Web-Sentry Toolkit : An OWASP Top 40 MITM Attack Model for Vulnerability Assessment & Penetration Testing of Web Application Projects

(A Subset Tool Framework for Project Grading System & Other Implications)

Aryan Parashar
Email: aryan25ic011@satiengg.in
Computer Science Engineering Department - CSE with Internet of Things, Cyber Security with Blockchain Technology (ICB)
Samrat Ashok Technological Institute, Vidisha (M.P.) - 464001

ABSTRACT

In the era of digital transformation, the proliferation of web applications has become ubiquitous, serving as critical components of modern businesses and user interactions. These applications, encompassing server-based programs, facilitate a multitude of functionalities, making web server security a paramount concern for organizations connected to the internet. Ensuring a secure online environment is crucial not only for safeguarding sensitive customer data but also for maintaining the integrity and trustworthiness of an organization's online presence. As the demand for intuitive and visually appealing web platforms continues to grow, so does the imperative for robust web security measures. The escalating number of internet users and the corresponding rise in web applications and user data have concurrently fueled an increase in cyber threats. Organizations worldwide are facing the formidable challenge of protecting their data from sophisticated cyber-attacks, which can severely damage their reputation and erode customer trust.

This paper introduces a novel web-based Open Web Application Security Project(OWASP) Man-In-The-Middle (MITM) Attack Model designed specifically for VAPT of web applications. Inspired by OWASP Zed Attack Proxy (ZAP), Web Sentry Toolkit aims to revolutionize the process of vulnerability assessment and penetration testing by leveraging the convenience and accessibility of web-based platforms.

The Web Sentry Toolkit addresses these challenges by providing a comprehensive framework for Man-In-The-Middle (MITM) attack modeling, specifically tailored for web application penetration testing. This toolkit covers the OWASP Top 40 vulnerabilities, offering a detailed examination of common security flaws and the prerequisites for conducting thorough security assessments. By leveraging this toolkit, penetration testers and security analysts can perform rigorous vulnerability assessments and penetration testing (VAPT), ensuring that all potential threats are identified and mitigated. Regular security testing and the application of timely security patches are critical in maintaining the security posture of web applications. The Web Sentry Toolkit not only identifies vulnerabilities but also provides actionable insights and guidelines on the do’s and don’ts of security assessments, aligned with each identified vulnerability. This proactive approach

to web security helps organizations prepare for and defend against potential cyber threats, ultimately safeguarding their data and preserving their reputation in an increasingly digital world.

Traditionally, conducting VAPT required users to download and install specialized tools onto their local machines, which could be time-consuming and cumbersome, especially for users with limited technical expertise. The integration of WebSentry to 'Submify' Intelligent Project Grading System shall ensure that toolkit remains aligned with industry best practices for VAPT. By offering a user-friendly web interface and an extensive range of vulnerability assessments, our project contributes to the development of a more accessible and efficient Automatic WebDev Project Grading System, also facilitating enhanced security analysis for all web applications.

In summary, this paper introduces the Web Sentry Toolkit as an essential resource for enhancing web application security through effective MITM attack modeling and comprehensive penetration testing, ensuring organizations can protect their digital assets.

Keywords

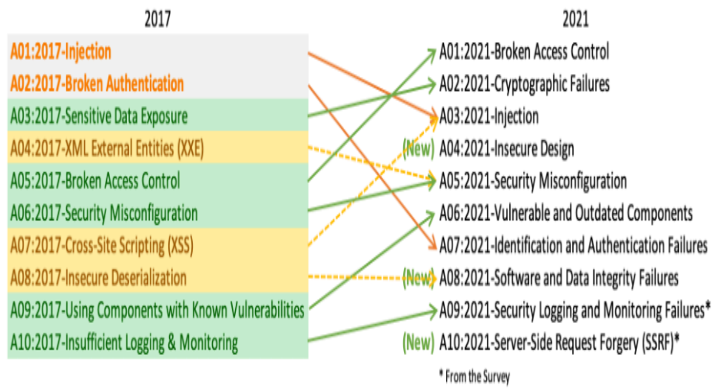
Open Worldwide Application Security Project(OWASP) , Vulnerabilities, False Positives, Injections, Penetration Testing, Threat Modules.

I. INTRODUCTION

In today's digital age, where over 2.14 billion people worldwide purchase goods and services online [1], ensuring the security of online transactions is paramount. Data breaches can have devastating consequences for both businesses and individuals, leading to financial losses, reputational damage, and even identity theft. Study by IBM found that the average cost of a data breach in 2023 was a staggering $4.35 million [2]. Just like any physical security system, online platforms can have weaknesses or loopholes known as vulnerabilities. Hackers exploit these vulnerabilities to gain unauthorized access to sensitive information. E-commerce websites, which have been around since the early days of the internet (Web 2.0), are a prime target for such attacks. With millions of e-commerce platforms operating worldwide, a Forgenix survey found that a concerning 75% are at risk of some form of cyberattack. Even well-established brands are not immune, as evidenced by the millions of dollars lost due to cyberattacks in recent years. A Verizon Data Breach Investigations Report found that 80% of hacking-related breaches involved web applications, highlighting the critical need for robust online security.

One of the most infamous data breaches in recent history involved credit reporting agency Equifax. In 2017, hackers exploited a vulnerability in Equifax's website, compromising the personal information of nearly 150 million Americans, including Social Security numbers, birth dates, and home addresses [3]. This breach exposed the severe consequences of neglecting cybersecurity measures and resulted in Equifax facing hefty fines and a significant loss of consumer trust.

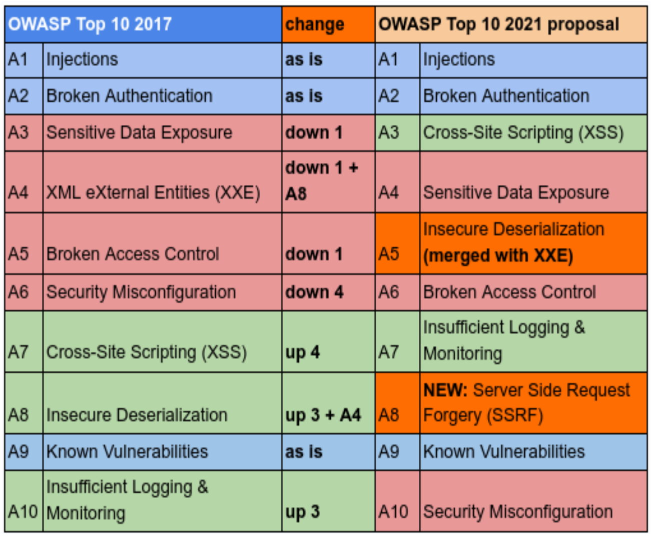
The best defense against cyberattacks is a proactive approach. Regular vulnerability assessments and penetration testing (pentesting) can identify weaknesses in a system before they can be exploited. The OWASP (Open Web Application Security Project) Top 10 serves as a foundational reference document for developers and web security professionals, outlining the most pressing security risks faced by web applications. These vulnerabilities, which are enumerated in the Top 10, encompass a range of critical issues.


2017
2021
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

For instance, Broken Access Control, The Top 1 OWASP Vulnerability, 2021 refers to the improper implementation of access controls, which can permit users to operate beyond their intended permissions. 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category. Cryptographic Failures entail inadequate protection of sensitive data during storage or transmission. Injection occurs when untrusted data is incorporated into an interpreter as part of a command or query. Insecure Design pertains to flaws in the software's design that compromise security. Security Misconfiguration involves insecure default configurations, incomplete configurations, or open cloud storage. Vulnerable and Outdated Components encompass the usage of components known to have vulnerabilities. Identification and Authentication Failures relate to weaknesses in authentication mechanisms. Software and Data Integrity Failures occur when the integrity of software and data is compromised. Security Logging and Monitoring Failures refer to inadequate logging and monitoring practices. Server-Side Request Forgery (SSRF) transpires when the server is deceived into making requests to unintended locations.

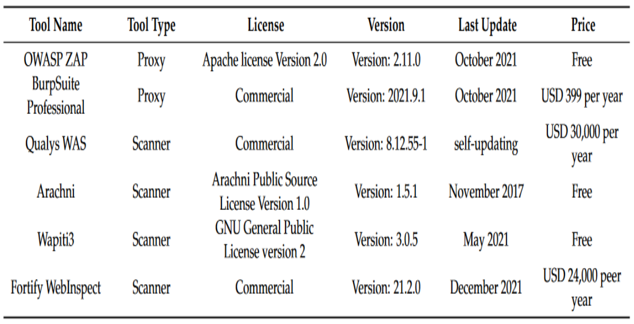
Addressing these vulnerabilities necessitates the adoption of robust mitigation strategies, including the implementation of strong access controls, encryption of sensitive data, rigorous input validation, secure design principles, regular updates and patching, strong authentication mechanisms, and comprehensive logging and monitoring systems.

Web Sentry takes a proactive approach to detecting Broken Access Controls, utilizing advanced payloads and techniques. This tool systematically assesses for improper access controls by attempting to access restricted resources with varied permission levels, employing a combination of automated scripts and payloads designed to simulate common attack vectors. For instance, the Broken Access Controls script within the Web Sentry Toolkit showcases its capability to detect and report unauthorized access attempts through payload injection, response analysis, and logging mechanisms. This systematic approach ensures the effective identification and mitigation of broken access controls, thereby bolstering the security posture of web applications. By offering comprehensive solutions for addressing these vulnerabilities, Web Sentry plays a pivotal role in safeguarding web applications against the Top 40 most significant threats identified by OWASP.



OWASP Top 10 2017		change	OWASP Top 10 2021 proposal	
A1	Injections	as is	A1	Injections
A2	Broken Authentication	as is	A2	Broken Authentication
A3	Sensitive Data Exposure	down 1	A3	Cross-Site Scripting (XSS)
A4	XML eXternal Entities (XXE)	down 1 + A8	A4	Sensitive Data Exposure
A5	Broken Access Control	down 1	A5	Insecure Deserialization (merged with XXE)
A6	Security Misconfiguration	down 4	A6	Broken Access Control
A7	Cross-Site Scripting (XSS)	up 4	A7	Insufficient Logging & Monitoring
A8	Insecure Deserialization	up 3 + A4	A8	NEW: Server Side Request Forgery (SSRF)
A9	Known Vulnerabilities	as is	A9	Known Vulnerabilities
A10	Insufficient Logging & Monitoring	up 3	A10	Security Misconfiguration

II. Analysis of Current Scenario of Commercial Web Vulnerability Scanners & Their Limitations



Tool Name	Tool Type	License	Version	Last Update	Price
OWASP ZAP	Proxy	Apache license Version 2.0	Version: 2.11.0	October 2021	Free
BurpSuite Professional	Proxy	Commercial	Version: 2021.9.1	October 2021	USD 399 per year
Qualys WAS	Scanner	Commercial	Version: 8.12.55-1	self-updating	USD 30,000 per year
Arachni	Scanner	Arachni Public Source License Version 1.0	Version: 1.5.1	November 2017	Free
Wapiti3	Scanner	GNU General Public License version 2	Version: 3.0.5	May 2021	Free
Fortify WebInspect	Scanner	Commercial	Version: 21.2.0	December 2021	USD 24,000 peer year

The current landscape of web vulnerability assessment and penetration testing (VAPT) tools is fraught with several significant limitations. Based on the analysis of the Papers "An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities" by Marwan Albahar, Dhoha Alansari and Anca Jurcut & "Evaluation of

Web Vulnerability Scanners Based on OWASP Benchmark" by Balume Mburano and Weisheng Si the primary limitations can be categorized into false positives, detection scope, performance issues, and user expertise requirements.

False Positives & False Negatives: A false positive occurs when a security scanner reports a vulnerability that doesn't actually exist, vice-versa for a false negative . This can lead to wasted effort as testers have to manually verify these non-issues. False positives can consume a significant amount of time for pen-testers, making the process inefficient and costly. They can also lead to real vulnerabilities being overlooked if testers become desensitized to warnings due to the high volume of false alerts . Following the False positive and True Negative formulae:

$$FPR = [FP \div (FP + TN)] \times 100 \quad (1)$$

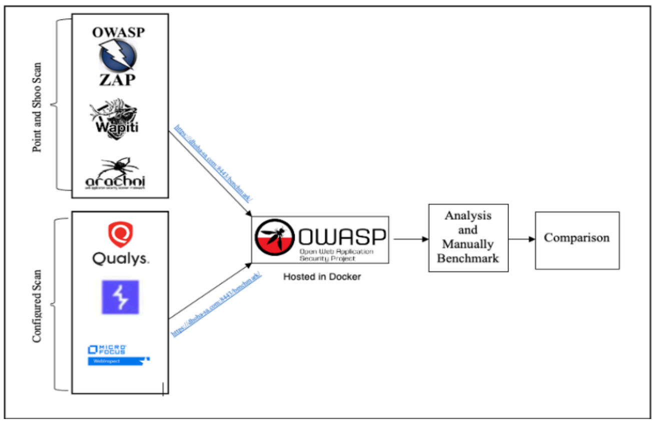
$$TPR = [TP \div (TP + FN)] \times 100 \quad (2)$$

Youden's Index(J), proposed for evaluating the effectiveness of analytical (diagnostic) tests, is a metric that yields either 1 or -1. A score of 1 indicates that the scanner accurately detects vulnerabilities without any false positives, while a score of -1 suggests that only false positives are detected, with no actual vulnerabilities identified. A score of 0 signifies that the tool produces results consistent with the expected outcome from the web application (FP, TP). The formula for calculating the Youden index, as outlined in reference aligns with the described scoring.

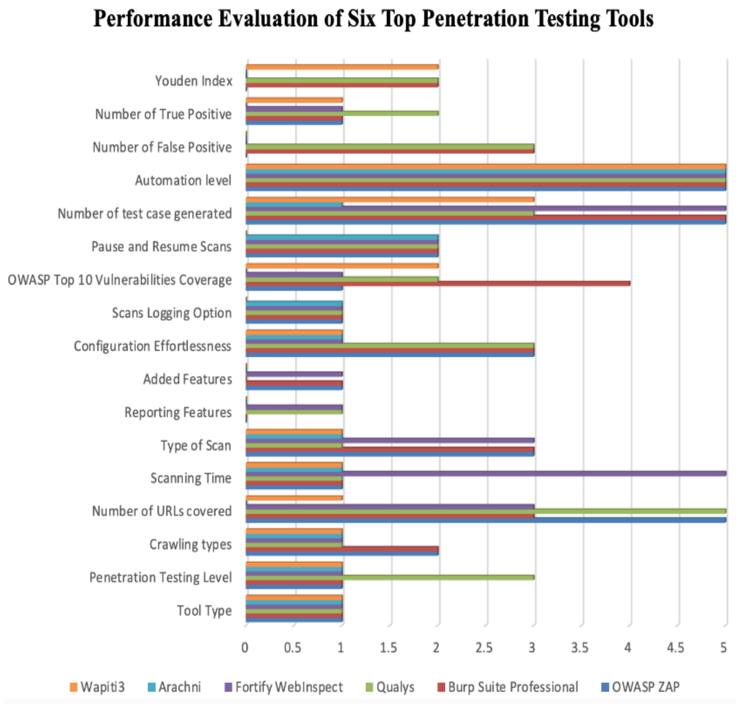
Youden's Index Formula:

$$J = \frac{TP}{TP + TN} + \frac{TN}{TN + FP} - 1 \quad (3)$$

Lower Vulnerability Detection Scope & Techniques: Most tools are designed for specific scenarios and may not cover all types of vulnerabilities comprehensively. This can result in undetected vulnerabilities if the wrong tool is used for a particular application. Different tools perform variably across different types of vulnerabilities. For example, some may excel at detecting SQL injections (SQLi) but fail to detect cross-site scripting (XSS) issues effectively. The "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark" highlights significant variability in the detection rates of different tools. For example, while SQL injection vulnerabilities were detected at a rate of 90%, cross-site scripting (XSS) vulnerabilities were only detected at a rate of 60%. This inconsistency indicates that no single tool provides comprehensive coverage across all vulnerability types, necessitating the use of multiple tools or supplementary manual testing. Prominent Commercial Web VAPT Applications such as OWASP Zed Access Proxy (ZAP) tends to cover just for the OWASP Top 10 Vulnerabilities.


Point and Shoo Scan
OWASP ZAP
Wapiti
arachni
Configured Scan
Qualys.
MICRO FOCUS WebInspect
https://dhoha-sa.com:8443/benchmark/
https://dhoha-sa.com:8443/benchmark/
OWASP Open Web Application Security Project
Hosted in Docker
Analysis and Manually Benchmark
Comparison

Performance Issues: Performance issues such as scan speed and resource usage are discussed in "An Empirical Comparison of Commercial and Open-Source Tools." The paper notes that some vulnerability scans can take over 2 hours to complete, which is impractical for frequent testing scenarios. Furthermore, the resource demands of these tools can be prohibitive, particularly for smaller organizations with limited computational resources. The same paper reports that 70% of users felt that a high level of expertise was necessary to effectively configure and interpret the results from these tools. This high barrier to entry can limit the accessibility and effectiveness of these tools, especially for organizations that do not have dedicated, highly skilled security personnel. For an all over-view of the existing tools we may consider an example, Burp Suite Professional and Qualys WAS excel in vulnerability detection despite potential delays in completing tasks. Conversely, Fortify WebInspect, an automated tool, may not detect vulnerabilities within a short scan duration; however, its manual attack simulation feature proves beneficial for assessing known vulnerabilities manually. Additionally, OWASP ZAP and Burp Suite Professional demonstrate powerful crawling capabilities. Future efforts will involve expanding the framework to incorporate more metrics and applying the benchmarking approach to new tools. The OWASP Top 10 Benchmark Project has the potential to extend to other benchmarks and real-world vulnerable environments, provided it yields comprehensive results aiding in the selection of most suitable tool for specific tasks.



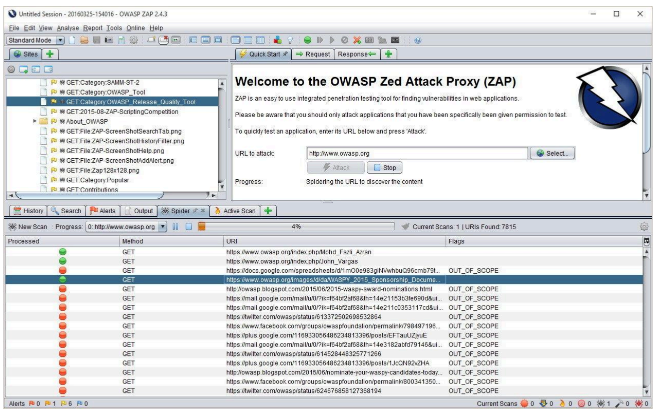
Addressing the identified limitations underscores the critical necessity for the development of Web Sentry, a specialized tool poised to tackle these challenges head-on. This innovative solution promises several enhancements crucial for effective web security testing. Web Sentry endeavors to elevate accuracy levels by leveraging advanced algorithms and machine learning methodologies, thereby significantly curbing false positives. This improvement ensures that penetration testers can devote their attention exclusively to authentic vulnerabilities, optimizing the testing workflow. The tool aims to broaden its detection scope, encompassing a comprehensive array of vulnerabilities by integrating multiple detection techniques and maintaining an updated vulnerability database. This proactive approach ensures thorough coverage across various vulnerability types, enhancing overall security posture. Furthermore, Web Sentry prioritizes performance improvements by implementing optimized scanning algorithms, thereby expediting scan speeds without compromising accuracy. Additionally, its resource-efficient design ensures accessibility for organizations with limited computational resources, fostering widespread adoption. The user experience is also a focal point of Web Sentry's development, with an emphasis on a user-friendly interface aimed at reducing the skill barrier for effective utilization. By offering intuitive navigation and automated reporting functionalities, the tool empowers users with clear and actionable insights, diminishing reliance on manual result interpretation. Through these advancements, Web Sentry emerges as a pivotal asset in fortifying web application security, poised to redefine the landscape of vulnerability assessment and mitigation.

III. Literature Reviews

Upon studying the pennings of Mandar Prashant Shah, titled “Comparative Analysis of Web Pen-testing Tool”, and of Raghu Vamsi Potukuchi, titled “A review of Web Application

Vulnerability Assessment and Penetration Testing" , and finally of Sulaiman Alazami & Daniel Conte De Leon, titled "A systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners", one can derive basic conclusion based on the field of Vulnerability Assessment and Penetration Testing (VAPT) being fundamental to cybersecurity strategies. Vulnerability assessment involving systematically scanning applications for known vulnerabilities, while penetration testing exploits these vulnerabilities to gauge their potential impact and develop effective remediation strategies. Various tools and methodologies exist for VAPT, each with specific strengths and weaknesses. The key studies evaluate web vulnerability scanners based on metrics such as detection capabilities, false positive rates, user-friendliness, and compliance with security standards like the OWASP Top 10 . The effectiveness of these tools is crucial for ensuring thorough and accurate security assessments. By comparing different scanners, the study highlights their respective strengths and limitations, which can inform the development of more robust security tools.

Penetration testing tools are vital for simulating attacks and identifying security weaknesses in web applications. The evaluation of these tools involves assessing their detection capabilities, false positive rates, ease of configuration, and compliance with security standards. Commonly used tools include OWASP ZAP, Burp Suite Professional, Qualys WAS, Arachni, Wapiti, and Fortify WebInspect.



GET:Category:SAMM-ST-2
GET:Category:OWASP_Tool
GET:Category:OWASP_Release_Quality_Tool
GET:2015-08-ZAP-ScriptingCompetition
About_OWASP
GET:File:ZAP-ScreenShotSearchTab.png
GET:File:ZAP-ScreenShotHistoryFilter.png
GET:File:ZAP-ScreenShotHelp.png
GET:File:ZAP-ScreenShotAddAlert.png
GET:File:Zap128x128.png
GET:Category:Popular
GET:Contributions

Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://www.owasp.org

Progress: Spidering the URL to discover the content

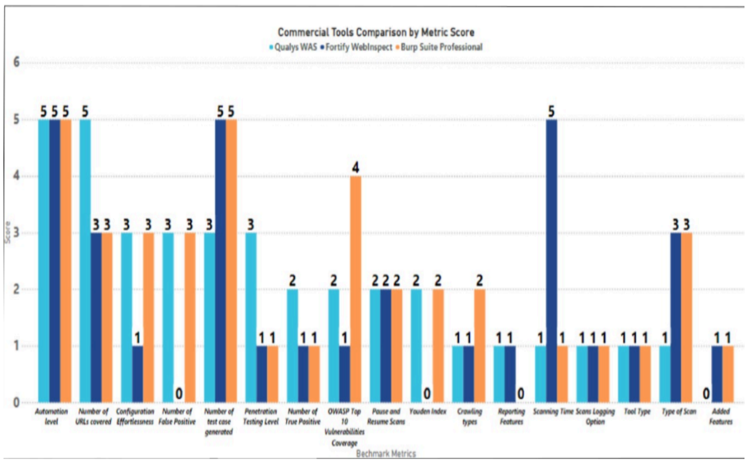
```
Progress: 0: http://www.owasp.org   4%   Current Scans: 1 | URIs Found: 7815

Processed  Method  URI                                                                         Flags
    ●      GET     https://www.owasp.org/index.php/Mohd_Fazli_Azran
    ●      GET     https://www.owasp.org/index.php/John_Vargas
    ●      GET     https://docs.google.com/spreadsheets/d/1mO0e983giNVwhbuQ96cmb79t...          OUT_OF_SCOPE
    ●      GET     https://www.owasp.org/images/d/da/WASPY_2015_Sponsorship_Docume...
    ●      GET     http://owasp.blogspot.com/2015/06/2015-waspy-award-nominations.html         OUT_OF_SCOPE
    ●      GET     https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e21153b3fe690d&ui...   OUT_OF_SCOPE
    ●      GET     https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e211c0353117cd&ui...   OUT_OF_SCOPE
    ●      GET     https://twitter.com/owasp/status/613372502698532864                         OUT_OF_SCOPE
    ●      GET     https://www.facebook.com/groups/owaspfoundation/permalink/798497196...      OUT_OF_SCOPE
    ●      GET     https://plus.google.com/116933056486234813396/posts/EFTauUZjyuE             OUT_OF_SCOPE
    ●      GET     https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e3182abfd79146&ui...   OUT_OF_SCOPE
    ●      GET     https://twitter.com/owasp/status/614528448325771266                         OUT_OF_SCOPE
    ●      GET     https://plus.google.com/116933056486234813396/posts/1JcQN92vZHA             OUT_OF_SCOPE
    ●      GET     http://owasp.blogspot.com/2015/06/nominate-your-waspy-candidates-today...   OUT_OF_SCOPE
    ●      GET     https://www.facebook.com/groups/owaspfoundation/permalink/800341350...      OUT_OF_SCOPE
    ●      GET     https://twitter.com/owasp/status/624676858127368194                         OUT_OF_SCOPE

Alerts 0 1 6 0                                                     Current Scans 0 0 0 0 1 0 0
```

OWASP ZAP is noted for its strong detection capabilities and extensive support for various vulnerability types. Burp Suite Professional excels in both vulnerability detection and exploitation, with comprehensive reporting features that enhance its utility. Qualys WAS offers robust scanning capabilities with a focus on ease of use and integration, making it suitable for a range of environments. Arachni demonstrates high performance in detecting vulnerabilities, especially in modern web applications, while Wapiti provides a good balance of detection capabilities and user-friendliness, particularly suited for smaller applications. Fortify WebInspect stands out with its strong automated testing features, making it ideal for enterprise environments. .

Criteria	Metric	Score Range
Test coverage	Test Coverage	1–5
	pen-testing Level	1–3
	Number of URLs covered	1–5
Attack coverage	Number of test case generated	1–5
Efficiency	Scanning Time	1–5
Vulnerability detection	OWASP Top 10 Vulnerabilities Coverage	1–5
	Number of False Positive	1–3
	Number of True Positive	1–4
	Youden Index	1–3
	Automation level	1–5
Other New	Crawling types	1–2
	Added features	0–1
	Reporting Features	0–1
	configuration Effortlessness	1–3
	Scans Logging Option	0–1
	Tool Cost	NA
	Tool Type	NA
	Scan Type	1–3
	Pause and Resume Scans	0–2

The experimental setup for evaluating web application security tools used by Marwan Albaha in his article titled "An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities" involves several key steps. Initially, the necessary hardware and software environments are configured for each tool. This setup is followed by benchmarking, using the OWASP Benchmark Project to evaluate the tools' effectiveness in detecting vulnerabilities. Subsequent testing involves conducting thorough assessments on various web applications to gauge the tools' performance. Finally, the results are analyzed and compared to identify the strengths and weaknesses of each tool.

Statistical analysis of the results often involves calculating detection rates, false positive rates, and the precision and recall of each tool. Precision is the ratio of true positive results to the total positive results (both true and false), while recall is the ratio of true positive results to the total actual positives. High precision and recall indicate a tool's effectiveness in accurately identifying vulnerabilities without generating many false positives. Such metrics provide a quantitative basis for comparing different tools and determining their relative strengths in various testing scenarios .



The evaluation of the selected penetration testing tools revealed significant insights. OWASP ZAP demonstrated robust detection capabilities, identifying a wide range of vulnerabilities with high accuracy. Burp Suite Professional also performed exceptionally well, particularly in its ability to exploit detected vulnerabilities, offering detailed reports that aid in remediation efforts. Qualys WAS was praised for its user-friendly interface and strong integration features, making it a valuable tool for continuous security monitoring. Arachni showed superior performance in detecting vulnerabilities in modern web applications, benefiting from its high-speed scanning capabilities. Wapiti offered a good balance of effectiveness and ease of use, making it suitable for smaller-scale applications. Fortify WebInspect stood out with its comprehensive automated testing features, making it ideal for large enterprises requiring extensive security assessments .

Statistical analysis of the results underscored the importance of using a combination of tools to achieve comprehensive security coverage. For instance, combining tools like OWASP ZAP and Burp Suite Professional can provide extensive detection and exploitation capabilities, while integrating Qualys WAS or Fortify WebInspect can enhance automation and reporting features. This multi-tool approach leverages the strengths of each tool, addressing their individual weaknesses and providing a more thorough security assessment.

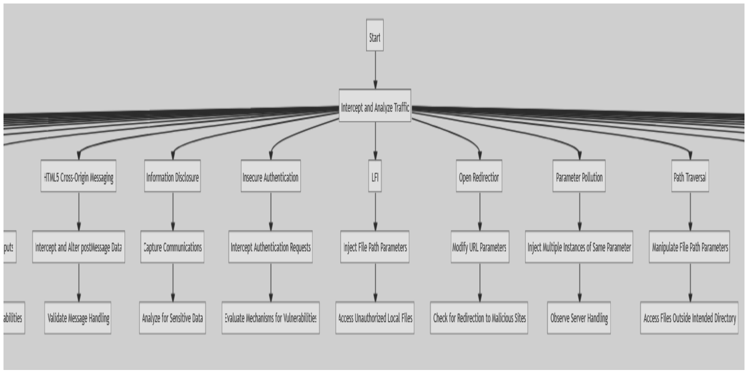
The development of a Web Sentry Toolkit incorporating MITM attack models for VAPT is crucial for enhancing web application security. By leveraging the strengths of various penetration testing tools and adhering to the OWASP Top 40 guidelines, the toolkit can provide a robust framework for identifying and mitigating vulnerabilities. Future work should focus on integrating advanced features such as machine learning for automated vulnerability detection and enhancing the user interface for improved usability.

The insights drawn from this literature review, based on extensive evaluation and comparison of existing tools, highlight the need for continuous improvement and adaptation in the face of evolving cyber threats. By incorporating the latest techniques and methodologies, the Web Sentry Toolkit significantly contributes to the security of web applications, safeguarding sensitive data and maintaining operational integrity .

IV. Formulating the Web Sentinel Tool-kit as 'WebSentry' for Effective Automation of Web VAPT and Documentation for CySA

The formulation for the Web Sentinel Toolkit initiated from studying and analyzing possible Zero Day Vulnerabilities and common mis-patching configuration updates often missed by organizations, this further developed into an analysis of trends seen in the OWASP Listing of

top vulnerabilities and further segregation of these vulnerabilities into the respective OSI Layers which are majorly affected by them. Web Sentry Toolkit began with searching for Injection Payloads and devising mechanism to automate injection of these Payloads into the Input Fields of a permission Web Page for conduction of the Vulnerability Assessment. Upon reviews and feedbacks provided by the Project guide and mentor, further provisioning was done into labeling and identifying each payload for a particular vulnerability, separating the successfully injected payloads and doing a vast study to analyze the Vulnerability Metric each payload holds in-terms of the injecting character's commonness and ubiquity.
Sensitive Data Leakage checking mechanism involved incorporation of Open Source Intelligence (OSINT) methods and Google Dorking Payloads from the Google Hacking Databases. By Intercepting requests to identify unauthorized access to resources. WebSentry attempts to modify parameters to gain access to restricted pages or functions under the Broken Access Controls analysis mechanism. For Brute Force Password guessing, WebSentry captures login requests and uses automated scripts to repeatedly attempt different username/password combinations from a defined list. WebSentry alters file download responses by embedding payloads like `=cmd|'  /C calc'!A0` to execute commands upon opening the CSV file.


Start
Intercept and Analyze Traffic
HTML5 Cross-Origin Messaging
Information Disclosure
Insecure Authentication
LFI
Open Redirection
Parameter Pollution
Path Traversal
puts
Intercept and Alter postMessage Data
Capture Communications
Intercept Authentication Requests
Inject File Path Parameters
Modify URL Parameters
Inject Multiple Instances of Same Parameter
Manipulate File Path Parameters
abilities
Validate Message Handling
Analyze for Sensitive Data
Evaluate Mechanisms for Vulnerabilities
Access Unauthorized Local Files
Check for Redirection to Malicious Sites
Observe Server Handling
Access Files Outside Intended Directory

For Cache Poisoning, it modifies cached responses to inject malicious data into subsequent requests. Client-side Template Injection is analyzed by injecting payloads such as `{{7*7}}` into user inputs, testing for unintended template processing. WebSentry identifies Content Management System Disclosure by examining HTTP headers and response content for CMS-specific details and vulnerabilities. The Content Security Policy (CSP) Check involves analyzing HTTP headers to ensure CSP directives are correctly implemented, thus preventing XSS and other attacks.

For Cross-Origin Resource Sharing (CORS), it manipulates `Origin` headers in requests to check for improper configurations that could leak data across domains. Cross Site Scripting (XSS) vulnerabilities are tested by injecting scripts (e.g., `<script>alert(1)</script>`) into inputs and observing for execution. WebSentry detects DOM Clobbering by inserting elements that might interfere with the DOM structure, identifying security weaknesses. Directory Listing is tested by accessing directories via URL manipulation to see if listings are exposed. HTML Injection involves inserting HTML tags into inputs to check for rendering vulnerabilities.

For HTML5 Cross-Origin Messaging, WebSentry intercepts and alters `postMessage` data to validate the handling of messages from different origins. Information Disclosure is tackled by capturing communications and analyzing them for sensitive data, such as error messages and internal paths. Insecure Authentication is probed by intercepting authentication requests and evaluating mechanisms for vulnerabilities like plain-text passwords or predictable tokens. Local File Inclusion (LFI) is tested by injecting file path parameters to access unauthorized local files.

Open Redirection is analyzed by modifying URL parameters to redirect users to malicious sites and checking if the redirection occurs. Parameter Pollution is tested by injecting multiple instances of the same parameter to see how the server handles them and if it results in unexpected behavior. Path Traversal vulnerabilities are probed by manipulating file path parameters (e.g., `../../../etc/passwd`) to access files outside the intended directory. Sensitive Data Exposure is identified by capturing and analyzing responses for sensitive information that should not be transmitted. Server-Side Request Forgery (SSRF) is tested by manipulating URL parameters to make the server request unintended resources, potentially leading to data leaks. Server-Side Template Injection is analyzed by injecting template expressions (e.g., `${7*7}`) into inputs to test for server-side execution.

Session Fixation involves capturing session identifiers and attempting to set them for another user, aiming to hijack the session. Session

Hijacking is performed by intercepting and stealing active session cookies to gain unauthorized access. Spider Link Parameter Input is analyzed using automated tools to crawl and evaluate all links and parameters for vulnerabilities. Unprotected API Endpoints are tested by intercepting API requests and checking for missing authentication or authorization checks. XML External Entity (XXE) vulnerabilities are tested by injecting XML payloads containing external entity references to see if the server parses them insecurely. For HTTP Security Headers Analysis, WebSentry captures HTTP headers and assesses them for security-related configurations like HSTS, X-Content-Type-Options, and X-Frame-Options. Custom Header Vulnerability is analyzed by injecting and evaluating custom headers to see if the application processes them insecurely. At the Network Layer, Open Ports Scanning is conducted using tools like nmap to identify open ports and analyze running services. Finally, Subdomain Analysis and Enumeration use DNS enumeration tools to discover subdomains and assess potentially vulnerable services hosted on them.

These could be elaborated by Listing the OWASP Top 40 Vulnerabilities into segregated chunks upon the basis of Logical Assessment Sequences.

1. Preliminary Network Scanning (Layer 3)

1.1 Open Ports Scanning: Identify open ports and running services to understand the network's attack surface.

Python script employs the `nmap` library to scan a specified target host for open ports and categorize their vulnerability levels into "High", "Medium", or "Low" based on predefined lists of high and medium vulnerability ports. The `classify_vulnerability` function assesses the vulnerability of each port, while the `scan_ports` function conducts the port scan and presents detailed scan results including host state, protocol, port number, port state, and vulnerability level. The `main` function serves as the entry point, extracting the target host from a given URL and initiating the port scan process. Upon execution, the script provides insights into the security posture of the target host by analyzing its open ports for potential vulnerabilities.

1.2 Subdomain Analysis and Enumeration: Discover subdomains to identify additional targets within the application's domain.

The script defines three asynchronous functions: `sublist3r_enum`, `crt_sh_enum`, and `dns_enum`, each responsible for discovering subdomains through different methods—Sublist3r, crt.sh, and DNS resolution, respectively. These functions utilize asyncio and aiohttp for asynchronous HTTP requests and dns.asyncresolver for DNS resolution. Within each function, discovered subdomains are printed to the console with color formatting to distinguish between the sources. The `main` function initializes the subdomain enumeration process by specifying a target domain, then sequentially invokes the asynchronous enumeration functions. Upon execution, the script outputs the discovered subdomains using the specified techniques for further analysis.

2. Initial Information Gathering and Configuration Checks

2.1 HTTP Security Headers Analysis: Check for security-related HTTP headers like HSTS, X-Content-Type-Options, and X-Frame-Options.

This Python script conducts HTTP security header analysis for a specified website URL. The `analyze_security_headers(url)` function retrieves the headers of the given URL using the `requests` library and compares them against a dictionary of recommended security headers and values. It checks for missing or misconfigured security headers, printing any discrepancies detected along with recommendations for remediation. The `main()` function prompts the user for a target website URL, ensures it is prefixed with 'http://' or 'https://', and then invokes the `analyze_security_headers` function. Upon execution, the script provides insights into the security posture of the analyzed website regarding HTTP security headers, highlighting any potential vulnerabilities or misconfigurations.

2.2 Content Security Policy (CSP) Check: Analyze CSP headers to ensure policies are correctly implemented to prevent client-side attacks.

This Python script is a Content Security Policy (CSP) Bypass Tester that checks for potential bypasses in websites' CSP configurations using various payloads. It begins by importing the `requests` library for HTTP requests and the `BeautifulSoup` class from the `bs4` module for HTML parsing. The script defines a list of payloads representing different attack vectors aimed at bypassing CSP restrictions. Each payload is then assigned to a global variable. The `test_csp_bypass` function tests a given URL against a specified payload to determine if the CSP can be bypassed. It sends a GET request to the target URL, analyzes the CSP header in the response, and checks for the presence of `'unsafe-inline'` directives. If found, it extracts and inspects inline scripts in the HTML content for potential injection points. If a successful injection is detected, it returns `True` along with the vulnerability metric. The `run_csp_bypass_test` function orchestrates the testing process by iterating over the payloads and invoking the `test_csp_bypass` function. If a successful injection is found, it prints a final report with details of the successful payload and vulnerability metric. Finally, the script is executed, and the CSP bypass test is performed on a predefined target URL.

2.3 Content Management System Disclosure: Identify the CMS and associated vulnerabilities through HTTP headers and responses.

Examining websites for potential disclosure of CMS-related information. It imports the `requests` library for making HTTP requests and the `BeautifulSoup` class from the `bs4` module for HTML parsing. The `check_cms_info_disclosure` function is responsible for fetching the website content, parsing it, and identifying potential CMS disclosure indicators, such as the presence of the meta generator tag. If such indicators are found, it suggests URLs to test for vulnerabilities and defines payloads to check against. It then iterates over the payloads, attempting to inject them into the target URL and evaluating the response. If a successful injection is detected, it returns details of the injected payload along with the vulnerability metric. The `run_cms_info_disclosure_test` function orchestrates the testing process, and the script is executed to perform the CMS information disclosure vulnerability test on a predefined target URL.

2.4 Information Disclosure: Look for sensitive information in communications, error messages, and response data.

The `check_information_disclosure` function takes a target URL as input and sends a GET request to fetch the website content. It then searches for sensitive keywords within the response body, HTML comments, and meta tags. If any potential sensitive information is found, it prints out details along with the context. The function also handles various exceptions such as timeouts and other errors, providing recommendations and a vulnerability metric based on the severity of the issue.

3. Access and Authentication Testing

3.1 Insecure Authentication: Intercept authentication requests and analyze for weak mechanisms, such as plain-text passwords or predictable tokens.

`test_authentication_vulnerability` function takes a target URL as input and sends a GET request without authentication. It then checks the status code of the response. If the status code is 401 (Unauthorized), it indicates a high vulnerability of insecure authentication, allowing unauthorized access to restricted pages or endpoints. If the status code is 403 (Forbidden), it indicates a medium vulnerability of weak authentication or insufficient access controls. If neither status code is encountered, it signifies a low vulnerability with no insecure authentication found. The function handles exceptions and provides recommendations based on the vulnerability severity.

3.2 Brute Force: Capture login requests and use automated scripts to attempt multiple username and password combinations.

Performs a brute force attack on a specified target URL by systematically trying a list of usernames and passwords. It generates random usernames and passwords using specified character sets and includes default lists of common usernames and passwords. During the attack, the script sends login attempts to the target URL and analyzes the server's responses. If protection mechanisms like CAPTCHA or two-factor authentication are detected, the script provides suggestions for manual intervention. By examining the responses, it attempts to identify successful logins or different protection mechanisms that might block automated access attempts. The payloads analyzed in the responses include indicators of protection mechanisms (e.g., CAPTCHA, two-factor authentication) and login success or failure messages.

3.3Session Fixation: Capture session identifiers and attempt to set them for another user to hijack the session.

Code tests for session fixation vulnerabilities on a specified target URL by analyzing the 'Set-Cookie' header in the HTTP response. It starts by initiating a session and sending a GET request to the target URL. The response's status code and 'Set-Cookie' header are extracted and displayed. The presence of a session ID in the 'Set-Cookie' header is then checked; if found, it warns of a potential session fixation vulnerability, assigning a high vulnerability level. If no session ID is found, it indicates a low vulnerability level.

3.4 Session Hijacking: Intercept and steal active session cookies to gain unauthorized access.

It defines a dictionary of payloads crafted to exploit such vulnerabilities, associating each payload with a vulnerability metric. `check_session_hijacking_vulnerability` function sends an HTTP GET request to a target URL, extracts session-related information from the HTML content using BeautifulSoup, and tests each payload for vulnerability by injecting it into the URL and analyzing the response. Output is formatted using `colorama` to distinguish between informational messages, successful payload injections, and errors.

3.5 Broken Access Controls: Intercept requests to identify unauthorized access to resources and attempt to modify parameters to gain access to restricted pages or functions.

The Python script imports the `subprocess` module to execute security tests on a specified URL. It defines a list of security test commands, each corresponding to a specific vulnerability, along with a dictionary mapping payloads to their respective vulnerability description. `Perform_security_tests` function iterates through the security commands, executing each one using `subprocess.run()`, and checks the return code to determine if the command executed successfully. If successful, it prints the payload and its associated vulnerability. Finally, it prints the successful payload along with its vulnerability metric. The script demonstrates a systematic approach to automating security testing for web applications, aiding in the identification of potential vulnerabilities.

4. Client-Side and Injection Vulnerabilities

4.1 Cross Site Scripting (XSS): Inject common XSS payloads (e.g., `<script>alert(1)</script>`) into user inputs and monitor responses for execution.

The script utilizes the `requests`, `BeautifulSoup`, and `colorama` modules to detect cross-site scripting (XSS) vulnerabilities on a target URL. First, it fetches all input fields from the URL using `find_input_fields`. Then, it tests each input field with predefined XSS payloads using `test_xss_vulnerability`. If a vulnerability is detected, it prints the vulnerable URL, payload, and input field. If no successful injection is found with existing payloads, it generates new payloads and tests them. If vulnerabilities are found, it prints the injected payloads and the input fields where they were detected. If no input fields are found on the webpage, it exits with a message indicating the absence of input fields.

4.2 HTML Injection: Inject HTML tags into user inputs to check if they are rendered, indicating vulnerability to HTML injection.

Defines a list of common HTML injection payloads. `analyze_injection_vulnerability` function takes the payloads as input and iterates through each payload to test for vulnerabilities. For each payload, it sends a GET request to the target URL, injects the payload into the HTML content, and checks if the payload is reflected in the response. If a vulnerability is detected, it prints a message indicating the detection and suggests a mitigation mechanism. Finally, it returns a dictionary containing the vulnerability metrics for each payload. The script demonstrates an automated approach to identify and assess HTML injection vulnerabilities in web applications.

4.3 Client-side Template Injection: Inject payloads like `{{7*7}}` into user inputs to test for template injection vulnerabilities.

Defines a function `check_template_injection_vulnerability` to send a GET request to the target URL with a payload and checks if the payload is reflected in the HTML response. It uses BeautifulSoup to parse the HTML content and looks for indicators of CSTI vulnerability. Another function, `guide_csti_testing`, provides guidance on testing for CSTI vulnerabilities, such as injecting template code and observing the results. The main function, `run_csti_vulnerability_test`, prompts the user for the target website URL, then iterates through a list of predefined payloads to test for vulnerabilities using the `check_template_injection_vulnerability` function. If a vulnerability is detected, it prints a message indicating the detection and suggests a high vulnerability metric.

4.4 DOM Clobbering: Inject elements with names or IDs that could interfere with the DOM structure to identify security weaknesses.

defines a function `test_dom_clobbering` to test for DOM Clobbering vulnerability. This function follows several steps: first, it fetches the HTML content of the webpage using the `requests` library. Then, it extracts all script tags from the HTML content using `BeautifulSoup`. Next, it defines various payloads known to exploit DOM Clobbering vulnerabilities. After that, it checks each payload for injection success by modifying the HTML content and searching for the payload. Finally, it displays the vulnerability status and testing instructions based on the injection success. The main function `main` prompts the user for the target website URL, and calls the `test_dom_clobbering` function if the URL is valid

4.5 CSV Injection: Intercept file download responses and inject payloads like `=cmd|' /C calc'!A0` into CSV fields to test for code execution upon opening the file.

Defines a function `generate_payloads()` to generate various CSV injection payloads. These payloads are designed to execute arbitrary commands or scripts when the CSV file is opened by an application that interprets CSV files.

Following that, the script defines a function `check_csv_injection_vulnerability(url)` to check for CSV Injection vulnerability on a specified URL. It constructs the URL for the vulnerable endpoint by appending `vulnerable_endpoint.csv` to the base URL. Then, it generates payloads using the `generate_payloads()` function and iterates through each payload. For each payload, it creates a malicious CSV file containing the payload and uploads it to the target URL using a POST request. If the response indicates a successful upload and the content type is `text/csv`, it checks if the payload was executed successfully. Based on the payload content and execution result, it assesses the vulnerability metric. If a vulnerable payload is found, it prints the payload and its vulnerability metric.

4.6 Server-Side Template Injection: Inject template expressions (e.g., `${7*7}`) into user inputs to test for execution on the server side.

Defines a function `test_ssti(url, payload)` to test for SSTI vulnerability on a specified URL with a given payload. It crafts the URL with the payload, sends a GET request, and checks if the response contains the expected output indicating SSTI. Based on the response status code and content, it prints whether a potential SSTI vulnerability is found.

4.7 Extensible Stylesheet Language Transformations - XSLT Injection Vulnerability: Leveraging XSLT injection to manipulate XML data transformations and potentially execute arbitrary code and injecting malicious XSLT code into an XML file.

The `xslt_vuln_test(url)` function is then defined to perform the vulnerability test. It defines HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) and XSLT payloads for injection. Each payload is analyzed to assign a risk metric based on its content. The function iterates over HTTP methods and payloads, sends requests with each payload, and checks the response for sensitive information leakage. If sensitive information is detected, it prints the vulnerability details and stops further testing.

4.8 LaTex Injection Vulnerability: Exploiting LaTeX input fields to inject and execute arbitrary LaTeX code. Injecting `\input|"calc.exe"` into a LaTeX input field.

The `test_latex_injection(url)` function is defined to perform the vulnerability test. It contains a list of payloads designed to retrieve sensitive information from the server. Each payload is sent as data in a POST request to the target URL. The function checks the response for indications of sensitive information leakage, such as the presence of system files or system-specific data.

5. Parameter Manipulation and Resource Access

5.1 Parameter Pollution: Inject multiple instances of the same parameter to test how the server handles them and if it leads to unexpected behavior.

`extract_parameters_from_url(url)` function sends a GET request to the target

URL and extracts query parameters from the URL's query string. It returns a dictionary of query parameters. `test_parameter_pollution_vulnerability(url, parameters)` function tests for parameter pollution vulnerabilities by modifying each parameter with a dummy value and sending a request with the modified parameters. If the response status code indicates a potential vulnerability, it prints a message indicating the parameter that might be affected.

5.2 Path Traversal: Manipulate file path parameters to access files outside the intended directory (e.g., `../../../etc/passwd`).

`test_path_traversal_vulnerability(target_url)` function sends a GET request to the target URL and extracts links from the HTML content using BeautifulSoup. It then checks each link for suspicious patterns indicative of Path Traversal vulnerabilities. If it finds any vulnerable links, it prints them out.

5.3 Local File Inclusion (LFI): Inject file path parameters to attempt accessing unauthorized local files on the server.

`test_local_file_inclusion(url)` function sends a GET request to the target URL and checks if the response contains indicators of LFI vulnerabilities, such as the presence of sensitive files like `/etc/passwd` or `root:`. It then prints out the vulnerability severity based on the findings.
`test_lfi_with_payloads(url)` function tests for LFI vulnerabilities with additional payloads. It iterates over a list of predefined LFI payloads, attempts to inject them into the URL, and checks if the response indicates successful file inclusion. It prints out the vulnerability severity for each payload and details of any successful injection.

5.6 XML External Entity (XXE): Inject XML payloads containing external entity references to test for vulnerabilities in XML parsing.

It defines multiple XML payloads, each containing different external entities that may be exploited to retrieve sensitive information or perform other malicious actions. These payloads include references to local files (`file:///`) like `/etc/passwd` and `/etc/hosts`, as well as external resources (`http://attacker.com/`). The `test_xxe_vulnerability` function sends these payloads as POST requests to the target URL and checks the response for indications of successful exploitation, such as the presence of sensitive data like 'root:'. If a successful injection is detected, the script identifies the payload and vulnerability metric as 'high'.

5.7 Server-Side Request Forgery (SSRF): Manipulate URL parameters to make the server request resources from unintended locations, potentially leading to data leaks.

It defines a function `test_srf` that crafts a forged request with specified parameters like method, URL, headers, and data, and then sends this request using the `requests` library. The function checks if the response status code indicates a successful redirection (3xx). If so, it returns a vulnerability level of "high"; otherwise, it returns "low". If an error occurs during the request (e.g., network issues), it returns a vulnerability level of "medium".

5.8 SQL Injection Attack: Exploiting SQL injection vulnerabilities to execute malicious SQL queries and gain unauthorized access to databases. Injecting `' OR 1=1 --` into a login form

It first defines a function `find_input_fields(url)` to retrieve input field names from the provided URL using BeautifulSoup. Then, it defines function `test_sql_injection(url, field_name, payload, payload_name)` to test for SQL injection vulnerabilities by sending POST requests with various payloads to the input fields identified earlier. If the response contains any SQL-related keywords or errors, it prints a message indicating the detection of a vulnerability.

6. Resource and Redirection Checks

6.1 Open Redirection: Modify URL parameters to redirect to malicious sites and analyze if the redirection occurs.

`check_redirection_vulnerability(target_url, redirect_url)` function sends a GET request to the target URL without allowing redirection. It then analyzes the response to determine if there's a vulnerability. If the status code indicates a redirection (300-399), it checks if the redirection URL matches the specified redirect URL. Depending on this comparison, it categorizes the vulnerability level as "high" (if the redirection URL matches the specified one), "medium" (if redirection occurs but not to the specified URL), or "low" (if no redirection occurs or the HTTP request fails).

6.2 Cross-Origin Resource Sharing (CORS): Modify `Origin` headers in requests to test for inappropriate CORS configurations that allow data leaks across domains.

`test_cors_vulnerability(target_url)` function initiates the CORS vulnerability test. It sends a preflight CORS request (OPTIONS) to the target URL and checks if the 'Access-Control-Allow-Origin' header is present in the response. If this header is found, indicating a potential CORS misconfiguration, it proceeds to test further for misconfigurations using additional payloads.

`test_additional_cors_misconfigurations(target_url, headers, idx)` function is executed concurrently to test each payload for potential bypasses. It sends a GET request to the target URL with the specified headers and checks if the bypass attempt was successful. Depending on the success of the bypass attempt and the content of the 'Origin' header in the payload, it assigns a vulnerability metric ('high', 'medium', or 'low').

6.3 HTML5 Cross-Origin Messaging: Intercept and modify `postMessage` data to test for improper validation of messages from different origins.

`test_html5_cors(target_url)` function initiates the HTML5 CORS vulnerability test. It sends a GET request to the target URL and parses the HTML content using BeautifulSoup. If iframe tags with the 'sandbox' attribute are found, it indicates potential vulnerability to HTML5 Cross-Origin Messaging. It provides instructions on how to test this vulnerability and suggests mitigations.

6.4 Unprotected API Endpoints: Intercept API requests and analyze for lack of authentication or authorization checks.

`test_api_vulnerability(url)` function tests for unprotected API endpoints on the target URL. It fetches the webpage content, extracts potential API endpoints, and attempts to bypass protection systems and exploit vulnerabilities.

The `exploit_vulnerability(endpoint)` function attempts to exploit identified vulnerabilities using predefined payloads. It iterates over the payloads, sends requests with modified parameters or headers, and checks for successful exploitation.

The script utilizes multithreading to scan the target website efficiently. It prints the vulnerable APIs found during the scanning process.

6.5 Sensitive Data Exposure and Google Dorking Queries : Capture and analyze responses for sensitive data, like PII or financial information, that should not be transmitted.

`fetch_url(url)` function attempts to fetch the content of a given URL using the `requests` library. It returns the response object if successful or `None` if an error occurs.

The `test_sensitive_data_exposure(url)` function is the main logic of the script. It utilizes multithreading with a `ThreadPoolExecutor` to concurrently fetch the content of the main URL, the login page (`/login`), and the admin page (`/admin`). It then checks each response for common sensitive data keywords, printing any potential matches found.

First it defines functions to execute dork queries (`execute_dork_queries`) and assign vulnerability metrics (`assign_vulnerability_metric`). In the `main()` function, it constructs URLs based on a domain and a list of queries, then assigns vulnerability metrics to each URL and stores the results. It then attempts to inject payloads into these URLs, printing out successful injections along with associated metrics and variable names.

7. Infrastructure and Caching

7.1 Directory Listing: Access directories directly via URL manipulation to check if directory listings are enabled.

`detect_directory_vulnerability` function analyzes the provided URL for directory listing vulnerability by sending a request, parsing the HTML content, and identifying directories in the response. If directories are found, it prints them and returns a vulnerability level (high, medium, or low) based on the severity. The `attempt_bypass_protection` function tries various techniques to bypass protection mechanisms on the URL, printing successful attempts and returning a vulnerability level accordingly.

7.2 Cache Poisoning: Modify cached content in the intercepted responses to serve malicious data on subsequent requests.

`perform_cache_poisoning_analysis` function conducts cache poisoning analysis on a specified URL by iterating through a list of predefined payloads, sending HTTP GET requests with these payloads, and analyzing the responses. It tracks the first successful response as a baseline and compares subsequent responses to detect successful poisonings, storing details of these occurrences. The severity of cache poisoning is assessed based on cache-control directives in the payloads. Upon successful poisonings, the function prints details of each successful response, including payload used, response time, and degree of vulnerability.

7.3 Spider Link Parameter Input: Use automated tools to crawl and analyze all links and parameters for potential vulnerabilities.

Defines a function named `spider` to crawl a website specified by a given URL. It sends an HTTP GET request to the URL and, upon a successful response, parses the HTML content using BeautifulSoup. It then extracts various elements such as links, forms, inputs, scripts, images, and file paths from the HTML content. Each type of extracted element is printed with relevant details.

8. Final Checks and Custom Vulnerabilities

8.1 Custom Header Vulnerability: Inject and analyze custom headers to see if the application processes them in insecure ways.

`test_custom_header_vulnerability` function tests for vulnerabilities by sending a GET request to a specified `target_url` with custom `headers`. Upon receiving the response, it checks for the presence of response headers; if found, it indicates vulnerability to custom header vulnerabilities and suggests exploiting it with a custom header. Otherwise, it concludes that the website is not vulnerable.

V. Technical Bucket Used

The Python programs provided leverage a diverse range of libraries and modules to execute various security testing tasks efficiently. The following is an overview of the libraries utilized and their specific roles:

1. requests: Widely used for sending HTTP requests to target URLs, facilitating web scraping, API interaction, and vulnerability testing.
2. concurrent.futures: Particularly the ThreadPoolExecutor class, employed for asynchronous task management and parallel execution, enhancing performance in multiple vulnerability testing scenarios.
3. urllib.parse: Utilized for constructing URLs by combining base URLs with specific paths, ensuring proper URL formatting in sensitive data exposure testing.
4. pyfiglet: Employed across all programs to generate visually appealing ASCII art banners, enhancing program identification and aesthetics.
5. BeautifulSoup: Applied for HTML parsing and analysis, aiding in identifying vulnerabilities like exposed API endpoints or insecure iframe tags in unprotected API endpoints and HTML5 CORS testing.
6. logging: Used for structured logging in the unprotected API endpoints testing program, enabling tracking of test progress and results for debugging and monitoring purposes.
7. re: Utilized for regular expression matching in the unprotected API endpoints testing program, facilitating the extraction of potential API endpoints from HTML content.
8. webbrowser: Executed Google Dork queries by opening search URLs in the default web browser, enhancing the Google Dork query targeting program's functionality.

Example: In CORS vulnerability testing, sensitive data exposure testing, and unprotected API endpoints testing programs, requests is employed to send HTTP GET requests to target URLs, while concurrent.futures.ThreadPoolExecutor manages parallel execution for improved efficiency. Additionally, BeautifulSoup parses HTML content to identify specific tags like iframe, and webbrowser executes Google Dork queries by opening search URLs, collectively enhancing the comprehensiveness and automation of vulnerability assessments.

VI. Use Cases and Benefits

- **Comprehensive Vulnerability Assessment:**
  - WebSentry meticulously examines web applications for vulnerabilities, including SQL injection and Cross-Site Scripting (XSS), ensuring proactive identification and resolution of potential weaknesses.
  -
- **Regulatory Compliance:**
  - By conducting regular VAPT assessments, WebSentry assists organizations in meeting regulatory requirements such as GDPR, HIPAA, and PCI DSS, thus avoiding potential fines and penalties.
- **Risk Management:**
  - WebSentry enables organizations to prioritize vulnerabilities based on severity, effectively allocating resources to mitigate significant security risks.
- **Proactive Threat Intelligence:**
  - Continuously scanning for vulnerabilities, WebSentry provides organizations with proactive threat intelligence, enabling them to anticipate emerging threats and lessen the likelihood of successful cyber attacks.
- **Incident Response Preparedness:**
  - Organizations conducting routine VAPT assessments with WebSentry are better prepared to respond to security incidents by understanding their vulnerabilities and crafting tailored incident response plans.
- **Cost Efficiency:**
  - By detecting and addressing vulnerabilities early on, WebSentry contributes to cost efficiency by reducing the expenses associated with security breaches and data compromise.

- **Continuous Monitoring and Evaluation:**
    - Robust Architecture of organizations must be ensured.

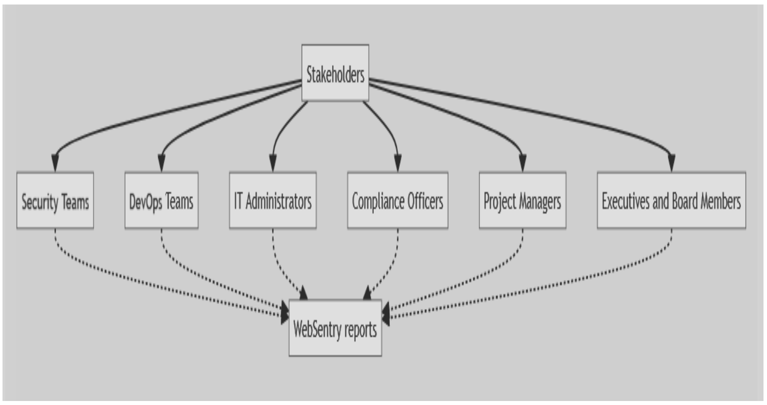
VII. Future Scope

1. **Kubernetes-specific vulnerabilities**: In the future, WebSentry will incorporate analysis of Kubernetes configurations to identify and rectify vulnerabilities that could lead to unauthorized access or data breaches. This will include assessing access controls, network policies, and pod security settings to ensure the overall security of Kubernetes deployments.
2. **Nginx-specific vulnerabilities**: WebSentry will be able to pinpoint common misconfigurations in Nginx deployments, such as inadequate SSL/TLS settings or weak cipher suites. By identifying and addressing these issues, WebSentry will enhance the security posture of Nginx servers and mitigate potential risks associated with web traffic handling.
3. **Traefik-specific vulnerabilities**: By analyzing Traefik configurations, WebSentry will detect misconfigurations that may expose sensitive services or resources to security risks. This will include assessing authentication mechanisms, routing rules, and access controls to prevent unauthorized access and potential data leaks.
4. **Consul-specific vulnerabilities**: In the future, WebSentry will identify misconfigurations and weak access control policies within Consul deployments, ensuring the security of service discovery mechanisms. This will involve evaluating ACL settings, encryption configurations, and service registration processes to mitigate the risk of unauthorized access to critical services.
5. **Linkerd Connect-specific vulnerabilities**: Through analysis of Linkerd Connect configurations, WebSentry will strengthen encryption and access control mechanisms within microservices architectures. This will include assessing mTLS configurations, service mesh policies, and identity management to protect communication channels between microservices and external entities.
6. **Istio Connect-specific vulnerabilities**: WebSentry will detect misconfigurations and insufficient access control rules within Istio service mesh deployments, mitigating associated risks. This will involve evaluating service mesh policies, RBAC configurations, and mutual TLS settings to ensure secure communication between services and enforce least privilege access controls.
7. **DDoS Prevention Mechanism**: In the future, WebSentry will incorporate advanced DDoS prevention mechanisms to defend against distributed denial-of-service attacks. This will involve implementing rate limiting, IP blacklisting, and behavioral analysis techniques to detect and mitigate malicious traffic, ensuring the availability and uptime of web services even under heavy attack.
8. **Enhanced McEliece Cryptography for SSL/TLS**: WebSentry will adopt Enhanced McEliece cryptography for SSL/TLS encryption to bolster security against quantum computing threats. This quantum-resistant encryption algorithm will provide robust protection for data transmission over secure channels, safeguarding sensitive information from potential decryption by quantum computers in the future.
9. **Zero-Day Vulnerability Detection**: In addition to known vulnerabilities, WebSentry will incorporate advanced vulnerability detection techniques to identify and mitigate zero-day vulnerabilities. By leveraging threat intelligence feeds, anomaly detection algorithms, and machine learning models, WebSentry will proactively detect and respond to emerging security threats, minimizing the risk of exploitation by cyber attackers.
10. **Cloud Security Posture Management (CSPM)**: WebSentry will expand its capabilities to include Cloud Security Posture Management, allowing organizations to assess and manage the security of their cloud environments comprehensively. This will involve continuous monitoring, configuration analysis, and policy enforcement to ensure

adherence to security best practices and compliance requirements across cloud services and infrastructure.

VIII. Stakeholders

- **Security Teams:**
    - Employ **WebSentry** for comprehensive Vulnerability Assessment and Penetration Testing (VAPT) on web applications to identify and mitigate potential vulnerabilities.
- **DevOps Teams:**
    - Collaborate with security teams to integrate security practices into development and deployment pipelines, leveraging **WebSentry** to ensure security requirements are met before deployment.
- **IT Administrators:**
    - Responsible for configuring and managing **WebSentry** instances within the organization's infrastructure.
- **Compliance Officers:**
    - Rely on **WebSentry** to ensure web applications adhere to regulatory requirements and industry standards, avoiding potential fines and penalties.
- **Project Managers:**
    - Utilize **WebSentry** to assess the security of websites submitted as projects, especially when integrated with an Automatic Project Grading system.
- **Executives and Board Members:**
    - Depend on **WebSentry** reports to understand the organization's security posture and make informed decisions about resource allocation for security initiatives.



REFERENCES

1. Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark, Balume Mburano , Weisheng Si

2. A Study of Penetration Process and Tools, Hessa Mohammed Zaher Al Shebli (NYIT, Abu Dhabi, UAE), Babak D. Beheshti, PhD (NYIT, Old Westbury,New York)

3. A Comprehensive Literature Review of Penetration Testing & Its Applications, Prashant Vats

4. An Empirical Comparison of commercial and open-source web vulnerability scanners, Richard Amankwah

5. Comparative Analysis of the Automated Penetration Testing Tools, Mandar Prashant Shah

6. A Review on Web Application Vulnerability Assessment and Penetration Testing, Urshila Ravindran and Raghu Vamsi Potukuchi

7. A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners, SULIMAN ALAZMI

8. An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities, Marwan Albahar

9. Study of the techniques used by OWASP ZAP for analysis of vulnerabilities in web

applications, Adam Jakobsson , Isak Häggström

10. OWASP (Open Web Application Security Project):SQL Injection URL: https://owasp.org/www-community/attacks/SQL_Injection

11. OWASP (Open Web Application Security Project):Cross-Site Scripting (XSS) URL: https://owasp.org/www-community/attacks/xss/

12. OWASP (Open Web Application Security Project):Command Injection URL: https://owasp.org/www-community/attacks/Command_Injection

13. OWASP (Open Web Application Security Project):Path Traversal URL: https://owasp.org/www-community/attacks/Path_Traversal

14. OWASP (Open Web Application Security Project):Broken Authentication URL: https://owasp.org/www-community/vulnerabilities/Broken_Authentication

15. OWASP (Open Web Application Security Project):Sensitive Data Exposure URL: https://owasp.org/www-community/attacks/Sensitive_Data_Exposure

16. OWASP (Open Web Application Security Project):Cross-Site Request Forgery (CSRF) URL: https://owasp.org/www-community/attacks/csrf

17. OWASP (Open Web Application Security Project):Insecure Deserialization URL: https://owasp.org/www-community/vulnerabilities/Insecure_Deserialization

18. OWASP (Open Web Application Security Project): Security Misconfiguration URL: https://owasp.org/www-community/vulnerabilities/Security_Misconfiguration

19. OWASP (Open Web Application Security Project):XML External Entity (XXE) URL: https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)

20. OWASP (Open Web Application Security Project): Insecure Direct Object References URL: https://owasp.org/www-community/vulnerabilities/Insecure_Direct_Object_References

21. OWASP (Open Web Application Security Project): Server-Side Request Forgery URL: https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

22. OWASP (Open Web Application Security Project):File Upload URL: https://owasp.org/www-community/vulnerabilities/File_Upload

23. OWASP (Open Web Application Security Project): Session Fixation URL: https://owasp.org/www-community/attacks/Session_fixation

24. OWASP (Open Web Application Security Project): Clickjacking URL: https://owasp.org/www-community/attacks/Clickjacking

25. OWASP (Open Web Application Security Project): Cross-Origin Resource Sharing (CORS) URL: https://owasp.org/www-community/attacks/CORS

26. OWASP (Open Web Application Security Project): Unvalidated Redirects and Forwards URL: https://owasp.org/www-community/attacks/Unvalidated_Redirects_and_Forwards

27. OWASP (Open Web Application Security Project): Content Security Policy (CSP) URL: https://owasp.org/www-community/attacks/Content_Security_Policy_(CSP)

28. OWASP (Open Web Application Security Project): HTTP Parameter Pollution URL: https://owasp.org/www-community/attacks/HTTP_Parameter_Pollution

29. OWASP (Open Web Application Security Project):Server-Side Template Injection URL: https://owasp.org/www-community/attacks/Server_Side_Template_Injection

30. OWASP (Open Web Application Security Project): Insecure Direct Object References URL:

https://owasp.org/www-community/attacks/Insecure_Direct_Object_References

31. OWASP (Open Web Application Security Project): Cross-Site Script Inclusion URL: https://owasp.org/www-community/attacks/cross-site_script_inclusion

32. OWASP (Open Web Application Security Project): Code Injection URL: https://owasp.org/www-community/attacks/Code_Injection

33. OWASP (Open Web Application Security Project): XML Injection URL: https://owasp.org/www-community/attacks/XML_Injection

34. OWASP (Open Web Application Security Project): Server-Side Request Forgery URL: https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

35. OWASP (Open Web Application Security Project): File Inclusion URL: https://owasp.org/www-community/attacks/File_Inclusion

36. OWASP (Open Web Application Security Project): Session Hijacking URL: https://owasp.org/www-community/attacks/Session_Hijacking

37. OWASP (Open Web Application Security Project):Brute Force Attack URL: https://owasp.org/www-community/attacks/Brute_force_attack

38. OWASP (Open Web Application Security Project): LDAP Injection URL: https://owasp.org/www-community/attacks/LDAP_Injection

39. OWASP (Open Web Application Security Project): Cross-Site Script InclusionURL: https://owasp.org/www-community/attacks/cross-site_script_inclusion

40. Google Hacking Database

41. OWASP (Open Web Application Security Project): OWASP HTTP Header Documentation URL: https://owasp.org/www-project-secure-headers/

42. Mozilla Developer Network (MDN) Web Docs:MDN HTTP Headers URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

43. RFC 7234: Hypertext Transfer Protocol (HTTP/1.1): Caching: RFC 7234URL: https://tools.ietf.org/html/rfc7234

44. Akamai Blog: Cache Poisoning URL: https://www.akamai.com/blog/security/cache-poisoning

45. PortSwigger Web Security Academy: Web Cache Poisoning URL: https://portswigger.net/web-security/web-cache-poisoning

46. OWASP (Open Web Application Security Project): OWASP Testing Guide: CRLF Injection URL: https://owasp.org/www-community/attacks/Testing_for_HTTP_Splitting/

47. PayloadsAllTheThings (GitHub repository)

48. PayloadsAllTheThings - CRLF InjectionURL: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Intrusion%20Detection%20%26%20Prevention/CRLF%20Injection.txt

49. PortSwigger Web Security Academy

50. Web Security Academy: CRLF InjectionURL: https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

51. Exploit Database URL: https://www.exploit-db.com/

.