USF | Muma Case Review

A publication of the Muma College of Business | University of South Florida

Volume 3 Number 10 31 DEC 2018

ONOCHIE FAN-OSUALA

A NOTE ON WEB VULNERABILITIES[1]

In October 2014, JP Morgan Chase, one of the biggest investment banking firms in the united states experienced a breach that caused the leak of the records of an estimated 76 million households and 7 million small businesses. The breach which is one of the largest in terms of number of records exposed was possible through some flaws found in the web applications used by the bank (TrendMicro, 2014). The attack, believed to have focused on servers housing sensitive information of customers who accessed the sites *chase.com* and *jpmorgan.com*, was carried out through a *zero-day vulnerability –*exploiting one of the security holes present in the bank's web application. Similarly in January 2009, Heartland Payment Systems, a payment processing company was a victim of data security breach that involved the exposure of tens of millions of cardholders' records (Vijayan, 2009). This breach was carried through an *SQL injection* attack on the company's website (SecureWorks, 2012). Consistent across these two examples and so many other security breaches is the attacker's point of entry: vulnerabilities in the web applications. Furthermore, a Verizon report (Blevins, 2014) on data breaches notes that web applications attack remains the most common security threat on the internet, with about 35% of all confirmed breaches linked to web application vulnerabilities.

Web application security vulnerabilities are defects, cracks or weaknesses in a web application, often as a result of design flaws or an implementation bugs, which allows an attacker to cause harm to the stakeholders of the web application (application owner, application user, and other entities that may rely on the application) [OWASP Vulnerability, n.d.] According to the open web application security project (OWASP), web application vulnerabilities and weaknesses are a result of: lack of or inadequate validation of inputs provided by users, insufficient logging mechanisms of activities on the web application, fail-open error handling, and not closing database connections properly.

The frequency in which new web applications and features are rolled out and the lack of thorough testing of these web apps compared to commercial software packages and operating systems, often makes it increasingly difficult not to have defects or vulnerabilities introduced into such applications. Though most people and organizations still do not have a security-first approach to their web applications (Kalman, 2014), organizations that do pay attention to web application security are not entirely security risk free. The variety and mixture of techniques available to attackers make protecting web applications a complex task (Geer, 2015). In order to keep abreast with web application security issues and challenges as well as **a**dhere to security best practices, organizations can rely on information and practices suggested by dedicated communities like OWASP (Geer, 2015).

[1] Copyright © 2018, *Onochie Fan-Osuala*. This technical note was developed to provide background information in support of one or more case studies published by the *Muma Case Review*. This note is published under a Creative Commons BY-NC license. It may be freely copied and shared for non-commercial purposes.

Editor: T. Grandon Gill

OWASP

The open web application security project (OWASP) is an international not-for-profit organization devoted to web application security and related issues. It is a community dedicated to enabling organizations conceive, develop, acquire, operate and maintain secure web applications (OWASP About, n.d). The community educates developers, designers, architects and organizations on the issues of web application security especially vulnerabilities through the provision of free articles on issues, guidelines, checklists, methodologies, documentation, tools, and technologies related to web security. OWASP is not affiliated to any technology company, however, membership is open and includes corporations, educational organizations, and individuals from around the world. OWASP projects generally cover many areas of application security and are categorized as follows:

- ***Flagship Projects:*** are projects in topics that have shown strategic value to the community and application security in general. Examples include: OWASP Web Testing Environment Project, OWASP Top Ten Project, OWASP CSRFGuard Project
- ***Lab Projects:*** are project that have produced an OWASP reviewed deliverable of value. Examples include: OWASP Mantra Security Framework, OWASP Security Shepherd, OWASP Top 10 privacy Risks Projects, OWASP proactive controls
- ***Incubator Projects***: are projects in the experimental phase that are still being proven and developed. Examples include: OWASP Broken Web Applications project, OWASP Joomla Vulnerability Scanner Project.

One of the major projects of OWASP as previously highlighted is the Top 10 Project. The Top 10 project started out as a list that identifies and describes the ten most common or prevalent web application vulnerabilities. However since 2010, the list prioritizes these vulnerabilities by risk rather than by prevalence.

Most Common Web Vulnerabilities

Over the years, the most common web vulnerabilities and the risk associated with them have evolved. However, some vulnerabilities have remained consistent and have been continuously exploited by attackers. The impacts of these attacks through these vulnerabilities and the ease in which attacks through these vulnerabilities can be social engineered has made them worthwhile to attackers. Outlined are some of the most consistent and risky web vulnerabilities:

Injection vulnerability

This type of vulnerability is easy to exploit and has remained consistently at the top of many web vulnerability lists. It is, perhaps, the most risky of web application vulnerabilities. Organizations consistently face the risk of breach through injections. Injection vulnerabilities such as SQL, OS and LDAP injections happen when untrusted data is sent to an interpreter as part of a command or query (OWASP, 2013) often causing the interpreter to execute unintended commands. In an injection attack, the attacker's uses malicious data (often text-based) that deceives the targeted interpreter (e.g. a server) into executing unintended commands which can lead to severe consequences (e.g. accessing data without proper authorization, complete host takeover, denial of access). These vulnerabilities are often found in legacy code; queries - Xpath, SQL, LDAP, NoSQL; OS commands; XML parsers; SMTP headers; and program arguments (OWASP, 2013). Injection vulnerabilities are often difficult to discover during testing, but can be easy to discover by examining the code (OWASP, 2013).

Prevention

Some procedures to reduce the presence of this vulnerability are:

1. Consistently check the code in an application to see if the applications uses the interpreters safely. This can be done through code analysis tools.
2. Keep untrusted data separate from commands and queries (OWASP, 2013). This can be done by using safe APIs that either avoid interpreters completely or provide some parameterized interface. However, even with APIs, be careful with parametrized stored procedures.
3. Doing input validation or white listing input is also recommended. In this case, web forms are designed to accept only legitimate inputs.

Broken authentication and session management (BASM)

This is another risky web vulnerability often resulting from poor design and implementation of application functions related to authentication and session management (OWASP, 2013). This vulnerability allows attackers to compromise passwords, keys, session tokens, exploit other flaws in the application, or hijack user identities due to improper protection of credentials and session tokens during connection lifecycle. BASM flaws are often introduced through ancillary authentication functions like logout, timeout, remember me, account management, and password management. An example of this vulnerability is when an application has user session IDs exposed in the URL with unencrypted connections. This can be easily hijacked and used to impersonate a user. BASM vulnerability can be difficult to detect with scanning tools, however, thorough code review and testing are quite effective in detecting this flaw.

Prevention

To prevent this flaw:
Use secure connections, communications and credential storage. This is done by using SSL as the option for authenticated parts of the application and storing credentials in hashed or encrypted forms.

1. Make concerted efforts to avoid cross-site scripting vulnerabilities in your web application design as these can be used to steal or hijack session IDs.
2. Carefully plan authentication design and consider human factors. For instance, ensuring that a logout destroys all server side session state and client side cookies without requiring human confirmations as humans can easily close the window without logging out successfully.

Cross-site scripting (XSS)

XSS is by far the most common web application vulnerability. It occurs anytime an application takes unverified data and sends it to a web browser without proper validation or escaping (OWASP, 2013). XSS allows attackers to execute scripts in a victim’s web browser which can lead to hijacking of user sessions, defacing of web sites, or redirection of victim to malicious sites. Typically, the attacker exploits a vulnerability in a web application to deliver malicious scripts to a victim’s web browser. For instance, if a vulnerable website or application requires user input in its pages, an attacker can insert malicious code as part of the HTML elements for rendering the webpage and while parsing the HTML elements for the page, the victim’s browser executes the malicious code. XSS attack is often employed within VBScript, ActiveX, Shockwave and Flash, but JavaScript still remains the most abused due to JavaScript’s near ubiquity in browsing experience (Acunetix, n.d). Figure 1 shows a high level view of a typical XSS attack. The variants of XSS attacks are numerous and the likelihood that a site has XSS vulnerabilities is very high. There are three main types of XSS: reflected XSS, stored or persistent XSS, and DOM injection.

- *Reflected XSS* attacks are the most simple to exploit. It is an XSS attack where the injected script is usually bounced off the web server or returned as part of the request, such as in an error message or search result where the response from the server contains part of or all the input sent to the server as part of a request. The attack is delivered to the victim through another trusted route like email or trusted web server and is deceived into triggering the malicious script by either clicking a link, submitting a specially designed web form, etc.
- *Stored XSS* takes malicious data, stores it (in a file, database, or back end system), and at some time in the future, presents the data unfiltered to the user. This type of XSS is extremely bad for content management systems (CMS), blogs, and forums where a multitude of users see inputs from other individuals.
- *DOM injection* typically alters the site JavaScript codes and variables instead of the HTML elements. Typically, DOM is focused on the client side execution of scripts since the JavaScript codes which have been altered executes differently from how the web application owners intended it to execute.

XSS Vulnerabilities can be detected through thorough testing and code analysis.

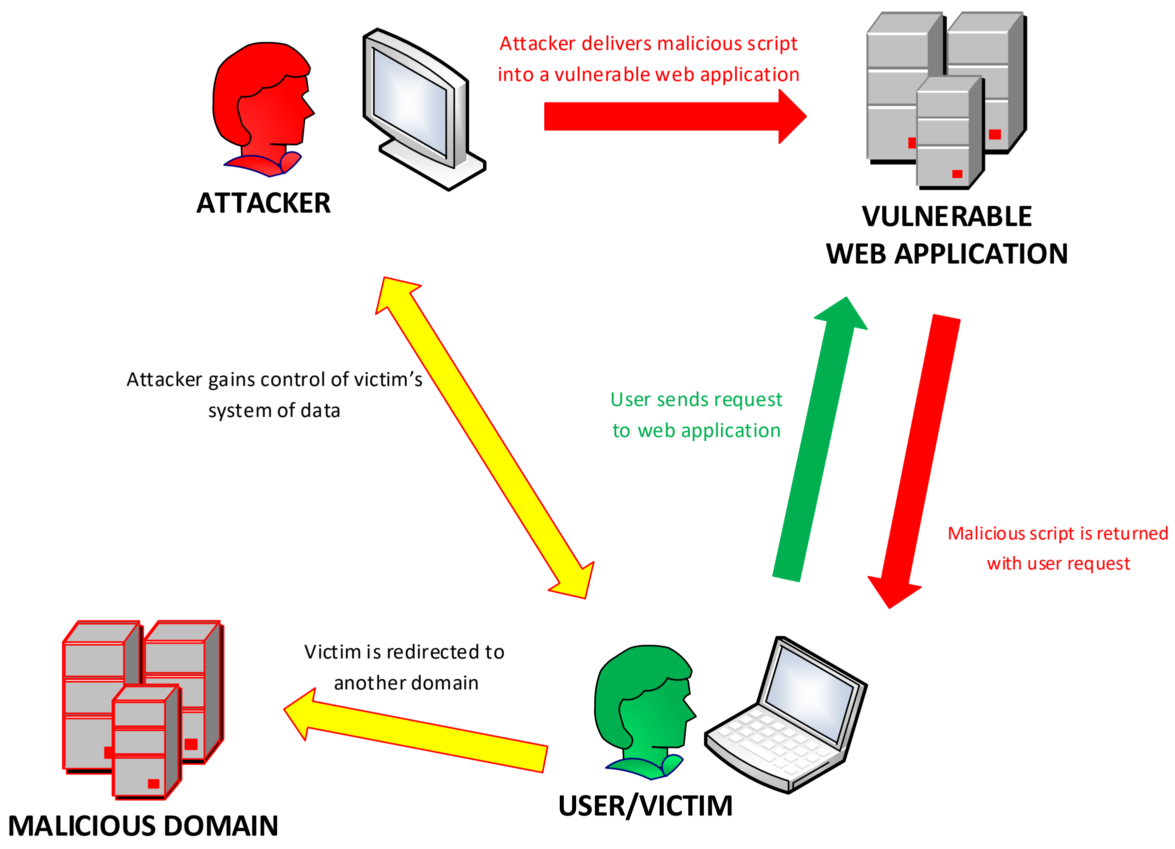
Attacker delivers malicious script into a vulnerable web application
ATTACKER
VULNERABLE WEB APPLICATION
Attacker gains control of victim's system of data
User sends request to web application
Malicious script is returned with user request
Victim is redirected to another domain
MALICIOUS DOMAIN
USER/VICTIM

Figure 1. A high level view of a typical cross-site scripting attack.

Prevention

1. Properly escape all untrusted data depending on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into (OWASP, 2013).
2. Doing input validation or white listing input is recommended. In this case, validation should include verifying length, formats, characters, and rules before using them as input.
3. Consider *Content Security Policy (CSP)* to guard against XSS across attacks (OWASP CSP, n.d).

Insecure direct object reference

This vulnerability occurs when a developer leaves the reference to an internal implementation object (e.g. file, directory, database key) exposed. The exposure allows attackers to easily alter these references in order to access unauthorized data. For instance, an application that uses the actual name or key of an object (e.g. database key) when generating a web page and does not verify if the user is authorized can easily be manipulated into referencing another system object (often restricted resources) by altering a parameter value that is used in referencing the predefined system object. The finest way to detect if this vulnerability is present in an application is to check that all object references have appropriate guards. Insecure direct object references flaws are easy to detect thorough code analysis which can show if authorization is properly verified. Testing can also help in detecting these flaws.

Prevention

1. Using per user or per session indirect object references as this can stop attackers from directly targeting unauthorized resources.
2. Checking access for each use of a direct object reference from an unknown source to guarantee that the source is authorized for the requested object.

Other Popular Web Vulnerabilities

While the prior listed four vulnerabilities have consistently remained at the top of the list for being the most risky vulnerabilities in the past 6 years, other prevalent vulnerabilities are:

Cross Site Request Forgery (CSRF)

CSRF is an attack that makes an authenticated user's browser perform unwanted actions on a vulnerable application to the benefit of the attacker (OWASP CSRF, n.d). A CSRF attack through some form of social engineering (links in email or chat) can deceive a victim's web application into submitting malicious requests when the user is authenticated into a site and executing sinister actions of the attackers choosing like funds transfer, changing of user passwords, or even compromising the server end web application. CSRF can be prevented by including unpredictable unique token in each HTTP request.

Use of Components with known vulnerabilities

Using components like libraries, frameworks, and modules with known flaws in building a web application can expose the application to attacks. This is because most of these components often run with full privileges and attackers can sabotage a web application by exploiting these components. Prevention include having a very well defined application design process and practices that pay attention to components used in building the application in terms dependencies, issues and possible weaknesses.

Sensitive data exposure

A significant number of web applications do not protect sensitive data like credit card details, tax IDs, and passwords. Such weakly protected data can be easily stolen or modified by attackers and used to their benefit. Prevention is done by identifying what data is sensitive and providing extra protection through encryption when the data is in transit or stored.

Similarly, the following vulnerabilities are worthy of mention from the large pool of web application security vulnerabilities: security misconfiguration, missing function level access control, unvalidated redirects and forwards, and information leakage and improper error Handling.

References

- Acunetix (n.d). Cross-site Scripting (XSS) Attack Retrieved from https://www.acunetix.com/websitesecurity/cross-site-scripting/
- Blevins, B. (2014). Verizon data breach report: Web application attacks a growing concern. Retrieved from http://searchsecurity.techtarget.com/news/2240219379/Verizon-data-breach-report-Web-application-attacks-a-growing-concern
- Geer, D. (2015). “Why are there still so many website vulnerabilities?” retrieved on October 3, 2015 from http://www.csoonline.com/article/2936619/data-protection/why-are-there-still-so-many-website-vulnerabilities.html
- Kalman, G. (2014). 10 Most Common Web Security Vulnerabilities. Retrieved from http://www.toptal.com/security/10-most-common-web-security-vulnerabilities
- OWASP About (n.d). About The Open Web Application Security Project. Retrieved from https://www.owasp.org/index.php/About_OWASP
- OWASP CSP (n.d) Content Security Policy. Retrieved on October 3rd, 2015 from https://www.owasp.org/index.php/Content_Security_Policy
- OWASP CSRF (n.d). Cross-Site Request Forgery. Retrieved on October 4th, 2015 from https://www.owasp.org/index.php/CSRF
- OWASP Vulnerability (n.d). Vulnerability Retrieved on October 4th, 2015 from https://www.owasp.org/index.php/Category:Vulnerability
- OWASP. (2013). “OWASP Top 10 – 2013: The 10 most critical web application security risk” retrieved on October 3, 2015 from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013
- SecureWorks (2012). A Famous Data Security Breach & PCI Case Study: Four Years Later. Retrieved from https://www.secureworks.com/blog/general-pci-compliance-data-security-case-study-heartland
- TrendMicro (2014). JP Morgan Breach Affects Millions, Shows the Need for Secure Web Apps. Retrieved from http://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/jp-morgan-breach-affects-millions-shows-need-for-secure-web-apps
- Vijayan, J. (2009). SQL Injection attacks led to Heartland, Hannaford Breaches. Retrieved from http://www.computerworld.com/article/2527185/security0/sql-injection-attacks-led-to-heartland--hannaford-breaches.html

Acknowledgements

Development of this note is based upon work supported by the *National Science Foundation* under Grant No. 1418711.

Biography

Onochie Fan-Osuala is a PhD Candidate in information systems (IS) at the Muma College of Business, University of South Florida. He is interested in using analytics and experimental designs to solve problems bothering on the IS-operations, IS-marketing and IS-entrepreneurship interfaces. His work mostly explore these problems in online platforms and marketplaces.