Requirements Based Web Application Security Testing – A Preemptive Approach!

Bhushan B. Gupta

Principal, Gupta Consulting, LLC.

bhushan@bgupta.com https://www.bgupta.com/

Abstract

Most web application security testing efforts are concentrated around penetration testing, which is an art based on hackers' psyche, thought process, and determination to exploit software vulnerabilities. However, does it yield a high level of confidence and sense of security in a developer's mind? The answer is a "maybe" especially when the "bad guy" is obsessed with figuring out new exploits to hack an application. The web application developers must begin to think of building security throughout the software development life cycle (SDLC). Applications are built based upon well-formed customer requirements. Should it not be a best practice then, to build applications upon the fundamental principles of security then harden security from the hackers' perspective?

This paper discusses an approach that aligns the web application security testing with the three basic principles of security; **c**onfidentiality, **i**ntegrity, and **a**vailability (CIA). The approach first establishes the requirements dictated by each element of CIA especially confidentiality as it places the most stringent requirements on an application. Using a simple approach, the paper illustrates the most vulnerable processes in an application, highlighting the test-intensive areas. It then deduces the security controls to derive an acceptance criteria and illustrates thought process to develop a test strategy which spans over both, static and dynamic (traditional testing) code analysis. The paper continues to demonstrate how to apply the DREAD model to prioritize the vulnerabilities found during testing, to facilitate the removal of the most critical vulnerabilities first.

Biography

Bhushan Gupta is passionate about development methods and tools that yield more secure web applications, especially in the agile software development environment. As a researcher, he has a keen interest in understanding and applying fundamental principles and known methodologies to develop dependable solutions. His interests extend to Social Engineering and Attack Surface Analysis. Bhushan worked at Hewlett-Packard for 13 years, in various roles including software quality lead, software process architect, and software productivity manager. He then developed a strong interest in web application security while working as a quality engineer for Nike Inc. Bhushan has been studying various facets of web application security and promoting how to apply common sense approach to build secure solutions. He is a certified Six Sigma Black Belt (HP and ASQ) and an adjunct faculty member at the Oregon Institute of Technology in Software Engineering.

1 Introduction

“Penetration testing” (mostly referred to as Pen Testing) of a web application is an approach in which a test engineer takes on the characteristics of a hacker to exploit application vulnerabilities. Most organizations also include OWASP Top 10 vulnerabilities (OWASP 2013) in their testing scope. While the approach can check a system for vulnerabilities, there are no assurances that the application will not be compromised for any other shortcomings. To really “pen test” an application a test engineer should pursue the approach that hackers take and test the application for all potential hacks. This makes the test metrics a huge task and given that the time to market is short, the organizations simply do not have the resources, both personnel and time, to shake out the application. The applications that are developed for internal use may allow a bit more time for testing.

What if an approach were pursued where the security requirements specific to the application are first established, and then the security controls that best secure the application are designed? Once the requirements and controls are in place, an acceptance criterion is derived that can help establish a robust test plan. It is the same approach taken for testing the software quality and security is a vital component of software quality. One should integrate security as an attribute of quality so that it becomes an intrinsic aspect of the application. Such an approach will provide a known confidence level in security and will be preemptive, as opposed to the reactive practices that are widely used today. This article is devoted to the understanding of security requirements, developing a list of security controls that can provide acceptance criteria for each requirement, and a test strategy that deploys a variety of testing methodologies, thus achieving a higher level of security confidence.

2 Hacking and Pen Testing

From the first impressions, hacking sounds like a disorganized activity without any structure. The fact is that hacking is a step-by-step activity where every proceeding step is based upon the results of the previous steps. It is entirely possible to take a path that will lead to an unsuccessful attempt. Kim (Kim 2015) has explained hacking using the analogy of American football, describing each hacking phase in the terms of the game; before the snap, the drive, the throw, the lateral pass, the onside kick etc. Each phase has a set of activities moving towards the goal. Before the snap, is the discovery phase in which the “bad guy” assesses his target. This phase includes target scanning to gather pertinent information. In the drive phase, the “bad guy” takes all the information gathered in the previous phase and uses these exploits to get his foot in the door. The web application’s vulnerabilities are then manually targeted in the “Throw phase”. In the “Lateral Pass” phase the “bad guy” moves through the network exploiting backdoors. In the “Screen Phase,” the focus shifts to client attacking using social engineering techniques and then the “Onside Kick” is where the bad guy looks for the physical access. This is a bird’s eye view of hacking.

Pen testing follows the same approach. As a matter of fact, Pen testing is carried out from the hacker’s perspective to provide some assurance that the potential vulnerabilities have been examined and any potential exploits have been plugged up. The success of Pen testing depends upon the reconnaissance, the tools used, understanding the attacker profile, and the scope – the types of vulnerabilities included in the scope of testing. Each of these factors requires significant knowledge to execute and gain a comfortable level of confidence. To make things worse, new tool kits are developed for hacking and time is an advantage for hackers since they are not on a schedule, but the application release is. It only makes business sense to deploy alternative approaches to security testing as is done for other attributes of quality to achieve a level of confidence that is desired.

3 A Look into the Past

It will be educational to review a few significant security breaches to understand the root cause behind exploitation. Armerding (Armerding 2017), CSO, has provided a list of the 15 worst data security breaches in the 21st century based upon how security practitioners weigh them. While this discussion is not devoted to each breach, it highlights a few with the potential root causes. This section also cites a malware that have significantly impacted web application security.

3.1 Yahoo (2013-14)

In September 2016, Yahoo reported a security breach that impacted 1.5 Billion user accounts involving two incidents. Yahoo was in a critical negotiation stage of being sold to Verizon. The root cause for these breaches were cookie forging by the hackers to falsify login credentials that led to access to the accounts that did not have passwords setup. Yahoo had the cookies encrypted with bcrypt.

3.2 eBay (May 2014)

In May 2014, eBay reported that 145 Million user records were compromised using phishing, a social engineering technique. The hackers compromised three corporate employee accounts and got unauthorized access into the company network

3.3 Heartland Payment Systems (March 2008)

In this breach caused by SQL injection, 134 million customer credit cards were exposed.

3.4 Target Stores (Dec. 2013)

Debit/credit card information of up to 110 million customers was stolen from Target stores. The hacker got access to Target’s POS (Point of Sale) system through a third party HVAC vendor and managed to load BlackPOS malware on the Target POS system. The hacker collected the customer data as the credit card purchases were being made. The network credentials were stolen from the third party HVAC vendor.

3.5 Heartbleed (CVE-2014-0160)

The transport layer security uses OpenSSLCryptography library to provide secure communication between the client and the server. The library had a buffer overflow vulnerability caused due to the lack of bound checking. As a result, a user was able to obtain the memory contents from the server that included the login credentials of the client. The bug was secretly reported on April 1, 2014 and patched on April 7, 2014. It is important to note that the vulnerability was caused by a programming error that could have been prevented by a code review.

To summarize, it is evident that web application breaches are caused by either software bugs (shortcomings) or social engineering; coercion by hackers to exploit a vulnerability. The rest of the paper discusses how the development group can take a preemptive approach to eliminate potential for vulnerabilities and their exploits.

4 Security Principles and Requirements

Security is bounded by three variables, **C**onfidentiality, **I**ntegrity, and **A**vailability (CIA) and balancing these variable yields a secure and a usable web application (Harris 2013). Overemphasizing one leads to constraining the other two. For example, keeping an article of common interest in a vault will provide extreme confidentiality but zero availability. In the software engineering realm, if the core of a web application or software is highly secure, it will be very difficult to grow that application. A very secure operating system is very constrained and will have very limited expansion opportunities. The following discussion highlights the requirements for each element:

4.1 Confidentiality

Confidentiality is the assurance that information is not disclosed to unauthorized individuals, programs, or processes (Harris 2013). This definition leads to well-designed access controls for any unauthorized activity and thus involves elements, identification, authentication, authorization, and audit. Each of these elements places unique requirements as discussed below:

4.1.1 Identification

Identification (ID) is the process of uniquely recognizing an entity before letting it use the system resources. In most cases, it is a unique string of characters such as a name, an email address, or an account number. In the simplest scenario, security involves verifying the string with the access control list. The methods to assure that the ID string is not programmatically generated are now emerging where the entity has to prove its physical existence.

The identification has the following requirements:

- Each user has a unique ID
- ID should follow a standard convention if needed such an email address
- A user ID is not shared with the other users or unauthorized system processes
- ID value is not reflective of position or role
- CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) if needed

4.1.2 Authentication

Authentication processes assure that a user has successfully proven to be a legitimate entity to utilize the system resources. Authentication is based on three aspects – something specific you know, something specific you have, and a characteristic unique to you. Something specific you know is normally a password the user has setup. The password is sometimes reinforced with a set of questions and answers, which also is a form of something you know.

Something you have can be a string of characters randomly generated by a hardware device such as a pin or a physical object, a card that you swipe to get access. A physical characteristic unique to you is biometric aspect such as your figure print or cornea. A strong authentication should include two factors.

The requirements for authentication will be driven by the type of mechanism the application is using. For a password (string of characters), a typical set of requirements will be:

- Complexity/Crackability–
    - Difficult to guess (minimum length, required character categories, prohibitive elements – last name, date of birth etc.)
    - Should not require extra efforts to remember to avoid noting it down which has the potential for compromise
    - Two factor authentication works adequately (if supported)
- Failure/Recovery Process
    - Number of attempts before lockout
    - Use of security questions for first login attempt from a new device
    - Recovery Mechanism – supported by secure communication such as email and not providing the capability to change it on the client side.

- No email distribution of password

4.1.3 Authorization

Not every user needs access to all system resources, programs, processes, or data. The access to these resources should be controlled by an access criterion based upon the security policy. These criteria may vary from “No Access” as a default to “Need to Know”. A sample set of requirements for authorization will be:

- Policy to control subject access to objects such as database servers
- Processes treated as subjects

4.1.4 Audit

A sound security system must include an audit log of both failed, and successful access attempts. Repeated unsuccessful access attempts are of particular interest to avoid a potential security breach. The successful attempts are of critical importance if a breach has occurred to perform a forensic analysis. The following is a small set of requirements for Audit:

- Maintaining read only audit logs
- Compliance with the legal requirements
- Deploying audit trail analysis tools to review logs

4.2 Integrity

Integrity is upheld when the assurance of accuracy and reliability of information is provided and any unauthorized modification is prevented. (Harris 2013). Simply stated, integrity is maintaining data accuracy at all times. A short list of requirements will include the following:

- Integrity maintained while data is at rest or in transit
- Role based access control
- Any malicious attempts logged with adequate tracking

This minimal set can be enhanced as needed based upon the application use case scenarios.

4.3 Availability

Availability protection ensures reliability and timely access to data and resources, and only to authorized individuals. Listed below is a small set of requirements for availability:

- Available as needed (24x7x365 or as per other criteria)
- Redundancy to reinforce availability

In a well-defined and practiced SDLC, the requirements must be comprehensive, agreed upon by all stakeholders, and signed off by security/product owner. A set of rigorous requirements will lead to a reliable foundation of a secure web application.

5 Establishing Security Controls

Once the security requirements have been identified, the DevOps team can establish the essential or critical security controls that should be built into an application. Establishing security controls is a process similar to establishing an acceptance criterion which is a well-established process. Some early work published by Bayse of SANS (Bayse 2004) provides general guidelines in the form of checklists that could be utilized to establish web application security controls. The table below shows the security control for the “authentication” requirement for a secure login.

Security Controls for Authentication Using a Password (character sting)	
Requirement	**Control**
Password Complexity/Crackability	
Difficult to Guess	• Must be at least 8 characters long • Characters must include at least one upper case letter and one digit • Must not resemble Login ID, Last Name, or First name • Must not be made of any combinations of login ID, First Name, or Last Name.
Efforts to remember	• Must not be randomly generated • Must not be made up of random letters and digits
Failure Recovery Process	
Number of attempts before Lockout	• Each unsuccessful attempt must display a Lockout warning message and number of attempts remaining • After 3 unsuccessful login attempts, the account must be locked out and the user must be informed of the lockout. • The procedure to get the account unlocked must be displayed when the account has been locked out.
Use of Security questions on the first login on a new device	Upon creation of a new account the user must provide answers to three questions of his/her choice for a question bank for future identification on a new devices.
Recovery Mechanism	• User must not be able to reset password on the portal • The password reset should be activated by sending an email with a unique code to the known valid address • User must use the code to reset the password
No email distribution of password	The user password must not be distributed via email

Table 1. Security Controls for the “Authentication” Requirements

Given the security controls the DevOps team can derive the acceptance criteria and the test team can determine the test strategy and develop a test plan. Alternatively, the security control list can be used as the acceptance criteria

6 Test Strategy

A web application depends upon its environment that often consists of a platform (operating system, firewall, antivirus), the language type (compiled or scripted) used to create the application, third party libraries and applications, and a database and its supporting language. The functionality, the standards, and the development practices, add another dimension to the security challenge. An analysis of OWASP Top 10 vulnerabilities reveals the root causes shown in Table 2. It is evident form Table 2 that web application security is significantly influenced by its environment.

Vulnerability	Root Cause
A1-Injection	SQL query manipulation, system call execution, lack of client code validation
A2-Broken Authentication	Unsecure transport layer, password handling and management, session ID management
A3-XSS	Lack of server side script validation
A4-Insecure DO References	Object exposure to unauthorized clients
A5-Security Misconfiguration	Versioning, system hardening, access control, verbose messaging (error, debug)
A6–Sensitive Data Exposure	Data format, encryption key compromise, unsecure data transport
A7-Missing Functional Level Control	Access control to sensitive request handlers
A8-CSRF	Caused due by session cookie stealing
A9-Using components with known Vulnerabilities	Exploitation of vulnerable framework components/libraries from 3rd party
A10-Unvalidated Redirects and Forward	Attacker redirects client to his site with the home page look and steals the login information

Table 2. OWASP Top 10 Vulnerabilities and Root Causes

The preceding discussion suggests that a robust testing strategy should be in place to have a high level of confidence in security. An approach such as penetration testing alone, will not achieve the desired results and it is important to deploy a strategy that includes code inspections, static code analysis, dynamic code analysis in addition to penetration testing.

6.1 Code Inspections

Inspections have been long used to improve software development practices aiming at producing better quality software. Fagan inspections have been applied to improve software requirement specifications, system architecture, code quality to adhere to coding standards, and software test scripts (Wikipedia).

Inspection can be effectively used to improve the security of a web application by focusing on both environment and the application aspects. For example, buffer overflow, a well know security flaw that can arise in programming language like C, can be addressed by code inspections at a very early stage. From the web application functionality perspective, an inspection of security requirements and controls and coding practices can have a significant impact. Just a review for input sanitization can be effective to prevent OWASP vulnerabilities such as A1-SQL Injection, A3-Cross Site Scripting, and A10-Unvalidated Redirects and Forwards. A review to enforce the use of prepared statement and not dynamic SQL queries can also be highly effective in preventing SQL injection.

It should be worth noting that the inspections are human interactions and require subject matter experts to assure good return on investment. The code inspection suffers from the three following shortcomings:

1. The process turns out to be complex if the data flows through multiple states via large number of methods or files
2. While shallow security flaws can be detected with reasonable efforts, it is difficult to assure completeness when the vulnerabilities run across multiple methods

3. A web application normally changes more frequently than a general software application and thus it becomes difficult to maintain the inspection discipline. However, inspection must be used when a new application is being developed.

Considering above factors, the code inspections must be integrated with the automated tools for an early detection of security vulnerabilities.

6.2 Static Code Analysis

Static code analysis is an automated process performed on the source or object code without actually executing the code. The early application of static code analysis to enhance security of life threatening software was studied by Livshits (Livshits 2006). Livshits developed a static code analysis tool based on program query language (PQL) (Martin 2005). A static code analysis tool returns a list of warning messages related to vulnerabilities. Use of a static code analyzer provides the following advantages:

- The potential vulnerabilities can be detected early in the lifecycle
- The analysis is comprehensive and explores all program execution paths where code inspection becomes relatively complex
- It avoids false negatives depending upon the soundness of the analyzer

There are commercially available tools (Veracode and Fortify by HP) for static code analysis to detect the web application vulnerabilities during development. Once these vulnerabilities are identified, the programmers can then take measures to resolve them.

6.3 Dynamic Code Analysis

Static code analysis provides early warnings on the vulnerabilities but, it may still suffer from false positives (Livshits 2006) and should be substantiated with a dynamic code analysis. Tools like Veracode and Fortify provide both static and dynamic analysis capabilities, can be integrated in the SDLC and help the DevOps team build a more secure web application.

Web scanners such as Zed Attack Proxy (ZAP) and Burp Suite can also be helpful in determining security vulnerabilities. ZAP is available from OWASP at no cost and is relatively easy to setup. Burp Suite is a low cost tool and provides good scanning support.

7 Vulnerability Risk Assessment Using DREAD Model

It is vital to keep track of the vulnerabilities identified during the development of a web application, similar to other software defects. It is also important to classify these vulnerabilities according to their impact to assess the overall threat level. Also, it is not always possible to remove all the vulnerabilities and a business decision has to be made regarding which vulnerabilities should be removed prior to release.

The DREAD model (Infosec Institute) offers a way to weigh the severity of a vulnerability. The acronym stands for:

D – Damage, R – Reproducibility, E – Exploitability, A – Affected Users, D – Discoverability

A DREAD index is calculated using the following formula:

DREAD Index = (DAMAGE + REPRODUCIBILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

The index can be used to prioritize vulnerabilities found during the development.

The following quantification table has been developed to assess the impact of each factor:

Category	Value = 0	Value = 5	Value = 10
Damage Impact (data)	None	Few Users	Entire System
Reproducibility	Very Hard	Few Steps Required	Use of Web Browser
Exploitability	Advanced Knowledge	Use of Kits	Just a Web Browser
Actual Users Impacted	None	Some but not all	All Users
Discoverability	Easy – apparent, Public Domain/Web browser	Guessing	Very hard (need special efforts)

Table 3. Quantification of DREAD

8 Conclusion

Just like building a high quality software requires a structured approach, developing secure software needs well-defined approach integrated with the SDLC. As a matter of fact, web application security must be considered one of the non-functional attributes of software quality.

Penetration testing is an approach that validates the web application security at the very end and only from a hacker's perspective. A hacker is not under pressure due to time, but a software release is. Like any other software defects, detection and removal of vulnerabilities late in the SDLC is expensive, and a preemptive approach throughout the SDLC will yield a more secure application with a high level of confidence. The use of automated tools along with the human interactions can prove to be both effective and efficient.

References

1. OWASP (Open Web Application Security Project), Top 10 https://www.owasp.org/index.php/Top_10_2017-Top_10
2. Kim, Peter, The Hacker Playbook 2, Secure Planet LLC., July 2015
3. Armerding, Taylor, The 15 worst security breaches of the 21st Centuary,
4. http://www.csoonline.com/article/2130877/data-protection/data-protection-the-15-worst-data-security-breaches-of-the-21st-century.html
5. Harris, Shon, All in One CISSP, 6th Edition, McGraw Hill, 2013
6. Bayse, Gail, SANS Institute, https://www.sans.org/reading-room/whitepapers/securecode/security-checklist-web-application-design-1389, 2004
7. Wikipedia, https://en.wikipedia.org/wiki/Fagan_inspection
8. Livshits, Benjamin, Improving Software Security with Precise Static and Runtime Analysis, Ph.D Thesis, Stanford University, 2006
9. Michael Martin, Benjamin Livshits, and Monica S. Lam. Finding application errors and security vulnerabilities using PQL: a program query language. Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications, October 2005.
10. Infosec Institute, http://resources.infosecinstitute.com/qualitative-risk-analysis-dread-model/