Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2019, 6(4):77-81

Research Article ISSN: 2394 - 658X

The Business Impact of OWASP Top 10 Vulnerabilities Risk Assessment and Mitigation Strategies

Rajashekar Reddy Yasani[1], Karthik Venkatesh Ratnam[2]

[1]Senior Security Engineer, Independent Security Researcher, Boston, MA
Cloud Security, Cloud Computing, Cyber Security
rajshekaryasani@gmail.com
[2]Cloud Engineer, Devsecops (cloud security)
Independent Security Researcher, Dallas, TX
karthikratnam1@gmail.com

ABSTRACT

The industry frequently makes use of open source software (OSS) libraries to hasten the creation of software goods. But the number of publicly known vulnerabilities affecting these libraries is growing. Application developers must promptly identify insecure library dependencies, accurately evaluate their impact, and take measures to limit any risk. In order to identify, evaluate, and counteract OSS vulnerabilities, this paper introduces a new approach. Web application security is of utmost importance in this digital age due to the pervasiveness of online services and interactions. The Open online Application Security Project (OWASP) is a worldwide group that is helping to direct initiatives related to online application security. To that end, it publishes what is known as the OWASP Top 10 on a regular basis; this list identifies the most pressing security threats to online applications. The purpose of this research report is to reveal how web application vulnerabilities have changed over the years by comparing the OWASP Top 10 lists from 2017 and 2019. It sheds light on the ever-shifting horizon by delving deeply into description of vulnerabilities, ranking changes of threats, and rising trend identification. Developers, security practitioners, and organisations can all benefit from the comparative study's findings by incorporating them into their own security policies and procedures. It is a guide for dealing with the modern, ever-changing threats to web application security.

Keywords: Open source software, Publicly known vulnerabilities, Code-centric vulnerability analysis, Combination of static and dynamic analysis, Usage-based update support.

INTRODUCTION

More and more people are making use of open source libraries, and the number of libraries available is also growing: According to Synopsys [1], more than 96% of the applications they examined make use of open source libraries, and their code often accounts for more than half of the average code-base. A study found that the number of Java OSS libraries available in Maven Central increased by 102%, while the number of Python libraries in the Python package index (Pypi) added another 40%.

While open-source libraries (OSS) might shorten development times, they also come with risks, such as the possibility that reliant applications could be affected by vulnerabilities found in OSS. A long way has been gone since the issue was first identified as one of the top 10 application security risks in 2013 (OWASP Foundation 2013, 2017) due to the usage of open source software components that have known vulnerabilities. There has been a steady stream of OSS vulnerabilities that have made it into the news, from 2014's Heartbleed and Shellshock bugs to 2017's (in)famous Equifax incident, which exposed the personal information of more than 140 million Americans due to a neglected update of a widely used OSS component. Further, as reported in [3], the number of vulnerabilities found in open source library software continues to rise, rising by 43% in 2017 and 33% in 2018. The fact that most data breaches in 2016 have OSS vulnerabilities as their basis is thus not surprising (as documented by [4]). It is widely recognised as a priority in the software industry to establish effective practices for managing open

source software vulnerabilities, backed by suitable tools. Nowadays, there are tools that can detect known vulnerable libraries, such as WhiteSource and Synopsys Black Duck, as well as OSS, like OWASP Dependency Check (OWASP DC) and Retire.js. In terms of detection capabilities, these tools vary. However, our understanding is that their approaches assume that the metadata associated with OSS libraries (such as name and version) and vulnerability descriptions (such as technical details and list of affected components) are always up-to-date and consistent. This information, which are utilised to associate each library with a list of recognised vulnerabilities, are frequently lacking, inconsistent, or insufficient [5,6]. This means that tools depending on them could either miss vulnerabilities altogether (false negatives) or mistakenly identify susceptible artefacts that don't really contain the code that causes the vulnerability (false positives). To get around these problems, most companies that sell security products and services have their own proprietary vulnerability databases where they "map" vulnerabilities onto the affected artefacts.

However, this process is labour-intensive, duplicated across vendors, and prone to mistakes. Finally, the requirements of the full software development life cycle cannot be met by just checking for vulnerable library inclusions. Since the required modifications to the application code can be executed as part of the regular development processes in the early stages of development, upgrading a library to a more current release is not too hard. Updates, on the other hand, need serious consideration as projects approach customer release dates and throughout their operational lifetimes. This is because updates can affect release schedules, increase effort requirements, cause system downtime, or introduce new defects [7]. To determine if a library update is essential and how urgent it should be, one must first determine if the vulnerability may be exploited given the specific manner the library is utilised in the application. Finding a reliable way to assess a vulnerability's exploitability and potential impact requires technical details and information at a lower level than what is typically provided in advisories, which are brief, high-level descriptions written in natural language. Therefore, answering this question is very challenging. As a first step in addressing this issue, we detailed our methodology for dynamically analysing code modifications brought about by security updates and determining the effect of the vulnerability on a specific application in [8]. A more thorough code-centric and usage-based strategy to identify, evaluate, and address OSS vulnerabilities is proposed in our follow-up article [9], which expands on [10]. Its main argument is that methods that focus on code and usage are better than ones that rely on meta-data. This is especially true when it comes to metrics that take application-specific library use into account, the ability to conduct impact assessments tailored to individual applications with the help of reachability analyses, and mitigation support.

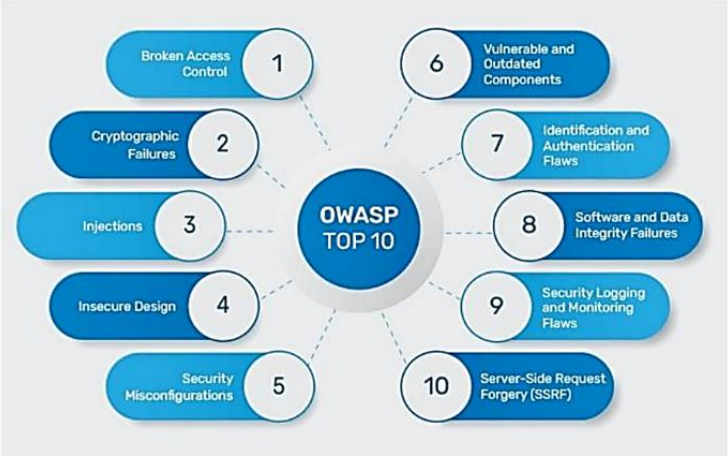

Fig 1. *Top 10 vulnerabilities identified in OWASP*

Examining and comparing the 2017 and 2019 versions of the OWASP Top 10 will help shed light on the changing trend of web application vulnerabilities. Its goal is to investigate how security measures, threats, and priorities have changed throughout this time in order to better understand the evolution of web application vulnerabilities. Now more than ever, web applications are part of our digital lives, not just tools, so it's crucial to understand them and find ways to protect ourselves from their vulnerabilities. Our goal in embarking on this research and analytical journey is to equip security practitioners, developers, and organisations with the information they need to make effective strategies and practice adaptations [11]. A more secure digital future is within our reach, thanks to the comparison study that leads us through the constantly shifting terrain of web application vulnerabilities.

LITERATURE REVIEW

Various ways to requirement elicitation from a security perspective were reviewed in [12]. Because they are so focused on getting the program up and running, developers often overlook security concerns. Security goals and

threat identification are not priorities throughout the requirement engineering phase. While the Microsoft Security Development Lifecycle [13] is useful for creating safe applications by including security into the software development lifecycle at every level, it doesn't specify when or how to conduct penetration tests. SDLC is not tailored to Android and is instead generic. While the web penetration framework suggested in [14] may be integrated into the software development life cycle (SDLC), it would not aid with the security of Android applications while they are being designed and developed. Demonstrated several penetration testing approaches, tools, and tactics on cellphones in [15]. Application security hardening was found to be the cause of 14.6% of difficulties in the attack outcomes. The fact that some apps still contain malware or malicious code exists, even though Google Play does not let such apps, was also brought up. To better model a system's artefacts, Unified Modelling Language (UML) is useful [16]. Because it lacks components tailored to modelling system interaction, UML has several drawbacks, as discussed in [17].

But, in order to describe the system appropriately, the UML profile can be improved for specialised modelling. Although a novel approach to secure modelling was introduced in [18], the paper provides no guidance on how to create such a model for an Android app. IN [18] added to the UML by defining new elements that allow for the definition of entity relationships. No new profiles for Windows 7 phones or Android OS have been released by Bup-ki that address the security concerns of handling permissions and modelling secure inter-process communication (IPC). Additionally, discussed the use of an event-based approach to automate test cases in mobile testing. When it comes to developing test case artefacts and finding vulnerabilities, this tool is useless. Despite the growing relevance of penetration testing in recent years, there is a dearth of comprehensive literature evaluations addressing the topic of penetration in Android-based applications. The methods and tools for scanning web applications for vulnerabilities were examined in [19]. The evaluation criteria were based on four distinct web penetration approaches and thirteen distinct vulnerability testing scanners. Moreover, four test environments—Webgoat, DVWA, BodgeIt, and WackoPicko—were examined by the author. The assessment led the author to the conclusion that current technologies fail to detect a large number of vulnerabilities. The six frameworks and approaches for penetration testing were examined in [20].

The author discovered that OWASP's OTG and ISSAF, two of the techniques used as evaluation criteria based on the ISO/IEC 25010:2013 quality model, have limitations and a lack of domain coverage. We also mapped out a set of 1,145 papers in a systematic way. Methodologies, tools, and models for penetration testing were evaluated in depth using quantitative and qualitative evaluation criteria. After discussing the potential uses of these methods for vulnerability assessment and the limitations of the different models used for penetration testing online applications, the author draws a conclusion.

According to the discussion in [21], penetration testing involves finding possible ways into a system by employing common hacking tools and techniques. We looked at several hacking tools, compared different tactics and techniques, and then put them to use on banking apps to find security flaws. Results on the effects of coverage requirements on feasibility and ROI were derived from an analysis of 119 papers and an examination of different model-based testing methodologies in [22].

We also looked at ways to exploit software development faults and other software security measures. The author comes to the conclusion that in order for software applications to be effective, developers should implement security measures such as security validation and fix any security vulnerabilities. A method for detecting malware on mobile devices using Q-learning was suggested in [23]. Applying a reinforcement learning technique and a dynamic architecture enhances the detection performance. But the article doesn't go into detail on how to use Q-learning to enhance vulnerability repository detection while building applications.

METHODOLOGY

To guarantee thoroughness, impartiality, and rigour, the technique utilised a comparison of the OWASP Top 10 2017 and OWASP Top 10 2019. The research technique included the following essential stages:

Data Collection

Collecting primary data for the study was the initial step:

OWASP Top 10 2017 Report:

To better comprehend the online application security vulnerabilities that were common in 2017, the OWASP Top 10 was a crucial resource. The official OWASP website was the source of this publication.

OWASP Top 10 2019 Report:

The most current overview of web application security vulnerabilities, the 2019 version of the OWASP Top 10, was likewise acquired from the official OWASP website.

Data Analysis

Comparative Analysis of Rankings:

There was a thorough comparison of the web application security risk rankings in the 2017 and 2019 OWASP Top 10. To find out if there were any shifts in priority, we looked at how each vulnerability ranked. The identification of vulnerabilities that rose to prominence and those that fell to obscurity over time was the primary objective.

Evolution of Vulnerability Descriptions:

• Vulnerability descriptions from both versions were carefully reviewed, taking note of any revisions, updates, or variations. The 2019 edition incorporates new terminology, concepts, and explanations to reflect emerging attack strategies and security considerations. Special care was made to identifying these changes.

Emerging Trends:

Trends and vulnerabilities highlighted or added in the OWASP Top 10 2019 were also intended to be discovered through the comparison analysis. The importance and relevance of these changes in the modern web application security landscape was investigated through a qualitative evaluation.

Ethical Considerations

Ethical concerns were of the utmost importance all through the study. All of the materials used to compile this research review came straight from the official OWASP website and are open to the public. For the sake of this study, we have not used any private or sensitive data.

RESULTS AND STUDY

Table 1. Overview of Analytical Comparisons

Rank in OWASP 2019	Vulnerability Name	Rank in OWASP 2017	Change	Remarks
A01	Broken Access Control	5th	Up 4	Moves up from the fifth position to the category with the most serious web application security risk.
A02	Cryptographic Failures	3rd	Up 2 (& Renamed)	Previously known as A03:2017-Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed name focuses on failures related to cryptography.
A03	Injection	1st	Down 2 + A07	Cross-site Scripting is now part of this category in this edition.
A04	Insecure Design	-	New	Insecure Design is a new category for 2019, with a focus on risks related to design flaws.
A05	Security Misconfiguration	6th	Up 1 + A04	The former category for A4:2017-XML External Entities (XXE) is now part of this risk category.
A06	Vulnerable and Outdated Components	9th	Up 3 (& Renamed)	Previously titled Using Components with Known Vulnerabilities and was #9 in the Top 10 community survey.
A07	Identification and Authentication Failures	2nd	Down 5 (& Renamed)	Was previously Broken Authentication and is sliding down from the second position.
A08	Software and Data Integrity Failures	-	New	New category for 2019, focusing on making assumptions related to software updates and critical data, without verifying integrity.
A09	Security Logging and Monitoring Failures	10th	Up 1 (Expanded & Renamed)	Was previously A10:2017-Insufficient Logging & Monitoring moving up from #10.
A10	Server-Side Request Forgery	-	New	Server-Side Request Forgery is added newly.

In keeping with prior research in this area, a thorough analysis was conducted to compare the two findings. You can see the results of the analysis in Table 1.

Key Findings

• Both versions are still vulnerable to injection attacks, weak access control, and exposed sensitive data.

•The fact that the XXE vulnerability is no longer included in the 2019 list suggests that its prevalence is decreasing.

• The rising awareness of the problems associated with insecure deserialisation is evidenced by its inclusion on the 2019 list.

• Secure data storage, strict authorisation, and robust authentication are cornerstones of both releases. Web application security incidents may go unnoticed and unresolved due to a lack of adequate logging and monitoring.

Role of OWASP in Shaping Security Practices

Over the years, the OWASP Top 10 has guided practices of online application security. Because it is a resource for cybersecurity education and a point of reference for people all around the world, its impact goes far beyond the list itself. The 2019 edition's upgrades and enhancements demonstrate OWASP's dedication to appropriately capturing the changing threat landscape. The Open Web Application Security Project (OWASP) will remain an invaluable resource for security experts and organisations.

CONCLUSION

Findings from this analysis of the OWASP Top 10 2017 and OWASP Top 10 2019 help shed light on the changing landscape of web application vulnerabilities. Web application security is in a constant state of flux, as evidenced by the persistence of some vulnerabilities despite revised descriptions, the identification of new trends, and the relevance of flexibility. It is recommended that organisations and security professionals use these insights to strengthen their web application security procedures and adapt to the constantly evolving digital ecosystem. That way, they can help with the never-ending battle to keep user data safe in an online environment that is becoming more and more linked. An improved web application security landscape and increased organisation resilience against new web application vulnerabilities can be achieved through implementing the aforementioned suggestions and exploring subsequent research directions.

REFERENCES

[1]. M. Howard, ‘‘The security development lifecycle,’’ Redmond, WA, USA: Microsoft Press, 2006.
[2]. A. Al-Ghamdi, ‘‘A survey on software security testing techniques,’’ Int. J. Comput. Sci. Telecommun., vol. 4, pp. 14–18, Apr. 2013.
[3]. M. Denis, C. Zena, and T. Hayajneh, ‘‘Penetration testing: Concepts, attack methods, and defense strategies,’’ in Proc. IEEE Long Island Syst., Appl. Technol. Conf. (LISAT), Farmingdale, NY, USA, Apr. 2016, pp. 1–6, doi: 10.1109/LISAT.2016.7494156.
[4]. A. Petukhov and D. Kozlov, ‘‘Detecting security vulnerabilities in Web applications using dynamic analysis with penetration testing,’’ in Proc. Appl. Secur. Conf., 2008.
[5]. B. Arkin, S. Stender, and G. McGraw, ‘‘Software penetration testing,’’ IEEE Secur. Privacy Mag., vol. 3, no. 1, pp. 84–87, Jan. 2005, doi: 10.1109/msp.2005.23.
[6]. Y. Arya, A. Bhalotiya, C. Sharma, V. Kag, and S. Snaghvi, ‘‘International journal of engineering sciences & research technology a study of metasploit tool,’’ Arya, vol. 5, p. 2, Feb. 2016.
[7]. A. G. Bacudio, X. Yuan, B. T. Bill Chu, and M. Jones, ‘‘An overview of penetration testing,’’ Int. J. Netw. Secur. Its Appl., vol. 3, no. 6, pp. 19–38, Nov. 2011, doi: 10.5121/ijnsa.2011.3602.
[8]. P. Baker and C. Jervis, ‘‘Early UML model testing using TTCN-3 and the UML testing profile,’’ in Proc. Test., Acad. Ind. Conf. Pract. Res. Techn. (TAICPART-MUTATION), Sep. 2007, pp. 47–54.
[9]. B. Baloglu, ‘‘How to find and fix software vulnerabilities with coverity static analysis,’’ IEEE Cybersecurity Develop. (SecDev), Nov. 2016, p. 153.
[10]. A. Bartel, J. Klein, Y. L. Traon, and M. Monperrus, ‘‘Automatically securing permission-based software by reducing the attack surface: An application to Android,’’ in Proc. 27th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE), Sep. 2012, pp. 274–277.
[11]. D. D. Bertoglio and A. F. Zorzo, ‘‘Overview and open issues on penetration test,’’ J. Brazilian Comput. Soc., vol. 23, no. 1, pp. 1–16, Dec. 2017.
[12]. S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, ‘‘Security, privacy and trust in Internet of Things: The road ahead,’’ Comput. Netw., vol. 76, pp. 146–164, Jan. 2015.
[13]. J. Bo, L. Xiang, and G. Xiaopeng, ‘‘MobileTest: A tool supporting automatic black box test for software on smart mobile devices,’’ in Proc. 2nd Int. Workshop Automat. Softw. Test, May 2007, p. 8.
[14]. B. Kitchenham and S. Charters, ‘‘Guidelines for performing systematic literature reviews in software engineering,’’ Softw. Eng. Group, Tech. Rep., 2007.
[15]. J. Burns, ‘‘Developing secure mobile applications for Android,’’ ISEC, Tech. Rep., 2008.
[16]. C. Carthern, W. Wilson, R. Bedwell, and N. Rivera, ‘‘Introduction to network penetration testing,’’ in Cisco Networks. Berkeley, CA, USA: Apress, pp. 759–772, 2015.
[17]. Y. Cheon, ‘‘Are java programming best practices also best practices for Android,’’ Dept. Comput. Sci., Univ. Texas El Paso, El Paso, TX, USA, Tech. Rep. 16-76, 2016.
[18]. E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, ‘‘Analyzing interapplication communication in Android,’’ in Proc. 9th Int. Conf. Mobile Syst., Appl., services (MobiSys), Jun. 2011, pp. 239–252.
[19]. H. M. Z. A. Shebli and B. D. Beheshti, ‘‘A study on penetration testing process and tools,’’ in Proc. IEEE Long Island Syst., Appl. Technol. Conf. (LISAT), Farmingdale, NY, USA, May 2018, pp. 1–7, doi: 10.1109/LISAT.2018.8378035.
[20]. A. Mosenia and N. K. Jha, ‘‘A comprehensive study of security of Internet-of-Things,’’ IEEE Trans. Emerg. Topics Comput., vol. 5, no. 4, pp. 586–602, Oct. 2017, doi: 10.1109/TETC.2016.2606384.
[21]. G. W. Cross, ‘‘Using a protocol analyzer to introduce communications protocols,’’ in Proc. 10th ACM Conf. SIG-Inf. Technol. Educ. (SIGITE), Oct. 2009, pp. 178–181.
[22]. D. Menoski, P. Mitrevski, and T. Dimovski, ‘‘Evaluating Website security with penetration testing methodology,’’ in Proc. Int. Conf. Appl. Internet Inf. Technol., Zrenjanin, Serbia, 2014.
[23]. J. Dawson and J. T. McDonald, ‘‘Improving penetration testing methodologies for security-based risk assessment,’’ in Proc. Cybersecurity Symp. (CYBERSEC), Apr. 2016, pp. 51–58.