# Open Redirect

An **Open Redirect** occurs when a web application **blindly redirects users** to a URL specified via **user input**, without validating or sanitizing it.

This allows attackers to:

* Redirect users to **malicious or phishing domains**
* Bypass security controls
* Chain into **credential theft, token harvesting**, or **XSS**

# Common Parameters

?url=

?redirect=

?next=

?return=

?dest=

?continue=

# Impact

| Risk | Description |
| --- | --- |
| Phishing | Redirect to a fake login or malicious site from a trusted domain |
| Token Theft | Steal OAuth access tokens via manipulated redirect\_uri |
| Bypass Filters | Evade CSP, SSO, or redirect protections |
| Chained Attacks | Combine with XSS, session fixation, etc. |
| Social Engineering | Trick user into clicking trusted URL that bounces to malware |

# Example

<https://example.com/login?next=https://phish-portal.com>

User clicks thinking it’s safe → redirected to phishing site.

# Payload Examples

| Payload | Goal |
| --- | --- |
| ?url=http://evil.com | Direct open redirect |
| ?redirect=//evil.com | Scheme-relative redirect |
| ?next=https:evil.com | Bypass via malformed scheme |
| ?continue=https://evil.com%23@real.com | Host confusion |
| ?r=https://example.com@evil.com | User sees example.com but lands at evil.com |

# Detection Techniques

| Method | Description |
| --- | --- |
| Manual Testing | Replace redirect values with http://attacker.com and observe |
| CSP / OAuth Abuse | Test redirect\_uri in SSO and 3rd party integrations |
| DNS/TLS Observations | Check where the browser ends up |
| Burp Suite | Intercept login → test redirect URL parameters |
| FFuF / ParamSpider | Discover hidden redirect parameters |

# Mitigation

## 1. Enforce Redirect Whitelisting

Only allow specific internal paths:

if redirect\_url not in ["/dashboard", "/home"]:

redirect\_url = "/home"

## 2. Use Relative Paths Only

Don’t allow full URLs in redirect params:

* ✅ /dashboard
* ❌ http://evil.com

## 3. Encode and Validate Redirects

* Reject URLs with:
  + Protocols (http://, https://)
  + Special chars (@, //, :)
* Normalize and resolve path:

parsed = urlparse(url)

if parsed.netloc: # block external domains

reject()

## 4. Harden OAuth Redirects

* Validate redirect\_uri against **exact match**
* Use **state** parameter for CSRF prevention
* Never allow wildcards (\*.example.com)

## 5. Educate Users

* Never blindly trust redirects after login
* Use **browser warnings** if possible

# Points

“Open Redirects **don’t harm the server directly**, but they’re dangerous because they **break user trust** and enable **phishing or OAuth abuse**.”

“It becomes critical when used in **redirect\_uri of OAuth** or **SSO flows**, where tokens may be stolen.”

“Fixes involve using **relative path-based redirects** and strict **whitelisting** of allowed endpoints.”