Master thesis Computing Science

Radboud University

Improving Burp Scanner using benchmarks and BChecks

Supervisor:
Bart Mennink
b.mennink@cs.ru.nl

Author:
Maurice Dibbets (s1022543)
maurice.dibbets@ru.nl

Second reader:
Hugo Jonker
hugo.jonker@ou.nl

Internship supervisor:
Sebastiaan Groot
sebastiaan.groot@kpn.com

July 2024

Contents

Acknowledgements

I want to thank Bart Mennink for helping me during the academic writing process, Hugo Jonker for agreeing to be my second reader for a second time and providing feedback on my research methodology, and Sebastiaan Groot for guidance with the technical aspects of this thesis and interesting discussions about red teaming.

Abstract

The increasing prevalence of web applications in our daily lives has increased our susceptibility to identity theft, financial fraud, supply chain attacks, and privacy violations. As part of a secure development lifecycle, dynamic analysis (DAST) tools help uncover vulnerabilities in web applications before they are rolled out to production systems. However, these tools can sometimes miss vulnerabilities or report non-existent vulnerabilities. This thesis evaluates Burp Scanner using six different benchmarks. Using the results of this evaluation and an analysis of the detection limitations of Burp Scanner and its publicly available extensions, we implement improvements to existing scan checks using the BChecks feature and provide recommendations to enhance Burp Scanner. We also critically analyze the relevance of web vulnerability scanner benchmarks, determine the impact of environmental factors on benchmark results, and provide insight into determining the quality of a scan check. Numerous improvements to Burp's crawler and Burp Scanner are identified that could increase coverage on our selected benchmarks and possibly against real-world applications. Additionally, broken test cases that threaten benchmark results' reliability are identified. Using the insights from this thesis, organizations could improve the effectiveness of their security automation by performing their own analysis of their preferred scanning tools.

1 Introduction

Web applications are used by billions of people worldwide [1, 2] to communicate with friends and family, manage their finances, find a job, order products, and for many other purposes. According to [3], the predicted average job growth for web developers and digital designers is 23% from 2021 to 2031. Additionally, more than half of small enterprises in the European Union have a website, according to estimates from [4]. It seems like the web plays an increasingly prominent role in our lives, and the security of web applications is vital to protect against identity theft, financial fraud, supply chain attacks, and to protect user privacy. To achieve appropriate security levels, companies follow a secure development lifecycle as recommended by [5].

The secure development lifecycle involves multiple steps, including manual and automated processes. During the automated processes, developers utilize static (SAST) and dynamic analysis (DAST) tools to uncover vulnerabilities in their applications before rolling out code changes to production systems [6]. However, these tools can sometimes miss vulnerabilities (false negatives) or report non-existent vulnerabilities (false positives). When a tool fails to identify a vulnerability or produces an overwhelming number of false positives, it could compromise the application's security, especially if other parts of the secure development lifecycle are not followed due to budget constraints or a lack of qualified personnel.

Burp Suite Professional is a multipurpose tool developers and penetration testers use to assess the security of web applications. This thesis aims to evaluate and improve Burp Scanner and the scanner extensions available on the BApp store and GitHub at the time of this writing [7, 8]. Based on this evaluation, we identify multiple improvements to Burp's crawler and Burp Scanner that could be used to increase the thoroughness and accuracy of scans.

The first set of improvements involves changing the JavaScript evaluation, HTML parsing, and response header parsing of Burp's crawler. However, this is left out of scope because Burp Suite Professional is closed-source software, and no official support enables us to improve its existing crawler instead of adding a completely different crawler. The second set of improvements involves adding new test inputs to the command injection, path traversal, CSS injection, and server-side template injection (SSTI) scan checks. We can use the new BCheck feature to facilitate this [9]. Additionally, we propose improvements to the benchmarks used during our evaluation and have rediscovered several bugs within the benchmarks that cause misleading results.

To summarize, our methodology involves scanning six benchmarks, identifying the test cases that produce false negatives, analyzing the source code of each test case that results in a false negative, and categorizing the false negatives into a list of variants for each vulnerability type to implement and evaluate BChecks that mitigate the false negatives.

In Section 2, we will provide a detailed description of our methodology, including the version numbers of the software used, setup instructions, and troubleshooting issues encountered during the scans. In Section 3, we will present the scan results and provide the BChecks mentioned earlier. In Section 4, we will analyze the limitations of the BCheck feature, discuss how benchmarks could be improved, and provide additional context, such as the limitations of security automation in general. We will also offer recommendations to PortSwigger on enhancing Burp Scanner's capabilities. Finally, in Section 5 we will propose future work and list the main lessons from our research.

2 Methodology

We will evaluate the thoroughness of Burp's crawler using Security Crawl Maze [10] (see Section 3.1) and Web Input Vector Extractor Teaser version 3 as included in the OWASP Broken Web Applications Project [11, 12] (see Section 3.2).

Furthermore, we will evaluate and improve Burp Scanner using the steps outlined below:

1. Use OWASP Benchmark v1.2 (see Section 2.1), Reinforced-Wavsep v1.8.1 (see Section 2.2), Firing Range v0.48 (see Section 2.3), and websitesVulnerableToSSTI commit `4427a3316dd9e51162bacefbcd1cb7da7eecb8ec` (see Section 2.4) to evaluate the accuracy of Burp Suite Professional v2023.12.1.5 up to v2024.4.5.

2. Identify the test cases that result in false negatives and review the source code of each test case (see sections 3.3, 3.5, 3.7, and 3.9).

3. Attempt to categorize the false negatives into a list of variants for each vulnerability type to better understand the scanner's limitations (see sections 3.3, 3.5, 3.7, and 3.9).

4. Implement high-quality BChecks (see Section 2.6) to reduce the number of false negatives if the extensions on the BApp store, the extensions or BChecks on GitHub, or Burp Bounty Professional can also not identify the vulnerability [8] (see sections 3.4, 3.6, 3.8, and 3.10).

5. Repeat the evaluation to validate that the number of false positives has not increased with the added BChecks, but the number of false negatives has decreased (see Section 3.11).

In sections 2.1, 2.2, 2.3, and 2.4, we discuss how we configured and performed the scans on our selected benchmarks. In Section 2.5, we list the extensions used to mitigate some false negatives identified during the scans. In Section 2.6 and Section 2.7, we provide the background information needed to implement high-quality BChecks.

2.1 Scanning the OWASP Benchmark

This section describes the setups and configuration used during our scans of the OWASP Benchmark. Initially, we attempted to do a full crawl and audit scan of each endpoint. However, in Section 3.3, we describe the changes we made because we needed to perform follow-up scans due to the number of false negatives.

2.1.1 Initial scan setups

We used two separate setups for our target virtual machines. This allowed us to compare the scan results for a Linux and Windows target and evaluate how they differed.

Linux setup

We installed our tools on a virtual machine (VM) for our initial scan with a minimal Ubuntu 22.04.3 installation, 13 GB of RAM, and 12 CPU cores. To set up the OWASP Benchmark, we used the Docker image from [13], the instructions from the Quick Start page [14], and the instructions from the Docker documentation [15]. To set up Burp Suite Professional, we followed the steps of [16]. Burp Suite Professional was used on the same VM as the OWASP Benchmark. One significant issue was that the OWASP Benchmark crashed repeatedly during our scans. Using the Logger tab, we noticed that

the issue occurred during scans of the command injection endpoints and could be solved by turning off time-based command injection checks. The cause of the problem was that the `ping` command uses a count parameter to determine how many ICMP packets should be sent, and on Linux, it runs indefinitely without this parameter.

Windows setup

We set up our scanning tools for the second scan on a virtual machine with Windows 10 Enterprise, 4 GB of RAM, and 8 CPU cores. To install the OWASP Benchmark, we followed the instructions using Java 8 and added Maven and the Java JDK to the Path environment variable [14]. For Burp Suite Professional, we followed the installation steps [16].

We used a similar configuration to that from Section 2.1.2 but re-enabled the time-based command injection checks because they do not cause issues on Windows.

Additionally, we made a significant change by adding more RAM to our desktop, bringing the total to 128 GB of host memory. This allowed us to run multiple scans and 20 virtual machines simultaneously. We conducted parallel scans by splitting the list of endpoints into 20 sub-lists using a basic script (`create-chunks.py`) and specifying the sub-lists in the URLs to scan option [17].

2.1.2 Thorough scan configuration for the OWASP Benchmark

Because our goal is to identify what vulnerabilities in the OWASP Benchmark are missed by Burp Scanner, we want to perform the most thorough scan available so that we do not end up implementing a check already included in the scanner. Our crawl configuration (`OWASP Benchmark Crawl.json`) can be found in `benchmark-results.zip` [17].

The auditing configuration was modified as follows:

- Audit speed: Thorough.
- Audit accuracy: Minimize false negatives.
- We deactivated "Skip checks unlikely to be effective due to insertion point's base value" and "Consolidate frequently occurring passive issues".
- We checked each option under "Modifying Parameter Locations".
- We removed each item in the Ignored Insertion Points list.
- We deactivated each option in the Frequently Occurring Insertion Points section.
- We deactivated static analysis techniques in the JavaScript Analysis section.
- In the Issues Reported section, we selected OS command injection without time-based techniques (i.e., we used the edit detection methods menu to turn off the "Time delays" technique because it resulted in application crashes, see Section 2.1.1), SQL injection, SQL injection (second order), File path traversal, File path manipulation, LDAP injection, XPath injection, Cross-site scripting (stored), and Cross-site scripting (reflected).

2.2 Scanning Reinforced-Wavsep

This section briefly describes how we set up a virtual machine running Reinforced-Wavsep and our scan configurations.

2.2.1 Installing the prerequisite tools

To install Reinforced-Wavsep, we followed the instructions of [18] and used the old style approach. To install it on Windows, we were able to follow the same instructions by replacing the Linux binaries with their Windows equivalents. We assigned 8 CPU cores to both the Linux (Ubuntu 22.04.3) and Windows (Windows 10 Enterprise) VMs and allowed the amount of RAM to be adjusted dynamically and use up to 100 GB of the memory on our desktop (128 GB). In practice, the VMs used about 10 GB of memory.

2.2.2 Scan configurations

Our scan configurations can be found in `benchmark-results.zip` at [17]. We performed targeted scans for each vulnerability type instead of a full scan of all endpoints.

2.3 Scanning Firing Range

The Firing Range is hosted at `http(s)://public-firing-range.appspot.com/`. It largely focuses on client-side vulnerabilities such as cross-site scripting. The Firing Range does not require any additional setup. Because of time limitations and to reduce redundant results, we did not perform cross-platform scans of the Firing Range. Unlike other benchmarks, the Firing Range contains several test cases where the results depend on whether an HTTP or HTTPS connection is used during the scan. Namely, the `allowInsecureScheme` endpoint of the CORS checks, the `mixedcontent` endpoint, and the `stricttransportsecurity` endpoint require the scan to be performed over an HTTPS connection. Firing Range does not publish a list of false positive endpoints. However, we were unable to come up with functional exploits for modern browsers of the `/escape/serverside/encodeUrl/` test cases and the `escapeHtml` test cases below:

- /escape/serverside/escapeHtml/body
- /escape/serverside/escapeHtml/body_comment
- /escape/serverside/escapeHtml/head
- /escape/serverside/escapeHtml/textarea

We do not exclude the possibility that XSS payloads exist that exploit the parsing behavior in browsers that we overlooked.

Of the DOM XSS cases, `/address/location/assign` and `/address/location/replace` did not seem exploitable. The content sniffing test cases also did not seem exploitable.

As before, we only scanned the test cases that Burp Scanner supports. Hence, the `badscriptimport`, `flashinjection`, `insecurethirdpartyscripts`, `leakedcookie`, and `reverseclickjacking` endpoints were kept out of scope.

2.4 Scanning websitesVulnerableToSSTI

To test how Burp Scanner performed on a more language-specific vulnerability type (SSTI), we used the websitesVulnerableToSSTI project [19]. We used a VM with Ubuntu 22.04.3, 8 CPU cores, and allowed the dynamic allocation of RAM (up to 100 GB). In practice, the VM used about 10 GB of RAM. During our scan, we discovered minor bugs in `python/flaskBasedTests/src/pythonEval.py` and `javascript/src/eval.js`. In `pythonEval.py` we had to replace line 86:

```
result = eval("%s % 7" % expression)
```

by

```
result = eval(expression + "% 7")
```

because of an incomplete format exception.

In `eval.js`, we had to replace each of the form action attributes with:

- /javascript/eval/simple
- /javascript/eval/evalInsideDoubleQuote
- /javascript/eval/evalInsideSingleQuote
- /javascript/eval/evalInsideExistingQuote
- /javascript/eval/evalBlind

2.5 Using extensions to mitigate false negatives

We carefully evaluated each extension in the BApp store and compared them to the list of vulnerability types in our selected benchmarks. Most scanner extensions enhance the capabilities of Burp Scanner but do not add additional payloads to check for the specific vulnerabilities included in the benchmarks. Examples include ActiveScan++ [20], Backslash Powered Scanner [21], and CMS Scanner [22].

The remaining extensions are specifically designed to enhance Burp's existing scan checks. We used the following extensions from the BApp store:

- SHELLING v2.0 is used to identify command injection vulnerabilities [23].
- Sentinel v0.9.1 is used to identify XSS, SQL injection, command injection, or template injection vulnerabilities [24].
- SQLiPy v0.8.5 is used to identify SQL injection vulnerabilities [25].
- Additional Scanner Checks v1.4 [26] adds checks for DOM XSS, missing or misconfigured HTTP headers, duplicate headers, and insecure redirections from HTTP to HTTPS.

We also used the following extensions from GitHub. We chose these extensions because they offered enhanced scan checks that were not included in the extensions on the BApp store:

- Burp DOM Scanner v1.0.1 [27] uses a custom crawling and scanning engine to identify DOM XSS vulnerabilities.
- ESLinter (commit `f2b58eaee43cd724678e04c602f3844a67a69558`) [28] uses ESLint to add JavaScript linting rules to check for DOM XSS and other client-side vulnerabilities.
- Blind SSTI Scanner v1.0 adds additional out-of-band detection techniques to identify blind template injection vulnerabilities [29, 30].
- tplmap (commit `616b0e527f62dd0930e6346ede6bef79e9bcf717`) supports over 15 template engines and checks for template injection vulnerabilities [31].

Finally, we used Burp Bounty Professional v1.0.8. It supports many vulnerability types, including command injection, path traversal, XSS, SQL injection, unvalidated redirect, and template injection vulnerabilities [32]. Hence, to our knowledge, it is the most comprehensive attempt at improving Burp's existing scan checks.

We only scanned the endpoints with false negatives, as the SQLiPy and Sentinel extensions do not offer straightforward methods to scan every endpoint listed in the Burp site map [33].

2.6 Determining the quality of a BCheck

In this section, we will attempt to define qualitative and quantitative measurements that can be used to determine the quality of a BCheck (or other scan checks). We will use these measurements while implementing BChecks. For consistency with the previous scan configurations we used while scanning the benchmarks, we performed targeted scans of each vulnerability type to test the BChecks. Hence, the quality measurements do not necessarily reflect how the BChecks perform against real-world applications.

2.6.1 Number of requests

Vulnerability scans are notorious for sending many requests to the tested application [34]. To illustrate, our initial scan of the OWASP Benchmark resulted in over 10 million requests. We want to minimize the additional number of requests to reduce the time it takes to perform our scan and reduce the cost of a scan for application owners. The additional number of requests is based on factors such as the number of new payloads used by a check and any follow-up requests used to confirm a potential finding. It is not always a linear function of the number of insertion points [35].

2.6.2 Effect on runtime

The number of requests resulting from a BCheck is insufficient to determine its implementation's efficiency. For example, using complex regular expressions to check for patterns in HTTP responses that indicate a vulnerability could significantly slow down the scans even when we send a few requests for that check. Hence, we will try to minimize the runtime of the BCheck implementations.

2.6.3 Generalizability

Adding ad hoc payloads that look for specific vulnerabilities is usually straightforward, but a check is more valuable when it can identify multiple vulnerabilities using a single payload. We often refer to payloads that can operate in various contexts as polyglot payloads. This requirement is related to minimizing the total number of requests.

2.6.4 Improved accuracy

Suppose a company is highly interested in detecting cross-site scripting, SQL injection, or any other vulnerability that could significantly impact its environment. They should be able to select the appropriate BCheck to increase their coverage. Our objective is to minimize the number of false negatives while avoiding increasing the number of false positives. This trade-off is a crucial aspect of developing vulnerability scanners [36].

2.6.5 Confidence levels

By default, Burp Scanner provides a confidence level for each issue it reports on [37, 33].

The confidence levels exist because introducing a few false positives with a new scan check is not unusual. Theoretically, we could determine the confidence levels of each check by recording how many false positives it generates when scanning the benchmarks. In practice, however, the confidence level might vary depending on the complexity of the target environment's architecture.

Hence, we will use the following qualitative criteria to determine the confidence level of our checks because using quantitative criteria would result in inflated confidence levels:

- Certain: It is highly likely the check uncovered a vulnerability. For example, the application sent an HTTP request to a Burp Collaborator server due to an out-of-band payload [38].
- Firm: It is likely the check uncovered a vulnerability. For example, a specific string included in our malicious request seems to have been evaluated in some way based on the HTTP response.
- Tentative: It is possible the check uncovered a vulnerability. For example, a time delay occurred that could have been caused by a network timeout, or there was a slight difference in the HTTP response size that added headers or tokens with a variable length could have caused [39].

2.7 Components of a BCheck

A BCheck is defined by several properties [40]. For completeness, we will briefly summarize the documentation. The first property is the metadata, which "contains information about the check itself". It contains basic information such as the check's description, the name of the check, and the version of the BCheck definition language it is written in. The second property is a "given... then" statement, or control flow statement. It determines when to apply a BCheck during a scan. The documentation lists four types of "given... then" statements [40].

Control flow statements can also contain conditional keywords. The conditional keywords support if-else logic and are used to determine whether sending a particular payload in a scan request might have resulted in a valid finding based on an HTTP response.

We will elaborate on specific BChecks when we include examples of them in other sections.

3 Results

This section will describe the scan results of our selected benchmarks, our attempts to mitigate the false negatives using the scanner extensions from Section 2.5, and the BChecks we created to mitigate the remaining false negatives (Section 3.11). The complete list of false negatives and positives, scan configurations, crawl results of the Security Crawl Maze and WIVET, and the BChecks can be found at [17].

3.1 Categorizing the missed endpoints of Security Crawl Maze

We performed multiple crawls with different crawl strategies to find potential limitations in Burp's crawler. Burp's crawler found 75 of 91 endpoints of the Security Crawl Maze. The missed endpoints could be put in four different categories. These include JavaScript parsing (I), response header parsing (II), HTML parsing (III), and the "miscellaneous" category (IV) (corresponding to the `/test/misc/` endpoints of the Security Crawl Maze).

We identified the limitations below:

I. Single-page applications frequently use JavaScript frameworks to construct requests. Burp's crawler seems to miss some framework-specific event handlers and attributes. It also missed a reference to an endpoint that used string concatenation.

II. References to locations in the Link header were missed [41].

III. Various HTML attributes were missed such as the `srcset` attribute [42].

IV. The URLs from `sitemap.xml` files are only extracted when the sitemap is referenced in a `/robots.txt` file.

The majority of false negatives were related to limitation III.

3.2 Categorizing the missed endpoints of WIVET

As in Section 3.1, we performed repeated crawls. Burp's crawler found 65 of 85 endpoints of WIVET. The missed endpoints could be put in three different categories. These include JavaScript parsing (I), Adobe Flash Player (II), and HTML parsing (III).

We identified the limitations below:

I. See Section 3.1.

II. Adobe Flash Player is no longer supported [43]. Burp's crawler missed an endpoint referenced in a Flash Player (SWF) file.

III. Burp's crawler does not seem to extract references in responses that contain a 302 redirect.

The majority of false negatives were related to limitation I.

3.3 Categorizing the false negatives and positives of the OWASP Benchmark scan

This section discusses the outcomes of our successful scans from Section 2.1.1. Initially, we did not detect LDAP injection, path traversal, SQL injection, or XPath injection vulnerabilities using the configuration described in Section 2.1.2. However, we addressed that by following instructions from [44].

We conducted targeted scans of specific sections and their corresponding vulnerability types, adjusting the number of concurrent requests to one instead of ten and the crawl strategy to Faster or Fastest. We performed crawls using the Crawl scan type and then used the results to conduct scans using the "Audit selected items" scan type.

Due to the need for more than 30 scans, we manually reviewed the expected results CSV [45] instead of using OWASP Benchmark's scoring card [14]. The two crawler-related issues from Section 3.3.2 repeatedly interfered with our analysis.

Vulnerability	Number found (Linux)	Number found (Windows)
Command injection	66 of 126	78 of 126
LDAP injection	19 of 27	19 of 27
Path traversal	50 of 133	50 of 133
SQL injection	187 of 272	187 of 272
XPath injection	8 of 15	8 of 15
XSS (Cross-Site Scripting)	186 of 246	186 of 246
Secure Cookie Flag	22 of 36	22 of 36

Table 1: Results of OWASP Benchmark scan

3.3.1 Command injection

Linux results

As discussed in Section 2.1.1, we could not use time-based techniques to fully scan for command injection vulnerabilities. However, when those techniques were enabled, false positives were generated because of network timeouts before the application crash.

Using the other techniques, Burp Scanner could identify 66 of 126 command injections. Forty-eight false negatives resulted from a crawler issue discussed in Section 3.3.2. The remaining 12 false negatives (i.e., BenchmarkTest00499, BenchmarkTest00500, BenchmarkTest01285, BenchmarkTest01609, BenchmarkTest01610, BenchmarkTest01689, BenchmarkTest01864, BenchmarkTest02146, BenchmarkTest02147, BenchmarkTest02154, BenchmarkTest02249, BenchmarkTest02250) were more interesting. The above test cases pass the command directly to `Runtime.getRuntime().exec()`. Hence, when Burp Scanner attempts to inject into an existing command using the pipe character | [46], the attempt fails, and the command injection is undetected.

Windows results

When scanning the OWASP Benchmark on Windows, the 12 false negatives from the Linux scan were absent. This might be because we inject into the `echo` command instead of passing a command directly and because `Runtime.getRuntime().exec()` seems to process shell meta-characters differently on Windows and Linux systems:

```
if (osName.indexOf("Windows") != -1) {
    cmd = org.owasp.benchmark.helpers.Utils.getOSCommandString("echo");
}
```

Listing 1: Platform specific logic from BenchmarkTest00499.java

It is unclear why this prefix was added to the test cases.

Additionally, we could use the time-based detections for command injections because the issue from Section 2.1.1 does not occur on Windows systems.

3.3.2 LDAP injection

Linux results

19 of 27 LDAP injection vulnerabilities were detected.

The OWASP Benchmark contains test cases related to weak randomness. Naturally, those test cases result in response size differences because numbers of different orders of magnitude are displayed on the weak randomness pages, and the pages generate new numbers on every page load. This frequently results in false positives [47].

The first category of false negatives involved test cases with custom headers set in the `submitHeaderForm` function of `testsuiteutils.js`.

Specific test cases like BenchmarkTest00012 and BenchmarkTest01023 set a custom header upon submitting an HTML form. Unfortunately, Burp's crawler does not seem to set this header, regardless of the thoroughness mode used. Burp Scanner does not support headers in its "modifying parameter locations" section. Hence, the only way to identify the vulnerabilities in these test cases is by specifying that the vulnerable header must be scanned using a manual insertion point. Note that this is likely a conscious design choice because Burp Scanner only uses active checks against `Referer`, `User-Agent`, and custom headers starting with `X-` by default [48, 49].

The second category of false negatives involved the test cases where one of the request parameters had to be set to BenchmarkTestX (where X is the benchmark test ID) before the LDAP injection could be triggered. One example of such a test case is BenchmarkTest02299. This was also related to Burp's crawler because the parameter was set dynamically by the `submitParameterNamesForm` function of `testsuiteutils.js`, and Burp's crawler did not dynamically add the required parameter to the HTTP request.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.3.3 Path traversal

Linux results

Burp Suite found 50 of 133 path traversals. It generated Informative [50] file path manipulation issues for the vulnerable endpoints but did not generate "file path traversal" issues. Because the two vulnerability categories and checks are similar, we added them to the number found column in Table 1.

Because there were 83 false negatives, we reviewed all 133 test cases vulnerable to path traversals to categorize them. Most false negatives resulted from the crawler issues we identified in Section 3.3.2.

In general, we identified three types of variants within the test cases:

- The first variant (file read variant) reads a file and prints the HTML-encoded start of the file (e.g., BenchmarkTest00001 and BenchmarkTest00062).
- The second variant (file write variant) opens a file with write access and prints an HTML-encoded message of the path it will write to (e.g., BenchmarkTest00002 and BenchmarkTest00028).
- The third variant (file access variant) accesses a file and prints whether the file access was successful and the HTML-encoded path it used (e.g., BenchmarkTest00011 and BenchmarkTest00040).

The file read variant was reported as file path manipulation instead of file path traversal because the application applies HTML encoding to the start of the file. Burp Suite looks for specific strings that are not present in such a response. After recompiling BenchmarkTest00001 and changing:

```
response.getWriter()
    .println(
            "The beginning of file: '"
                    + org.owasp.esapi.ESAPI.encoder()
                        .encodeForHTML(fileName)
                    + "' is:\n\n"
                    + org.owasp
                            .esapi
                            .ESAPI
                            .encoder()
                            .encodeForHTML(new String(b, 0, size)));
```

Listing 2: Original lines in BenchmarkTest00001

to

```
response.getWriter()
    .println(
            "The beginning of file: '"
                    + org.owasp.esapi.ESAPI.encoder()
                        .encodeForHTML(fileName)
                    + "' is:\n\n"
                    + new String(b, 0, size));
```

Listing 3: Modified lines in BenchmarkTest00001

Burp Suite correctly identified the path traversal issues using the raw file contents. This confirmed that HTML encoding was the cause of the false negative.

The file write and file access variants are challenging to detect, and the checks are based on response size and content differences. Many file write variants are not detected because there is no discernible difference between the response to the base input and another arbitrary input, and a file does not need to exist to be written to. An example of such an undetected test case is `BenchmarkTest00002`. The file write test cases that were detected performed a file read (existence check) before performing the file write (e.g., `BenchmarkTest00045`). Without prior knowledge of the directory structure used on a web server, it can be challenging for a scanner to detect whether a file was written due to a path traversal payload. It needs to access the written file to confirm that a file write occurred.

Two types of file access test cases also remained undetected. The first category included the test cases where a file path was passed directly to `java.io.File` (e.g., `BenchmarkTest00040`), and the second category included test cases where a suffix (i.e., `/Test.txt`) was added to the end of the file path. We are unaware of techniques that could be used to remove the suffix. Because the OWASP Benchmark does not include proof of concepts for exploitable vulnerabilities, we cannot confirm that the second category allows an attacker to read files that are not named `Test.txt`.

Windows results

The false negatives and true positives for the Linux and Windows scans were the same, but on Windows, 18 of 50 findings were marked as file path traversal instead of file path manipulation and assigned a High severity score instead of an Informative severity score. This is because the file contents Burp Suite looks for in its Windows-based checks are not modified by HTML encoding.

3.3.4 SQL injection

Linux results

One of the signs of an SQL injection is the generation of a syntax error because of characters that break the query syntax [51]. Some of those errors start with "unterminated quoted string at or near" [52] or "unterminated quoted string literal" [53]. When we scanned the pages vulnerable to command injection, a similar error message was generated by bash during the SQL injection checks:

```
sh: 1: Syntax error: Unterminated quoted string
```

Hence, the command injection was mistakenly marked as an SQL injection.
187 of 272 SQL injections were detected.

We soon noticed that the false negatives could be categorized the same way as in Section 3.3.2. Namely, the crawler did not detect dynamically set headers or parameters.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.3.5 XPath injection

Linux results

8 of 15 XPath injections were detected. No false positives were generated. Each XPATH injection vulnerability was caused by the same query: `String expression = "/Employees/Employee[@emplid='" + bar + "']";`. The false negatives resulted from the crawler issues from Section 3.3.2.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.3.6 XSS (Cross-Site Scripting)

Linux results

186 of 246 XSS vulnerabilities were detected. There were no false positives. Some Referer-based XSS issues might only be exploitable in older browsers because the Referer header is URL-encoded in recent browser versions [54]. Still, most test cases of the OWASP Benchmark URL decode the value in the Referer header before reflecting it in the response.

In addition to the crawler-related false negatives we encountered in Section 3.3.2 and Section 3.3.4, two other false negatives were interesting.

In BenchmarkTest00146, the last character of the Referer header is replaced by a Z before reflecting it.

Similarly, in BenchmarkTest00291, the last character of the Referer header is removed. This behavior resembles that of a broken XSS filter that uses a non-recursive `replace` call instead of a recursive `replaceAll` call to remove the smaller than and bigger than characters characterizing HTML tags [55]. It seems like Burp Scanner's developers avoided including certain types of filter bypasses to balance the thoroughness and efficiency of a scan.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.3.7 Secure Cookie Flag

Linux results

22 of 36 missing Secure Cookie Flag issues were detected. Ten false negatives resulted from the crawler issues described in Section 3.3.2. The remaining four (i.e., BenchmarkTest00903, BenchmarkTest01789, BenchmarkTest02709, BenchmarkTest02710) were because of a length check. The value of `SomeCookie` is set to a concise value for each of those test cases (i.e., `bar`) and is unlikely to contain sensitive information. Hence, to avoid generating uninteresting issues, Burp Suite did not report this instance.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.4 Reducing the false negatives of the OWASP Benchmark scan using Burp extensions

In Section 3.3, we discovered that most false negatives were because of issues in Burp's crawler, which cannot be worked around using BChecks. Hence, in this section, we will only try to solve the false negatives in the command injection (see Section 3.3.1), path traversal (see Section 3.3.3), and cross-site scripting checks (see Section 3.3.6).

3.4.1 Command injection

Because of the problem from Section 2.1.1, we could not perform a comprehensive check for command injections using a regular Burp Suite Professional scan. However, this problem was caused by a particular time-based payload, and we were curious whether the SHELLING extension would allow us to conduct a full scan without the timeout issues. Unfortunately, the extension output showed a warning at `Extensions -> Installed -> SHELLING -> Output`:

```
Warning: the PREFIX_HOLDER" /D file://PAYLOAD_MARK.BURP_COLLAB_DOMAIN/PAYLOAD_MARK
payload does not contain the nslookup command and/or the
PAYLOAD_MARK.BURP_COLLAB_DOMAIN
argument. Argument separator could not be determined.
```

Listing 4: Warning shown by the SHELLING extension

No requests were sent after this message appeared. We later discovered that this issue could be solved by changing the Target OS option to Nix and using a custom character range of lowercase characters when brute-forcing argument flags.

After the SHELLING extension, we attempted to use the Burp Bounty Professional extension. By default, Burp Bounty Professional uses gf [56] patterns to scan requests with interesting parameter names. Hence, you must enable the "Scan all requests with all Profiles" rule to scan every parameter. Burp Bounty Professional did not solve the false negatives we discussed in Section 3.3.1.

Finally, we used the Sentinel extension to check whether it could solve the false negatives. Sentinel used similar command injection payloads to Burp Suite but added a

payload for the Shellshock vulnerability [57]. Unfortunately, it did not include payloads where a command is sent as-is and did not solve the false negatives.

3.4.2 Path traversal

Burp Bounty Professional is the only extension from Section 2.5 that supports additional checks for path traversal vulnerabilities. Unfortunately, Burp Bounty did not solve the false negatives that we encountered.

3.4.3 XSS (Cross-Site Scripting)

Burp Bounty Professional solved the false negatives we encountered in Section 3.3.6 by adding extra characters (such as a dot) to the end of its XSS payloads or by looking for the beginning of a reflected XSS payload (such as the starting `<script>` tag) instead of the entire payload. The Sentinel extension does not support scanning headers, so it cannot find the XSS vulnerabilities in the Referer header processing.

3.5 Categorizing the false negatives and positives of the Reinforced-Wavsep scan

This section discusses the outcomes of the Reinforced-Wavsep scan. Because of the difficulties we experienced with our initial scan of the OWASP Benchmark, we limited ourselves to targeted scans of specific sections from the start instead of trying to conduct a full scan of every endpoint and every vulnerability type. We also encountered another crawler issue where forms were not submitted unless we only crawled a small list of endpoints per crawl (around 1-5 endpoints). As in Section 3.3, we conducted separate crawls and audits using the "Crawl" and "Audit selected items" options.

Because of bugs in the local file inclusion and command injection test cases, the "number found" results for those vulnerability types are misleading. Furthermore, Reinforced-Wavsep includes JSON and XML insertion points, but they are incomplete and not counted towards the vulnerable test cases. Like the OWASP Benchmark, Reinforced-Wavsep contains specific false positive test cases. Hence, each vulnerability category has a fixed number of possible false positive results when benchmarks are performed, and false positive results generated for other endpoints are not considered as part of the benchmark.

Finally, the Reinforced-Wavsep repository states that there are 158 vulnerable SQL injection test cases and 748 vulnerable local file inclusion test cases, but the original WAVSEP documentation states that there are 816 vulnerable LFI test cases, and we counted 161 vulnerable SQL injection test cases in the updated benchmark [58]. There are 12 pages for the XXE test cases, but only two vulnerable endpoints are referenced by those pages. We excluded the Obsolete Files vulnerability type because Burp Scanner does not contain a relevant scan check.

Vulnerability	Number found (Linux)	Number found (Windows)
DOM XSS	4 of 4	4 of 4
Local File Inclusion	756 of 816	698 of 816
Remote File Inclusion	108 of 108	108 of 108
Command Injection	8 of 120	112 of 120
Reflected XSS	98 of 116	98 of 116
SQL injection	160 of 161	160 of 161
Unvalidated Redirect	60 of 60	60 of 60
XXE	1 of 2	1 of 2

Table 2: Number of findings Reinforced-Wavsep scan

Vulnerability	False positives (Linux)	False positives (Windows)
DOM XSS	N/A	N/A
Local File Inclusion	2 of 8	2 of 8
Remote File Inclusion	1 of 6	1 of 6
Command Injection	N/A	N/A
Reflected XSS	0 of 7	0 of 7
SQL injection	3 of 10	3 of 10
Unvalidated Redirect	0 of 9	0 of 9
XXE	N/A	N/A

Table 3: Number of false positives Reinforced-Wavsep scan

3.5.1 DOM XSS

Linux results

4 of 4 DOM XSS vulnerabilities were detected. However, two were marked as JavaScript injection (i.e., Case01-InjectionDirectlyInToDomXssSinkEval.jsp and Case03-InjectionInToVariableBeingAssignedToDomXssSinkEval.jsp) and the other two were marked as open redirects (i.e., Case02-InjectionDirectlyInToDomXssSinkLocation.jsp and Case04-InjectionInToVariableBeingAssignedToDomXssSinkLocation.jsp). The last two test cases could have been reported as XSS vulnerabilities because `javascript` URIs were accepted as part of the input. This possibility is also mentioned in the open redirect issue description [59].

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.5.2 Local File Inclusion

Linux results

756 of 816 Local File Inclusion vulnerabilities were detected. The false negative cases were:

- Case27-LFI-ContextStream-FilenameContext-Unrestricted-OSPath-DefaultEmptyInput-AnyPathReq-Read.jsp

- Case37-LFI-FileClass-FilenameContext-SlashTraversalValidation-OSPath-DefaultFullInput-AnyPathReq-Read.jsp
- Case45-LFI-ContextStream-FilenameContext-SlashTraversalValidation-OSPath-DefaultFullInput-AnyPathReq-Read.jsp
- Case53-LFI-FileClass-FilenameContext-SlashTraversalRemoval-OSPath-DefaultFullInput-AnyPathReq-Read.jsp
- Case61-LFI-ContextStream-FilenameContext-SlashTraversalRemoval-OSPath-DefaultFullInput-AnyPathReq-Read.jsp

According to the descriptions of Case27, Case45, and Case61, they are implemented to prevent accessing files outside of the application directories. Thus, exploiting them requires ad hoc payloads not included in Burp Scanner. Case37, Case45, Case53, and Case61 also implement validation logic that blocks payloads with a forward slash character. This seems to be a bug in WAVSEP that has not been fixed since it was reported in 2012 [60], and it prevents the proof of concept exploits included in WAVSEP. Therefore, the number of false negatives is much lower and seems limited to Case27 and its variants.

Case01-LFI-FalsePositive-Forward-TextHtmlValidResponse-FilenameContext-Unrestricted-OSPath-DefaultRelativeInput-NoPathReq-Read.jsp and Case07-LFI-FalsePositive-FileClass-TextHtmlValidResponse-FilenameContext-EnumerationResponseOnly-OSPath-DefaultRelativeInput-NoPathReq-Read.jsp of the false positive tests allow file enumeration attacks, and Burp Scanner reported this.

Windows results

698 of 816 Local File Inclusion vulnerabilities were detected. However, this is partly because of a WAVSEP bug reported in 2012 [61].

The following test cases are broken due to an incorrect path concatenation operation:

- Case10-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultFullInput-NoPathReq-Read.jsp
- Case12-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultInvalidInput-NoPathReq-Read.jsp
- Case14-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultEmptyInput-NoPathReq-Read.jsp
- Case16-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultFullInput-SlashPathReq-Read.jsp
- Case18-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultInvalidInput-SlashPathReq-Read.jsp
- Case20-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultEmptyInput-SlashPathReq-Read.jsp
- Case24-LFI-FileClass-FilenameContext-Unrestricted-FileDirective-DefaultFullInput-BackslashPathReq-Read.jsp

The following test cases were not detected because of the ad hoc payloads required:

- Case27-LFI-ContextStream-FilenameContext-Unrestricted-OSPath-DefaultEmptyInput-AnyPathReq-Read.jsp

- Case45-LFI-ContextStream-FilenameContext-SlashTraversalValidation-OSPath-DefaultFullInput-AnyPathReq-Read.jsp
- Case61-LFI-ContextStream-FilenameContext-SlashTraversalRemoval-OSPath-DefaultFullInput-AnyPathReq-Read.jsp

There was no difference between the false positives of the Linux and Windows scans.

3.5.3 Remote File Inclusion

Linux results

108 of 108 Remote File Inclusion vulnerabilities were detected. There was one false positive (i.e., Case05-RFI-FalsePositive-UrlClass-TextHtmlValidResponse-FilenameContext-EnumerationResponseOnly-OSPath-DefaultRelativeInput-NoPathReq-Read.jsp) because Burp Scanner does not include a scan check for Remote File Inclusion vulnerabilities specifically. Instead, we used the Out-of-band resource load (HTTP) issue to check whether Burp Scanner could help identify the associated HTTP requests sent by the server.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.5.4 Command Injection

Linux results

8 of 120 Command Injection vulnerabilities were detected. Upon further review, the command injection test cases did not work correctly on Linux. The command injection test cases included a pipe character | in the used prefixes or postfixes, and it is inadvertently interpreted as an argument to the `cat` command because of the way `Runtime.getRuntime().exec()` works on Linux. Hence, the test cases represent argument injection vulnerabilities targeting the `cat` command instead of command injection vulnerabilities. The authors of Reinforced-Wavsep seem to be aware of this issue [62]. The following test cases:

- Case1-OSCmdInjection-GenericOS-InitialCommandContext-SimpleStatement-DefaultOsCommandInputNoValidation.jsp
- Four variants of Case2-OSCmdInjection-GenericOS-InitialCommandContext-SimpleStatement-DefaultEmptyInput-NoValidation.jsp.
- Case3-OSCmdInjection-GenericOS-InitialCommandContext-SimpleStatement-DefaulInvalidInput-NoValidation.jsp
- Case4-OSCmdInjection-GenericOS-InitialCommandContext-SimpleStatement-DefaultRelativeOsCommandInput-NoValidation.jsp
- Case29-Command-Injection-SimpleCase.jsp
- Case30-Command-Injection-Blind-TimeDelay.jsp
- Case31-Command-injection-Blind-OutputRedirection.jsp
- Case32-Command-injection-Blind-OutOfBand.jsp

used a different injection context and were affected by a command injection vulnerability on Linux. Case1, Case3, and Case4 were not detected, and this could be remediated using the BCheck we created in Section 3.11.1 or by setting the `target` parameter to an empty value and adding a manual insertion point. Hence, in practice, three test cases resulted in false negatives. There are no false positive test cases for command injection in Reinforced-Wavsep.

Windows results

112 of 120 Command Injection vulnerabilities were detected. Eight JSP endpoints (corresponding to Case29 to Case32) use a prefix with the Bourne shell. Hence, there were no false negatives, and every command injection was detected when Reinforced-Wavsep was running on Windows. There are no false positive test cases for command injection in Reinforced-Wavsep.

3.5.5 Reflected XSS

Linux results

98 of 116 reflected XSS vulnerabilities were detected. There were three types of false negatives. The first category, which caused four false negatives, resulted from the presence of a CSRF token in Case03-Tag2HtmlPageScope-ConstantAntiCSRFToken.jsp and Case04-Tag2HtmlPageScope-ChangingAntiCSRFToken.jsp. We could remediate the false negative for Case03 by performing a "Crawl and audit" scan instead of separate "Crawl" and "Audit selected items" scans because the constant CSRF token was updated to the correct value ("Crawl and audit" scans are recommended for this reason [63]). Case04 could be remediated using macros, as discussed at [64].

The second category, which caused two false negatives, involved a "scriptless injection" in a base tag. This could be worked around by selecting the Link manipulation issue, which remediates the false negatives because Case06-ScriptlessInjectionInBaseTagHrefAttribute.jsp does not allow injecting JavaScript directly, and the proof-of-concept exploit also did not result in JavaScript execution.

The third category, which caused twelve false negatives, involved multiple variants of CSS injection:

- Case08-InjectionInToCssSelector.jsp
- Case09-InjectionInToCssSelectorAttributeName.jsp
- Case10-InjectionInToCssProperty.jsp
- Case11-InjectionInToCssPropertyValue.jsp
- Case12-Tag2CSSQuotedStringScope.jsp
- Case13-Tag2CSSCommentScope.jsp

Interestingly, Burp Scanner's CSS injection check did not report these vulnerabilities.

There were two false positives (Case12-Tag2CSSQuotedStringScope.jsp and Case13-Tag2CSSCommentScope.jsp) because the CSS injections were marked as reflected XSS vulnerabilities due to injected unescaped `script` tags within the CSS stylesheets.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.5.6 SQL injection

Linux results

160 of 161 SQL injection vulnerabilities were detected. There was one false negative, Case04-InjectionInUpdate-String-CommandInjection-WithDifferent200Responses.jsp, related to an UPDATE query. Additionally, there were three false positives:

- Case02-FalsePositiveInjectionInLogin-PsAndIv-500SyntaxErrorOnIvFailure.jsp
- Case04-FalsePositiveInjectionInLogin-PsAndIv-200SyntaxErrorOnIvFailure.jsp
- Case06-FalsePositiveInjectionInLogin-HoneyPotNoSQL-Fake500SyntaxErrorOnIvFailure.jsp

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.5.7 Unvalidated Redirect

Linux results

60 of 60 unvalidated redirect vulnerabilities were detected. No false positives were generated.

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.5.8 XXE

Linux results

1 of 2 XXE vulnerabilities were detected. This was related to the crawler issues discussed in Section 3.3.2 (i.e., a dynamically generated HTTP request using `XMLHttpRequest` was not included in the crawl).

Windows results

There was no difference between the Linux and Windows scan results for this vulnerability type.

3.6 Reducing the false negatives of the Reinforced-Wavsep scan using Burp extensions

In Section 3.5, we identified three vulnerability types with false negatives unrelated to crawler issues. They included CSS injection (from the Reflected XSS category), command injection, and SQL injection.

3.6.1 CSS injection

As far as we know, no Burp extensions improve Burp Scanner's CSS injection checks.

3.6.2 Command injection

As discussed in Section 3.5.4, most command injection test cases were broken on Linux. Hence, we only looked at whether the extensions could remediate the test cases with a different injection context (i.e., the functional test cases). We used the Burp Bounty Professional extension in an attempt to remediate the false negatives. Unfortunately, Reinforced-Wavsep repeatedly crashed during those scans, and we had to restart the application manually. This caused the scan results to become unreliable. Burp Bounty Professional resulted in false positives because it used a `cat /etc/passwd` payload that identified the argument injection as a command injection. Furthermore, it did not mitigate the false negatives. After that, we tried to use the SHELLING extension. However, the SHELLING extension took multiple days to run, resulting in many network errors. Finally, we used the Sentinel extension. The Sentinel extension did also not mitigate the false negatives.

3.6.3 SQL injection

The SQLiPy extension remediated the false negative using boolean-based and time-based payloads. SLEEPTIME is the default parameter sqlmap uses to determine the time that is passed to the SLEEP function:

```
1' AND 3745=(SELECT (CASE WHEN (3745=3745) THEN 3745 ELSE (SELECT 8936 UNION SELECT 4179) END))-- aONg
1' AND (SELECT 3460 FROM (SELECT(SLEEP([SLEEPTIME])))mQhE)-- XKwL
```

Listing 5: Payloads used by SQLiPy to detect vulnerable UPDATE query

3.7 Categorizing the false negatives and positives of the Firing Range scan

In this section, we discuss the outcomes of the Firing Range scan. Four test cases were broken on the public instance of the Firing Range and were excluded from the results table:

- /escape/serverside/encodeUrl/href
- /escape/serverside/escapeHtml/a
- /escape/serverside/escapeHtml/href
- /reflected/url/a

We also excluded the test cases from Section 2.3 that we believe to be unexploitable, and the `tags` test cases because they used an allow list that only accepted one tag and attribute and a generic reflected XSS check cannot detect such a situation. Because of the manual triage we had to perform for this benchmark (determining the exploitability of the vulnerabilities), the results are less reliable than those of the previous sections.

Vulnerability	Number found
DOM XSS	59 of 98
Clickjacking	1 of 1
CORS	7 of 7
Reflected XSS	61 of 67
Client-side template injection (AngularJS)	12 of 18
Mixed content	1 of 1
Unvalidated Redirect	2 of 3
HTTP Strict Transport Security	2 of 5
Vulnerable JavaScript libraries	1 of 1

Table 4: Number of findings Firing Range scan

3.7.1 DOM XSS

59 of 98 DOM XSS vulnerabilities were detected. Multiple DOM XSS vulnerabilities were not detected, such as vulnerable postMessage handlers and form submissions. Some of the DOM XSS vulnerabilities were not directly exploitable without specific conditions being met, such as the ones that insecurely used items from `localStorage` or `sessionStorage`. Others involved less commonly encountered sinks, such as an SVG tag's `xlink:href` attribute. Nevertheless, this result indicates that the DOM XSS scan checks require numerous improvements. Burp Scanner contains scan checks related to web messages and storage manipulation. Still, it does not cover web message event listeners unless a `postMessage` call is present on the same page and does not cover storage sources being passed to sinks [65, 66]. The DOM Invader tool can be used to manually test for DOM XSS, DOM clobbering, prototype pollution, and web message vulnerabilities, but this is a component separate from the scanner [67].

3.7.2 Clickjacking

1 of 1 clickjacking vulnerability was detected (i.e., `clickjacking_csp_no_frame_ancestors`). There were two clickjacking test cases. However, the second test case `clickjacking_xfo_allowall` did not include an `X-Frame-Options` header, and neither test case included anything to click on.

3.7.3 CORS

7 of 7 CORS vulnerabilities were detected.

3.7.4 Reflected XSS

61 of 67 reflected XSS vulnerabilities were detected. Four of the false negatives were related to the CSS injection checks identified in Section 3.5.5. The other two false negatives were related to vulnerable pages using Adobe Flash Player. Note that Flash Player is no longer supported since 2021 [43].

3.7.5 Client-side template injection (AngularJS)

12 of 18 client-side template injection vulnerabilities were detected. Each false negative required additional context from the JavaScript on the vulnerable pages, but Burp Scanner does not contain static analysis checks for client-side template injection.

3.7.6 Mixed content

1 of 1 mixed content vulnerability was detected. Modern browsers often block mixed content by default [68].

3.7.7 Unvalidated Redirect

2 of 3 unvalidated redirect vulnerabilities were detected. The false negative `/redirect/meta` included user-provided content in the `content` attribute of a `meta` refresh tag.

3.7.8 HTTP Strict Transport Security

2 of 5 HTTP Strict Transport Security vulnerabilities were detected.
The `hsts_includesubdomains_missing` test case is only relevant if a domain has subdomains and does not indicate a vulnerability by itself. The `hsts_preload_missing` test case could be exploited in some scenarios if an attacker can intercept a first-time connection [69]. The `hsts_max_age_too_low` test case indicates a weak HSTS configuration and would ideally be included in the scan checks. However, what value should be used for the `max-age` property is subjective and depends on the application's security requirements.

3.7.9 Vulnerable JavaScript libraries

1 of 1 vulnerable JavaScript libraries test case was detected. Burp Scanner also detected the outdated versions of Angular used for the client-side template injection test cases.

3.8 Reducing the false negatives of the Firing Range scan using Burp extensions

In Section 3.7, we identified five vulnerability types with false negatives. They included DOM XSS, CSS injection (from the Reflected XSS category), client-side template injection, unvalidated redirect, and HTTP Strict Transport Security.

3.8.1 DOM XSS

We attempted to use the Burp DOM Scanner to mitigate the false negatives. Unfortunately, the extension seemed to stop scanning after the 27th payload, regardless of the browser settings. We also used the Additional Scanner Checks extension, which mitigated one false negative and added four false positives. Finally, we tried to use the ESLinter extension, but it does not seem compatible with recent versions of Burp Suite Professional.

3.8.2 CSS injection

See Section 3.6.1.

3.8.3 Client-side template injection

We could not find any scanning extensions that search for client-side template injection using static analysis techniques.

3.8.4 Unvalidated redirect

We could not find scanning extensions to search for unvalidated redirects except for Burp Bounty Professional. Burp Bounty did not mitigate the false negative.

3.8.5 HTTP Strict Transport Security

The Additional Scanner Checks extension includes a Strict Transport Security check that supports configuring a minimum acceptable value for `max-age` [26]. This mitigates the false negative from Section 3.7.8. Two existing BChecks can be used to mitigate the `includeSubDomains` and `preload` flag false negatives [70, 71].

3.9 Categorizing the false negatives and positives of the websitesVulnerableToSSTI scan

The websitesVulnerableToSSTI benchmark contains two types of test cases: SSTI test cases and code injection test cases. Many of the SSTI test cases are also vulnerable to reflected XSS. There are 34 SSTI endpoints and 17 code injection endpoints.

Vulnerability	Number found
SSTI	27 of 34
Code injection (JavaScript)	2 of 5
Code injection (PHP)	4 of 4
Code injection (Python)	7 of 7
Code injection (Ruby)	1 of 1

Table 5: Number of findings websitesVulnerableToSSTI scan

3.9.1 SSTI

27 of 34 SSTI vulnerabilities were found. Four types of false negatives were found. The first type was caused by unsupported template engines. SSTI payloads are heavily dependent on the underlying template engine, and it is likely not feasible to support all of them. Burp Scanner missed a Golang SSTI where the `html/template` library was used [72], the `doT.js` template engine [73] and the `Dust.js` template engine [74]. The second type was caused by two endpoints that required a slightly different syntax (i.e., {{`= template code`}} instead of {{ `template code` }}) that is similar to the syntax used by `doT.js`. The third false negative was caused by a blind SSTI in code using the mako template engine. The fourth false negative was caused by a second-order SSTI vulnerability in code using the mako template engine. No false positives were generated.

3.9.2 Code injection

14 of 17 code injection vulnerabilities were found. During our scans, the JavaScript injection check seemed to give inconsistent results. We worked around this using a manual insertion point, setting the expression parameter to a valid base expression (the digit 1). No false positives were generated.

3.10 Reducing the false negatives of the websitesVulnerableToSSTI scan using Burp extensions

To mitigate the false negatives, we used the Blind SSTI Scanner extension, using the curl command, enabling code context escaping and enabling collaborator polling. This mitigated three of the seven false negatives (`/javascript/dot`, `/python-flask-based/outputNotAccessible/`, and `/python-flask-based/resultOtherPage/`). We also attempted to use the tplmap extension. Unfortunately, it no longer works on recent Burp Suite Professional versions. Burp Bounty Professional and Sentinel did not mitigate the false negatives.

3.11 Creating BChecks to improve benchmark performance

In the previous subsections, we identified detection gaps in Burp Scanner, Burp Bounty Professional, and the scanner extensions on the BApp store. In this section, we will develop BChecks to remediate those false negatives. However, due to the limitations described in Section 4.1, we did not implement BChecks for every false negative we identified. We also did not implement a BCheck for the code injection from Section 3.9.2 because a scanner usually requires valid or functional base requests to work correctly, and the issue can be worked around by providing such a base request. We used the thorough configuration from Section 2.1.2 for each BCheck-only scan we performed. To scan the OWASP Benchmark, we limited the number of concurrent requests to one because of the stability issues of the benchmark application. For the other benchmarks, we used ten concurrent requests. The number of additional requests and the increase in runtime caused by a BCheck partially depend on the endpoints to which it is applied. This explains the significant differences between those values in the subsections below.

3.11.1 Remediating the command injection false negatives

We created a BCheck to address command injection false negatives in the OWASP Benchmark and Reinforced-Wavsep on Linux. The BCheck used four payloads from [75]. After implementing the workaround from Section 3.4.1 (i.e., setting the target OS and specifying a custom character range when brute-forcing argument flags), we found that the SHELLING extension could also mitigate these false negatives. Additionally, the BCheck published at [76] also mitigated the false negatives but took multiple days to run and needed to be modified to process the HTML-encoded responses. It also did not support Windows systems and required more than three million additional requests.

The first payload checks for reflected inputs based on the `echo` command, the second uses `ping` to trigger DNS lookups, the third applies a filter bypass method, and the fourth prefixes the `ping` command with bash. The last two payloads only work against Linux systems.

We will now look at each quality measurement from Section 2.6. Our scan required 41163 additional requests on the OWASP Benchmark. However, only the first payload was needed to mitigate the false negatives. Our BCheck significantly increased the runtime by 9 hours. Our scan required 9233 additional requests on Reinforced-Wavsep and increased the runtime by 2 hours. We used multiple payloads for better generalizability, but assessing its adherence to this principle is challenging as we only tested it on the OWASP Benchmark and Reinforced-Wavsep.

Our check did not yield any false positives and successfully addressed all false negatives, resulting in improved accuracy. Note that using a private Collaborator server when using this BCheck is recommended because the availability of the public Collaborator server is not guaranteed [77].

3.11.2 Remediating the path traversal false negatives

As discussed in Section 3.3.3, the reason Burp Suite reported file path manipulation vulnerabilities instead of path traversal vulnerabilities for specific endpoints was that it did not decode the HTTP responses, which resulted in no path traversal issues generated on Linux. To mitigate this, we added a passive BCheck `path-traversal-passive.bcheck` [17] that looks for the HTML-encoded contents of `/etc/passwd`. After adding this check and using it to scan the requests sent by Burp Scanner during the path traversal audit, Burp Suite correctly reported the issues.

Because this was a passive BCheck, there were no additional requests. The check took around an hour after modifying our initial scan to only include the URL parameter values, Body parameter values, and Cookie parameter values insertion point types. Before that change, the check took multiple days to run. We suspect that the reason for this is that passive BChecks were not intended to be used to analyze Scanner logs in this way. It is not generalizable because it searches for hardcoded strings in the response. A generalizable check would require rewriting existing scan checks to use various decoding operations (e.g., HTML entity decoding, base64 decoding, etc.) instead of only operating on raw response contents. After applying the BCheck, the accuracy improved, and the 18 test cases marked as path traversal on Windows were also correctly marked as path traversal on Linux.

3.11.3 Remediating the CSS injection false negatives

In Section 3.5.5 and Section 3.7.4, we identified false negatives related to the CSS injection checks. To mitigate this, we created a BCheck that uses a regular expression to look for a specific reflected string within the contents of a style tag. Though this is a practical approach, it does not demonstrate an exploitable vulnerability on its own. However, it does increase the generalizability of the check. We needed 190 additional requests, and the check took 3 seconds when using it against Firing Range. We needed 1192 additional requests, and the check took 7 seconds against Reinforced-Wavsep. Our check successfully mitigated the false negatives for the Reinforced-Wavsep and Firing Range test cases without introducing new false positives.

3.11.4 Remediating the unvalidated redirect false negatives

In Section 3.7.7, we identified a false negative related to unvalidated redirects using meta-refresh tags. Similar to the BCheck in Section 3.11.3, this BCheck does not demonstrate an exploitable vulnerability on its own but uses regular expressions to identify suspicious reflected inputs following the pattern associated with this vulnerability type. The new check resulted in 110 additional requests and took 20 seconds to run. It is not generalizable because it checks for a specific type of unvalidated redirect. The check successfully mitigated the false negative on the Firing Range test case without introducing new false positives.

3.11.5 Remediating the SSTI false negatives

The remaining four SSTI false negatives could be mitigated by adding a new payload for the missing template engines and a payload with the different syntax mentioned in Section 3.9.1. This resulted in 3549 additional requests and took 2 minutes to run. The check adding support for a new syntax could be generalizable to template engines that use that syntax.

4 Discussion

4.1 Limitations of BChecks

BChecks do not support time-based detections, making it challenging to implement scan checks for blind vulnerabilities. There is also no support for JSON parameters [78]. Additionally, we cannot perform computations based on variations between HTTP responses, which can cause issues when we try to implement boolean-based detections on endpoints that have reflected inputs. Finally, we cannot implement static analysis checks for JavaScript vulnerabilities using BChecks. We expect PortSwigger to address some of these issues in future versions of BChecks.

4.2 Improving the OWASP Benchmark

Several improvements were outlined by [79], which proposed requirements that must be satisfied to obtain an accurate benchmark.

Currently, the OWASP Benchmark contains test cases written in the Java programming language. While vulnerable web applications in other languages exist [80], they do not include the scoring feature. Therefore, any experiments using those applications require a manual review of the relevant metrics. This is related to requirement 2 from [79].

Another interesting point from [79] is that the benchmark does not offer "fixed" versions of its vulnerable endpoints. This makes it more challenging to determine whether a finding was generated because of a vulnerability or a false positive triggered by another property of the vulnerable endpoint. This might not be solved by the presence of unrelated secure endpoints in the benchmark.

Additionally, the OWASP Benchmark does not include many of the vulnerabilities that are commonly discovered today. Assuming that the trends found in bug bounty reports represent the trends within the technology industry, the OWASP Benchmark could be improved by including client-side vulnerabilities such as DOM XSS [81] or server-side vulnerabilities such as broken access controls [82]. Even if not many tools support a particular category of vulnerability, it might be helpful to include them to allow benchmark users to track tool support. One of the main contributors to the OWASP Benchmark discussed dropping the cryptographic categories that we excluded in Section 3.3 and adding vulnerability types such as XXE [83] [84]. We excluded the cryptographic categories because Burp Suite Professional does not include scan checks for them.

Furthermore, the scoring component only supports findings that are within the scope of the benchmark itself. Hence, if a scanner generates a false positive outside of this scope, the score assigned to the scanner will not be reduced. Conversely, when a scanner correctly identifies that our self-hosted OWASP Benchmark instance does not use a trusted TLS certificate, uses an outdated version of jQuery, or that none of the forms are protected by a CSRF token, it will not increase its score on the benchmark. Hence, the final score generated by the OWASP Benchmark is not entirely representative of the quality of a scanner. It also depends on what types of findings you consider valid vulnerabilities, which can be an area of debate, and this is exacerbated by the fact that the benchmark does not include proof of concept attacks to demonstrate vulnerabilities.

The OWASP Benchmark, in many ways, provides an ideal scanning environment. No web application firewalls block our payloads, the application architecture is simple, and every page is referenced somewhere within the application. We also do not need to deal with cross-site request forgery tokens [85], authenticating to the application, CAPTCHAs [86], network firewalls, and other issues that complicate scans and result in false negatives, such as second-order vulnerabilities [87].

Finally, many of the vulnerabilities in the OWASP Benchmark can be discovered using identical payloads. For example, most XSS vulnerabilities can be found using a classic `<script>alert("TEST");</script>` payload [88]. Most SQL injection vulnerabilities can be found by using incremental techniques such as first using a single quote and then using two single quotes [89], and there were no blind SQL injections [90] that required the use of different techniques. Every SQL injection attempt in the benchmark resulted in error messages displayed on the vulnerable page. Each SQL injection and command injection vulnerability could be detected without using time-based or out-of-band [30] techniques. A high-quality benchmark should include different injection contexts to test whether a scanner can identify many vulnerability variants instead of including copies of the same variants. This would also mitigate the issue of scanners being heavily penalized for one false negative with the same root cause across every test case.

4.3 The ad hoc nature of vulnerability checks

In Section 2.6.3, we discussed the generalizability requirement of the payloads used in BChecks. Additionally, in Section 4.2, we mentioned the OWASP Benchmark lacks many injection contexts or variants of the same vulnerability type.

Unfortunately, even if we could consider every variant, we would still have to consider environmental factors.

For example, a payload using out-of-band techniques [30] might fail to identify a vulnerability if a firewall rule is blocking outbound requests from the affected system, whereas other viable attack vectors could exist.

A command injection payload could fail if the binaries used in the payload cannot be run (e.g., due to a restricted shell) or do not exist on the scanned system. A path traversal payload could fail if the file we attempt to read does not exist or the web server cannot access that file. An SQL injection payload might fail because the injection context could be within an SQL query that uses DBMS-specific functionality (there are hundreds of DBMS systems [91]). An XSS payload might fail because of a broken filtering solution implemented by an application's developers (e.g., see Section 3.3.6).

Regardless of the vulnerability type we scan for, certain parts of an application might only be reachable if a feature is enabled in the application settings.

Numerous vulnerable applications can be used to test newly introduced scan checks [80], and we can also write examples of vulnerable code. Still, every payload makes assumptions about what that vulnerable code looks like.

The steps required to research a closed-source scanner include setting up virtual machines, installing vulnerable applications, correctly configuring the scanner, accounting for unstable applications that minimize the number of concurrent requests we can send, and time-consuming thorough scans. This limits the testing we can perform.

These issues pose research challenges, and it does not seem easy to measure the accuracy of a closed-source scanner without months of testing and horizontal or vertical scaling [92].

4.4 Vulnerability scans are part of a more extensive security process

In [93], the authors describe the management considerations associated with applying vulnerability scanners. For example, they describe the importance of proper communication between security and development teams and the need to align security and business objectives.

In [94], the authors discuss the usability concerns when asking developers to integrate security tools into their DevOps workflows.

Hence, it is essential to remember that the accuracy of a scanner is only one criterion used to decide whether to use it in an organization. A perfect vulnerability scanner would not immediately solve the larger vulnerability management problem and does not replace manual testing [95].

4.5 Note about sources

Throughout my thesis, I have included several references from the PortSwigger website. This might raise the question of whether these citations have impacted my research or if I have any professional affiliation with PortSwigger. While I am a customer of PortSwigger and hold a license for Burp Suite Professional, I would like to clarify that I am not associated with them in a professional capacity. The references I have incorporated were solely intended to provide insight into the concepts underlying vulnerability types and explain Burp Suite Professional's correct usage.

4.6 Unpredictable scan results and the importance of a correct scan configuration

As briefly mentioned in Section 2.1.1, we had to monitor for network errors to ensure that our scan did not miss vulnerabilities because of request timeouts instead of a lack of coverage in the scan engine. Some scanners do not include a feature that allows you to perform this type of monitoring, and they might obtain a lower benchmark score if only one scan is performed.

Other factors can influence the scan results. In Section 3.3, we noticed a significant difference between scans using multiple concurrent requests and one concurrent request. Our scans went from not identifying any LDAP injections to identifying 19 of the 27 LDAP injections, and we observed similar trends with other vulnerability categories.

Additionally, certain insertion points are not detected automatically and will not be scanned unless manually pointed out to Burp Scanner [96].

Finally, when working on Section 2.1.1, our Windows VMs were initially set to an incorrect timezone. This meant the "Cookie jar" was not updated because the cookies were already expired in that timezone, and vulnerabilities in the processing of the cookie values were missed [97].

Scan configurations can be unintuitive and strongly influence the outcome of our benchmarks. Though it is possible to compare the quality of a default configuration across multiple scanners, an adequately configured scanner provides higher quality results when applied by a vulnerability assessments expert.

4.7 Recommendations to improve Burp Scanner

4.7.1 Burp's crawler

We identified four areas of improvement to Burp's crawler:

1. JavaScript frameworks frequently use unique ways of referencing endpoints. Burp's crawler would benefit from supporting additional JavaScript frameworks and associated navigation mechanisms, as we noticed in Section 3.1. This area seems to be worked on [98]. Additionally, the handling of dynamically generated requests could be improved (see Section 3.3.2).

2. Instead of skipping the response content of redirect responses, the references included in such responses could be processed by Burp's crawler (see Section 3.2).

This could also make it easier to identify Execution After Redirect vulnerabilities [99].

3. Burp's crawler includes advanced options that can be used to disable certain optimizations, such as skipping form submissions if there are too many forms on a particular page. It would be beneficial if those options were added to the official documentation.

4. The crawl strategies could be renamed to emphasize the context in which they should be applied [100]. The current naming describes the speed of the crawl strategy but does not make it evident that certain crawl strategies only work for stateless websites instead of stateful websites, as described in the documentation.

4.7.2 Burp Scanner

We identified seven areas of improvement to Burp Scanner:

1. The CSS injection checks failed to identify multiple variants of reflected inputs between `style` tags [101, 102] (see Section 3.5.5 and 3.7.4). The checks should be updated to detect such variants.

2. Currently, the DOM XSS checks rely on sources and sinks to be present on the same page. Hence, a vulnerable postMessage listener remains undetected unless a call to postMessage occurs on the page containing that listener. This is unlikely to be the case because postMessage is used to send messages between different pages. Furthermore, no checks identify the insecure use of storage sources such as `localStorage` or `sessionStorage`. Finally, less commonly encountered sinks are not yet incorporated in the scan checks (see Section 3.7.1).

3. Some scan checks search for strings in an HTTP response. For example, the path traversal check targeting Linux systems looks for part of the content of the `/etc/passwd` file. Hence, if an application encodes the contents of that file before returning it in the response, the path traversal vulnerability could remain undetected. It is recommended to decode HTML entities, base64, and other encodings identified in HTTP responses during these scan checks so that encoded strings indicative of a vulnerability are also detected (see Section 3.3.3).

4. Currently, Burp Scanner limits itself to scanning the `User-Agent` and `Referer` header, as well as custom headers starting with `X-` [48]. This might cause it to miss vulnerable custom headers not starting with `X-`. Instead of only scanning custom headers starting with `X-`, scanning every uncommon header by default could be beneficial.

5. There were several false negatives because the OS command injection check did not include a regular command that does not use shell meta-characters [103] (see Section 3.3.1). Adding a regular command to the payload list would remediate those false negatives.

6. Open redirects in meta-refresh tags were not detected (see Section 3.7.7). We recommend enhancing the server-side open redirection checks to identify them [104, 105].

7. The SSTI checks could be enhanced to cover additional template engines. This would remediate most of the false negatives from Section 3.9.1.

5 Conclusion

In this thesis, we comprehensively evaluated Burp Suite's capabilities. Our benchmarking results led us to identify and implement several small improvements to Burp Scanner's existing scan checks. Future work could focus on fixing the broken test cases of existing web scanner benchmarks, implementing extensions to improve the detection of DOM XSS vulnerabilities, and evaluating and enhancing the effectiveness of scanning extensions that check for vulnerabilities not included in our selected benchmarks.

Our research confirmed that utilizing scanner extensions improves the coverage of existing scan checks. Furthermore, thorough scan configurations and monitoring scan logs to identify malformed base requests could also help improve scan coverage.

It is crucial to analyze and evaluate security tooling before use rather than relying on default installations and built-in scan checks. Even the most thorough scan setup may not uncover every application vulnerability. The popularity of the technology in use affects the availability of scan checks, and broken filter implementations or hardened environments can thwart scan checks.

Ideally, a dedicated security engineer should implement custom-tailored scan checks suited to their organization's environment. Moreover, an organization should adhere to each step of the secure development lifecycle if feasible [5].

References

[1] Wikipedia. *List of countries by number of Internet users*. URL: https://en.wikipedia.org/wiki/List_of_countries_by_number_of_Internet_users (visited on 05/16/2024).

[2] International Telecommunication Union. *Measuring digital development: Facts and Figures 2023*. 2023. URL: https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx.

[3] Robert Preston. *Web Developer vs. Software Developer: What's the Difference?* June 2023. URL: https://www.indeed.com/career-advice/finding-a-job/web-developer-vs-software-developer.

[4] Koen van Gelder. *Share of small and medium-sized enterprises that have a website in the European Union from 2012 to 2023*. Dec. 2023. URL: https://www.statista.com/statistics/910088/smes-in-europe-that-have-a-website/.

[5] Microsoft. *What are the Microsoft SDL practices?* URL: https://www.microsoft.com/en-us/securityengineering/sdl/practices (visited on 05/16/2024).

[6] SoftwareTestingHelp. *Differences Between SAST, DAST, IAST, And RASP*. June 2023. URL: https://www.softwaretestinghelp.com/differences-between-sast-dast-iast-and-rasp/.

[7] PortSwigger. *Burp Suite Professional*. 2023. URL: https://portswigger.net/burp/pro.

[8] PortSwigger. *BApp Store*. 2023. URL: https://portswigger.net/bappstore.

[9] PortSwigger. *BCheck definitions*. 2023. URL: https://portswigger.net/burp/documentation/scanner/bchecks.

[10] Google. *Security Crawl Maze*. Feb. 2024. URL: https://github.com/google/security-crawl-maze.

[11] Bedirhan Urgun and Andres Riancho. *Web Input Vector Extractor Teaser*. Jan. 2022. URL: https://github.com/bedirhan/wivet.

[12] chuckatsf. *OWASP Broken Web Applications Project*. Sept. 2016. URL: https://sourceforge.net/projects/owaspbwa/.

[13] OWASP. *The OWASP Benchmark Project*. URL: https://github.com/OWASP-Benchmark/BenchmarkJava (visited on 02/02/2024).

[14] OWASP. *OWASP Benchmark wiki page*. 2023. URL: https://owasp.org/www-project-benchmark/.

[15] Docker. *Install Docker Engine on Ubuntu*. Jan. 2024. URL: https://docs.docker.com/engine/install/ubuntu/.

[16] PortSwigger. *Download and install*. Mar. 2024. URL: https://portswigger.net/burp/documentation/desktop/getting-started/download-and-install.

[17] Maurice Dibbets. *Benchmark results*. July 2024. URL: https://github.com/oxygen005/evaluation-results-thesis.

[18] Luigi Urbano, Gaetano Perrone, and Simon Pietro Romano. "Reinforced WAVSEP: a Benchmarking Platform for Web Application Vulnerability Scanners". In: *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE, July 2022. DOI: 10.1109/icecet55527.2022.9872956.

[19] Diogo Silva. *Vulnerable websites*. Oct. 2022. URL: https://github.com/DiogoMRSilva/websitesVulnerableToSSTI.

[20] James Kettle. *Active Scan++*. URL: https://portswigger.net/bappstore/3123d5b5f25c4128894d97ea1acc4976 (visited on 02/12/2024).

[21] James Kettle. *Backslash Powered Scanning: hunting unknown vulnerability classes*. Nov. 2016. URL: https://portswigger.net/research/backslash-powered-scanning-hunting-unknown-vulnerability-classes.

[22] Nicolas DaLomba. *CMS Scanner*. URL: https://github.com/portswigger/cms-scan (visited on 02/12/2024).

[23] Julian Horoszkiewicz. *SHELLING - a comprehensive OS command injection payload generator*. URL: https://github.com/PortSwigger/command-injection-attacker (visited on 02/12/2024).

[24] Dobin Rutishauser. *Burp Sentinel*. URL: https://github.com/portswigger/sentinel (visited on 02/12/2024).

[25] Josh Berry. *SQLiPy Sqlmap Integration*. URL: https://github.com/PortSwigger/sqli-py (visited on 02/12/2024).

[26] Paul Johnston. *Additional Scanner Checks*. Dec. 2018. URL: https://github.com/PortSwigger/additional-scanner-checks.

[27] Filippo Cavallarin. *Burp DOM Scanner*. Apr. 2023. URL: https://github.com/fcavallarin/burp-dom-scanner.

[28] Parsia Hakimian. *Manual JavaScript Linting is a Bug*. Mar. 2021. URL: https://github.com/parsiya/eslinter/.

[29] efecankaya. *Blind SSTI Scanner for Burp Suite*. Feb. 2024. URL: https://github.com/efecankaya/BlindSSTIScanner.

[30] PortSwigger. *Out-of-band application security testing (OAST)*. URL: https://portswigger.net/burp/application-security-testing/oast (visited on 02/09/2024).

[31] epinna. *Tplmap*. Feb. 2022. URL: https://github.com/epinna/tplmap.

[32] bountysecurity.ai. *Burp Bounty*. URL: https://burpbounty.net/ (visited on 02/12/2024).

[33] PortSwigger. *Site map*. Jan. 2024. URL: https://portswigger.net/burp/documentation/desktop/tools/target/site-map.

[34] Colin Watson and Tin Zaw. *OWASP Automated Threat Handbook Web Applications*. Feb. 2018, p. 58. ISBN: 978-1-329-42709-9. URL: https://owasp.org/www-pdf-archive/Automated-threat-handbook.pdf.

[35] PortSwigger. *Audit items*. May 2024. URL: https://portswigger.net/burp/documentation/desktop/automated-scanning/results/audit-items.

[36] Hannes Holm et al. *A quantitative evaluation of vulnerability scanning*. 2011. URL: https://www.diva-portal.org/smash/get/diva2:545791/FULLTEXT01.pdf.

[37] Hannah, PortSwigger Agent. *Issue type Certain , confirm and tentative(what does this mean)*. URL: https://forum.portswigger.net/thread/issue-type-certain-confirm-and-tentative-what-does-this-mean-b01d4f6d (visited on 02/08/2024).

[38] PortSwigger. *Burp Collaborator*. Jan. 2024. URL: https://portswigger.net/burp/documentation/collaborator.

[39] PortSwigger. *Crawling: Crawling volatile content*. Mar. 2024. URL: https://portswigger.net/burp/documentation/scanner/crawling.

[40] PortSwigger. *BCheck definition reference*. URL: https://portswigger.net/burp/documentation/scanner/bchecks/bcheck-definition-reference (visited on 02/08/2024).

[41] MDN contributors. *Link*. Feb. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link.

[42] MDN contributors. *Responsive images*. May 2024. URL: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images.

[43] Adobe. *Adobe Flash Player EOL General Information Page*. Jan. 2021. URL: https://www.adobe.com/products/flashplayer/end-of-life.html.

[44] PortSwigger Agent Dominyque. *I'm getting errors while using Burpsuite against the OWASP benchmark.* June 2023. URL: https://forum.portswigger.net/thread/i-m-getting-errors-while-using-burpsuite-against-the-owasp-benchmark-6666814a.

[45] Dave Wichers. *expectedresults-1.2.csv*. June 2016. URL: https://github.com/OWASP-Benchmark/BenchmarkJava/blob/master/expectedresults-1.2.csv.

[46] PortSwigger. *Testing for OS command injection vulnerabilities*. Jan. 2024. URL: https://portswigger.net/burp/documentation/desktop/testing-workflow/input-validation/command-injection/testing.

[47] PortSwigger. *LDAP injection*. URL: https://portswigger.net/kb/issues/00100500_ldap-injection (visited on 02/19/2024).

[48] PortSwigger. *Audit options*. Mar. 2024. URL: https://portswigger.net/burp/documentation/scanner/scan-configurations/audit-options.

[49] PortSwigger. *Complementing your manual testing with Burp Scanner*. May 2024. URL: https://portswigger.net/burp/documentation/desktop/testing-workflow/scanner-manual-testing.

[50] PortSwigger. *Issue definitions*. Mar. 2024. URL: https://portswigger.net/burp/documentation/desktop/tools/target/issue-definitions.

[51] PortSwigger. *How to detect SQL injection vulnerabilities*. URL: https://portswigger.net/web-security/sql-injection#how-to-detect-sql-injection-vulnerabilities (visited on 02/12/2024).

[52] Anuj Mehta. *ERROR: unterminated quoted string at or near*. Aug. 2010. URL: https://stackoverflow.com/questions/3499483/error-unterminated-quoted-string-at-or-near.

[53] Satishbabu Gunukula. *syntax error: unterminated quoted string literal*. Nov. 2015. URL: https://www.oracleracexpert.com/2015/11/syntax-error-unterminated-quoted-string.html.

[54] Ray Doyle. *Referer XSS with a Side of Link Injection*. Aug. 2019. URL: https://www.doyler.net/security-not-included/referer-xss.

[55] PortSwigger. *XSS: Beating HTML Sanitizing Filters*. URL: https://portswigger.net/support/xss-beating-html-sanitizing-filters (visited on 02/23/2024).

[56] Tom Hudson. *gf*. 2020. URL: https://github.com/tomnomnom/gf.

[57] NIST. *CVE-2014-6271 Detail*. Sept. 2014. URL: https://nvd.nist.gov/vuln/detail/cve-2014-6271.

[58] Shay Chen. *wavsep*. URL: https://code.google.com/archive/p/wavsep/ (visited on 04/29/2024).

[59] PortSwigger. *Open redirection (reflected DOM-based)*. URL: https://portswigger.net/kb/issues/00500111_open-redirection-reflected-dom-based (visited on 06/06/2024).

[60] Tasos Laskos. *LFI test case 37 & similar test cases don't function under linux*. July 2012. URL: https://code.google.com/archive/p/wavsep/issues/10.

[61] Unknown WAVSEP contributor. *LFI test cases throwing: java.lang.IllegalArgumentException: URI has an authority component*. July 2012. URL: https://code.google.com/archive/p/wavsep/issues/8.

[62] giper45. *Publish Docker Image*. Jan. 2023. URL: https://github.com/luigiurbano/Reinforced-Wavsep/issues/1#issuecomment-1404828479.

[63] PortSwigger. *PortSwigginar: Burp Scanner for pentesters - March 2023*. Mar. 2023. URL: https://www.youtube.com/watch?v=mDYsmfeSxd8&t=1068s.

[64] PortSwigger. *Using Burp's Session Handling Rules with anti-CSRF Tokens*. URL: https://portswigger.net/support/using-burp-suites-session-handling-rules-with-anti-csrf-tokens (visited on 04/30/2024).

[65] PortSwigger. *HTML5 web message manipulation (DOM-based)*. URL: https://portswigger.net/kb/issues/00500e00_html5-web-message-manipulation-dom-based (visited on 06/06/2024).

[66] PortSwigger. *HTML5 storage manipulation (DOM-based)*. URL: https://portswigger.net/kb/issues/00500f00_html5-storage-manipulation-dom-based (visited on 06/06/2024).

[67] PortSwigger. *DOM Invader*. May 2024. URL: https://portswigger.net/burp/documentation/desktop/tools/dom-invader.

[68] MDN contributors. *Mixed content*. Dec. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

[69] PortSwigger. *Strict transport security not enforced*. URL: https://portswigger.net/kb/issues/01000300_strict-transport-security-not-enforced (visited on 05/22/2024).

[70] NetSPIWillD. *HSTS - Missing includeSubDomains*. July 2023. URL: https://github.com/NetSPIWillD/BChecks/blob/main/passive/HSTS_Misconfigured_includeSubDomains.bcheck.

[71] NetSPIWillD. *HSTS - Missing preload*. July 2023. URL: https://github.com/NetSPIWillD/BChecks/blob/main/passive/HSTS_Misconfigured_preload.bcheck.

[72] Google. *template package*. May 2024. URL: https://pkg.go.dev/html/template.

[73] Laura Doktorova. *doT*. July 2020. URL: https://github.com/olado/doT.

[74] LinkedIn. *Dust.js*. July 2023. URL: https://github.com/linkedin/dustjs.

[75] swisskyrepo. *Command Injection*. URL: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Command%20Injection/README.md (visited on 04/17/2024).

[76] MrW0l05zyn. *RCE - Linux*. Aug. 2023. URL: https://github.com/MrW0l05zyn/bchecks/blob/main/RCE%20-%20Linux.bcheck.

[77] PortSwigger. *Burp Collaborator server*. Jan. 2024. URL: https://portswigger.net/burp/documentation/collaborator/server.

[78] Hannah-PortSwigger. *given query or body insertion point not working for Body paramters*. Aug. 2023. URL: https://github.com/PortSwigger/BChecks/issues/77#issuecomment-1686534176.

[79] Reza M. Parizi et al. "Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools". In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). July 2018. DOI: 10.1109/COMPSAC.2018.00139.

[80] OWASP. *OWASP Vulnerable Web Applications Directory*. URL: https://owasp.org/www-project-vulnerable-web-applications-directory/ (visited on 02/06/2024).

[81] HackerOne. *The HackerOne Top 10 Vulnerability Types*. URL: https://www.hackerone.com/top-ten-vulnerabilities (visited on 02/12/2024).

[82] Bugcrowd. *Priority one report 2022*. Nov. 2023. URL: https://www.bugcrowd.com/wp-content/uploads/2023/11/Priority-One-Report-2022-Edition.pdf.

[83] Dave Wichers. *Category updates OWASP Benchmark*. Nov. 2017. URL: https://github.com/OWASP-Benchmark/BenchmarkJava/issues/43#issuecomment-343580993.

[84] OWASP. *XML External Entity (XXE) Processing*. URL: https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing (visited on 02/20/2024).

[85] OWASP. *Cross Site Request Forgery (CSRF)*. URL: https://owasp.org/www-community/attacks/csrf (visited on 02/09/2024).

[86] Carnegie Mellon University. *The Official CAPTCHA Site*. 2000. URL: http://www.captcha.net/.

[87] Johannes Dahse and Thorsten Holz. *Static Detection of Second-Order Vulnerabilities in Web Applications*. Aug. 2014. URL: https://www.usenix.org/sites/default/files/conference/protected-files/sec14_slides_dahse.pdf.

[88] KirstenS et al. *Cross Site Scripting (XSS)*. URL: https://owasp.org/www-community/attacks/xss/ (visited on 02/23/2024).

[89] Eugene Lim. *Same Same But Different: Discovering SQL Injections Incrementally with Isomorphic SQL Statements*. Apr. 2020. URL: https://spaceraccoon.dev/same-same-but-different-discovering-sql-injections-incrementally-with/.

[90] PortSwigger. *Exploiting blind SQL injection by triggering time delays*. URL: https://portswigger.net/web-security/sql-injection/blind#exploiting-blind-sql-injection-by-triggering-time-delays (visited on 02/06/2024).

[91] Carnegie Mellon Database Group. *Database of Databases*. URL: https://dbdb.io/ (visited on 03/11/2024).

[92] Cody Slingerland. *Horizontal Vs. Vertical Scaling: How Do They Compare?* May 2023. URL: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/.

[93] Sarah Elder. "Vulnerability detection is just the beginning". In: *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings*. ICSE '21. Virtual Event, Spain: IEEE Press, 2021, pp. 304–308. DOI: 10.1109/ICSE-Companion52605.2021.00133. URL: https://doi.org/10.1109/ICSE-Companion52605.2021.00133.

[94] Roshan Namal Rajapakse, Mansooreh Zahedi, and Muhammad Ali Babar. "An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps". In: *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ESEM '21. Bari, Italy: Association for Computing Machinery, 2021. ISBN: 9781450386654. DOI: 10.1145/3475716.3475776. URL: https://doi.org/10.1145/3475716.3475776.

[95] PortSwigger. *Web application security testing*. URL: https://portswigger.net/burp/application-security-testing (visited on 06/11/2024).

[96] Syed, PortSwigger Agent. *Audit insertion points in nested GET parameters*. URL: https://forum.portswigger.net/thread/audit-insertion-points-in-nested-get-parameters-6fae2a91 (visited on 06/26/2024).

[97] PortSwigger. *Cookie jar*. May 2024. URL: https://portswigger.net/burp/documentation/desktop/settings/sessions#cookie-jar.

[98] PortSwigger. *Scanning single-page apps*. Mar. 2024. URL: https://portswigger.net/burp/documentation/scanner/scanning-spas.

[99] PortSwigger. *Long redirection response*. URL: https://portswigger.net/kb/issues/00400800_long-redirection-response (visited on 06/12/2024).

[100] PortSwigger. *Crawl strategy*. Mar. 2024. URL: https://portswigger.net/burp/documentation/scanner/scan-configurations/crawl-options#crawl-strategy.

[101] PortSwigger. *CSS injection (reflected)*. URL: https://portswigger.net/kb/issues/00501300_css-injection-reflected (visited on 06/12/2024).

[102] PortSwigger. *CSS injection (stored)*. URL: https://portswigger.net/kb/issues/00501301_css-injection-stored (visited on 06/12/2024).

[103] PortSwigger. *OS command injection*. URL: https://portswigger.net/kb/issues/00100100_os-command-injection (visited on 06/12/2024).

[104] PortSwigger. *Open redirection (reflected)*. URL: https://portswigger.net/kb/issues/00500100_open-redirection-reflected (visited on 06/12/2024).

[105] PortSwigger. *Open redirection (stored)*. URL: https://portswigger.net/kb/issues/00500101_open-redirection-stored (visited on 06/12/2024).