**Homework 2**

**OWASP Security Testing Framework Assignment**

**Student**: Patrick Walsh

**School**: University of Maryland Global Campus

Date: 9/14/2021

**Course**: SDEV 460 6380

**Professor**: Dr. Chris Howard

**Part 1: Develop Testing Framework**

You will need to fill in the details for each phase by describing what each phase encompasses and listing three or more activities you will engage in for each phase.

**Before development begins**

1. *Define an SDLC*

Activities:

1. Before development begins, a Software Development Lifecycle (SDLC) should be defined. SDLC should contain security built-in to each phase, rather than added as an afterthought (OWASP, 2021).
2. SDLC should include a development methodology. Popular development methodologies include Rational Unified Process, eXtreme development, Agile development, and traditional waterfall methodologies (OWASP, 2021).
3. *Review Policies and Standards*

Activities:

1. Document appropriate policies and standards. Documentation is key so that development teams have written and well-defined policies and standards in place throughout the SDLC (OWASP, 2021).
2. Develop secure coding standards for the appropriate programming languages involved. Secure coding standards should aim to cover the most common issues related to that language rather than attempt to cover every possible scenario (OWASP, 2021).
3. *Develop Measurement and Metrics Criteria and Ensure Traceability*

Activities:

1. Define which criteria will be measured and define how they will be measured prior to development (OWASP, 2021).
2. Traceability measurements can be defined using a Requirements Traceability Matrix (RTM). The RTM “captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the SDLC” (Hamilton, 2021).

**During definition and design**

1. Review Security Requirements

Activities:

1. Develop test cases to evaluate all assumptions pertaining to the functionality and design of the application from a security standpoint (OWASP, 2021).
2. Define security requirements related to the application. Important security requirements may include user management, authentication, authorization, data confidentiality, and data integrity (OWASP, 2021).
3. Review Design and Architecture

Activities:

1. Document the design and architecture of the application such as application models, textual documents, and other artifacts. Test these application artifacts to ensure that they meet the security requirements as defined in the Review Security Requirements section listed above (OWASP, 2021).
2. Identify and correct any security flaws. Fixing security flaws during the design phase is far more cost effective and efficient compared to fixing them in later development stages (OWASP, 2021).
3. Create and Review UML Models

Activities:

1. Should be conducted after finishing the Review of Design and Architecture phase. Unified Modeling Language (UML) can be used to describe how the application is meant to work (OWASP, 2021).
2. UML models may reveal previously undiscovered security flaws in the application. If these are discovered during this procedure, they should be reviewed and fixed by system architects (OWASP, 2021).
3. UML models “can be used to model dynamic aspects of systems” (Johnstone, 2010, p.21). They can also be used to create various diagrams such as activity diagrams, sequence diagrams, communication diagrams and state diagrams (Johnstone, 2010).
4. Create and Review Threat Models

Activities:

1. Once the design and architecture review are complete and UML models have been developed, a detailed, realistic threat modeling exercise should be conducted (OWASP, 2021).
2. Threat models should contain input from the client and are used to identify threats not found in the previous design review phases (OWASP, 2021).
3. Any threats identified during this procedure should be documented along with mitigation strategies. If a threat is identified that does not have a mitigation strategy, the application architecture and design should be sent back to system architects for review (OWASP, 2021).

**During development**

1. Code Walkthrough

Activities:

1. Begin implementation of code design (OWASP, 2021).
2. Perform high-level code walkthrough with developers, security team, and sometimes system architects to understand the logic and flow of each piece of code (OWASP, 2021).
3. Code Reviews

Activities:

1. Once security team has general understanding of code structure and logic, code can be reviewed in more detail for security issues (OWASP, 2021).
2. Perform static code review against security checklists such as business requirements for availability, confidentiality, and integrity; the OWASP Top 10 Checklists for technical exposures, specific issues relating to the application framework or language, and any industry-specific requirements (OWASP, 2021).
3. Static code review uses automated scanning tools that can run various scans which can include Data Flow Analysis, Control Flow Graph (CFG), Taint Analysis, and Lexical Analysis (Dewhurst, 2021). Static code review scanning tools offer speed, repeatability, and scalability even on large applications; however, they can reduce in false positives as well as false negatives, so any automated scans should be adjudicated by other manual means (Dewhurst, 2021).

**During deployment**

1. Application Penetration Testing

Activities:

1. Conduct vulnerability analysis prior to penetration testing. The vulnerability analysis should include anticipated avenues of attack, countermeasures put in place to prevent exploitation, and impact to the application should various exploits occur (Penetration Testing Execution Standard, 2014).
2. Perform penetration testing of various application components once code implementation and review are complete. Some security flaws may not be discovered during code design, implementation, or review phases, so it is vital to perform this testing to ensure security flaws have not been overlooked (OWASP, 2021).
3. Configuration Management Testing

Activities:

1. Perform penetration of configuration management settings alongside the application itself. Testing should include examining how application is deployed and secured to ensure that no settings may be vulnerable to exploitation (OWASP, 2021).

**Maintenance and operations**

1. Conduct Operational Management Reviews

Activities:

1. Document the review process for how both the application and infrastructure will be handled on the operational side (OWASP, 2021).
2. Conduct Periodic Health Checks

Activities:

1. Perform monthly or quarterly health checks on both the application and infrastructure to ensure that security posture is still maintained and that no new vulnerabilities have been discovered (OWASP, 2021).
2. Software health checks can include both internal and external auditors as well as manual and automated reviews (DNSstuff, 2020).
3. Software audits include checks for International Organization for Standardization (ISO) compliance, HIPPA compliance for handling private medical data, PCI DSS compliance for handling of personal financial data, and SOX Act compliance for proper documentation and handling of investor and company financial data (DNSstuff, 2020).
4. Ensure Change Verification

Activities:

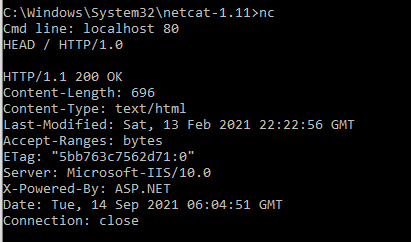
1. Document a change management process for handling how changes to the application or infrastructure are handled (OWASP, 2021).
2. Conduct security assessments of the application after any changes have been made in the Quality Assurance (QA) environment and implemented in the production environment (OWASP, 2021).

**Part 2: Develop Security Controls**

In addition, you will apply part of this framework in the phase “During development” by engaging in three tests/security controls outlined below:

1. **Apply Fingerprint Web Server (OTG-INFO-002) to the testing framework**

To apply fingerprinting to the web server, I downloaded Netcat from <https://eternallybored.org/misc/netcat/>. I then ran the following command from an elevated command prompt to fingerprint the local web server:



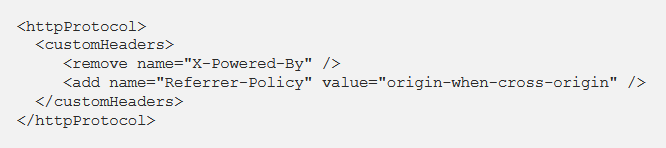
The report listed the web server as a *Microsoft-IIS version 10.0*.

**Report documented vulnerabilities with the web server version and explain how to mitigate these vulnerabilities.**

There are a number of known vulnerabilities associated with *Microsoft-IIS version 10.0.* One vulnerability is unsecure cookies (Mold, 2020). It is recommended that cookies are set to httpOnly in order to secure them (Mold, 2020). This prevents cookies from being accessed by anything other than SSL.

Another documented vulnerability is allowing HTTP connections on port 80 rather than HTTPS connections on port 443 (Mold, 2020). In order to ensure that all websites run through the secure HTTPS port, the web server IIS has a module called the URL Rewrite model to handle these requests. If this module is not listed in the IIS Manager, the module must first be installed (Mold, 2020).

Another documented vulnerability is not having a referrer policy setup (Mold, 2020). This is important because sensitive user information can inadvertently be sent to external websites when following a link if the referrer policy is not properly configured (Mold, 2020). For example, a user logs in and the URL path includes their name or other personal information. To prevent that information from being sent to an external website, a referrer policy header such as origin-when-cross-origin can be added as seen below (Mold, 2020):



1. **Review webpage comments and metadata for information leakage from provided website files (****OTG-INFO-005). What are three or more categories of information that would be considered   
   unacceptable information leakage?**

Some examples of categories of information leakage that could cause security issues are database SQL code snippets, login information such as usernames and passwords, internal IP addresses, and debugging information (Muller, n.d.). The two HTML files tlogin.html and index.html did not contain any comments and did not appear to have any information leakage that fit into one of these categories. However, the tutors.css file did contain a name of one of the application developers on line 4, which could potentially be used for login information. Additionally, the logdata.txt file is stored in plain text which could be a potential security concern since any attacker could look at the log information to find a weakness.

**Recommendation:** Remove the developer’s name from the tutors.css file. Store the log file as a secure configuration file with strong encryption rather than sitting on the website application in plain text.

The dbparms.txt file contains a username and password of what appears to be an administrative user. This information is stored in plain text along with the hostname and database name. This represents a serious security vulnerability since any attacker could use this information to access the application and cause serious damage, steal information, or perform other malicious operations.

**Recommendation:** Do not store this sensitive login information in a plain, unencrypted text file with the application. This information should be stored in an entirely different location such as a secure database with robust encryption.

The emailparms.txt file contains the hostname and port number for the email configuration, along with the email used to send auto-generated replies to users, presumably for administrative functions. Moreover, a parameter for smtpauth is set to false, meaning that authentication is not occurring. All this information is stored in plain text and represents a serious security vulnerability.

**Recommendation:** Do not store this configuration information in a plain, unencrypted text file with the application. This information should be stored in an entirely different location such as a secure database with robust encryption.

The SQLFunctions.php file contains an avenue for a possible SQL injection on line 22. The application builds an SQL query as follows:

**$Myquery = "SELECT firstName, lastName, eMail, tychoName from Students**

**where tychoName='$tname' and eMail='$em'";**

Since the application is not sanitizing user input on this line, it would be possible for an attacker to insert an SQL injection such as an OR statement to make the query always return true.

**Recommendation:** Sanitize user input using a prepared statement. Below is an example of how to construct a prepared statement in PHP (Marin, 2020). Rather than accepting un-sanitized user input to build the query, the code below shows how to build a prepared statement to prevent SQL injections (Marin, 2020):

$delete\_query = 'DELETE FROM students where id = :id';

$prepared\_statement = $pdo->prepare( $delete\_query );

$prepared\_statement->bindParam( 'id', $id );

$prepared\_statement->execute();

$result = $prepared\_statement->rowCount();

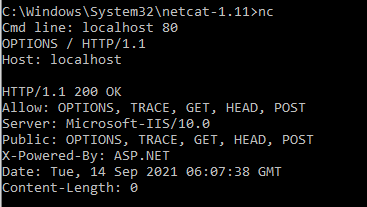
The ShowSessions.php file contains potentially sensitive information and Webex links from lines 193-261. The links take a user to a Webex meeting, and the meeting phone number and access code are stored as plain string information in the file. Having this information public could allow any attacker to access those meeting locations and cause potential damage.

**Recommendation:** Do not stored this information as plain string data. The meeting link, phone number, and access code should be entered as parameters and retrieved from a secure database. The information should not be sitting in plain, unencrypted format for any attacker to access.

**Method of discovery**: manually reviewing source code of website application.

1. **Test HTTP Methods (OTG-CONFIG-006) and document which HTTP methods are available on the website and which pose risks.**

From an elevated command prompt, I ran the following command with Netcat:



The HTTP methods allowed are OPTIONS, TRACE, GET, HEAD, and POST.

**OPTIONS**

While OWASP does not specifically state any security risks with OPTIONS method (n.d.), this method is very powerful because it gives direct information about which communications options are available on the request/response side of the web server. In fact, the OPTIONS method is the method that was used in the screenshot above to retrieve the information about available communication options.

**TRACE**

The TRACE method is designed mainly for debugging purposes, sending a string back to the client of whichever message was sent to the server (OWASP, n.d.). However, it can pose a security risk by allowing an attacker to launch a Cross Site Tracing attack (OWASP, n.d.). This can be done be either leveraging a server-side vulnerability and injecting a hostile JavaScript snippet into the TRACE request, or by leveraging a client-side vulnerability and creating a malicious website that contains a hostile JavaScript request (OWASP, n.d.).

**GET/HEAD**

The GET and HEAD methods are very similar except that the GET method returns a message body in the response whereas the HEAD method does not (Fielding et al, 1999, p. 54). Fielding et al specified (1999) that the GET and HEAD methods should not be used for anything other than information retrieval, adding that these methods ought to be considered safe (p. 51). While OWASP does not list any specific security risks for the GET or HEAD methods (n.d.), OWASP does point out that they could potentially be exploited by an attacker if there are no security constraints preventing non-authenticated users from sending this type of request.

**POST**

The POST method is used to request the server to accept the entity enclosed in a request (Fielding et al, 1999). Along with GET, POST is one of the most common HTTP methods, and OWASP considers (n.d.) code which has been explicitly checked for either a GET or POST method to be safe.

**References**

Dewhurst, R. (2021). *Static Code Analysis Control | OWASP Foundation*. Owasp.Org. https://owasp.org/www-community/controls/Static\_Code\_Analysis

DNSstuff. (2020, October 12). *IT Security Audit: Standards, Best Practices, and Tools - DNSstuff*. Software Reviews, Opinions, and Tips - DNSstuff. https://www.dnsstuff.com/it-security-audit

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999, June). *rfc2616*. Datatracker.Ietf.Org. https://datatracker.ietf.org/doc/html/rfc2616

Hamilton, T. (2021, August 28). *What is Requirements Traceability Matrix (RTM)? Example Template*. Guru99. https://www.guru99.com/traceability-matrix.html

Johnstone, M. N. (2010, November 30). *Threat Modelling with Stride and UML*. Research Online. https://ro.ecu.edu.au/ism/88/

Marin, D. (2020, June 15). *SQL Injection in PHP: Practices to Avoid*. Okta Developer. https://developer.okta.com/blog/2020/06/15/sql-injection-in-php

Mold, D. (2020, May 14). *Fix typical IIS 10 vulnerabilities and ace your security audit*. Medium. https://medium.com/swlh/fix-typical-iis-10-vulnerabilities-and-ace-your-security-audit-19077b94b94a

Muller, A. (n.d.). *Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005) | Owasp Testing Guide v4*. Frontispiece. https://kennel209.gitbooks.io/owasp-testing-guide-v4/content/en/web\_application\_security\_testing/review\_webpage\_comments\_and\_metadata\_for\_information\_leakage\_otg-info-005.html

Online Web Application Security Project (OWASP). (n.d.). *WSTG - v4.1 | OWASP*. Owasp.Org. https://owasp.org/www-project-web-security-testing-guide/v41/4-Web\_Application\_Security\_Testing/02-Configuration\_and\_Deployment\_Management\_Testing/06-Test\_HTTP\_Methods

Online Web Application Security Project (OWASP). (2021). *WSTG - Stable | OWASP*. OWASP.Org. https://owasp.org/www-project-web-security-testing-guide/stable/3-The\_OWASP\_Testing\_Framework/0-The\_Web\_Security\_Testing\_Framework

Penetration Testing Execution Standard. (2014, August 16). *Exploitation - The Penetration Testing Execution Standard*. Pentest-Standard.Org. http://www.pentest-standard.org/index.php/Exploitation