually building such code into Tard for several reasons. It made more sense for the kernel to handle the tarpitting as it has more direct access to the network, had the highest execution priority on the machine, and would therefore be able to process the incoming network traffic more quickly than a non-kernel process. Additionally, it was trivial to have Tard add hosts to the tarpit list via an external shell script. Thus, this was a much faster solution.

Unfortunately, as the DDOS attack began to ramp up, it became apparent that the stickypit could not handle more than 4000 or 5000 hosts on the tarpit list.

Temporary measures were developed to handle the problem. Watchdog scripts were run to monitor the number of tarpitted hosts and flush the IPTables chains when they reached a certain threshold. The threshold was adjusted up and down as appropriate while the root cause was investigated. It became apparent that the problem was due to the way IPTables processes incoming packets against the list of rules. IPTables processing rules are called 'Chains' and for good reason. They are just that—linear chains of tests to which a packet is compared to determine how it will be processed. Every packet that received by Sticky it had to be compared sequentially to each of the tarpit entry rules. Sticky was quickly overloaded by 3000 rules times the number of packets entering per second. Traffic peaked above 64,000 packets per second. There was no way to handle so many transactions per second using a linear search of a rule set thousands of entries long.

The problem stems from the inefficient way that IPTables parses its rules. A linear search pattern is one of the slowest algorithms possible to search a list of rules. While it makes sense for IPTables in its normal operation (in which it might be doing more than one thing to an inbound packet), in the simple case where there is an extremely long list of rules with only two results (tarpit or no tarpit), it's highly inefficient.

There were three solutions considered to deal with the problem. The first was to rewrite the IPTables code itself so that it used a more efficient search method. The second was to move the tarpit code out of the IPTables module and into the Tard process as was originally considered. The third was to use sub-chains within IPTables to presort the packets. The first two options were much more labor intensive and difficult than the third, (though they are potentially better long-term solutions) and since the attack was still in progress, the third option was chosen.

The third option was a quicker and easier solution to allow IPTables to use more intelligent sorting and rule comparison. 256 sub-chains were created, each named from 0-255. Then 256 rules were created that automatically forwarded incoming packets based on their first octet into the appropriate sub-chain. The script that Tard called to add hosts to the tarpit (Sample 4: Stickypit scripts) was updated accordingly so that it inserted rules into the appropriate sub-chain.

This resulted in a much faster processing speed. This modified search algorithm was able to handle higher traffic load without requiring flushes of the IPTables database as frequently.

Since hosts that were infected did not typically remain attackers for long, periodically flushing the IPTables rules prevented Sticky from becoming overloaded by IPTables processing, and still tarpitted most hosts for an effective period of time. Some anomalous hosts were found that continued to attack, but UFIRT was unable to track down the appropriate contacts for those infected

hosts to determine why they were not subject to the same faults that crashed most infections.

Eradication

A very fortunate circumstance of the Netsky worm was that, not only did it have its own built in accidental self-destruct mechanism, but the attack was only designed to run for a short period of time. Eradication of the DDOS attack was only a matter of out-waiting the attackers. Indeed, as the bandwidth graph demonstrates (Figure 3: Tarpit Bandwidth), the bandwidth returned to pre-attack levels at the expected time.

Future DDOS recipients may not be so lucky. If the attack were to continue for a longer period of time, or if it was strong enough to flood UF's upstream connection, other action would be needed. One such possibility would be to simply remove the www.medinfo.ufl.edu domain entirely. Without any DNS, the worms would fail to resolve the domain and be unable to attack. As mentioned previously, the full domain name was redundant and while there were some users (specifically users who found links via search engines) who were using www.medinfo.ufl.edu, the internal audience was easily able to update their bookmarks to the new URL.

Since the targets were hard-coded, there was little the Netsky author could do to update the attack as situations changed. Thus, any domain name or IP address that is pre-programmed can either be removed, or blocked at the upstream provider to reduce the impact of the denial of service on the target network, though the targeted host may be unavailable to legitimate users.

The worst case scenario is when a malicious individual controls a network of compromised machines through an active control channel and they can be retargeted as necessary. It is possible that a modified stickypit could be used in such a scenario depending on the situation.

Recovery

When the DDOS was confirmed to be complete, the original DNS configuration with www.medinfo.ufl.edu as a CNAME to medinfo.ufl.edu was restored. Note that since that time, the CNAME has been retired, and only the shorter URL is used. Ironically, if the attack were to occur today, there is no layered DNS structure of a CNAME to a separate name that could be used as it was last April and May.

Since this incident did not involve any compromised machines on the UF network, the recovery process was very short and simple. Extended members of the UFIRT that had been called in specifically for their role in this specific incident were retired. Sticky was removed from service, and incident handling returned to a steady state.

Further analysis was conducted of the medinfo server to determine why it had been targeted, but no conclusions could be drawn. No evidence of a compromise or use in any other activity was found, but lack of evidence unfortunately proves nothing. The reasoning behind the target selection for the DDOS remains unclear to this day.

Lessons Learned

During this event, the first and second steps of incident handling were reversed since identification that the event would occur was received before the actual event, and preparation occurred in between identification and the event. UFIRT was able to develop appropriate responses in time to mitigate the attack. The experience with the Netsky DDOS adds another technique to UFIRT's response capabilities.

One lesson that was definitely learned was that no network can ever be considered safe from DDOS attacks. Despite never finding any reasons why, UF was still attacked by a fairly large scale denial of service attack.

It's difficult to speculate what could be done to prevent an attack like this from reoccurring since the actual cause of the initial attack is still unknown. There are, however, improvements that could be made to the stickypit system that would hopefully enable it to be a more robust system in future DDOS attack incidents.

Future directions for improvement include:

- ⇒ Proxy detection: Post review of logs during the attack indicated that some hosts were not being detected as worms because their HTTP traffic was modified by Proxies in between the attacker and the stickypit. More robust pattern matching could be added to Tard to enable such detection.

- ⇒ Per session sticking: The easiest way to avoid having to track a list of IP addresses that need to be tarpitted would be to selectively tarpit a specific TCP session immediately after the worm had made the unique HTTP request. In that way there are no overhead issues associated with trying to maintain state on all infected hosts. However, this requires that the tarpit technology be moved inside of a Tard like process instead of executed externally as the code does now.

- ⇒ Use of stickypit technology in other situations. For example; combine firewall technology with tarpit technology and only allow access to legitimate web servers, while tarpitting HTTP requests to all other hosts.

With the incident long finished, and after additional research for this paper, there are still questions left to be answered:

⇒ Why was the www.medinfo.ufl.edu server targeted in the first place?

⇒ What causes the Netsky DDOS code to crash?

⇒ How would the current stickypit design handle a repeat attack from a fixed version of Netsky?

While this incident handler has no desire to find out the answer to some of these questions on a professional level, from a personal perspective, it's always fun to guess.

References

1. Roberts, Paul. "Experts weigh Sasser-Netsky worm connection". 03 May 2004. URL: http://www.computerworld.com/securitytopics/security/virus/story/0,10801,92871,00.html
2. Lemos, Robert. "Netsky authors possibly penned Sasser". 03 May 2004. URL: http://news.com.com/2100-7355_3-5204930.html
3. Sophos. "Suspected Sasser worm author caught; could trigger more arrests, says Sophos". 13 May 2004. URL: http://www.sophos.com/virusinfo/articles/sasserarrest.html
4. Lyman, Jay. "Author of Sasser, Netsky Worms Indicted". 09 September 2004. URL: http://www.technewsworld.com/story/36491.html
5. Wikipedia, the free encyclopedia. "Hacker". URL: http://en.wikipedia.org/wiki/Hacker
6. Lorek, Laura. "Russian Mafia Net Threat". 16 July 2001. URL: http://www.eweek.com/article2/0,3959,1237772,00.asp
7. Bullough, Oliver. "Police say Russian hackers are an increasing threat". 28 July 2004. URL: http://www.usatoday.com/tech/news/internetprivacy/2004-07-28-russian-hackers_x.htm
8. Verton, Dan. "Organized Crime Invades Cyberspace". 30 August 2004. URL: http://www.computerworld.com/securitytopics/security/story/0,10801,95501,00.html
9. Windows XP Service Pack 2. URL: http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/winxpsp2.mspx
10. Starr Andersen, Vincent Abella. Changes to Functionality in Microsoft Windows XP Service Pack 2. URL: http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2chngs.mspx
11. Postel, Jonathan B. "RFC 821". August 1982. URL: http://www.ietf.org/rfc/rfc0821.txt?number=821
12. Klensin, et al. "RFC 2821". April 2001. URL: http://www.ietf.org/rfc/rfc2821.txt?number=2821
13. Freed, et al. "RFC 2045". November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt?number=2045
14. Levinson, et al. "RFC 2112". March 1997. URL: http://www.ietf.org/rfc/rfc2112.txt?number=2112

15. Fielding, et al. “RFC 2616”. June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt?number=2616
16. Gudmundsson, et al. “RFC 3658”. December 2003. URL: http://www.ietf.org/rfc/rfc3658.txt?number=3658
17. VirusAll.com. Netsky.X Screenshot. URL: http://virusall.com/images/worm_Netsky_x_img1.gif
18. Burdach, Mariusz. “Hardening the TCP/IP stack to SYN attacks”. 10 September 2003. URL: http://www.securityfocus.com/infocus/1729
19. Lemos, Robert. “Web worm targets White House”. July 19, 2001. URL: http://news.com.com/2100-1001-270272.html
20. Smith, Philip and Barry Greene. “Remote Triggering Black Hole Filtering”. Draft 0.2. 02 August 2002. URL: http://www.netsys.com/library/papers/Remote_Triggered_Black_Hole_Filtering-02.pdf
21. Chromiak, Chris. “Mydoom in the Case of the Unsuspecting vendor”. 14 June 2004. URL: http://www.giac.org/practical/GCIH/Chris_Chromiak_GCIH.pdf
22. Gebhart, Glenn. “Worm Propagation and Countermeasures”. 02 February 2004. URL: http://www.sans.org/rr/papers/36/1410.pdf
23. Gadsden, Richard. “Mass-Mailing Worms: Prevention, Detection and Response (A Case Study)”. 08 August 2003. URL: http://www.sans.org/rr/papers/36/1148.pdf
24. SANS and Ed Skoudis. “Incident Handling Step-by-Step and Computer Crime Investigation”. SANS Institute Track 4 –Hacker Techniques, Exploits & Incident Handling. April 2004 (2004).
25. Crocker, et al. “RFC 2142”. May 1997. URL: http://www.ietf.org/rfc/rfc2142.txt?number=2142
26. The Trojan Horses Research Mailing List. URL: http://ecompute.org/th-list/index.html
27. Labrea – SourceForge. URL: http://labrea.sourceforge.net/
28. IPTables Tarpit Module. URL: http://cvs.netfilter.org/patch-o-matic-ng/TARPIT/
29. Mandrakesoft. URL: http://www.mandrakesoft.com/
30. CERT. “TCP SYN Flooding and IP Spoofing Attacks”. 19 September 1996. URL: http://www.cert.org/advisories/CA-1996-21.html
31. Xinetd. URL: http://www.xinetd.org/
32. BSD Man Page. “listen - listen for connections on a socket”. 23 July 1993. URL: http://www.freebsd.org/cgi/man.cgi?query=listen

Appendix A – Figures, Tables and charts

Table 1: Variants

Variant	Date	Notable/New Characteristics	References
A	02/06/04	Random email payload, copied to shared directories, may be in a zip file, removes Mydoom and Mimail registry entries	M, S, T, P, F
B	02/18/04	Similar to B	M, S, T, P, F
C	02/25/04	Different copies packed different with Petite, Aspack, or UPX, beeps	M, S, T, P, F
D	02/25/04	Announces the author(s) as the Skynet.cz AntiHacker Crew, PE packed	M, S, T, P, F
E	03/01/04	Similar to E	M, S, T, P, F
F	03/03/04	Embedded text calls Bagle a loser, tries to remove Bagle registry entries	M, S, T, P, F
G	03/04/04	Embedded text suggests to Bagle and Mydoom authors a meeting in person. Location appears to be encrypted. Removes registry entries of Mydoom, Mimail, Neysky(a,b), Nachi, and Bagle.	M, S, T, P, F
H	03/05/04	Embedded text: “Skynet AntiVirus - Mydoom and Bagle are children”	M, S, T, P, F
I	03/08/04	Similar to H	M, S, T, P, F
J	03/08/04	Packed with TeLock	M, S, T, P, F
K	03/08/04	Author claims this is the last variant in hidden text. Mar 13th popup displays “SkyNet has full control of your system now”, and Mar 16th popup displays “Please remove the file avpguard from your Windows-Directory and do not open attachments anymore. It can be a virus like bagle and mydoom or similar malicious code. This is the SkyNet-Antivirus!” Variant occasionally uses password protected zips; a technique stolen from bagle. Additionally, a backdoor opens up on port 26 on the Mar16th on infected machines. Authors promise source will be released soon.	M, S, T, P, F
L	03/10/04	Back to UPX packing. Appears to be a much stripped down version (T). Email texts and addresses are much less varied, and the mutex used to prevent multiple infections is new to the family.	M, S, T, P, F

M	03/11/04	Similar to L	M, S, T, P, F
N	03/17/04	New author claims to have ownership of the source code, calling itself NetDy. Begins deleting other registry entries again, which L and M did not.	M, S, T, P, F
O	03/17/04	Similar to N	M, S, T, P, F
P	03/21/04	Another new style of variant. Spreads via P2P again, is FSG packed, exploits Outlook preview vulnerability to allow it to be executed in unpatched Outlook environments when viewed, without requiring the attachment actually be opened. Additionally, this variant makes use of a dropper that drops an UPX packed dll which contains the worm payload.	M, S, T, P, F
Q	03/29/04	First variant with an attack payload. Similar DLL structure to P. Ddos against the following websites: www.cracks.st www.cracks.am www.emule-project.net www.kazaa.com www.edonkey2000.com	M, S, T, P, F
R	03/31/04	Similar to Q, removes even more registry entries (SDBot is added), and attacks the following hosts: www.keygen.us www.kazaa.com www.emule-project.net www.cracks.am www.emule.de	M, S, T, P, F
S	04/04/04	Stripped down from previous versions; does not spread via P2P anymore, and does not uninstall other worms. Has a backdoor on port 6789, and uses two processes to make shutting it down difficult. The irony is of course that the original Netsky team derided the evils of bagle because it had backdoors. Now sites attacked sites include: www.emule.de www.cracks.am www.freemule.net www.kazaa.com www.keygen.us	M, S, T, P, F
T	04/06/04	Similar to S	M, S, T, P, F

U	04/08/04	Similar to S (minor packing differences)	M, S, T, P, F
V	04/14/04	Backdoor ports on 5556 (ftp) and 5557 (http). The worm attempts to exploit IE enabled mail clients and then use the previously opened ftp and http servers to transfer itself. Same DDOS recipients.	M, S, T, P, F
W	04/16/04	Minor changes; occasionally emailed a specific AOL address.	M, S, T, P, F
X	04/20/04	See separate analysis	M, S, T, P, F
Y	04/20/04		M, S, T, P, F
Z	04/21/04		M, S, T, P, F
AA	04/27/04	No new behaviors; fake error message; no backdoor or payload.	M, S, T, P, F
AB	04/28/04	Removes Bagle	M, S, T, P, F
AC	05/02/04	Claims credit for Sasser in comments and shows sample source as proof	M, S, T, P, F
AD		There are discrepancies between AV vendors on details of these variants, even whether they exist.	M, S, T, P, F
AE			M, S, T, P, F
AF			M, S, T, P, F

Figure 1: Timeline

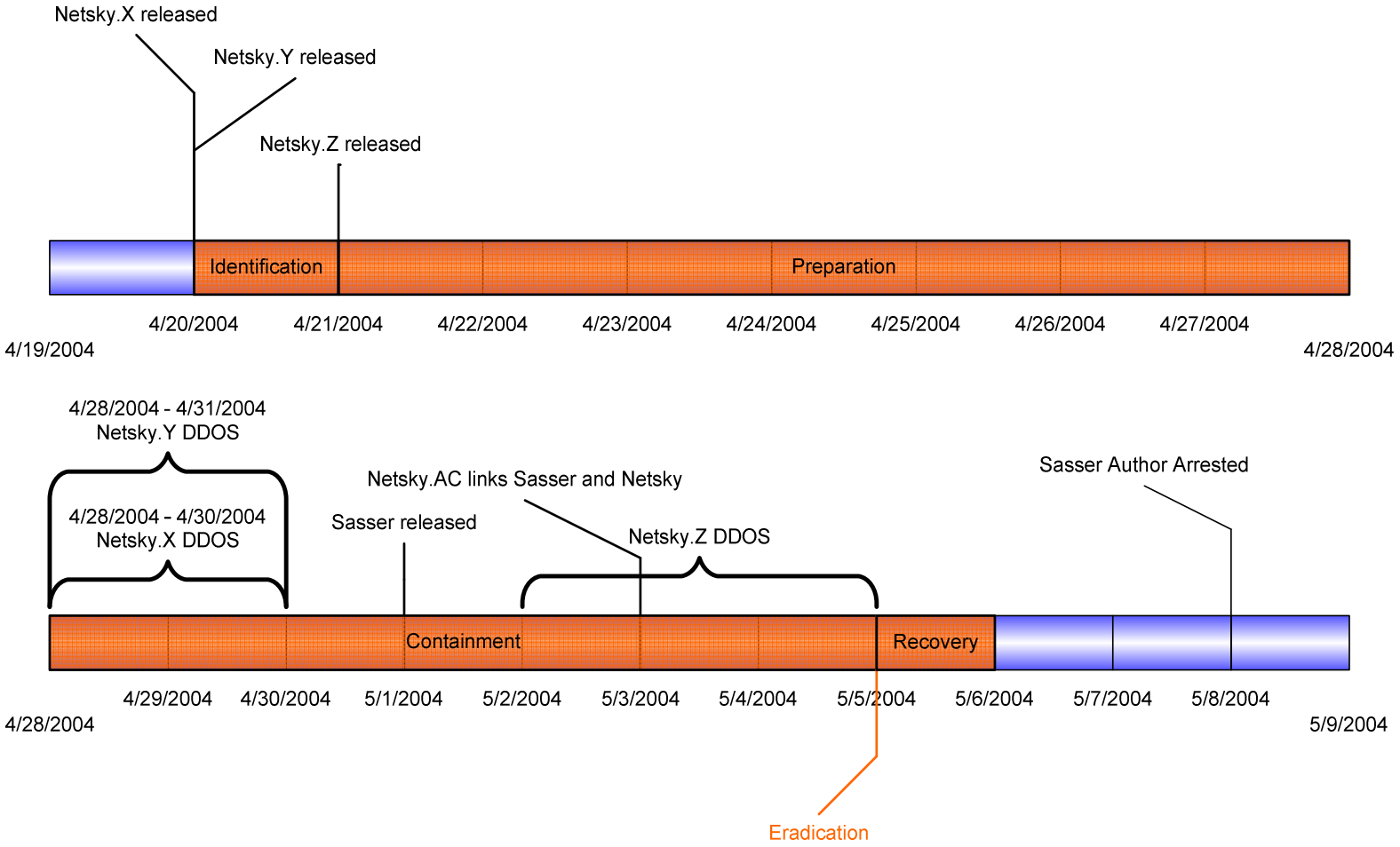


Figure 2: Network Topology

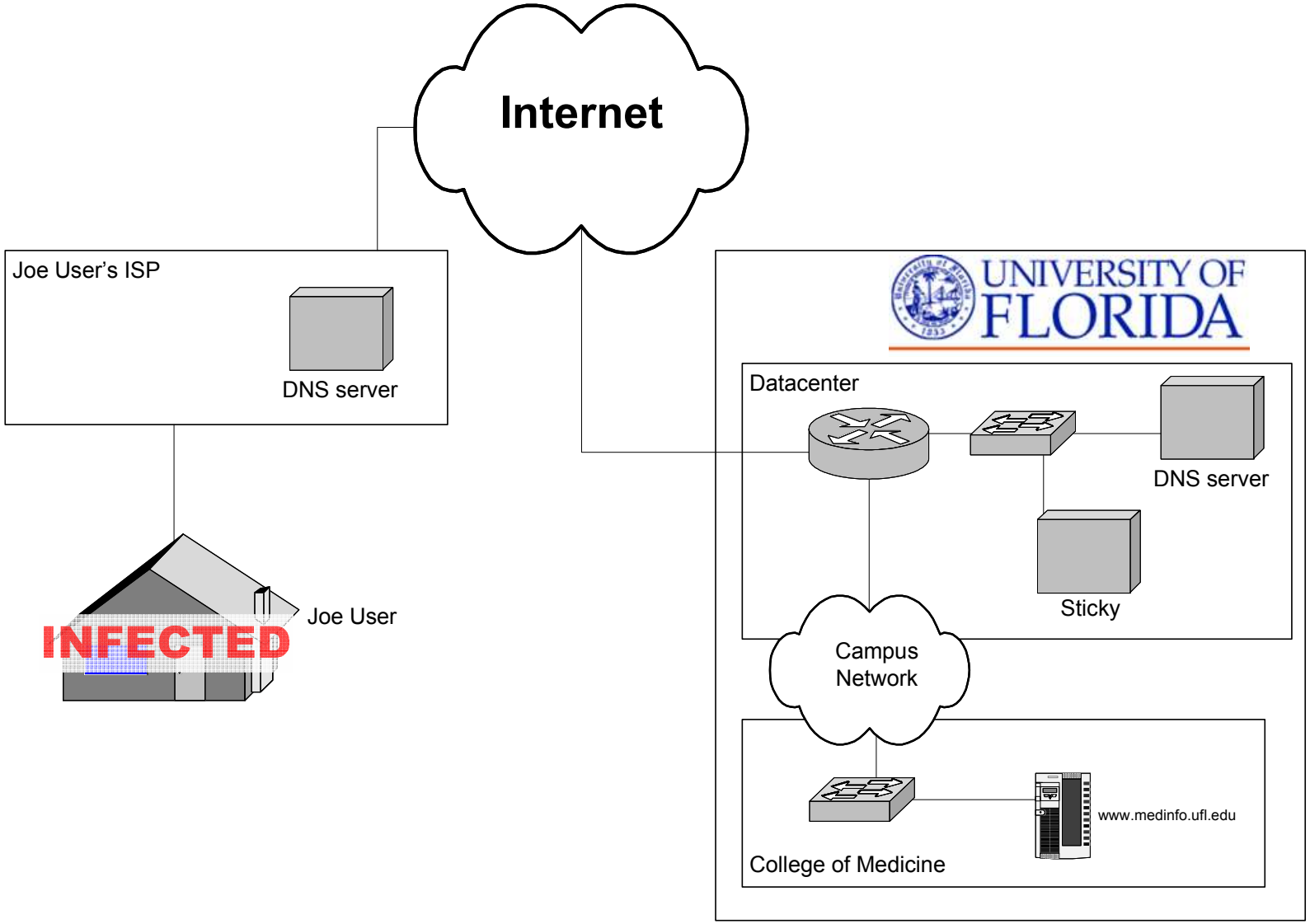


Figure 3: Tarpit Bandwidth

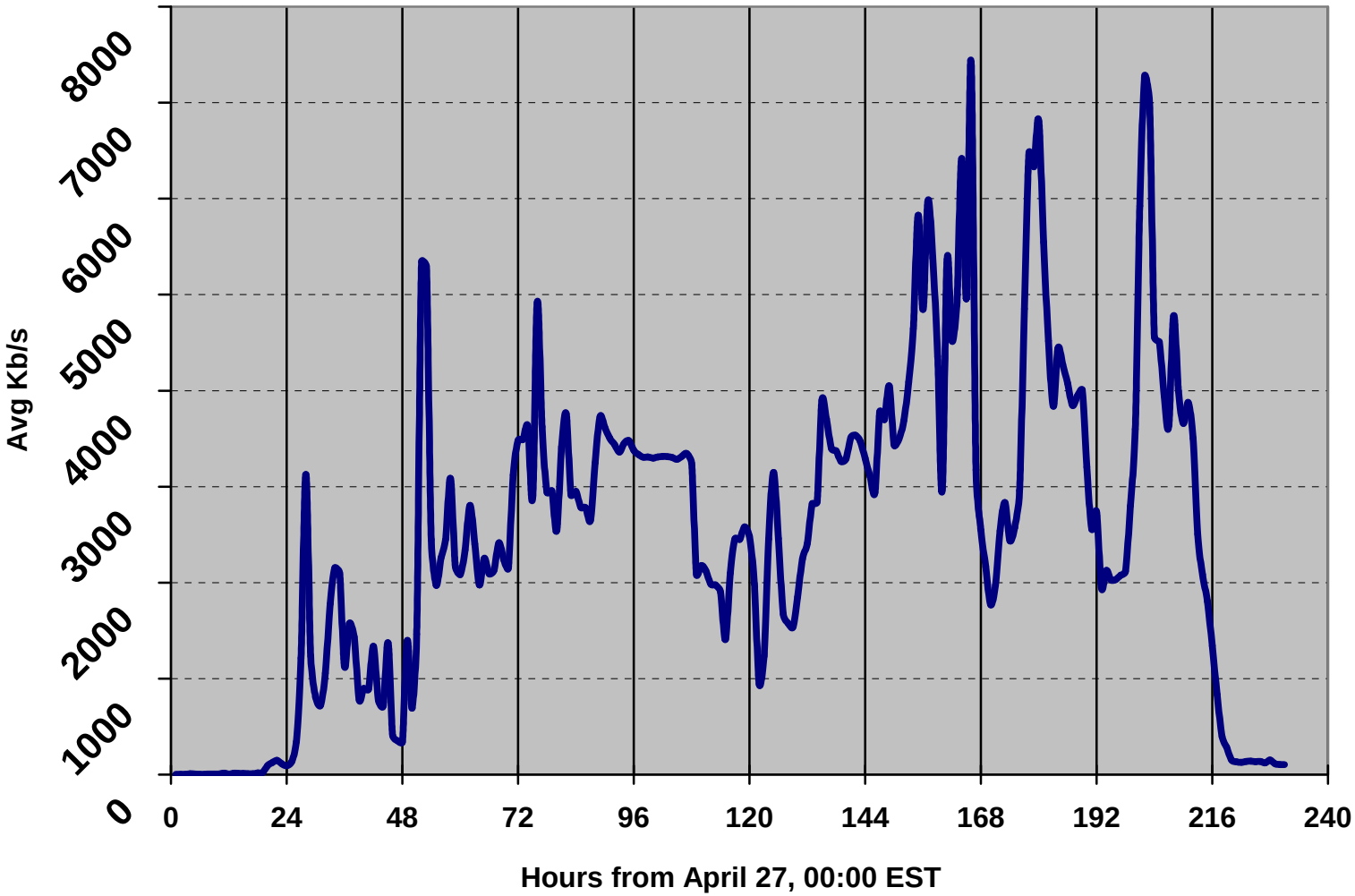
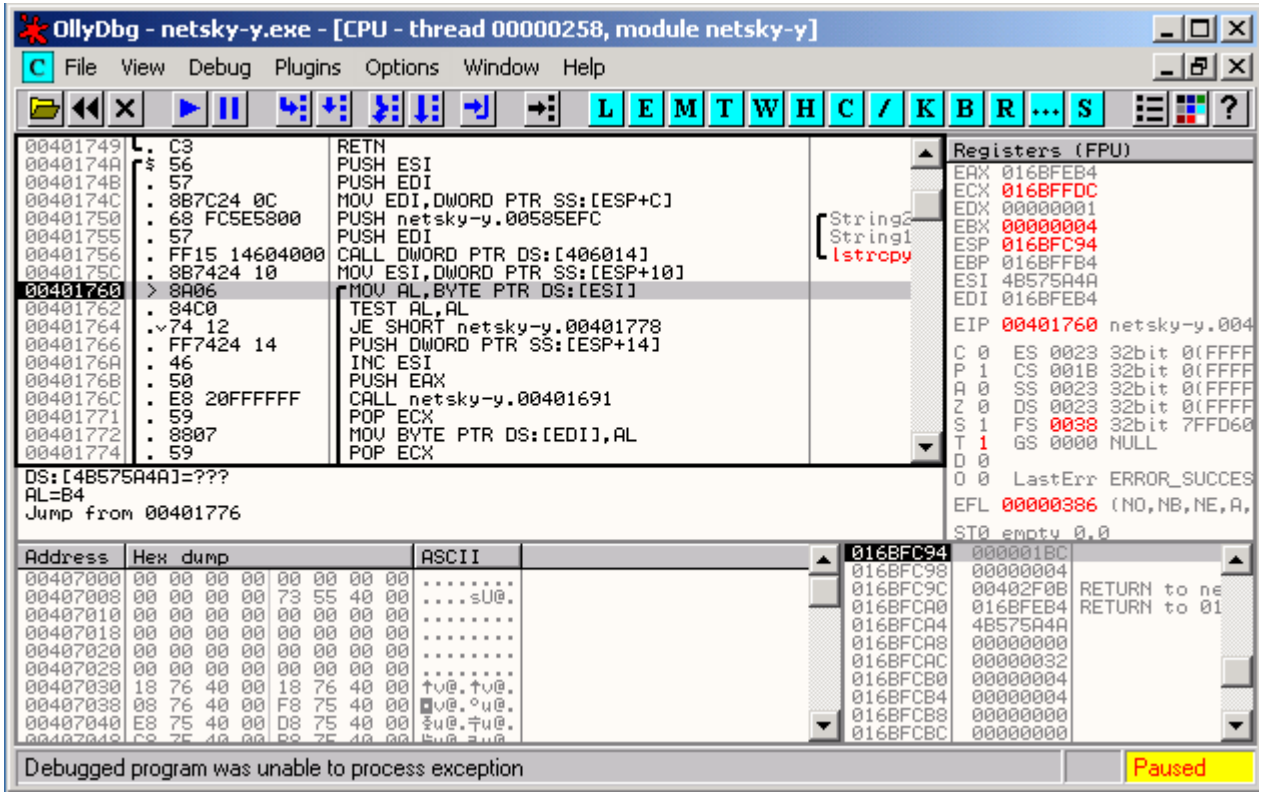


Figure 4: Netsky Exception



Appendix B – Code and Data samples

Sample 1: Sample SMTP transaction

```
220 localhost.my.domain ESMTP Sendmail 8.12.11/8.12.11; Fri, 30 Apr 2004 21:59:22 -0400
(EDT)
EHLO yahoo.com
250-localhost.my.domain Hello yahoo.com [192.168.42.128], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-DELIVERBY
250 HELP
MAIL FROM:<hukanmikloiuo@yahoo.com>
250 2.1.0 <hukanmikloiuo@yahoo.com>... Sender ok
RCPT TO:<hukanmikloiuo@yahoo.com>
250 2.1.5 <hukanmikloiuo@yahoo.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: hukanmikloiuo@yahoo.com
To: hukanmikloiuo@yahoo.com
Subject: Delivery failure notice (ID-000053C6)
Date: Fri, 30 Apr 2004 21:59:23 -0400
MIME-Version: 1.0
Content-Type: multipart/mixed;
.boundary="----=_NextPart_000_0004_0000673B.000076EB"
X-Priority: 3
X-MSMail-Priority: Normal

This is a multi-part message in MIME format.

------=_NextPart_000_0004_0000673B.000076EB
Content-Type: text/plain;
.charset="Windows-1252"
Content-Transfer-Encoding: 7bit

--- Mail Part Delivered ---
220 Welcome to [yahoo.com]
Mail type: multipart/related
--- text/html RFC 2504
MX [Mail Exchanger] mx.mt2.kl.yahoo.com
Exim Status OK.

New message is available.



------=_NextPart_000_0004_0000673B.000076EB
Content-Type: application/octet-stream;
.name="www.yahoo.com.hukanmikloiuo.session-000053C6.com"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
.filename="www.yahoo.com.hukanmikloiuo.session-000053C6.com"

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAqAAAAERuqUgAD8cbAA/HGwAPxxuDE8kbDA/HG+gQzRsZD8cbgweaGwIPxxsAD8cb
Aw/HGwAPxhtcD8cbYhDUGwkPxxvoEMwbBQ/HG1JpY2gAD8cbAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAUEUAAEwBBADtJ4RAAAAAAAAAAADgAA8BCwEGAABMAAAAAhgAAAAAG4+AAAAEAAA
AGAAAAAAQAAAEAAAAAIAAAQAAAAAAAAABAAAAAAAAAAAAoBgAAAQAAAAAAACAAAAAAAQAAAQ
                        <MAJORITY OF DATA CUT HERE>
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=

------=_NextPart_000_0004_0000673B.000076EB--


.
250 2.0.0 i8J1xMOS000251 Message accepted for delivery
QUIT
221 2.0.0 localhost.my.domain closing connection
```

Sample 2: Netsky HTTP DDOS Requests

```
HEX Dump                                                ASCII DUMP
00 0c 29 a0 e4 d7 00 0c 29 96 b5 9f 08 00 45 00   ..).....).....E.
00 4e 08 84 40 00 80 06 1b d4 c0 a8 2a 80 c0 a8   .N..@.......*...
2a 81 08 fd 00 50 d9 d6 48 f4 aa ff da 2f 50 18   *....P..H..../P.
44 70 73 e9 00 00 47 45 54 20 2f 20 48 54 54 50   Dps...GET / HTTP
2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e   /1.1..Host: www.
65 64 75 63 61 2e 63 68 0d 0a 0d 0a               educa.ch....

00 0c 29 a0 e4 d7 00 0c 29 96 b5 9f 08 00 45 00   ..).....).....E.
00 55 08 80 40 00 80 06 1b d1 c0 a8 2a 80 c0 a8   .U..@.......*...
2a 81 08 fc 00 50 20 9d 1e 32 ad 08 47 2d 50 18   *....P ..2..G-P.
44 70 22 d4 00 00 47 45 54 20 2f 20 48 54 54 50   Dp"...GET / HTTP
2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e   /1.1..Host: www.
6d 65 64 69 6e 66 6f 2e 75 66 6c 2e 65 64 75 0d   medinfo.ufl.edu.
0a 0d 0a                                          ...
```

Sample 3: Tard.c

```
/*
**  Tarpit daemon...   by Chuck Logan
*/

#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <errno.h>
#include <unistd.h>
#include <syslog.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* various static strings */

const char virus_line1[] = "GET / HTTP/1.1";
const char virus_line2[] = "Host: www.medinfo.ufl.edu";

const char redirect[] =
    "HTTP/1.1 302 Temporary Redirect\r\n"
    "Location: http://medinfo.ufl.edu\r\n"
    "Connection: close\r\n"
    "Server: Daemon/1.0\r\n\r\n";

const char tarpit_cmd[] = "/usr/local/sbin/tarpit";
#define TARPIT_ARGC 2
char tarpit_argv0[] = "tarpit";
/* argv1 is variable IP */
#define tarpit_argv2last NULL

/* input line buffer for web request */

#define MAXLINE 512
int linelen = 0;
char linebuf[MAXLINE+1];

/*
** log an error message and exit
*/
void log_error_die (const char *mess)
{
    /* fprintf (stderr, "tard: %s\n", mess); /* DEBUG */
    syslog (LOG_ERR, "%s", mess);
    exit (1);
}

/*
** log an error message plus errno value and exit
*/
void log_errno_die (const char *mess)
{
    /* fprintf (stderr, "tard: %s: %s\n", mess, strerror (errno)); /* DEBUG */
    syslog (LOG_ERR, "%s: %m", mess);
    exit (1);
}
```

```
/*
** read next line from stdin into static buffer; will accept CR, LF, or
** CRLF as line delimiter; if line too long extra part discarded; using
** C stdlib because it's already optimized for speed and I don't need to
** do anything fancy.
*/
read_line ()
{
    static int eat_lf = 0;    /* true means: if next char is LF, it belongs
                                 to prev line, discard it */
    int nextchar;             /* input lookahead */

    linelen = 0;
    nextchar = getchar();             /* first char */

    if (eat_lf && nextchar == '\n') {
        /* this is LF from prev line, discard it */
        nextchar = getchar();
    }

    while (nextchar != EOF && nextchar != '\r' && nextchar != '\n') {
        /* end of line not found */
        if (linelen < MAXLINE) {
            /* add to buffer only if not full */
            linebuf[linelen++] = nextchar;
        }
        nextchar = getchar();         /* next char */
    }
    linebuf[linelen] = '\0';          /* null-term */

    /* if CR or LF here, discard just by doing nothing; if CR, set flag
       that LF probably follows */

    eat_lf = nextchar == '\r';
}

/*
** main program; no command line args
*/
int main (int argc, char *argv[])
{
    int virus_found = 1;
    struct sockaddr_in peer_addr;
    socklen_t peer_addr_len = sizeof (peer_addr);
                              /* for getpeername call */
    char *peer_addr_str;      /* for inet_ntoa call */
    char *child_argv[TARPIT_ARGC+1];
                              /* for execv call */

    /* on entry stdin/stdout/stderr should refer to open TCP socket */

    close (2);                        /* close stderr right away to prevent
                                         info leaks */
    openlog ("tard", LOG_CONS | LOG_PID, LOG_USER);
                                      /* open syslog */

    /********** read and test HTTP request **********/

    /* if GET line matches and no headers, or if GET matches, Host:
       matches, and no other headers, it's the virus */

    read_line();                      /* get request line (GET/HEAD) */
    if (linelen == 0) {                       /* discard 0..1 leading blank lines */
        read_line();
    }

    if (0 != strcmp (linebuf, virus_line1)) {
        virus_found = 0;              /* no match, not virus */
    }

    if (linelen != 0) {                       /* read first header line */
        read_line();
    }

    if (linelen != 0 && 0 != strcmp (linebuf, virus_line2)) {
        virus_found = 0;              /* present & no match, not virus */
    }

    if (linelen != 0) {                       /* try second header line */
        read_line();
    }
```

```
    if (linelen != 0) {
        virus_found = 0;              /* there is a second line, not virus */
    }

    while (linelen != 0) {            /* read header lines until blank */
        read_line();
    }

    /* blank line marks end of header; if there's any body, we're going to
       close without reading it */

    /********** send browser redirect **********/

    write (1, redirect, sizeof (redirect) - 1);
                                      /* omit last byte of string which is
                                         \0; ignore errors; use low-level
                                         so I don't worry about flushing */

    /********** get peer IP **********/

    if (getpeername (0, (struct sockaddr *) &peer_addr, &peer_addr_len)) {
        log_errno_die ("getpeername failed");
    }

    if (peer_addr.sin_family != AF_INET) {
        log_error_die ("getpeername error: socket family is not IP");
    }

    peer_addr_str = inet_ntoa (peer_addr.sin_addr);

    /********** close socket **********/

    close (1);                        /* close output first */
    close (0);                        /* close input */

    /********** handle virus **********/

    if (virus_found) {

        syslog (LOG_NOTICE, "Adding %s to tarpit", peer_addr_str);

        /* exec program to put virus in tarpit */

        child_argv[0] = tarpit_argv0;
        child_argv[1] = peer_addr_str;
        child_argv[2] = tarpit_argv2last;
        execv (tarpit_cmd, child_argv);

        log_errno_die ("Exec tarpit failed");

    } else {
        syslog (LOG_INFO, "Redirected %s", peer_addr_str);
    }

    return 0;
}
```

Sample 4: Stickypit scripts

```
#!/bin/sh
#/usr/local/sbin/tarpit
#  Simple script to tarpit in the appropriate subchain
#  Called by tard
#
OCTET=`echo $1|awk -F. '{print $1}'`
/sbin/iptables -I CHAIN$OCTET 1 -j TARPIT -p tcp -m tcp -s $1



#!/bin/sh
#/usr/local/sbin/load-chains
#
# Load the 256 iptable sub-chains for 2-level scheme
#
COUNT=0


while [ $COUNT -lt 256 ]; do
    while true ; do
        iptables -A CHAIN$COUNT -j ACCEPT 2>&1 | grep '[a-zA-Z0-9]'
        if [ $? -ne 0 ]; then
            break;
        fi
        echo iptables -A CHAIN$COUNT -j ACCEPT failed, retrying...
    done
    while true ; do
        iptables -A INPUT -j CHAIN$COUNT -p tcp -m tcp -s ${COUNT}.0.0.0/8 \
            2>&1 | grep '[a-zA-Z0-9]'
        if [ $? -ne 0 ]; then
            break
        fi
        echo iptables -A INPUT -j CHAIN$COUNT rule... failed, retrying...
    done
    COUNT=$(($COUNT + 1))
Done
```