## Аннотация

Данный текст написан 2025-11-30 за 3 дня автором nesca4 на русском языке, и переведен тем же днем ChatGPT на английский. Вероятно, в оригинальной версии текста не сложно найти орфографические ошибки; кроме того, текст написан в виде трактата: все это обусловлено тем, что мне пришлось работать над ним в спешке. Впрочем, я очень надеюсь что даже так он послужит на пользу.

lomaster 2025.

# INITIVM

**(I).** Итак, несмотря на то, что разработка nesca4 уже оффициально не ведется, мне продолжают поступать вопросы ее касающиеся; в силу этого, я решил написать оный трактат, который ставит своей целью рассказать некоторую информацию о nesca4.

Вообще, я писал nesca4 в молодости, как проект чтобы научится писать код, в следствие чего она несколько раз полностью переписывалась; впрочем, последняя версия не совсем дописана. Первая версия nesca4 была написана где-то зимой 2023 года; весной того же года ее разработка стала публиковатся в телеграм канале «OldTeam»; 31 июля была опубликована в телеграм канале «Точка сбора». Должно уточнить: автор не имеет отношения к авторам nesca3, но, возможно, сам того не зная пересекался с ними.

По-видимому, проект nesca4 сейчас врятли удивляет, но тогда, относительно моих способностей и возвраста удивителен; я даже раньше любил хвастатся, говоря что nmap-у потребовалось 25 лет на то, на что мне 1 год (насколько это правда, я сам судить не буду). Конечно, можно еще раз ее переписать, сделать очень да-

же хорошей, но честно я уже устал и хочу ее оставить: пускай будет такой, какая есть.

Что касается libncsnet (раньше ncsock) (ncs, n — nesca), то это библиотека со временем вылившиеся из nesca4, подобно libpcap из tcpdump: уж так бывает, что со временем в коде программы накапливается много такого, что было бы хорошо вынести в библиотеку от нее отдельную.

Вообще, разработка libncsnet велась чаще чем nesca4, но уже тоже оставлена. И хотя большая часть ее кода вначале была взята из nmap и libdnet, и код является довольно однотипным, но содержащийся в ней интерфейс raw.h и сейчас я считаю довольно-таки искусным решением.

**(II).** Итак, если подвести то, что сейчас умеет nesca4, — и сделать это кратко — будет следующее:

**i**. Сканировать порты методами: (TCP): syn, xmas, null, fin, psh, maimon, window, ack; (SCTP): init, cookie; (UDP): udp.

**ii**. Выполнять пинг методами: (TCP): syn, ack; (ICMP): echo, timestamp, info; а также: arp, udp.

**iii**. Брутфорсить и сканировать сервисы: HTTP basic auth, FTP. **Раньше** еще была воз-

можность: Hikvision, RVI, SSH, RTSP.

**iv**. Пробивать данные по базе данных (такая функция опознания была и в nesca3).

**v**. Сохранять результаты в HTML (как и nesca3).

**vi**. В качестве цели принимать: CIDR4, DNS, IPv4, RANGE4.

Старая версия (до последнего переписывания) которая поддерживает брутфорс камер RVI и Hikvision. Но она довольно плоха, и это руководство к ней не подходит, может лишь чуть-чуть.

**(III).** Перед использованием должно учитывать следующее:

**i**. nesca4 довольно сложна в использовании, работает только на Linux, и совсем не рассчитана на простого пользователя.

**ii**. nesca4 сделана для более «профессиональных» вещей нежели nesca 3. Она рассчитана в первую очередь на различные сканирование портов, нежели на брутфорс, однако поддерживает брутфорс.

**iii**. nesca4 во многом похожа на nmap: у нее отличаются «flags», но обычно лишь графически.

**iv**. Единственное руководство по nesca4 до этого — ее «usage menu»: это такая страшная

стена текста с опциями типа: (`-pps <pps> set your pps holy shit abcd 1234`); она выводится при пустом запуске nesca4, или при запуске с флагом -help: `./nesca4 -help`. Впрочем, когда-то давно (что можно найти в коммитах на github) в resources/ был файл «do_not_read.txt», где было что-то вроде документации.

**v**. Не стоит ждать от нее точно то же, что и от nesca 3.

**(IV).** Прежде чем запустить nesca4, ее необходимо скомпилировать, т. е. собрать из исходного кода.

**1.** Для этого нужно иметь на своем Linux: g++, gcc, make, и git. Если их нет, то должно установить: вероятно, это не сложно.

**2.** Если все это наличиствует, для компиляции следует последовательно выполнить следующие четыре команды в терминале Linux:

```
cd ~

git clone --recurse-submodules --depth=1 \
    https://github.com/oldteamhost/nesca4

cd nesca4
```

```
./configure

make -j
```

**3.** Если компютер не справляется с компиляцией, вместо `make -j` напишите просто: `make`.

Если компиляция все равно зависает, то следует внутри директории `nesca4`, (куда вы попали написав `cd nesca4`), прописать пять следующих команд:

```
./configure

cd libncsnet

make CFLAGS="-fPIC -DHAVE_CONFIG_H \
            -flto -I."

cd ..

make CFLAGS=""
```

**4.** По моим тестам, компиляция успешно проходит на Arch Linux и Ubuntu.

**5.** Если компиляция прошла успешно, в директории `nesca4` появится файл `nesca4`; чтобы убедится в его наличии, можно написать `ls`: эта

команда выведит все файлы в текущей директории. Узнать текущую директорию можно написав: `pwd`; должно быть что-то вроде:

```
/home/<user>/nesca4
```

# BASIS

**(I).** Итак, nesca4 — это консольная утилита для получения различной информации о сетевых хостах; сканирование — это процесс получения это информации. Сама nesca4 стремится к тому, чтобы предоставлять как можно быстрее, красивее, точнее, и больше информации о сетевых хостах.

Любой сеанс использования nesca4 выглядит следующим образом: (i) запустить nesca4 передав ей при запуске некоторые «входные данные», (ii) получать результаты ее работы до тех пор, пока она не завершиться.

Очень важно заметить: nesca4 будет выполнять свою работу исходя из того, какие «входные данные» мы передали ей при запуске. Именно посредством этих «входных данных» производится ее настройка, передача целей сканирования, и т. д. Разные «входные данные» = разная работа nesca4.

**(II).** Любой запуск nesca4 выполняется с помощью ввода в терминал команды следующего вида:

```
./nesca4 "входные данные"
```

Техническое уточнение: nesca4 сможет полноценно работать только в том случае, если запущена от прав root; команда наверху предполагает что вы являетесь root пользователем, однако в большинстве случаев это не так, из-за чего к команде приходиться добавлять `sudo`:

```
sudo ./nesca4 "входные данные"
```

Можно также просто написать,

```
su
```

и все следующие команды автоматически будут от прав root (благодаря чему добавлять `sudo` не придется).

Далее, мы также будем опускать `sudo`, как и сделали выше.

**(III).** Входные данные бывают следующих трех типов:

**1.** Цели сканирования — это такие входные данные, которые представляют собой хосты для сканирования; их можно передать через пробел в формате IPv4, DNS, CIDR4, RANGE4:

```
./nesca4 google.com 104.237.160.0/19
```

```
./nesca4 77.88.55.88 youtube.com
./nesca4 104.237.160.0-104.237.191.255
```

**2.** Флаги (или опции, аргументы) — это такие входные данные, которые что-то указывают: например то, как следует выполнять работу, что использовать, и т. д.; кроме того, флаги могут указывать даже хосты для сканирования, но не напрямую как было выше, а немного иначе, скорее косвенно.

Для того, чтобы отличать флаги от целей сканирования, они указываются через (-), например:

```
./nesca4 -syn -pe -p 80 google.com
```

Среди этих трех флагов выделяется флаг (`-p 80`), так как он имеет свой «параметр», в данном случае — 80.

Вообще, флаги бывают двух видов: (1) те, что имеют «параметр», и (2) те, что не имеют «параметр». Как видно выше, «параметр» любого флага указывается через пробел после самого этого флага: (`-flag param`). Причем важно заметить: «параметр» флага может быть только один (именно поэтому `google.com` выше не считается «параметром» флага `-p`).

Впрочем, «параметр» флага можно указать и вот так:

```
-p80 google.com
```

В этой записе, 80 — это также «параметр» флага `-p`, и `google.com` до сих пор не является его «параметром».

Многие программы, но не nesca4, различают короткие (односимвольные), и длинные (двух или более символьные) флаги; причем длинные в них указываются через два (−−), а короткие через один (-):

```
--flag
-f
```

Nesca4 ради простоты так не делает.

**3.** Стандартный файл настройки — это такие входные данные, которые являются файлом настройки, и передаются (даже если вы этого не видите) при каждом запуске nesca4, если конечно не указан другой файл настройки с помощью флага `-cfg` (далее мы поговорим и об этом флаге, и о файлах настройки). Этот файл находится по пути: `resources/config/default.cfg`.

**(IV).** Итак, мы запустили nesca4 передав ей входные данные; далее, как сказано выше, «мы получаем результаты ее работы до тех пор, пока она не завершиться».

По умолчанию, nesca4 дает результаты прямо в терминал, однако может сохранять их и в HTML формате подобно nesca3. Сначала поговорим о первом.

**1.** Итак, вначале nesca4 выводит «start line»: это строка сообщающая о запуске nesca4; она содержит версию, время и дату запуска:

```
Running NESCA4 (v20240926) time
        05:52:30 at 2025-11-29
```

**2.** Далее, после этой строчки, nesca4 начинает выводить результаты своей работы (т. е. начинается основая часть вывода).

Все результаты своей работы nesca4 выводит блоками: результат сканирования одного хоста = один блок; такие блоки мы будем называть — «блоки-результаты». Вот пример двух таких:

```
Report nesca4 for 142.250.74.46 ...,

  ports  '80/tcp/open/http(syn)'

Report nesca4 for 77.88.44.55 ...,
```

```
  ports  '80/tcp/open/http(syn)'
```

**3.** Наконец, после вывода всех результатов, перед самым завершением своей работы nesca4 выводит «end line»: это строка сообщающая о завершении nesca4; она содержит общее количество доступных хостов, и общее время работы:

```
NESCA4 finished 2 up IPs (success)
                    in 201.60 ms
```

**(V).** Любой блок-результат выводимый nesca4 состоит из «header», и из нуля или более «nodes»:

**1.** «header» — это единственный обязательный элемент блока-результата, который представляет собой заголовок этого блока; в своей самой полной форме он содержит: IP адрес цели, MAC адрес цели, DNS цели, методы используемые на цели, и время ответа (RTT) цели:

```
Report nesca4 for 1.1.1.1 (dns.com)
        [43:2a:61:5b:89:71] '.eS' 8.51 ms
```

Вероятно, не совсем понятно то, что значит: «методы используемые на цели» (хотя это совершенно не важный элемент):

```
'.eS'
```

По сути, это строка формата: `'.<methods>'`, где `<methods>` — это последовательность букв, каждая из которых обозначет используемый на этой цели метод сканирования портов или пинга. Например, если мы включили ACK & ICMP пинг и ACK сканирование, строка будет такой: `.eaA`. Полное соответствие букв, если то надо, есть внутри кода в функции `strmethod()`.

Как было сказано, «header» может содержать MAC адрес цели; это происходит в том случае, если он был получен при ARP пинге.

Также, было сказано то, что DNS и время ответа (RTT) может быть не одно, но несколько, касательно последнего: один успешный пинг = один RTT.

**2.** «node» — это такой элемент блока результата, который содержит какую-то информацию о цели этого блока; он не является «header», и не является обязательным, хотя «nodes» и содержат самую главную информацию типа: открытых портов, логина и пароля, сервисах (HTTP, FTP) и т. д.

Каждый «node» имеет формат (`<info type> <info>`):

```
ports    '80/tcp/unfiltered/http(ack)'
```

```
http(title)     google
```

В последнем примере, <info> это — google, а <info type> это — http(title).

Формат <info> может быть очень различным, в случае портов (ports) это довольно-таки тяжелый формат:

```
format = '<port>','<port>',...,

port = <num>/<protocol>/
       <status>/<service>(<method>)

protocol = "tcp" | "sctp" | "udp" | "???"

status = "open" | "closed" | "filtered" |
       "error" | "open|filtered" |
       "unfiltered" | "???"

method = "syn" | "xmas" | "fin" | "ack" |
       "window" | "null" | "maimon" |
       "psh" | "init" | "cookie" |
       "udp" | "???"
```

Формат <info> для результатов брутфорса следующий:

```
login:pass@
```

Вообще, если говорить правильно, то «node» результатов брутфорса и сервисов это одно и то же: «node» с HTTP заголовком, и «node» с паролем и логином от HTTP страницы — суть «nodes» сервисов, в данном случае сервиса HTTP.

Сервисы лежат в файле resources/nesca-services, именно используя эту базу nesca4 определяет сервис.

Формат «node» для результатов опознания по базе данных следующий:

```
db(<key>)    <info>
```

Здесь, `<key>` это то, что nesca4 нашла в информации о цели, чтобы заключить о ней `<info>`. Например, если nesca4 в HTTP перенаправлении нашла слово «login», то она заключит что это страница аунтификации:

```
db(login)    auth
```

База данных с таким опознанием находится

в resources/nesca-database; если разобрался, то ее можно менять самому. Она довольно неплоха, учитывая что поддерживает regex.

Вообще, формат там такой:

```
Default:
'<key>','<info>',0,<where looking?>;

Regex:
R'<key>','<info>',0,<where looking?>;

<where are we looking?> =
    FIND_ALL  -1
    FIND_HTTP_REDIRECT 0
    FIND_HTTP_TITLE  1
    FIND_HTTP_HTML  2
    FIND_MAC  3
    FIND_DNS  4
    FIND_IP   5
    FIND_FTP_HELLO  6
```

Например:

```
    'kek','is kek',0,1
```

Формат «node» для сервисов следующий:

```
<service>(type)     <info>
```

Здесь, `<service>` — это имя сервиса, например `http` или `ftp`, а `<type>` — это наконец сам тип информации об этом `<service>`.

**(VI).** Наконец, для сохранения результатов в HTML можно использовать флаг `-html <file>`, который принимает в свой параметр путь к файлу (имя файла) HTML:

```
-html myfile.html
```

Этот файл следует открывать с помощью браузера; это можно сделать прямо из терминала с помощью команды `firefox` (если у вас он установлен):

```
firefox myfile.html
```

Или Google Chrome:

```
google-chrome myfile.html
google-chrome-stable myfile.html
```

Если вы прочитали главу выше, формат сохранения в HTML будет вам знаком.

Вообще, стиль CSS для HTML файлов которые создает nesca4 находится в resources/style.css.

**(VII).** Кроме настройки через флаги, nesca4 может настраиваться через файлы настройки. По сути, это такая же настройка через флаги, но она лежит в файле, и имеет немного другой формат.

**1.** О стандартном файле настройки мы уже сказали выше, уточним лишь то, что кроме настройки по умолчанию, внутри этого файла есть небольшое руководство по созданию файлов настройки; здесь мы его опустим.

**2.** Чтобы указать другой файл настройки, можно использовать флаг `-cfg file`, который принимает в параметр путь к файлу настройки который следует включить:

```
-cfg resources/config/insane.cfg
```

Когда вы указываете свой файл настройки, nesca4 перестает автоматически использовать стандартный файл настройки.

# NVCLEVS

**(I).** По сути, вся работа nesca4 заключается в следующем: (i) получить цели сканирования, (ii) просканировать их, (iii) дать результаты, (iv) либо повторить i-iii, либо — если все цели просканированы и результаты даны — завершится.

Спрашиваю: как nesca4 сканирует переданные ей цели? Отвечаю: nesca4 для каждой полученной цели выполняет 6 стадий сканирования последовательно.

Эти 6 стадий суть таковы: (1) пинг сканирование, (2) получение DNS имен, (3) сканирование портов, (4) сканирование сервисов, (5) опознание по базе данных, (6) брутфорс.

Когда цель проходит эти 6 стадий, мы постепенно получаем о ней информацию, причем информацию полученную на одной стадии, nesca4 может использовать на другой: например, время ответа цели (RTT) nesca4 получает на первой стадии, а затем использует его на четвертой для расчета таймаута. Истинно: это расширенная схема работы nmap.

Чтобы nesca4 выдала хоть что-то, необходима хотя бы первая, вторая, или третья стадия;

их можно назвать — главными.

Впрочем, не смотря на то, что первая и вторая по умолчанию включены, их методы по умолчанию не выбраны, поэтому де-факто они все же отключены. Такая же ситуация и с четвертой стадией, она тоже по умолчанию включена, но де-факто выключена, ведь порты для ее работы по умолчанию не указаны.

Важно понимать: некоторые стадии могут зависить друг от друга, и поэтому могут иногда быть, а иногда не быть: например, если ни одна цель не пройдет первую стадию, остальные даже не начнутся.

Текущая стадия сканирования отображает-

ся в статус баре внизу:

```
/ PASSED 44    PORTS SCANNING
            2.55 KiB  206.00 B
```

Впрочем, некоторые стадии выполняются настолько быстро, что вы даже не успеете увидить их в статус баре.

Значение `passed` это то, сколько целей было просканировано.

Остальные два значения, это количество трафика получаемого на вашу сетевую карту, и отправляемого с вашей сетевой карты. Оно просто читает соответствующие файлы из `/sys/class/net/`.

Некоторые флаги (все они не имеют параметра) вовсе позволяют их отключить: флаг `-n-ping` — позволяет отключить первую стадию, флаг `-n` — вторую, флаг `-sn` — третью, флаг `-n-db` — пятую, флаг `-n-brute` — шестую. Для четвертой стадии флага нет.

**(II).** Итак, далее будем приводить пример
400 команды, и затем его пояснять.

**1.** Привожу:

```
./nesca4 -html out.html -syn \
```

```
-pe -p 80,443 google.com
```

Поясняю: такой вызов nesca4 сможет дать нам следующую информацию о нашей единственной цели `google.com`: (1) статус TCP портов 80 и 443, (2) время ответа, (3) DNS, (4) его IP адрес, (5) опознание по базе данных.

Флаг `-html out.html` — понятен, так как мы уже говорили о HTML сохранении.

Флаг `-syn` — включает SYN метод для сканирования портов; благодаря этому мы сделаем третью стадию рабочей.

Флаг `-pe` — включает ICMP echo метод для пинг сканирования; благодаря этому мы сделаем первую стадию рабочей.

Флаг `-p 80,443` — устанавливает порты для сканирования, а именно: 80 и 443 оба TCP.

> Команда выше указана несколько странно:

во-первых, я перенес ее часть на другую строку, во-вторых, на предыдущей строке поставил \.

Первое я сделал для того, чтобы она поместилась в страницу, а второе для того, чтобы она работала в терминале если ее скопировать и вставить.

Терминал (вернее shell в нем запущенный), если вы переносите команду на следующую строку, требует от вас на предыдущей поставить \, иначе он подумает что команда на следующей строке это не продолжение старой, а новая команда.

Т. е., если я сделаю просто,

```
./nesca4 -html out.html -syn
    -pe -p 80,443 google.com
```

то терминал подумает что тут две команды. Но если поставить \ как выше, то он поймет что это одна команда.

Кроме просто перечисления портов через за-

пятую, с помощью `-p` можно указать диапазон портов:

```
-p 80-85
```

Запись выше эквивалентна следующей:

```
-p 80,81,82,83,84,85
```

**2.** Привожу:

```
./nesca4 -html out.html -n -syn \
        -s http:80,8000,8080,8888 \
        -pe -p 80,8000,8080,8888 \
        -random-ip 40000
```

Поясняю: такой вызов nesca4 уже довольно-таки объемный: во-первых, мы сканируем 40000 случайных IPv4 адресов, во-вторых, получаем намного больше информации, ведь в силу флага `-s` мы сделали четвертую стадию рабочей, в третьих, мы указали намного больше портов.

Вся информация о цели, которую мы можем получить при таком вызове, это: (1) статус TCP портов 80, 8000, 8080, 8888, (2) время ответа, (3) IPv4 адрес, (4) HTTP ответ, (5) HTTP перенаправление, (6) HTTP логин и пароль от стра-

ницы, (7) опознания по базе данных, (8) HTTP заголовок.

Такое сканирование прекрасно подходит для поиска всяких «тайных сайтов», ведь то обстоятельство что у хоста открыт порт 80, 8000, 8080, или 8888, почти всегда говорит о том, что это HTTP страница (т. е. сайт). Также, большую роль здесь играет еще и опознание по базе данных: благодаря нему можно понять, какая страница «типична», а какая уже что-то иное. Да и другая информация может быть полезна для этого.

Флаг `-n` — отключает вторую стадию сканирования; в целом, DNS не самая нужная вещь, но ее получение занимает много времени, поэтому, за счет этого флага можно неплохо ускорить сканирование.

Флаг `-random-ip 40000` — указывает nesca4 использовать 400000 случайных IPv4 адресов в качестве целей сканирования; важно заметить: это не мешает указать вам и другие цели.

Флаг `-s http:80,8000,8080,8888` — указывает nesca4 то, что порты 80, 8000, 8080, и 8888 следует воспринимать как порты на которых есть HTTP сервис, вследствие этого, мы сделаем четвертую стадию рабочей, ведь она теперь будет иметь порты и сервис с которыми ей сле-

дует работать.

> По умолчанию, nesca4 обрезает очень длинные сервисные «nodes», например «node» с HTTP ответом. Если вы хотите это отключить, есть флаг:
>
> ```
> -detal
> ```

**3.** Привожу:

```
./nesca4 -html out.html -n -syn -pe \
        -pa 21,80 \
        -s http:80,8000,8080,8888,ftp:21 \
        -p 80,8000,8080,8888,21 -v \
        -import ip.txt
```

Поясняю: эта подобна команде выше, но теперь мы берем цели сканирования из файла `ip.txt` (можете класть туда DNS, IPv4, CIDR4, RANGE4), получаем информацию о FTP, используем еще один метод пинг сканирования, и выводим много логов.

Вся информация о цели, которую мы можем получить при таком вызове, это: (1) статус TCP портов 80, 8000, 8080, 8888, 21 (2) время ответа,

(3) IPv4 адрес, (4) HTTP ответ, (5) HTTP перенаправление, (6) HTTP логин и пароль от страницы, (7) опознания по базе данных, (8) HTTP заголовок, (9) FTP заголовок, (10) FTP логин и пароль от сервера.

Флаг `-import ip.txt` — указывает nesca4 то, что цели сканирования следует брать из файла `ip.txt`.

Флаг `-v` — включает отображение откладочной информации, это превратит терминальный вывод nesca4 скорее в логи, чем вывод результатов, однако, предполагается что результаты вы смотрите в `out.html`.

Флаг `-pa 21,80` — влючает TCP ACK метод пинг сканирования (до этого у нас был только ICMP echo), причем указывает ему использовать порты 21 и 80.

**4.** Привожу:

```
./nesca4 -html out.html -n -syn -pe \
        -pa 21,80 -login l.txt -pass p.txt \
        -s http:80,8000,8080,8888,ftp:21 \
        -p 80,8000,8080,8888,21 -v \
        -onlyopen -import ip.txt
```

Поясняю: Точно такая же как команда выше, но теперь логины и пароли для брутфорса

мы берем из своих файлов, и даем результаты только для целей с открытыми портами.

Флаг `-onlyopen` — указывает nesca4 выводить и сохранять только те цели, которые имеют открытые порты. Очень полезно чтобы откинуть «мусор».

Флаг `-login l.txt` — указывает nesca4 то, что логины для брутфорса следует брать из файла `l.txt`.

Флаг `-pass p.txt` — указывает nesca4 то, что пароли для брутфорса следует брать из файла `p.txt`.

По умолчанию, nesca4 берет логины и пароли из файлов:

```
resources/login.txt
resources/pass.txt
```

**(III).** Итак, теперь поговорим о том, как увеличить скорость сканирования, и что на это влияет. Важно заметить: все эти способы не только повышают скорость, но вместе с этим и понижают точность сканирования. Способы таковы:

**1.** Как уже было сказано выше, повысить скорость сканирования можно отключением вто-

рой стадии которая не так уж и полезна; это можно сделать с помощью флага `-n` (он не имеет параметр). Важно заметить: отключение не второй но первой стадии, может напротив, только замедлить сканирование.

**2.** Далее, повысить скорость сканирования можно за счет применения как можно меньшего количества методов пинг сканирования, и сканирования портов. Впрочем, комбинации `-pe -syn` обычно всегда достаточно.

**3.** Далее, повысить скорость сканирования можно за счет уменьшения времени ожидания пакетов при первой стадии: дело в том, что если время ожидания (таймаут) для третьей стадии nesca4 расчитывает (что безумно увеличивает скорость), то для первой стадии таймаут всегда один, так как мы еще не имеем данных чтобы его расчитать. По стандарту, это: 800 милисекунд.

Чтобы его понизить, можно использовать флаг `-wait-ping`, который принимает в параметр новое значение времени ожидания для первой стадии. Например:

```
-wait-ping 500ms
```

Вообще, nesca4 все времена принимает в таких форматах:

| | |
|---|---|
| `10000` | `1000 nanoseconds` |
| `500ms` | `500 milliseconds` |
| `1s` | `1 second` |
| `10m` | `10 minutes` |
| `5h` | `1 hour` |

**4.** Далее, повысить скорость сканирования можно за счет изменения параметров групп.

Дело в том, что nesca4 подобно nmap делит все цели которые вы передали на группы, и сканирует их группами. Принцип работы таков: nesca4 получает 1000 целей, так как минимальный размер группы 100 (т. е. `gmin=100`) nesca4 формирует группу из 100 целей, и сканирует всю эту группу целиком; далее, так как увеличитель группы 100 (т. е. `gplus=100`), nesca4 увеличивает текущий размер группы на 100 (был 100 стал 200); nesca4 продолжит подобным образом увеличивать размер группы каждую итерацию до тех пор, пока не будет достигнут максимальный (т. е. `gmax`, по умолчанию он 800).

Такая схема работы нужна для того, чтобы мы могли получать результаты сканирования прямо во время его выполнения, а не жда-

ли его полного окончания. Также это полезно для того, чтобы не заполнить всю оперативную
570 память, загружая туда за раз 1 млн адресов.

Итак, изменить параметры группы можно следующими флагами:

```
-gmax 3000
-gplus 300
-gmin 350
```

Наиболее сильно влияет размер максимальной, и размер минимальной группы, так как последний есть размер самой первой.

**5.** Далее, повысить скорость сканирования
580 можно за счет увелечения максимального количества открытых сокетов.

Вообще, чтобы отправлять и слать пакеты (и соответственно выполнять сканирование), требуется открыть сокет, через который и будут проводится эти операции. В nesca4 один сокет = один поток, значит, чем больше мы откроем сокетов, тем больше сканирований мы сможем выполнять одновременно. Важно учитывать: как правило, Linux поддерживает макси-
590 мум 1024 открытых сокета, и некоторые из них могут быть заняты другими приложения, поэтому ставить все 1024 обычно не выходит. Стало быть делаем так:

```
-maxfds 1010
```

Это должно значительно ускорить сканирование; правда, стоит учитывать, что сокеты еще приходится закрывать, а это занимает много времени. Мое решение этой проблемы в nesca4 очень плохое, возможно даже смешное, но рабочее: сокеты в потоках. Я почти уверен в том, что это генерирует утечки памяти, разные ошибки, но скорость которая достигается действительно высока.

**6.** Далее, повысить скорость сканирования можно за счет уменьшения умножителя времени ожидания для сканирования портов.

Дело в том, что nesca4, как уже было сказано, в случае сканирования портов ставит не статичное, а расчитанное значение времени ожидания пакетов. Она расчитывает это время на основе времени ответа, которое получает на первой стадии; именно по этой причине, отключение первой стадии может только замедлить сканирование. Расчет производится по такой формуле: rtt * mtpl-scan = время ожидания, где `mtpl-scan` — это умножитель, по умолчанию: 10.

Чем ниже время ожидания, тем быстрее; исходя из этой формулы, понизив `mtpl-scan`, мы

понизим и время ожидания. Сделать это можно так:

```
-mtpl-scan 7
```

Впрочем, я не рекомендую это делать, — обычно это излишне.

**(IV).** Итак, теперь поговорим о том, как увеличить точность сканирования (и, в том числе, обходить различные защиты, firewalls, и т. д.). Очень важно заметить: все способы в предыдущей главе хоть и повышают скорость, но если их «перевернуть», они будут повышать точность. Например, вместо `-maxfds 1010`, указывать `-maxfds 10`. Вообще, почти во всех случаях работает такая формула: чем выше точность, тем ниже скорость, и наоборот. Так что, в силу этого, в этой главе придется немного повториться:

**1.** Повысить точность можно за счет увеличения количества пинг сканирования, и сканирования портов. По умолчанию, nesca4 проводит их один раз на один хост, но можно увеличить это значение:

```
-num-scan 10
-num-ping 10
```

**2.** Чем больше методов, чем больше шанса что метод сработает на хосте. Для TCP сканирования портов методы и их флаги суть:

```
-syn -xmas -fin -null
-maimon -psh -null
```

Впрочем, все кроме `-syn` срабатывают уж очень редко, и некоторые вовсе, даже не позволяют определить открыт ли порт, но `-fin` иногда срабатывает.

Для пинг сканирования методы суть:

```
-pe -pa80 -ps80 -py80 -pu53 -pi -pm
```

Я привел их в порядке от самых часто рабочих, к самым редко рабочим; также, я не учел ARP пинг (-pr), так как он работает только на локальных хостах. Следует уточнить: метод (-pu) есть UDP метод, а (-py) есть SCTP метод. Если хотите, вы можете указать другие порты, а не использовать мою схему с 80 и 53.

Для SCTP сканирования портов методы суть:

```
-init -cookie
```

Для UDP есть:

```
-udp
```

Важно: если вы хотите сканировать SCTP или UDP порты (что очень специфично и редко требуется), вам необходимо указать для них порты:

```
-p S:80,443
-p U:53
```

Здесь буква `U` — UDP, и `S` — SCTP; для TCP букву можно не указывать, как мы это делали выше, а можно и указать `T`:

```
-p 80,443
-p T:80,443
```

Если указываете несколько, следует делать так:

```
-p 80,443,U:53,S:443
```

Чтобы быстро указать все методы сканиро-

вания портов и пинга, можно использовать флаги:

```
-all-ping
-all-scan
```

**3.** Остальные методы мы уже разбирали:

```
-maxfds -gmin -gmax -gplus
-wait-ping -mtpl-scan
```

Просто указывайте «наоборот» от того, как мы указывали в предыдущей главе: `maxfds` делайте меньше, `gmin` меньше, `gmax` меньше, `gplus` меньше, `wait-ping` больше, `mtpl-scan` больше.

**4.** Теперь следует сказать касательно обхода защит, что тоже может повысить точность; обычно они обходятся разными методами сканирования, но есть также:

```
-dlen 100
-dhex 0xa
-dstr hellohost
```

Все эти флаги позволяют добавить некоторые данные к тем пакетам, которые отправляются при сканировании и пинге; иногда это не возможно, но где возможно, они будут добавлены.

Первый добавляет в пакет случайные данные размером в 100, или в другое значение которое вы передали параметром.

Второй добавляет в пакет ваши данные указанные в hex (в данном случае `0xa`). Hex — шестнадцатеричная система счисления, причем тут это, мы объяснять не будем.

Третий добавляет в пакет ваши ASCII данные, т. е. какую-нибудь строку, в данном случае — `hellohost`.

Также есть следующие флаги (приведу лишь основные):

```
-ipopt 0xa
-off df/mf/rf
-win 400
-adler32
```

Первый позволяет указать вам ваши IPv4 опции, он делает это также как `-dhex` в hex.

Второй позволяет указать вам флаги фрагментации, в примере выше указаны все три флага, но можно указать и какой-то один из них. Как видно, два или более указываются через `/`.

Третий позволяет указать вам размер окна для TCP; в нашем случае 400.

Четвертый говорит nesca4 о том, что checksum

для SCTP следует расчитывать не CRC32C методом, но Adler-32.

**(V).** Итак, теперь поговорим о всех тех флагах, о которых не сказали ранее:

Флаг `-dev ifname` — позволяет указать свой сетевой интерфейс, который nesca4 будет использовать для отправки и приема пакетов при сканировании и пинге; в параметр он принимает имя этого сетевого интерфейса; имена можно узнать написав `ip a`. По умолчанию, nesca4 берет первый рабочий интерфейс.

Флаг `-dst mac` — позволяет указать свой MAC адрес получателя; по умолчанию, nesca4 вычисляет его сама для выбранного сетевого интерфейса.

Флаг `-src mac` — позволяет указать свой MAC адрес отправителя; по умолчанию, nesca4 вычисляет его сама для выбранного сетевого интерфейса.

Флаг `-ip4 ipv4` — позволяет указать свой IPv4 адрес отправителя; по умолчанию, nesca4 вычисляет его сама для выбранного сетевого интерфейса.

Флаг `-ip6 ipv6` — позволяет указать свой IPv6 адрес отправителя; впрочем, это безполезно, так как nesca4 не умеет в IPv6 (хотя там все

для этого и есть).

Флаг `-pps num` — позволяет ограничить количество пакетов в секунду как при сканировании портов, так и при пинг сканировании; по умолчанию, не ограничено.

Флаг `-stats` — позволяет включить статистику как сканирования портов, так и пинг сканирования. Он включается при флаге `-v`, и является его облегченной версией.

Флаг `-ackn num` — позволяет указать свой acknowledgment number для TCP пакетов при пинг сканировании, и сканировании портов.

Флаг `-badsum` — указывает nesca4 то, что следует расчитывать неправильные checksum для пакетов при пинг сканировании, и сканировании портов; обычно это ведет к тому, что мы не получаем ответа. Важно отметить: на checksum IP это не влияет.

Флаг `-dbpath filepath` — позволяет указать свой путь к базе опознания, которая используется на предпоследней стадии.

Флаг `-wait-brute time` — позволяет указать время ожидания пакетов при брутфорсе; указывается также как `-wait-ping`.

Флаг `-threads-brute num` — позволяет указать количество потоков для брутфорса; хоть это и может его ускорить, но вместе с тем и по-

низит точность.

Флаг `-delay-brute time` — позволяет указать задержку между пакетами при брутфорсе; хоть это и может его замедлить, но вместе с тем и повысить точность.