Web Application Hacking

SensePost Research

SensePost Information Security

research@sensepost.com

haroon@sensepost.com

+27 (0)12 667 4737

+27 (0)83 786 6637

P.O.Box 10692

Centurion

0046

South Africa

ABSTRACT

The security world has spent the last decade focusing on protecting networks from traditional security attack vectors. Network Firewalls and related filtering solutions today have reached levels sophisticated enough to allow drag and drop enforcement of security policies. The goal posts however have shifted once more with the wide spread deployment of custom and COTS web based applications.

These web applications can not be protected by the solutions that security professionals have become accustomed to, and in many cases need to be re-written from the ground up with security in mind. This talk will highlight some of the attack vectors in this new security playground and discuss potential solutions and work arounds.

Web Application Hacking

1. Introduction

The information security world has spent the bulk of its lifespan developing and updating firewalling technologies, to restrict access to critical servers and networks. The last 2 years however has seen a dramatic increase in the deployment of web-based applications. This application space has therefore become the new playground for attackers providing access to potentially sensitive information and possible inroads into internal private networks.

A number of explanations exist for the glut of trivially exploitable web applications that exist on the Internet today. One of the most plausible is that developing web enabled applications is far simpler today than application development ever has been. Current generation development environments allow an end user to publish information directly from the company database to a web site with a few mouse-clicks and (maybe) a few lines of cut and paste code. This has resulted in a large supply of "developers" with little or no understanding of secure coding practices.

In addition to this we can add the problem we refer to as "following the blind". The Internet has made the dissemination of information incredibly easy. While it took tremendous effort to publish ones work (in the past) it is nowadays as simple as FTP'ing your document to any one of the countless free hosting providers that exist. With all the benefit that comes with this free-speech nirvana we now have the hidden downside of lost reliability. Using a code snippet from Kerrigan and Ritchie's "The C Programming Language" is not quite the same as a code snippet from http://www.geocities.com/~s00perK00l. This problem is compounded infinitely when even "trusted" sources of information are misleading (seen in section on SQL Query injection).

The rest of this paper will highlight some of the vectors used by attackers when attacking web applications and will discuss briefly the counter-measures available to developers.

2.1. Anatomy of a web application

A typical web application resembles the diagram below. The individual components illustrated may reside on the same machine or more often than not reside on multiple machines that exist at distinct locations within the company network.

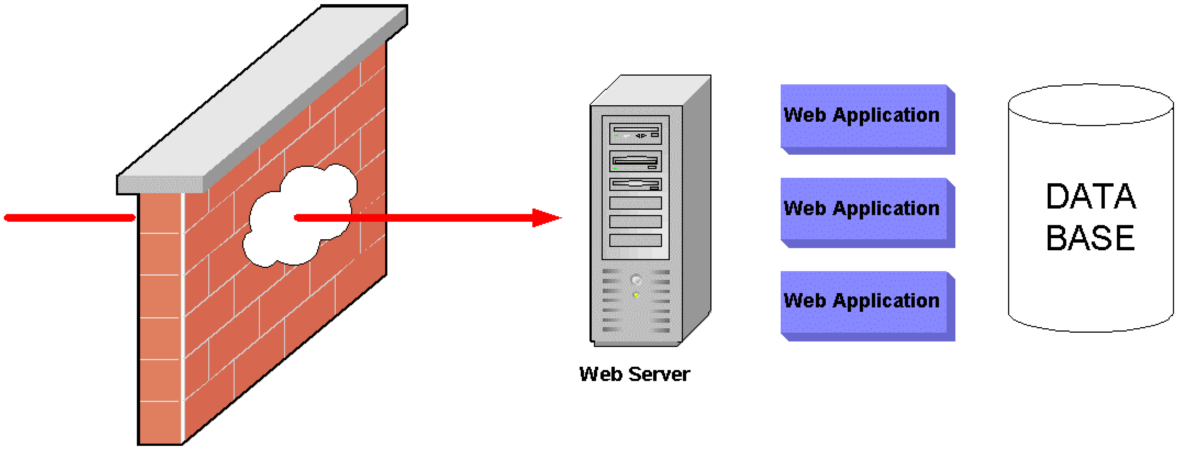


Figure 1. Typical architecture for a Web application

Once the decision has been made to permit HTTP access to the web-server, very little can be done to prevent an attacker from attempting to "hack" the web application that resides on it. Web application attacks are conducted totally over HTTP and are seen as acceptable traffic to the firewall / filter. The introduction of SSLv2 does nothing to inhibit the attacks, and gives the attacker the added comfort of privacy on the wire.

2. Attack Vectors

For the purposes of this paper the following vectors of attack will be discussed.

- Information Gathering;
- Directory Traversal;
- Command Execution;
- Parameter Passing;
- Cookie Manipulation;
- State Tracking.

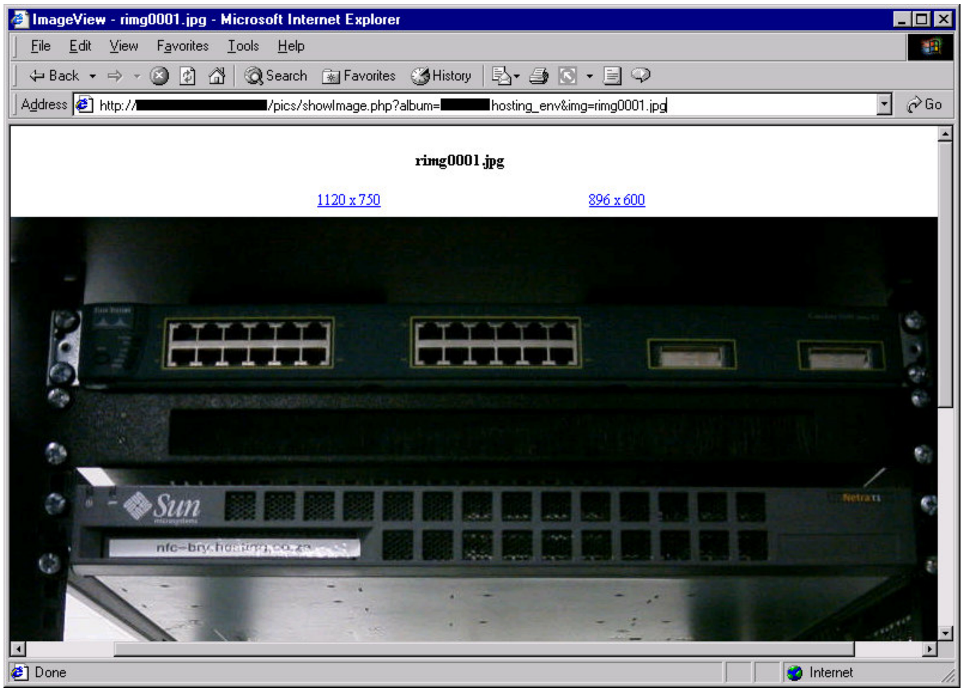
2.1. Information gathering

Information gathering is often overlooked as an attack vector but plagues most of the custom written web applications. The problem is often overlooked because one assumes that its implications are small. Web application hacking however is often achieved in little pieces as opposed to the instant root achieved when attacking conventional services. Information gathering is normally accomplished due a number of possible developer errors:

- Comments in client side code;
- Verbose error messages;
- Confusion over client side vs. Server side code.

HTML as its name suggests is merely a mark-up language. Developers often embed comments in client side code that is of invaluable use to potential attackers. Even innocuous details like the authors name become potentially harmful, providing attackers with a possible username for the system. This is a trivial matter to rectify and normally requires just awareness on the part of web page designers.

The second error that leads very often to information leakage occurs due to error messages within the application. Potential attackers often feed known “unfriendly” characters into an application in an attempt to force the application to fail. Many systems return verbose error messages that reveal information that could greatly assist a potential attacker. Verbose error messages provide application developers with an effective means of debugging applications but should be replaced with more generic messages when these systems are rolled into production. Applications developed in PHP have long been a target of such attacks since the default configuration returned verbose error messages to the end user. The following web site uses PHP to create an online photo catalogue. A neatly designed front page allows the user to select the album and the picture number that the site visitor wishes to view.


ImageView - rimg0001.jpg - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Done
Internet
Sun

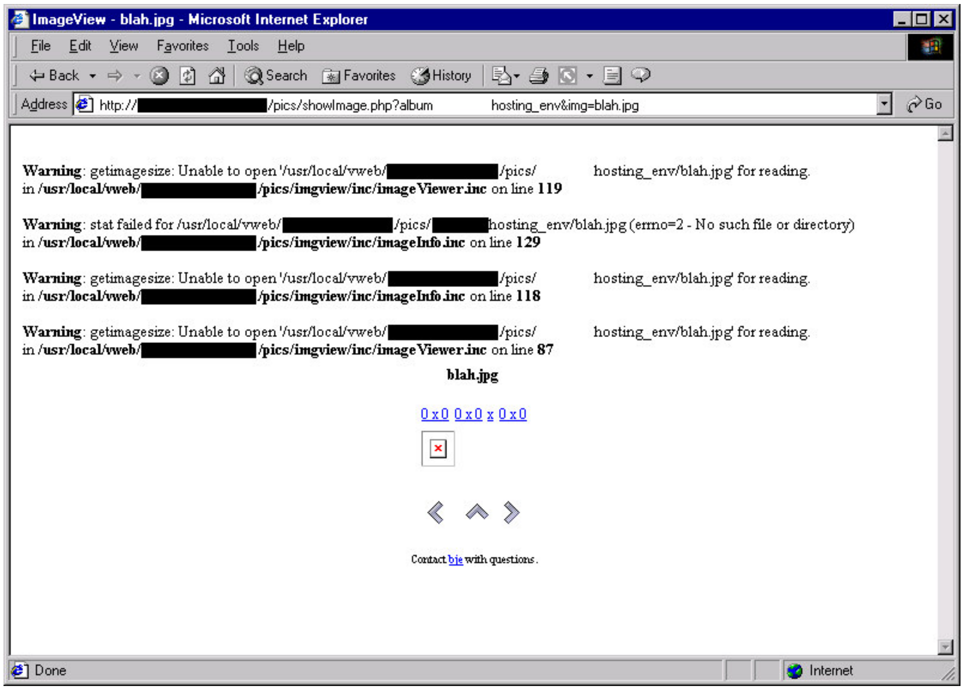
Address: http://[redacted]/pics/showImage.php?album=[redacted]hosting_env&img=rimg0001.jpg

rimg0001.jpg

1120 x 750   896 x 600

Figure 2. A PHP Photo Album

Entering an erroneous value for either one of these variables, returns a page as follows.


ImageView - blah.jpg - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Done
Internet

Address: http://[redacted]/pics/showImage.php?album hosting_env&img=blah.jpg

```
Warning: getimagesize: Unable to open '/usr/local/vweb/[redacted]/pics/        hosting_env/blah.jpg' for reading.
in /usr/local/vweb/[redacted]/pics/imgview/inc/imageViewer.inc on line 119

Warning: stat failed for /usr/local/vweb/[redacted]/pics/[redacted]hosting_env/blah.jpg (errno=2 - No such file or directory)
in /usr/local/vweb/[redacted]/pics/imgview/inc/imageInfo.inc on line 129

Warning: getimagesize: Unable to open '/usr/local/vweb/[redacted]/pics/        hosting_env/blah.jpg' for reading.
in /usr/local/vweb/[redacted]/pics/imgview/inc/imageInfo.inc on line 118

Warning: getimagesize: Unable to open '/usr/local/vweb/[redacted]/pics/        hosting_env/blah.jpg' for reading.
in /usr/local/vweb/[redacted]/pics/imgview/inc/imageViewer.inc on line 87
```

blah.jpg

0 x 0 0 x 0 x 0 x 0

Contact bje with questions.

Figure 3. PHP Photo Album failing "un-gracefully"

The attacker has now been treated to the real path of the web servers document root *(/usr/local/vwweb/XXXX/)*, and has been introduced to a few new directories to peruse *(/imgview/inc)*. In the case of the above application, the /inc directory was further found to be indexable, allowing us to peruse, and download scripts we were never intended to see (these sorts of scripts often contain connection strings and credentials that can be harvested).

The 3rd error often made is the confusion between server and client side code. Developers using certain technologies often assume that their code is not visible to the end user and are often mistaken in this regard. JavaScript and Java often lull developers into this false sense of security. JavaScript runs in the context of the user's browser, and users are therefore able to select whether to run, ignore or examine the script. Even compiled Java applets, are often mistaken for a server side technology. The applet may indeed reside on the server, but is downloaded and run on the user's machine. The user is once more able to simply download the applet to his machine, where freeware (and easily available) Java Decompilers are able to return the compiled Java bytecode to source code at the click of a button (Once more, we find numerous applets containing JDBC connection strings, usernames and passwords within the decompiled code)

2.2. Directory traversal

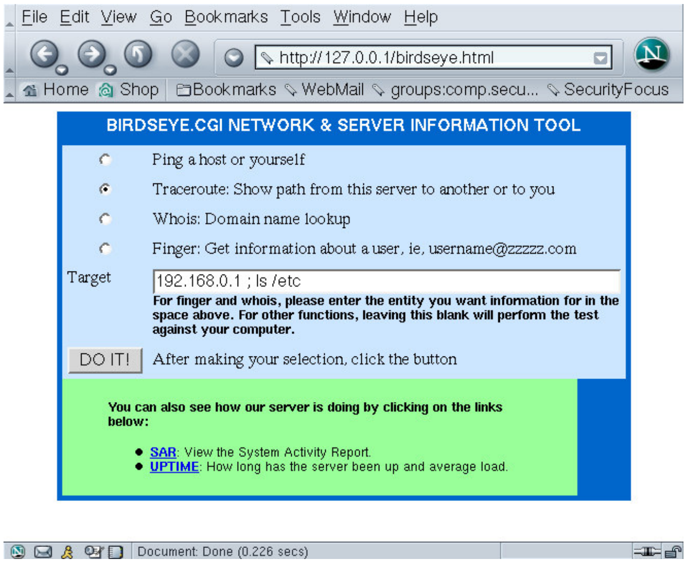
Directory traversal refers simply to the ability to cross from one directory to another. A user who chooses to run a file by typing /etc/init.d/apache while still being in his own home directory /home/users/mh is effectively doing directory traversal. This fairly innocuous almost mundane activity becomes a security risk when web applications permit directory traversal without being aware of the repercussions. A typical example of this can be found in a well known web based mathematical imaging program. This program generates images (graphs) based on user input and saves these images using random names to a directory on the server. A URL to the pictures path is then handed to the user. What follows is an example of such a URL : http://www.sensepost.com/webApplication/SENSEAPP?SENSEStoreID=SP88808199_324246989&SPStoreType=image/gif

The SENSEAPP application always expected the passed parameter to be an image like the one above (SP88808199_324246989). The application fails however to prevent directory traversal. By replacing the requested image, with the path ../../../../../../etc/passwd , one has simply to view the source code of the returned web page to view the contents of the coveted password file. What the application should have done, was to ensure that the ../../ ‘s were not acceptable input. (Many developers in an attempt to rectify this problem employ some sort of black-list, forbidding certain characters from being entered. This solution fails however when an attacker uses some form of encoding on his input for example UTF-8 / UNICODE. Developers should therefore employ white-listing instead. I.e. specifying that the field can only be numeric / etc.)

2.3. Command Execution

Many of the initial “slapped together” web applications were created to assist administrators perform their tasks from a friendly point and click interface. It is for this reason that it is not uncommon at all to find web applications that are front-ends as well as scripts and utilities that run on a single machine. As a potential attacker, these sorts of applications provide one with a world of possibilities. What follows is a typical example of a sys-admin web application. The page allows end users to perform basic network diagnostics using a web interface (ping/ traceroute/ whois/ finger). As a potential attacker one has only to guess at how the application is able to accept our input, and then perform an action on it. (The quickest, dirtiest, most typical way to do this would be to accept the user input, lets call it $input, and then perform an action as follows: `ping $input`.

The question one needs to ask is the possible implications of malicious characters being used as *$input*. The second time we run the script, we therefore replace the simple “IP address”, with “IP address ; Another Command” (*192.168.0.1 ; ls /etc)*


File Edit View Go Bookmarks Tools Window Help
http://127.0.0.1/birdseye.html
Home Shop Bookmarks WebMail groups:comp.secu... SecurityFocus
Document: Done (0.226 secs)

BIRDSEYE.CGI NETWORK & SERVER INFORMATION TOOL

○ Ping a host or yourself

● Traceroute: Show path from this server to another or to you

○ Whois: Domain name lookup

○ Finger: Get information about a user, ie, username@zzzzz.com

Target: 192.168.0.1 ; ls /etc

For finger and whois, please enter the entity you want information for in the space above. For other functions, leaving this blank will perform the test against your computer.

DO IT! After making your selection, click the button

You can also see how our server is doing by clicking on the links below:

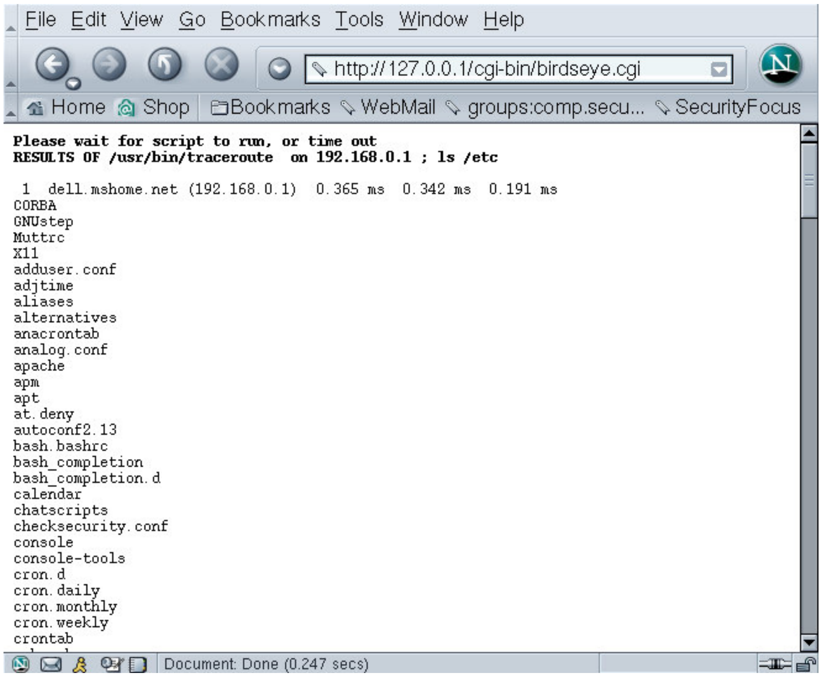
- SAR: View the System Activity Report.
- UPTIME: How long has the server been up and average load.

Figure 4. Web based network diagnostic utility

As can be understood from the explanation above, the back end of this script now looks as follows :

'ping 192.168.0.1 ; ls /etc'

This effectively tells the application to execute the ping / traceroute / finger as expected, but to then do a ls of the /etc directory (dir).


File Edit View Go Bookmarks Tools Window Help
http://127.0.0.1/cgi-bin/birdseye.cgi
Home Shop Bookmarks WebMail groups:comp.secu... SecurityFocus
Document: Done (0.247 secs)

```
Please wait for script to run, or time out
RESULTS OF /usr/bin/traceroute  on 192.168.0.1 ; ls /etc

 1  dell.mshome.net (192.168.0.1)  0.365 ms  0.342 ms  0.191 ms
CORBA
GNUstep
Muttrc
X11
adduser.conf
adjtime
aliases
alternatives
anacrontab
analog.conf
apache
apm
apt
at.deny
autoconf2.13
bash.bashrc
bash_completion
bash_completion.d
calendar
chatscripts
checksecurity.conf
console
console-tools
cron.d
cron.daily
cron.monthly
cron.weekly
crontab
```

Figure 5. The results of our modified traceroute.

The results are then happily displayed in the browser window. Once more, the simple solution would be for the developer to white list acceptable characters for input (thereby dis-allowing the possibility of potentially harmful characters like the & or &&). This problem is found even when the dreaded back-tick operators are avoided in favour of the “more secure” open file-handle approach. In cases like that, even simple mail-to forms using the format *open(MAIL, “|/sbin/sendmail $address”)* become vulnerable to trivial attacks. An enterprising attacker for example could enter his email address as haroon@sensepost.com < /etc/passwd.

The effective executed command is therefore /sbin/sendmail haroon@sensepost.com</etc/passwd, effectively mailing the attacker the systems password file. The problem above could be remedied by first passing the $address variable through a regular expression that accepts only alphabets, periods, hyphens and numerics.

```
# remove nastyness
$address =~ tr/a-zA-Z0-9\@\-\.//dc;
```

2.4. Query injection

Many web applications interface in some way to some sort of back end database. These applications typically make use of SQL, some sort of scripting language and a database connection. These sorts of applications become vulnerable when a user is able to alter the structure of the passed / generated SQL query before the query is passed to the back end database server. The most widely publicized version of this attack has to be the “one = one login”. In this case we typically have a web form that accepts a username and password through and then submits these details to a backend ASP script.

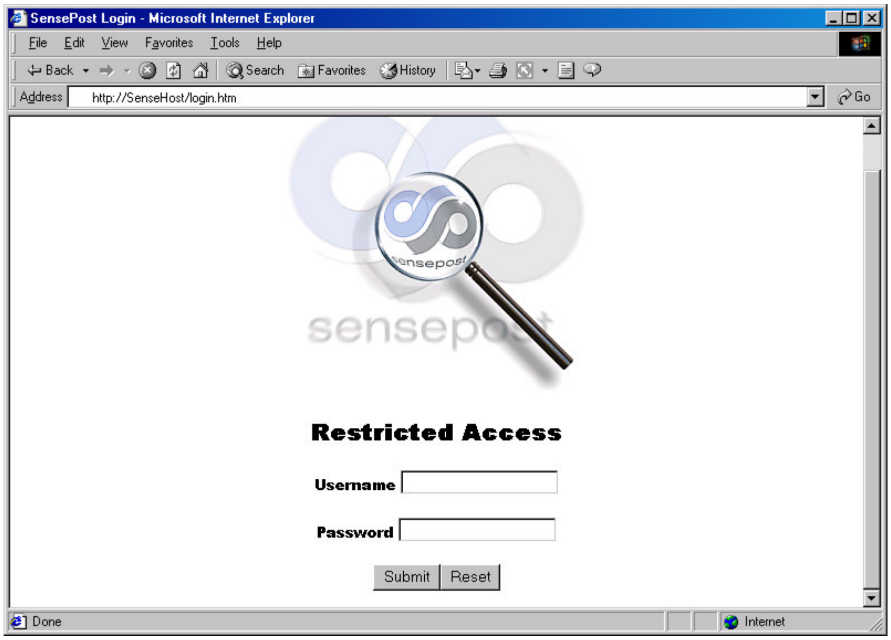

SensePost Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://SenseHost/login.htm
Go
sensepost
Restricted Access
Username
Password
Submit
Reset
Done
Internet

Figure 6. Typical Login Situation

The *username* and *password* from this form are then used to build the following pseudo SQL Query.

Select ALL_RECORDS from TABLE where UNAME=’username’ and PASS=’password’

The logic used in this query is that the query will return a non zero result if and only if a valid username and password combination is used. The next step in this authentication process is therefore to check if the result is non-zero. If it is, the user is logged in. The problem occurs when the user submits ‘ *OR 1=1*—as his username. The resultant pseudo query now reads :

Select ALL_RECORDS from TABLE where UNAME=’’ OR 1=1- ~~-’ and PASS=’password’~~

This query now returns all rows from the table, since even though we don’t have a valid username, 1 is always equal to 1. The addition of the -- characters (MS-SQL’s comment characters) ensure that the rest of the SQL string are just ignored. According to the logic of the login application, the results of the query are non-zero, and the application therefore assumes that a valid user has logged in. This “buggy” login code was once posted publicly, which resulted in thousands of developers worldwide copying and pasting the same buggy example. At a point in time, even shop.Microsoft.com was found to be vulnerable.

The problem is magnified manifold when you consider that some SQL servers make extensive use of stored procedures. MS_SQL for example contain hundreds of stored procedures, like *XP_CmdShell* which allows the execution of shell commands through SQL queries.

A recent test conducted by DigitalDefense (http://www.digitaldefense.net) found that 6 of 8 books on .NET used code samples susceptible to Query injection

2.5. Parameter Passing

This problem is abused by wily attackers manipulating the information passed to the back end system. This information is normally passed through HTML forms using GET's and POST's. Information posted in sometimes passed using <HIDDEN> fields, allowing the developers to pass certain information without the end user seeing it. Assuming that end users will not look under the hood however is a dangerous assumption to make. A well known credit card gateway for example operated under the following conditions. A user would select his product (or products) from a catalogue and would then eventually click on "PAY". This would then send an HTTP Post containing the following fields :

	ITEM	= SILK_TIE
	PRODUCT_ID	= 456546
<hidden>	PRICE	= R500 </hidden>

A user could simply shop to his hearts content, and then alter the hidden price field before clicking on submit. The back end system would then receive the request for the item, while deducting our new altered price from our credit card.

2.6. Cookie Manipulation

Cookies are often used within web applications as a means of keeping state, or as a method of tracking authentication. The cookie is however stored on the client machine and is thus susceptible to tampering before being passed to the server. In some cases, once authenticated, a user's name is simply stored in his cookie. This cookie is requested by the server at different points of the site, to ascertain who the logged on user is. A simple attack against a system like this is to log in as an unprivileged user, then change your cookie to contain the username of a user with higher privileges before continuing to surf the page.

2.7. State Tracking

Perhaps one of the biggest problems with using the web for applications is that HTTP is by its nature a stateless protocol. A web server ordinarily has no way of determining that the user who just requested to see his bank balance is the same user that authenticated successfully 20 seconds ago. Several kludges have therefore been built on top of it to force some method of tracking state. These range from the cookies we mentioned earlier, to URL state strings that are passed around with HTTP GETS, to hidden fields passed along with every page visited. HTTP Session ID's are used to the same effect. For example,. a randomly generated string is given to the user on successful login. This string is then requested at every page he visits to determine his identity.

This mechanism is often attacked due to insufficient randomness in the generation of the session-id. If this session id can be predicted or guessed, it then becomes possible for an attacker to simply walk into someone else's persona.

This system also fails when developers fail to track state consistently throughout the application. For example, User A logs in, obtains his session-id, but then requests to view the details of User B. The application checks to ensure that it is indeed User A, and checks to ensure that he has a current state-string, but fails to link the state string to just User A's account details. These kinds of bugs are harder to track down and harder to fix often resulting in major portions of the application being re-written.

3. Conclusion

The length of this paper does not begin to do justice to some of the points that need to be considered. It totally ignores some others. It is therefore by no means an authoritative work on the subject and should be seen only as an introduction to some of the concepts.

As developers we need to start paying more attention to the basics: sanitizing end user input, validating end user fields, ensuring that our applications fail gracefully etc.

As defenders we need to realise that that traditional defences are useless against these threats and that so far, we have only scratched the tip of the iceberg.