electronics

Review

A Survey on Web Application Penetration Testing

Esra Abdullatif Altulaihan *, Abrar Alismail and Mounir Frikha

Department of Computer Networks and Communications, King Faisal University, Al-Ahsa 31982, Saudi Arabia
* Correspondence: 221400737@student.kfu.edu.sa

Abstract: Websites are becoming increasingly effective communication tools. Nevertheless, web applications are vulnerable to attack and can give attackers access to sensitive information or unauthorized access to accounts. The number of vulnerabilities in web applications has increased dramatically over the past decade. Many are due to improper validation and sanitization of input. Identifying these vulnerabilities is essential for developing high-quality, secure web applications. Whenever a website is released to the public, it is required to have had penetration testing to a certain standard to ensure the security of the information. Application-level security vulnerability detection is possible for many commercial and open-source applications. However, developers are curious about which tools detect security vulnerabilities and how quickly they do so. The purpose of this study is to discuss penetration testing and how it can be implemented. This paper also explores the hazards and vulnerabilities associated with the web environment as well as the protective measures that can be taken. In addition, a comprehensive review and comparison of common web penetration testing tools is provided. The aim of this paper is to help web penetration testers choose a technology that is optimal for their requirements. The paper also sets out to guide and provide recommendations to users for choosing the best web penetration test tool and increasing their awareness of secure web environments. The study results indicate that not all web penetration testing tools offer the same features and that combining analysis tools can provide detailed information about web vulnerabilities.

Keywords: web application pen-testing; pen-testing tools; vulnerability; security; mitigation techniques; threats

Citation: Altulaihan, E.A.; Alismail, A.; Frikha, M. A Survey on Web Application Penetration Testing. *Electronics* **2023**, *12*, 1229. https://doi.org/10.3390/electronics12051229

Academic Editors: Marko Horvat, Igor Mekterović, Manuel Palomo-Duarte and Juan Antonio Caballero-Hernandez

Received: 1 February 2023
Revised: 26 February 2023
Accepted: 27 February 2023
Published: 4 March 2023

1. Introduction

In the era of information and digitalization, web applications are essential. The number of people using the Internet and web applications has recently increased. Due to their nature, web applications and services are prone to security risks because they are connected to the Internet [1] As the Internet and web applications become more widespread, every web application needs an adequate level of security to prevent cyberattacks and store information safely. Various logical and technical vulnerabilities can affect web applications. Technical vulnerabilities include structured query language (SQL) injection, cross-site scripting (XSS), remote file inclusion, and local file inclusion. These vulnerabilities compromise the security of web applications and can be caused by poor programming or an outdated system. Web applications go through rapid development phases with short turnaround times, making it a challenge to eliminate vulnerabilities. Despite the difficulty of eliminating these vulnerabilities, it is important to ensure that any web applications are kept up to date and securely coded. A penetration test, which is a technique for gaining access to a system to secure it, can be used to analyze the vulnerability of a web application. To conduct this type of testing legally, we must ask permission from the application's owner; as a result, penetration testing is effective in addressing network security issues. Furthermore, web penetration testing refers to testing web-based applications, including thin client applications, file transfers, appliances, and portals, to discover vulnerabilities prone to exploitation, check for the appropriate controls, and conduct probing and vulnerability analyses [2]. To stay safe against cyber-attacks, penetration testing can be used to assess the effectiveness and ineffectiveness of web application security arrangements.

In a penetration test, the target systems and the goal are identified, then the information is reviewed, and measures are taken to achieve the goals. By performing a penetration test, one can determine if a system is vulnerable to attack, if its defenses are adequate, and if any defenses have been overcome. This information is paired with an accurate assessment of potential impacts on the organization and a range of technical and procedural countermeasures to mitigate them [3]. Once the vulnerability scan of a host has been completed, the results of these tests or attacks are documented and presented as a report to the system owner. A penetration test can be conducted using various tools, so selecting the right one is very important.

During penetration testing, it is important to implement secure coding practices, educate web application developers, and implement automated security scans. So, organizations can be better prepared to prevent malicious attacks [4]. Moreover, during penetration testing, security measures must be improved to prevent harmful actors from exploiting these vulnerabilities. In the case of [5], while particular security measures, such as encryption and authentication procedures, were installed, they were not fully employed. Additionally, other steps, such as the use of firewalls and intrusion detection systems, could have been taken to strengthen the security of the websites. These security procedures could have offered a stronger defense against prospective attacks and notified the company of any suspicious activity on its websites. Additionally, the use of secure coding practices and regular security audits was recommended to ensure that any newly discovered vulnerabilities would be identified and patched promptly.

Web application penetration testing is a necessity today. In contrast to other types of penetration testing, website penetration testing usually focuses on a specific target and is more detailed. The main purpose of this type of testing is to identify vulnerabilities and cybersecurity risks in websites, their databases, and their code, as well as their backend networks. To protect a website from security flaws, one needs to take an in-depth approach. It is crucial to penetration-test all web applications before they go online and become prone to hacking by black hat cyber warriors. There is a constant hunt for vulnerabilities in web apps by hackers, and it is imperative to learn the hackers' methodologies to mitigate their attacks. There are numerous tools available on the market for achieving the goal of web application pen testing, and they have varying degrees of effectiveness and provide quick and easy results. Consequently, individuals and organizations must decide which tool is the most effective for performing a web penetration test. The increasing number of cyberattacks in the web environment necessitates the creation of new technologies to ensure a secure environment. In this article, we go beyond simply advocating for a technical solution to discuss the challenges concerning security in web applications. This article aims to provide a comprehensive review of penetration testing approaches and tools used for web applications and to analyze the previous literature in this area and determine the advantages and limitations of each solution proposed. In addition, it sets out to provide recommendations on how to select the appropriate tool for conducting web penetration tests as well as recommendations for future research in this area. This article is important from both a scientific and an industrial perspective. It may be helpful for penetration testers to review the results that are presented to make better decisions. It may also be beneficial for future researchers in this area to have a clear picture of the limitations and future directions.

Therefore, this study discusses penetration testing and how to implement it. Furthermore, it examines the vulnerabilities, dangers, and hazards of the web environment, as well as the protective solutions that may be used. A comprehensive assessment and comparison of common web penetration testing tools is also provided. We aim to help web penetration testers choose a technology that is optimal for their requirements. As a result of the study's literature review and selection and analysis of relevant research, individuals and organizations will become more aware of the best tools for performing web penetration tests. Therefore, this study aims to achieve the following:

- Highlight the most common vulnerabilities and threats targeting web applications.
- Review and analyze the literature on web penetration testing and its associated methods.
- Describe the recent mitigation techniques to defend against web application threats.
- Review the available tools for conducting web penetration tests and make comparisons between them.
- Provide recommendations for individuals and businesses on how to decide which tool is the best for performing web penetration tests.

This paper is organized as follows. Section 2 presents the methodology. Section 3 defines penetration testing and explains its types. Section 4 presents and analyzes the related work. Section 5 discusses web application vulnerabilities and techniques to mitigate them. Section 6 reviews web penetration tests and compares the tools that can be used for it. Section 7 concludes the research and suggests future work.

Penetration Testing

Penetration testing is a security evaluation process that simulates an attack by an ethical hacker on a network or computer system. It differs from hacking in the sense that it is done with a license under a signed contract with an organization or company, and the output is provided as a report. Penetration tests are designed to improve data security [1], and the information and weaknesses that they identify remain confidential until all deficiencies have been resolved [6]. As part of penetration testing, developers check the vulnerability and exploitation of an organization's system to create a secure system that meets the organization's requirements. Keeping track of the severity of security issues is very important for any organization or company.

It is possible to perform a penetration test manually or automatically. In a manual test, everything needs to be controlled by a skilled and experienced tester team, and a physical presence is required throughout the course of the test. As a result, this option is not affordable. An automatic penetration test is a simple and safe way to carry out all penetration test tasks. In addition, since most of the work is automated, it is more time-efficient. A further advantage of this type of test is that the parameters can be reused. Table 1 shows the fundamental differences between manual and automated penetration tests [1,6].

Table 1. Manual vs. Automated Penetration Tests.

Criteria	Manual Penetration Testing	Automated Penetration Testing
Testing process	The process is manual, non-standard, and capital intensive; expensive to customize.	Easy to use, provides clear, actionable reports, and eliminates errors and tedious manual tasks.
Network modification	Results in numerous system modifications.	There is no change in the systems.
Exploiting development and management	Maintaining an exploit database is time-consuming and requires considerable expertise. To achieve cross-platform functionality, it is necessary to rewrite and port code.	All exploits are developed and maintained by the product vendor. For maximum effectiveness, exploits are continuously updated. These exploits have been professionally developed and thoroughly tested, and they are safe to use. Various platforms and attack vectors are used in the development of exploits.
Reporting	Needs significant effort to record and collate all results manually. It also requires all reports to be manually generated.	Reports can be customized and include comprehensive histories and findings.
Clean-up	Every time a vulnerability is discovered, the tester must manually undo the changes.	Clean-up solutions are offered by automated testing products.

Table 1. *Cont.*

Criteria	Manual Penetration Testing	Automated Penetration Testing
Logging/auditing	The process is slow, cumbersome, and often inaccurate.	All activities are automatically recorded.
Training	Testing methods that are not standardized and ad hoc must be learned by testers.	Testing with automated tools is easier than testing manually.

Based on this comparison, advanced security specialists with many years of experience are needed to perform the complex manual process of penetration testing. Manual testing is time-consuming and expensive, and competent penetration testers are hard to find. Penetration testers can create an automated tool that combines their experience. Non-expert users, rather than a penetration team, can perform a security scan using automated tools to get a picture of the situation in the organization's system.

Traditionally, manual penetration testing was an integral part of application security. Although manual web application penetration testing is highly effective, it is also time-consuming and expensive, which limits its viability and scalability. Although automated penetration testing can help integrate penetration testing more cost-effectively into software development lifecycles, its testing tools may still be monitored by security professionals. Nevertheless, the tools can scan far more applications in less time, bringing down costs and preventing development deadline delays.

2. Methodology

The search was guided by the four stages of PRISMA. During the identification stage, the Saudi Digital Library and Google Scholar databases were searched using the inclusion and exclusion criteria in Table 2. The papers that were included were those that described penetration testing tools, cybersecurity threats, and the web environment and were published between January 2018 and December 2022 in academic journals or conference proceedings, which were listed as the source type.

Table 2. Inclusion and Exclusion Criteria.

Inclusion Criteria	Exclusion Criteria
Papers describing cybersecurity threats or penetration testing tools in the web environment	Papers that do not address risks or penetration testing tools.
Papers published between January 2018 and December 2022	Papers that are not written in English
Papers published in academic journals or conference papers	Papers that are not available online

Papers that were excluded were those that did not address risks or penetration testing tools in the web environment; were not written in English; and were not directly relevant to web cybersecurity threats or penetration testing tools. Furthermore, some documents were not available online.

Figure 1 explains the PRISMA methodology. The PRISMA flow diagram passes through four stages. The identification stage is when items are identified for review, and during this stage, the papers were identified from two main sources, namely the Saudi Digital Library and Google Scholar. The screening stage is when the papers are screened and selected for review. The eligibility stage is when the papers are eligible to be included. The inclusion stage yields a list of studies to be included in the systematic review. Figure 1 shows that 22,280 articles were selected during the identification stage after duplicates had been removed. At the screening stage, 16,810 articles were reviewed for title and abstract, and all but 680 were rejected at the eligibility stage because they did not closely fulfill the requirements. In the inclusion stage, 667 were rejected, leaving 13 for review.

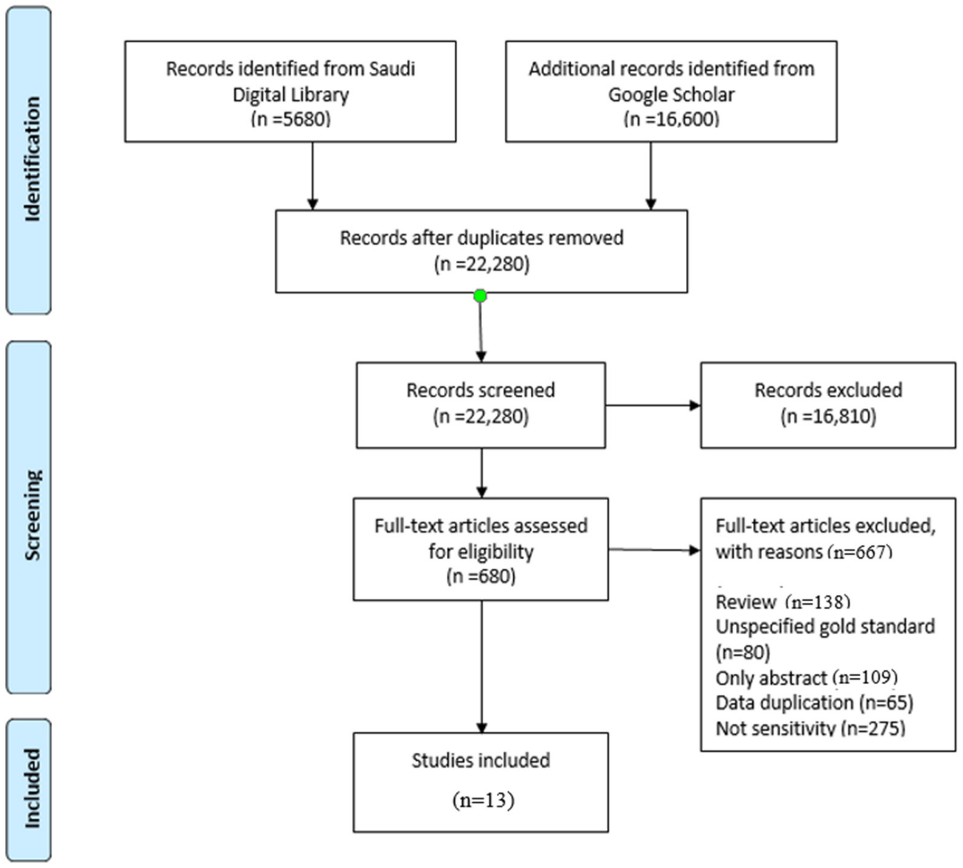
Identification
Screening
Included
Records identified from Saudi Digital Library (n =5680)
Additional records identified from Google Scholar (n =16,600)
Records after duplicates removed (n =22,280)
Records screened (n =22,280)
Records excluded (n =16,810)
Full-text articles assessed for eligibility (n =680)
Full-text articles excluded, with reasons (n=667)
Review (n=138)
Unspecified gold standard (n=80)
Only abstract (n=109)
Data duplication (n=65)
Not sensitivity (n=275)
Studies included (n=13)

Figure 1. PRISMA flow chart.

3. Literature Review

Mirjalili et al. [1] proposed the design and development of a distributed framework to automate web pentation testing, the major components of which are an operational unit called an executor that conducts attacks, and a control unit called an orchestrator that orchestrates them across consecutive stages. The authors defined the general activities carried out during a penetration test and presented a method for integrating the attackers that execute such jobs. Moreover, they defined a flexible method for integrating external tools to achieve the desired hacking goals by mapping vulnerabilities to the framework's integrated tools. To realize the full potential of this distributed framework, they presented a suite of tools with a web-based user interface that integrates seamlessly into their system. This framework has many advantages, such as scalability, a distributed nature, and ease of use, which allow the user to rapidly develop and deploy attacks that leverage the wealth of the available resources. This framework provides a comprehensive solution for building distributed hacking frameworks, allowing users to access, modify, and integrate tools into their framework with minimal effort. It enables users to maximize the effectiveness of their attacks and quickly respond to changing conditions to capitalize on vulnerabilities before they are patched. As a result, the framework allows users to be nimble and agile when exploiting vulnerabilities, enabling them to stay ahead of their adversaries. This, in turn, provides users with unprecedented control and flexibility over their hacking activities. In conclusion, the distributed hacking framework provides an invaluable resource for users looking to enhance their cyber security. Despite the benefits of distributed hacking, it suffers from process synchronization, although global knowledge is required; resource management, since global security knowledge is required; fault tolerance; and error recovery.

Fredj et al. [7] covered the 10 most significant attacks for web applications based on the best-known web vulnerabilities disclosed by the Open Web Application Security Project (OWASP) project. Furthermore, they discussed the primary recommended techniques for mitigating web threats and the associated countermeasures. Their report provided an overview of the OWASP Top 10 Web Application Security Risks, including detailed descriptions and scenarios of how each risk was posed. The report offered a comprehensive look at the web security landscape, demonstrating how the different threats interact with each other and how organizations should prioritize their security efforts to address them. The report also discussed a variety of security controls and best practices for managing web application security risks. In conclusion, the report provides a valuable resource for organizations that wish to understand web application security and implement the appropriate countermeasures. By providing this overview of the OWASP Top 10, the report offers an excellent means for organizations to take a proactive approach to securing their web applications. The report outlined the current threat landscape, highlighted the OWASP Top 10 security risks, and discussed the risk mitigation measures organizations can take to better protect their web applications. Additionally, the report discussed the importance of implementing secure coding practices, educating web application developers about the OWASP Top 10 risks and how to avoid them, and implementing automated security scans that can detect vulnerabilities in web applications.

Wibowo et al. [5] stated that web applications are required to respond to the ease of use of Internet technology. Cross-site scripting (XSS) is one of the most widespread security threats or attacks. Numerous solutions may be utilized to avoid cyber-attacks, such as OWASP Security Shepherd, a secure platform with various tools and techniques, including XSS, to protect web applications from cyber-attacks. Using a combination of secure coding practices, automated tools, and manual code reviews, OWASP Security Shepherd provides an effective solution to protect web applications from XSS attacks. With OWASP Security Shepherd, developers and organizations can mitigate the risks associated with XSS attacks by ensuring that their web applications are properly tested and patched. Additionally, OWASP Security Shepherd provides an intuitive interface and features such as easy-to-use reports, real-time monitoring, and support for multiple programming languages. This makes it possible for developers to quickly identify and address potential vulnerabilities in their web applications, thereby improving the overall security of their online assets.

Muhammet et al. [8] stated that the languages used by web-based systems can cause certain inherent security flaws. Numerous free and paid technologies are available to identify online application security flaws. The purpose of this study was to evaluate and analyze the most popular online vulnerability test programs, and its findings showed that some online vulnerability test programs were more accurate than others. The results indicated that the technologies used to detect security flaws were not always reliable and, in some cases, failed to identify any potential vulnerabilities. Despite their differences in accuracy, all vulnerability testing programs showed potential for discovering security flaws that could expose applications to malicious attacks. Therefore, the results demonstrated the need for organizations to use multiple vulnerability testing programs to identify potential security issues. This underscores the importance of using multiple layers of security measures, since more than a single vulnerability test program may be required to identify all potential security issues. In addition, organizations should ensure that the security measures they use are up to date, since newer technologies may be more accurate in detecting security flaws. Organizations can utilize multiple vulnerability testing programs to ensure that their applications remain secure against malicious attackers.

Wardana et al. [5] declared that vulnerability assessment and penetration testing on published websites following specific standards are critical for information security. Testing was done utilizing the NIST 1100–125–125 Standard through the four primary stages of preparation, discovery, attack, and reporting. One high-level, two medium-level, and four low-level vulnerabilities were identified. During the penetration testing, it was found that while specific security measures had been implemented, including encryption and authentication techniques, they were not utilized to their full potential. Furthermore, other measures could have been implemented to improve the security of the websites, such as using firewalls and intrusion detection systems. These security measures could have provided more robust protection against potential attacks and alerted the organization to any malicious activity occurring on their websites. This assessment and testing showed that while specific security measures were in place, there was still a need to strengthen security by taking advantage of all the available resources. The testing demonstrated the need for improving security measures to prevent potential malicious actors from exploiting vulnerabilities. In conclusion, the penetration testing revealed that while security measures were already in place, they needed to be strengthened and utilized more effectively. To this end, recommendations were put forth to ensure that advantage was taken all the available resources and that the security measures in place were sufficient.

Nagendran et al. [9] described client-side and server-side attacks in classifying web attacks. In addition, they provided an in-depth explanation of how to perform a manual penetration test in web applications to ensure their integrity and security as well as a guide to test the OWASP's top 10 security vulnerabilities. The authors also discussed manual web application penetration testing methodologies, which they classified into five phases: reconnaissance, scanning, exploitation, maintaining access and privilege escalation, and clearing tracks and reporting.

Auricchio et al. [10] proposed a design for an automated web application penetration testing architecture. In addition, they developed an orchestration-based approach to web application penetration testing that they called hacking goals. The researchers identified the generic tasks performed during a penetration test and developed a mechanism for integrating attacks that implement these tasks into a component that executes them. To enable the orchestration of tasks across all phases of a testing campaign, they also defined a communication protocol between the two components. To illustrate how the framework could be applied, the researchers demonstrated how multiple types of attacks could be integrated and an ad-hoc behavioral model that can be embedded to detect cross-site scripting attacks.

Alanda et al. [11] performed penetration testing using the black-box method on web applications based on OWASP's most common attack, namely SQL injection. The researchers randomly attacked several commercial, government, and school websites with various SQL injection techniques. Testing was performed randomly on 10 websites by looking for gaps in security using the SQL injection attacks. According to the testing results, 80% of the websites tested were vulnerable to SQL injection attacks. According to the authors' findings, SQL injection remains the most prevalent threat to web applications. In addition, they provided detailed information about SQL injection and how to prevent it.

Alhassan et al. [12] proposed the fuzzy classifier (FC) vulnerability assessment and penetration testing (VAPT) model using the intelligent learning scheme. This model detects vulnerabilities in web applications and indicates the threat or penetration level for recognized cases. The proposed FCVAPT model was evaluated using XSS and SQL injections. Using the established values of the variables considered, the RAE showed a 14.81% deviation. Its classification performance for MSE, MAPE, and RMSE was 33.33, 14.81%, and 5.77%, respectively. The FCVAPT effectively detected vulnerabilities and identified web application threats/risks.

Hasan et al. [13] conducted a literature survey, provided an overview of VAPT, and identified several limitations. They found it beneficial to use VAPT to secure a web application. They also discussed several tools that can be useful for conducting a VAPT process to detect SQL injection, XSS, local file inclusion, and remote file inclusion vulnerabilities. They found that VAPT helped identify security defects very effectively.

Albahar et al. [14] compared pen-testing tools for detecting vulnerabilities in web applications based on approved standards and methods to facilitate penetration testers' selection of the most appropriate tools. To enhance the effectiveness of web penetration testers and penetration testers in real life, they proposed a benchmarking framework that incorporates the latest research into benchmarking and evaluation criteria in addition to new criteria that provide more coverage with benchmarking metrics. Moreover, a score-based comparative analysis was used to evaluate the tool's abilities. They also carried out simulation tests of commercial and noncommercial pen-testing tools. In their study, Burp Suite Professional was rated the highest out of the commercial tools, while OWASP ZAP was rated the highest out of the non-commercial tools.

Ablahd [15] proposed a system that detects web application vulnerabilities before they can be exploited by attackers. To detect these vulnerabilities, a special scanner was built using Python 3.7's built-in tools, including AST, CFG, Flask, and Django. This proposed system solves two types of risks that can infect a web application due to vulnerability. The proposed scanner detects injection flaws such as command execution and XSS. Fixed-point algorithms are used to determine web application vulnerabilities after analysis and extract their features. A flexible set of tools was designed for the scanner, called SCANSCX, and several vulnerable applications were designed for testing and evaluating SCANSCX's ability. It was a long-term project that spent a great deal of time on analysis and designing an application, and eventually it ended up being terminated.

Pareek [16] explained the types of penetration testing used for web applications, including white box, black box, and gray box penetration testing. In addition, they described the seven phases of penetration testing for web applications. A review of OWASP's top 10 web application security risks was also conducted. Moreover, they presented five tools for web application penetration testing, namely Astra's Pentest, NMAP, Wireshark, Metasploit, and Burp Suite, in terms of their features.

Based on the reviewed studies, Table 3 presents the key findings on the penetration test types and the suggested technique, advantages, and limitations of each study.

Table 3. Summary of related works.

Author	**Publication Year**	**Pen Test Type**	**Suggested Technique**	**Advantages**	**Limitations**
Mirjalili et al. [1]	2014	Automated	Automated penetration testing framework with the following two major components: 1. An operational unit called an executor that conducts attacks; 2. A control unit called an orchestrator that orchestrates attacks across consecutive stages.	The distributed hacking framework provides scalability, a distributed nature, and ease of use and is an invaluable resource for users looking to enhance their cybersecurity.	Suffers from process synchronization, resource management, fault tolerance, and error recovery.
Fredj et al. [7]	2018	Automated	A proactive approach was taken covering the top 10 OWASP projects. A variety of security controls and best practices for managing web application security risks were also provided.	The report outlined the current threat landscape, highlighted the OWASP Top 10 security risks, and discussed risk mitigation measures that organizations can take to better protect their web applications, and it emphasized the need to implement automated security scans that can detect vulnerabilities in web applications.	No limitation was found.

Table 3. *Cont.*

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
Wibowo et al. [5]	2021	Integrated (automated and manual)	An integrated approach for OWASP Security Shepherd based on using a combination of secure coding practices, automated tools, and manual code reviews.	OWASP Security Shepherd provides the following: 1. An effective solution for protecting web applications from XSS attacks; 2. An intuitive interface and features such as easy-to-use reports, real-time monitoring, and support for multiple programming languages. This makes it possible for developers to quickly identify and address potential vulnerabilities in their web applications, thereby improving the overall security of their online assets.	The present web application firewalls only offer basic protection rules that do not consider advancements in the sector. The authors wanted to build and create a lightweight and adaptable web application firewall in the future as part of their ongoing development.
Muhammet et al. [8]	2021	Manual	Manual online vulnerability test programs based on available free and paid technologies for identifying security flaws in online applications are important because the languages used by web-based systems may cause certain inherent to security.	By utilizing multiple vulnerability testing programs, organizations can ensure that their applications remain secure against malicious attackers. In addition, organizations should ensure that the security measures they use are up to date, since newer technologies may be more accurate in detecting security flaws.	The results indicated that the technologies used to detect security flaws were not always reliable and, in some cases, failed to identify any potential vulnerabilities. This underscores the importance of using multiple layers of security measures, since a single vulnerability test program may not be sufficient to identify all potential security issues.
Wardana et al. [5]	2022	Manual	Manual penetration testing on published websites following certain standards with four primary stages: 1. Preparation 2. Discovery 3. Attack 4. Report	To improve the security of the websites, other measures could have been implemented, such as using firewalls and intrusion detection systems. These security measures could have provided more robust protection against potential attacks and alerted the organization to any malicious activity occurring on their websites.	The testing demonstrated the need for an improvement in security measures to prevent potential malicious actors from exploiting vulnerabilities. In conclusion, the penetration testing revealed that while security measures were already in place, they needed to be strengthened and utilized more effectively.

Table 3. *Cont.*

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
Nagendran et al. [9]	2019	Manual	Manual web application penetration testing with the following five phases: 1. Reconnaissance 2. Scanning 3. Exploitation 4. Maintaining access and privilege escalation 5. Clearing tracks and reporting	An in-depth explanation was provided for how to perform a manual penetration test on web applications.	Performing manual penetration tests requires a great deal of expertise in working with HTTP requests and responses.
Auricchio et al. [10]	2022	Automated	Developed an orchestration-based approach to web application penetration testing called hacking goals.	An automated framework was presented for penetration testing. Since the proposed framework is flexible enough to accommodate different attack models, it can be easily customized for different domains.	No limitations were found.
Alanda et al. [11]	2021	Automated and manual	Implemented the black-box method to test web applications for vulnerabilities using OWASP's most common attack, SQL injection.	Various web applications were examined in terms of their penetration methods and SQL injection impact. A detailed explanation of SQL injection, and how to prevent it, was provided.	The conclusion is short and does not include future work.
Alhassan et al. [12]	2018	Automated	Proposed an FCVAPT model using an intelligent learning scheme called a fuzzy classifier.	The proposed model is extremely effective in detecting vulnerabilities and identifying web application threats/risks.	Details were limited about how to implement the proposed model.
Hasan et al. [13]	2018	Automated and manual	Used VAPT to secure a web application.	Security defects can be identified very effectively with VAPT.	The mentioned tools that can be helpful during VAPT processes need to be compared.
Albahar et al. [14]	2022	Automated and manual	To enhance the effectiveness of web penetration testers and penetration testers in real life, a benchmarking framework was proposed incorporating the latest benchmarking and evaluation research.	A comprehensive framework is offered with all the necessary features for pen testers.	Benchmarking should be applied to other tools, and the framework should be extended to include more new metrics.

Table 3. *Cont.*

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
Ablahd [15]	2023	Automated	A system was proposed that detects web application vulnerabilities using Python 3.7 to identify injection flaws such as command execution and cross-site scripting.	The proposed scanner is easy to use (in each web application) and flexible when it comes to updating.	The authors needed to adapt the scanner to detect other types of web vulnerabilities.
Pareek [16]	2019	Automated	Five web application penetration testing tools were presented, namely Astra's Pentest, NMAP, Wireshark, Metasploit, and Burp Suite.	It explains the types, phases, and tools of web penetration testing.	There is not enough discussion about the tools and how to select the optimal one.

4. Web App Vulnerabilities

4.1. The Evolution of Web App Vulnerabilities

FORM components, such as buttons and text fields, are commonly used by web programs to communicate with users. Moreover, the GET or POST variables are important during the communication process. GET and POST variables are part of the hypertext transfer protocol (HTTP), which is responsible for facilitating communication between web programs and users by providing them with certain form components, such as text fields and buttons. Improper handling of data items inside HTTP requests results in the most severe security vulnerabilities in web programs. SSL does not avoid security issues, since it enables secure data transport and does not evaluate HTTP queries. Web apps serve as a gateway to databases that contain crucial application data and assets. Some of the key dangers to the database server layer are SQL injection, illegal server access, and password-cracking attacks. Most SQL injection vulnerabilities are triggered by insufficient input validation. Most online applications store sensitive data in databases or file systems. Developers routinely make errors in the encryption approaches they use to secure this information. Because HTTP is a stateless protocol, web programs utilize different methods to retain the session state. A session is a sequence of interactions between the user and the web app during a single visit to the website. Session management is often accomplished by a unique string, called a session ID, which is sent to the web server with each request. Most web programming languages enable sessions using GET variables and/or cookies. If an attacker can guess or steal a session ID, they can modify the session of another user [7].

4.2. Top 10 Security Threats for the Web Environment

OWASP has produced a renowned top-10 list that includes the most critical security vulnerabilities in web applications and provides ideas about how to cope with such faults. Figure 2 lists the top 10 OWASP vulnerabilities.

In the following, we provide an investigation of the top 10 security threats for web environments. The investigation covers a wide range of topics, such as malicious software, unpatched vulnerabilities, inadequate network and server security practices, insecure user authentication protocols, and other security risks.

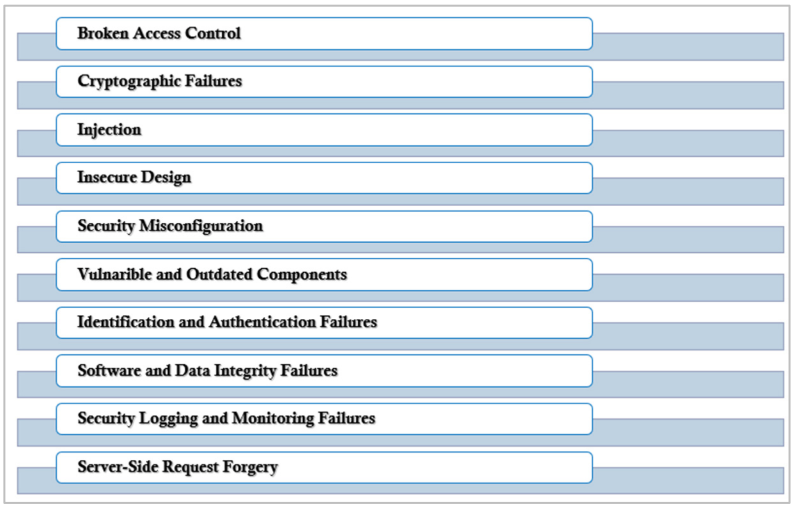
Broken Access Control
Cryptographic Failures
Injection
Insecure Design
Security Misconfiguration
Vulnarible and Outdated Components
Identification and Authentication Failures
Software and Data Integrity Failures
Security Logging and Monitoring Failures
Server-Side Request Forgery

Figure 2. Top 10 Security Threats for Web Applications.

4.2.1. Broken Access Control

Some web applications verify access permissions at the function level before making the functionality available to the user. Nonetheless, once each feature is accessible, programs must pass the same access control check as the server. When requests are not verified, attackers could get access to features without the necessary authorization.

The following are some examples of attacks that can take advantage of the broken access control flaw. In a local file inclusion attack, the attacker attempts to locate a page that accepts as input a path to a file that will be included in the calling page. Moreover, the distant file inclusion attack is similar to the local file inclusion attack, except instead of including files on the same server, the attacker manipulates user input to include remote files [5,8].

4.2.2. Cryptographic Failure

Cryptography refers to the methods and procedures used to maintain secrecy, non-repudiation, integrity, and authenticity. A cryptographic failure is a significant online application security issue that exposes sensitive application data due to a poor or non-existent cryptographic method. These data can include passwords, patient health details, company secrets, credit card information, email addresses, and other sensitive user information. Modern online applications process data both at rest and in transit, necessitating sophisticated security procedures for full threat mitigation.

The following are some examples of attacks that can take advantage of the cryptographic failure flaw. Some deployments use poor cryptographic algorithms that can be cracked in a reasonable amount of time. Cryptographic failures include the transmission of secret material in plain text and the use of an outdated or insecure algorithm, as well as exploitable side-channel information or cryptographic error signals. Inadequate randomness for cryptographic functions and the presence of sensitive data in source control are among the common causes of these failures [17].

4.2.3. Injection

An interpreter may receive or be sent untrusted information from an attacker. The attacker can trick the interpreter and trigger unauthorized instructions by supplying malicious information. The following three types of injection attacks are the most serious: SQL injection, code injection, and XPath injection.

The following are some examples of attacks that can take advantage of the injection flaw. The first category of attack is called an SQL injection attack, and it involves introducing SQL instructions into input forms or queries to access databases or modify their contents, such as by deleting or changing database information. The second category of attack is called code injection, and it involves injecting code that the application understands and runs to take advantage of the sloppy handling of untrusted input. The third category of attack is called XPATH injection, and it occurs when a web application constructs an XPath query for XML data using user input [5,8].

4.2.4. Insecure Design

To avoid security holes, developers are advised to employ safe design patterns, well-designed threat modeling, and reference architecture when creating apps.

The following are some examples of attacks that can take advantage of the insecure design flaw. Insecure design flaws occur when developers as well as quality assurance and/or security teams fail to foresee and analyze dangers during the code design process. These flaws are also the result of failing to follow security best practices while creating an application. Mitigating design vulnerabilities as the threat environment changes necessitates ongoing threat modeling to prevent known attack methods. It is difficult to discover and correct architectural faults such as unprotected credential storage, trust boundary violations, the generation of error messages containing sensitive information, and improper isolation or compartmentalization without a secure design [18].

4.2.5. Security Misconfiguration

A security misconfiguration problem happens when one or more of the components of the system, such as the applications, frameworks, application server, web server, database server, network router, and platform, is not well configured. Secure settings need to be developed, implemented, and maintained.

The following are some examples of attacks that can take advantage of the security misconfiguration flaw. Default settings are frequently the source of such danger [19]. The attacker might use this issue to launch several attacks. The intensity of the attack is determined by the degree and location of the misconfiguration.

4.2.6. Vulnerable and Outdated Components

A software component, such as a module, software package, or API, is part of a system or application that expands the functionality of the program.

The following are some examples of attacks that can take advantage of the vulnerable and outdated components flaw. When a software component is unsupported, out of date, or vulnerable to a known attack, a component-based vulnerability occurs. Inadvertently using insecure software components in production situations might endanger the web application. For example, a company may download and utilize a software component, such as OpenSSL, but neglect to update or fix it when weaknesses are revealed. Because many software components share the same rights as the web application, any vulnerabilities or faults in the component can pose a danger to the application. Using components that have known vulnerabilities exposes an application to attacks that can target any portion of the application stack. For example, the following attack scenarios may target known component vulnerabilities: code injection, buffer overflow command injection, XSS, and vulnerable and obsolete components. In the following scenario, the attacker uses an unpatched system to execute malicious code on the server. The attacker obtains access to the internal network of a company and then uses a scanning tool to identify internal systems with unpatched or obsolete components. The attacker then takes advantage of a defect in the outdated component to install malicious code on the application server [20].

4.2.7. Identification and Authentication Failures

Hackers exploit this vulnerability to take advantage of improper authentication, as its name suggests. A hacker can access user information, passwords, ID sessions, and other login credentials, posing a security risk [21]. The following is an example of attacks that can take advantage of the identification and authentication failure flaw. Credential stuffing is considered a type of broken authentication attack that is driven by brute force.

4.2.8. Software and Data Integrity Failures

Failures of software and data integrity are caused by code and infrastructure that do not protect against integrity violations.

The following are some examples of attacks that can take advantage of the software and data integrity flaws. A good example is when an application depends on plugins, libraries, or modules downloaded from untrustworthy sources, repositories, or content delivery networks. CI/CD pipelines that are insecure can expose systems to unauthorized access, malicious code, and system compromises. Furthermore, many applications now allow auto-updates, whereby updates are downloaded without enough integrity verification and applied to previously trusted applications. Attackers could potentially upload their updates to all installations and distribute them [22].

4.2.9. Security Logging and Monitoring Failures

Without logging, suspicious actions and events can stay unmonitored for longer periods, potentially allowing security breaches to continue undetected for longer than they would with better logging. Website hackers can do a lot of damage, but hacking becoming more difficult if web application owners are not monitoring code behavior for suspicious activity. Monitoring systems can be very useful in this situation. The following is an example of attacks that can take advantage of the security logging and monitoring failures flaw. Cyberattacks can have repercussions that leave one with a limited understanding of what has happened to one's system if one lacks a proper logging and monitoring process [23].

4.2.10. Server-Side Request Forgery (SSRF)

An attacker can exploit this vulnerability to send requests to an unintended location via a server-side application. The following are some examples of attacks that can take advantage of the SSRF flaw. In a typical SSRF attack, an attacker might also use SSRF to connect the server to internal-only services within an organization's infrastructure. It is also possible for them to force the server to connect to arbitrary external systems, exposing credentials and sensitive data [24].

4.3. Web Vulnerabilities Mitigation Techniques

This section aims to present the various security countermeasures that have been proposed to mitigate web vulnerabilities. In addition, it provides an overview of web vulnerabilities and maps suitable detection methodologies for each vulnerability.

4.3.1. Protection against SQL Injection Attacks

Many methods have been developed to deal with SQL injection, the most used attack [5]. Shahriar et al. [25] suggested an information-theoretic paradigm for identifying SQL injection attacks. The proposed framework statically calculates query entropy based on a query's token probability distribution. The entropy of each query in an application is computed twice; that is, once before deployment and again during operation. Moosa et al. [26] presented a system based on semantic comparison. The system is divided into two steps: training and working. During the training phase, the system is given a collection of regular and malicious data to train the artificial neural network (ANN). To identify web application threats, the collected ANN is integrated into the web application firewall. Mamadhan et al. [27] presented a strategy based on semantic comparison. During training and run-time, a semantic comparison is conducted between the two syntax trees of a query. If the two trees are comparable, the query is classified as benign; otherwise, it is classified as malicious. Halfond et al. [28] presented a technique for preventing SQLIAs based on the concept of "positive tainting" an on-the-spot assessment. In situations of incompleteness, it produces more false positives than negatives.

4.3.2. Protection against Broken Authentication and Session Hijacking Attacks

The standard countermeasure strategy for session hijacking is to bind the client's IP address. In more detail, the web server connects a user's session to a specific IP address and subsequently rejects any request arriving from a different IP address. This method necessitates each client to have a unique and unchanging public IP address. However, because a network employs the NAT protocol to share the same IP address with different clients, this strategy is rendered worthless [29]. Another approach to avoiding session hijacking is to track the user's browser fingerprint. A browser fingerprint is made up of several features of the user's browser. Any change to the user's browser fingerprint could indicate an attacker hijacking a session [30]. Session lock incorporates integrity checks into each client request based on a secret shared with the server. A valid request cannot be calculated if a session identification is stolen, since the value of the secret is unknown. Session lock's vulnerability to script-based attacks is one of its limitations [31]. To mitigate session hijacking threats, Ryck et al. [32] advocated replacing the static session identification with disposable tokens for each request, inspired by the idea of Kerberos service tickets [33]. Macaroons [34] target cloud services and control cookie access. They use chains of nested hash-based message authentication codes built from a shared secret and a message chain.

4.3.3. Protection against XSS Attacks

A first line of defense against XSS on the server side is to employ user input validation to enforce security. Validation can employ either blacklisting or whitelisting strategies. Moreover, once user input is confirmed to be harmful, it can either be sanitized or rejected [35]. The secure input processing approach, on the other hand, cannot be used, nor can it attain comprehensive protection, especially for complicated sites. A second defensive line, which is increasingly being deployed in webservers, is based on the content security policy, which typically establishes trustworthy origins so that the browser is allowed to download resources (which may be a script, a stylesheet, an image, etc.) from them. Therefore, although an attacker can inject vulnerable material into the website, the CSP approach may stop its execution. Wurzinger et al. [36] proposed a secure web application proxy for detecting and preventing XSS attacks. The proposed system includes a reverse proxy that intercepts returned HTML messages before utilizing an altered web browser to find vulnerable scripts. Shahriar et al. [37] suggested a proxy-level detection mechanism for XSS attacks based on the Kullback–Leibler divergence (KLD) metric. The approach is predicated on the perception that valid web application JavaScript code should remain equivalent to or extremely close to a displayed web page's JavaScript code. With this in mind, Shahriar et al. [38] continued with the tokenization of the considered script code into unique components and computed the probability of their occurrence to create two sets: P (legal JavaScript code available on the application page) and Q (observed JavaScript code available in the response page). The distance between these two suggested probability distributions is calculated by KLD. When there is a large difference between the two sets, an XSS attack is identified.

4.3.4. Protection against Insecure Direct Object References and Missing Function-Level Access Control

Most security systems employ access control techniques to safeguard access to resources and the usage of internal web application functions. In role-based access control (RBAC) [38], for example, programmers control objects through permissions, assign permissions to roles, and assign roles to users. Permission grants a user access to a role in a particular session. Separation of duty constraints prohibits a user from taking on two or more competing responsibilities. RBAC, for example, is used by the Cisco ACE WA Firewall to specify the administrative responsibilities of the web application firewall (WAF) itself. The authors of [39] presented a secure cookie-based implementation of RBAC with role hierarchies on the web. The user's role information is encoded in a collection of secure cookies and sent to the appropriate web servers. They employ reasonably good privacy to design cookie-verification processes to validate the cookies. The authors of [39] presented an access control mechanism for open web service applications. Their work [40] is built on the eXtensible access control markup language, which is an access control language.

4.3.5. Protection against Sensitive Data Exposure

There are three different categories of sensitive data exposure. The first is information leakage, against which only the developer can improve security by paying attention to what they leave in their code and managing it in a safe way to reduce the faults that might arise. The second category is transmission attacks, which are mostly prevented by a robust encryption method, and we are unaware of any well-known solution employed in WAFs. The third category is database theft, and encryption is a vital solution to deal with this threat. For Access databases using a suitable security strategy, the authors of [41] presented a dynamic database security strategy as a remedy for this type of attack.

4.3.6. Protection against CSRF Attacks

There are several server-side methods to minimize CSRF attacks [5,41,42]. OWASP embarked on a project called CSRFGuard [41], which is a library that implements a modified version of the synchronizer token pattern to reduce the danger of CSRF attacks. The developers of [5] created a server-side proxy named NoForge that can be plugged into the system under consideration to find and avoid CSRF attacks, and it is transparent to users and apps. This proxy's primary function is to identify and defend PHP applications against CSRF attacks. Zeller et al. [43] listed the features of server-side safeguards to protect users. They [41] also created a server-side plug-in to protect users from the attacks.

4.3.7. Protection against Unvalidated Redirects and Forwards

Scott et al. [44] classified phishing countermeasures into four types: blacklist-based, heuristic-based, visual similarity–based, and machine learning–based. The blacklist-based solutions provide a library of found phishing URLs, which should be updated regularly. The most notable solutions in this category are the Google Safe Browsing API [36], PhishNet [45], which predicts phishing URLs based on known ones, and automated individual whitelist [46], which maintains a list of trusted login user interfaces (LUIs). Nevertheless, this list suffers from the problem of untrustworthy LUI prediction; in general, blacklists have high true-positive rates but low false-positive rates. SPHERES [10] is a behavior-based WAF installed on a web application server that prevents phishing attacks by creating a profile for each web client parameter.

5. Web Penetration Test

Web security is an important concern as the Internet expands and web applications are increasingly used in different fields, including the military, health care, and finance. Web security is ensured by penetration tests. Manual or automatic penetration tests can be conducted.

5.1. Web Penetration Testing Tools

This section aims to present attack tools that can be utilized to perform penetration testing based on the type of vulnerability present in the web environment. Moreover, it provides an overview of web penetration testing tools. Table 4 shows the list of web vulnerability and a corresponding tool we can use to detect it.

5.2. Overview of Penetration Testing Tools

Seven commercial and open-source testing tools are covered in this section. These are Netsparker, Acunetix, Vega, OWASP ZAP, Wapiti, IronWASP, and W3af. Each tool has unique features and advantages that can be used to identify a variety of web application security vulnerabilities.

Table 4. Web Vulnerabilities and Attack Tools.

Web Vulnerability	**Attack tool**
Carriage return and line feed (CRLF) injection	RLF-Injection scanner
Components with known vulnerabilities	Vulners API
Cross-origin resource sharing (CORS) policy	CORScanner
Cross-site scripting (XSS)	XSSMap
Injection flaw	Custom
Directory traversal	LFISuite
HTTP response splitting	Custom
HTTP verb tampering	nmap HTTP-methods script
Improper certificate validation MassBleed	MassBleed
Insufficient transport layer protection	Custom
Lightweight directory access protocol (LDAP) injection	Custom
Improper certificate validation	MassBleed
Insufficient transport layer protection	Custom
Lightweight directory access protocol (LDAP) injection	Custom
Operating system (OS) command injection	Commix
Remote file inclusion (RFI)	Fimap
SQL injection SQLmap	XML
External entities (XXE)	Custom

5.2.1. Netsparker

Netsparker is an online security testing tool. It detects and discloses security flaws at the application level of any website. Netsparker comes in two flavors: desktop and cloud. We can scan hundreds of websites or web-based apps at the same time using the cloud version [47]. A desktop version is a convenient tool that can be used on individual websites, while the cloud version enables users to scan multiple websites simultaneously, making it an incredibly powerful tool for website administrators and developers.

5.2.2. Acunetix

Acunetix is an online security testing tool that comprehensively monitors and regulates websites, particularly those dependent on HTML and JavaScript. The software development lifecycle interfaces with project management or bug-tracking systems and contains extensive compliance reports. It runs independently of the operating system by using web browsers [48]. All one needs to do is enter the URL of the target website, and it comes with all the necessary features. Acunetix is the ideal tool for monitoring and regulating websites, especially those that are heavily reliant on HTML and JavaScript.

5.2.3. Vega

Vega is a free and open-source online security testing tool for detecting flaws in web applications, and its graphical user interface (GUI) is built in Java. Vega has two points of view, which are scanner and proxy. For debugging, the Vega interactive web app provides a blocking proxy. The attack modules for Vega are written in JavaScript, and because these are open source, they may be enhanced via a JavaScript API and modified by the user [49]. Vega is a very powerful tool for debugging web applications, since it is capable of identifying security flaws that are hidden from the user. It also offers great flexibility, allowing the user to customize their attack scenarios by adding new attack modules and modifying existing ones.

5.2.4. OWASP ZAP

OWASP is a multinational non-profit organization dedicated to improving software security. ZAP is a simple, open-source integrated penetration testing tool for discovering vulnerabilities in online applications. OWASP openly distributes papers, methodologies, documentation, and tools on the subject of web app security [50]. Utilizing the security tools provided by OWASP, such as ZAP, and following its methodologies for secure coding are essential for organizations when building or maintaining applications. In addition to providing security tools such as ZAP, OWASP also offers educational resources for those involved in the process of building or maintaining software.

5.2.5. Wapiti

Wapiti is a free online security testing tool for detecting flaws in web applications. It runs a black-box test, which means it does not examine the application's source code but instead scans the web pages of the web application being tested and searches for scripts and forms that potentially inject data. Wapiti functions as a fuzzier after it has a list of URLs, forms, and their inputs, injecting payloads to test if a script is susceptible [51]. This can be used to detect common issues, such as SQL injection, XSS, local and remote file inclusion, LDAP injection, and server-side request forgery. Wapiti can also detect different kinds of vulnerabilities in an application, such as weaknesses in authentication systems, improper error handling, and weak encryption functions.

5.2.6. IronWASP

Iron Web Application Advanced Security Testing Platform (IronWASP) is a free and open-source web security testing tool for detecting flaws in web applications. It can identify more than 25 web vulnerabilities. It is a GUI-based utility created in Python and Ruby and can identify false positives and false negatives. IronWASP generates HTML and RTF reports. It can be supplemented using plug-ins or modules written in Python, Ruby, C#, or VB.NET [52]. IronWASP is easy to use and set up and includes a range of tools such as fizzers, proxies, crawlers, traffic analyzers, and even site map generation tools. It is highly versatile, and its modularity makes it an ideal tool for penetration testing, allowing users to combine different features and create powerful solutions tailored to their own needs.

5.2.7. W3af

W3af is a free, open-source tool for automating the scanning of web applications. Both GUI and command line interfaces are available for this tool, which can assess a web application for vulnerabilities and exploit them. There are interconnected plugins that share information between them [10]. This allows W3af to crawl a web application, map its contents, detect known vulnerabilities, and identify problems that may arise from the application's source code. It is important to note that W3af should only be used by experienced professionals, as its powerful capabilities can lead to system damage if used incorrectly. W3af offers users the ability to customize and fine-tune an application according to their specific needs.

In Table 5, we use a set of metrics in terms of the technology being used, the programming language used during tool development, the supported platform on which the tool can be used, the supported interface on which the tool was developed, the online or offline status during tool use, the vulnerabilities detected through this tool, tool usability, and tool cost. These metrics will be useful for the relevant decision-makers during the tool selection process.

Table 5. Penetration Testing Tools.

Tool	Technology	Platform	Interface	Online or offline	Vulnerabilities	Usability	Cost
Netsparker	PHP Java	Web	Command line interface	Online	Identify vulnerabilities such as heartbleed SSL in web applications.	Setup and use are extremely simple.	Request a quote from Sales
Acunetix	Java	Uses web browsers to run independently of the operating system	GUI	Online	More than 4500 vulnerabilities	Easy-to-use and intuitive	Request a quote from Sales
Vega	Java	Linux, OS X, and Windows	GUI	Online	Identify vulnerabilities such as reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file inclusion, shell injection, and more.	Easy to use.	Free
Wapiti	Python	Unix/Linux, FreeBSD Mac OS, OSX, Windows	GUI	Online	More than 23 vulnerabilities	Easy and fast activation and deactivation of attack modules.	Free
OWASP ZAP	Java	Linux, Mac OS, OSX, Windows	GUI	Online	Examines the web application for issues linked to SQL injection. Authentication failure. Exposed sensitive info. Compromised access control. Misconfiguration of security. XSS deserialization is insecure. Components that have known flaws.	Easy to use and report vulnerabilities.	Free
IronWASP	Python and Ruby	Linux, Mac OS, OSX, Windows	Both the GUI and command line interfaces.	Online	More than 25 web vulnerabilities	Beginners may utilize it, since it is extremely simple to use.	Free
W3af	Python	Linux, Mac OS, OSX, Windows	Both the GUI and command line interfaces.	Online	Identify vulnerabilities such as SQL injection, cross-site scripting, guessable credentials, unhandled application problems, and PHP misconfigurations.	Fairly simple to install, and the automatic SVN updates will assist both users and writers in resolving problems rapidly.	Free

6. Discussion and Future Research

The results of one study indicates that not all web penetration testing tools offer the same features and that combining analysis tools may enable detailed information to be provided about web vulnerabilities. In this study, the goal is to guide users in selecting the best web penetration test tool and increase their awareness of secure web environments. In addition, each of these tools has its advantages and disadvantages, so the choice will depend on the needs of the organization or the individual. Our recommendation for choosing a web application penetration testing tool is to look for the following features:

detection and exploitation (the tool should be able to detect as well as exploit vulnerabilities) and reports on results (the tool should generate detailed reports on the results of the tests and scans performed). The tool must also enable testing across multiple operating systems and devices.

We reviewed and compared several tools that are available to perform web penetration testing. However, more tools are available for that purpose that can be reviewed and compared. Furthermore, the installation and use of these tools are not described in detail. As a result, we recommend the following future research directions. First, we recommend the further study of automated techniques to enhance web security and privacy. Second, despite the growing interest in web cybersecurity, little research has been conducted on web penetration testing tools. Our recommendation is for further research on web penetration testing tools to be conducted in the future. Third, our paper identifies the most critical web application security vulnerabilities according to the OWASP top 10 list. Other researchers can explore other types of web threats, such as weak passwords, code injection, cross-site scripting, data breaches, and phishing attacks, and how to mitigate them. Fourth, in this article, we have reviewed the most popular web penetration testing tools and compared them based on several criteria. We recommend that other researchers look for different tools and compare them based on a variety of criteria, such as their advantages and disadvantages. Fifth, other researchers can investigate the installation and experiments of the penetration testing tools in greater detail.

7. Conclusions

The current study analyzed research on the subject of penetration testing, and mainly web penetration testing. Since manual penetration testing is inefficient in terms of time, money, and effort, its automated counterpart was examined. Web scanners are used to execute automated web penetration tests, and testing with automated tools is less time-consuming than testing manually. This paper began by explaining penetration tests and identifying the differences between manual and automated tests. It then reviewed articles about web penetration testing and its associated methods. The most common web application variabilities and techniques to mitigate or prevent attacks were presented, after which, most of the vulnerability types present in the web environment were linked with attack tools that could be utilized to perform penetration testing to detect these vulnerabilities. Furthermore, this paper reviewed and compared some of the available penetration test tools. According to publications that examined numerous scanners, the Netsparker web Vulnerability Scanner, Acunetix, Vega, Wapiti, OWASP ZAP, IronWASP, and W3af were more important than the others. In the last phase, seven test criteria were introduced in terms of technology, platform, interface, online/offline, vulnerability, usability, and cost. One study found that not all web penetration testing tools offered the same features and that combining the tools might provide detailed information about web vulnerabilities. Additionally, all these tools have their own advantages and disadvantages, so the choice depends on the organization's or the individual's needs. Nevertheless, we recommend that they consider features such as vulnerability detection, detailed reports, and compatible operating systems and devices when selecting a tool. The purpose of this paper was to assist web penetration testers in choosing a technology that is optimal for their needs. This study can assist individuals and organizations in determining the best tools for performing web penetration tests. It may be helpful for penetration testers to review the presented results to make better decisions. Furthermore, a clear understanding of the limitations and directions in this area may be helpful for future researchers. Our goal in this paper was not just to advocate for a technical solution but also to describe the challenges faced by web applications regarding security.

8. Managerial Implications

Web applications are becoming more prevalent in corporate, public, and government services today as a result of advances in web technologies and a changing business environment. While web applications can make life easier and more efficient, several security threats could pose significant risks to an organization's IT infrastructure if they are not handled correctly. Since attacks are now specifically targeting security flaws in web application designs, the traditional network security measures and technologies may no longer be adequate for safeguarding web applications from new threats. Along with the development of web applications, new security measures, both technical and administrative, should be implemented. Since the number of web applications continues to increase, it is important to know what threats exist for web applications. An organization can be targeted by a web application threat through its website or applications. During the development process, organizations should address these security concerns by implementing penetration testing to find web vulnerabilities and secure them before real attackers can exploit them. In this paper, we aim to help organizations understand web application threats, mitigate them, and choose the best tool for performing web penetration testing.

9. Practical/Social Implications

Web applications are becoming increasingly vulnerable to cyber-attacks as the internet grows and evolves. The importance of testing one's web application's cyber security has never been greater. This article covered some of the best web application penetration testing tools to ensure that individuals and organizations can test their applications' security and ensure they will withstand any attacks.

Author Contributions: Conceptualization, E.A.A. and A.A.; methodology; E.A.A. and A.A. formal analysis, E.A.A. and A.A.; investigation, E.A.A. and A.A.; resources, E.A.A. and A.A.; writing original draft preparation, E.A.A. and A.A.; writing—review and editing, E.A.A., A.A. and M.F.; supervision, M.F.; funding acquisition, M.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by King Faisal University, Saudi Arabia [Project No. GRANT2,948].

Acknowledgments: This work was supported through the Annual Funding track by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [Project No. GRANT2,948].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mirjalili, M.; Nowroozi, A.; Alidoosti, M. A survey on a web penetration test. *Adv. Comput. Sci. Int. J.* **2014**, *3*, 117–121.
2. Kam, H.J.; Pauli, J.J. Work in progress—web penetration testing: Effectiveness of student learning in Web application security. In Proceedings of the 2011 Frontiers in Education Conference (FIE), Rapid City, SD, USA, 12–15 October 2011; p. F3G-1.
3. Mukhopadhyay, I.; Goswami, S.; Mandal, E. Web penetration testing using nessus and metasploit tool. *IOSR J. Comput. Eng.* **2014**, *16*, 126–129. [CrossRef]
4. Baykara, M. Investigation and comparison of web application vulnerabilities test tools. *Int. J. Comput. Sci. Mob. Comput. (IJCSMC)* **2018**, *7*, 197–212.
5. Wibowo, R.M.; Sulaksono, A. Web vulnerability through cross site scripting (XSS) detection with OWASP security shepherd. *Indones. J. Inf. Syst.* **2021**, *3*, 149–159. [CrossRef]
6. Abu-Dabaseh, F.; Alshammari, E. Automated penetration testing: An overview. In Proceedings of the 4th International Conference on Natural Language Computing, Copenhagen, Denmark, 28–29April 2018; pp. 121–129.
7. Fredj, O.B.; Cheikhrouhou, O.; Krichen, M.; Hamam, H.; Derhab, A. An OWASP top ten driven survey on web application protection methods. In *Risks and Security of Internet and Systems, Proceedings of the 15th International Conference, CRiSIS 2020, Paris, France, 4–6 November 2020*; Springer: Cham, Switzerland, 2021; pp. 235–252.
8. Wardana, W.; Almaarif, A.; Widjajarto, A. Vulnerability assessment and penetration testing on the xyz website using NIST 800-115 standard. *J. Ilm. Indones.* **2022**, *7*, 520–529. [CrossRef]
9. Nagendran, K.; Adithyan, A.; Chethana, R.; Camillus, P.; Varshini, K.B.S. Web application penetration testing. *IJITEE* **2019**, *8*, 1029–1035. [CrossRef]
10. Auricchio, N.; Cappuccio, A.; Caturano, F.; Perrone, G.; Romano, S.P. An automated approach to web offensive security. *Comput. Commun.* **2022**, *195*, 248–261. [CrossRef]
11. Alanda, A.; Satria, D.; Ardhana, M.; Dahlan, A.A.; Mooduto, H.A. Web application penetration testing using SQL Injection attack. *JOIV Int. J. Inform. Vis.* **2021**, *5*, 320–326. [CrossRef]

12. Alhassan, J.K.; Misra, S.; Umar, A.; Maskeliūnas, R.; Damaševičius, R.; Adewumi, A. A fuzzy classifier-based penetration testing for web applications. In *Advances in Intelligent Systems and Computing, Proceedings of the International Conference on Information Technology & Systems (ICITS 2018), Libertad City, Ecuador, 10–12 January 2018*; Springer: Cham, Switzerland, 2018; pp. 95–104.
13. Hasan, A.; Meva, D. Web application safety by penetration testing. *Int. J. Adv. Stud. Sci. Res.* **2018**, *3*, 159–163.
14. Albahar, M.; Alansari, D.; Jurcut, A. An empirical comparison of pen-testing tools for detecting web app vulnerabilities. *Electronics* **2022**, *11*, 2991. [CrossRef]
15. Ablahd, A.Z. Using python to detect web application vulnerability. *resmilitaris* **2023**, *13*, 1045–1058.
16. Pareek, K. A study of web application penetration testing. *IJITE* **2019**, *7*, 1776–2321.
17. Sołtysik-Piorunkiewicz, A.; Krysiak, M. The cyber threats analysis for web applications security in industry 4.0. In *Recent Advances in Computational Optimization*; Springer Science and Business Media LLC: Cham, Switzerland, 2020; pp. 127–141.
18. Alenezi, M.; Agrawal, A.; Kumar, R.; Khan, R.A. Evaluating Performance of web application security through a fuzzy based hybrid multi-criteria decision-making approach: Design tactics perspective. *IEEE Access* **2020**, *8*, 25543–25556. [CrossRef]
19. Qi, L.; Meng, S.; Zhang, X.; Wang, R.; Xu, X.; Zhou, Z.; Dou, W. An exception handling approach for privacy—Preserving service recommendation failure in a cloud environment. *Sensors* **2018**, *18*, 2037. [CrossRef]
20. Gupta, R. An innovative security strategy using reactive web application honeypot. *arXiv* **2020**, arXiv:2115.04773. [CrossRef]
21. Priyawati, D.; Rokhmah, S.; Utomo, I.C. Website vulnerability testing and analysis of website application using OWASP. *Int. J. Comput. Inf. Syst. (IJCIS)* **2022**, *3*, 142–147. [CrossRef]
22. Willberg, M. Web Application Security Testing with Owasp Top 10 Frameworks. Bachelor's Thesis, Turku University of Applied Sciences, Turku, Finland, 2019.
23. Lauinger, T.; Chaabane, A.; Arshad, S.; Robertson, W.; Wilson, C.; Kirda, E. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. *arXiv* **2017**, arXiv:1811.00918. [CrossRef]
24. Shahriar, H.; Zulkernine, M. Information-theoretic detection of SQL injection attacks. In Proceedings of the 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, Omaha, NE, USA, 25–27 October 2012; pp. 40–47.
25. Mnif, A.; Cheikhrouhou, O.; Ben Jemaa, M. An ID-based user authentication scheme for wireless sensor networks using ECC. In Proceedings of the ICM 2011 Proceeding, Hammamet, Tunisia, 19–22 December 2011; pp. 1–9. [CrossRef]
26. Moosa, A. Artificial neural network-based web application firewall for SQL injection. *Int. J. Comput. Inf. Eng.* **2011**, *4*, 611–6120.
27. Mamadhan, S.; Manesh, T.; Paul, V. SQLStor: Blockage of stored procedure SQL injection attack using dynamic query structure validation. In Proceedings of the 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA), Kochi, India, 27–29 November 2012; pp. 240–245. [CrossRef]
28. Halfond, W.; Orso, A.; Manolios, P. WASP: Protecting web applications using positive tainting and syntax-aware evaluation. *IEEE Trans. Softw. Eng.* **2008**, *34*, 65–81. [CrossRef]
29. De Ryck, P.; Desmet, L.; Piessens, F.; Johns, M. *Primer on Client-Side Web Security*; Springer: Cham, Switzerland, 2014. [CrossRef]
30. Nikiforakis, N.; Kapravelos, A.; Joosen, W.; Kruegel, C.; Piessens, F.; Vigna, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In Proceedings of the 2013 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 19–22 May 2013; pp. 541–555. [CrossRef]
31. Adida, B. Sessionlock: Securing web sessions against eavesdropping. In Proceedings of the 17th International Conference on World Wide Web, Beijing China, 21–25 April 2008; pp. 517–524.
32. Dacosta, I.; Chakradeo, S.; Ahamad, M.; Traynor, P. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Trans. Internet Technol. (TOIT)* **2012**, *12*, 1–24. [CrossRef]
33. Johns, M.; Braun, B.; Schrank, M.; Posegga, J. Reliable protection against session fixation attacks. In Proceedings of the SAC 2012 ACM Symposium on Applied Computing, Trento, Italy, 26–30 March 2012; pp. 1531–1537.
34. Kallin, J.; Valbuena, I.L. Excess XSS: A Comprehensive Tutorial on Cross-Site Scripting. 2016. Available online: https://excess-xss.com (accessed on 1 February 2023).
35. Wurzinger, P.; Platzer, C.; Ludl, C.; Kirda, E.; Kruegel, C. SWAP: Mitigating XSS attacks using a reverse proxy. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, Vancouver, BC, Canada, 19 May 2009; pp. 33–311.
36. Shahriar, H.; North, S.; Chen, W.C.; Mawangi, E. Design and development of Anti-XSS proxy. In Proceedings of the 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), London, UK, 9–12 December 2013; pp. 4114–41111.
37. Ferraiolo, D.; Cugini, J.; Kuhn, D.R. Role-based access control (RBAC): Features and motivations. In Proceedings of the 11th Annual Computer Security Application Conference, New Orleans, LA, USA, 11–15 December 1995; pp. 241–248.
38. Park, J.S.; Sandhu, R.; Ghanta, S. RBAC on the Web by Secure Cookies. In *Research Advances in Database and Information Systems Security*; Springer: Boston, MA, USA; pp. 411–462.
39. Ardagna, C.A.; Vimercati, S.D.C.D.; Paraboschi, S.; Pedrini, E.; Samarati, P.; Verdicchio, M. Expressive and deployable access control in open web service applications. *IEEE Trans. Serv. Comput.* **2010**, *4*, 96–109. [CrossRef]
40. Doshi, J.; Bhushan, T. Sensitive data exposure prevention using dynamic database security policy. *Int. J. Comput. Appl.* **2014**, *106*, 18600–19869.
41. Kiernan, J.; Agrawal, R.; Haas, P.J. Watermarking relational data: Framework, algorithms and analysis. *VLDB J.* **2003**, *12*, 157–169. [CrossRef]
42. Pasha Deshmukh, A.; Qureshi, R. Transparent data encryption–Solution for security of database contents. *arXiv* **2013**, arXiv:1303.

43. Jovanovic, N.; Kirda, E.; Kruegel, C. Preventing cross site request forgery attacks. In Proceedings of the 2006 Securecomm and Workshops, Baltimore, MD, USA, 28 August 2006–1 September 2006; pp. 1–10. [CrossRef]
44. Scott, D.; Sharp, R. Specifying and enforcing application-level web security policies. *IEEE Trans. Knowl. Data Eng.* **2003**, *15*, 771–783. [CrossRef]
45. Google. Overview. Safe Browsing Apis (V4). Available online: https://developers.google.com/safe-browsing/v4/ (accessed on 20 December 2022).
46. Cao, Y.; Han, W.; Le, Y. Anti-phishing based on automated individual white-list. In Proceedings of the 4th ACM Workshop on Digital Identity Management, Alexandria, VA, USA, 31 October 2008; pp. 51–60.
47. Joshi, C.; Singh, U.K. Performance evaluation of web application security scanners for more effective defense. *Int. J. Sci. Res. Publ. (IJSRP)* **2016**, *6*, 660–667.
48. Elisa, N. Usability, accessibility and web security assessment of e-government websites in tanzania. *Int. J. Comput. Appl.* **2017**, *164*, 42–48. [CrossRef]
49. Tundis, A.; Mazurczyk, W.; Mühlhäuser, M. A review of network vulnerabilities scanning tools: Types, capabilities, and functions. In Proceedings of the 13th international Conference On Availability, Reliability, and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–10.
50. Bennetts, S. *Owasp Zed Attack Proxy*; AppSec USA: San Francisco, CA, USA, 2013.
51. Alsaleh, M.; Alomar, N.; Alshreef, M.; Alarifi, A.; Al-Salman, A. Performance-Based comparative assessment of open source web vulnerability scanners. *Secur. Commun. Netw.* **2017**, *2017*, 1–14. [CrossRef]
52. Amankwah, R.; Chen, J.; Kudjo, P.K.; Towey, D. An empirical comparison of commercial and open—Source web vulnerability scanners. *Softw. Pr. Exp.* **2020**, *50*, 1842–1857. [CrossRef]