Global Information Assurance Certification Paper

Web Application Security for managers

GSEC Practical Assignment Version 1.4, option 1

Pierre de la Brassinne

August 24, 2002

1 Abstract

As a manager, part of your job is to plan and prioritize tasks and to control the work done. To plan and prioritize tasks, you need to have a global overview of all issues. To control the work done, you need a basic understanding of the subject.

The first part of the article intends to convince the reader that web application security matters. This may not be obvious to all managers. They sometimes believe that a firewall and the use of the SSL protocol are enough to secure a web application. The second part of the article surveys some of the potential problems and discusses solutions. We will look at issues like data manipulation, input validation, SQL query poisoning, session hijacking, and some others. The article ends with a summary of the recommendations.

2 Why does web application security matter?

It is still quite often heard from managers that there is no reason to invest more in security since the company is protected by a great firewall and the web application uses SSL. For sure, firewalls are great tools and help a lot to secure your applications. But they are only one element of the security chain. They protect the network. They filter network traffic. But they do not help to detect if an intrusion occurred and they can also be bypassed. Just like in a house, security involves more than locking doors: usually, there is an alarm system and a fire-resistant safe for example. Computer security is not different. You need a firewall to protect your network. But, you also need (among others), ways to detect intrusions (the alarm system), protection around the computer and around the application running on that computer (the safe).

On the one hand, firewalls can prevent hackers from accessing machines or services (programs running on a machine) without authorization. But, by definition, a web application must be accessible from the internet. Otherwise, nobody can use the application! Usually, system administrators restrict network access to a minimum: allowing only web traffic (http) to the web server trough the http port 80. When hackers encounter such a configuration, they do not have much choice. They can only use the http protocol to connect to your site, so they must use http. They only have access to your web server, so they can try to find weaknesses in your web server configuration or in the web server itself if it is not properly patched. They only have access to the applications running on the web server, so they can try to exploit weaknesses in the application. As you can guess, and as the article will show, there are potential weaknesses in web applications. Since hackers usually have no other choice, exploiting web applications vulnerabilities becomes more and more popular. This is the primary reason why investing in protecting web applications is important.

On the other hand, SSL is a protocol that encrypts the communication between the browser and the web server. When using SSL, somebody able to listen to the communication between the client (the browser) and the server cannot understand

the dialog because the channel is encrypted. But, on each end of channel, the information must be decrypted and is stored in clear at some point in time. It is therefore possible for a user of the client machine to edit specific data before it is sent to the server. As we will see later, there are ways to attack unprotected web applications using that technique. SSL does not help to prevent such attacks.

To summarize. It is very important to consider web application security because

- The firewall does not help. By definition, http traffic must be allowed to go through the firewall. Otherwise no web application is possible. (For more information on what firewalls cannot prevent, see [21])
- SSL does not help either.
- People don't usually think about securing web applications
- Usually web sites are protected by well configured firewalls and attacking your web application might be one of the only ways for a hacker to get in.

3 Overview of problems and their solutions

For those not too familiar with the http protocol, figures 1 & 2 (in the appendix) describe the way a browser interacts with a web server.

3.1 "Do not trust client-side data!" [2]

3.1.1 Data manipulation

Data is passed between the browser and the server either in the URL itself, in hidden fields contained in forms or in cookies. Since URLs and hidden fields values are hard-coded in the HTML source code, they are very easy to modify. An attacker simply uses the "display source" option of the browser and modifies the source. Modifying cookies is a bit more complex but still feasible. There are two types of cookies: persistent and non persistent. Persistent cookies are stored on disk and can be easily modified. Non persistent cookies are stored in memory only. This makes them a little harder to modify but there are tools available.

"Here is a real world example on a travel web site: the information stored on the client (in this case in a cookie) was the following: *lang=en-us; admin=no; y=1; time=10:30GMT*. No need to have a lot of hacking experience to imagine that if you replace *admin=no* with *admin=yes*, you will get a few more privileges on the system!"[1] p.68.

But data is not the only information coming from the browser. As part of the http protocol, a browser must send additional information (that the user does not see) to the web server in a so-called http header. One field of the http header is the "referer" field. This field contains the URL of the web page from which the request originated. To avoid the previously mentioned problem "(attackers saving web pages, modifying forms, and posting them off their own computer), some web sites check the 'referer' field in order to make sure the request originated from a page generated by them"[1] p.70. Even though every browser on the market fills the http header with the right values, there is no way to guarantee that the request originates from a browser. Anybody could write a program of its own, or use an http proxy, that connects to your web server, follows the http protocol rules but modifies the "referer" field value to make it look like it came from the original site. This example shows that you should

never trust any information coming from the client, including the http header. (See [16] or [1] p.70 for more information on the "referer" field).

Solutions to data manipulation

- Never store critical information (prices for example) on the client. Always store critical data on the server. Storing user preferences on the client is OK. Storing the 'admin=yes/no' as in the example above, is not acceptable. To be able to store data on the server, you need a "session mechanism" to associate data with a particular user of the application (not all users might have the same price for an article). The session mechanism is described later in the article.
- Never validate data client-side. Of course, for user-friendliness, you can validate data on the client. What I mean is that data should always be validated on the server, regardless of the checks done on the client. Remember that you should never trust anything that comes from the client.
- Always check input format and syntax (see [10] and [1] p.46-51). The safest approach is to « only allow valid input ». The approach « remove invalid input » is more dangerous, since you never know whether you have removed all vulnerabilities.
    - Allow only characters in a predefined character set. For example, the ';' character should be banned from any field value that will be inserted into a SQL query.
    - A very good practice is to "have all requests come in to a central location and leave from a central location. The problem is then easier to solve and you can make use of a common component."[1] p.54.
    - Input validation, however is not that easy. There are other issues explained in the section "input validation" below.
- Always check access rights. If your application is asked to display the details of a bank account, you now know you never can trust the account number that is given to you. Always verify, at each step in your application, that the user has the appropriate rights to see the details of the account.
- Encrypt values stored in cookies, hidden fields or URLs.

For more information on data manipulation, see [1] p.45, [10], [16].

3.1.2 SQL Query manipulation

This method consists of injecting SQL queries into database systems. The following example is taken from [1] p.56. Imagine a web form allowing you to change your password. Usually the form contains 4 fields: your userid (let's name the field *uid*), your old password (*oldpwd*), the new password (*newpwd*), and a "confirm new password" field (*newpwdconfirm*). The application will check that *newpwd* and *newpwdconfirm* are identical, that your old password is correct and then issue a query to the database to update your password. The query looks like: (the '$INPUT[fieldname]' syntax is used to represent the value entered in the html form 'fieldname' field).

Update usertable set pwd='$input[newpwd]' where uid='$input[uid]';

Everything works fine if the user follows the rules and inputs his or her uid. What if, the user sends the following value in the *uid* field: "*myuid' or uid='administrator*"?

The query really sent to the database will be:

Update usertable set pwd=’newpassword]’ where uid=’myuid' or uid='administrator’;

Yes, the administrator password has been reset with the user password! If you are not quite sure of the exact uid of the administrator account, you can try the following value *uid*=“*myuid' or uid like ‘%admin%*”, resetting the password of all accounts containing the string *admin*!

But poisoning SQL statements in web application does not stop there. There are ways to execute other SQL statements simply by appending them to the original statement. If you take the previous example and fills the *uid* field with: “*myuid'; insert into usertable ...*”, the DBMS will execute the update statement followed by the insert statement, resulting in the application to create a new user. Worse, a “*drop table usertable*” could be appended resulting in the application to become unusable!

Database servers also have features to execute operating system commands. Those commands can also be appended to the field value as in the following example (working for a system running on NT, with IIS and SQL server): uid=”*myuid'; exec master..xp_cmdshell copy c: \winnt\system32\cmd.exe c:\inetpub\scripts”*. This will copy the cmd.exe file (the windows command shell) in the web server directory where scripts are executed. The hacker can now call the cmd.exe in the script directory and execute any OS command from his browser. Any other OS command could be executed using the same mechanism. The point here is to show that security awareness among web developers is crucial. Without knowing about SQL query manipulation, anybody would probably write the statement shown above and nobody would have thought there was a potential danger.

If you send DBMS error messages back to the browser, you should read the article referenced in [4]. The article explains how to exploit the error messages to get the list of tables in the database, to get the field names of the tables, to retrieve data, ... As you could see from all the examples, knowing the database structure is the starting point of SQL query manipulation attacks.

Remarks

- SQL query manipulation is not specific to web applications. It also applies to client-server applications.
- It is worth noting that all the network traffic used in those attacks is perfectly valid from the firewall point of view.

Solutions to SQL query manipulation

- Basically, solutions are the same as those presented for data manipulation: filter any value coming from the client.
- "Delete stored procedures that you are not using like: *master..xp_cmshell, xp_startmail, xp_sendmail, sp_makewebtack*"[4]
- "Run the DBMS under a low privilege account"[4]

For more information on SQL query manipulation, see [4], [5], [6], [7].

3.1.3 Input validation

As we saw in previous examples, it is very important to check input data. However, there are also special issues to deal with when building an input validation algorithm.

- The “null byte” problem (see [9]). It is possible to encode « null bytes » in data transferred to the server. The problem is that « null bytes » are interpreted by several programming languages as « end of string ». So if your “input validation” algorithm does not filter “null bytes” before being applied, only the first portion (the one before the null byte) will be checked.
- Meta Characters (see [10]). Depending on the context, some characters (called meta characters) should be filtered (for example the ‘;’ character in a string inserted into an SQL query (see earlier in the document)). A good approach is to have a list of allowed characters instead of a list of forbidden characters. Examples of meta characters can be found in [10].
- Path traversal (see [11]). Some web applications store temporary information in files. An application can then take a filename as input parameter and send the file (where the temporary information is stored) back to the client browser. In that case, the hacker can manipulate the filename and try to gain access to interesting files using the classical “../../..” syntax. An example of an interesting file is “/etc/passwd” on a Unix machine, which contains all user ids and passwords. "Counter measures are to avoid using the file system unless absolutely necessary, to validate user input, to use databases instead of the file system, to never pass unchecked user input to file system commands"[11].

For more information on “input validation”, see [9], [10], [11].

3.1.4 Buffer overflows

A buffer overflow occurs when the size of data received from the client is larger than the size of the buffer where the data must be stored. Data is than copied outside the border of the buffer. That extra data may contain malicious code or may crash the system.

Some developers try to prevent the user from entering long strings in a HTML form field by setting the “maxlength” parameter of the form field. But of course, since this information is contained in the HTML source code, it can be changed too. There is absolutely no guarantee that the value of the field sent to the server will not be longer than expected.

Solutions

- Always check the input length on the server.
- Use languages like Java or C# which automatically checks bounds of buffers.
- If switching to Java or C# is not an option and you have to stay with C or C++, consider using “a library module that implements safe, bounds-checked buffers such as the standard C++ string module or libmib”[14]
- Always make sure that the programs handling user input do not run in the same process as your web server. With such a configuration, a buffer overflow would crash the application but not the web server as a whole. See [19] for a detailed explanation on how to do this with IIS.

For more information on "buffer overflow", see [8], [14] and [19] for example. There are also a lot of other articles available on the web.

3.2 Session hijacking

Before explaining the principles behind session hijacking, let me explain what a session is and why we need it. HTTP is a stateless protocol. Every time you ask for a page, or for a graphic within a page, a new connection is set up between your browser and the web server. There is no relationship at all between one connection and another. There is no state. The second connection does not know anything about what took place during the first connection. Though, this is perfectly acceptable when using the web to search for information, it is not appropriate for a web application where a context needs to be transmitted from page to page. For example, an e-commerce site, on the page where you enter your credit card number to complete the transaction, needs to know exactly the items that you have chosen on previous pages in order to compute the total amount. The e-commerce site needs a mechanism that identifies a "session", a virtual "established connection" between a browser and a web server to pass a context from page to page. Technically, this is done as follows (see figure 3 in the appendix): when the browser connects to the web server for the first time, the user has not been authenticated yet. The web server asks for credentials and generates a unique identifier (the session ID). The server can associate a context to each session ID, storing any kind of information in that context. The generated session ID is sent back to the browser. For every subsequent call to the server, the browser sends the session ID to the server. The server can then use the context associated with the transmitted session ID and "remember" data from page to page.

The system is working fine but there are risks associated with the technique. Since all you need to get access to the application is a valid session ID, one must make sure that it is not possible to guess a session ID or to steal a valid one. The process of stealing a session ID is called "session hijacking". Once a hacker has successfully obtained access to a legitimate user's session, "he or she can perform all normal application functions with the same privileges of the legitimate user"[15]. Web sites have been vulnerable to this kind of attack because "while it is generally clear that username/password pairs are indeed authentication data and therefore sensitive, it is not generally understood that these session IDs are also just as sensitive"[17].

Ways to capture a legitimate session ID:

- Guess it. Some session IDs are generated with a predictable sequence.
- Sniff traffic. If the traffic is not encrypted, the session ID can be sniffed.
- (*3)Usually, users exit the application using a logout feature. This invalidates the session ID and clears the context. The session ID cannot be used anymore. But many users do not use the logout feature and simply exit the browser. The problem is that browsers have "history" features. "Another user may come by a few minutes later, open up the browser, look at the history and be able to go in and perform operations under the original user's identity"[16] (though this would not work in all circumstances, depending among others, on the method used to store the session id and whether or not the browser was closed).

- (*4)An interesting scenario (though a bit theoretical) is the following. Remember the “referer field” we talked about earlier? If the user does not use the logout feature, the first web server the user visits after finishing the session will receive, in the “referer field”, the URL of the last page he visited during the session. If the session ID is contained in the URL, the referer field will also contain the session ID... "If the web server is operated by some sort of malicious administrator or someone is packet sniffing the connection, we give them the tools to go back into the session we just left and act as us."[16]

Solutions (inspired by the references [15], [16] (see also the interesting discussion at the end of the article), [17])
- Re-authenticate the user before critical actions are performed. With this kind of protection, the user must not only steal the session ID but also the user ID and password.
- Provide a logout mechanism and educate the users to use it! This would avoid the potential problems (*3) and (*4) described above.
- Do not use a predictable sequence for your session IDs.
- Require SSL so that the session IDs cannot be easily sniffed.
- Do not set the session time-out to a too long value.

For more information on session hijacking, see [15], [16], and [17].

3.2.1 A word on cross site scripting

Web browsers can execute client-side languages like JavaScript, VBScript, ... A script could potentially access local data and transmit it to a third party. Typically, session IDs stored in cookies can be obtained with that technique. This is not really a web application problem. It is more a client browser problem. But since the technique could be used to hijack sessions, it is worth mentioning.

Discussion groups or news lists give the possibility to directly input HTML into the messages. To exploit the vulnerability described in the previous paragraph, a hacker simply posts a message containing the malicious script in a discussion group. Anyone reading the message will run the script!

It is worth mentioning that more and more engines (like Yahoo for example) now filter the input and replace the malicious commands with harmless text.

For more information on cross-site scripting, see [12], [22].

3.3 Other issues

When putting in a place a Web application, there are many more issues to consider. This section highlights some of them.

3.3.1 Comments in HTML pages

Since it is a recommended best practice in software development, code is usually well documented. It is amazing what you can find in comments. Since on the web, the HTML page code is sent to the client, information can sometimes be inadvertently revealed! (Just try to do a ‘view source’ of your favorite HTML page and you may already find a lot of information...).

OWASP says the following about comments in HTML pages:

> Examples of information that can be found: "server path structure, true location of web root, debug information, cookie structure and even names and phone numbers (useful for social engineering). Automated comments generated by web page editors can inform the attacker about the precise software package (even down to the actual release) that is being used on the site. Known vulnerabilities in those packages can then be tried out against the site."[1] p.75
>
> A simple solution to the problem is to have a simple filter that removes all comments before pages are pushed to the production server. This should be part of the deployment methodology. ([1] p.75).

3.3.2 System administrators issues

Of course, network, firewall and web server configurations, the status of patches, … all have an influence on the overall security of the application. The focus of the article is not on those issues. In this section, I try to mention some system administrators related issues that are relevant to web applications.

It is always a good idea to segregate services. Typically, the security registry (the place where users and passwords are stored) should be separated (on another machine) from the content services. "This is a good idea because a compromise of the content service (the web server) does not compromise the user registry"[1] p.22. The same reasoning applies to the data server.

Make sure, your applications run with accounts that have the least possible privileges. In that case, if the application or the account is compromised, it restricts the damage that can be done.

Only run processes that are needed by your application. This limits the number of possible exploits.

For more information on system administrators issues, see [23], [24], and [25].

3.3.3 Authentication

A Web application must authenticate users. This subject is not covered in the article but there are lots of papers available on the web. For example, refer to [18], [19] and [20] for articles about authentication with IIS.

Do not forget it is also crucial that your users can authenticate your server. "On the web, the focus is reversed and protecting the consumer from spoofed servers is critical." [18]

3.3.4 Management issues

Plan for security from the very beginning of the project. Involve system administrators early in the development cycle. The overall security depends as much on the system configuration as on the way the application is programmed. Developers and system administrators should work hand in hand. From the very beginning of the project, developers should apply the rules defined by system

administrators. For more information on this issue, see [23].

4 Summary

In the article, I described several methods to attack web applications while perfectly respecting the http protocol, making firewalls and intrusion detection systems unable to detect the attacks. Usually, the methods described in the article are entry points to the system. For example, [21] describes how such a very small hole can widen and lead to a fully compromised system.

The most important points to remember from the article are:

For developers

- The way you program your application has a tremendous influence on the overall security.
- Always strongly validate any information coming from the client
- Make sure your session mechanism is bullet-proof.
- Use an appropriate authentication mechanism.

For system administrators

- Apply all best practices regarding network and host security.
- Run the application under accounts with the fewest possible privileges.
- Separate services as much as possible.

For managers

- Plan for security from the very beginning of the project.
- Make sure developers and system administrators work hand in hand.

5 Appendix

Figure 1 describes (in a very simplified way) how clients and servers interact on the web.

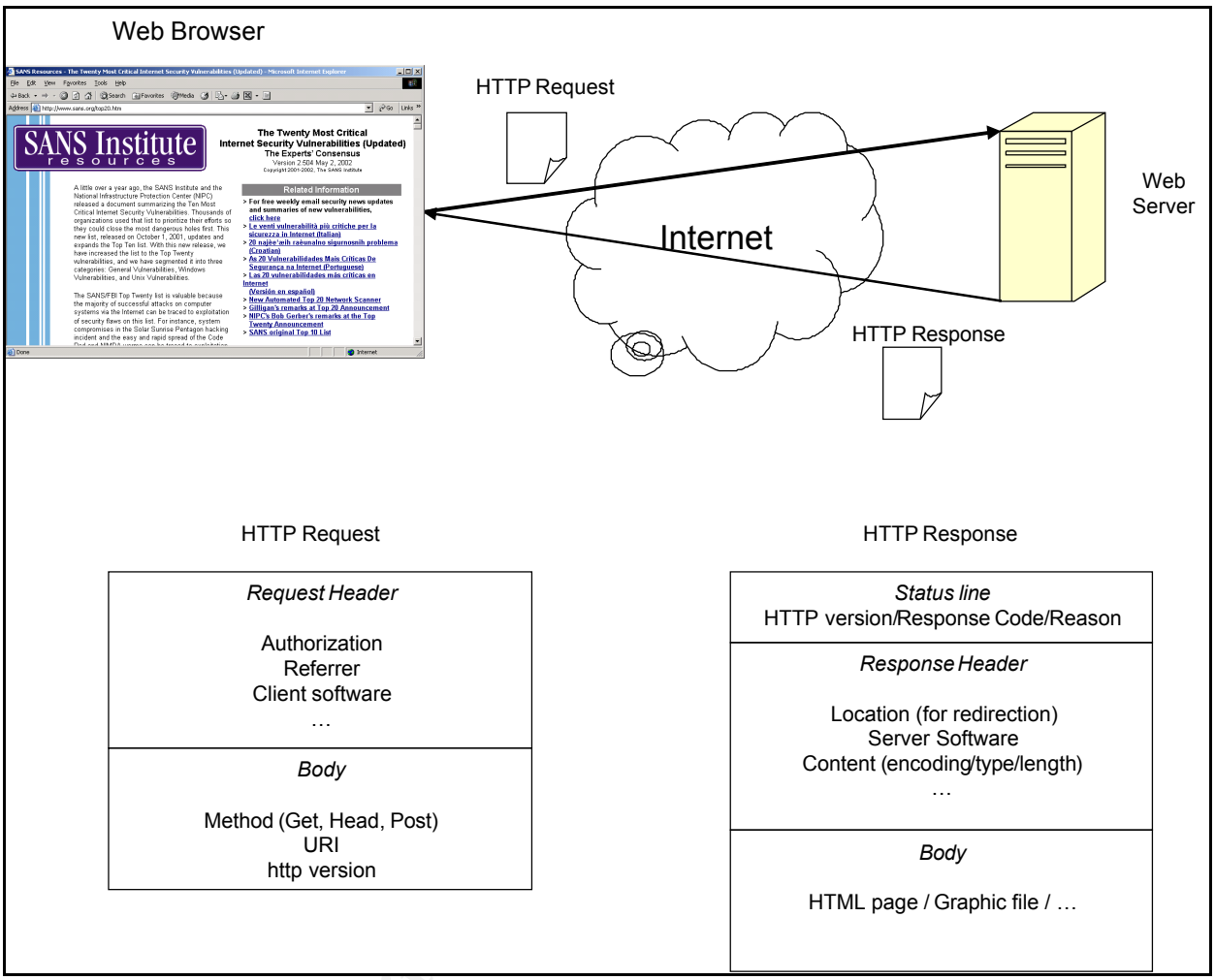

Figure 1

The browser sends an Http request to the web server. The request contains two parts:

- a header, containing information like authorization information, an identification of the client software (IE5, NS5, ...), a referrer field (I talk about this field in the article), ...
- the body of the request, containing information about
  - the version of http used
  - the method
    - GET: returns the page specified in the URI
    - POST: to send data to the server (in forms)
    - HEAD: returns the response header only
  - The URI: (ex: www.mycompany.com/index.html)

The web server processes the request and sends an http response back to the browser. The response contains

- A status line, indicating whether the request was processed successfully or not (a typical rejection reason is "not authorized").
- A response header that contains information about
  - The content of the response (its type (html, graphics, …), its length, its method of encoding)
  - The server software
  - A location (in case the response is a redirection to another page)
  - ...
- The body of the response
  - The HTML page, a graphic, …

With the http protocol, a new connection is established between the browser and the server for each request. For an HTML page containing 2 pictures, 3 connections will be established: one for the HTML page and one for each picture.

Figure 2 describes how a browser can send information to the server.

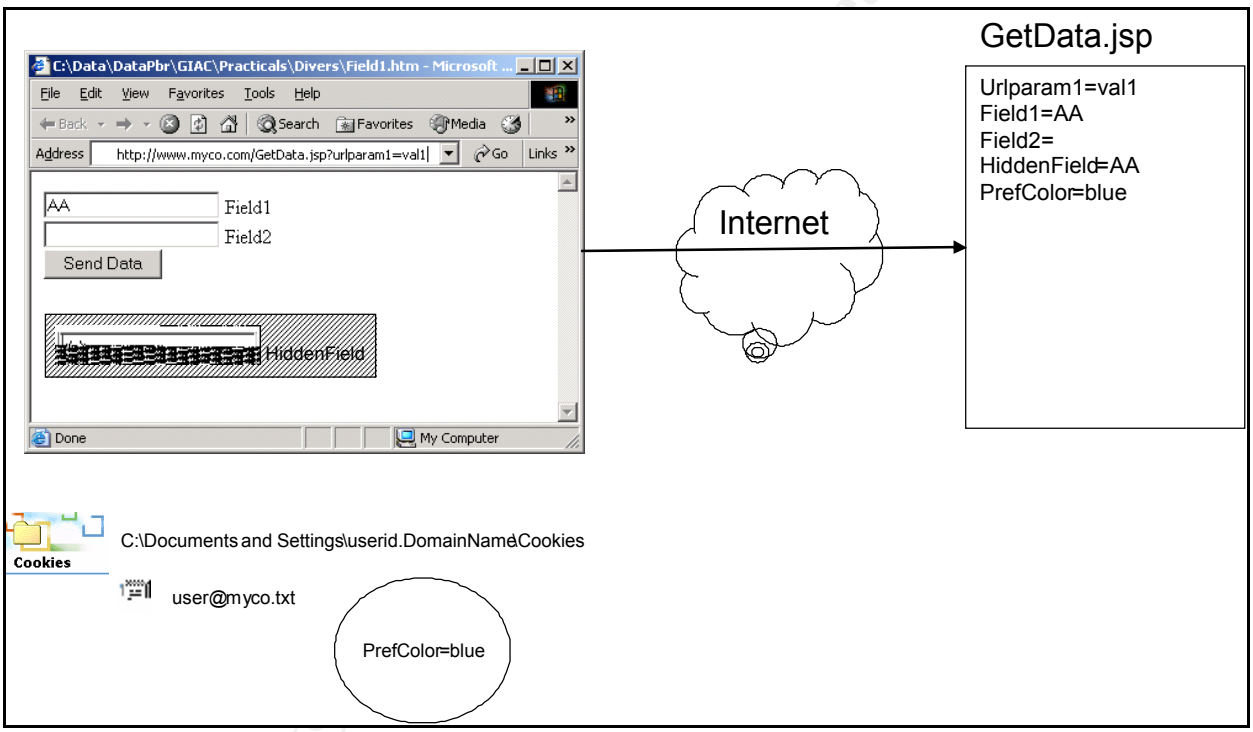

Figure 2

As we saw in figure 1, some information is already sent to the server in the http request header.

The browser can transmit data to the server in three different ways:

- in the URL itself ('*urlparam1=val1'* in the example)
- in fields contained in HTML forms (*Field1* and *Field2* in the example). Forms can also contain hidden fields that are not displayed by the browser but can contain information.
- in cookies. Cookies contain information generated by the server. They are stored either in memory (non persistent cookies) or on your hard drive

(persistent cookies) in the directory shown in figure 2 (for IE). The values stored in cookies are transmitted in every http request. But, cookies also have a scope. They are only sent to the web server that generated the request.

Figure 3 describes the session mechanism.

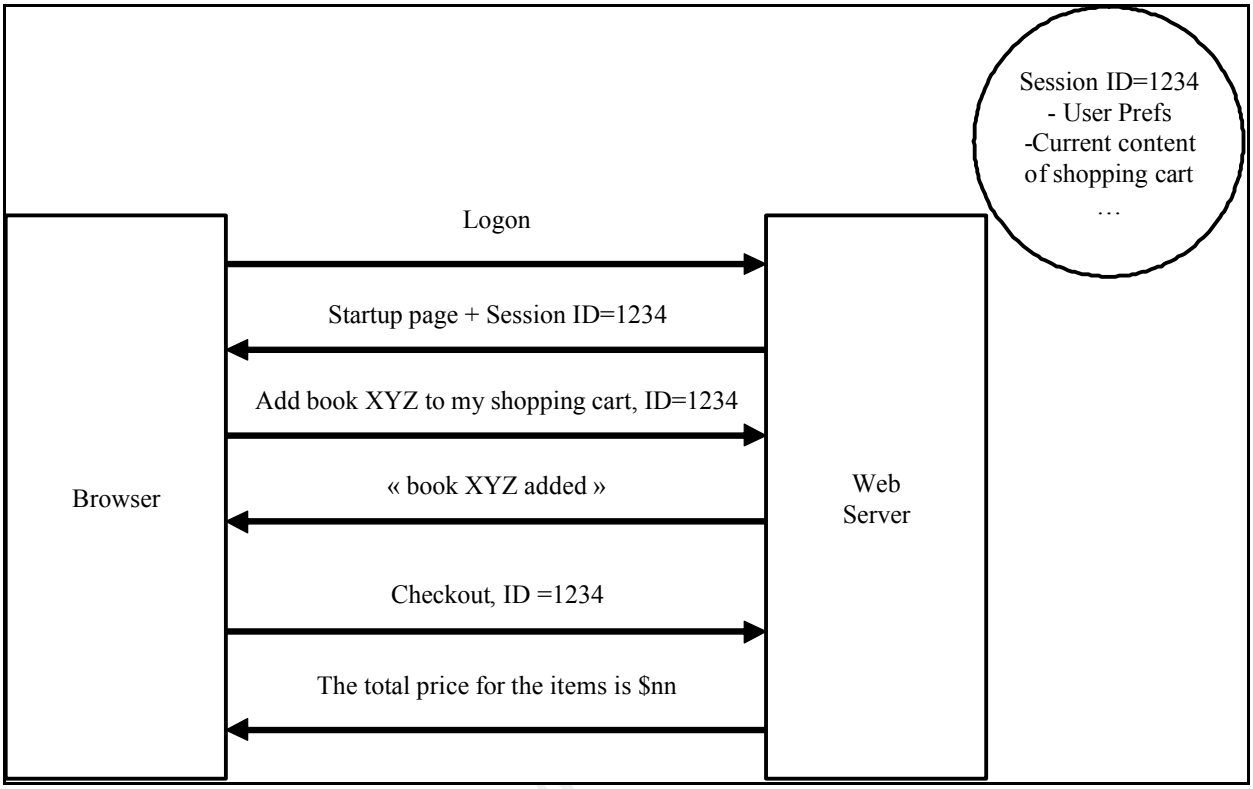

Figure 3

6 References

General references on web application security

[1] OWASP. “The Open Web Application Security Project – A guide to building secure web applications and web services”. 24 Jun 2002. http://www.cgisecurity.com/owasp/OWASPBuildingSecureWebApplicationsAndWebServices-V1.0.pdf (2 Aug 2002)

[2] GrossMan, Jeremiah. "Web application security and presenting the whitehat arsenal". http://www.blackhat.com/presentations/win-usa-02/grossman-winsec2002.ppt (7 Aug 2002)

[3] Musa, Zarina. "Web application security". GSEC Practical Assignment. 21 Feb 2002. http://www.giac.org/practical/Zarina_Musa_GSEC.doc (2 Aug 2002)

References on SQL query manipulation

[4] Beyond-Security Ltd. “SQL injection walkthrough”. http://www.securiteam.com/securityreviews/5DP0N1P76E.html (2 Aug 2002)

[5] Anley, Chris. "Advanced SQL injection in SQL server applications". 2002. http://www.nextgenss.com/papers/advanced_sql_injection.pdf (2 Aug 2002)

[6] (no author) "SQL injection FAQ". http://www.sqlsecurity.com/faq-inj.asp (2 Aug 2002)

[7] SPI Dynamics. "SQL Injection. Are your web applications vulnerable?". 2002. http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf (2 Aug 2002)

References on input validation and buffer overflow

[8] Beyond-Security Ltd. "Writing buffer overflow exploits – a tutorial for beginners". http://www.securiteam.com/securityreviews/5OP0B006UQ.html (2 Aug 2002)

[9] Eizner, Martin. “Null Bytes”. http://www.owasp.org/asac/input_validation/nulls.shtml (2 Aug 2002)

[10] Eizner, Martin. “Meta characters”. http://www.owasp.org/asac/input_validation/meta.shtml (2 Aug 2002)

[11] Eizner, Martin. “Path traversal”. http://www.owasp.org/asac/input_validation/pt.shtml (2 Aug 2002)

[12] CERT Advisory CA-2000-02. "Malicious HTML tags embedded in Client Web

Requests". 3 Feb 2000.
http://www.cert.org/advisories/CA-2000-02.html (2 Aug 2002)

[13] CERT Coordination center. "Understanding malicious content mitigation for web developers". 2 Feb 2000.
http://www.cert.org/tech_tips/malicious_code_mitigation.html (16 Aug 2002)

[14] Frykholm, Niklas. "Countermeasures against buffer overflow attacks". 30 Nov 2000
http://www.rsasecurity.com/rsalabs/technotes/buffer/buffer_overflow.html (21 Aug 2002)

References on session management

[15] Pennington, Bill & Endler, David. "Authentication and session management". http://www.owasp.org/asac/auth-session/hijack.shtml (2 Aug. 2002)

[16] Olson, Joshua. "Session hijacking, Cold Fusion, Dynamic Proxies". 14 Oct 2000.
http://www.evolt.org/article/Session_Hijacking_Cold_Fusion_Dynamic_Proxies/20/3516/ (2 Aug. 2002)

[17] Pennington, Bill. "Authentication and session management".
http://www.owasp.org/asac/auth-session/replay.shtml (2 Aug. 2002)

References on Authentication

[18] Brown, Keith. "Web Security: Putting a secure front end on your COM+ distributed applications". June 2000.
http://msdn.microsoft.com/msdnmag/issues/0600/websecure/websecure.asp. (2 Aug 2002)

[19] Brown, Keith. "Web Security: Part 2: Introducing the web Application Manager, Client Authentication Options, and Process Isolation". July 2000.
http://msdn.microsoft.com/msdnmag/issues/0700/websecure2/websecure2.asp (2 Aug 2002)

[20] Sabbadin, Enrico. "Implementing secured web applications wit IIS5".
http://www.vb2themax.com/HtmlDoc.asp?Table=Articles&ID=320 (2 Aug 2002)

Various references

[21] Glaser, JD & Shah, Saumil. "One-Way Hacking: Futility of Firewalls in Web Hacking". 21 Nov 2001
http://www.blackhat.com/presentations/bh-europe-01/jd-glaser/glaser.ppt (2 Aug 2002)

[22] Grossman, Jeremiah. "Cross Site Scripting".

http://www.owasp.org/asac/input_validation/css.shtml (2 Aug 2002)

[23] Pogue, Matt. "Securing Microsoft Web applications – A guide for systems administrators". 10 Dec 2001.
http://rr.sans.org/web/web_apps.php (requires logon to SANS reading room) (1 Aug 2002)

[24] Skoudis, Edward. "Cracker Tools and techniques. Faster, stealthier ...more dangerous". July 2002.
http://www.infosecuritymag.com/2002/jul/faster.shtml (13 Aug 2002)

[25] Symantec. "The behaviors and tools of today's hackers". 18 June 2002.
http://enterprisesecurity.symantec.com/article.cfm?articleid=1398&PID=12493901&EID=0 (13 Aug 2002)