Internet Security [1]
VU 184.216

Engin Kirda engin@infosys.tuwien.ac.at
Christopher Kruegel chris@auto.tuwien.ac.at

Outline

- Web Application Security, Part I
  - Brief introduction to HTML and Web applications (e.g., scripts)
  - The Top Ten Web application vulnerability risks
- SQL Injections
- Real examples ;-)
- Parameter Injections
- Broken Authentication

News from the Lab

- 205 (!) Registrations
- To date, 158 attempts to solve Challenge 1
  - 148 successes (respect)
- One candidate submitted 40 (!) times and eventually succeeded – brute force solving ;-)
- Challenge 2 will be announced today after the lecture
- Registration ends today (after the lecture)

Web Application Security

- When an organization puts up a web application, they invite everyone to send them HTTP requests.
- Attacks buried in these requests sail past firewalls without notice because they are inside legal HTTP requests.
- Even “secure” websites that use SSL just accept the requests that arrive through the encrypted tunnel without scrutiny.
- **This means that your web application code is part of your security perimeter**!

Web Application Security

- The security issues related to the Web are not new. In fact, some have been well understood for decades.
  - For a variety of reasons, major software development projects are still making these mistakes and jeopardizing not only their customers' security, but also the security of the entire Internet.
  - There is no "silver bullet" to cure these problems. Today's assessment and protection technology is improving, but can currently only deal with a limited sub-set of the issues at best.
  - To address the security issues, organizations will need to change their development culture, train developers, update their software development processes, and use technology where appropriate.

On a typical Web server…

- your host has an open 80/8080 port (firewall)
- following components are running
  - OS
  - Web Server
    - main application (e.g. Apache)
    - plugins
    - servlets
    - scripts (CGI, Perl, ...)

HTTP and Web Application Basics

- All HTTP transactions follow the same general format. Each client request and server response has three parts: the request or response line, a header section, and the entity body. The client initiates a transaction as follows:
  - *GET /index.html?param=value HTTP/1.1*
- After sending the request and headers, the client may send additional data. This data is mostly used by CGI programs using the POST method.
  - Note that for the GET method, the parameters are encoded into the URL

Web Server Scripting

- allows easy implementation of functionality (also for non-programmers – Think: Is this good?)
- Example scripting languages are Perl (e.g., used in the InetSec challenges), Python, ASP, JSP, PHP
- Scripts are installed on the Web server and return HTML as output that is then sent to the client
- Template engines are often used to power Web sites
  - E.g., Cold Fusion, Cocoon, Zope (see TUWIS)
  - These engines often support/use scripting languages

Web Application Example

- Objective: To write an application that accepts a username and password and prints (displays) them
  - First, we write HTML code and use forms

```
<html><body>
<form action="/scripts/login.pl" method="post">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Login" name="login">
</form>
</body></html>
```

Web Application Example 2

- Second, here is the corresponding Perl script that prints the username and password passed to it:

```
#!/usr/local/bin/perl
uses CGI;
$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
…
print "<html><body> Username: $username <br>
Password: $password <br>
</body></html>";
```

OWASP

- The Open Web Application Security Project (www.owasp.org)
  - OWASP is dedicated to helping organizations understand and improve the security of their web applications and web services.
  - The Top Ten vulnerability list was created to point corporations and government agencies to the most serious of these vulnerabilities.
  - Web application security has become a hot topic as companies race to make content and services accessible though the web. At the same time, attackers are turning their attention to the common weaknesses created by application developers.

Top Ten Web Application Vulnerabilities

- Unvalidated Input: Information from web requests is not validated before being used by a web application.
  - Attackers can use these flaws to attack backend components through a web application.
- Broken access control: Restrictions on what authenticated users are allowed to do are not properly enforced.
  - Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions

Top Ten Web Application Vulnerabilities

- Broken authentication and session management: Account credentials and session tokens are not properly protected.
  - Attackers that can compromise passwords, keys, session cookies, or other tokens, can defeat authentication restrictions and assume other users' identities.
- Cross-site scripting (XSS): The web application can be used as a mechanism to transport an attack to an end user's browser.
  - A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.

Top Ten Web Application Vulnerabilities

- Buffer overflows: Web application components in languages that do not properly validate input can be crashed and, in some cases, used to take control of a process.
  - These components can include CGI, libraries, drivers, and web application server components
- Injection flaws: Web applications pass parameters when they access external systems or the local operating system.
  - If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application

Top Ten Web Application Vulnerabilities

- Improper error handling: Error conditions that occur during normal operation are not handled properly.
  - If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server
- Insecure storage: Web applications frequently use cryptographic functions to protect information and credentials.
  - These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.

Top Ten Web Application Vulnerabilities

- Denial of service: Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application
  - Attackers can also lock users out of their accounts or even cause the entire application to fail.
- Insecure configuration management: Having a strong server configuration standard is critical to a secure web application.
  - These servers have many configuration options that affect security and are not secure out of the box.

Unvalidated Input

- Web applications use input from HTTP requests (and occasionally files) to determine how to respond.
  - Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms.
  - Common input tampering attempts include XSS, SQL Injection, hidden field manipulation, parameter injection
- Some sites attempt to protect themselves by filtering out malicious input.
  - Problem: there are so many different ways of encoding information

Unvalidated Input

- A surprising number of web applications use only client-side mechanisms to validate input
  - Client side validation mechanisms are easily bypassed, leaving the web application without any protection against malicious parameters
- How to determine if you are vulnerable?
  - Any part of an HTTP request that is used by a web application without being carefully validated is known as a “tainted” parameter.
  - The simplest way: to have a detailed code review, searching for all the calls where information is extracted from an HTTP request

Unvalidated Input

- How to protect yourself?
  - The best way to prevent parameter tampering is to ensure that all parameters are validated before they are used.
  - A centralized component or library is likely to be the most effective, as the code performing the checking should be all in one place.
- Parameters should be validated against a “positive” specification that defines:
  - Data type (string, integer, real, etc…); Allowed character set; Minimum and maximum length; Whether null is allowed; Whether the parameter is required or not; Whether duplicates are allowed; Numeric range; Specific legal values (enumeration); Specific patterns (regular expressions)

SQL Injections

- Injection flaws allow attackers to relay malicious code through a web application to another system
  - These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL
- SQL injection is a particularly widespread and dangerous form of injection attack
  - To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database.

SQL Injections

- By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database
- The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

Simple SQL Injection Example

- Perl script that looks up *username* and *password:*

```
…
$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
…
$sql_command = "select * from users where
username='$username' and password='$password'";
$sth = $dbh->prepare($sql_command)
…
```

No Validation!

Simple SQL Injection Example 2

- If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:
  - select * from users where username=‘ ‘ and password = ‘‘‘
  - An SQL error message would be generated
- If the user enters (injects): ‘ or username=‘john as the password, the SQL statement in the script would become:
  - select * from users where username=‘ ‘ and password = ‘‘ or username= ‘john‘
  - Hence, a *different* SQL statement has been injected than what was intended by the programmer!

Obtaining Information using Errors

- Errors returned from the application might help the attacker (e.g., ASP – default behavior)
  - Username: ' union select sum(id) from users--
  - Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
    [Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
    /process_login.asp, line 35
- Make sure that you do not display unnecessary debugging and error messages to users.
  - For debugging, it is always better to use log files (e.g., error log).

Not good!

Some SQL Attack Examples

- select * …; insert into user values(“user”,”h4x0r”);
  - Attacker inserts a new user into the database
- The attacker could use “stored procedures” (e.g., in SQL Server)
  - xp_cmdshell()
  - “bulk insert” statement to read any file on the server
  - e-mail data to the attacker’s mail account
  - Play around with the registry settings
- select *… ; drop table SensitiveData;
- Appending “;” character does not work for all databases. Might depend on the driver (e.g., MySQL)

Advanced SQL Injection

- Web applications will often escape the ‘ and “ characters (e.g., PHP).
  - This will prevent most SQL injection attacks… but there might still be vulnerabilities
- Stored procedures (might not be completely safe)
- In large applications, some database fields are not strings but numbers. Hence, ‘ or “ characters not necessary.
- Attacker might still inject strings into a database by using the “char” function (e.g., SQL Server):
  - insert into users values(666,char(0x63)+char(0x65)…)

Second Order SQL Injection

- SQL is injected into an application, but the SQL statement is invoked at a later point in time
  - e.g., Guestbook, statistics page, etc.
- Even if application escapes single quotes, second order SQL injection might be possible
  - Attacker sets user name to: john'--, application safely escapes value to john''– (-- is used for expressing comments in SQL Server)
  - At a later point, attacker changes password (and "sets" a new password for victim ***john)***:
    - update users set password= … where *database_handle*("username")='john'--'

Live demos ;-)

- Some real SQL vulnerabilities

Simple Parameter Injection Example

- Perl script that lists (embeds in HTML) the directory contents by calling the shell *ls* command:

```
…
$query = new CGI;
$directory = $query->param("directory");
#Call the ls command in the shell using back ticks
$directory_contents = `ls $directory`;
print "
<html><body>
$directory
</body></html>";
```

Unvalidated input!

Simple Parameter Injection Example 2

- If the user enters a ; cat /etc/passwd as the directory, she can gain access to the contents of the passwd file as well!
  - The shell command in the script becomes ls ; cat /etc/passwd
- How can such a simple attack be prevented?
  - Do not use shell commands directly in Web scripts
  - Filter out characters such as | ; * > < etc. that have a special meaning for the shell

And now, a little online demo…

- Parameter injection

Discovering “clues“ in HTML code

- Developers are notorious for leaving statements like FIXME's, Code Broken, Hack, etc... inside the source code. Always review the source code for any comments denoting passwords, backdoors, or something doesn't work right.
- Hidden fields (<input type=“hidden“…>) are sometimes used to store temporary values in Web pages. These can be changed with ease (Hidden Field Tampering!)

Broken Authentication and Session Management

- Authentication and session management includes all aspects of handling user authentication and managing active sessions. Authentication is a critical aspect of this process
  - even solid authentication mechanisms can be undermined by flawed credential management functions, including password change, forgot my password, remember my password, account update, and other related functions.
- Development teams frequently underestimate the complexity of designing an authentication and session management scheme that adequately protects credentials in all aspects of the site.

Broken Authentication and Session Management 2

- HTTP does not provide this capability, so web applications must create it themselves. Frequently, the web application environment provides a session capability
  - Many developers prefer to create their own session tokens
  - If the session tokens are not properly protected, an attacker can hijack an active or inactive session and assume the identity of a user.
- How to protect yourself?
  - Careful and proper use of custom or off the shelf authentication and session management mechanisms

Broken (Weak) Session Management Example

- Suppose you are ordering something online. You are registered as user *john*. In the URL, you notice:
  - www.somecompany.com/order?s=john05011978
  - What is *s*? It is probably the session ID…
  - In this case, it is possible to deduce how the session ID is made up...
- Session ID is made up of user name and (probably) the user‘s birthday
  - Hence, by knowing a user ID and a birthday (e.g., a friend of yours), you could hijack someone‘s session ID and order something
- The session ID gives a *false sense of protection*

Broken Access Control

- Developers frequently underestimate the difficulty of implementing a reliable access control mechanism. Many of these schemes were not explicitly designed, but have simply evolved along with the web site.
  - Many of these flawed access control schemes are not difficult to discover and exploit
- One specific type of access control problem: administrative interfaces that allow site administrators to manage a site over the Internet.
  - Prime target for attacks

Broken Access Control 2

- Some specific access control issues:
  - Insecure ID's (should be guessable and no reliance on their secrecy)
  - Forced Browsing Past Access Control Checks (e.g., by using a different path trough a different page)
  - Path Traversal e.g. "../../target_dir/target_file"
  - File Permissions (don't run server with root privileges!)
  - Client Side Caching (sensitive pages should not be cached, e.g., by using HTTP headers and meta-tags)

Conclusion

- In this lecture, we took a first look at “higher-level” security issues.
  - We started with the Web and will continue looking at it next week.
- Now is the time to go and check the Web application code you wrote :-)
- Good luck (and fun ;-)) with Challenge 2.
- See you next week!