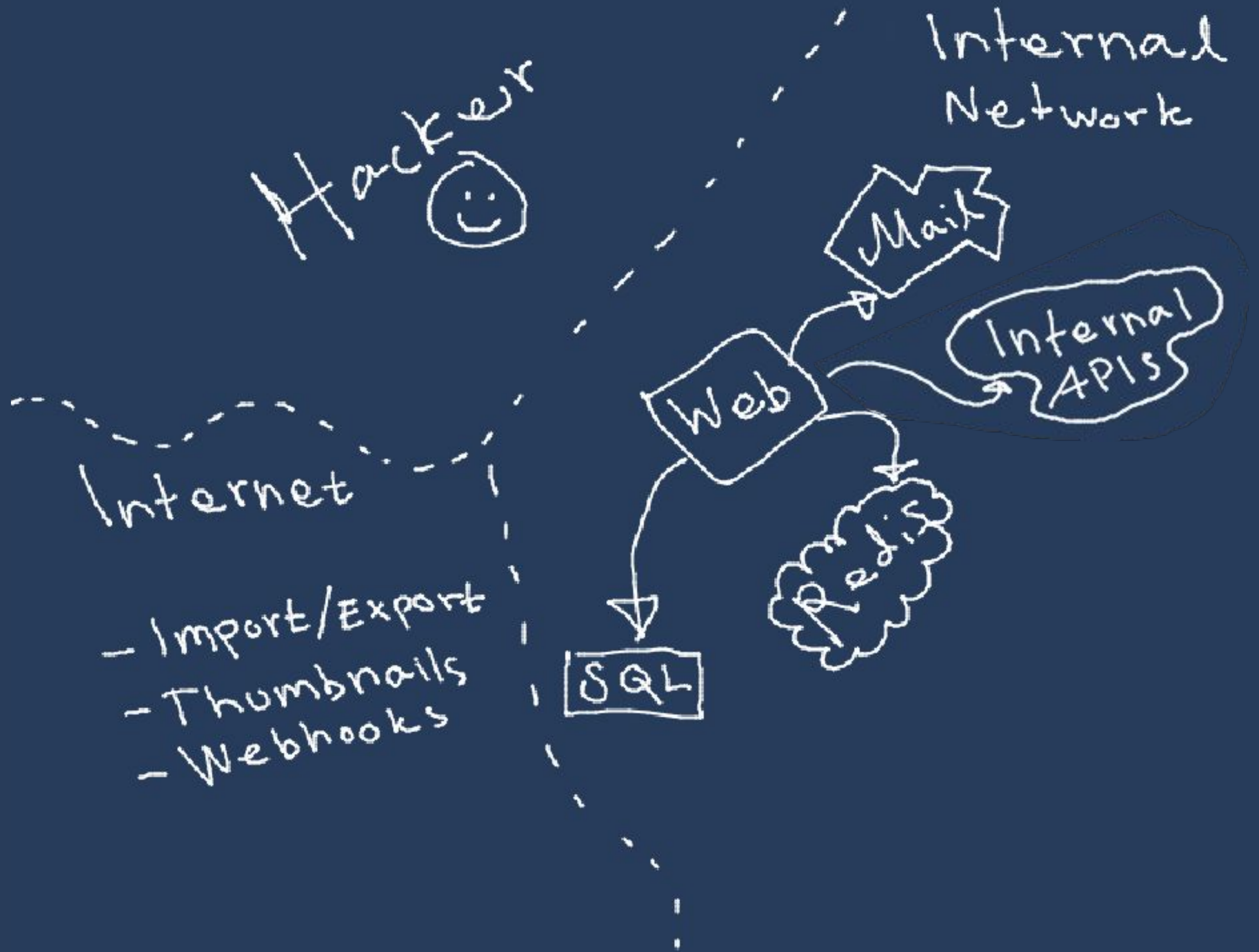
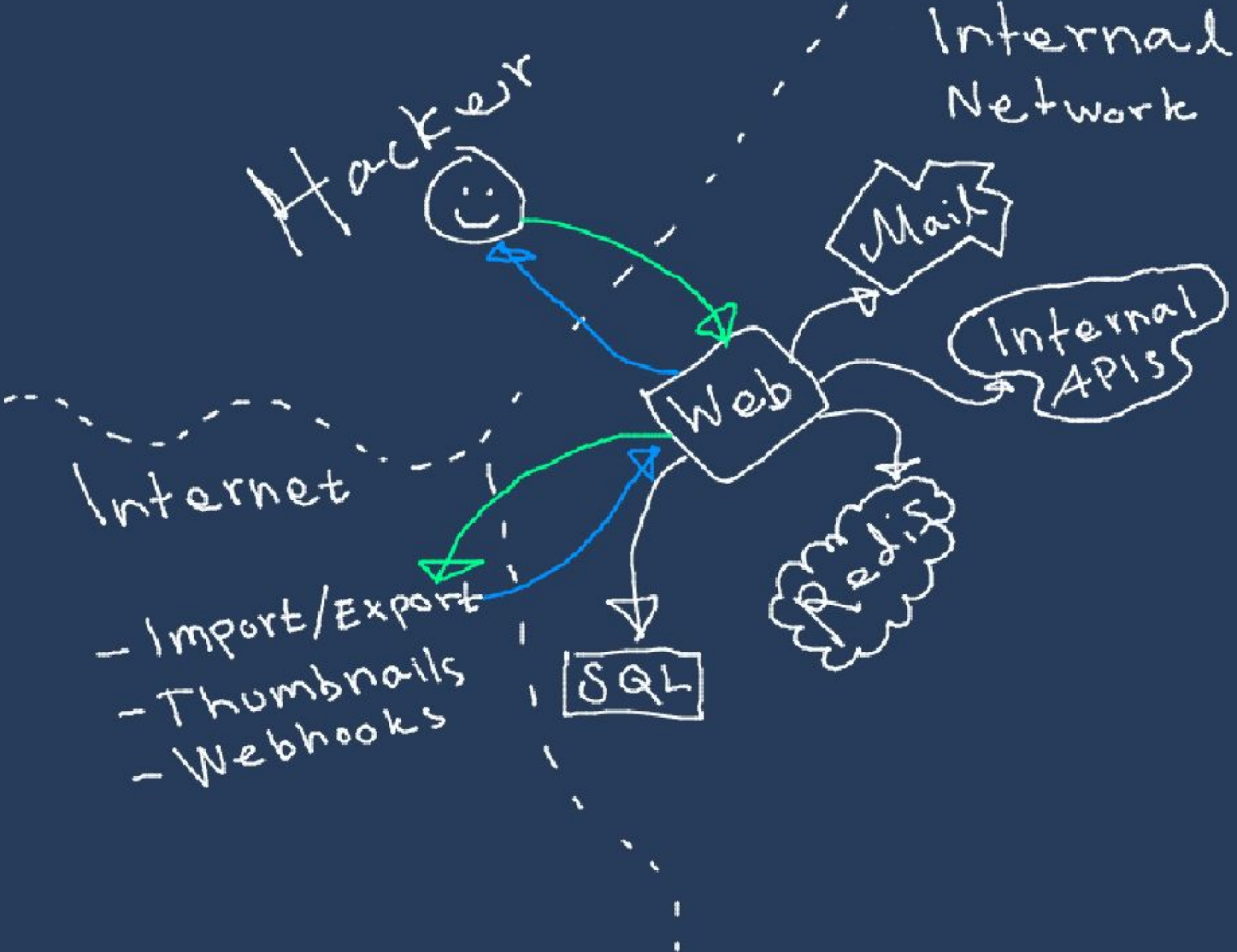
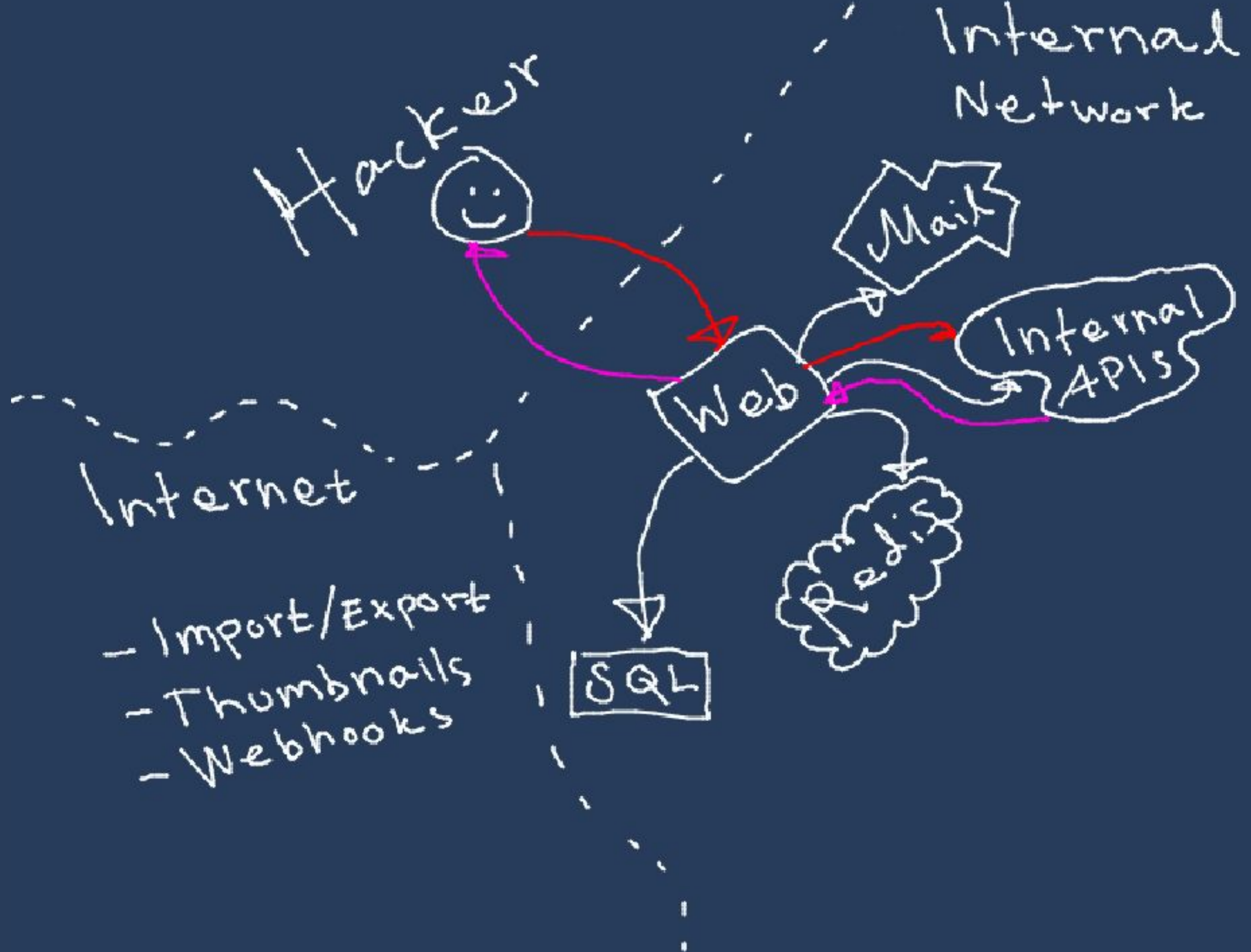
Server-side Request Forgery


Hacker
Internal Network
Internet
- Import/Export
- Thumbnails
- Webhooks
Web
Mail
Internal APIS
Redis
SQL


Internal
Network
Hacker
Mail
Internal
APIS
Web
Internet
Redis
SQL
-Import/Export
-Thumbnails
-Webhooks



- Import/Export
- Thumbnails
- Webhooks

Attack vectors

- Import/Export
- Thumbnailer
- Webhooks
- Media converters
- FTP-servers (!?)
- much more!

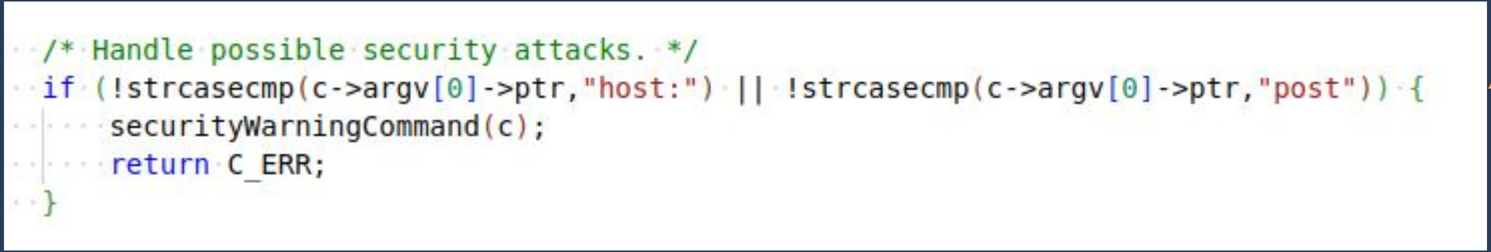
Targets

- Cloud metadata API
- Internal APIs/Services
- Cross protocol targets

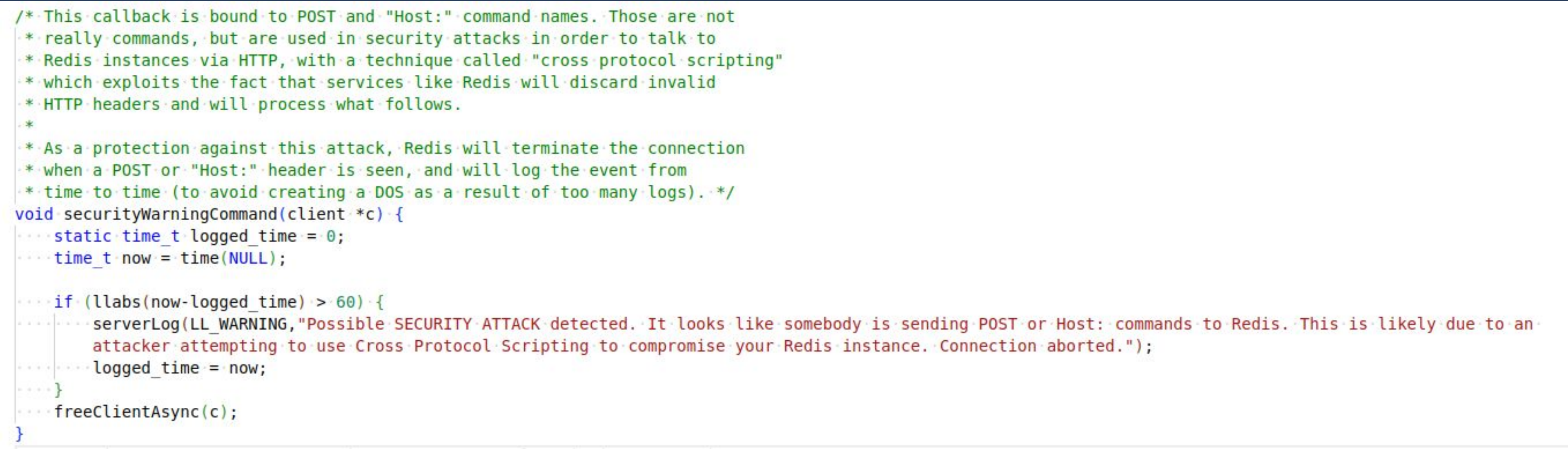
Extra trickery

- Redirects
- IP encoding (127.0.0.1 == 2130706433)
- DNS rebinding + protocol state
  - TLS poisoning
  - Cookies?

CPT (cross protocol trickery) protection in Redis



```
/* Handle possible security attacks. */
if (!strcasecmp(c->argv[0]->ptr,"host:") || !strcasecmp(c->argv[0]->ptr,"post")) {
    securityWarningCommand(c);
    return C_ERR;
}
```



```
/* This callback is bound to POST and "Host:" command names. Those are not
 * really commands, but are used in security attacks in order to talk to
 * Redis instances via HTTP, with a technique called "cross protocol scripting"
 * which exploits the fact that services like Redis will discard invalid
 * HTTP headers and will process what follows.
 *
 * As a protection against this attack, Redis will terminate the connection
 * when a POST or "Host:" header is seen, and will log the event from
 * time to time (to avoid creating a DOS as a result of too many logs). */
void securityWarningCommand(client *c) {
    static time_t logged_time = 0;
    time_t now = time(NULL);

    if (llabs(now-logged_time) > 60) {
        serverLog(LL_WARNING,"Possible SECURITY ATTACK detected. It looks like somebody is sending POST or Host: commands to Redis. This is likely due to an
        attacker attempting to use Cross Protocol Scripting to compromise your Redis instance. Connection aborted.");
        logged_time = now;
    }
    freeClientAsync(c);
}
```

TLS Poisoning

TLS Session IDs are random values that reference cached DH-results

The SID is stored between connections in clients like curl

Who says this random value can’t be a string like “\nSET x 10\n…”?

But this is kind of lame on its own… We want to send this attacker-controlled value to an internal resource.

TLS Poisoning + DNS Rebinding!

> curl https://attacker.com/stage/1

DNS: attacker.com = 13.3.3.37, TTL=0

TLS: Session id = “\nSET x 10\n…”

> curl https://attacker.com/stage/2

DNS attacker.com = 10.10.10.10, TTL=0

Client sends “\nSET x 10\n” to the internal server

TLS Poisoning + DNS Rebinding! Alt. strat

> curl https://attacker.com/stage/1

DNS: attacker.com = 13.3.3.37 AND 10.10.10.10

TLS: Session id = “\nSET x 10\n…”

> curl https://attakcer.com/stage/2

curl: 13.3.3.37 is down, trying 10.10.10.10

Client sends “\nSET x 10\n” to the internal server

Why are FTP-Servers interesting?

SSRFs are literally built in

Passive mode: client connects to the server’s data-port

Active mode: server connects to the client’s data-port (?!)

1. Upload TCP-packet as a file to server
2. Ask the server to send the file to another server with PORT+RETR

More of a CTF-goof in bad impls, than something you will see in the real world.

Real-world example: SSRF in Slack

https://hackerone.com/reports/671935

https://hackerone.com/reports/878779

Simple exercise! Work in groups (:

http://46.101.147.11:3000

(down if you are reading after the meetup)

Further reading

- Gentle introduction https://portswigger.net/web-security/ssrf
- TLS Poisoning https://www.youtube.com/watch?v=udpamSmD_vU
- IMDSv1 vs IMDSv2 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
- Advanced SSRF-Writeup https://x-c-3.github.io/posts/maplectf-2022-art-gallery/