Web Vulnerability Scanner and Assessment Tool

# Chapter 1: Introduction

In today’s digital era, corporate websites serve not only as business portals but also as repositories of sensitive data, making them prime targets for cyber-attacks. The consequences of these attacks can be dire, ranging from data breaches and intellectual property theft to severe financial losses and reputational damage. As cybercriminals continuously evolve their tactics, the vulnerabilities inherent in web applications—such as SQL Injection, Cross-Site Scripting (XSS), and Command Injection—remain exploited entry points for unauthorized access. According to the Ponemon Institute’s 2019 study, the average cost of a data breach now hovers around $3.86 million, highlighting the substantial financial burden that such incidents impose on organizations (Ponemon Institute, 2019).

Beyond the immediate financial implications, cyber-attacks on corporate websites have far-reaching consequences. For tech firms and businesses alike, these breaches can result in the loss of customer trust, legal liabilities, and a decline in market competitiveness. A report from Verizon’s Data Breach Investigations Report (DBIR, 2020) underscores that a significant percentage of cyber-attacks exploit vulnerabilities in web applications, which not only jeopardizes confidential information but also disrupts critical business operations. Furthermore, the ripple effects of these security breaches extend to society at large, where compromised personal data undermines public confidence in digital systems and services. The strategic damage—ranging from compromised intellectual property to weakened national security in cases of cyber espionage—demonstrates that the stakes are exceedingly high.

Despite the growing awareness of these risks, several challenges persist in effectively mitigating web vulnerabilities. The current market is characterized by a fragmented ecosystem of vulnerability scanners, where each tool is typically specialized for detecting certain types of vulnerabilities. This piecemeal approach complicates the process of securing web applications comprehensively. Organizations often struggle with integrating these tools into a unified system, which leaves gaps in the overall security posture. Moreover, while many of these scanners successfully detect vulnerabilities, they frequently fall short in providing actionable remediation guidance, leaving security teams with the onerous task of manually deciphering and addressing the issues.

The project presented in this document aims to tackle these challenges head-on. Our approach is to develop a comprehensive Web Vulnerability Scanner and Assessment Tool that integrates multiple open-source vulnerability scanners into a single platform. This tool focuses primarily on critical vulnerabilities such as SQL Injection, XSS, and Command Injection. What sets our solution apart is its dual functionality: not only does it detect vulnerabilities with high accuracy, but it also incorporates a detailed mitigation guide that offers step-by-step recommendations for remediation. By combining detection and mitigation, the platform is designed to bridge the gap that currently exists in the market, thereby reducing the time and complexity involved in securing corporate websites.

By leveraging the strengths of established open-source tools and building upon insights from current cybersecurity research, our project embodies a proactive and integrated approach to web security. In a landscape where cyber threats are evolving at an unprecedented pace, this integrated solution is particularly vital for tech firms and other organizations that cannot afford the strategic and financial fallout of a security breach. Ultimately, the project contributes to the broader goal of enhancing cyber resilience, ensuring that companies are better prepared to defend against and respond to the ever-changing threat landscape.

In summary, cyber-attacks on corporate websites represent a significant and growing threat, one that has both immediate and long-term repercussions. The combination of financial, strategic, and societal impacts necessitates a comprehensive approach to web security—one that our integrated vulnerability scanning platform seeks to provide. As supported by contemporary research and industry reports, there is an urgent need to move from fragmented detection tools to holistic solutions that not only identify vulnerabilities but also guide organizations in their remediation efforts.

# Chapter 2: Background

## 2.1 Understanding Vulnerabilities and Attacks

At its core, a **vulnerability** is a weakness or flaw in a system’s design, implementation, or configuration that could be exploited by an adversary to compromise confidentiality, integrity, or availability of data. In contrast, an **attack** is the active exploitation of that weakness to cause harm or gain unauthorized access. For instance, while an improperly validated input field in a web application represents a vulnerability, an attacker leveraging that weakness through a SQL injection for example attack to access sensitive customer data is carrying out an attack.

This fundamental distinction is critical; vulnerabilities exist as latent risks, and it is their exploitation—whether by automated tools or skilled adversaries—that leads to breaches and other security incidents.

## 2.2 Examples of Web Vulnerabilities

Web applications are particularly susceptible to a variety of vulnerabilities, including but not limited to:

* **SQL Injection:** An attacker can manipulate backend database queries by injecting malicious SQL code into input fields, potentially gaining access to sensitive information or modifying the database.
* **Cross-Site Scripting (XSS):** By injecting malicious scripts into web pages viewed by other users, attackers can steal session cookies, deface websites, or redirect users to phishing sites.
* **Command Injection:** This involves inserting system commands into input fields, causing the server to execute unintended commands.

These vulnerabilities are not only common but also highly damaging if exploited. Their impact extends from data breaches to full system compromise, thereby affecting both corporate reputation and financial stability.

## 2.3 Vulnerability Scanners in the Market

To defend against such risks, organizations deploy **vulnerability scanners**—tools that automatically assess systems, networks, and web applications for known vulnerabilities. The market offers a spectrum of these tools, broadly categorized into paid (commercial) and open-source solutions.

* **Commercial Scanners:** Products like Nessus, Acunetix, and Qualys are known for their robust vulnerability detection capabilities, ease of integration into enterprise workflows, and extensive reporting features. They often come with subscription models that can range from a few thousand dollars per year for small-scale deployments to tens of thousands for enterprise-level services.

These tools typically include dedicated support and regular updates, which can be essential for organizations with limited in-house security expertise.

* **Open-Source Scanners:** Tools such as OWASP ZAP, OpenVAS, Nikto, and SQLMap offer many of the same core features without licensing costs. However, they usually require more technical know-how for installation, configuration, and maintenance. Open-source scanners allow for greater customization and can be integrated with other tools, making them ideal for organizations that want a tailored solution while keeping costs low.

While commercial scanners offer streamlined, out-of-the-box functionality, open-source tools provide flexibility and cost-effectiveness. The price difference is significant: while enterprise subscriptions may run into tens of thousands of dollars annually, open-source solutions are typically free—though they may incur additional costs related to deployment and ongoing maintenance.

## 2.4 The Case for Integration

One promising approach to overcoming the limitations of both paid and standalone open-source scanners is to **integrate multiple open-source tools into a unified platform**. Such integration offers several benefits:

* **Comprehensive Coverage:** By combining the strengths of different scanners, a unified tool can detect a broader range of vulnerabilities than any single scanner on its own. For example, while OWASP ZAP excels at detecting XSS in web applications, tools like SQLMap are specialized in uncovering SQL injection flaws. An integrated platform leverages these complementary strengths to provide a more holistic security assessment.
* **Cost Efficiency:** Integrating free and open-source solutions can dramatically reduce licensing costs while still offering high-quality vulnerability detection. This is particularly beneficial for small to medium-sized enterprises (SMEs) that may not have the budget for expensive commercial tools.
* **Centralized Reporting and Management:** A unified platform can consolidate scan results, correlate findings from various tools, and offer actionable remediation guidance from a single dashboard. This streamlines the process for security teams and improves the overall efficiency of vulnerability management.
* **Customization and Flexibility:** Open-source tools are inherently adaptable. When integrated, they can be configured to meet the specific needs of an organization, incorporating custom scanning routines and tailored reporting formats that align with existing security processes.

By integrating these tools, our project not only enhances the detection of vulnerabilities across multiple vectors but also provides a mitigation guide—bridging the gap that often exists between vulnerability detection and remediation. This approach aligns with the industry’s move toward comprehensive, cost-effective, and agile security solutions.

## 2.5 Enriching the Context

Beyond technical capability, the integration of open-source scanners addresses broader market challenges. Many organizations struggle with fragmented security tools that operate in silos, leading to gaps in coverage and inefficient incident response. A consolidated platform simplifies security operations, allowing teams to quickly identify, prioritize, and remediate vulnerabilities. Moreover, as cyber threats continue to evolve, an integrated system can be more readily updated and extended with new modules, ensuring that security measures keep pace with emerging attack vectors.

## 2.6 Implementation Overview

The application is built using Python’s Flask web framework, which provides a lightweight yet flexible web server and routing system. The main file (VulnScanX.py) sets up several endpoints for interacting with the scanner:

* Home Page and User Interface:  
  The root endpoint (/) renders the home page (via index.html), while additional routes (such as /results and /blog) provide pages for viewing scan outcomes and educational content on vulnerabilities.
* Scan Initiation:  
  The /start-scan endpoint accepts JSON input from the client. Parameters such as the target URL, HTTP headers, scan type, and toggles for subdomain enumeration, crawling, and specific vulnerability tests (XSS, SQL injection, command injection) are collected. Based on these parameters, the system spawns a separate thread to run either a full scan or a custom scan. This use of threading ensures that the Flask app remains responsive while resource-intensive scans are performed in the background.
* Results Aggregation:  
  Once scans have been completed, the /getresults route reads from a JSON file (vulnerabilities.json) where all vulnerability findings are aggregated. This file is updated by each scanning module through a common helper function that appends new findings to the JSON structure.

## Integration of Multiple Vulnerability Scanners

The system integrates several specialized open-source scanners through modular Python files. Each module is responsible for a particular vulnerability category and follows a similar pattern:

### Command Execution via Subprocess:

Each module (e.g., sqlinjection.py, dalfox.py, commandinjection.py, and the XSStrike module) calls the corresponding external tool (such as SQLMap, Dalfox, Commix, or XSStrike) using Python’s subprocess.run(). The commands are built dynamically based on parameters (like target URL, cookies, or scan level) provided by the user or set in the scan routines.

### Output Parsing and Data Extraction:

* + - SQL Injection Module: Invokes SQLMap with parameters (e.g., --flush-session, --batch, and specified risk levels) and parses the output for vulnerable parameters and payloads. It then constructs a vulnerability record that includes details such as the affected URL, the parameters tested, and a remediation suggestion.
    - XSS Scanning with Dalfox and XSStrike: The Dalfox module calls the Dalfox command-line tool on a file containing URLs. It processes the returned output—decoding payloads and associating them with query parameters—to build a structured report on potential XSS vectors. Similarly, the XSStrike integration (named here as xsstrikee) runs XSStrike in crawl mode, extracts details like vulnerable components and severity, and organizes these into JSON.
    - Command Injection Module: Utilizes Commix to test URLs for injectable parameters. It scans the output for strings indicating that parameters are injectable and extracts sample payloads. These findings are then stored along with a severity rating and remediation guidance.

### Common Data Storage:

All modules use a shared helper function (save\_to\_json()) to append vulnerability findings to the central vulnerabilities.json file. This standardization facilitates a unified view of vulnerabilities detected by different scanning techniques.

## Workflow and Scan Modes

* **Full Scan:**  
  When a “full” scan is requested, the system always performs reconnaissance (via a bash script called by the recon() function) to enumerate subdomains and crawl the target website. Following reconnaissance, all integrated tests are executed in sequence—XSS scanning with Dalfox, command injection tests, and SQL injection tests.
* **Custom Scan:**In a “custom” scan, the user can selectively enable features (such as subdomain enumeration, crawling, or specific vulnerability tests). If crawling or subdomain enumeration is toggled on, the system runs the reconnaissance process; otherwise, it directly writes the target URL to a file for subsequent scanning. Each selected test module is then executed according to the user’s configuration.
* **Threaded Execution:**To keep the web interface responsive, the scan processes (full or custom) run in a separate thread. This asynchronous execution allows users to start a scan and later retrieve results without blocking the main Flask application.

## Interfacing with External Tools

A notable design aspect is the reliance on external, well-established open-source tools to conduct the actual vulnerability assessments. For example:

* **Reconnaissance:**A bash script (automate.sh) is invoked to perform subdomain enumeration and crawling, capturing URLs in a file (urls.txt). This file is then used by subsequent scanning modules.
* **Tool-Specific Modules:**  
  Each scanner module is a thin wrapper around its command-line tool, focusing on constructing the right command, handling output, and parsing results. This modular design makes it straightforward to add support for additional vulnerability scanners in the future.

# Chapter 3: Implementation

## 3.1 Cross-Site Scripting (XSS) Detection

To address XSS vulnerabilities, our system integrates the open-source scanner Dalfox. This tool is specifically designed to discover XSS issues in web applications by crawling URLs and identifying exploitable query parameters. We have built a dedicated Python module (dalfox.py) that wraps Dalfox’s functionality, parses its output, and standardizes the results for inclusion in our unified vulnerability report.

**3.1.1 Workflow and Invocation**

When a scan is initiated via our Flask-based application (see VulnScanX.py), the system prepares a file (typically urls.txt) containing the list of URLs extracted either by the reconnaissance process or provided directly by the user. The function “run\_dalfox\_on\_url(url\_file)” in our dalfox.py module is then called. This function constructs a command to execute Dalfox in “file” mode:  
” command = ["dalfox", "file", url\_file]”

Dalfox runs against the URL list, and its output is captured for further processing. If the command fails (i.e., a nonzero return code), an error entry is saved to our JSON results file.

**3.x.2 Output Parsing and Data Extraction**

Once Dalfox completes its scan, the module reads its standard output line by line. Each line is analyzed with regular expressions to extract the vulnerable URL along with any query parameters and payloads. The following steps illustrate the process:

1. **URL Extraction and Normalization:**  
   The code uses a regex pattern to match URLs from the tool’s output. For each found URL, the module employs Python’s “urllib.parse” functions to:
   * Break down the URL into components (scheme, netloc, path, query).
   * Construct a normalized base URL (comprising the scheme, host, and path) that excludes query parameters.
2. **Parameter and Payload Parsing:**  
   The query part of the URL is parsed to obtain a dictionary of parameters. For every parameter, the module iterates through its associated payloads. Since Dalfox may return encoded payloads, the code decodes them using “urllib.parse.unquote()” so that the exact injected code can be recorded.
3. **Result Aggregation:**  
   For each base URL, all associated vulnerable parameters (along with their decoded payloads) are aggregated in a dictionary. Finally, for each URL entry with vulnerabilities, a JSON record is constructed with fields for the vulnerability type ("XSS"), an assigned severity level (set to "Medium" by default), the affected URL, and a detailed description. The description includes the list of vulnerable parameters and their payloads, and it also provides a link to a blog post with guidance on remediating XSS issues.

Below is an excerpt of the relevant code snippet from dalfox.py that illustrates these steps:

**“Python Code”**  
for line in result.stdout.splitlines():   
 url\_match = re.search(r"(https?://[^\s]+)", line)   
 if not url\_match: continue

full\_url = url\_match.group(0)   
parsed\_url = urlparse(full\_url)   
base\_url = parsed\_url.scheme + "://" + parsed\_url.netloc + parsed\_url.path   
query\_params = parse\_qs(parsed\_url.query)

if base\_url not in findings:   
findings[base\_url] = []

for param, payloads in query\_params.items():   
 for payload in payloads:   
 decoded\_payload = urllib.parse.unquote(payload)   
 findings[base\_url].append({   
 "parameter": param,   
 "payload": decoded\_payload   
 })

After iterating through all the output lines, the module loops over the collected findings and calls the helper function “save\_to\_json()” to append each XSS vulnerability record into the “vulnerabilities.json file”.

**3.1.3 Integration into the Main Application**

Within our main Flask application (VulnScanX.py), the XSS scanning routine is triggered as part of both full and custom scan workflows. When a user selects the "xss" option in the scan configuration, the code calls:

**“Python Code”**  
if(xss == "on"):   
 dalfox.run\_dalfox\_on\_url(absolute\_path)

Here, “absolute\_path” is the absolute path to the file containing URLs (generated from the reconnaissance process or manually updated). By running Dalfox in this way, our system seamlessly integrates XSS vulnerability detection with other tests (like SQL injection and command injection). Once Dalfox processes the URLs, its results are automatically merged with the overall scan report available through the “/getresults” endpoint.

**3.1.4 Reporting and Remediation Guidance**

Each XSS finding is stored with a standardized format. For example, a sample JSON entry might appear as:

**“json”**

{   
"vulnerability": "XSS",   
"severity": "Medium",   
"url": "https://example.com/search",   
"description": "Vulnerable parameters: [{'parameter': 'q', 'payload': '<script>alert(1)</script>'}] \nwhat you should do : http://127.0.0.1/blog?post=xss"   
}

This not only informs security teams of the precise vulnerability details but also directs them to additional documentation and remediation advice hosted on our blog. By centralizing the scan results, our system provides a holistic view of the target’s security posture.

## 3.2 SQL Injection Detection

## 3.3 Command Injection Detection

# Chapter 4: Conclusion & Future Work