Web application security

DOI: 10.46932/sfjdv3n4-002

Received in: April 14th, 2022
Accepted in: June 30th, 2022

Buket Erşahin
Doctor of Computer Engineering
Institution: Izmir Institute of Technology
Address: Izmir, Turkey
E-mail: buketoksuzoglu@iyte.edu.tr

Mustafa Erşahin
Doctor of Computer Engineering
Institution: Commencis Technology
Address: Izmir, Turkey
E-mail: mustafa.ersahin@commencis.com.tr

ABSTRACT

This study aims to show how security flaws of web applications can threat information security. Web Application Security is a branch of Information Security which focuses on web application level security flaws and their solutions. Evolution of Web continues with a big momentum. Amount of information shared over Web increases every day, various business domains continue to integrate their operations to digital world. This brings its own risks and makes Information Security of Web Applications more important than ever. Most common and serious Web vulnerabilities have been analyzed along with their solutions. This study focuses on how web developers can already prevent security problems during the development life cycle. What are the best practices to follow before/during the development and post-development phases? Which security tools can be used to support developers? Building totally secure web applications is not an easy job. Following security standards and development cycles with security concerns can already prevent most of the potential problems. A security checklist for web developers came out at the end of the study. Evolving web technologies and new security threats force us to keep this checklist up to date. We are working on a mechanism which will keep this checklist up to date.

Keywords: web application, security, development, checklist, security knowledge framework.

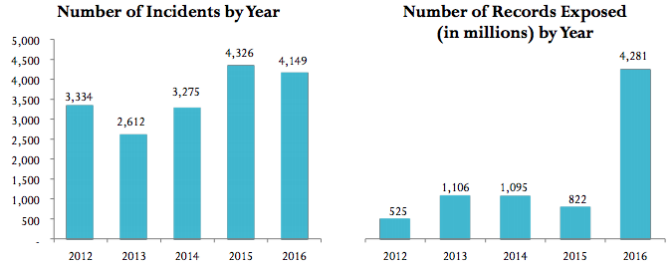
1 INTRODUCTION

It has been long time that web applications became much more than web sites with static content. Evolution of the World Wide Web created web applications with greater abilities and still continues to provide brand new solutions to all kind of market needs.

Web applications are the most used technology for information and service delivery over Internet. Organizations from various different domains continue to move their operations to the digital world. Mobile and Web based technologies come at the top of the list. These digital services can be security critical and may contain sensitive information (e.g., financial, health). Fast growing and evolving web environment brings up its own risks. Guaranteeing security for vast amount of information is not an easy

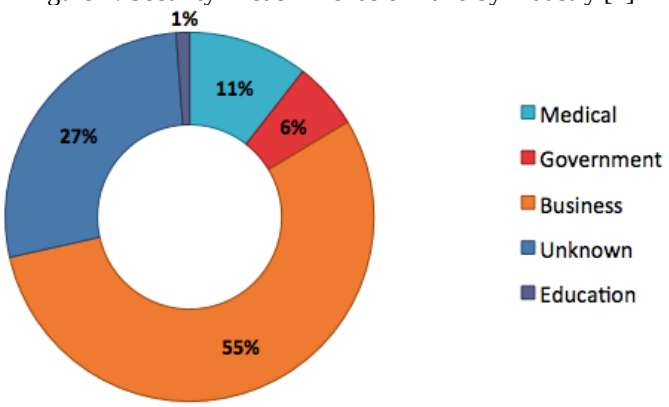
task and not totally achievable. On the other hand, security breaches may result in loss of enormous amount of information and money along with ethical and legal consequences [1].

Figure 1: Security Breach Trends of 2012 – 2016 [2]



According to Data Breach Trends 2016 report of RiskBased Security, 4,281 millions of records have been exposed in 2016, while it was only 822 millions in 2015 [2]. Even though the number of reported incidents has decreased in 2016, consequences of the incidents became much more worse. We see that there is not any correlation between the number of incidents and their results. This fact reminds us that one simple security breach may have serious consequences. Security standards must be followed completely without ignoring any potential threat to minimize all kind of risks.

Figure 2: Security Breach Trends of 2016 by Industry [2]



Business industry was the first target of attackers, followed by medical and government establishments [2].

Sources of these incidents may vary but this study will focus only on web application threats and their precautions. Study will try to look from developers' point of view and examine their role on developing secure applications.

2 STATE OF WEB APPLICATIONS

Web application attacks represent the greatest threat to an organization's security. Web app attacks represented 40% of breaches in 2015 [3]. According to analyzes of WhiteHat Security, 5 to 32

vulnerabilities have been found on thousands of web applications across different industries. Same study shows us that only 5% of web applications have exceptional application security, 40% of applications have a lot to improve about security. Detecting a vulnerability is not solving the problem automatically. Taking action to fix these vulnerabilities takes time also. Average open time of critical vulnerabilities is over 300 days. High-risk vulnerabilities have more than 500 days [3]. Average open time of a vulnerability is an important parameter. High age values increase the chance of attackers to profit these vulnerabilities. Unfortunately, considerable number of web applications stay always vulnerable, not only in specific time periods. Around 40% of Banking & Financial Services web applications, half of the Health and Retail web applications and more than half of the Manufacturing, Food & Beverage, and IT web applications remain always vulnerable [3]. These values show us that organizations cannot fix detected vulnerabilities directly.

Once your application goes live, fixing these vulnerabilities may cost much more than taking precautions during the development phase. Remediating vulnerable servers may create down times which will cause money loss, decrease the customer satisfaction and reputation of the product. Transforming a web application which has been developed without security concerns to a secure web application can be more painful than initial implementation. For example, changing the old and weak password policy of a web application will force all existing users to update their password. Managing this process will cost to the organization. This could be avoided by following security standards at the beginning and launching the web application with correct password policy.

In today's world, development with security concerns is a must. Software companies must cultivate culture of security prioritized development. Security must be a priority as much as the functionality of an application.

Software developers should stay competent on security subjects. List of security threats is not static. It is evolving along with technologies. More complex technologies create more complex security problems. Developers cannot cope with this situation alone; they must be supported with standards and other helper tools.

OWASP is the main source of information about security standards, practices and supporting tools. Thanks to its big and open community, OWASP remains as an up to date, reliable and educational address for individuals to organizations. OWASP has many educational documents and projects for software developers to cope with security problems.

3 SECURITY THREATS AND PREVENTIONS

The OWASP Top Ten project aims to decide what are the most critical web application security flaws. These top ten list is created by broad consensus of project community. Original goal of the Top 10

was raising awareness amongst developers but it has been a *de facto* application security standard with time. Last official release of Top 10 was in 2013 but 2017 version is on the way with its second candidate.

The OWASP Top 10 for 2017 is constructed based on more than 40 security firm reports and industry surveys. Data is collected from more than 100,000 applications and APIs.

Web technologies and architectures have been significantly changed over the last few years. Single page applications have replaced many server rendered web sites. We can now call JavaScript as assembly of web. Rise of the SPA frameworks like Angular and React moved servers' responsibility to the client side (browsers). Event based server Node.js became very popular. Mobile apps and single page applications started to share same APIs.

Microservices written in node.js and Spring Boot are replacing older enterprise service bus applications using EJBs and so on. Old code that never expected to be communicated with directly from the Internet is now sitting behind an API or RESTful web service. The assumptions that underlie this code, such as trusted callers, are simply not valid [4].

This evolution has reshaped the OWASP Top 10 list as below.

Figure 3: OWASP Top Ten 2013 - 2017 Industry [4]

As we can see above XML External Entity (XXE), Insecure Deserialization and Insufficient Logging & Monitoring have entered to the list. Insecure Direct Object References and Missing Function Level Access Control merged as Broken Access Control and took 5th place. Cross-Site Request (CSRF) and Unvalidated Redirects and Forwards take no more place in Top 10 list but they should not be forgotten.

1) Injection

SQL, OS and LDAP injections happen when untrusted data and instructions are sent as a part of command or query. Injected commands may have very serious results like dropping whole database or accessing data without authorization.

Preventions;

- Building secure APIs which avoid query or command concatenation. Using ORMs or

Entity Framework with parameterized queries is the best defense mechanism.

- Applying white list input validation.
- Escaping special characters in dynamic queries.
- Using LIMIT and other SQL controls in queries will keep the damage low in case of injection.

2) Broken Authentication and Session Management

Limiting application functions by user identification is generally vulnerable. Vulnerable session management allow attackers to access passwords, keys or sessions tokens and gain all access rights of real users.

Preventions;

- Never go to live with default credentials. Specially for admin users.
- Store hashed password with modern algorithms like Argon2 or PBKDF2.
- Set strong password policy. Detect weak passwords including commonly used words. Accept pass phrases.
- Define strong registration and credential recovery procedures against account enumeration attacks.
- Implement multi-factor authentication.
- Keep and watch authentication logs to detect brute force and other attacks and alert administrator.

3) Sensitive Data Exposure

Web APIs are not threating sensitive data with extra protection. Sensitive data must be protected with encryption and special precautions must be taken during the exchange with the browser.

Preventions;

- Classify all processed, stored or transmitted data. Apply different level of security control per class.
- Apply privacy laws and regulations to sensitive data.
- Never store sensitive data if not necessary. Safe delete the data as soon as possible.
- Encrypt all sensitive data. Use TLS. Enforce it with HTTP Strict Transport Policy HSTS.
- Use up to date and strong encryption algorithms, keys, protocols and ciphers.
- Disable cache for response which contain sensitive data.
- Store hashed password with modern algorithms like Argon2 or PBKDF2. Keep delay factor as high as possible.

4) XML External Entity (XXE)

This new vulnerability entered to Top 10 list from a high position. Reason of this serious vulnerability is old and poorly configured XML processors which evaluates external entity references within XML documents. External entities can access to internal files, execute remote code and make denial of service attacks like Billion Laughs attacks.

Figure 4: XXE attack for extracting data from server [4]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
 <foo>&xxe;</foo>
```

Preventions;

- Developer awareness is important to detect potential XXE attacks.
- Disable external entity processing in all XML parsers of the application.
- Validate all incoming XML and XSL file uploads by XSD validation.
- Use white listing input validation to check XML file content.
- Upgrade SOAP to latest version.
- Upgrade all third party and underlying XML processor libraries including OS libraries. Using dependency checker supported by CVE feeds will decrease the risk.

5) Broken Access Control

Constructing a robust authentication and session management sometimes may not be enough. Attackers can't steal session info or credentials but they can still use more privileged user rights if user roles and their restrictions have not been defined properly. Attackers may access and modify other users' data without breaking any authentication rules.

Preventions;

- Access control must be implemented in trusted server-side where attackers can't modify access conditions.
- Implement user role and access right control mechanisms once and re-use them in all application.
- Give users only required rights, don't give unnecessary read, create, update or delete rights. Give these rights by record.
- Disable web server directory listing. Don't keep metadata files in web roots.
- Log and watch access failures and alert admins.

6) Security Misconfiguration

Bad security configurations or using default configurations without any changes is very common. Wrong configurations can expose sensitive data.

Preventions;

- Development, QA and Prod environments must be configured identically including all dependency versions.
- Deployment processes must be standardized and should be automated.
- Never install unnecessary dependencies, documentations.
- Follow CVE updates.

7) Cross-Site Scripting (XSS)

Two thirds of applications contain this vulnerability. Attacker can execute scripts in victim's browser which can steal sessions, deface web sites or redirect to malicious sites.

Preventions;

- Using some up to date frameworks like React prevents XSS attacks automatically.
- Escaping untrusted HTTP request data and applying context sensitive encoding. They are explained in detail in OWASP cheat sheet.
- Enabling Content Security Policy header.

8) Insecure Deserialization

Applications may deserialize insecure objects without any control. This leads remote code execution, user spoofing and privilege elevations.

Preventions;

- Don't accept serialized objects from untrusted sources.
- Deserialize only primitive data types if possible.
- Run deserialization in isolated, very low privileged environments.
- Enforce strict type constraints during deserialization. Accept only predefined object patterns.
- Log and observe deserialization operations.

9) Using Components with Known Vulnerabilities

Nearly all complex applications have series of dependencies which have same privileges as application itself. An application can be implemented according to high level security standards but one vulnerable dependency can make whole application vulnerable as well. Because of that dependency management and vulnerability tracking is vital.

Preventions;

- Remove unused dependencies.

- Use vulnerability detection tools like versions, DependencyCheck and retire.js.
- Follow feeds of CVE and NVD vulnerabilities of dependencies with an automated process.
- Accept dependencies only from official sources, repositories. Prefer signed packages.

10) Insufficient Logging & Monitoring

Lack of logging in your application causes delay in breach detection. Detailed logging of your application supported with log analyze tools can detect incidents much more earlier.

Preventions;

- All login, access control and input validation failures must be logged.
- Enable log monitoring which can detect suspicious log patterns and raise alerts.
- Don't forget to obfuscate sensitive data in log files.

4 SECURE SOFTWARE DEVELOPMENT

This fresh Top 10 list will update the priorities of developers and whole industry. Even though the top ten covers 80-90% of all common attacks and threats we can't stop with it. OWASP Developer's Guide, Testing Guide, Code Review Guide and Cheat Sheet documents are essential for complete protection. Top 10 list and these supporting documents keep evolving. Even without changing single line of code, applications may become vulnerable with time [4].

Developers must digest various documents and standards in order to create and maintain an application with OWASP Application Security Verification Standard (ASVS). Developers alone can't achieve this. Developers should use supporting tools and focus on standards rather than focusing on specific vulnerabilities.

As discussed in previous chapters, having an application secure by design in much more better and easier than chasing vulnerabilities and patching your application after incidents occur. Developers must be educated and guided about up to date security standards. Developers must apply security principles to software development cycles in order to have secure by design applications. All these responsibilities can create too much work for developers and specialization on security can take too much time.

Security Knowledge Framework project is a perfect match for developers and security specialist requirements. This project aims to educate and guide developers for secure development. It contains knowledge base, checklists and code examples to achieve security standards on desired levels.

We have installed a SKF instance for test purposes. We saw that it has been designed work in parallel with agile software development methods. Following steps are followed on SKF during software development cycle.

- Firstly, a project must be created in SKF which will represent a security clone of our real software project.

- ASVS level (opportunistic, standard, advanced) must be decided according the requirements of the project.
- Name, description and version of security project must be cloned from original project.
- Pre-development settings questions must be answered correctly. According to answers SKF will shape the security checklist at the end.
- We add our first sprint and answer questions about the sprint content which will shape the security checklist.
- Security project is now created and a security checklist has come out for first sprint.
- Developers can review the checklist and update the status of every item in the list like updating a functional task status on a Scrum board.
- After developer closes the security item Security Specialist user role can verify the item and task is closed before the next sprint.
- Every time a new sprint starts, a security sprint must be started in parallel. SKF produces a new security checklist for every Sprint according to the Sprint content.

Security checklists produced by SKF come with explanations and necessary directions to cheat sheets and knowledge bases.

5 CONCLUSION

Applying secure code practices and creating secure by design web applications is not easy to achieve. On the other hand, there are plenty of guides and standards for testers, developers, managers and organizations, provided by OWASP community. These guides show us the path towards secure web applications. However, digesting all these standards in daily development life does not seem realistic. Using Security Knowledge Framework will discipline the developers and increase the security awareness of organizations.

Tracking vulnerabilities of application dependencies is as important as secure development of application itself. Your fully secure application can become vulnerable because of its dependencies. CVE tracking tools must be absolutely used.

We believe in that; developers will create more secure web applications with support of SKF. SKF will provide up to date security checklists, decrease the effort of developer to follow evolving security standards.

REFERENCES

[1] X. Li, Y Xue, *A Survey on web application security*. Vanderbilt Univ., Nashville, TN, 2011.

[2] *RiskBased Security Data Breach Quick Report*, Risk Based Security, Inc., Ricmond, AZ, 2017. Available: https://pages.riskbasedsecurity.com/hubfs/Reports/2016_Year_End_Data_Breach_QuickView_Report.pdf

[3] *Web Applications Security Statistics Report*, WhiteHat Security, Inc., 3970 Freedom Circle Santa Clara, CA, 2016. Available: https://info.whitehatsec.com/rs/675-YBI-674/images/WH-2016-Stats-Report-FINAL.pdf

[4] *OWASP Top Ten Project 2017 Report Release Candidate 2.* Available: https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf