Protecting Web Servers from Security Holes in Server-Side Includes*

Jared Karro Jie Wang
Division of Computer Science
University of North Carolina at Greensboro
Greensboro, NC 27402, USA
Jared_Karro@uncg.edu, wang@uncg.edu

Abstract

This paper first investigates and analyzes security holes concerning the use of Server-Side Includes (SSI) in some of the most used Web server software packages. We show that, by exploiting features of SSI, one could seriously compromise Web server security. For example, we demonstrate how users can gain access to information they are not supposed to see, and how attackers can crash a Web server computer by having an HTML file execute a simple program. Such attacks can be made with no trace left behind. We have successfully carried out all the attacks described in this paper on dummy servers we set up for this investigation. We then suggest several practical security measures to prevent a Web server from such attacks.

1. Introduction

Computer and network security has been studied intensely for many years and a number of defense tools, such as various encryption techniques [15] and firewall technologies [4], have been developed and put in use. World Wide Web is a new Internet application; it offers new concepts and technologies for a large scale, on-line, multi-media repository of information, which has created many new ways for conducting business and has greatly influenced the daily life of many people. But the use of World Wide Web has also interjected a new set of security problems at the same time. For example, while new programming languages such as Java and JavaScript have made it easy to produce active Web documents, some of the new programming techniques introduced in these new languages have also created new security flaws. The study of Web security has therefore become an important issue. Previous research of Web security has been primarily focused on security flaws in Java [11], ActiveX [1], and other Web programming environments [10]. The integrity and security of Web server software packages, however, have not seemed to attract much attention. To make things worse, one could use the World Wide Web without having to use a Java or an ActiveX environment, but one is forced to use Web server software. Hence, the issue on the security of Web server software deserves serious consideration.

Web servers offer certain features that ordinary network servers do not allow. Such features are useful for users to create dynamic Web documentations. For instance, a Web server may provide certain mechanisms that allow users to automatically update the "date last modified" information of their Web pages. Such features, however, may also unknowingly give away certain valuable information to unauthorized Web users. Information about server computers obtained through Web servers may be used by a malicious user to harm the Web server in a number of ways, including crashing the Web server computer. Thus, investigating what type of information is obtainable from a Web server and what harm can be made by obtaining such information becomes interesting and important, particularly to universities, Internet service providers, and other organizations where a variety of users are allowed (and often are encouraged) to create their own home pages.

Most Web servers support Server-Side Includes (SSI), which can be used to execute CGI (Common Gateway Interface) scripts and echo valuable information to Web browsers. The ability to do SSI is independent from the Web browsers, making it particularly convenient in the multi-browser world we currently live in. Thus, SSI may serve as a powerful tool to obtain information and perform

*This work was supported in part by the University of North Carolina at Greensboro under grant 3-19612 and by NSF under grant CCR-9424164.

other tasks. In this paper, we will discuss flaws and areas of security concerns on SSI in many of the major Web server software packages used on the Internet. We show that, by exploiting features of SSI, a user can gain valuable information about a Web server computer as well as do harm to the server. For instance, a user can obtain information about the server computer configuration and what processes are currently in running on the server computer and by whom; he can kill a process that does not belong to him; and he can even crash the server computer. He can do so from a client computer without having access log into the actual server computer.

We investigate several Web server software packages for computers running UNIX, Windows 95, and Windows NT operating systems. We restrict our discussion to Web server software packages that are available either free of charge; or as free, fully functional, evaluation copies. We do not intend to cover all Web server software packages in this paper, nor do we intend to include all of the versions for the Web servers we do discuss. Certain security holes found in an early version of a particular Web server software package may have already been corrected in a newer version; but we found that a number of organizations, including our own, were/are still using an early version of certain Web servers, and so we feel that we are obligated to publishing our findings. Findings of this sort may also help server software developers and server administrators to identify some of the gray areas of security that might have been overlooked.

For convenience, in what follows, we will use "Web server" to denote "Web server software package" and "Web server computer" to denote the computer that runs the server software. We will use "Web viewer" to denote the individual who is viewing the Web pages and "Web user" to denote the individual who is creating the pages. The "Web facilitator" refers to the UNIX user ID under which all the Web server processes are executed. Depending on the Web server, the Web facilitator does not need to be an actual user on a UNIX system. In other words, the Web facilitator may simply be a fictitious user who does not own a directory nor have explicit rights (in fact, as we will discuss later, it may be safer to use a fictitious user rather than a real user). For simplicity, we sometimes omit the word "Web".

This paper is organized as follows. In Section 2, we show that one can exploit SSI loop holes to accomplish certain useful tasks on one hand, and to compromise the security of the underlying Web server on the other hand. Since most published works that deal with Web security (e.g., [14], [7], and [2]) have only briefly, if at all, mentioned the possible hazards of SSI, we demonstrate several examples of attacks that we have devised and carried out on dummy servers we set up for this investigation. These examples of attacks include methods to crash Web server computers, to kill other users' processes, and to have the Web server automatically send out e-mail messages to selected individuals. Such attacks can be made with no trace left behind. In Sections 3 and 4, we show and discuss how to obtain the necessary rights to execute those harmful SSI feature even though the user may be denied such rights by the default setting of SSI. We analyze the following Web server software packages on the UNIX platforms in Section 3: Apache, Stronghold, Netscape Fast-Track 3.0, Zeus Server; and we analyze the following software packages on the Windows platforms in Section 4: Netscape Fast-Track 2.0, Netscape Enterprise (NT only), and O'reilly WebSite Professional (NT only). The Apache server, according to a recent survey [12], is the most used Web server under UNIX platform. In Section 5 we offer several security measures, based on our investigation and analysis of security holes in SSI, to protect Web servers from the attacks we describe in this paper. We hope that this paper will serve as a useful tool to help the reader in choosing a suitable Web server and Web environment, as well as a guide for securing his Web server from attacks of these types.

2. SSI and security holes

We will first show in this section how SSI can be used to perform certain useful things when users are denied access to CGI scripts. We will then show how SSI can be used to compromise the security of the underlying Web server. We assume that the user has rights to execute the exec and include directives. This opens a door for users to create their own scripts (programs), which could be used to do useful things such as setting up counters, or could be used to do harmful things such as crashing the server computer.

2.1. Useful Exploitation of Loop Holes

Depending on the server environment, it may become necessary for the users to exploit loop holes in SSI to accomplish certain useful tasks. For instance, the Web server at our university does not allow users to execute their own CGI scripts, and it only provides a CGI script for mailmerge. This means that if we want to set up a counter of visits in our pages, we have to use a service provided by other Web servers outside of the campus. This will slow down the loading of our pages and we cannot guarantee that the counter will always work.

To get around this problem, we wrote a C++ program to look at a log file, count the number of hits on the given

page periodically, and output the result with a graphic image. The program needs to know the current time for the counting purpose, which is done by executing the date system call. The CGI variable $REMOTE_HOST or $REMOTE_ADDR provides information on which location the viewer was hitting the page.

2.2. Examples of attacks

We describe five possible attacks using SSI in this subsection, which have all been carried out successfully in our experiments.

Example 1: Crashing the Web server computer. A malicious user can crash a Web server computer by running a CGI script that runs the following simple C program.

```
for (;;) malloc(1024);
```

Using an SSI line to execute this C program will use up all available RAM of the Web server computer in a matter of seconds, yielding a system crash. Moreover, while the Web server computer is completely unfunctional, it is still able to respond to pings. This means that common administrative programs which ping network servers periodically to ensure that servers are up and running will not notice anything wrong. However, the computer will not be able to allocate memory to create any new process. Hence, all connection requests will be refused. Any connections that were already established will lock. As a consequence, unsaved data may be lost.

Example 2: Starting a daemon process. Users can write an SSI file to start a daemon process (e.g., another server) from the Web server computer. For instance, although our Web server does not allow user logins on the server, we were able to start an ftp server on the server computer. It is interesting to note that the Web server could not tell who actually caused the ftp server to start. The Web server thought that the Web facilitator (which did not even exist) started the process. This means that user activities are covered (or shielded) from the Web server. The Web server log files, which we will discuss later in Section 2.3, are also of no use. Attackers could possibly write their own server program so that it contained flaws, which could then be used to further exploit the server. If the attacker made the program appear to be a legitimate process that normally runs on the server (by making the name look similar to httpd or some other common process), the system administrator may not notice the attackers' program. If the program is served as a back door into the system, the attacker could get control the entire system. The ftp server that we started, for instance, did not require logins; thus, the ftp client had gained access equivalent to the Web facilitator.

Example 3: Killing a process. An attacker can kill a process generated by the Web facilitator. This means that the attacker could kill all of the other children processes started by the Web facilitator. Even if the original process were to re-generate the children process (as in the case of the most Web servers), a program could be written to continuously kill these processes as soon as they are created. Thus, all Web processes created to answer requests would be killed before requests could be answered. Since the processes generated by the Web facilitator can be killed by the attacker, the attacker can kill any process on the Web server computer if the root of the Web server is also the Web facilitator.

Example 4: AutoMail. Users can write a program that would in turn connect to other computers and do certain things. These activities would appear to be generated by the Web server rather than by the users. For instance, the user could have the Web server send out fake e-mail to random recipients or to particular persons (e.g., the chancellor of a university or a state senator). Since the Web server creates the message, there would be no information in the e-mail headers showing that it was the user's computer that was responsible for generating the message.

Example 5: Probing Web server computers. We describe below steps that could be used to probe a Web server computer. The idea is consistent for either the UNIX or the Windows operating system, but the syntax and the process is slightly different. As a result, knowing what operating system the Web server is running is the first issue that must be resolved. This can be done with the following SSI lines:

```
<!--#echo var="SERVER_SOFTWARE"-->
<!--#echo var="PATH"-->
```

Executing an SSI file that contains these two lines will display the Web server software and the path for the Web facilitator. This should give the user enough information to figure out which operating system the Web server is running. In some cases, only one of these may be necessary.

In an UNIX system, one can use a series of the SSI lines below to probe a system. For a Windows 95/NT system, one must use a batch file containting the equivalent DOS commands.

```
<!--#exec cmd="pwd"-->
```

```
<!--#exec cmd="ls <directory>
                  <switches>"-->
<!--#exec cmd="rm <file>"-->
<!--#exec cmd="more <file>"-->
```

2.3. Logging of SSI

All of the Web servers we have tested appear to generate similar logs, and so we will discuss the logging of SSI in a general framework. We will show how to make the attacks we described above with no trace left behind.

If the execution of an SSI file does not generate an error, there will be no special entry in the Web server's log file. As usual, the Web document that was requested is entered into the access log, but there is no mentioning of SSI execution. If someone tries to execute an SSI file, but the execution is denied by the server for the directory containing the SSI file, then the URL will be logged as illegally attempting to perform execution.

The lack of logging successful SSI execution means that if someone runs a program that crashes the server computer, the server administrator would be unable to see exactly what happened or who did it. Chances of catching the perpetrator would be reduced even more if the attacker had the program wait for sufficiently long time before executing the deadly functions. In this way, although the access of the SSI file that runs the attacker's program was entered into the access log, this entry might have already been deleted after the actual attack was initiated. The attacker could also have the program altered the SSI file after the file was parsed, thus removing the line that dealt the deadly blow.

3. Web servers on UNIX platforms

We analyze the following Web servers for UNIX operating systems: Apache 1.2.0, Stronghold 2.0, Netscape Fast-Track 3.0, and Zeus Server. The Apache server, according to a recent survey by Netcraft [12], was the most used Web server. It was used by almost 44% of about 1.2 million sites that responded to the Netcraft survey. Stronghold is an enhanced commercial version of Apache, which, according to Netcraft [12], is the second most popular commercial Web server. We will discuss how a user may obtain rights to execute SSI directives exec and include, even though the user may be denied such rights by the default setting of the Web server. Once the user obtains such rights, then the attacks we described in Section 2 can be carried out.

3.1. Apache 1.2.0

The Apache server is the result of the joint work of many independent programmers, which is based on the NCSA Web server. Apache 1.2.0 has two basic configuration files: access.conf and httpd.conf. Some versions of Apache also have a srm.conf file, but in recent versions of Apache the srm.conf file can be included at the end of httpd.conf. The httpd.conf file stores the basic Web server configuration information, including things such as the port to listen to, the server administrator, the user directory path, the server namespace, and the AccessFileName variable (which will be described later). The access.conf file tells the server what type of services is allowed and where the requests for these services can be made from.

We find that with Apache, users are allowed to override the settings that have been set by the server administrator. This can be done by creating a file named .htaccess in the user's directory, which is, by default, the access override file that does the overriding job. The administrator can change the file name .htaccess to some other file names by editing the AccessFileName entry in the httld.conf file. But the server administrator cannot simply remove this line to prevent the user from using access overriding file because if this line is removed, or the filename is actually a directory followed by a file (i.e. <directory>/<file>), then the default, .htaccess will be used. While overriding the server settings for the server's root directory is turned off by default, access overriding is allowed for user directories. This means that even if the administrator does not intend for users to parse or execute CGI scripts, users can still give themselves access rights to do so, unless specifically denied.

As a default setting, Apache does not allow users to parse SSI files. This feature may seem secure enough to prevent users from running the attacks we described in Section 2. However, if overriding settings is permitted, the user can activate the Web server to parse SSI files using one of the following Apache configurations: ExecCGI, Includes, and IncludesNOEXEC. The user can also have the Web server parse any extensions they desire. The following is an example of a .htaccess file:

```
Options Indexes FollowSymLinks Includes
AddType text/x-server-parsed-html .html
AddType text/x-server-parsed-html .htm
```

The first line turns on all SSI features so that SSI files can be parsed and executed. The other three lines tell the Web server to parse files with extensions .cgi, .html, and .htm. Files with other types of extensions could also be added if desired.

When the Web server finds a .htaccess file (such as the one described above) in any of the parent directories of the directory where the requested HTML file is located, the server will use the settings in that file unless the server is explicitly told to ignore the access override file. By adding "AllowOverride NONE" to the section of the access.conf file for the directory where overriding is not to be allowed, the administrator removes the ability for users to override the desired settings. For more information about the options available for this feature, see the Apache documentation [3].

Apache 1.2.0 also comes with suEXEC [16], which is a CGI wrapper. This feature is designed to change the UID of a CGI script to match that of the user who owns the script. But suEXEC is not part of the standard installation. (Installing suEXEC is not an easy process.)

Apache 1.2.0 also has features that allow the server administrator to set bounds on server resources, such as the CPU time and RAM, for CGI programs. The directives RLimitCPU, RLimitMEM, and RLimitPROC all have default values of whatever the underlying operating system defaults are. Together these directives should be able to prevent against attacks such as complete memory allocation. If used independently, however, the user could get around the limitation by exploiting the other unlimited methods. For example, if the server administrator merely sets the server so that the number of processes could not exceed five, the attacker could make four processes which all have high priority so that they would use up all or most of the CPU time. Since these features do not appear in versions of Apache prior to 1.2.0, it is highly recommended to upgrade to the latest version of Apache (version 1.3 is currently in its alpha testing phase).

3.2. Stronghold 2.0

Stronghold is an enhanced commercial version of Apache. The configuration on the version we tested was identical to that of Apache. The new release of Stronghold will have a GUI configuration. Stronghold is designed to act as a secure server. It has built-in capabilities for generating certificates and doing SSL encryption, and comes with a Server Certificate. Since it is a commercial product, the technical support is probably better than that of Apache. An NT version of Stronghold 2.0 is now in the beta stage.

3.3. Netscape Fast-Track v3.0

The Netscape Fast-Track Web server, by default, does not parse any SSI files. Once SSI parsing is enabled, Netscape servers would only parse files with the .shtml extension. The server can be configured to parse all files. The servers can also be configured to parse files according to directories such that they can either allow or deny execution from within an SSI line. If execution is not allowed, then the user cannot use the exec or include directives. The server administrator can set which file extensions and which directories the server will parse. However, by default, making changes to these settings will effect the entire Web server.

When setting up the server, the server administrator can specify the Web facilitator to be a user that does not physically exist. Root, or whoever originally starts the server, will execute the original process; and all other processes will be run by the Web facilitator. This prevents scripts (programs) from accidentally having access to files they are not supposed to, since scripts will have the same access to files as the Web facilitator. If the Web facilitator is a fictitious user, scripts will only be permitted to write or delete files that any user on the system can.

3.4. Zeus server

Zeus Server is a product of a British company. A fully functional evaluation copy can be obtained free of charge for a 30-day trial from their Web site[17]. By default, the server does not parse any SSI files. The administrator can turn on SSI parsing for the root directories and user directories separately. Once SSI parsing is turned on, Zeus server only parses files with extension .shtml. Turning on parsing does not allow CGI execution. CGI execution is turned on via another option, and once again it can be turned on for the root and user directories separately.

Once CGI execution is permitted, CGI programs must be executed either directly (specifically calling the .cgi file in the URL) or via the cgi parameter with the exec directive. If the user tries to use the cmd parameter, he will receive an error message in his HTML file. Another nice feature of Zeus Server is that it has something called Sandbox CGI. This allows the server administrator to set bounds on the CPU time and RAM that CGI programs have access to. This prevents users from purposely, or accidentally, interfering with the normal process of the server. Unfortunately, not all operating systems support this feature. Hence, the server administrator may want to test this feature on his server computer to determine whether he can safely rely on this feature.

One drawback of this server is that the Web facilitator must actually exist on the server. This means that the server administrator should generate a user with minimal rights to run the server. As before, if the Web server was run by the root, CGI programs would have full access to the server, which could lead to tragic results.

4. Web servers on windows platforms

The Web servers on the Windows platforms have slightly different features and options from the Web servers on the UNIX platforms. SSI is also handled differently in Windows 95 and in Windows NT.

Windows 95 and Windows NT are two different operating systems, but the Web servers on these platforms act in similar ways. We have tested the following three different Web servers: Netscape Fast-Track 2.0 (Windows 95 and NT versions), Netscape Enterprise 3.0 (NT only), and O'reilly WebSite Professional (NT only). Due to the extreme similarities in functionality of the different Web servers, this section is divided into functionality subsections instead of product subsections.

4.1. Capabilities of SSI

The Netscape Web servers for Windows 95 and NT all have default configurations that forbid users to run SSI. These servers can be configured to parse any file extensions that are desired. By default, once parsing is enabled, Netscape servers will only parse files with the .shtml extension. The servers can be configured to parse certain extensions in one directory and other extensions in another directory. WebSite Professional parses SSI files with the .html-ssi extension. Like the Netscape servers, extensions can be manually added or removed as desired.

SSI is divided into two different categories under all of these Web servers. The first category only allows SSI directives that do not execute user programs. This means that users are restricted to such SSI directives as echo and flastmod. The second category allows SSI directives that can execute user programs and CGI scripts.[1] Thus, this category is potentially more dangerous.

Users can write CGI scripts to do virtually anything they desire under the Windows 95 and Windows NT platforms. If a CGI script is made as a batch file, then it can do anything a person can normally do in a DOS window, including viewing and deleting files, even if they lie outside the normal scope of the Web server. With a series of SSI lines (or CGI scripts), users could scan the entire system on the Web server computer, and delete or view any file of others. They can do so without even knowing anything about how the system is configured (e.g. drives, directory structure). See Example 5 in Section 2 for an example of the steps an attacker could use to probe a system under the Windows platforms.

As shown in Example 1 in Section 2, a malicious user could also crash a Windows 95 Web server computer by running a simple C program: for(;;) malloc(1024). Executing this program or a similar one in DOS-CGI or WinCGI will use up all available RAM of a Windows 95 Web server computer in a few seconds, yielding a system crash. No commands (such as soft reboot) can bring the Web server up and running, and so manual reboot is required to bring the server computer back to life.

If the same program is executed on a Windows NT Web server, however, the effect is much different. Windows NT employs a robust memory management scheme. It automatically utilizes unoccupied hard disk space as virtual memory. Thus, although running the C program described above will slow down the Web server computer, it does not bring it to a complete halt until it uses up all memory and available hard disk space. We have tested this program on a PC with 80MB RAM and a 90MHZ CPU running Windows NT, and we found that after about half an hour the program was still running and had allocated 184MB of virtual memory. Although the computer was noticeably slower, it was still able to respond to new commands and open new processes. In other words, the Web server, and more importantly the Web server computer, was still functional.

4.2. Directory access

Access rights of parent and children directories are another possible point of security holes. In all of the Windows Web servers we have mentioned, if the parent directory is set to have greater rights than a child directory, the child directory will inherit the parent's rights. This inheritance of rights holds true even if the child's rights are explicitly set to be lower than its parent's. Although this structure does not typically produce security problems if virtual directories are used, it may create a security problem if a virtual directory is accessible through its parent directory. If the URL uses the virtual name, then the user will be restricted to the rights assigned to that virtual directory. If a user gains access through the directory structure, he will inherit the greatest rights of the ancestor directories.

Now for the good part, Windows 95 and Windows NT servers can configure certain directories as CGI directories and allow users to execute CGI scripts only in these designated directories. Moreover, the server can be configured to only allow certain types of CGI. In fact, the administrator must manually list the directories they want to allow CGI access from within. Therefore, the Web server can prevent users from generating CGI scripts by forbidding users to gain write access to these directories. However, once users gain rights to create their own scripts, there is no way of stopping them from running harmful

[1] There are several different types of CGI: DOS (or shell), standard (Perl or other scripting languages), and Windows.

CGI scripts.

5. Security solutions

Garfinkel and Spafford [7] suggest that it might be more secure to run a Web server on a Macintosh computer. Macintosh computers have no command-line interpreter, so break-ins may be more difficult. We have not tested this concept, but it seems logical. However, it may be possible to execute commands if the Macintosh Web server has a PC board, which may be used to comprise the advantage of using a Macintosh computer as a Web server.

In this section, we will focus on Web servers running UNIX, Windows 95, and Windows NT operating systems. We form a number of security measures, based on our investigation and analysis of security holes in SSI, to protect Web servers from the attacks we described earlier in the paper. We first provide a list of general security measures that are independent of the operating system used by the Web server. Following these recommendations, we further provide a number of more specific solutions according to which operating system the Web server is running.

5.1. General security measures

1. Keep the server software up to date. New versions of server software and patches are routinely released to fix security holes or other bugs found in early versions.

2. The server administrator should carefully review the needs of his users and the security risks involved with those needs. Once the server administrator has decided what types of access are allowed and to whom, he will need to configure his server to allow only the access as specified. If the server administrator discovers that his Web server does not support the type of configuration he desires, he should consider using a different Web server.

3. Once the server configuration is determined, the system administrator should thoroughly test the Web server to confirm that restrictions have been properly set and his clients have been confined in an appropriate environment to work within. In other words, clients should be given access rights for doing the things they need do, but nothing more.

4. As suggested in [2], consider not allowing SSI at all. Most things that can be done via SSI, can also be done with the use of scripts and crontabs. For example, we can use utilities such as sed and awk to create headers and footers. For a good source on how to use these tools to automate the creation of headers and footers, the reader is referred to [6].

5. Run the Web server on a computer designated to running only the Web server. Running other daemons on the same server may compromise the Web server if one of these daemons are compromised.

6. Do not store critical Web files and user Web files on the same server [14]. By separating this type of information, we can restrict the average user from having access to certain critical Web files or capabilities, while allowing the official pages to have access to these features and files.

7. Keep any scripts that come with the Web server up-to-date. Software vendors often release new versions of these scripts to fix known bugs or security holes.

5.2. Solutions for UNIX web servers

The following are our suggestions for Web servers running in an UNIX environment. These recommendations are in addition to, and not in place of, those listed in Section 5.1.

1. Do not run the Web server as root. While the CERT advisory 95-04 [5] suggests to run the Web server as nobody, Ruben et al. [14] suggest to run it as a genuine and unique user with membership in a unique group. This way, there is no need to worry about the Web server sharing rights with other programs which are running as nobody.

2. If SSI is allowed, all users must be restricted to SSI directives that cannot execute programs. Users can still easily include dynamic features such as "date last modified" and where the viewer is coming from, but they would not be able to execute commands or scripts from within directives, and so they could not use the attacks we have described. We can restrict users from using executable SSI directives in NCSA-derived Web servers, such as Apache and Stronghold, by putting the following line in the server options:

   ```
   Options IncludesNOEXEC
   ```

   In Web servers with a Web-based configuration, such as those by Netscape and Zeus, make sure that the "Without exec tag" is chosen instead of "With exec tag".

3. If an NCSA-derived server is used, disable the ability of users to override the server settings with the use of their own .htaccess file. This may not allow users to create password protected directories, but it will be well worth it. This can be accomplished by putting the following line in the user's directory entry in the Web server configuration file:

   ```
   AllowOverride None
   ```

   This will prevent users from overriding any settings. If really necessary, we could replace None with a different directory directive, but do not put the Options directive on this line, as this would allow users to turn back on all SSIs and use them.

4. Disable any unnecessary services [7]. Commenting out their lines in inetd.conf will eliminate some of these services. Other services require that their "rc" files be eliminated.

5.3. solutions for windows 95/NT Web servers

The following are our suggestions for Web servers running in a Windows environment. These recommendations are in addition to, and not in place of, those listed in Section 5.1.

1. Allow CGI to be run from within only secure and well-monitored directories. These directories are the only ones in which execution of scripts should be allowed.

2. Only trusted users should be allowed to create or modify CGI scripts lying within executable directories. The Web administrator should review these scripts after any alterations are done. The focus of the review should be two-fold. First, make sure the script itself does not initially compromise security. Second, make sure that any Web viewer or Web user arguments are thoroughly checked for correctness. If this is not done, the scripts could be used to achieve undesired results and possibly execute one of the attacks discussed in this paper.

3. Make sure no interpreters or shells are in any of the Web server's cgi-bin directories [7].

4. In Windows NT, use the access list to deny access to all ports except desired ones; disabling IP services requires manual manipulation of the registery file, which can be dangerous in itself [7]. Not doing this could open a computer to attack such as the Windows OOB Bug [8].

Final Remarks. After seeing an early version of this paper, the Web server administrator of our university has taken certain necessary security measures we suggeseted in this paper, to improve the Web server security of our university. However, this should in no way be interpreted as an invitation to try some of the attacks we described in this paper on our Web servers; and we humbly beseech our gentle readers: Please Don't.

References

[1] ActiveX and Web Security Loopholes. E-mail received from *TipWorld*, 1997. (TipWorld is located at `http://www.tipworld.com`. This particular issue can also be found at URL `http://www.uncg.edu/~jqkarro/Netscape/Security/ActiveX.html`.)

[2] Anonymous. *Maximum Security*. Sams.net Publishing, 1997.

[3] Apache Core Features. URL `http://www.apache.org/docs/mod/core.html`, 1997.

[4] W. Cheswick and S. Bellovin. *Firewalls and Internet Security*, Addision-Wesley, 1994.

[5] CERT advisory 95-04: NCSA HTTP Daemon for UNIX
vulnerability. URL `ftp://info.cert.org/pub/cert_advisories/CA-95:04.NCSA.http.daemon.for.unix.vulnerability`, 1995.

[6] D. Dougherty. *SED and AWK*. O'Reilly & Associates Inc., 1992.

[7] S. Garfinkel, and G. Spafford. *Web Security and Commerce*. O'Reilly & Associates Inc., 1997.

[8] S. Jenkins. Windows OOB Bug. URL `http://www.winfiles.com/bugs/oob.html`, 1998.

[9] J. Karro. A C++ counter program. URL `http://www.uncg.edu/~jqkarro/counterCode.C`, 1997.

[10] B. McWilliams. Can Your Webmaster Watch You? URL `http://www.pcworld.com/cgi-bin/database/body.pl?ID=970827180448`, 1997.

[11] G. McGraw, and E. Felton. *Java Security: Hostile Applets, Holes, and Antidotes*. Wiley Computer Publishing, 1997.

[12] The Netcraft Web Server Survey. URL `http://www.netcraft.com/survey/`, 1997.

[13] N. Neulinger. CGIWrap. URL `http://wwwcgi.umr.edu/~cgiwrap/`, 1997.

[14] A. Rubin, D. Geer, and M. Ranum. *Web Security Sourcebook: A Complete Guide to Web Security Threats and Solutions*. John Wiley & Sons, Inc., 1997.

[15] B. Schneier. *Applied Cryptography—Protocols, Algorithms, and Source Code in C*, 2nd edition, John Wiley and Sons, 1996.

[16] Apache suEXEC Support. URL `http://www.apache.org/docs/suexec.html`, 1997.

[17] Zeus Technology. URL `http://www.zeus.co.uk/`, 1997.