ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Technical Review on Web Application Security Vulnerabilities

Chandan Singh
Research scholar, In-Charge, Centre for Computer Science, Sri JNPG College(KKC), Lucknow(U.P.), India
1sichandansingh@gmail.com

Abstract

The web is an indispensable part of our lives. Every day, millions of users purchase items, transfer money, retrieve information and communicate over the web. Although the web is convenient for many users because it provides anytime, anywhere access to information and services, at the same time, it has also become a prime target for miscreants who attack unsuspecting web users with the aim of making an easy profit. The last years have shown a significant rise in the number of web-based attacks, highlighting the importance of techniques and tools for increasing the security of web applications. An important web security research problem is how to enable a user on an entrusted platform (e.g., a computer that has been compromised by malware) to securely transmit information to a web application. Solutions that have been proposed to date are mostly hardware-based and require (often expensive) peripheral devices such as smartcard readers and chip cards. In this paper, we discuss some common aspects of client-side attacks (e.g., Trojan horses) against web applications and present two simple techniques that can be used by web applications to enable secure user input. We also conducted two usability studies to examine whether the techniques that we propose are feasible.

Keywords:- Electronic Commerce, Security, Vulnerabilities, Scripting, Hacking.

Web Application Security

The Web is the playground of 800 million citizens, home to 100 million Web sites, and transporter of billions of dollars every day. International economies have become dependent on the Web as a global phenomenon. It's not been long since Web mail, message boards, chat rooms, auctions, shopping; news, banking, and other Web-based software have become part of digital life. Today, users hand over their names, addresses, social security numbers, credit card information, phone numbers, mother's maiden name, and annual salary, date of birth, and sometimes even their favorite color or name of their kindergarten teacher to receive financial statements, tax records, or day trade stock. And did I mention that roughly 8 out of 10 Web sites have serious security issues putting this data at risk? Even the most secure systems are plagued by new security threats only recently identified as Web Application Security, the term used to describe the methods of securing web-based software.

The organizations that collect personal and private information are responsible for protecting it from prying eyes. Nothing less than corporate reputation and personal identity is at stake. As vital as Web application security is and has been, we need to think bigger. We're beyond the relative annoyances of identity theft, script kiddy defacements, and full-disclosure antics. New Web sites are launched that control statewide power grids, operate hydroelectric dams, fill prescriptions, administer payroll for the majority of corporate America, run corporate networks, and manage other truly critical functions. Think of what a malicious compromise of one of these systems could mean. It's hard to imagine an area of information security that's more important. Web applications have become the easiest, most direct, and arguably the most exploited route for system compromise. Until recently everyone thought firewalls, SSL, intrusion detection systems, network scanners, and passwords were the answer to network security. Security professionals borrowed from basic military strategy where you set up a perimeter and defended it with everything you had. The idea was to allow the good guys in and keep the bad guys out. For the most part, the strategy was effective, that is until the Web and e-commerce forever changed the landscape. E-commerce requires firewalls to allow in Web (port 80 Hypertext Transfer Protocol [HTTP] and 443 Hypertext Transfer Protocol Secure sockets [HTTPS]) traffic.

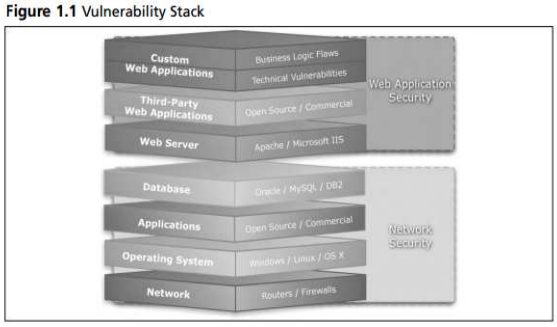
Essentially meaning you have to let in the whole world and make sure they play nice.

ISSN: 2277-9655
Impact Factor: 1.852

Seemingly overnight the Internet moved from predominantly walled networks to a global ecommerce bazaar. The perimeter became porous and security administrators found themselves without any way to protect against insecure Web applications.

Web developers are now responsible for security as well as creating applications that fuel Web business. Fundamental software design concepts have had to change. Prior to this transformation, the average piece of software was utilized by a relatively small number of users. Developers now create software that runs on Internet-accessible Web servers to provide services for anyone, anywhere. The scope and magnitude of their software delivery has increased exponentially, and in so doing, the security issues have also compounded. Now hundreds of millions of users all over the globe have direct access to corporate servers, any number of which could be malicious adversaries. New terms such as cross-site scripting, Structured Query Language (SQL) injection, and a dozen of other new purely Web-based attacks have to be understood and dealt with.

Figure 1.1 Vulnerability Stack



Web application security is a large topic encompassing many disciplines, technologies, and design concepts. Normally, the areas we're interested in are the software layers from the Web server on up the vulnerability stack as illustrated in Figure 1.1.This includes application servers such as JBoss, IBM WebSphere, BEA WebLogic, and a thousand others. Then we progress in the commercial and open source Web applications like PHP Nuke, Microsoft Outlook Web Access, and SAP. And after all that, there are the internal custom Web applications that organizations develop for themselves .This is the lay of the land when it comes to Web application security. One of the biggest threats that Web application developers have to understand and know how to mitigate is XSS attacks. While XSS is a relatively small part of the Web application security field, it possible represents the most dangerous, with respect to the typical Internet user. One simple bug on a Web application can result in a compromised browser through which an attacker can steal data; take over a user's browsing experience, and more.

Ironically, many people do not understand the dangers of XSS vulnerabilities and how they can be and are used regularly to attack victims. This book's main goal is to educate readers through a series of discussions, examples, and illustrations as to the real threat and significant impact that one XSS can have.

Review from Literature

No language can prevent insecure code, although there are language features which could aid or hinder a security-conscious developer [1]. Insecure software is already undermining our financial, healthcare, defence, energy, and other critical infrastructure. As our digital infrastructure gets increasingly complex and interconnected the difficulty of achieving application security increases exponentially. We can no longer afford to tolerate relatively simple security problems like those presented below. The vulnerabilities [2] explained in this paper are:

1. **Remote code execution**
2. **SQL injection**
3. **Format string vulnerabilities**
4. **Cross Site Scripting (XSS)**
5. **Ajax security: Are AJAX Applications Vulnerable**
6. **Cross Site Scripting – XSS – The Underestimated Exploit**
7. **Google Hacking**
8. **Directory Traversal Attacks**

Remote Code Execution

In computer security, arbitrary code execution is used to describe an attacker's ability to execute any commands of the attacker's choice on a target machine or in a target process. It is commonly used in arbitrary code execution vulnerability to describe a software bug that gives an attacker a way to execute arbitrary code. A program that is designed to exploit such a vulnerability is called an arbitrary code execution exploit. Most of these vulnerabilities allow the execution of machine code and most exploits therefore inject and execute shell code to give an attacker an easy way to manually run arbitrary commands. The ability to trigger arbitrary code execution from one machine on another (especially via a wide-area network such as the Internet) is often referred to as remote code execution.

It is the worst effect a bug can have because it allows an attacker to completely take over the vulnerable process. From there the attacker can potentially take complete control over the machine the process is running on. Arbitrary code execution vulnerabilities are commonly exploited by malware to run on a computer without the owner's

ISSN: 2277-9655
Impact Factor: 1.852

consent or by an owner to run homebrew software on a device without the manufacturer's consent.

Arbitrary code execution is commonly achieved through control over the program counter (also known as the instruction pointer) of a running process. The instruction pointer points to the next instruction in the process that will be executed. Control over the value of the instruction pointer therefore gives control over which instruction is executed next. In order to execute arbitrary code, many exploits inject code into the process (for example by sending input to it which gets stored in an input buffer) and use a vulnerability to change the instruction pointer to have it point to the injected code. The injected code will then automatically get executed. This type of attack exploits the fact that Von Neumann architecture computers do not make a general distinction between code and data, so that malicious code can be camouflaged as harmless input data. Many newer CPUs have mechanisms to make this harder, such as a no-execute bit.

Once the invader can execute arbitrary code directly on the OS, there is often an attempt at a privilege escalation exploit in order to gain additional control. This may involve the kernel itself or an account such as Administrator, SYSTEM, or root. With or without this enhanced control, exploits have the potential to do severe damage or turn the computer into a zombie - but privilege escalation helps with hiding the attack from the legitimate administrator of the system. An arbitrary remote code execution with privilege escalation vulnerability in widely-deployed software is thus the worst vulnerability sub-type of them all. If bugs of this kind become known, fixes are usually made available within a few hours.

Affected

- Symantec Backup Exec CP Server (BE CPS)11.0, 12.0, 12.5 All
- Symantec Veritas NetBackup (NBU) with NetBackup Operations Manager (NOM) installed 6.0.x, 6.5.x Windows, Solaris
- Symantec Veritas NetBackup RealTime Protection 6.5 All
- Symantec Veritas Backup Reporter (VBR) 6.0.x, 6.2.x, 6.5.x, 6.6 Windows, Solaris
- Symantec Veritas Storage Foundation (SF) 3.5 onwards All
- Symantec Veritas Storage Foundation for Windows (SFW) ?? All
- Symantec Veritas Storage Foundation for High Availability (SFHA) 3.5 onwards All
- Symantec Veritas Storage Foundation Manager (SFM) 1.0, 1.1, 1.1.1Ux, 1.1.1Win, 2.0 All
- Symantec Veritas Cluster Server Management Console (VCSMC) 5.0, 5.1, 5.5 All
- Symantec Veritas Storage Foundation Cluster File System (SFCFS) 3.5 (HP-UX), 4.0, 4.1, 5.0 (AIX, HP-UX, Linux, Solaris) Various
- Symantec Veritas Cluster Server Traffic Director ?? All
- Symantec Veritas Application Director (VAD) 1.x, 1.1 PE, 1.1 PE-RPx All
- Symantec Veritas Cluster Server One (VCSOne) 2.x All
- Symantec Veritas Storage Foundation for Oracle (SFO) 4.1 (HP-UX, Solaris)
- 5.0 (AIX, HP-UX, Linux, Solaris)
- 5.0.1 (HP-UX) Various
- Symantec Veritas Storage Foundation for DB2 4.1 (Solaris, Linux)
- 5.0 (Solaris, AIX, Linux) Various
- Symantec Veritas Storage Foundation for Sybase 4.1, 5.0 Solaris
- Symantec Veritas Command Central Storage (CCS) 4.3, 5.0 GA, 5.0 MP1, 5.0 MP1 RP1-RP6, 5.1 All
- Symantec Veritas Command Central Enterprise Reporter (CC-ER) 5.0 GA, 5.0 MP1, 5.0 MP1RP1, 5.1 All
- Symantec Veritas Command Central Storage Change Manager (CC-SCM) 5.1 All
- Symantec Veritas Virtual Infrastructure (VxVI) 1.0, 1.0SP1 Linux
- Symantec Veritas MicroMeasure 5.0 All
- NOTE:
- Only the versions listed above are affected.
- NetBackup is affected only if NetBackup Operations Manager (NOM) is installed.
- PureDisk is not vulnerable in the default configuration

SQL Injection

This is a hacking method that allows an unauthorized attacker to access a database server. It is facilitated by a common coding blunder: the program accepts data from a client and executes SQL queries without first validating the client's input. The attacker is then free to extract, modify, add, or delete content from the database. In some circumstances, he may even penetrate past the database server and into the underlying operating system. Hackers typically test for SQL injection vulnerabilities by sending the application input that would cause the server to generate an invalid SQL query[4]. If the server then returns an error message to the client, the attacker will attempt to reverse-engineer portions of the original SQL query using information gained from these error messages. The typical administrative

ISSN: 2277-9655
Impact Factor: 1.852

safeguard is simply to prohibit the display of database server error messages. Regrettably, that's not sufficient. If your application does not return error messages, it may still be susceptible to "blind" SQL injection7 attacks.

In the following example, assume that a web site is being used to mount an attack on the database. If you think about a typical SQL statement, you might think of something like:

```
SELECT ProductName, QuantityPerUnit, UnitPrice
FROM Products
WHERE ProductName LIKE 'G%'
```

The objective of the attacker is to inject their own SQL into the statement that the application will use to query the database. If, for instance, the above query was generated from a search feature on a web site, then they user may have inserted the "G" as their query. If the server side code then inserts the user input directly into the SQL statement, it might look like this:

```
string sql = "SELECT ProductName,
QuantityPerUnit, UnitPrice "+
   "FROM Products " +
   "WHERE ProductName LIKE
"'+this.search.Text+"%';
SqlDataAdapter da = new SqlDataAdapter(sql,
DbCommand);
da.Fill(productDataSet);
```

This is all fine if the data is valid, but what if the user types something unexpected? What happens if the user types:

```
' UNION SELECT name, type, id FROM sysobjects;-
-
```

Note the initial apostrophe; it closes the opening quote in the original SQL statement. Also, note the two dashes at the end; that starts a comment, which means that anything left in the original SQL statement is ignored.

Now, when the attacker views the page that was meant to list the products the user has searched for, they get a list of all the names of all the objects in the database and the type of object that they are. From this list, the attacker can see that there is a table called Users. If they take note of the id for the Users table, they could then inject the following:

```
' UNION SELECT name, '', length FROM
syscolumns
WHERE id = 1845581613;--
```

This would give them a list of the column names in the Users table. Now they have enough information to get access to a list of users, passwords, and if they have admin privileges on the web site.

Prevention

- Encrypt sensitive data.
- Access the database using an account with the least privileges necessary.
- Install the database using an account with the least privileges necessary.
- Ensure that data is valid.
- Do a code review to check for the possibility of second-order attacks.
- Use parameterised queries.
- Use stored procedures.
- Re-validate data in stored procedures.
- Ensure that error messages give nothing away about the internal architecture of the application or the database.

Format String Vulnerabilities

Format String Attack

Format String Attacks[5] alter the flow of an application by using string formatting library features to access other memory space. Vulnerabilities occur when user-supplied data are used directly as formatting string input for certain C/C++ functions (e.g. fprintf, printf, sprintf, setproctitle, syslog).

If an attacker passes a format string consisting of printf conversion characters (e.g. "%f", "%p", "%n", etc.) as a parameter value to the web application, they may:

- Execute arbitrary code on the server
- Read values off the stack
- Cause segmentation faults / software crashes

Format String attacks are related to other attacks in the Threat Classification: Buffer Overflows and Integer Overflows. All three are based in their ability to manipulate memory or its interpretation in a way that contributes to an attacker's goal.

Example

Let's assume that a web application has a parameter emailAddress, dictated by the user. The application prints the value of this variable by using the printf function:

```
printf(emailAddress);
```

If the value sent to the emailAddress parameter contains conversion characters, printf will parse the conversion characters and use the additionally supplied corresponding arguments. If no such arguments actually exist, data from the stack will be used in accordance with the order expected by the printf function.

ISSN: 2277-9655
Impact Factor: 1.852

The possible uses of the Format String Attacks[6] in such a case can be:

- Read data from the stack:

If the output stream of the printf function is presented back to the attacker, he may read values on the stack by sending the conversion character "%x" (one or more times).

- Read character strings from the process' memory:

If the output stream of the printf function is presented back to the attacker, he can read character strings at arbitrary memory locations by using the "%s" conversion character (and other conversion characters in order to reach specific locations).

- Write an integer to locations in the process' memory:

By using the "%n" conversion character, an attacker may write an integer value to any location in memory. (e.g. overwrite important program flags that control access privileges, or overwrite return addresses on the stack, etc.)

Cross Site Scripting – XSS – The Underestimated Exploit

What is Cross Site Scripting?

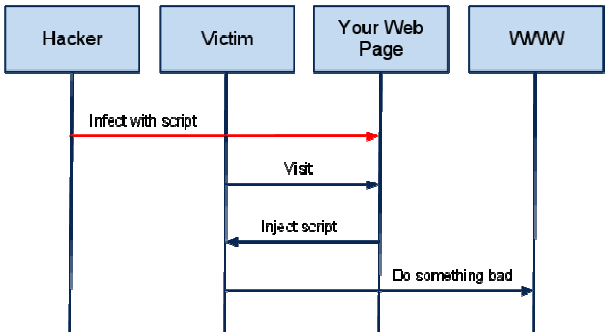
XSS is an attackusing a browser side scripting language (usually JavaScript)[7]. The goal of the attacker is to make the malicious script appear to be from the site being attacked, so the user's browser can't tell the script being executed is not meant to be aprt of the site they are viewing. This is usually accomplished by an attacker by submitting specially crafted values into the target site's URL or web forms, or anywhere user generated content is displayed on the site. Users can fall into an XSS attack primarily in two ways:

1. Tricking a user to click on a link, via having them view an email, or having them view another site under attacker control. This could be as benign as a forum where image tags are allowed, and an attacker posts something like
2. Creating an XSS attack and storing it on the target site, such as in a forum post, profile, or other method. This type of attack may also be self-propagating, creating an XSS worm.

XSS arises in a variety of ways. Code is planted on a site or in a link a user will be tricked into clicking, causing the XSS exploit to execute on the client's browser. Cross site scripting attempts can be notoriously hard to detect as they may take many forms, such as normal human readable text, or specially encoded characters used to trick attempts to detect it.

There are two broad attack surfaces which must be protected from XSS. The first is the users browser environment, and any JavaScript or other code which is executed by the browser, and the second is server side. Browser attacks are executed via variables like the http referrer (page the user was last on and clicked from), or other http type methods such as document.location or document.URL. These variables are supplied by the user's browser, and not the site the page was requested from, so the site has less control. If these values are written into the document at the user side, then the page may be modified with an XSS attack after it has been delivered to the user, as opposed to server-side XSS, where the attack is rendered by the server prior to being sent. In-Body attacks are less likely (in some cases impossible) to prevent with server-side input checking, and should be prevented directly in the client-side code instead

In a typical XSS attack the hacker infects a legitimate web page with his malicious client-side script. When a user visits this web page the script is downloaded to his browser and executed. There are many slight variations to this theme, however all XSS attacks follow this pattern, which is depicted in the diagram below.



A High Level View of a typical XSS Attack

A basic example of XSS is when a malicious user injects a script in a legitimate shopping site URL which in turn redirects a user to a fake but identical page. The malicious page would run a script to capture the cookie of the user browsing the shopping site, and that cookie gets sent to the malicious user who can now hijack the legitimate user's session. Although no real hack has been performed against the shopping site, XSS has still exploited a scripting weakness in the page to snare a user and take command of his session. A trick which often is used to make malicious URLs less obvious is to have the XSS part of the URL encoded in HEX (or other encoding methods). This will look harmless to the user who recognizes the URL he is familiar with, and

ISSN: 2277-9655
Impact Factor: 1.852

simply disregards and following 'tricked' code which would be encoded and therefore inconspicuous.

Site owners are always confident, but so are hackers!

Without going into complicated technical details, one must be aware of the various cases which have shown that XSS can have serious consequences when exploited on a vulnerable web application. Many site owners dismiss XSS on the grounds that it cannot be used to steal sensitive data from a back-end database. This is a common mistake because the consequences of XSS against a web application and its customers have been proven to be very serious, both in terms of application functionality and business operation. An online business project cannot afford to lose the trust of its present and future customers simply because nobody has ever stepped forward to prove that their site is really vulnerable to XSS exploits. Ironically, there are stories of site owners who have boldly claimed that XSS is not really a high-risk exploit. This has often resulted in a public challenge which hackers are always itching to accept, with the site owner having to later deal with a defaced application and public embarrassment.

The repercussions of XSS

Analysis of different cases which detail XSS exploits teaches us how the constantly changing web technology is nowhere close to making applications more secure. A thorough web search will reveal many stories of large-scale corporation web sites being hacked through XSS exploits, and the reports of such cases always show the same recurring consequences as being of the severe kind.

Exploited XSS is commonly used to achieve the following malicious results:

- Identity theft
- Accessing sensitive or restricted information
- Gaining free access to otherwise paid for content
- Spying on user's web browsing habits
- Altering browser functionality
- Public defamation of an individual or corporation
- Web application defacement
- Denial of Service attacks

Any site owner with a healthy level of integrity would agree that none of the above can really be considered us frivolous or unimportant impacts on a vulnerable site. Security flaws in high-profile web sites have allowed hackers to obtain credit card details and user information which allowed them to perform transactions in their name. Legitimate users have been frequently tricked into clicking a link which redirects them to a malicious but legitimate-looking page which in turn captures all their details and sends them straight to the hacker. This example might not sound as bad as hacking into a corporate database; however it takes no effort to cause site visitors or customers to lose their trust in the application's security which in turn can result in liability and loss of business.

XSS Attack Vectors

Internet applications today are not static HTML pages. They are dynamic and filled with ever changing content. Modern web pages pull data from many different sources. This data is amalgamated with your own web page and can contain simple text, or images, and can also contain HTML tags such as <p> for paragraph, <img> for image and <script> for scripts. Many times the hacker will use the 'comments' feature of your web page to insert a comment that contains a script. Every user who views that comment will download the script which will execute on his browser, causing undesirable behaviour. Something as simple as a Facebook post on your wall can contain a malicious script, which if not filtered by the Facebook servers will be injected into your Wall and execute on the browser of every person who visits your Facebook profile.

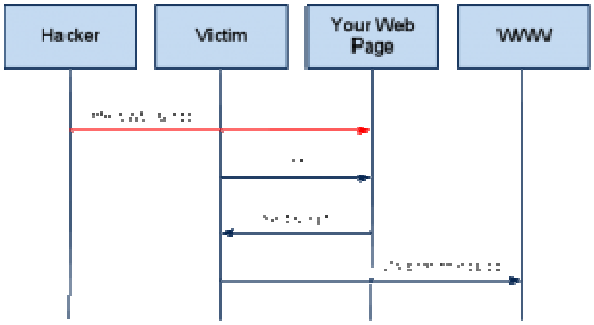
If you would like a deeper discussion on the different XSS attack vectors and examples of what they look like you should refer to the following article, which explains XSS, it's attack vectors and some more examples of what an attack looks like.

A practical example of XSS on an Acunetix test site.

The following example is not a hacking tutorial. It is just a basic way to demonstrate how XSS can be used to control and modify the functionality of a web page and to re-design the way the page processes its output. The practical use of the example may be freely debated; however anyone may see the regular reports which describe how advanced XSS is used to achieve very complex results, most commonly without being noticed by the user. I encourage also those individuals with no hacking knowledge to try the following example, I am sure you will find it interesting.

1. Load the following link in your browser: http://testasp.vulnweb.com/search.asp, you will notice that the page is a simple page with an input field for running a search

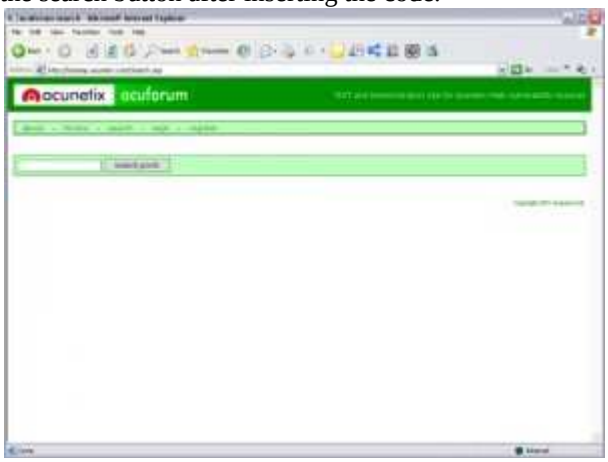
ISSN: 2277-9655
Impact Factor: 1.852



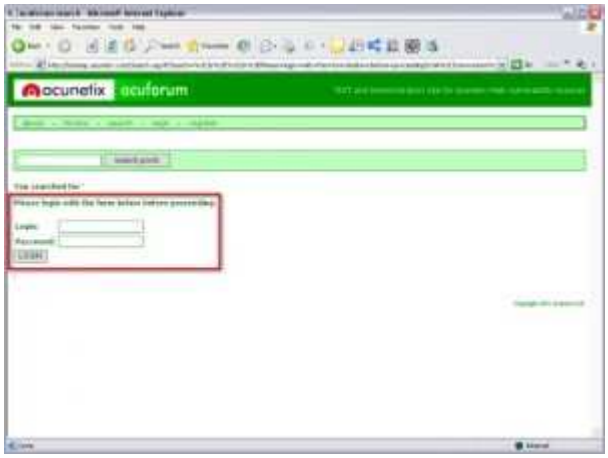
A High Level View of a typical XSS Attack

2. Try to insert the following code into the search field, and notice how a login form will be displayed on the page:
Please login with the form below before proceeding:

Please login with the form below before proceeding:<form action="destination.asp"><table><tr><td>Login:</td><td><input type=text length=20 name=login></td></tr><tr><td>Password:</td><td><input type=text length=20 name=password></td></tr></table><input type=submit value=LOGIN></form>, then simply hit the search button after inserting the code.



Through the XSS flaw on the page, it has been possible to create a FAKE login form which can convince gather a user's credentials. As seen in step 2, the code contains a section which mentions "destination.asp". That is where a hacker can decide where the FAKE login form will send the user's log-in details for them to be retrieved and used maliciously.
A hacker can also inject this code by passing it around via the browser's address bar as follows:
http://testasp.vulnweb.com/Search.asp?tfSearch=%3Cbr%3E%3Cbr%3EPlease+login+with+the+form+below+before+proceeding%3A%3Cform+action%3D%22test.asp%22%3E%3Ctable%3E%3Ctr%3E%3Ctd%3ELogin%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dlogin%3E%3C%2Ftd%3E%3C%2Ftr%3E%3Ctr%3E%3Ctd%3EPassword%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dpassword%3E%3C%2Ftd%3E%3C%2Ftr%3E%3C%2Ftable%3E%3Cinput+type%3Dsubmit+value%3DLOGIN%3E%3C%2Fform%3E



This will create the same result on the page, showing how XSS can be used in several different ways to achieve the same result. After the hacker retrieves the user's log-in credentials, he can easily cause the browser to display the search page as it was originally and the user would not even realize that he has just been fooled. This example may also be seen in use in all those spam emails we all receive. It is very common to find an email in your inbox saying how a certain auctioning site suspects that another individual is using your account maliciously, and it then asks you to click a link to validate your identity. This is a similar method which directs the unsuspecting user to a FAKE version of the auctioning site, and captures the user's log-in credentials to then send them to the hacker.

Why wait to be hacked?

The observation which can be made when new stories of the latest hacks are published is that the sites which belong to the large brands and corporations are hacked in exactly the same way as those sites owned by businesses on a much smaller budget. This clearly shows how lack of security is not a matter of resources, but it is directly dependant on the lack of awareness among businesses of all size. Statistically, 42% of web applications which request security audits are vulnerable to XSS, which is clearly the most recurring high-risk exploit among all the applications tested. The effort to raise awareness about how easy it is for an expert hacker to exploit a

ISSN: 2277-9655
Impact Factor: 1.852

vulnerable application does not seem to be going too far. It is still very common to see the "We'll see when I get hacked" mentality still lingering among site owners who finally risk losing a lot of money and also the trust of their customers. Anybody with the interest to research this matter will see how even individuals claiming to be security experts feel comfortable to state that XSS is over-rated and cannot really be used to achieve serious results on a web application. However further research will also prove that statistical figures speak for themselves, and those same statistics keep growing at a rate which will eventually overcast the claims of those incredulous "experts".

Preventing Cross Site Scripting Attacks

The purpose of this article is define Cross Site Scripting attacks and give some practical examples. Preventing XSS attacks requires diligence from the part of the programmers and the necessary security testing. You can learn more about preventing cross-site scripting attacks here.

Scan your site for XSS with the Trial Edition of Acunetix WVS.

Acunetix Web Vulnerability Scanner Trial Edition offers the functionality for anyone who wants to test their own application for Cross Site Scripting. Acunetix encourages all site owners and developers to visithttp://www.acunetix.com/vulnerability-scanner/download/ and to download the Trial Edition of Acunetix WVS. This Trial Edition will scan any web application for XSS and it will also reveal all the essential information related to it, such as the vulnerability location and remediation techniques. Scanning for XSS is normally a quick exercise (depending on the size of the application) and indeed can surprise all those who really wish to see where their web site stands from a security point of view.

Web Site Security Center: Check & Implement Web Site Security

Web security is the most overlooked aspect of securing data. Acunetix Web Site Security Center offers a series of articles and whitepapers on web security, a web application security blog and up to date news on web security. Also, information on the latest website security concepts and the most important web attacks, such as SQL injection & Cross site scripting.

In addition to explaining how website security attacks work, the Web Security Center also provides information on how to find and fix these web security vulnerabilities.

Directory Traversal Attacks

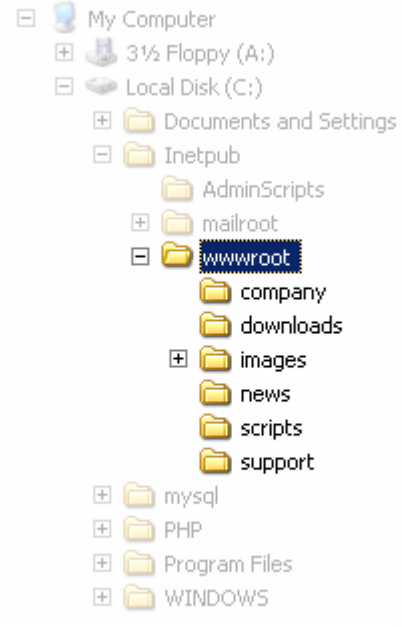
What is a Directory Traversal Attack?

Properly controlling access to web content is crucial for running a secure web server. Directory Traversal is an HTTP exploit which allows attackers to access restricted directories and execute commands outside of the web server's root directory.

Web servers provide two main levels of security mechanisms:

- Access Control Lists (ACLs)
- Root directory

An Access Control List is used in the authorization process. It is a list which the web server's administrator uses to indicate which users or groups are able to access, modify or execute particular files on the server, as well as other access rights.



The root directory is a specific directory on the server file system in which the users are confined. Users are not able to access anything above this root. For example: the default root directory of IIS on Windows is C:\Inetpub\wwwroot and with this setup, a user does not have access to C:\Windows but has access to C:\Inetpub\wwwroot\news and any other directories and files under the root directory (provided that the user is authenticated via the ACLs).

The root directory prevents users from accessing sensitive files on the server such as cmd.exe on Windows platforms and the passwd file on Linux/UNIX platforms.

This vulnerability can exist either in the web server software itself or in the web application code. In order to perform a directory traversal attack, all an attacker needs is a web browser and some knowledge on where to blindly find any default files and directories on the system.

ISSN: 2277-9655
Impact Factor: 1.852

What an Attacker can do if your Website is Vulnerable

With a system vulnerable to Directory Traversal, an attacker can make use of this vulnerability to step out of the root directory and access other parts of the file system. This might give the attacker the ability to view restricted files, or even more dangerous, allowing the attacker to execute powerful commands on the web server which can lead to a full compromise of the system.

Depending on how the website access is set up, the attacker will execute commands by impersonating himself as the user which is associated with "the website". Therefore it all depends on what the website user has been given access to in the system.

Example of a Directory Traversal Attack via Web Application Code

In web applications with dynamic pages, input is usually received from browsers through GET or POST request methods. Here is an example of a GET HTTP request URL:

http://test.webarticles.com/show.asp?view=oldarchive.html

With this URL, the browser requests the dynamic page show.asp from the server and with it also sends the parameter "view" with the value of "oldarchive.html". When this request is executed on the web server, show.asp retrieves the file oldarchive.htm from the server's file system, renders it and then sends it back to the browser which displays it to the user. The attacker would assume that show.asp can retrieve files from the file system and sends this custom URL:

http://test.webarticles.com/show.asp?view=../../../../../Windows/system.ini

This will cause the dynamic page to retrieve the file system.ini from the file system and display it to the user. The expression ../ instructs the system to go one directory up which is commonly used as an operating system directive. The attacker has to guess how many directories he has to go up to find the Windows folder on the system, but this is easily done by trial and error.

Example of a Directory Traversal Attack via Web Server

Apart from vulnerabilities in the code, even the web server itself can be open to directory traversal attacks. The problem can either be incorporated into the web server software or inside some sample script files left available on the server.

The vulnerability has been fixed in the latest versions of web werver software, but there are web servers online which are still using older versions of IIS and Apache which might be open to directory traversal attacks. Even tough you might be using a web werver software version that has fixed this vulnerability, you might still have some sensitive default script directories exposed which are well known to hackers.

For example, a URL request which makes use of the scripts directory of IIS to traverse directories and execute a command can be:

http://server.com/scripts/..%5c../Windows/System32/cmd.exe?/c+dir+c:\

The request would return to the user a list of all files in the C:\ directory by executing the cmd.exe command shell file and run the command "dir c:\" in the shell. The %5c expression that is in the URL request is a web server escape code which is used to represent normal characters. In this case %5c represents the character "\".

Newer versions of modern web server software check for these escape codes and do not let them through. Some older versions however, do not filter out these codes in the root directory enforcer and will let the attackers execute such commands.

How to Check for Directory Traversal Vulnerabilities

The best way to check whether your web site & applications are vulnerable to Directory Traversal attacks is by using a Web Vulnerability Scanner. A Web Vulnerability Scanner crawls your entire website and automatically checks for Directory Traversal vulnerabilities. It will report the vulnerability and how to easily fix it.. Besides Directory Traversal vulnerabilities a web application scanner will also check for SQL injection, Cross site scripting & other web vulnerabilities.

Acunetix Web Vulnerability Scanner scans for SQL Injection, Cross Site Scripting, Google Hacking and many more vulnerabilities. Download the trial version of Acunetix WVS.

Preventing Directory Traversal Attacks

First of all, ensure you have installed the latest version of your web server software, and sure that all patches have been applied.

Secondly, effectively filter any user input. Ideally remove everything but the known good data and filter meta characters from the user input. This will ensure that only what should be entered in the field will be submitted to the server.

Check if your Website is Vulnerable to Attack with Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner ensures website security by automatically checking for SQL Injection, Cross Site Scripting, Directory Traversal and other vulnerabilities. It checks password strength on authentication pages and automatically audits shopping carts, forms, dynamic content and other web applications. As the scan is

ISSN: 2277-9655
Impact Factor: 1.852

being completed, the software produces detailed reports that pinpoint where vulnerabilities exist.

Ajax security: Are AJAX Applications Vulnerable to Hack Attacks?

AJAX and JavaScript: The Technologies Explained

Fuelled by the increased interest in Web 2.0, AJAX (Asynchronous JavaScript Technology and XML) is attracting the attention of businesses all round the globe.

One of the main reasons for the increasing popularity of AJAX is the scripting language used – JavaScript (JS) which allows for a number of advantages including: dynamic forms to include built-in error checking, calculation areas on pages, user interaction for warnings and getting confirmations, dynamically changing background and text colours or "buttons", reading URL history and taking actions based on it, open and control windows, providing different documents or parts based on user request (i.e., framed vs. non-framed).

AJAX is not a technology; rather, it is a collection of technologies each providing robust foundations when designing and developing web applications:

- **XHTML or HTML and Cascading Style Sheets (CSS)** providing the standards for representing content to the user.
- **Document Object Model (DOM)** that provides the structure to allow for the dynamic representation of content and related interaction. The DOM exposes powerful ways for users to access and manipulate elements within any document.
- **XML and XSLT** that provide the formats for data to be manipulated, transferred and exchanged between server and client.
- **XML HTTP Request:** The main disadvantages of building web applications is that once a particular webpage is loaded within the user's browser, the related server connection is cut off. Further browsing (even) within the page itself requires establishing another connection with the server and sending the whole page back even though the user might have simply wanted to expand a simple link. XML HTTP Request allows asynchronous data retrieval or ensuring that the page does not reload in its entirety each time the user requests the smallest of changes.
- **JavaScript (JS)** is the scripting language that unifies these elements to operate effectively together and therefore takes a most significant role in web applications.

As such, AJAX is meant to increase interactivity, speed, and usability. The technologies have prompted a richer and friendly experience for the user as web applications are designed to imitate 'traditional' desktop applications including Google Docs and Spreadsheets, Google Maps and Yahoo! Mail.

At the start of a web session, instead of loading the requested webpage, an AJAX engine written in JS is loaded. Acting as a "middleman", this engine resides between the user and the web server acting both as a rendering interface and as a means of communication between the client browser and server.

The difference which this functionality brings about is instantly noticeable. When sending a request to a web server, one notices that individual components of the page are updated independently (asynchronous) doing away with the previous need to wait for a whole page to become active until it is loaded (synchronous).

Imagine webmail – previously, reading email involved a variety of clicks and the sending and retrieving of the various frames that made up the interface just to allow the presentation of the various emails of the user. This drastically slowed down the user's experience. With asynchronous transfer, the AJAX application completely eliminates the "start-stop-start-stop" nature of interaction on the web – requests to the server are completely transparent to the user.

Another noticeable benefit is the relatively faster loading of the various components of the site which was requested. This also leads to a significant reduction in bandwidth required per request since the web page does not need to reload its complete content.

Other important benefits brought about by AJAX coded applications include: insertion and/or deletion of records, submission of web forms, fetching search queries, and editing category trees - performed more effectively and efficiently without the need to request the full HTML of the page each time.

AJAX Vulnerabilities

Although a most powerful set of technologies, developers must be aware of the potential security holes and breeches to which AJAX applications have (and will) become vulnerable.

According to Pete Lindstrom, Director of Security Strategies with the Hurwitz Group, Web applications are the most vulnerable elements of an organization's IT infrastructure today. An increasing number of organizations (both for-profit and not-for-profit) depend on Internet-based applications that leverage the power of AJAX. As this group of technologies becomes more complex to allow the depth and functionality discussed, and, if organizations do not secure their web applications, then security risks will only increase.

ISSN: 2277-9655
Impact Factor: 1.852

Increased interactivity within a web application means an increase of XML, text, and general HTML network traffic. This leads to exposing back-end applications which might have not been previously vulnerable, or, if there is insufficient server-side protection, to giving unauthenticated users the possibility of manipulating their privilege configurations.

There is the general misconception that in AJAX applications are more secure because it is thought that a user cannot access the server-side script without the rendered user interface (the AJAX based webpage). XML HTTP Request based web applications obscure server-side scripts, and this obscurity gives website developers and owners a false sense of security – obscurity is not security. Since XML HTTP requests function by using the same protocol as all else on the web (HTTP), technically speaking, AJAX-based web applications are vulnerable to the same hacking methodologies as 'normal' applications.

Subsequently, there is an increase in session management vulnerabilities and a greater risk of hackers gaining access to the many hidden URLs which are necessary for AJAX requests to be processed.

Another weakness of AJAX is the process that formulates server requests. The Ajax engine uses JS to capture the user commands and to transform them into function calls. Such function calls are sent in plain visible text to the server and may easily reveal database table fields such as valid product and user IDs, or even important variable names, valid data types or ranges, and any other parameters which may be manipulated by a hacker.

With this information, a hacker can easily use AJAX functions without the intended interface by crafting specific HTTP requests directly to the server. In case of cross-site scripting, maliciously injected scripts can actually leverage the AJAX provided functionalities to act on behalf of the user thereby tricking the user with the ultimate aim of redirecting his browsing session (e.g., phishing) or monitoring his traffic.

JavaScript Vulnerabilities

Although many websites attribute their interactive features to JS, the widespread use of such technology brings about several grave security concerns.

In the past, most of these security issues arose from worms either targeting mailing systems or exploiting Cross Site Scripting (XSS) weaknesses of vulnerable websites. Such self-propagating worms enabled code to be injected into websites with the aim of being parsed and/or executed by Web browsers or e-mail clients to manipulate or simply retrieve user data.

As web-browsers and their technological capabilities continue to evolve, so does malicious use reinforcing the old and creating new security concerns related to JS and AJAX. This technological advancement is also occurring at a time when there is a significant shift in the ultimate goal of the hacker whose primary goal has changed from acts of vandalism (e.g., website defacement) to theft of corporate data (e.g., customer credit card details) that yield lucrative returns on the black market.

XSS worms will become increasingly intelligent and highly capable of carrying out dilapidating attacks such as widespread network denial of service attacks, spamming and mail attacks, and rampant browser exploits. It has also been recently discovered that it is possible to use JS to map domestic and corporate networks, which instantly makes any devices on the network (print servers, routers, storage devices) vulnerable to attacks.

Ultimately such sophisticated attacks could lead to pinpointing specific network assets to embed malicious JS within a webpage on the corporate intranet, or any AJAX application available for public use and returning data.

The problem to date is that most web scanning tools available encounter serious problems auditing web pages with embedded JS. For example, client-side JS require a great degree of manual intervention (rather than automation).

Summary and Conclusions

The evolution of web technologies is heading in a direction which allows web applications to be increasingly efficient, responsive and interactive. Such progress, however, also increases the threats which businesses and web developers face on a daily basis.

With public ports 80 (HTTP) and 443 (HTTPS) always open to allow dynamic content delivery and exchange, websites are at a constant risk to data theft and defacement, unless they are audited regularly with a reliable web application scanner. As the complexity of technology increases, website weaknesses become more evident and vulnerabilities more grave.

The advent of AJAX applications has raised considerable security issues due to a broadened threat window brought about by the very same technologies and complexities developed. With an increase in script execution and information exchanged in server/client requests and responses, hackers have greater opportunity to steal data thereby costing organizations thousands of dollars in lost revenue,

ISSN: 2277-9655
Impact Factor: 1.852

severe fines, diminished customer trust and substantial damage to your organization's reputation and credibility.

The only solution for effective and efficient security auditing is a vulnerability scanner which automates the crawling of websites to identify weaknesses. However, without an engine that parses and executes JavaScript, such crawling is inaccurate and gives website owners a false sense of security. Read about the JavaScript engine of Acunetix.

Google Hacking

What is Google Hacking?

Google hacking is the term used when a hacker tries to find exploitable targets and sensitive data by using search engines. The Google Hacking Database (GHDB) is a database of queries that identify sensitive data. Although Google blocks some of the better known Google hacking queries, nothing stops a hacker from crawling your site and launching the Google Hacking Database queries directly onto the crawled content.

More information about Google hacking can be found on: http://www.informit.com/articles/article.aspx?p=170880.

What a Hacker can do if your Website is Vulnerable

Information that the Google Hacking Database identifies:

- Advisories and server vulnerabilities
- Error messages that contain too much information
- Files containing passwords
- Sensitive directories
- Pages containing logon portals
- Pages containing network or vulnerability data such as firewall logs.

How to Check for Google Hacking Vulnerabilities

The easiest way to check whether your web site & applications have Google hacking vulnerabilities, is to use a Web Vulnerability Scanner. A Web Vulnerability Scanner scans your entire website and automatically checks for pages that are identified by Google hacking queries. (Note: Your web vulnerability scanner must be able to launch Google hacking queries).

Acunetix Web Vulnerability Scanner includes an offline copy of the Google Hacking Database (GHDB), allowing to identify pages which can be exploited using search engines.

Preventing Google Hacking Attacks

Verify the all pages identified by Google hacking queries. Since these pages generally provide information which should not be found on your web site, you should generally remove such pages from your site. If these pages are required by your site, arrange the page so that it is not indexed by search engines and arrange the wording so that it is not easy to detect by Google hacking queries.

Email Injection

Description

This script is possibly vulnerable to Email injection attacks. Email injection is a security vulnerability that allows malicious users to send email messages using someone else's server without prior authorization. A malicious spammer could use this tactic to send large numbers of messages anonymously.

Impact

One of the input parameters of the [bold]mail[/bold] function are not properly validated. Therefore, it's possible for a remote attacker to inject custom SMTP headers. For example, an attacker can inject additional email recipients and use the script for sending spam.

Recommendation

You need to restrict CR(0x13) and LF(0x10) from the user input. Check references for more information about fixing this vulnerability.

References

Email Injection

PHP mail() Header Injection Through Subject and To Parameters

Which Vulnerabilities does Acunetix Web Vulnerability Scanner Check for?

Acunetix Web Vulnerability Scanner automatically checks for the following vulnerabilities, among others:

Web Server Configuration Checks

- Checks for Web Servers Problems – Determines if dangerous HTTP methods are enabled on the web server (e.g. PUT, TRACE, DELETE)
- Verify Web Server Technologies
- Vulnerable Web Servers
- Vulnerable Web Server Technologies – such as “PHP 4.3.0 file disclosure and possible code execution.

Parameter Manipulation Checks

- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- SQL Injection
- Code Execution
- Directory Traversal
- HTTP Parameter Pollution
- File Inclusion
- Script Source Code Disclosure
- CRLF Injection
- Cross Frame Scripting (XFS)
- PHP Code Injection

ISSN: 2277-9655
Impact Factor: 1.852

- XPath Injection
- Path Disclosure (Unix and Windows)
- LDAP Injection
- Cookie Manipulation
- Arbitrary File creation (AcuSensor Technology)
- Arbitrary File deletion (AcuSensor Technology)
- Email Injection (AcuSensor Technology)
- File Tampering (AcuSensor Technology)
- URL redirection
- Remote XSL inclusion
- DOM XSS
- MultiRequest Parameter Manipulation
- Blind SQL/XPath Injection
- Input Validation
- Buffer Overflows
- Sub-Domain Scanning

File Checks

- Checks for Backup Files or Directories - Looks for common files (such as logs, application traces, CVS web repositories)
- Cross Site Scripting in URI
- Checks for Script Errors

File Uploads

- Unrestricted File uploads Checks

Directory Checks

- Looks for Common Files (such as logs, traces, CVS)
- Discover Sensitive Files/Directories
- Discovers Directories with Weak Permissions
- Cross Site Scripting in Path and PHPSESSID Session Fixation.
- Web Applications
- HTTP Verb Tampering

Text Search

- Directory Listings
- Source Code Disclosure
- Check for Common Files
- Check for Email Addresses
- Microsoft Office Possible Sensitive Information
- Local Path Disclosure
- Error Messages
- Trojan Shell Scripts (such as popular PHP shell scripts like r57shell, c99shell etc)

Weak Password Checks

- Weak HTTP Passwords
- Authentication attacks
- Weak FTP passwords

Google Hacking Database (GHDB)

- Over 1200 Google Hacking Database Search Entries

Port Scanner and Network Alerts

- Finds All Open Ports on Servers
- Displays Network Banner of Port
- DNS Server Vulnerability: Open Zone Transfer
- DNS Server Vulnerability: Open Recursion
- DNS Server Vulnerability: Cache Poisoning
- Finds List of Writable FTP Directories
- FTP Anonymous Access Allowed
- Checks for Badly Configured Proxy Servers
- Checks for Weak SNMP Community Strings
- Finds Weak SSL Cyphers

Conclusions

Web applications achieve out to a larger, less-trusted user base than legacy client-server applications, and yet they are more defenseless to attacks. Many companies are starting to take initiatives to prevent these types of break-ins. Code reviews, extensive penetration testing, and interruption detection systems are just a few ways that companies are battling a growing problem. Unfortunately, most of the solutions available today are using negative security logic (working with a list of attacks and trying to prevent against them). Negative security logic solutions can prevent known, generalized attacks, but are ineffective against the kind of targeted, malicious hacker activity outlined in this paper. In this paper, I have demonstrated some common web application vulnerabilities, their countermeasures and their criticality. If there is a consistent message among each of these attacks, the key to moderate these vulnerabilities is to disinfect user's input before processing it.

References

[1] OWASP Top 10 Web Application Vulnerabilities.http:// www.applicure.com/blog/owasp-top-10-2010
[2] MITRE. Common vulnerabilities and exposures. http:// cve.mitre.org/cve/, 2007
[3] http://www.acunetix.com/websitesecurity/xss/
[4] http://www.codeproject.com/Articles/9378/SQL-Injection-Attacks-and-Some-Tips-on-How- to-Prev
[5] http://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf
[6] https://www.owasp.org/index.php/Format_string_attack
[7] https://www.golemtechnologies.com/articles/prevent-xss