

инструменты Octopus, Leech и Mycelium для атак на Linux

rt-solar.ru/solar-4rays/blog/6253

January 1, 2000



В марте этого года мы выпустили статью о новых [инструментах](#) Shedding Zmiy. В том инциденте были обнаружены не только руткиты Puma и Kitsune, но и еще несколько новых утилит, написанных на Rust, которые могут использовать злоумышленники для закрепления в системе и постэксплуатации. Техники, которые используют эти вредоносы и то, как они «взаимодействуют» между собой, показались нам интересными, поэтому мы решили описать их подробнее.

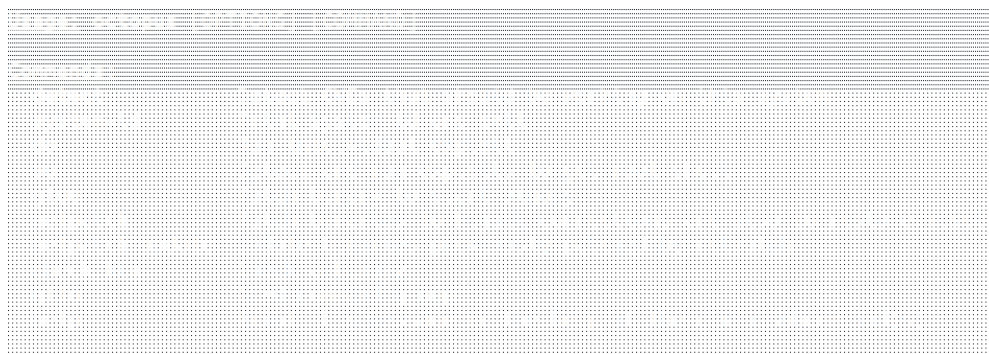
В этом материале мы рассмотрим, как утилиты помогают повысить привилегии, каким образом происходит их закрепление в системе и дальнейшая маскировка и, наконец, какие возможности получает злоумышленник, когда эти вредоносы попадают в систему.

Octopus

Octopus — это интерактивная утилита командной строки для Linux-систем, написанная на языке Rust. Она используется злоумышленниками для повышения привилегий в системе и дальнейшего закрепления. В упомянутом выше инциденте мы не видели Octopus, его семплом с нами поделились коллеги по индустрии. Вероятно, два других вредоноса появились в системе именно благодаря ему.

Повышение привилегий

У Octopus есть меню с большим количеством режимов:



Меню Octopus

Как видим, в нем реализовано два способа повышения привилегий:

- Эксплуатация уязвимостей в системе.
- Эксплуатация мiskonfigураций с использованием GTFO.

Утилита имеет на борту несколько эксплойтов для известных уязвимостей:

- pwnkit — CVE-2021-4034,
- looney-tunables — CVE-2023-4911,
- baron — CVE-2021-3156,

- route4-filter — CVE-2022-2588.

Помимо эксплуатации этих уязвимостей Octorus также предоставляет функциональность для их выявления, поскольку хранит в себе данные об уязвимых операционных системах и версиях пакетов. Фрагмент таких данных показан ниже:

```
{'by_distro': {'Centos': {'by_codename': {'6': {'cve_conditions': {}},
'7': {'cve_conditions': {'CVE-2021-3156': {'Or': [{'PackageVersion': {'package':
'patched_version': '0:1.8.23-10.el7_9.1'}}},
{'PackageVersion': {'package':
'patched_version': '0:1.8.23-10.el7_9.1'}}]}},
'CVE-2021-4034': {'Or': [{'PackageVersion': {'package':
'patched_version': '0:0.112-26.el7_9.1'}}},
{'PackageVersion': {'package':
'patched_version': '0:0.112-26.el7_9.1'}}},
{'PackageVersion': {'package':
'patched_version': '0:0.112-26.el7_9.1'}}]}},
...
'sudo',
'sudo-devel',
'polkit',
'polkit-devel',
'polkit-docs',
...

```

Таким образом, Octorus может автоматически выбирать, какую уязвимость лучше использовать для текущей системы жертвы.

Также утилита может анализировать систему на наличие исполняемых файлов, у которых установлен бит `suid` или атрибут `security.capability = CAP_SUID`. Если такие файлы найдены, Octorus проверяет, есть ли они в базе данных [GTFO](#), которую он хранит внутри себя. Фрагмент этой базы данных:

```
{'cap_setuid': {'gdb': [{'script': './gdb -nx -ex 'python import os; "
os.setuid(0)' -ex '!sh' -ex quit"}],
'node': [{'script': './node -e 'process.setuid(0); "
require("child_process").spawn("/bin/sh", "
{stdio: [0, 1, 2]}')"}],
'perl': [{'script': './perl -e 'use POSIX qw(setuid); "
POSIX::setuid(0); exec "/bin/sh";\''}],
'php': [{'script': 'CMD="/bin/sh"\n'
'./php -r "posix_setuid(0); '
'system(\"$CMD\");"\n'}],
'python': [{'script': './python -c 'import os; os.setuid(0); "
os.system("/bin/sh")\''}],
'ruby': [{'script': './ruby -e 'Process::Sys.setuid(0); exec "
...

```

Если скрипт есть в базе GTFO, то Octorus предлагает использовать его для получения оболочки с повышенными привилегиями.

После того как злоумышленнику удалось повысить свои привилегии, у него появляется возможность закрепиться в системе жертвы, используя Octorus и встроенные в него утилиты.

Закрепление в системе

Для закрепления в системе внутри Octorus хранятся несколько исполняемых файлов — это библиотеки *Leech* и исполняемый файл *Musclium*, они закодированы CBOR и сжаты с помощью *zstd*. *Leech* и *Musclium* будут описаны далее, но для начала рассмотрим процесс патчинга системных сервисов, в результате которого в системе закрепляются «нежелательные гости».

Для патчинга в меню Octorus есть три пункта: *glue*, *netpatch* и *netpatch-update*. Последние два связаны и являются автоматизацией первого, *glue* дает возможность злоумышленнику полностью сконфигурировать *Leech*-библиотеку и запустить процесс патчинга. Для этого есть три типа целей:

- *bin*,
- *bin-network*,
- *bin-atexit*.

От вида цели зависит, какой тип *Leech* будет использоваться и в какой последовательности будет происходить патчинг. Цели *bin*- и *bin-network* будут пропатчены в первую очередь, также для них можно задать параметр *interval*, через который будет происходить проверка и при необходимости повторный патчинг уже на стороне самой *Leech*-библиотеки, если не задан параметр *once*. Отличие этих типов заключается лишь в том, что в *bin* будет встроена *basic-lib*, а в *bin-network* — *network-lib*. Эти библиотеки будут описаны позже.

Bin-atexit патчатся в последнюю очередь, и их проверка и повторный патчинг будут проводиться в конце работы Leech-библиотек. В *bin-atexit* встраивается *basic-lib*. В качестве целей help Ostorus рекомендует выбирать пакетные менеджеры. Возможно, такой совет обусловлен тем, что Leech в пакетном менеджере сможет выжить даже после обновления всех остальных зараженных пакетов.

Также параметром *glue* можно задать имя встроенного бинаря или передать путь к какому-то стороннему исполняемому файлу, который будет встроен в Leech для дальнейшего использования. В анализируемом семпле встроенной нагрузкой был *Mycelium*.

Теперь рассмотрим непосредственно сам процесс патчинга. Для начала Ostorus подготавливает Leech-библиотеки — встраивает конфиг и добавляет выбранную полезную нагрузку, если это необходимо. Утилита «приклеивает» конфиг и полезную нагрузку к библиотеке (закодированные и упакованные), создавая при этом дополнительную секцию с типом 0x7 — NOTE секция.

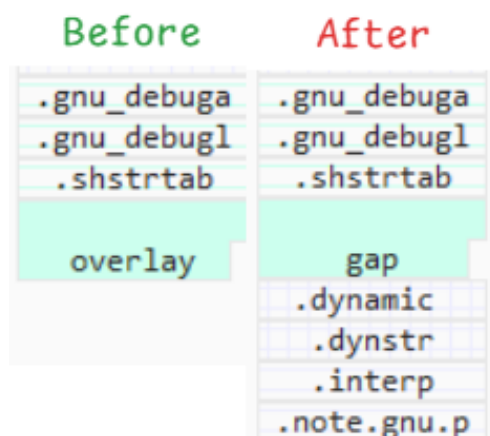
Далее подготовленные Leech-библиотеки помещаются в папку, где хранится системная *glibc*. Стандартными именами для них являются:

- *libzst.so.0* для *basic-lib*,
- *libsockopt.so.1* для *network-lib*,
- *libsockopt.so.2* для *network-lib mosquito*.

Также у файлов устанавливаются новые временные метки (с помощью системного вызова *utimensat*) и атрибуты *selinux*, такие же как у файла *glibc*. Это делается для того, чтобы файлы библиотек меньше выделялись среди других файлов в директории. После всего этого библиотеки готовы для встраивания.

Для того чтобы заставить процесс-жертву подгружать Leech-библиотеку при запуске, Ostorus редактирует исполняемый файл, меняя секции, которые использует динамический линковщик во время загрузки ELF в память.

Ostorus создает дополнительный LOAD-сегмент в конце файла и копирует туда *.dynamic*- и *.dynstr*-секции, в которых описаны динамические зависимости ELF файла.



Новый LOAD-сегмент в конце файла

При копировании в конец *.dynstr*-секции добавляется имя вредоносной библиотеки, а в начало новой *.dynamic*-секции добавляется запись с тегом *DT_NEEDED* и смещением до имени вредоносной библиотеки. Таким образом, у исполняемого файла появляется еще одна зависимость в виде Leech-библиотеки, которая загрузится раньше всех остальных библиотек, что позволит произвести перехват функций. Такого же эффекта злоумышленник мог бы достичь, если бы запускал процесс с установленной переменной окружения *LD_PRELOAD*. Но техника с переменной окружения является более заметной и хорошо известна DFIR-специалистам.

```
Dynamic section at offset 0x2e00 contains 24 entries:
Tag          Type          Name/Value
0x0000000000000001 (NEEDED)      Shared library: [libc.so.6]
0x000000000000000c (INIT)        0x1000
0x000000000000000d (FINI)        0x11b8
0x0000000000000019 (INIT_ARRAY)   0x3df0
0x000000000000001b (INIT_ARRAYSZ) 8 (bytes)
```

Dynamic до патча

```
Dynamic section at offset 0x5000 contains 25 entries:
  Tag          Type          Name/Value
0x0000000000000001 (NEEDED)      Shared library: [libsockopt.so.1]
0x0000000000000001 (NEEDED)      Shared library: [libc.so.6]
0x000000000000000c (INIT)        0x1000
0x000000000000000d (FINI)        0x11b8
0x0000000000000019 (INIT_ARRAY)  0x3df0
0x000000000000001b (INIT_ARRAYSZ) 8 (bytes)
```

Dynamic после патча

После того как процесс патчинга завершился, Octopus пытается скрыть факт изменения этих файлов, для чего он исправляет контрольные суммы пакетов, которые лежат в директории `/var/lib/dpkg/info`, если в системе используется пакетный менеджер `dpkg` — стандартный пакетный менеджер в Debian-системах, например, таких как Ubuntu. Другие пакетные менеджеры Octopus версии 16.7.0, который мы анализировали, не поддерживает.

Теперь, когда контрольные суммы исправлены и проверка по ним целостности пакетов не выявит изменений Octopus должен избавиться еще от одной улики — аномалии во временных метках файлов. Поскольку изменяются только `.md5sums`-файлы, их временные метки `mtime` (время последнего изменения содержимого файла) будут выделяться на фоне остальных. Значения `mtime`-метки пользователь может увидеть, например, с помощью команды `ls -l`. Octopus меняет эту временную метку для того, чтобы измененные `.md5sums`-файлы не выделялись на фоне остальных.

Тем не менее аномалию все еще можно обнаружить, если посмотреть на другую временную метку — `ctime` (время последнего изменения метаданных файла). Изменить эту метку так же, как `mtime`, не получится. Посмотреть значение `ctime` можно с помощью команды `ls -lc` или `stat`. Для наглядности аномалия во временных метках файла `.md5sums` представлена ниже:

```
user@noubuntu:~$ ls -l /var/lib/dpkg/info | grep openssh-server
```

`mtime` и `ctime` до патча

mtime и ctime после патча

Пункт меню netpatch делает то же самое, что и glue, но при этом самостоятельно выбирает цели. В этом режиме Octopus сам ищет сервисы с именами: crond, sshd, postgres, nginx, apache2, lighttpd, mariadb и пытается их пропатчить. А если запустить Octopus в режиме netpatch-update, то у злоумышленника появляется возможность обновить конфигурацию Leech-библиотек и Myscelium или полностью заменить их на новые.

Таким образом, Octopus помогает злоумышленникам сначала повысить свои привилегии в системе, а затем закрепиться в ней с помощью патчинга различных системных сервисов, что делает его очень опасным инструментом в арсенале злоумышленников.

Leech

Leech — вредонос для Linux-систем, который может использоваться атакующими для поддержания присутствия в системе и для удаленного управления. Он становится следующим после Octopus звеном в цепочке получения контроля над системой.

Техническое описание

Leech распространяется в виде so-библиотеки для того, чтобы подгружаться в процессе работы зараженных сервисов. Как мы уже говорили, существует два типа Leech-библиотек:

- basic-lib,
- network-lib.

Basic-lib в процессе работы проверяет зараженные сервисы и заражает их заново в случае, если они каким-то образом были исправлены, — например, с помощью обновления пакета. Сам процесс патчинга в Leech практически ничем не отличается от Octopus. Цели для патчинга вредонос берет из конфига, который был встроен в него предварительно. Вот пример конфигурации:

```

Config {
  interval: Some(
    43200s,
  ),
  interval_spread: Some(
    14400s,
  ),
  background_paths: ["/usr/sbin/cron"],
  atexit_paths: ["/usr/bin/dpkg"],
  network_paths: [
    "/usr/sbin/sshd",
    "/usr/sbin/nginx"
  ],
  env: [],
  basic_lib: "libzst.so.0",
  network_lib: "libsockopt.so.1",
  package_manager: Some(
    Dpkg,
  ),
  packages: Some(
    [ "cron", "nginx", "dpkg", "openssh-server" ],
  ),
}

```

В ней указан интервал, с которым библиотека будет проводить проверку, и все зараженные цели.

Network-lib, помимо того что делает *basic-lib*, еще и перехватывает входящие подключения к зараженному сервису. Это происходит в момент вызова функций ассерт или ассерт4 процессом-жертвой. Leech перехватывает определенные подключения и дальше с ними взаимодействует. Это объясняет выбор таких целей, как, например, sshd или nginx, так как к ним зачастую можно получить доступ извне, что вызовет меньше подозрений, чем подключение к нестандартному порту без легитимного сервиса.

Для того чтобы начать взаимодействие с Leech-библиотекой, на порт, прослушиваемый зараженным сервисом, нужно отправить TLS 1.3 Client Hello. В таком случае библиотека обратит внимание на это подключение и начнет его обработку. В пакете Client Hello вредонос будет искать TLS-расширение GREASE-типа. Этот тип расширения описан в [RFC 8701](#). Он создан для того, чтобы находить узлы в сети, которые не поддерживают актуальную версию TLS. У таких расширений есть зарезервированные значения типов: 0x0A0A, 0x1A1A, 0x2A2A и так до 0xFAFA. В теле такого расширения обычно содержатся случайные байты. Как таковой смысловой нагрузки они не несут и работу обычных приложений не нарушают. Это объясняет, почему злоумышленники выбрали именно этот тип — зашифрованная внутри тела расширения полезная нагрузка выглядит не так подозрительно.

TLS-расширение, используемое злоумышленниками, имеет следующую структуру:

Смещение	Содержимое
0..2	Тип расширения
2..4	Длина тела расширения
4..8	magic bytes (0x33325052)
8..N-4	payload
N-4..N	CRC32 предыдущих байтов (от 4 до N-4)

Magic bytes, payload и контрольная сумма зашифрованы с помощью xor. В качестве ключа используется 32-байтный Client Random.

Если тело расширения прошло все проверки, начинается обработка полезной нагрузки. Она сериализована с помощью CBOR и может иметь различную структуру. Структура определяется по полю, которую разработчик назвал Marker::service. Дальнейшая работа вредоноса будет зависеть от того, какой сервис выбран злоумышленником. Таких сервисов несколько, в этом разделе мы остановимся на одном, так как он непосредственно касается самой Leech-библиотеки.

Его имя — lrex. Возможно, это расшифровывается как Leech Remote Execute, так как в задачи этого сервиса входит исполнение команд, которые передал злоумышленник. В этом случае payload будет иметь следующую структуру:

```
{
  0:"lrex",    # Marker::service
  1:task,      # ExecuteTask:task
  2:{0:sig, 1:cert} # ExecuteTask::auth - {0:Auth::signature, 1:Auth::certificates}
}
```

В payload-структуре содержится как сам task, так и данные для верификации злоумышленника. Проверяется переданный сертификат, а также подпись task'a. Если проверка не пройдена, выполнение прерывается. В случае успеха вредонос обрабатывает саму команду, которая также сериализована и закодирована с помощью CBOR:

```
{
  0:issued,    # Task::issued - unix время создания таска
  1:expires_in, # Task::expires_in - количество секунд, в течение которых таск действителен
  2:"/path/to/executable", # Task::command
  3:[arg1, arg2] # Task::args
}
```

Таким образом, злоумышленник может уже исполнять какие-то команды с помощью самой Leech-библиотеки. Но этим контроль над системой не ограничивается, так как эта «матрешка» может нести в себе еще один вредонос — Mycelium. Он распаковывается и помещается в memfd с именем *kernel*, откуда и происходит запуск. Так картина зараженной системы выглядит после запуска всех трех вредоносов:

Victim machine

Вредоносы в системе

Стоит отметить, что у *network-lib* есть облегченная версия — Mosquito. В инциденте его использование не было замечено, но Ostopus может вместо Leech патчить сервисы с помощью этой lite-версии. В комплектацию этого вредоноса входит только сетевая часть, функции патчинга отсутствуют. Для того чтобы уменьшить размер исполняемого файла, в нем используется минимальное количество Rust-крейтов.

Mycelium

Mycelium является третьим и завершающим инструментом в этом наборе. Мы уже видели упоминание этого инструмента, когда исследовали [веб-панель](#) злоумышленников. Он расширяет пространство возможностей для постэксплуатации, добавляя еще три сервиса к тому, что был описан выше.

Архитектурно этот вредонос является асинхронным приложением, написанным с использованием крейта Tokio. Этот крейт довольно часто встречается во вредоносах, написанных на Rust, которые каким-то образом взаимодействуют с сетью. Помимо этого, в его работе используется мультиплексирование для более эффективной обработки подключений.

У Mycelium есть несколько режимов работы: пассивный, когда он ожидает входящих подключений, и активный, когда он подключается к злоумышленнику самостоятельно. В активном режиме вредонос использует конфигурацию, которая хранится закодированной в base64 в переменной окружения TOKEN. Пример конфигурации:

```

MyceliumConfig {
    client_connection: Some(
        MyceliumClientConnection {
            host: "192.168.56.101",
            port: 1337,
            ping_interval: 200s,
            success_reconnect_interval: 43200s,
            failure_reconnect_interval: 14400s,
        },
    ),
}

```

В пассивном режиме Muselium сначала демонизируется, используя double fork, после чего сканирует все процессы в системе и просматривает их имена, для того чтобы выбрать себе новое. Имя выбирается с целью мимикрировать под какой-нибудь служебный процесс. Например, в инциденте, который мы расследовали, Muselium был запущен с именем [kintegrityd]. Оригинальный сервис служит для проверки целостности данных, а мимикрирующий под него процесс отличается от оригинального наличием квадратных скобок в имени.

Взаимодействие в системе

Поскольку в одной системе может быть несколько зараженных сервисов, может случиться так, что будет запущено несколько образцов Muselium. В таком случае вредоносам придется «договориться» — кто будет «хозяйничать» в системе.

Для этого Muselium использует AF_UNIX (локальный) сокет. Файл этого сокета может быть создан в нескольких местах на диске:

```

/run/dbus/auxiliary_bus_socket
/var/run/dbus/auxiliary_bus_socket
$XDG_RUNTIME_DIR/systemd/generic
/run/user/systemd/generic

```

В начале работы Muselium отправляет в него свой pid и версию. Если какой-то другой образец пришлет сообщение с версией ниже либо равной, чем его собственная, более новый Muselium завершит процесс с помощью *kill*. Таким образом, в системе остается единственный образец с самой актуальной версией.

На фрагментах ниже, где представлен вывод команды *strace*, можно увидеть обмен информацией о pid и версии вредоносных:

Отправка pid muselium

```
socket(AF_UNIX, SOCK_DGRAM|SOCK_CLOEXEC, 0) = 5
```

Обмен версиями

Но только этим взаимодействие через этот сокет не ограничивается. Muselium не получает подключения извне напрямую. Это происходит через Leech, с помощью обращения к конкретным сервисам. В прошлом разделе мы описали один из них, а Muselium, в свою очередь, реализует еще три сервиса. И если Leech-библиотека получает запрос от злоумышленника, в котором тип сервиса отличается от lrex, то файловый дескриптор этого подключения с помощью *sendmsg()* будет передаваться в Muselium, где и происходит обработка этого запроса.

Вот как это выглядит в выводе утилиты *strace*:

```
pid=20001, uid=0: "Transferring fd 4 (socket) to h2mx" 35Transferring fd 4 (socket) to h2mx)
```

Передача fd-сокета

Muselium сервис h2mx

Сервис h2mx является асинхронным веб-сервером, который работает на HTTP/2 и в качестве транспорта использует TLS. Название указывает на то, что он реализован с помощью rust-крейта h2 и использует мультиплексирование. Он предоставляет злоумышленнику API, которое можно использовать для постэксплуатации системы.

API содержит следующие роуты:

Роут	Назначение
/version	Возвращает версию

/config	Возвращает конфигурацию
/connections	Возвращает информацию об активных подключениях к Mycelium
/nu/run	Запускает команду nushell
/fs/list	Возвращает содержимое указанной директории
/fs/file	Скачивает с системы указанный файл
/host-info	Возвращает информацию о системе: об ОС, пользователе и сетевых интерфейсах
/bin	Используется для эксфильтрации файлов. Подробности найдете ниже
/command	Выполняет shell-команду
/arp	Возвращает arp-кеш из файла /proc/net/arp
/arp/scan	Используется для сетевого сканирования
/echo	Возвращает присланные данные
/connect/:addr/:port	Позволяет злоумышленнику создать активное подключение с зараженной системы на указанный хост и порт
/log	Возвращает логи подключений
/serve/h1	Используется для эксфильтрации файлов. Подробности найдете ниже
/rsh/:id	Получение rsh-подключений по id
/yamux-proxy/:id	Получение yamux-проху-подключений по id
/subscribe	Пока не известно

Пояснения к некоторым роутам:

- **/bin/:hash** — роут поддерживает несколько методов:
 - PUT — позволяет передать файл в Mycelium. Для него создается анонимный файл с именем tempfd, и содержимое записывается туда.
 - DELETE — удаляет файл по хешу.
 - FETCH — позволяет загрузить файл с другого хоста с помощью HTTP/1.1.
- **/serve/h1** — после обращения к этому роуту Mycelium запускает еще один веб-сервер на случайном порте, который уже работает по HTTP/1.1. Обращаясь на этот сервер с GET-запросом /bin/:hash, можно получить предварительно сохраненный файл. Этот роут явно имеет совместимость с методом FETCH предыдущего роута.

Mycelium-сервисы rsh и yamux-proxy

Два последних сервиса не используются злоумышленником напрямую и являются вспомогательными. Rsh реализует функциональность reverse shell, а yamux-proxy, в свою очередь, является reverse-прокси, который использует мультиплексирование. Эти два сервиса «работают» в паре с другим инструментом Shedding Zmiy — Bulldog Backdoor, о котором мы уже писали в нашей [статье](#).

Во время анализа Bulldog Backdoor мы обнаружили функции, которые реализуют взаимодействие с Mycelium. Его можно сконфигурировать для использования Mycelium в качестве протокола для revsh и проху. Например, на скриншоте ниже изображен фрагмент функции myceliumConnect из модуля revshell_client. В этом фрагменте видно, что создается сообщение к Mycelium с запросом к сервису rsh:

```

else
{
    // Successful connected to mycelium
    data = mycelium_con_struct.0.data;
    MyceliumHandshakeRequest = red_team_go_red_util_net_GetMyceliumHandshakeRequest(
        "rsh", // requested service
        3u,
        client_Host->info.ID.ptr,
        client_Host->info.ID.len);
    *((_QWORD *)&v20 + 1) = MyceliumHandshakeRequest._r4;
    if ( MyceliumHandshakeRequest._r3 )

```

Фрагмент функции myceliumConnect

Похожий фрагмент есть в модуле проху в функции getMyceliumProxyConn, но там запрашивается уже сервис yamux-проху:

```

{
    MyceliumHandshakeRequest = red_team_go_red_util_net_GetMyceliumHandshakeRequest(
        "yamux-proxy", // requested service
        11u,
        client_Host_ID_ptr,
        client_Host_ID_size);
    *((_QWORD *)&v21 + 1) = MyceliumHandshakeRequest._r4;

```

Фрагмент функции getMyceliumProxyConn

Mycelium закрывает эту линейку Rust-вредоносов, предоставляя злоумышленнику широкий набор инструментов для контроля скомпрометированной системы. Конечно, его использование для постэксплуатации необязательно, Leech *network-lib* уже позволяет выполнять команды в системе, но Mycelium дает возможность атакующему значительно упростить этот процесс.

Заключение

Группировка Shedding Zmiy располагает широким набором различных инструментов. И, помимо добавления новых вредоносов в свой арсенал, злоумышленники выстраивают для себя экосистему инструментов, связывая новые со старыми. Важно и то, что при разработке применялся язык программирования Rust, что в последнее время по понятной причине стало некой модой, — вредоносы, написанные на этом языке, зачастую сложно анализировать.

Стоит отметить, что в этих инструментах Shedding Zmiy немного изменила свою любимую технику подмены файлов системных сервисов и утилит. Описанные в данной статье вредоносы паразитируют на легитимных процессах и скрывают свой трафик среди трафика сервиса-жертвы. Тем не менее тенденция маскировки вредоносной активности среди обычных для системы процессов сохраняется.

IOCs

Files

MD5

```

864ee7604990860e0959e118e3197a96
00e6c33906e956aa744a928f19d858cf
b67ea0808bd7928608a7a7b84f912271
32fd0405eb5c5b1ac2a889d2a7831506
7c28de099682b375d1419ce28d27e4b4
0c020fe0267b97f456097c654d87bfc3
acf519e0d60a973491a5ed9ba9599353
033b37b9ef83dd94f7c24591a666e5cd
31afd8acd7a11de4c3d811a3f501443f
e30f45a6de01a95928046785cd29b620
4037db8b3f1f51eedcd83a0d00777182

```

SHA1

```

05245a5aff54c598adef5e67d8ccd8b19401c60
3f5d0bc0c7b8acacc614763179f810f29ab6fe48
c84b601193c61b41fb7b86db1d2aeecf97f8bca9
5d7fdb73e1038d722e7d07c3c9d4c9163672662a
69fd4dde66052af6f56afa74c0a059f427515520
dd9e05ae3f7240e91d6212d98f0405b396091d5e
8c2e474e2b7ae077c341f1e0a2d2ae6190effb01
d508ea3450883e44cf56e83e9a8ab352378ba4f0
13bcb2739dedc42bca8c8e1a761385c111b14d7c
10172b59abf695bd6fc4b7469b39ba2e96564571
9aa649623ca2c7c190ce66d0526e5830b08bbd50

```

SHA256

8b43430136fcc9ef43f201febddbdd608cf6bd25c611c13a146a66883b0f201e82581c8043ac12ef3e5c35572759fb494775aff210b060ebd7492d3134b25024e9b89ae1cf3c7263cfcd4c56ce826565fa619b1a5f867d2a139f84d19bc921a3e890107ebb7086f992d41fa739a572644a98edf1da6421fa76abb78b721d4059ad79af28fb9a5086dde3c071c32ff1fe9eae934c6edd1da88776fe6d41c201e157d42999aa0010a0c2c96076ff5097bf48bd487949733a15242bdd58d2f3553b29194e0d1464211b92cef881c6b5dec6cc353fb791715448bc1d2705e711edb21eba25e3090d8ad943fd5371da3e6598c486b67571f0b75fc910c6aec8edde15535ec963580b36561293c9bfff918dee4c6c8438abd7a76a31925f4245bcd65c6491be745661f1d1f01eaf9d5c4f15dbc494c81ca11d2ea04aedd7b8b0d0314dd7b95d9e2a8317d3f7f6480b30677d95aa5398ee4dab56e55b34f1c36bad5f3

Приложение. Расширенная информация по файловым IOCs

Имя	MD5	SHA1	SHA256
-	864ee7604990860e0959e118e3197a96	05245a5aff54c598adef5e67d8ccd8b19401c60	8b43430136fcc9ef43f201febddbdd608cf6
libsockopt.so.1	00e6c33906e956aa744a928f19d858cf	3f5d0bc0c7b8acacc614763179f810f29ab6fe48	82581c8043ac12ef3e5c35572759fb4947
libsockopt.so.1	b67ea0808bd7928608a7a7b84f912271	c84b601193c61b41fb7b86db1d2aee9f7f8bca9	e9b89ae1cf3c7263cfcd4c56ce826565fa6
libsockopt.so.1	32fd0405eb5c5b1ac2a889d2a7831506	5d7fdb73e1038d722e7d07c3c9d4c9163672662a	e890107ebb7086f992d41fa739a572644a
libzst.so.0	7c28de099682b375d1419ce28d27e4b4	69fd4dde66052af6f56afa74c0a059f427515520	ad79af28fb9a5086dde3c071c32ff1fe9eae
memfd:kernel	0c020fe0267b97f456097c654d87bfc3	dd9e05ae3f7240e91d6212d98f0405b396091d5e	57d42999aa0010a0c2c96076ff5097bf48t
memfd:kernel	acf519e0d60a973491a5ed9ba9599353	8c2e474e2b7ae077c341f1e0a2d2ae6190effb01	29194e0d1464211b92cef881c6b5dec6c
cron	033b37b9ef83dd94f7c24591a666e5cd	d508ea3450883e44cf56e83e9a8ab352378ba4f0	21eba25e3090d8ad943fd5371da3e6598
cron	31afd8acd7a11de4c3d811a3f501443f	13bcb2739dedc42bca8c8e1a761385c111b14d7c	5535ec963580b36561293c9bfff918dee4c
nginx	e30f45a6de01a95928046785cd29b620	10172b59abf695bd6fc4b7469b39ba2e96564571	6491be745661f1d1f01eaf9d5c4f15dbc49
sshd	4037db8b3f1f51eedcd83a0d00777182	9aa649623ca2c7c190ce66d0526e5830b08bbd50	dd7b95d9e2a8317d3f7f6480b30677d95a

Orion Skywalker

Содержание статьи

Получать новости

-
-

Сила киберразведки: знания об угрозах на страже защиты бизнеса

[Читать](#)