

Performance Analysis of Heapdumpfinder and Dirsearch on Searching Vulnerability Heapdump Files in Spring Boot Web-Based Applications

Nelmiawati¹, Wahyu Kurniady², Hamdani Arif³, and Nur Cahyono K.⁴

¹Rekayasa Keamanan Siber, Politeknik Negeri Batam, Batam, Indonesia,

²Teknik Komputer, Politeknik Negeri Batam, Batam, Indonesia

*Corresponding author: nelmia.wati@gmail.com

ABSTRACT

This research aims to analyze and compare the speed and accuracy of the Heapdumpfinder tool with Dirsearch in finding heapdump file vulnerabilities in spring boot-based web applications. Heapdump is a file that often contains sensitive data, such as credentials and authentication tokens, making it vulnerable to exploitation by attackers. Although Dirsearch is widely used in hidden file search, it has limitations in finding vulnerability of heapdump files in terms of speed and accuracy. To overcome this, Heapdumpfinder was created and customized to search for heapdump file vulnerabilities in the spring boot web framework. This tool utilizes fuzzing techniques and python-based asynchronous architecture to increase speed and accuracy in the search process. In the process of building the tool, the Rapid Application Development (RAD) framework is used to accelerate the development cycle iteratively. In addition, the performance evaluation of the tool is done using Confusion Matrix to measure the performance of the tool in finding heapdump files in more detail. The test results show that Heapdumpfinder excels in search speed and accuracy, and is able to minimize the risk of false positives compared to Dirsearch. Thus, Heapdumpfinder is expected to help pentesters and security researchers in finding vulnerabilities of heapdump files in the Spring Boot web framework.

Keywords: Heapdumpfinder, Dirsearch, File heapdump, Spring boot, Rapid Application Development (RAD), Confusion Matrix

1. INTRODUCTION

Web application security is a critical issue in the digital era as these platforms handle increasing amounts of sensitive data (Alsaedi et al., 2022). Web applications are widely used by various users, from large enterprises to individuals, expanding the potential risk landscape for unauthorized data access (I. M. P. Utama et al., 2022). Misconfigurations during development, often due to limited understanding of framework security, are a common source of vulnerabilities. Spring Boot, a popular Java-based web framework, simplifies development with features like auto-configuration and microservice architecture support. However, this convenience can also lead to security gaps, especially in memory management related to heapdump files. Heapdumps, which capture snapshots of JVM heap memory, may contain sensitive data such as credentials and tokens (Gul, 2024). Unauthorized access to heapdump files can expose sensitive data, posing significant security risks as highlighted by the OWASP MASTG-TEST-0011 guideline. Dirsearch is a common tool used to detect such vulnerabilities by brute-forcing web directories (Nurelia et al., 2024), but it suffers from slow scanning and lower accuracy, often leading to false positives and negatives. To overcome these limitations, this research introduces HeapdumpFinder, a tool designed with a more focused and systematic scanning method to improve detection speed and accuracy.

Several prior studies have explored related areas, including web application vulnerability testing using Dirsearch, application development using the Rapid Application Development (RAD) method, and the use of heapdump files within Spring Boot environments. One such study by Ferzha Putra Utama and Raden Muhamad Hilmi Nurhadi, titled "Uncovering the Risk of Academic Information System Vulnerability through PTES and OWASP Method," demonstrated Dirsearch's role in identifying unprotected directories within an academic information system (F. P. Utama & Nurhadi, 2024). Their findings revealed the presence of accessible directories such as /config, /module, and /main, along with sensitive files like version.txt, suggesting significant vulnerabilities within university systems in Indonesia.

Another study by Abu Baker and Shiraz (Gul, 2024) examined the use of heapdump files for analyzing Java application performance and identifying memory leaks. Their research indicated that poor memory allocation management leads to resource overconsumption and system crashes. Furthermore, they highlighted that heapdumps often contain sensitive data, thus posing security risks if not properly secured.

In the context of software development (Afrianto Singgalen et al., 2024), explored the application of the RAD method in their work titled "Implementation of Rapid Application Development (RAD) for Community-Based Ecotourism Monitoring System." The study found that RAD effectively accelerated development cycles by involving users throughout the planning, prototyping, and implementation stages, resulting in flexible systems that could adapt quickly to user needs.

Meanwhile, Alif Muhammad Akmal, Nono Heryana, and Arip Solehudin conducted a security analysis of the Singaperbangsa Karawang University website using vulnerability assessment methods (Akmal et al., 2022). In this study, Dirsearch was utilized to identify unsecured directories and files during the scanning phase, exposing folders such as /config, /install, and /admin that could be accessed without authentication. These findings highlight significant weaknesses in web application security, especially when relying solely on generic directory brute-forcing tools like Dirsearch. However, their research did not focus on specific memory-related risks such as heapdump exposure. This marks a key difference with our study, which aims to address this gap by introducing a specialized tool, HeapdumpFinder, to detect memory leak-related vulnerabilities more precisely. By focusing on heapdump detection rather than general directory listing, our tool aims to reduce false positives and better target critical security flaws that are often overlooked in broader assessments.

Lastly, Yash Jani's research, "Spring Boot Actuator: Monitoring and Managing Production-Ready Applications," discusses the utility of Spring Boot and its Actuator module in developing and monitoring Java-based applications (Jani, 2021). The study emphasizes the benefits of using built-in endpoints for monitoring application health, performance, and resource management. While the Actuator module enhances observability, it can also unintentionally expose sensitive information if misconfigured, particularly endpoints like /heapdump which may contain confidential memory snapshots. However, Jani's research does not explore the security implications of these endpoints when exposed publicly. Building on this, our research takes a security-oriented approach by evaluating the exposure risk of such endpoints and offering a targeted detection solution through HeapdumpFinder. This development contributes by extending the insights from Actuator functionality into the area of vulnerability detection and mitigation.

This study aims to analyze the performance of Heapdumpfinder compared to Dirsearch tool in detecting heapdump vulnerabilities in Spring Boot-based web applications. The analysis focuses on comparing the speed and accuracy of both tools, using a confusion matrix to measure classification errors (false positives and false negatives).

2. METHODS

This research adopts a quantitative approach with an experimental design to evaluate and compare the performance of two tools, Heapdumpfinder and Dirsearch, in locating heapdump files within Spring Boot-based web applications. The tool development process follows the Rapid Application Development (RAD) method to accelerate the development cycle through iterative stages (Afrianto Singgalen et al., 2024; Yuricha & Phan, 2024). In addition, tool performance is evaluated using a confusion matrix (Normawati & Prayogi, 2021; Phillips et al., 2024; Sathyanarayanan, 2024), which enables a more detailed assessment of their capabilities in detecting heapdump files. This approach allows for a fast-paced development process while ensuring that performance testing is conducted in a structured and measurable manner.

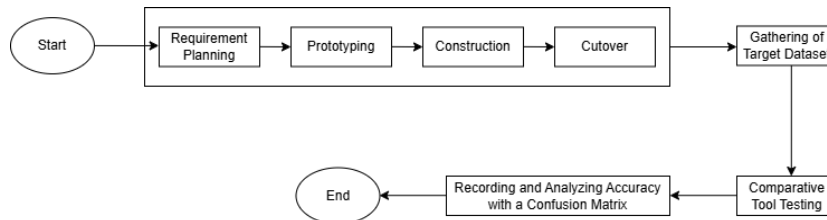


Figure 1. Research Methods

2.1 Rapid Application Development (RAD) Framework

As explained by (Afrianto Singgalen et al., 2024; Manuhutu et al., 2025), RAD is a software development approach oriented towards accelerating the system development process through shorter, iterative, and collaborative cycles. According to (Afrianto Singgalen et al., 2024), RAD includes four main stages, namely Requirement Planning, Prototyping, Construction, and Cutover. This method allows the development of tools to be carried out with a focus on the specific needs of users while ensuring the flexibility to make adjustments at each stage of development (Afrianto Singgalen et al., 2024). In the context of this research, the RAD method is applied as a framework to build the Heapdumpfinder tool. The following stages in the RAD method are implemented:

2.1.1 Requirement Planning

In the early stages of building the Heapdumpfinder tool, the main focus is on identifying the main requirements and objectives of the tool (Afrianto Singgalen et al., 2024; Hidayat & Hati, 2021; Panja & Manongga, 2023). This process begins with designing the architecture of the tool to be built, as well as compiling a flowchart to describe the system workflow in a structured manner.

Tool Builder Architecture

Heapdumpfinder						
Aiohttp	Asyncio	Async Timeout	Urljoin from Urllib Parse	Time	Argparse	OS
Python 3.12.6						
WSL						
(Linux version 5.15.133.1-microsoft-standard-WSL2 (root@1c602f52c2e4) (gcc (GCC) 11.2.0, GNU ld (GNU Binutils) 2.37)						
LAPTOP-27VHMATP (Intel Core i5-11300H, 24GB RAM, NVIDIA RTX 3050)						

Figure 2. Tool Builder Architecture

1. aiohttp: Python library for asynchronous HTTP clients and servers, enabling concurrent requests without blocking.
2. asyncio: Python module for writing asynchronous programs using event loops and non-blocking I/O to handle tasks concurrently.
3. async_timeout: Library to set timeouts on async operations, preventing delays from slow server responses.
4. urljoin from urllib.parse: Used to correctly combine URLs, useful in searching heapdump files.
5. Time: Tracks execution time to measure performance.
6. argparse: Parses command-line arguments, allowing flexible CLI inputs like URLs or file lists.
7. OS: Interacts with the operating system for file and directory management.

Heapdumpfinder Algorithm Pseudocode

```

1.Display the program logo
2.Initialize the wordlist with paths that may lead to heapdump files
3.Define function add_protocol(url):
    If the url does not start with "http://" or https://
        Add "http://" in front of the URL
    Return the url
4. Define function fetch_and_validate(session, url, ssl_context,
timeout):
    Try:
        Send a GET request to the url
        If the response status is 200:
            Get headers: Content-Type, Content-Length, Content-
Disposition
            If Content-Length is 0:
                Read a small portion of the response and calculate its

```

```

size
    If the content matches the heapdump format and size > 0:
        Print the url as a valid heapdump
        Return the url
    Else:
        Print the url as invalid
Else:
    Print that heapdump was not found with the status code
    Catch network, SSL, timeout, or other general errors
    Return None
5. Define function check_heapdump(session, base_url, ssl_context,
timeout):
    For each path in the wordlist:
        Combine base_url and path into a complete url
        Call fetch_and_validate(url)
    Wait for all results to finish
    Return a list of valid URLs (heapdumps found)
6. Define function main():
    Display the logo
    Read user arguments (-s or --t)
    Initialize SSL context (without certificate verification)
    Set connection timeout
    Record the start time
    If the user specifies -s (single URL):
        Add protocol to the URL
        Call check_heapdump()
        If results are found:
            Print all detected heapdump URLs
        Else:
            Print "not found"
    Else if the user specifies --t (file containing URLs):
        Read all URLs from the file
        For each URL in the list:
            Add protocol
            Call check_heapdump()
        Combine all results
        Print all detected heapdumps or a "not found" message
    Print program execution time
7. Run main() in an asyncio event loop
Handle KeyboardInterrupt (Ctrl+C)

```

The pseudocode above outlines the main workflow of the HeapdumpFinder tool, which is designed to detect exposed heapdump files in Spring Boot-based web applications. The program begins by displaying a logo for user interface clarity and then initializes a predefined

wordlist containing common URL paths where heapdump files are likely to be located, such as /heapdump, /actuator/heapdump, or /management/heapdump. A helper function `add_protocol()` ensures all input URLs have the correct HTTP or HTTPS scheme before processing. The core scanning functionality resides in `fetch_and_validate()`, which handles the logic of sending HTTP GET requests to potential heapdump URLs, checking for a 200 OK status code, and verifying whether the returned content matches characteristics of a heapdump file (such as content type, file size, or content disposition). If a valid heapdump is detected, the URL is reported as vulnerable. The function `check_heapdump()` iterates through all wordlist paths for a given base URL and performs asynchronous validation of each potential heapdump endpoint. It collects and returns any URLs confirmed to be exposing heapdump files. The `main()` function serves as the program's entry point, accepting either a single URL (-s) or a list of URLs from a file (-t). It initializes SSL settings to bypass certificate checks, sets timeout constraints, and tracks the tool's runtime. It handles each target accordingly, consolidates the results, and prints detected heapdump endpoints or informs the user if no vulnerabilities were found. Finally, the tool runs within an `asyncio` event loop for better performance and responsiveness. It also gracefully handles user interruption (e.g., Ctrl+C) to ensure a clean exit.

2.1.2 Prototyping

This phase involves the initial development of the tool's prototype, as described in (Afrianto Singgalen et al., 2024). The prototype is developed with a primary focus on detecting heapdump files in Spring Boot-based applications using fuzzing techniques. Initial testing was conducted directly on real-world targets that utilize the Spring Boot web framework to evaluate the tool's performance in locating heapdump files.

2.1.3 Construction

After the initial prototype was developed and refined based on testing results and feedback, the full construction phase of the tool began (Afrianto Singgalen et al., 2024). During this phase, performance optimization was carried out in the Python code to improve both the speed and accuracy of heapdump file vulnerability detection. Library integration was also enhanced to accelerate the scanning process, optimize connection management, and improve the accuracy of heapdump file validation.

2.1.4 Cutover

This stage represents the final implementation phase (Afrianto Singgalen et al., 2024), where HeapdumpFinder is ready to be used as a Command Line Interface (CLI) tool on Linux systems. After a series of tests and refinements, the tool can now be executed directly from the terminal without the need for a graphical interface or deployment to a production server. To ensure stable performance and reliability in various scenarios, the tool was tested against a large set of target URLs. These tests aimed to identify potential issues and ensure the tool operates effectively. If problems were encountered during implementation, further improvements and optimizations would be applied to enhance the tool's stability and capability.

2.2 Gathering of Target Dataset

The target dataset collection in this research was conducted using Shodan. Shodan is a search engine specifically designed to find devices connected to the internet (Rajasekar & Rajkumar, 2023; Roble & Abualkibash, 2021). The search process involved using specific search formats tailored to the research needs, aiming to discover targets running the Spring Boot web framework. The search query format used in Shodan is shown below:

`http.html:"Whitelabel Error Page"`

Shodan Search Format for Web Framework Spring Boot I

`http.favicon.hash:116323821`

Shodan Search Format for Web Framework Spring Boot II

The use of Shodan for data collection is a common and accepted practice in cybersecurity research for examining publicly accessible internet-connected systems. This is consistent with the findings of studies (Rajasekar & Rajkumar, 2023). However, the dataset gathered in this study was not manually validated for accuracy or timeliness, as the status of target systems may change over time. This limitation should be acknowledged when interpreting the findings. The research focused exclusively on public systems that do not require special authorization. No exploitation, damage, or personal gain was involved; all activities were conducted strictly for educational and ethical purposes.

2.3 Comparative Tool Testing

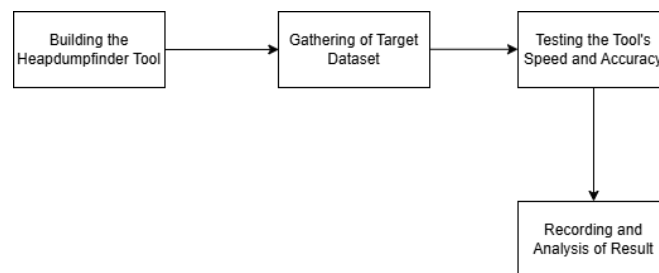


Figure 3. Testing Process Flow

After developing HeapdumpFinder using the RAD method and collecting the target dataset, a comparative evaluation was performed to assess speed and accuracy in detecting heapdump file vulnerabilities. As shown in Figure 3, the evaluation compared HeapdumpFinder and Dirsearch based on search speed measuring how fast each tool locates heapdump files and detection accuracy assessing how well they distinguish valid from invalid files. Speed was measured by execution time, while accuracy used a confusion matrix to evaluate classification errors.

2.4 Recording and Analyzing Accuracy with a Confusion Matrix

After the comparative testing of the tools was completed, the next step was to analyze the results to determine which tool is superior in detecting heapdump files in Spring Boot-based web applications. The analysis method uses a confusion matrix. A confusion matrix is a

standard evaluation metric used to assess the accuracy of a model in classifying data into the correct categories (Maulana Al-Fakih & Lammada, 2024). The confusion matrix includes four key parameters (Indransyah et al., 2022): True Positive (TP) for correctly detected heapdump files, False Positive (FP) for non-heapdump files wrongly classified as heapdumps, False Negative (FN) for missed heapdump files, and True Negative (TN) for correctly rejected non-heapdump files. Accuracy test results are further evaluated using metrics such as Accuracy (overall correctness), Precision (proportion of true positives among detections), Recall (ability to find all actual heapdump files), and F1-Score (a balance of Precision and Recall).

3. RESULT & DISCUSSION

This study used an experimental method to evaluate the speed and accuracy in detecting heapdump files. Tests were conducted on three groups of Spring Boot web applications, consisting of 100, 300, and 500 targets. These target sizes were chosen to achieve an optimal balance between performance evaluation and system stability. The selected scales allow for a gradual and measurable analysis of performance changes, while avoiding system overload caused by excessively large test sets. HeapdumpFinder and Dirsearch were tested separately on each scale, and each test was repeated three times to ensure consistent and reliable results. The evaluation metrics focused on average execution time and accuracy in identifying valid heapdump files.

3.1 Speed

Testing was conducted on target lists of 100, 300, and 500 entries, each repeated three times. The average execution times of HeapdumpFinder and Dirsearch were compared to evaluate their speed under different workloads. The results are shown in the graph below.

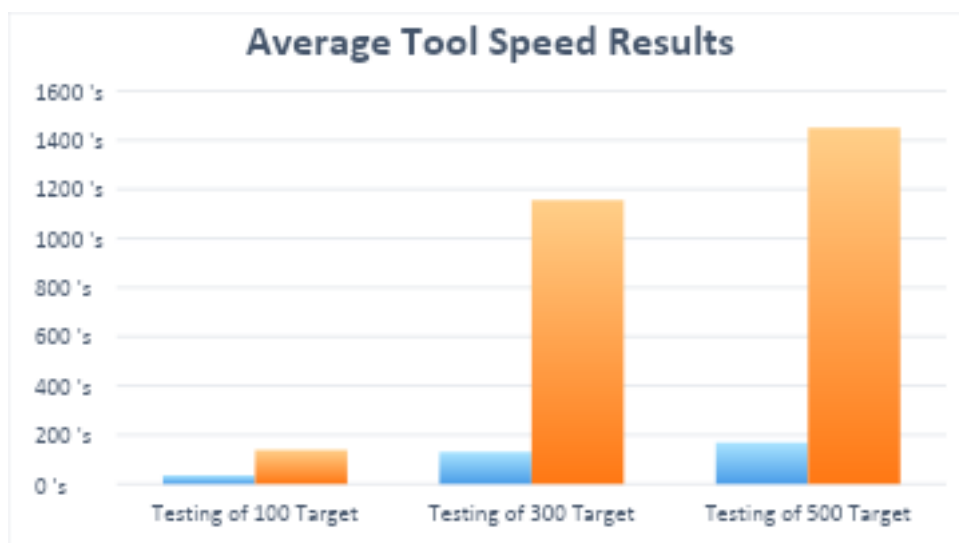


Figure 4. Results of Speed Testing on Both Tools

Based on the test results shown in Figure 4, HeapdumpFinder demonstrates a clear speed advantage over Dirsearch in locating heapdump files. For 100 targets, HeapdumpFinder averaged 34.8 seconds versus Dirsearch's 138.8 seconds. With 300 targets, HeapdumpFinder

completed scans in 130.56 seconds compared to Dirsearch's 1115 seconds. The largest gap was in 500 targets, where HeapdumpFinder took 167.97 seconds while Dirsearch required 1451 seconds. This speed is mainly due to HeapdumpFinder's use of asynchronous I/O with aiohttp and asyncio, enabling parallel HTTP requests, unlike Dirsearch's slower thread-based approach.

3.2 Accuracy

In this aspect, the success rate of HeapdumpFinder and Dirsearch in detecting valid heapdump files from the tested URLs was measured. The tests were conducted using the same target counts as the speed tests 100, 300, and 500 URLs with each test repeated three times to ensure consistent results. The accuracy data was calculated as the percentage of successful detections of heapdump files. Furthermore, the accuracy assessment was complemented by an analysis using a confusion matrix(Helmud et al., 2024) to provide a more comprehensive view of the results. The figure below shows the heatmap of the accuracy test results for HeapdumpFinder and Dirsearch, illustrating the detection success rates for heapdump files across different test scales.

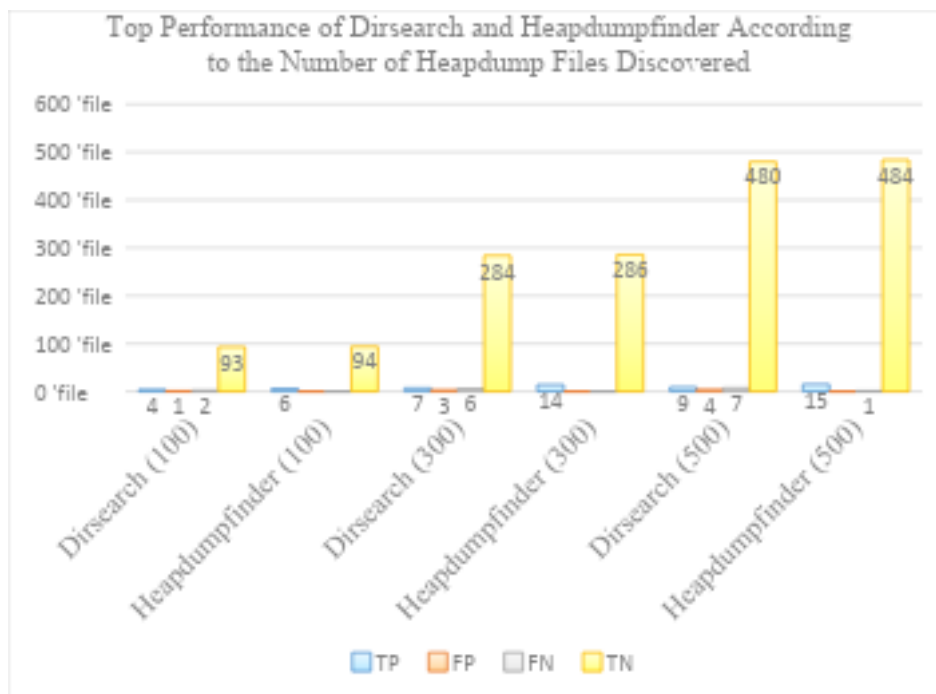


Figure 5. Top Performance Datasets of Both Tools

Table 1. Confusion Matrix Calculation Results Based on the Top Performance Dataset of Both Tools

Tools	Target	Confusion matrix calculation results are based on the best performance dataset of both tools, with the
-------	--------	--

		evaluation metrics calculated using (1), (2), (3), and (4)			
		Accuracy	Precision	Recall	F-1 Score
Dirsearch	100	97 %	80%	60%	68,5%
Dirsearch	300	97%	70%	53,8%	60,8%
Dirsearch	500	97,8%	69,2%	56,2%	62%
Heapdumpfinder	100	100%	100%	100%	100%
Heapdumpfinder	300	100%	100%	100%	100%
Heapdumpfinder	500	99,8%	100%	93,7%	96,7%

$$\text{Accuracy} = \frac{(TP+TN)}{(TP+TN+FP+FN)} \times 100\% \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

The test results shown in Figure 5 and Table 1 demonstrate that HeapdumpFinder outperforms Dirsearch in locating heapdump files across different scenarios. Using Precision, Recall, and Accuracy as metrics, HeapdumpFinder consistently achieved near-perfect precision and recall on tests with 100, 300, and 500 targets, indicating no false positives or negatives. In contrast, Dirsearch's performance dropped notably with 500 targets, showing more false detections and lower accuracy. HeapdumpFinder reached 99.8% accuracy on 500 targets, compared to Dirsearch's 97.8%. This higher accuracy is due to technical improvements in HeapdumpFinder's code, enhancing its ability to find heapdump files in Spring Boot web frameworks. These factors include:

1. More Flexible SSL and Redirect Management

HeapdumpFinder handles SSL certificate validation and redirects better. Some servers hosting heapdump files use invalid or self-signed SSL certificates, causing other tools to fail to connect. HeapdumpFinder overcomes this by disabling SSL validation with the following configuration:

```
import ssl
ssl_context = ssl.create_default_context()
ssl_context.check_hostname = False
ssl_context.verify_mode = ssl.CERT_NONE
```

Code for SSL and Redirect Management

This allows HeapdumpFinder to access targets despite certificate issues, improving search success. In contrast, Dirsearch struggles with such cases, making some targets inaccessible and reducing its search performance.

2. Heapdump File Validation and Checking

HeapdumpFinder does not rely solely on the 200 OK status code to confirm the presence of a heapdump file. It also implements a file size check, where in this scenario, HeapdumpFinder reads only a small portion of the data from the HTTP response to conserve resources, before determining the actual size of the heapdump file:

```
# If the file has 0 bytes, read only a portion of the data
if content_length == 0:
    data = await response.content.readany()
    content_length = len(data)
```

Code for Reading Heapdump Files

From the code, it means that if `content_length` is 0, it indicates that the server does not provide information about the file size, or the file itself is empty. To handle this, HeapdumpFinder reads a small portion of data from the HTTP response using `await response.content.readany()`. After the data is read, HeapdumpFinder updates the file size using `content_length = len(data)`, ensuring that it can verify whether the file is truly empty or contains data, before proceeding with further validation. After this size checking process, validation is performed using the following code.

```
# Heapdump validation including .gz
if ('application/octet-stream' in content_type or
    'application/vnd.ibm.jvm.heapdump' in content_type or
    'application/zip' in content_type or
    'application/gzip' in content_type or
    'heapdump.zip' in content_disposition or
    'heapdump.gz' in content_disposition or
    url.endswith('.gz')) and content_length > 0:

    print(f"\033[92m[+] Valid heapdump found at: {url} (Type:
    {content_type}, Size: {content_length})")
    return url
```

Code for Validation Improvement

From the code above, HeapdumpFinder employs a layered validation process to ensure detected files are truly valid heapdumps. It first checks the Content-Type for known heapdump formats such as `vnd.ibm.jvm.heapdump`, `application/octet-stream`, `application/zip`, or `application/gzip`. Additionally, it examines the Content-Disposition header for filenames indicating heapdumps, like `heapdump.zip` or `heapdump.gz`, since some servers may provide inaccurate Content-Type information. Next, the URL extension is checked for `.gz`, a common format for compressed heapdumps, increasing the likelihood that the file is valid. Finally, HeapdumpFinder verifies the file size is greater than zero to avoid false positives from empty files that still return a 200 OK status. By combining these validation methods, HeapdumpFinder improves detection accuracy and reduces false positives and negatives, ensuring that all identified files meet the required technical criteria.

4. CONCLUSION

Based on test results, HeapdumpFinder shows strong potential in detecting heapdump file vulnerabilities in Spring Boot frameworks compared to Dirsearch. It outperforms in fuzzing speed, identifying files in an average of 168 seconds for 500 targets, while Dirsearch takes 1451 seconds. HeapdumpFinder also achieves higher accuracy at 99.8%, versus Dirsearch's 97.8%. These results highlight HeapdumpFinder's superiority in vulnerability detection. This research, conducted ethically and solely for educational purposes on public systems, aims to aid penetration testers and security researchers. Future work will compare HeapdumpFinder with other tools like Gobuster, Wfuzz, and FFUF for broader analysis.

REFERENCES

- Afrianto Singgalen, Y., Jend Sudirman No, J., Semanggi, K., Setiabudi, K., Jakarta Selatan, K., & Khusus, D. (2024). Implementation of Rapid Application Development (RAD) for Community-based Ecotourism Monitoring System. *Journal of Information System Research (JOSH)*, 5(2), 530. <https://doi.org/10.47065/josh.v5i2.4749>
- Akmal, A. M., Heryana, N., & Arip Solehudin. (2022). Analisis Keamanan Website Universitas Singaperbangsa Karawang Menggunakan Metode Vulnerability Assessment. *Jurnal Pendidikan Dan Konseling*, 4(4), 6298–6309.
- Alsaedi, A., Alhuzali, A., & Bamasag, O. (2022). Effective and scalable black-box fuzzing approach for modern web applications. *Journal of King Saud University - Computer and Information Sciences*, 34(10), 10068–10078. <https://doi.org/10.1016/j.jksuci.2022.10.006>
- Gul, S. (2024). Optimizing Java Dump Analysis : Techniques for Debugging and Performance Improvement. December. <https://doi.org/10.13140/RG.2.2.19299.67364>
- Helmud, E., Helmud, E., Fitriyani, F., & Romadiana, P. (2024). Classification Comparison Performance of Supervised Machine Learning Random Forest and Decision Tree Algorithms Using Confusion Matrix. *Jurnal Sisfokom (Sistem Informasi Dan Komputer)*, 13(1), 92–97. <https://doi.org/10.32736/sisfokom.v13i1.1985>
- Hidayat, N., & Hati, K. (2021). Penerapan Metode Rapid Application Development (RAD) dalam Rancang Bangun Sistem Informasi Rapor Online (SIRALINE). 1, 8–17.
- Indransyah, R., Chrisnanto, Y. H., Sabrina, P. N., & Kom, S. (2022). Klasifikasi Sentimen Pergelaran MotoGP di Indonesia Menggunakan Algoritma Correlated Naive Bayes Clasifier. *INFOTECH Journal*, 8(2), 60–66. <https://doi.org/10.31949/infotech.v8i2.3103>
- Journal, E., & Jani, Y. (2021). Spring Boot Actuator : Monitoring and Managing Production-Ready Applications. 8(1), 107–112.
- Manuhutu, M., Manuhutu, A., Manuhutu, M., Manurung, T., & Uktolseja, L. J. (2025). Penerapan Metode Rapid Application Development Pada Sistem Informasi Sipani

- Store. RIGGS: Journal of Artificial Intelligence and Digital Business, 4(1), 8–18.
<https://doi.org/10.31004/riggs.v4i1.364>
- Maulana Al-Fakih, M., & Lammada, I. (2024). Analisis Perbandingan Kinerja Algoritma Mlpclassifier Dan Decision Tree Classification Pada Sistem Inspeksi Alat Kesehatan. JATI (Jurnal Mahasiswa Teknik Informatika), 8(4), 7907–7913.
<https://doi.org/10.36040/jati.v8i4.10484>
- Normawati, D., & Prayogi, S. A. (2021). Implementasi Naïve Bayes Classifier Dan Confusion Matrix Pada Analisis Sentimen Berbasis Teks Pada Twitter. Jurnal Sains Komputer & Informatika (J-SAKTI), 5(2), 697–711.
- Nurelia, F., Putri, S., Utomo, Y. B., & Kurniasari, I. (2024). Pengaruh Penetration Testing Dalam Menguji Celah Keamanan Pada Website (Studi Kasus: Dinas Kependudukan dan Pencatatan Sipil Kota Kediri). Jurnal Multinetics, 10(2), 121–128.
- Panja, E., & Manongga, D. (2023). Perancangan Sistem Informasi Keuangan Berbasis Web Pada Gks Mauliru Menggunakan Metode Rapid Application Development. JATI (Jurnal Mahasiswa Teknik Informatika), 7(1), 579–584.
<https://doi.org/10.36040/jati.v7i1.6401>
- Phillips, G., Teixeira, H., Kelly, M. G., Salas Herrero, F., Várbiro, G., Lyche Solheim, A., Kolada, A., Free, G., & Poikane, S. (2024). Setting nutrient boundaries to protect aquatic communities: The importance of comparing observed and predicted classifications using measures derived from a confusion matrix. Science of the Total Environment, 912(November 2023). <https://doi.org/10.1016/j.scitotenv.2023.168872>
- Rajasekar, V. R., & Rajkumar, S. (2023). A Study on Internet of Things Devices Vulnerabilities using Shodan. International Journal of Computing, 22(2), 149–158.
<https://doi.org/10.47839/ijc.22.2.3084>
- Roble, M., & Abualkibash, M. (2021). Using Shodan and Shodan API as a Vulnerability Tool for Security Testing. 6(4), 195–199.
- Sathyanarayanan, S. (2024). Confusion Matrix-Based Performance Evaluation Metrics. African Journal of Biomedical Research, December, 4023–4031.
<https://doi.org/10.53555/ajbr.v27i4s.4345>
- Utama, F. P., & Nurhadi, R. M. H. (2024). Uncovering the Risk of Academic Information System Vulnerability through PTES and OWASP Method. CommIT Journal, 18(1), 39–51. <https://doi.org/10.21512/commit.v18i1.9384>
- Utama, I. M. P., Putri, K. R., Wirayuda, A. A. E., Herlambang, V. A. T. P., Listartha, I. M. E., & Saskara, G. A. J. (2022). Analisis Perbandingan Kinerja Tool Website Directory Brute Force dengan Target Website DVWA. Informatik: Jurnal Ilmu Komputer, 18(3), 278.
<https://doi.org/10.52958/iftk.v18i3.5256>

Yuricha, Y., & Phan, I. K. (2024). Rancang Bangun Aplikasi Jurnal Perkuliahan Berbasis Progressive Web Application Menggunakan Metode Rapid Application Development. MALCOM: Indonesian Journal of Machine Learning and Computer Science, 4(3), 901–910. <https://doi.org/10.57152/malcom.v4i3.1370>