Lecture Notes on Web Vulnerabilities

Ton van Deursen and Saša Radomirović

March 23, 2010

1 Web Security

1.1 Introduction

Today's web applications are often vulnerable to attacks. There are several reasons why many web applications contain vulnerabilities. Programmers are generally not aware of good security practices, security is hard to get right, security is inconvenient. Moreover, there is no direct return on investment on secure applications. There is also the predominant thinking that one web applications vulnerability does not have the capability of inflicting major damage.

Table 1 gives a non-exhaustive overview of today's web vulnerabilities. Web vulnerabilities can be classified by the victim (either a remote system, or a local system) and the target (either data, or a complete system).

It may not come as a surprise that web vulnerabilities cannot be prevented completely. However, one must be aware of the mistakes and vulnerabilities that are present, and adapt his web applications to minimize the chance that they are compromised.

The subsequent sections present common web vulnerabilities in today's web applications. The common divisor of these vulnerabilities is that they rely on input from the attacker that is executed. Therefore, a web application should always sanitize input provided by the user.

Table 1: Classification of web vulnerabilities

	Data	Computer System
Remote System	SQL injection	Apache Buffer Overf.
Local Computer	Resp.-splitting XSS, Phishing cookie pois. CSRF	Email attachments, WMF/TIFF Vulner. Trojans, Malware

To understand why input validation is necessary, it is essential to understand the communication between a browser and a remote server. The communication starts by the browser issuing an HTTP GET command to the server:

```
 GET /members/sasa/fun/ HTTP/1.1
 Host: satoss.uni.lu
```

The server then responds with:

```
HTTP/1.1 200 OK
Date: Wed, 31 Oct 2007 11:27:42 GMT
Server: Apache/1.3.33 Ben-SSL/1.55 (Unix) DAV/1.0.3 PHP/4.3.10
Last-Modified: Tue, 30 Oct 2007 13:27:49 GMT
ETag: "6e000e-7f-47273155"
Accept-Ranges: bytes
Content-Length: 127
Content-Type: text/html

<HTML>
<HEAD><TITLE>Demo Website</TITLE></HEAD>
<BODY onload='alert("Printer on Fire!");'>
<H1>Hello World</H1>
</BODY></HTML>
```

The response is interpreted by the browser in three steps:

1. The browser reads the header:

   ```
   HTTP/1.1 200 OK
   Date: Wed, 31 Oct 2007 11:27:42 GMT
   Server: Apache/1.3.33 Ben-SSL/1.55 (Unix) DAV/1.0.3 PHP/4.3.10
   Last-Modified: Tue, 30 Oct 2007 13:27:49 GMT
   ETag: "6e000e-7f-47273155"
   Accept-Ranges: bytes
   Content-Length: 127
   Content-Type: text/html
   ```

2. The browser reads and displays the HTML code:

   ```
   <HTML>
   <HEAD><TITLE>Demo Website</TITLE></HEAD>
   <BODY onload='alert("Printer on Fire!");'>
   <H1>Hello World</H1>
   </BODY></HTML>
   ```

3. The browser interprets and executes scripts in the content:

   ```
   alert("Printer on Fire!");
   ```

1.2 Cross-site scripting (XSS)

Cross site scripting allows attackers to execute scripts in the user's browser. This may result in hijacked user sessions, defaced web sites, hostile content in web sites, phishing attacks, and hostile browser take-overs. The malicious script is usually JavaScript, but may be any scripting language that is interpreted by the victim's browser. Cross site scripting flaws occur whenever an application takes data that originates from the user and does not validate the user input.

The attacks are usually implemented in JavaScript, which is a powerful scripting language. Using JavaScript allows attackers to manipulate any aspect of a rendered page, such as adding a login box which forwards credentials to a hostile site. Another possibility is to perform a phishing attack on the user (see Section 1.6). The evolution of JavaScript malware, finding its way into more and more attackers toolboxes, has made finding and fixing this vulnerability more vital than ever.

XSS attacks can be protected against by validation of all incoming data ("whitelist validation") and appropriate encoding of all output data. Validation allows the detection of attacks, and encoding prevents any successful script injection from running in the browser.

1.3 Injection flaws

Injection flaws are common in web applications. There are many types of injections, for instance SQL, LDAP, XPath, XSLT, XML, and OS Command injections. Injection flaws occur when user-supplied data is passed to an interpreter as part of a command or query. The most common type of injection flaw allows for SQL injection. Structured Query Language (SQL), is a computer language designed for the retrieval and management of data in relational database management systems, database schema creation and modification, and database object access control management.

A typical SQL query looks like

```
SELECT field FROM table WHERE condition;
```

For instance, if we have a table of all students and their grades, we might want to issue the following command

```
SELECT student FROM allstudents WHERE grade < 10;
```

to list all those students whose grade is below 10.

A classic way to verify a username and password provided by a user is to search the database for that username/password. If the database returns a non-empty set, the user has provided a valid pair, and passes authentication. A common way to do this is to access the database from a PHP script using the following code:

```
$sql    = "SELECT * FROM table WHERE username = '" . $user . "'
          AND password='" . $password . "';";
$result = mysql_query($sql);
```

A simple SQL injection that would allow access to the website could be to provide `admin` as username and `anypassword' OR '1'='1` as password. Giving such values would result in the following SQL query being executed:

```
$sql    = "SELECT *
           FROM table
           WHERE username = 'admin'
           AND password='anypassword' OR '1'='1';
```

The latter part of the injection (`' OR '1'='1`) ensures that the WHERE-clause is always satisfied. This would allow the attacker to pass the authentication mechanism and login to the system without knowing a valid username/password combination.

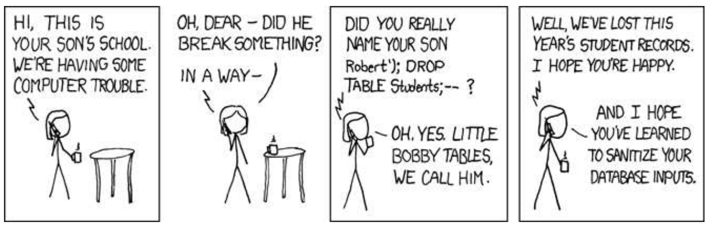
Although passing the authentication scheme is undesired, much more damage can be done by exploiting the fact SQL queries can be composed using a semicolon `;`. For instance, the following input for *password* would add a user "Ton" to the database with password "12345".

```
anypassword' OR '1'='1'; INSERT INTO table VALUES ('Ton','12345');--
```

Since the values of `$user` and `$password` are entirely under the control of the attacker, any malicious input can be passed on to the database.

Any application that allows users to input data which is subsequently passed on to another system is potentially vulnerable to injection flaws. The main solution to this problem is to sanitize the data (for instance by disallowing certain characters in user input), before passing it on to another system.

SQL Injection has been at the center of some of the largest credit card and identity theft incidents. Today's backend website databases store highly sensitive information, making them a natural, attractive target for malicious hackers. Names, addresses, phone numbers, passwords, birth dates, intellectual property, trade secrets, encryption keys and often much more could be vulnerable to theft. With a few well-placed quotes, semi-colons and commands entire databases could fall into the wrong hands.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

1.4 Cross-site request forgery (CSRF)

Cross-site request forgery is not a new attack, but it is gaining in popularity. A CSRF-attack forces a logged-on browser to send a request to any web site of the attacker's choosing, which then performs the chosen action on behalf of the victim. The attack works by including a link into a web page that accesses another web site to which the user has logged in. The attack allows attackers to make an HTTP request to e.g. the victim's bank, blog, or web mail. The following tag in any web page viewed by the victim will generate a request to the web page that logs out the user:

```
 <img src="http://www.example.com/logout.php">
```

If an online bank allowed its application to process requests without explicitly verifying the user's credentials, the following code asks for a transfer of funds:

```
<img src="http://www.example.com/transfer.php?toIBAN=123
 &toBIC=456&amount=500">
```

Although XSS flaws are not required for CSRF attacks to work, any web site that is vulnerable to XSS attacks is also vulnerable to CSRF attacks. When building defense against CSRF attacks, eliminating XSS vulnerabilities in a web site is essential, since such flaws can be used to get around CSRF defense mechanisms.

The only protection against CSRF attacks is that applications do not rely on credentials that are automatically submitted without the user's knowledge.

1.5 Information leakage

Applications can unintentionally leak information about their configuration, internal working, or violate privacy through a variety of application problems. Examples of such information are developer comments, user information, internal IP addresses, source code, software version numbers. While information leakage itself does not have to be a problem, the information may be used by an attacker to launch, or even automate more powerful attacks. Examples of information leakage:

- Detailed error handling such as failed SQL statements, or other debugging information.
- Functions that display different results based on different inputs. For example a failed login should display the same message (e.g. `"login failed"`) irrelevant of whether username, password, or both were incorrect.

1.6 Phishing attacks

Phishing is an attempt to criminally and fraudulently acquire sensitive information by masquerading as a trustworthy entity in an electronic communication. The first phishing technique was described in detail in 1987. Phishing was first applied on a large scale in the early 90s. While early phishing attempts were targeted at large groups of users, recent phishing attempts are becoming more and more targeted at individuals using additional information of the user that is attacked.

In a phishing attempt, the attacker tries to spoof a legitimate web site, with the aim of obtaining data the victim would normally supply to the original web site. This data could be usernames and passwords or credit card information.

The first part of a phishing attack is to fool the user into visiting a malicious web site. Common techniques are to make the anchor text for a link appear to be valid, while the link actually goes to the malicious web site. Another common technique is to use misspelled URL's, the use of subdomains (e.g. `http://www.bcee.lu.com` or `http://www.bcee.com.lu` when the attacker possesses resp. the domains `lu.com` or `com.lu`), or the use of IP addresses instead of URL's. Another, more dangerous attack involves poisoning the DNS cache of the user. If the attacker succeeds in poisoning the DNS cache of the user, the user will be redirected to the wrong server.

After fooling the user into visiting the malicious web site, the attacker might try to alter the address bar. He can either use JavaScript to alter the address bar, place a picture of a legitimate URL over the address bar of close the original address bar and open a new one with the URL of the legitimate URL.

In general phishing attacks are performed through e-mail or instant messaging communication.

1.7 Browsers compromising privacy

Recently, two examples have shown that websites with CSS can be used to violate the privacy of the user visiting the website. The website uses the fact that the browser stores a history of visited pages. Depending on whether a user has visited a page that is linked to from a browser, the color of the hyper link is changed. The following two web sites show whether a user visited a page or not:

```
http://ha.ckers.org/weird/CSS-history-hack.html
http://ha.ckers.org/weird/CSS-history.cgi
```

The second web site shows how a hostile web site may abuse this information: it includes an image iff the user has visited the page. In doing this, the browser essentially tells the web site which pages have been visited.

1.8 Predictable Resource Location (PRL)

Over time, many pages on a website become unlinked, orphaned, and forgotten - especially on websites experiencing a high rate of content and/or code updates. These Web pages sometimes contain payment logs, software backups, post dated press releases, debug messages, source code - nothing, or everything. Normally, the only mechanism protecting the sensitive information within is the predictability of the URL. Automated scanners have become adept at uncovering these files by generating thousands of guesses. However, although a scanner can guess at a filename, it has no contextual reasoning to tell if the data received is sensitive or how valuable it might be. Humans need to make this determination.

1.9 Sources

These notes are among others based on the following:

- OWASP Top 10 Project[1].
- MITRE vulnerability trends[2].

[1]http://www.owasp.org/index.php/OWASP_Top_Ten_Project
[2]http://cwe.mitre.org/documents/vuln-trends/index.html