The ISC Int'l Journal of Information Security

November 2021, Volume 13, Number 3 (pp. 59–67)

http://www.isecure-journal.org

Selected Paper at the ICCMIT'21 in Athens, Greece

Open Web Application Security Project Components with Known Vulnerabilities: A Comprehensive Study **

Mohammed S. Albulayhi [1], and Dina M. Ibrahim [1,2,*]

[1] *Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia.*

[2] *Computers and Control Engineering Department, Faculty of Engineering, Tanta University, Tanta, Egypt.*

ARTICLE INFO.

Keywords:
Web Application Security,
OWASP, Vulnerability

Type: Research Article

doi:
10.22042/ISECURE.2021.0.0.0

dor: 20.1001.1.20082045.2021.13.3.7.4

ABSTRACT

The Open Web Application Security Project (OWASP) is a nonprofit organization battling for improvements in software protection and enhancing the security of web applications. Moreover, its goal is to make application security "accessible" so that individuals and organizations can make educated decisions about security threats. The OWASP is a repository of tools and standards for web security studies. OWASP released an annual listing of the top 10 most common vulnerabilities on the web in 2013 and 2017. This research paper proposed a comprehensive study on Components with known vulnerabilities attack, which is the ninth attack (A9) among the top 10 vulnerabilities. Components with known vulnerabilities are the third-party components that the focal system uses as authentication frameworks. Depending on the vulnerability it could range from subtle to seriously bad. This danger arises because the app's modules, like libraries and frameworks, are almost always run with the highest privileges. If a compromised aspect is abused, the hacker's task of causing significant loss of information or server takeover is easier.

1 Introduction

Web application security refers to a range of web servers, web applications, and web service protective procedures, technologies, and approaches, including Internet-based threats-reducing APIs. The protection of data, consumers, or organizations against data theft, business continuity disruptions, or other adverse effects from cybercrime is critical to the protection of web application security. More than three-quarters of all cybercrime apps and their vulnerabilities are calculated according to most estimations. Web application security products and policies aim to protect applications through measures such as web application firewalls (web application firewalls), user multi-factor authentication (MFA), cookie utilization, security and validation to maintain the status of users, and different methods for validating user input to prevent maliciousness before the application processes this input [1]. Open-source technologies and nowhere have transformed the new tech industry is this more apparent than web appli-

* Corresponding author.

**The ICCMIT'21 program committee effort is highly acknowledged for reviewing this paper.

Email addresses: 421100196@qu.edu.sa, d.hussein@qu.edu.sa, dina.mahmoud@f-eng.tanta.edu.eg

ISSN: 2008-2045

cation creation. Thanks to open-source languages, structures, and advancement tools, creators can build rich and sophisticated web apps in a relatively short time and the resources it took in previous years. The Open Web Application Security Project was developed to provide uniform standards for organizations dealing with web application security to help them leverage open-source applications responsibly. The OWASP develops and distributes documents, articles, software, and technology to assist those businesses [2].

OWASP's Ten Best list of program security threats is one such piece of evidence, which details the most popular errors that occur regularly and can be quickly manipulated. With the 2013 upgrade, OWASP A9 has become"Using Components with Known Vulnerabilities", while previous iterations of the list have "Inadequate Transport Layer Protection" as the ninth object [3]. As a result, the focus is on open-source security as never before. When the Equifax hack was revealed in September 2017, the replicated OWASP Ten Best entry may have avoided the breach if the firm had been more vigilant in its use of components with identified flaws. In November 2017, the OWASP topmost ten records were revised, with the Known Vulnerabilities vulnerability remaining in ninth place.

The examples are of utilizing components with known vulnerabilities. By failing to have an identity key, attackers can use whatever web service with full authorization. The Spring Framework for Java-based applications eliminates the necessity of expression language injection vulnerability. The Preventive Mechanisms Identify all components and the database/frameworks versions used in mobile applications are not the only ones. Maintain the most recent versions of all modules. Including online directories, project mailing lists, etc., and adding security wrappers around vulnerable components [4].

Known vulnerabilities are discovered in Open-source components that have been released in the NVD, safety advisories, or problem trackers. Hackers who discover the documents will exploit a flaw from the moment it is released. Applying components with known vulnerabilities is common, according to the OWASP organization [3, 5, 6]. Furthermore, open-source components are so widely used that even development leaders are unaware of what they have. Open-source bugs may have a wide variety of consequences, from mild flaws to other more serious security attacks ever found. The notorious Equifax hack, for example, was triggered by the use of an Apache Struts update that had been considered to be vulnerable since March 2017.

It is known that at least some open source components are present in well over 80% of all applications.

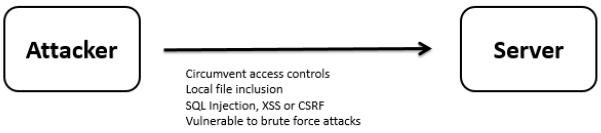

Figure 1. Scenario 1 (courtesy: GBHackers on security)

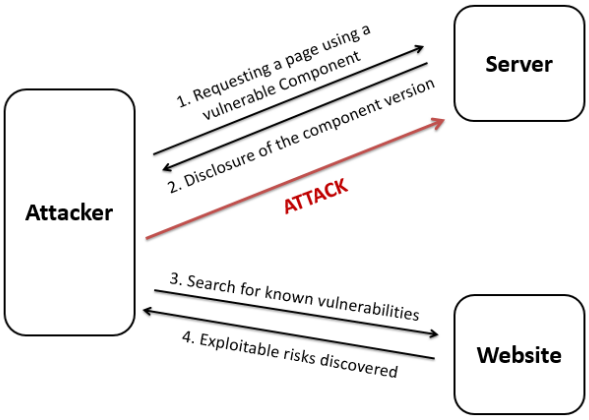

Figure 2. Scenario 2 (courtesy: GBHackers on security)

As a result of their extensive use, third-party modules are an enticing choice for future hackers. When any of the above considerations are considered, it is easy to see why OWASP added "A9: Using Components with Known Vulnerabilities" to their uppermost ten catalogs. Although OWASP recognizes that the easiest way to prevent established security threats is to avoid using third-party modules, they also stress that this would be an impractical choice. Such a strategy will rob a corporation of vital capital while also significantly increasing the expense and duration of future construction programs [7–10].

The risk's exploitability is average; the attacker must search or manually analyze the vulnerable component to find it. However, since most programming teams do not focus on ensuring that their parts/libraries are breakthroughs, the risk is widespread; many systems have these problems. The creators don't know any of the components they're using, and it doesn't matter what style they're using. Injection, compromised access control, XSS, and other vulnerabilities are all conceivable. The effect may be minor to significant [11–13]. Figure 1 and Figure 2 gives the two type of attacks encountered in web security mechanism under the known vulnerabilities. In the first scenario, explained in Figure 1, the intruder and the webserver are now attempting to manipulate a defective component on the server; a sensitive component is simply a commodity or library that is potentially vulnerable [14, 15].

While in the second scenario, as illustrated if Fig-

ure 2, in this case, the attacker attempts to submit the website, which, let's say, loads a tab that contains the vulnerability. The website replies to the inquiry, revealing the insecure components as well. After identifying the insecure components and versions, the attacker can scan the Internet for known vulnerabilities. Regardless, attackers can easily identify risks associated with insecure components on the internet, giving them insight on how to circumvent this danger [11, 16]. The intruder then proceeds to initiate an assault on the website. Table 1 depicts the consolidation of the different losses that occurred by the components with known vulnerabilities.

The rest of this paper is organized as follows: Section 2 presents a comprehensive literature analysis of Components with Known Vulnerabilities. In Section 3, an investigation of the several examples of using components with known vulnerabilities and their prevention mechanism is demonstrated. Discussions and findings are presented in Section 4. Finally, conclusion is given in Section 5.

2 Literature Review of Components with Known Vulnerabilities

2.1 Establish an Internal Security Policy

Many businesses fail to develop clear guidelines to regulate their software creation, particularly in which open-source components are appropriate. This is particularly critical because not all vulnerabilities are created equal [14]. "The United States government registry of rules vulnerability management data", according to the National Vulnerability Database (NVD). Vulnerabilities are released to the NVD using the Common Vulnerabilities and Exposure (CVE) scheme, which provides a centralized location to monitor security-related program flaws.

Removing every third-party component which has a flaw is not always realistic, and many open-source developers sometimes do not release security fixes against older versions of software, instead merely fixing the problem in future updates. The CVE rating system can be beneficial in this situation. When the CVE number is small enough, there could be no reason to stop when using the affected component while waiting for a new update [17–19]. As a result, implementing a company-wide approach will be highly beneficial to the development team and the bottom line. What score can be used to determine when a component's usage is halted? What kinds of weaknesses would be tolerated? Ensure that all of the creators are all on the same page, whether for open-source or proprietary technology, to ensure a clear and stable solution.

2.2 Identifying Affected Components

It is one thing to have a strategy in place; it is quite another to stick to it. The interdependence of open-source applications is one aspect leading to this issue. According to research conducted by White Source, "Indirect open-source dependencies are used in 91 of program projects [20–23]. The typical project uses 64 different libraries, each with its own set of licenses". "In 65 percent of instances, open-source modules come with external dependencies which are already subject to a separate license", says the study. Finding all relevant CVEs manually can be challenging, especially with several interdependencies, license variants, and individual libraries. Often the connection between that CVE and the impacted program is not immediately apparent.

Therefore, an open-source management solution is so vital for a modern company. Through evaluating the tasks and the open-source determination, White Source has streamlined this process. Open-source modules are being used, and the related CVEs are being cross-referenced to see which, if any, are sensitive. Moreover, White Source's monitoring system constantly checks the applications to guarantee that you are alerted as quickly as possible if any bugs occur. Using the NVD, learning CVE ratings, developing a company-wide growth strategy, and using an integrated open-source management tool are all things that can be done [20].

This danger arises because the app's modules, like libraries and frameworks, are approximately forever operated through complete rights. If a compromised aspect is abused, the hacker's task of causing significant loss of information or server takeover is easier.

The following are a few explanations of how to use modules that have known flaws –

- By failing to have an identity key, attackers can have any web application with full authorization.
- The Spring Framework for Java-based applications introduces eliminating the necessity for Expression Language injection vulnerability.

3 Preventive Mechanisms for Components with Known Vulnerabilities

This section investigates several examples of using components with known vulnerabilities and their prevention mechanism are demonstrated. No matter how secure your code is, attackers can exploit APIs, dependencies, and other third-party components if they are not themselves secure.

Table 1. Different losses occurred by the components with known vulnerabilities

	Human - social	Physical	Economic	Cultural environment
Direct losses	Fatalities	Structural damage or collapse to buildings	Interruption of business due to damage to buildings and infrastructure	Sedimentation
	Injuries			Pollution
	Loss of income or employment	Non- structural damage and damage to contents	Loss of productive workforce through fatalities, injuries and relief	Endangered species
	Homelessness	Structural damage infrastructure		Destruction of cultural heritage
Indirect losses	Diseases	progressive deterioration of damaged buildings and infrastructure which are not repaired	economics losses due to short term disruption of activities	loss of biodiversity
	permanent disability		long term economic losses	loss of cultural diversity
	psychological impact		insurance losses weakening the insurance market	
	loss of social cohesion due to disruption of community		less investment	
	political unrest		capital costs of repair	
			reduction in tourism	

- Not only databases or frameworks but also all modules and versions used in web apps should be identified.
- Maintain the most recent versions of all modules, including online directories, project mailing lists, and so on.
- Wrap fragile materials in security wrappers.

Table 2 illustrates some terminologies used in components with vulnerabilities (Courtesy: Tutorials Point), which depicts the Threat Agents, Attack Vectors, Security weaknesses, Technical Impact, and Business Impacts associated with this vulnerability.

Most Web apps stored the data in SQL databases. Almost every Web application has a SQL database running in the background. SQL syntax, like most other languages, allows database commands to be combined with customer data. If developers aren't careful, user data can be interpreted as commands, allowing remote users to do more than just input data into Web applications; they can even run arbitrary commands on the database.

4 Discussion and Findings

When a company is hacked, you would like to think that the perpetrator created a new exploit based on a zero-day flaw that no one can defend against. However, it is much more plausible that the perpetrator took advantage of well-known flaws that had been lurking in their programs throughout the years, if not years. Attackers use automated scripts to scan web applications for suspected flaws and then hack the flaws they find. A large percentage of attackers would not spend sufficient time and effort necessary to create an exceptional exploit to gain access to your systems, particularly if they can quickly identify security vulnerabilities in several of your frameworks or their dependencies. Using components with documented bugs has resulted in some of the most severe attacks to date, as evidenced by its long-term ranking upon this OWASP Top 10 list [24]. The sophistication, including its web application itself, is an extremely familiar risk to the enterprise rather than an unknown vulnerability. Most of the programming in a typical web application is not written from scratch or in-house. APIs, microservices, repositories, open-source, and legacy technology are all cobbled together as a loose array of third-party developed and bundled components. Moreover, such components are often a framework with several subcomponents that must remain stable and complete for the entire web application [24]. Many security bugs are caused by ancillary program dependencies that have identified problems that have been ignored, acknowledged as a possibility, or are not correctly handled. These flaws are frequently other OWASP Top 10 problems like Cross-Site Scripting and Injection.

Online technologies are often regarded as forerun-

Table 2. Terminologies in components with vulnerabilities

Terminology	Definition
Threat Agents	The term threat agent is used to denote an individual or group that can manifest a threat as a party that causes harm to an organization, or seeks to do so. Threat actors may relate to their targets internally, externally, or as partners, and their goals may differ. External threats can be individuals, groups, or organizations and sometimes hostile agents. Hacktivism is a form of "hacktivism" in which people try to hack into computers and steal data from them. The intent and method targeted at the intentional exploitation of a vulnerability or a situation and the method that may accidentally trigger a vulnerability. Synonymous with Threat Agent.
Attaché's Approach	The attacker identifies a weak component through scanning or annual analysis. It gets more complex to identify if the used component is deep in the application Attackers need to access only a few accounts or just a single system admin account. This can authorize money according to the scope of the application. Misconfiguration attacks: as a result of faulty administration of such systems, hackers have a field day around an incorrectly configured system. Due to the intricacies of today's systems, not very well-trained administrators are caught by hackers. Operating systems attack: because of the intricacies of current networks, operating systems operate several services, ports, and access modalities. Large service deals are also maintained when the default settings of operating systems in the installation process are implemented. Hackers, therefore, search for and exploit flaws in operating systems to obtain unauthorized access to network systems.
Security weakness	Virtually all the applications have these issues because most development teams don't focus on ensuring their components/libraries are up to date and are often used in cybersecurity. It is crucial to understand the distinction between these terminologies. It permits enterprises to implement the cybersecurity operations and controls correctly, and document and evaluate them. Here, we examine safety vulnerabilities in more detail. Suppose an attacker can identify the unsafe components of a program. In this situation, the assault is immediately misused as exploit methods are already on the internet and must be employed by the attacker. This can result in a slight consequence, serious or even complete breach of data, or even a takeover of the server/host for companies.
How to spot	Easier when the library file is at the topmost layer of the app. It became difficult as it becomes deeper Track your code security against standard OWASP & SANS categories. While some known vulnerabilities have little effect on the use of known vulnerabilities in components, some of the largest infringements yet have been reported. Perhaps this danger should be at the top of the list depending on the assets you protect.
Technical impact	A full range of weaknesses is possible including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data. Making Your Information Technology Effective and Keeping It. In the application, components like libraries and frameworks nearly usually work with full rights. It makes it easier for a hacker to inflict serious data loss or server takeover if a susceptible component is used.
Business impact	It could be trivial or it could mean a complete compromise Loss of customer confidence and credibility Data loss and goodwill damage Experts claim that online financial transactions are the companies most at risk. Business loss Closing temporarily or permanently Exposure to arbitration or legal proceedings Headlines for all the wrong reasons

ners of the modern age. We think about digital goods and facilities as brand-new, fully functional machines. The issue is that these robots look very much like pickup trucks. There have been gaps in the upholstery, common engine problems, and rusted parts. Imagine that many pieces of such a startup machine rust, wear out, and crack at varying rates to get a clearer view of web applications [25]. These components deteriorate gradually over time, while others become obsolete almost instantly. Sections of it are to be replaced regularly. It would not work correctly otherwise. It is the same for tech in that it has to be maintained to continue to function correctly. Software that is readily accessible protects the confidentiality of the records, which keeps private information private is readily available. You can make apps more usable by making them more stable.

The objective is to dedicate time and energy to a vulnerability detection strategy. The first step is to learn everything you can about the web applications you already have, including what they are doing, how they deal with one another, and so on, and which libraries they depend on. Without a clear understanding of what a web application does and how knowledge flows through it, it is difficult to make significant scale security changes. It would be beneficial if you already took note of the collection used within a client or server program [26, 27].

You will need to cut the fat until you know the application's intended feature. Unused dependencies, redundant functionality, unreferenced files, and metadata must be removed. This reduces the security vulnerabilities of the submission and makes it easier to manage. Several resources are available to help diagnose this, including retire.js and dependability. Many package managers, such as npm, can inspect dependencies. There is more than enough open-source tooling available in this area as well. Continuing to monitor the programs for end-of-life (EOL) apps and unmaintained systems and libraries would be beneficial. These dependencies must be eliminated as soon as they are discovered. When all of the fat has been eliminated, a procedure should be put in place to upgrade any components that are old or considered to be insecure.

Finally, we must avoid incorporating insecure elements into the web program. Just obtain new dependencies with authoritative reports through secure channels when bringing them in. New modules can still be enabled if they include functionality not already provided by anyone else in the app. It is unrealistic to expect never to use third-party dependencies; by limiting oneself to components written and managed in-house, we will become inefficient and sacrifice the competitive advantage.

We will have a mechanism for deleting components from our framework that have identified bugs if we follow these procedures. There is another advantage of having this mechanism in place: we will monitor and fix any glitches that arise with the submission. Security flaws are glitches that compromise the availability, honesty, or secrecy of a device. One kind of bug fix is security fixes. We will keep the web application stable and usable by implementing a solid patch management program. Cyber-terrorism vulnerabilities and attacks emerge every day, placing Cyber-terrorism vulnerabilities and attacks emerge every day, placing customers in danger except some of them are zero-day vulnerabilities. Most of these risks are caused by software dependencies, such as using libraries and applications that have already been identified as insecure or become vulnerable due to unencrypted software patches or changes that are not introduced on time.

Most third-party modules and frameworks used in an application are run with maximum rights. The attackers may use automatic search tools or do a manual inspection of the program to detect bugs. Suppose they discover that the application uses a previously identified feature as insecure. In that case, they can quickly develop strategies to mitigate that vulnerability and determine the effect and benefit ahead of time. To find the dependencies, the hackers use fingerprinting techniques such as searching for recognized HTML components, causing errors, forcing browsing, and so on.

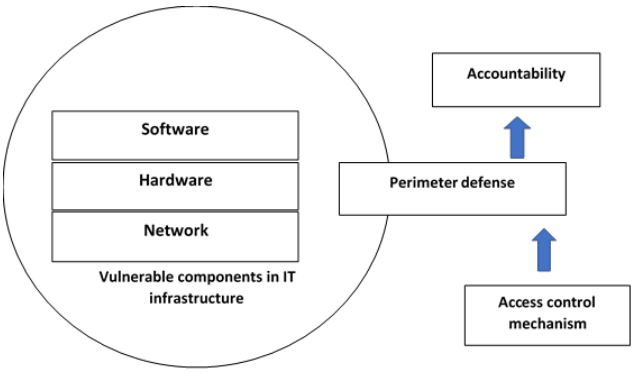

Figure 3. Vulnerability components in IT infrastructure

4.1 Impact on Businesses Using Components with Known Vulnerabilities

Along with it being easily available, this weakness poses a significant danger for any corporation. Suppose an attacker can identify the insecure components that a given program uses. In that case, the attack is quickly abused because the exploit methods are now available on the internet, and the attacker has to use them. This can result in a minor effect, severe or even complete data breach, or even server/host takeover for organizations.

This flaw can easily circumvent device security defenses and serve as a pivoting point for various other attacks, such as invoking a web server with complete approval without having a permission token or executing code remotely. Injection, XSS, and compromised access control are all vulnerabilities when using insecure elements, as in Figure 3.

System vulnerability is a failure in the design or execution, via loss of confidentiality, integrity, or availability, of an information system that can be used deliberately or involuntarily to adversely affect the operations or assets of an organization. System vulnerabilities are based on application loopholes/software or restrictions connected to human use. That shows that no system can potentially achieve the status of total immunity against possible violations of security. It is commonly stated that there is nothing that can be done to get a hacker inside your system or network. Maybe, all the possible ways to make the hacker more difficult to break the security of the system are to be exhausted. This study addressed certain ways to expose networks to malicious attacks and suggested techniques to curb or contain these possibilities. In

the future, we will look into the cloud and distributed computing vulnerabilities [28, 29].

4.2 Past Victims and Prevention

Amongst the most common vulnerabilities is a breach caused by existing insecure components, as in Figure 4. Listed below only samples from a lot of small names on the survivor list:

- Hack at a US credit bureau "Equifax" due to an unsecured "Apache Struts web system CVE-2017-5638"
- Infringement at Mossack Fonesca (the law firm that handled the Panama Papers): an unsecured edition of Drupal CMS was used.
- Breach of the Ubuntu forums - unpatched Forum runner add-on
- Vertical Scope (internet media company) - old vBulletin forum software was used. The protection flaw here is that most production teams do not check whether the elements/collections remain advanced
- Citations: know the purpose and arrange thorough citations of all the modules (Operating system, HTTP server, libraries, network resources, and so on) and latest editions utilized through the program, and get convinced to be supported.
- Track and validate compliance evaluations daily.
- Assessments: conduct internal and external vulnerability assessments and penetration testing regularly to ensure that the application is safe.
- Patch management scheme: implement a proper patch management scheme, ensuring that only reputable suppliers provide upgrades and security patches and remove any unnecessary or redundant modules to harden the program.
- Check that the modules and the subcomponents are secure and up to date.
    - Make use of the OWASP Dependency Framework. Check to see if all of the modules you are using have a widely known flaw.
    - Install a Web Application Firewall to provide layered protection.

5 Conclusion

On the HTTP protocol level, we can use AWS WAF to prevent existing web apps against different attack vectors properly. In terms of OWASP security bugs, AWS WAF is incredibly efficient at minimizing weaknesses to the point that these attack patterns can be detected in HTTP queries. We may also combine AWS WAF's capabilities with other AWS providers to build robust security automation. The AWS WAF Security Automations is a group of such tools found on our website. These tools allow us to create a series of defenses that can adapt to the various types of attacks that your applications can face. In the form of a Cloud Formation template, the solution offers several easy-to-deploy automation for rate-based IP blacklisting, credibility list IP blacklisting, scanner and probe prevention, bot and scraper tracking, and blocking. It is critical to ensure that our project does not have any high-risk bugs added by third-party modules, which developers frequently ignore. Setting up an automated method to compare dependencies to a vulnerability database will go a long way toward avoiding security breaches.

References

[1] Akbar Iskandar, Muhammad Resa Fahlepi Tuasamu, Suryadi Syamsu, M Mansyur, Tri Listyorini, Sulfikar Sallu, S Supriyono, Kundharu Saddhono, Darmawan Napitupulu, and Robbi Rahim. Web based testing application security system using semantic comparison method. In *IOP Conference Series: Materials Science and Engineering*, volume 420, page 012122. IOP Publishing, 2018.

[2] Sangeeta Nagpure and Sonal Kurkure. Vulnerability assessment and penetration testing of web application. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pages 1–6. IEEE, 2017.

[3] A Muller, M Meucci, E Keary, D Cuthbert, et al. Owasp testing guide 4.0. *Maryland (USA): The OWASP Foundation*, 4:165–166, 2014.

[4] OWASP. Owasp top 10 no. 9 using components with known vulnerabilities, 2021. Accessed 18 February 2022.

[5] Vincent C Hu, Michaela Iorga, Wei Bao, Ang Li, Qinghua Li, Antonios Gouglidis, et al. General access control guidance for cloud systems. *NIST Special Publication*, 800(210):50–2ex, 2020.

[6] Jasper van Vliet. Direct and indirect loss of natural area from urban expansion. *Nature Sustainability*, 2(8):755–763, 2019.

[7] Narayanan Anantharaman and Bharati Wukkadada. Identifying the usage of known vulnerabilities components based on owasp a9. In *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pages 88–91. IEEE, 2020.

[8] Philip Sarrel, David Portman, Patrick Lefebvre, Marie-Hélène Lafeuille, Amanda Melina Grittner, Jonathan Fortier, Jonathan Gravel, Mei Sheng Duh, and Peter M Aupperle. Incremental direct and indirect costs of untreated vasomotor symptoms. *Menopause*, 22(3):260–266, 2015.

[9] Tony B Amos, Neeta Tandon, Patrick Lefeb-

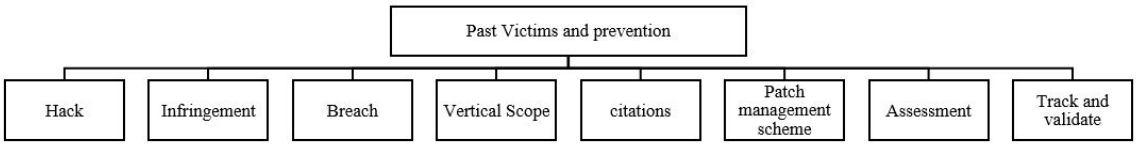

Figure 4. Samples of past victims and prevention cases

vre, Dominic Pilon, Rhiannon L Kamstra, Irina Pivneva, and Paul E Greenberg. Direct and indirect cost burden and change of employment status in treatment-resistant depression: a matched-cohort study using a us commercial claims database. *The Journal of clinical psychiatry*, 79(2):5360, 2018.

[10] Flora Angeletaki, Andreas Gkogkos, Efstratios Papazoglou, and Dimitrios Kloukos. Direct versus indirect inlay/onlay composite restorations in posterior teeth. a systematic review and meta-analysis. *Journal of dentistry*, 53:12–21, 2016.

[11] VS Kumar. Ethical hacking and penetration testing strategies. *International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)*, 11(2):0976–1353, 2014.

[12] Te-Shun Chou. Security threats on cloud computing vulnerabilities. *AIRCC's International Journal of Computer Science and Information Technology*, 5(3):79–88, 2013.

[13] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 171–180. IEEE, 2008.

[14] OWASP. Top 10–2017 a9-using components with known vulnerabilities, 2021. Accessed 18 February 2022.

[15] Dimitris E Simos, Jovan Zivanovic, and Manuel Leithner. Automated combinatorial testing for detecting sql vulnerabilities in web applications. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 55–61. IEEE, 2019.

[16] Kannan Balasubramanian. Web application vulnerabilities and their countermeasures. In *Cryptographic Solutions for Secure Online Banking and Commerce*, pages 209–239. IGI Global, 2016.

[17] SE Idrissi, N Berbiche, F Guerouate, and M Shibi. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076, 2017.

[18] Tomohisa Ishikawa and Kouichi Sakurai. Parameter manipulation attack prevention and detection by using web application deception proxy. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, pages 1–9, 2017.

[19] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In *2006 Securecomm and Workshops*, pages 1–10. IEEE, 2006.

[20] Henrik Plate, Serena Elisa Ponta, and Antonino Sabetta. Impact assessment for vulnerabilities in open-source software libraries. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 411–420. IEEE, 2015.

[21] Muhammad Noman, Muhammad Iqbal, and Amir Manzoor. A survey on detection and prevention of web vulnerabilities. *International Journal of Advanced Computer Science and Applications*, 11(6):521–540, 2020.

[22] Emil Semastin, Sami Azam, Bharanidharan Shanmugam, Krishnan Kannoorpatti, Mirjam Jonokman, Ganthan Narayana Samy, and Sundresan Perumal. Preventive measures for cross site request forgery attacks on web-based applications. *International Journal of Engineering and Technology (UAE)*, 2018.

[23] S Shalini and S Usha. Prevention of cross-site scripting attacks (xss) on web applications in the client side. *International Journal of Computer Science Issues (IJCSI)*, 8(4):650, 2011.

[24] OWASP. Owasp press release, 2021. Accessed 18 February 2022.

[25] OWASP. Clm press release, 2021. Accessed 18 February 2022.

[26] OWASP. Sonatype's owasp a9 blog post, 2021. Accessed 18 February 2022.

[27] Timothy Casey, Patrick Koeberl, and Claire Vishik. Defining threat agents: Towards a more complete threat analysis. In *ISSE 2010 Securing Electronic Business Processes*, pages 214–225. Springer, 2011.

[28] Raymond AJ Brown and Peter D Renshaw. Collective argumentation: A sociocultural approach to reframing classroom teaching and learning. 2000.

[29] Oludele Awodele, Ernest Enyinnaya Onuiri, and Samuel O Okolie. Vulnerabilities in network infrastructures and prevention/containment mea-

sures. In *Proceedings of Informing Science & IT Education Conference (InSITE)*, 2012.

Mohammed Sulaiman M. Albulayhi graduate as a computer engineer since 2016, he was working as a teaching assistance in prince Sattam University. He is currently a master student in cybersecurity program at Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia.

Dina M. Ibrahim Assistant Professor at Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia from September 2015 till now. In addition, Dina works as an Assistant Professor in the Computers and Control Engineering Department, Faculty of Engineering, Tanta University, Egypt. She was born in the United Arab Emirates, and her B.Sc., M.Sc., and Ph.D. degrees have taken from the Computers and Control Engineering Department, Faculty of Engineering, Tanta University in 2002, 2008, and 2014, respectively. Dina works as a consultant engineer, then a Database administrator, and finally acts as a vice manager on Management Information Systems (MIS) Project, Tanta University, Egypt, from 2008 until 2014. Her research interests include networking, wireless communications, machine learning, security, and the Internet of Things. She is serving as a reviewer in Wireless Network (WINE) the Journal of Mobile Communication, Computation, and Information since 2015, and recently in the International Journal of Supply and Operations Management (IJSOM). Dina has also acted as a Co-Chair of the International Technical Committee for the Middle East Region of the ICCMIT conference since 2020.