Chasing SSRF in Downstream Asynchronous Workflows

CornCon 11 | October 10th, 2025
Davenport, IA

David Garlak & Jon Schnell

Agenda

Speaker Introductions

Server-Side Request Forgery (SSRF) Primer

Categories of SSRF

Protections & Bypasses

Case Study: SSRF in Asynchronous Workflows

Demo Lab Example

Questions & Answers

Speaker Introductions

David Garlak
Independent Security Professional
Cleveland, OH

- Qualifications
  - B.S., Computer Science
  - M.S., Information Technology
  - PNPT , AWS Certified Security – Specialty
- Current Interests
  - Cloud privilege escalation & lateral movement
  - Container orchestration
  - Modern authentication specifications

Jon Schnell
Offensive Security Supervisor, RSM US LLP
Des Moines, IA

- Qualifications
  - B.S.E, Cyber Security Engineering, Iowa State University
  - CREST, AWS Practitioner
- Current Interests
  - Cloud security
  - Application & API testing
  - Embedded systems security
  - Home lab-ing & makerspace
  - Containerization technologies and security

Server-Side Request Forgery (SSRF) Primer

OWASP Top 10 Web Application Security Risks (A:10 2021)

*"Server-side request forgery is a web security vulnerability that allows an attacker to cause the server-side application to **make requests to an unintended location**.*

*In a typical SSRF attack, the attacker might cause the server to **make a connection to internal-only services** within the organization's infrastructure. In other cases, they may be able to force the server to **connect to arbitrary external systems**. This could leak sensitive data, such as authorization credentials"*

- PortSwigger Academy [1]

Often Leverageable in Vulnerability Linkage

- CWE-601: URL Redirection to Untrusted Site (Open Redirect)
- CWE-73: External Control of File Name or Path (File Inclusion)
- CWE-1390: Weak Authentication
- CWE-611: Improper Restriction of XML External Entity Reference
- CWE-16: Misconfiguration (e.g., Instance Metadata Service Options)

Categories of SSRF

Transparent SSRF

- Response from the forged request is returned to the attacker following exploitation
- Target server effectively acts as an intermediate proxy during abuse

Semi-Blind SSRF

- Forged request details are disclosed to the attacker (e.g., via collaborator interactions)
- Responses to the forged request are not disclosed

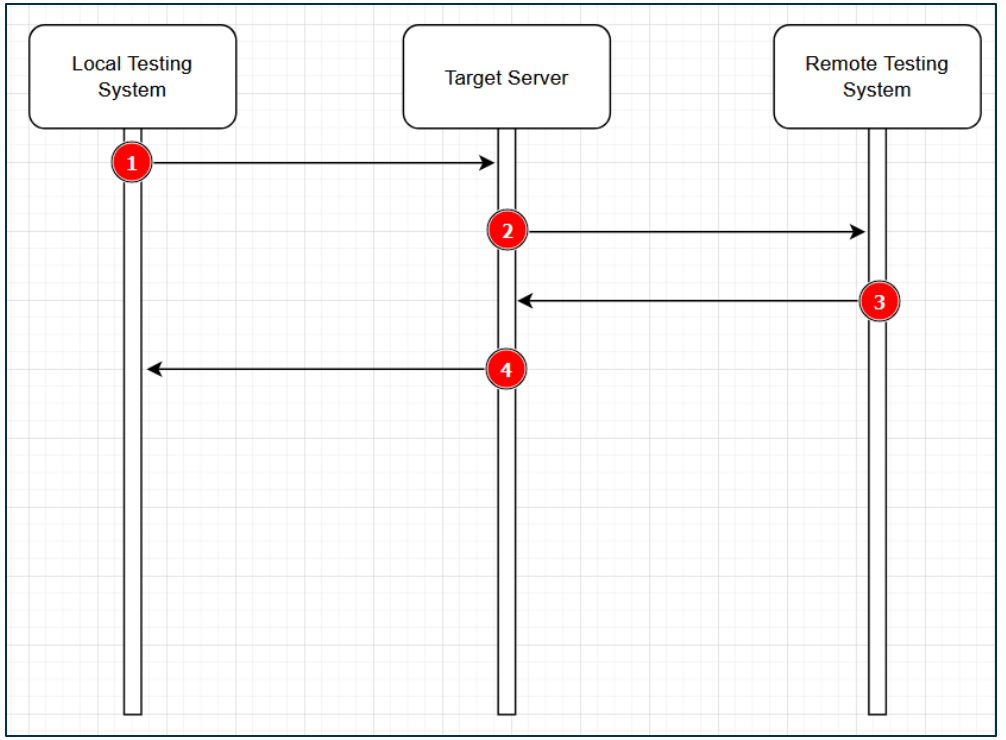
Blind SSRF

- Neither the forged request or response details are accessible to the attacker
- Detection typically relies on DNS interaction, time delays, or changes to integrated services / actions

Transparent SSRF Example Flow

- Example Transparent SSRF Example
  1. Testing system issues initial web request to the target containing the SSRF payload (e.g., a Burp Suite Collaborator reference)
  2. Target server consumes the payload and issues a request to the remote system
  3. Remote system provides web (or other protocol) response
  4. Target server completes the initial web request and returns data from the remote system's response

- Target server acts similarly to a web proxy



Example Sequence of SSRF Behavior

SSRF Protection Mechanisms

Input Validation

Using strong URI parsing processes to ensure the resolved destination of a request is a trusted / expected resource over the correct protocol.

- Proper URI host identification
- Address verification following resolution (IPv4 and IPv6)

Strict Response Handling

Ensuring that all response attributes (status code, headers, body format) are validated against expected patterns and handled appropriately.

- Restriction of response codes (e.g., 3XX series) and behavior
- Validation of response body contents

System Isolation

Implementing isolation of web servers to ensure traffic is only allowed to and from expected and verified sources / destinations.

- Strong firewall ingress and egress rules
- Offloading external service interactions to separate compute resources

Strong Authentication Controls

Configuring integrated services to require strong authentication that cannot be supplied arbitrarily through user inputs.

- Strict SSL/TLS connection requirements
- Message signing to identify and reject tampered requests

SSRF Protection Pitfalls

Insufficient Input Validation

Improper verification of request inputs that are leveraged to construct requests to an integrated system.

- Basic string parsing lacking consideration to URI syntax
- Exact string matching for host or IP address deny-lists prior to resolution (CNAME, A, AAAA, etc.)
- Acceptance of arbitrary request schemas

Lack of Response Validation

Acceptance of integrated services due to implicit trust of the third-party system.

- Tampered response contents containing unexpected or malformed body parameters
- Following of web redirects (i.e., 3XX responses) to restricted locations

Excessive Network Access

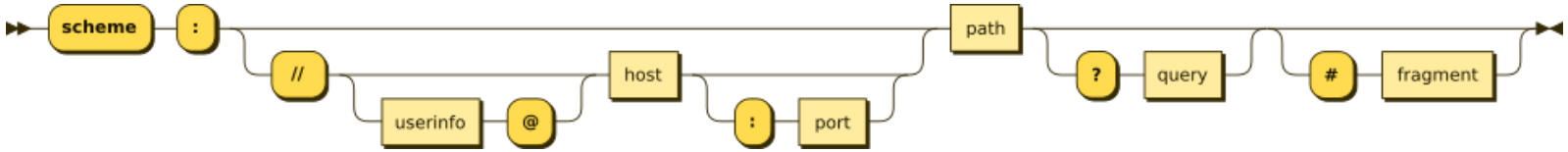
Horizontal or vertical network access control lists which allow for interactions with unexpected systems

- Port scanning and other interactions with adjacent internal infrastructure
- Connections to arbitrary external locations despite a limited set of expected destinations

Weak or Missing Authentication

Internal services, adjacent on the network or locally on the target server, which implicitly trust connections from the internal network.

- Interactions with alternative web services running on the loopback interface
- Password attacks against systems using HTTP basic authentication.

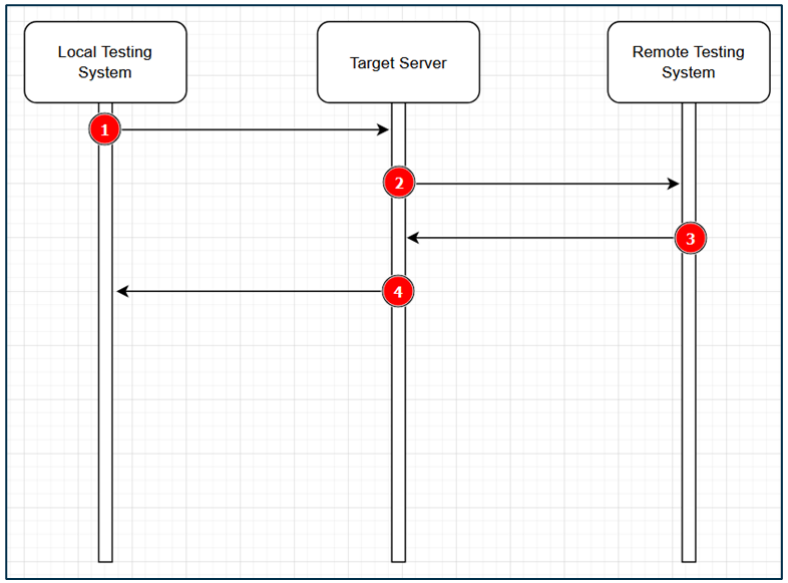


Uniform Resource Indicator (URI) Syntax [2]

Synchronous & Asynchronous Workflows

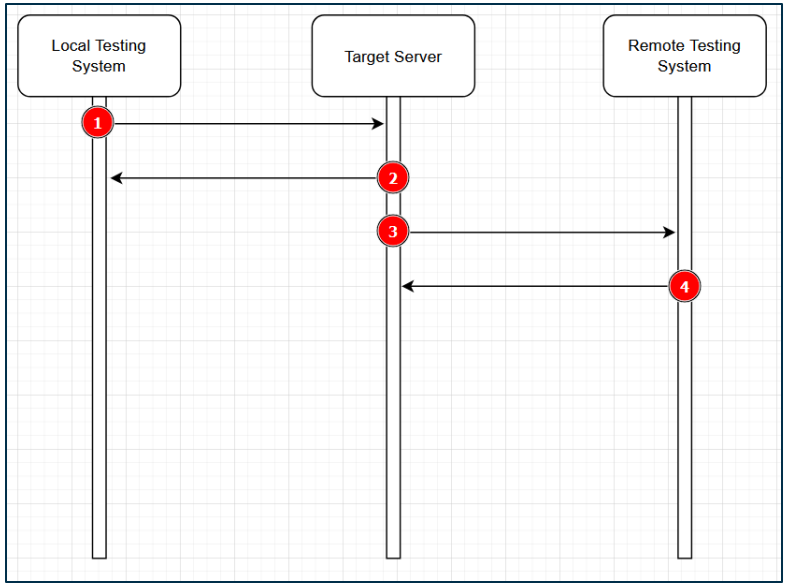
Synchronous SSRF Scenarios

- Forged request is performed prior to the target server's response, which depends upon the forged request.



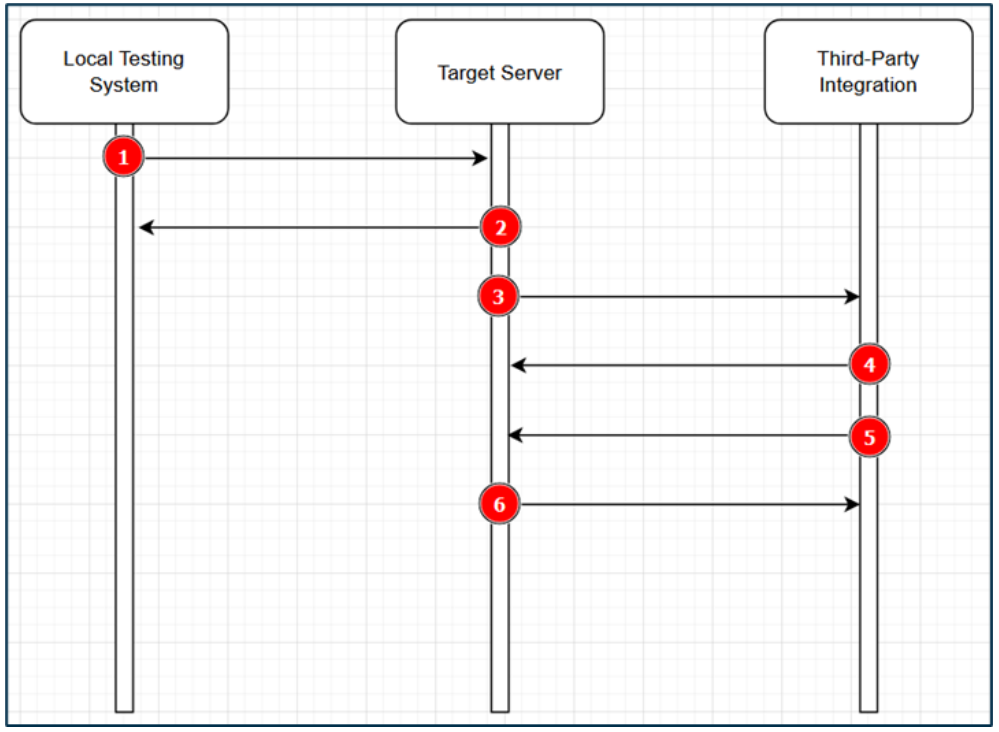
Asynchronous SSRF Scenarios

- Forged request is performed independently and does not affect the target server's response.



Downstream Asynchronous Workflows

1. Request is issued to the target server
2. Target server returns synchronous response
3. Asynchronous workflow begins with third-party
4. Third party handles async request
5. Third party integration performs an async action
6. And so on…

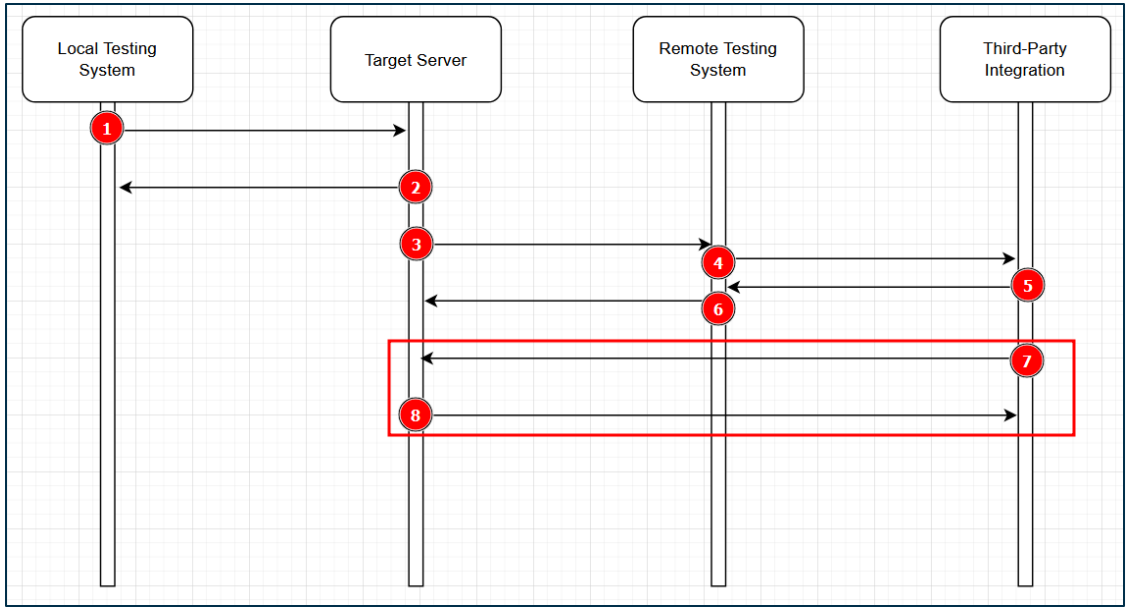


Downstream Async Workflow in Third-Party Integration

Case Study: SSRF in Asynchronous Workflows

- Exploration of subsequent steps by mimicking the responses from the third-party integration
  - This allows completion of the asynchronous workflow despite injection of a payload to abuse the SSRF vulnerability.
  - Downstream vulnerabilities may rely upon successful completion of prior steps
  - Forged responses from SSRF interactions can be used to inject payloads into subsequent phases.

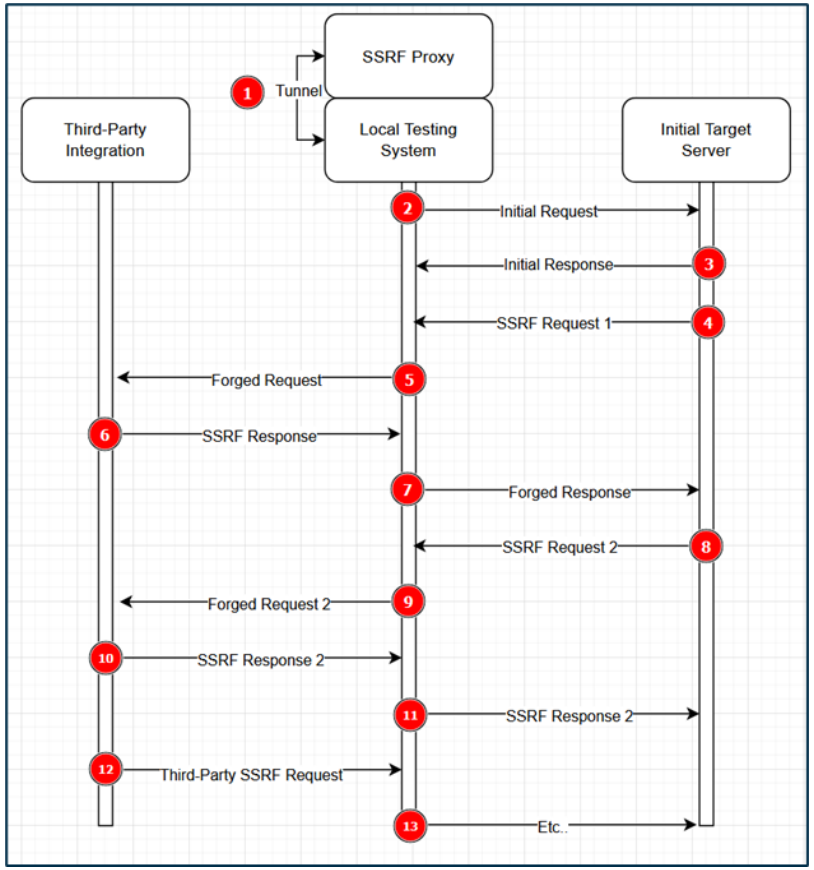
- Problem: Collaborator Constraints
  - Collaborator behavior echoing subdomain string
  - Custom response structure is required



Area of Interest in Steps 7-8

Case Study: SSRF in Downstream Async Flows

1. SSH tunnel is established to SSRF proxy
2. Initial request containing SSRF payload is sent
3. Synchronous response returned by target
4. Target server performs SSRF request
5. Forged request is forwarded to the third party
6. Third party response intercepted by SSRF proxy
7. SSRF proxy tampers with third-party response
8. Target performs secondary request
9. Second forged request is tampered with by proxy
10. Third party provides second response
11. SSRF proxy forwards third-party response to target
12. Third-party performs SSRF to the location injected in step 9



SSRF in Downstream Third-Party Workflow

Case Study: SSRF in Downstream Async Workflows

```python
# Relay an intercepted request to the target
@serve.route('/intercept/<destination:re:.*>', method='POST')
@serve.route('/intercept/<destination:re:.*>', method='GET')
def intercept(destination: str):
    target_host = destination.split('/')[2]

    # Trim out request URI authentication and service port if exists
    at_idx = target_host.find('@')
    colon_idx = target_host.find(':')
    start = at_idx if (at_idx != -1) else 0
    end = colon_idx if (colon_idx != -1) else len(target_host)

    headers = dict(request.headers)
    headers['Host'] = target_host[start:end]

    logger.info(f'Relaying to {target_host}')
    try:
        res = requests.request(request.method, f'{destination}?{request.query_stri
        response.status = res.status_code
        response.headers = res.headers

        for key, value in res.headers.items():
            response.set_header(key, value)


    except Exception as err:
        logger.warning(f'Exception thrown during interception relay: {err}')
        return 'failure'

    logger.debug(f'Relayed response headers: {res.headers}')
    logger.debug(f'Relayed response body: {res.text}')

    return res.content
```

SSRF Interception Handler / Proxy [3]

- Lightweight HTTP API to serve as a web proxy
  - Essentially creating a machine-in-the-middle (MitM) situation
  - Hook back through Burp Suite to allow for on-the-fly tampering with request & response contents
- Various other useful endpoints
  - Authorization header echoing
  - Request logging
  - Response code & redirect fuzzing

Demo Example & Walkthrough

Recorded Demonstration

- SSRF Identification
- Transparent vs Semi-Blind SSRF Example
- Downstream Workflow Dependency Problems
- Interception and Traffic Proxy
- Injection into Forged Responses
- Downstream Vulnerability Exploitation

Questions & Answers

References

[1] PortSwigger Academy, Server-side request forgery (SSRF)

- https://portswigger.net/web-security/ssrf

[2] Wikipedia, Uniform Resource Identifier

- https://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Syntax

[3] Github, SSRFReceiver Repository

- https://github.com/vexance/SSRFReceiver/tree/main