Mark Kardash

Professor Carl Eyler

SDEV 460: Software Security Testing

9 April 2024

University of Maryland Global Campus

**Before Development:**

**Outline SDLC basics Read Over Company Policies Read Over Standards**

1. Planning: 1. Go through all policies to 1. Formatting the
2. Define Requirements ensure compliance. code (Style, Naming
3. Perform Cost-Benefit Analysis 2. Review company standards conventions,
4. Create a Schedule to ensure compliance. Naming,
5. Determine Resources Needed Indentations,
6. Allocate Resources Comments, etc.).
7. Design: 2. Addressing Issues:
8. Perform Analysis of Requirements Includes bug
9. Develop Ways to Create the Software tracking, reporting
10. Identify Useful Tools for Development , fixing, and using
11. Implementation: tracking tools.
12. Analyze Requirements Again Developers must
13. Break Coding into Smaller Tasks have efficient
14. Testing (Manual and Automated): methods of
15. Penetration Testing communication,
16. Regression Testing and prioritization
17. Unit testing of tasks).
18. Deployment: 3. Creating Secure
19. Have two separate environments: Code: This involves using best practices to ensure code

is safe from vulnerabilities. These

best practices include: Making

regular security updates,

Build Environment (For developers to code) patching the system,

Production Environment (For customers to use) performing input

1. Package build copy validation, handling
2. Configure environments errors, etc.).
3. Install finalized software 4. Documenting:

Essential for maintenance of the software. Must contain information on

functionality, use, and modification of software.

(Clark, n.d.).

**During Definition and Design**

**Review Requirements Review Design Create and Create**

**and Architecture Review UML and Review**

**Models Threat Models**

1. This involves testing 1. Analyze technical 1. When creating 1. Develop

Requirement assump- documentation a model: threat

tions such as: and models. a. Define scope. scenarios.

1. User management. 2. Test any elements b. Create classes 2. Analyze
2. Authentication in need of testing. c. Determine design and
3. Authorization 3. Identify any potential attributes. architecture
4. Data confidentiality threats or vulnerabilities. d. Add methods. to ensure
5. Integrity 4. Make any changes or e. Demonstrate proper
6. Accountability enhancements necessary relationship mitigation.
7. Session Management for a more secure code between 3. If no
8. Transport Security (ex. Developing central components. mitigation
9. Tiered System Segregation frameworks if needed). took place,
10. Legislative and Standards review and

compliance (Including Privacy, modify design.

Government and Industry Standards).

**During Development**

**Walk Through the Code Perform Code Reviews**

1. Walk through the entire code 1. Checklists for reviewing code:

together with developers and a. Confidentiality, availability, and

system architects. and integrity business requirements.

2. Develop a better understanding b. Various Top 10 checklists and OWASP

of the code, its layout guide.

and the application's functionality. c. Framework or language issues

3. Identify any areas of the code (ex. PHP Scarlett paper, ASP.NET

needing improvement. Microsoft Secure Coding Checklists).

d. Industry requirements (ex. COPPA,

ISO/IEC 27002).

**During Deployment**

**Perform Penetration Testing Test Configuration Management**

1. Penetration testing helps 1. Ensure that no part of the software

ensure that the application configuration is at a default installa-

cannot be breached by attackers tion stage. This will confirm the out-

after being deployed. comes of the penetration tests

about the application being

secure.

**Maintenance and Operations**

**Review Operational Management Perform Health Checks Ensure Change Verification**

1. Develop a plan for managing 1. Perform a check every 1. Record

application operations and infra- month or quarter, testing all new

structure. things like: changes

2. Inform, educate, and train developers a. Authorization made to

on how to follow above plan. b. Authentication the software,

c. Data Integrity. to ensure that

the application

has remained secure.

**Tests and Security Controls**

**Task 1: Fingerprint Web Server (OTG-INFO-002) Research**

Based on the response received from fingerprinting, the software under examination is Version 2.4.7 of an Apache server, produced by the vendor Ubuntu. Details on the web were scarce, but after some time, I was able to find information not only on the vendor and release, but of its detected and fixed vulnerabilities.

The Vendor:

Established in 1999, the Apache Software Foundation is a non-profit software company supported by volunteers, relying on corporate sponsors and individual donations for funding. Being one of the largest companies in the industry, its standards for software development have been widely accepted and embraced. Per the statement on their website, they offer “an established framework for intellectual property and financial contributions that enables millions of people around the world to collaborate and deliver freely available software.” (Apache.org, n.d.). They contribute greatly to open-source projects, and have a very specific internal structure: The Foundation’s leadership is elected by its members, and governed on its own (Apache.org, n.d.).

The Release:

An article by SD Times, titled “Apache 2.4.7 Server Released”, and published in November 2013, hails the then-new server as “the latest GA release of the new generation 2.4.x branch of Apache HTTPD” (SD Times, 2013). It claims that the product is the result of 15 years of development, and teases several features it will have. These include WinNT MPM enhancements, significant mod\_proxy\_fcgi updates, and an event MPM of higher performance (SD Times, 2013). The article provides links to lists of changes made to the 2.4 server version, as compared to the 2.2 iteration. The changes include asynchronous support, reduced memory usage, run-time loadable MPM’s, override configuration, and others (The Apache HTTP Server Project, n.d.). To install the 2.4.7 server, users must have Apache Portable Runtime (APR) version 1.5x, and APR-Util version 1.5x, along with upgraded APR libraries. This server builds off of the 2.2 version, and as such, uses recompiled modules from it. Finally, users are advised to have thread-free modules if intending to use version 2.4.7 with any threaded MPM besides Prefork (SD Times, 2013).

Vulnerabilities:

As previously said, in addition to a brief history and characteristics of the vendor and release, I was tasked with finding information on the release’s vulnerabilities. According to the website cvedetails.com, there were a total of 67 vulnerabilities found in the server. However, the vulnerabilities listed were not specifically related to the 2.4.7 version, but rather to the 2.4 server in general. They were of various levels, and went all the way back to previous versions of the software, as far as 2006. However, most of these were fixed in the further iterations, and, for the purpose of this assignment, I took only the ones that have still not been fixed. For this reason, I will be using and discussing only the vulnerabilities included in the 2024 Common Vulnerability Scoring System (CVSS) score. The score for current year contains only the following two vulnerabilities:

CVE-2024-24795: The HTTP server has a response that allows attackers to inject malicious headers into backend applications. This can cause a desynchronization attack.

CVE-2023-38709: The core of the Apache has a problematic input validation, which makes malicious backend/content generators vulnerable to the splitting of HTTP responses.

(CVEdetails.com, n.d.).

Regarding fixes, I first thought about how I would address these vulnerabilities. However, after doing a bit of research, I found out that the only solution to both of them was upgrading to the next version (Which at the time was 2.4.59).

**Task 2: Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005)**

Research on Information Leakage:

According to Goldman (2024), the following types of information are considered sensitive, and, if leaked, could have serious consequences:

1. Financial Information: This is, of course, anything that has to do with the finances of the company or its clients. The category includes things like receipts, bank account details, tax returns, and others. Attackers can use such data for anything from financial schemes to identity theft.
2. User Credentials: Just like financial information, this is one of the first things that come to mind when thinking about sensitive data. The username and password are essentially keys to personal information, and, by obtaining them, hackers can access anything from payment systems to bank accounts and IT infrastructure.
3. Email Addresses: This is yet another key credential that should never end up in an attacker’s hands. Emails are commonly used to execute scams, such as phishing, access control, social engineering, or perform other criminal actions, such as identity theft and unauthorized access to information.
4. Personally Identifiable Information (PII): This is quite obvious, given that, by obtaining another person’s birthdate, address, name, phone number, and other such information, a hacker could very easily commit identity theft, even using their new identity to commit more criminal actions. Things can turn out even worse if a hyper-sensitive piece of information, such as a Social Security Number (SSN) is leaked, as this gives the attacker access to government forms and data.
5. Account Identifiers: This refers to less generic user data, such as a bank account number, that, if leaked, may become useful in brute-force attacks on lucrative databases.
6. Intellectual Property or Trade Secrets: Just as an individual user would want to protect their personal data to avoid a complete downfall of their life, so a company would want to ensure that all of their projects and secrets are in safe hands, to avoid bankruptcy and/or public backlash.
7. Database Structure: This is perhaps one of the most terrifying things for an attacker to know, as, if they are familiar with the structure of a database, they can perform SQL injection attacks, completely crippling entire corporations.

(Goldman, 2024).

Code Review:

In this step, I looked through every single “php” file in the code given, and assessed the sensitive information included within the HTML comments. The first file of the code, authcheck.php, did not seem to have any noteworthy comments at first glance. The only thing I found somewhat suspicious was that the username and email were hardcoded into the app (“wsuser”, “wsemail”). As I have learned from all of my previous software classes, this is to be avoided at all costs. This is why, if taking part in the development of this application, I would ask for the hardcoded credentials to be removed. However, after re-reading the course materials and reviewing the file once more, I did notice a piece of metadata at the very top of the code (<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">). It seems to be setting the type of the content, and is aimed at altering response headers. Because of its structure, this element makes the application vulnerable to SQL injection, in which the attacker will make a malicious input to damage the system. Since it is a piece of metadata, my understanding is that it should be removed from the code before actual production begins. However, because of its importance, I am unsure about whether such action is needed. Besides this, all the other comments in the file seemed to be simply explaining the purpose of each section of the code.

The second file, Cancelit.php, contained the exact same piece of metadata as the first. This made me even more reluctant to stand by my proposed mitigation of removing it. However, continuing my review, I noticed something even more concerning: The username (“wsuser”) and ID (“sid”) were once again hardcoded into the system. This allows hackers to view the credentials, and use them for system access. To avoid hardcoding values, both in this and the previous file, I would retrieve the usernames, passwords, and IDs from a separate “.txt” file.

The third file, CancelSession.php, showcases more of the same problems, with the credentials all being hardcoded. As mentioned before, I would recommend moving all usernames, passwords, and IDs to a separate “.txt” file that would be called and read by the application.

Almost all of the files have the same problem with data hardcoding. However, things start to get particularly bad when we arrive at the ShowSessions.php file, where the session url, access code, and even the tutors’ phone numbers, are practically embedded into the code. Although there weren’t any comments with the “<! --” tag, which would characterize their importance, or any pieces of metadata besides the one at the top, there were many that contained sensitive information, such as first name, last name, email, ID, and even location. They were especially widespread in the “SearchSessions.php”, “ShowSessions.php”, and “tauthcheck.php” files.

In regard to how address such issues, I recommend that developers:

1. Place usernames, passwords, and any other sensitive information, into a separate “.txt” file, as has been previously suggested multiple times throughout this paper.

1. Sparingly use HTML comments, removing them before the application goes into production.

**Task 3: Test HTTP Methods (OTG-CONFIG-006)**

Based on the response, the HTTP methods enabled on the website are: GET, HEAD, POST, OPTIONS, and TRACE. This means that the methods disabled are PUT, DELETE, CONNECT, and PATCH (HTTP Request Methods, n.d.).

According to AppCheck.com (2021), each one of the HTTP methods has security risks associated with it. These risks are outlined below:

1. GET: The vulnerability of the GET method is that any input into it by the user is submitted in a query string within the request URL. Since URLs are frequently stored in caches (ex. CDN caches, web server logs, web proxy, etc.), hackers may be able to retrieve and read sensitive data if any is included in the query string. In addition to this, the GET request can be made with one single click, which opens the possibility of both Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). The former involves the user being tricked into clicking on a link that injects malicious content into the system, while the latter involves an attacker executing unwanted commands or actions in the system.
2. PUT: This method has a great risk involved, as, should hackers get a hold of it, they will be able to harm the system by uploading malicious files onto it. One of such files could be a static HTML page, filled with links leading users to malicious websites. There could also be ASP or PHP files allowing server-side processing, which could, in turn, allow hackers to upload arbitrary code, compromising other users.
3. DELETE: Being in control of the DELETE statement would allow attackers to delete almost anything they want from the server. This can lead to things like a Denial of Service attack, that may even go unnoticed initially, if the hackers manage to delete server-side configuration files. The only way to potentially prevent this is to make a backup of every file.
4. HEAD: The HEAD method is often used in access control attacks. For instance, if a resource has access control only when receiving GET requests, then the hacker will use other requests to perform the attack.
5. POST: This method is one of the least safe HTTP methods. That is because, while sending data to the server, it not only reads, but modifies it as well. Because a POST request is different every time it is made, perpetrators may attempt a timing attack, making multiple POST requests a few seconds apart, which can interrupt any ongoing actions, such as bank transactions.
6. CONNECT: Having access to the CONNECT method would mean that an attacker could connect to the server, or even directly reach a hacking target, using requests that, to the naked eye, look like they came from the server. This way, any security controls can be bypassed.
7. OPTIONS: Although this method is completely safe itself, attackers can use it as a tool to find out what other methods are supported by the server. If these methods are exposed, perpetrators may prepare attacks on them with the help of the OPTIONS method.
8. TRACE: TRACE is a method vulnerable to Cross-Site Tracing attacks. This is when attackers use it to bypass security controls, due to it carrying Cookies as plaintext rather than objects. However, this attack is extremely rare, which makes the TRACE method a very safe one.
9. PATCH: This method has almost the same security risks as PUT, but with two more. As this method is non-idempotent, meaning it changes every time, applying it two times when not necessary may corrupt a file. Additionally, since the PATCH method is able to split and transform data, it can be used to smuggle malicious payloads across various requests to bypass firewalls.

(AppCheck.com, 2021).

References

*Apache HTTP server version 2.4.7 : Security vulnerabilities, CVEs published in 2024*. (n.d.). CVE security vulnerability database. Security vulnerabilities, exploits, references and more. <https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-524064/Apache-Http-Server-2.4.7.html?page=1&year=2024&month=-1&order=1&trc=67&sha=040678d18a00332af27959220ca3e1e62669cbc9>

Clark, H. (n.d.). *The Software Development Life Cycle (SDLC): 7 Phases and 5 Models*. The Product Manager. <https://theproductmanager.com/topics/software-development-life-cycle/>

Goldman, D. (2024, February 26). *Understanding information leakage: What you need to know*. Panorays. Retrieved April 9, 2024, from <https://panorays.com/blog/information-leakage/>

*HTTP request methods*. (n.d.). MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

*HTTP verbs & their security risks*. (2021, July 5). AppCheck | A Complete Enterprise Security Testing Solution -. <https://appcheck-ng.com/http-verbs-security-risks/>

*Overview of new features in Apache HTTP server 2.4*. (n.d.). Welcome! - The Apache HTTP Server Project. <https://httpd.apache.org/docs/trunk/new_features_2_4.html>

*OWASP Testing Guide* (4.0). (2011). Open Web Application Security Project.

SD Times Newswire. (2013, November 26). *Apache HTTP server 2.4.7 released*. SD Times. <https://sdtimes.com/apache-http-server-2-4-7-released/>