(Ex)Cobalt. Обзор инструментов группы в атаках за 2024–2025 годы

By Positive Technologies

Published: 2025-12-02 · Archived: 2026-04-10 02:37:45 UTC

2.4. Руткит PUMAKIT

Наиболее интересным инструментом группировки (Ex)Cobalt, обнаруженным в ходе расследований, является руткит PUMAKIT, который маскируется под легитимные компоненты ОС и скрывает свое присутствие.

Ранее инструмент был описан в публикации Elastic Security Labs: коллеги описали дроппер и часть функциональности LKM-руткита. Позднее исследователи из Solar 4RAYS опубликовали собственный анализ, в котором описали особенности работы бэкдора и механизм кражи данных.

В настоящей статье мы более подробно рассмотрим некоторые функции данного инструмента, а также проследим этапы его развития.

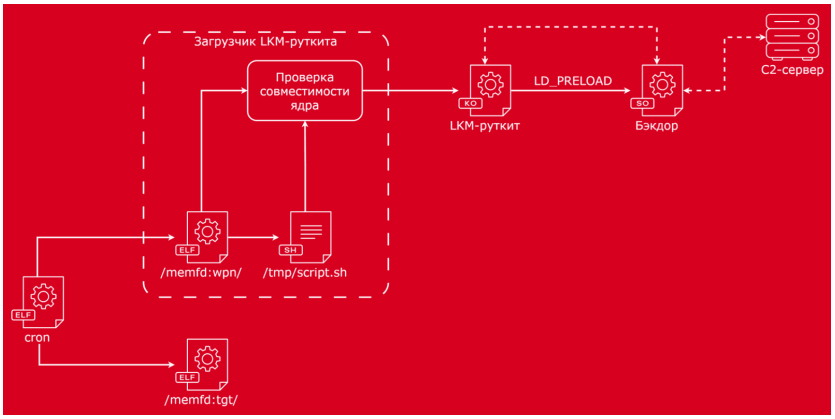

Рисунок 8. Диаграмма взаимодействия компонентов PUMAKIT

Обнаружить его удалось при помощи изучения аномалий во временных метках файлов. Для этого в инструментах, разработанных командой департамента комплексного реагирования на киберугрозы PT ESC, применяется алгоритм сопоставления временных меток файлов, который выявляет отклонения от времени создания или модификации файлов в каталоге.

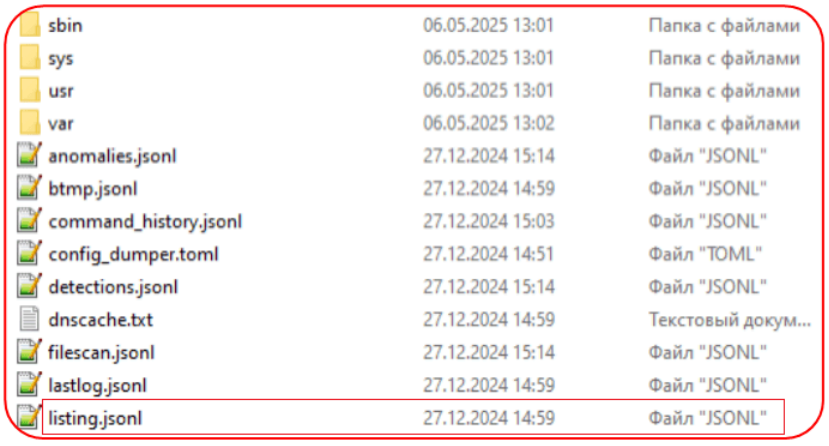

Рисунок 9. Результаты работы инструмента для поиска аномалий

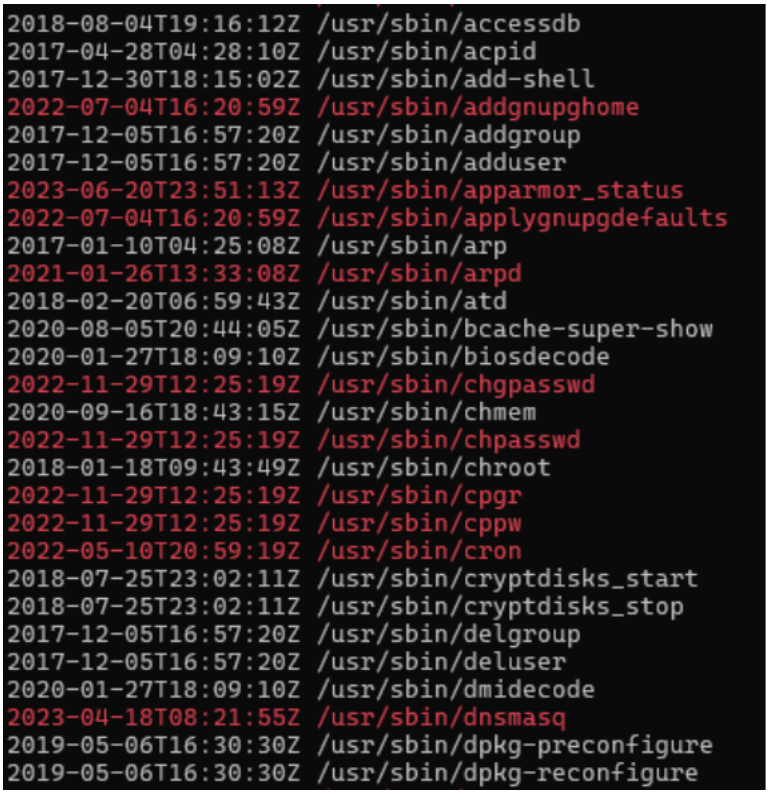

Рисунок 10. Обнаруженные аномалии во временных метках

Таким образом был определен круг подозрительных файлов в служебных каталогах ОС. Затем путем ручного анализа удалось выявить файл, маскировавшийся под компонент операционной системы (cron), размер которого отличался от размера легитимного файла более чем в 20 раз.

2.4.1. Первая стадия: cron

Найденный файл, подменявший системный файл cron, представлял собой дроппер и состоял из двух основных компонентов — оригинального cron (tgt, target) и установщика вредоносного модуля (wpn, weapon). Вместо записи исполняемых файлов на диск дроппер загружал их в анонимные файловые дескрипторы (/memfd:wpn и /memfd:tgt), а затем запускал комбинацией функций fork и execveat, обеспечивая их выполнение без сохранения в файловой системе. Таким образом, оригинальная функциональность cron сохранялась, что не вызывало подозрений, а вредоносная нагрузка незаметно выполнялась при каждом запуске системы.

Получив первоначальный доступ к системе с повышенными привилегиями, злоумышленники определяли версию ядра и выбирали установщик модуля, совместимый с конфигурацией системы жертвы. Это обеспечивало корректную интеграцию модуля в целевую среду и гарантировало его успешное выполнение.

2.4.2. Вторая стадия: weapon

Основная задача wpn — установка LKM-руткита, совместимого с системой жертвы. Для сокрытия активности загрузчик маскируется под системный процесс /usr/sbin/sshd.

Логика установки начинается с идентификации узла: вычисляется его уникальный идентификатор, формирующийся следующим образом:

1. с помощью Netlink-сокета в ядро отправляется запрос для получения информации о сетевых интерфейсах;
2. MAC-адреса обнаруженных интерфейсов последовательно сохраняются, за исключением начинающихся с «docker», «veth» или «br»;
3. полученные адреса объединяются в общий буфер, разделенный символами новой строки «\n»;
4. к получившемуся результату дописывается локальное имя системы и подается на вход алгоритма хеширования MD5;
5. вычисленное значение сохраняется в качестве agent_id системы.

Как только был получен и сохранен идентификатор системы — формируется команда sh -c «dmesg | grep ’ecure boot enabled’», которая выполняется с помощью интерпретатора командной строки. Результат ее выполнения позволяет определить состояние Secure Boot — механизма защиты, предотвращающего загрузку неподписанных или измененных загрузочных образов:

- если Secure Boot активен — выполнение немедленно прерывается;
- иначе — установка модуля ядра считается возможной: инициируется обращение к системному файлу /lib/modules/<версия_ядра>/build/Module.symvers для проверки соответствия экспортируемых символов ядра (функций, переменных) символам модуля.

При успешном доступе к файлу выполняется последовательное чтение его записей, при этом для каждой сохраняются контрольная сумма (cyclic redundancy check, CRC) и само имя символа.

2.4.2.1. Формирование собственного списка Module.symvers

Если не удалось получить информацию напрямую — создается собственная версия экспортируемых символов ядра:

1. Процесс обращается к двум файлам — /proc/version и /proc/cmdline, извлекая из них информацию о версии ядра.
2. В каталоге /boot выполняется поиск файла, начинающегося с «vmlinuz-».
3. Оставшаяся часть имени, определяющая версию ядра, сравнивается с теми версиями, которые были получены на шаге 1.
4. Если все три версии ядра совпали — создается файл с именем /tmp/script.sh, в который записывается скрипт для распаковки сжатого файла ядра.

```
 #!/bin/sh

c() {
  if file "$1" | grep -q "ELF"; then
    exit 0
  else
    return 1
  fi
}

d() {
  for p in tr "$1\n$2" "\n$2=" < "$i" | grep -abo "^$2"
  do
    p=${p%%:*}
    tail -c+$p "$i" | $3 > $r 2>/dev/null
    c $r
  done
}

i=$1
r="/tmp/vmlinux"
[[ -z $vmlinuz_path ]] || exit 0
d '\037\213\010' xy gunzip
d '\3757zXZ\000' abcde unxz
d 'BZh' xy bunzip2
d '\135\0\0\0' xxx unlzma
d '\211\114\132' xy 'lzop -d'
d '\002!L\030' xxx 'lz4 -d'
d '(\265/\375' xxx unzstd
c $i
exit 1
```

Листинг 1. Базовый скрипт для распаковки сжатого файла ядра

5. Данный скрипт запускается с помощью команды bash /tmp/script.sh “/boot/vmlinuz-<KERNEL_RELEASE>”.
6. В результате выполнения скрипта в каталоге /tmp появляется разархивированный файл ядра.
7. Сам скрипт удаляется.

После получения файла ядра выполняется перебор его таблицы заголовков для получения и сохранения размеров и смещений конкретных секций, таких как:

- __kcrctab_gpl: таблица контрольных сумм для символов, экспортируемых под лицензией GPL. Каждая запись содержит CRC, соответствующую символу из __ksymtab_gpl;
- __ksymtab_gpl: таблица символов, используется для разрешения ссылок на символы в модулях, совместимых с лицензией GPL. Каждая запись имеет поля:
    - value — адрес экспортируемого символа;

    - name — указатель на строку в секции __ksymtab_strings, содержащую имя символа;
- __ksymtab: список всех экспортированных символов, кроме тех, которые помечены GPL. Структурно идентична __ksymtab_gpl.

Перед переходом к извлечению информации о символах ядра с использованием перечисленных заголовков из секции .rodata считывается версия разархивированного ядра (рис. 11). С ее помощью определяется размер единичной записи в перечисленных выше секциях и идентификатор одного из четырех имеющихся обработчиков, который должен быть установлен для корректного парсинга и сохранения записей.

```
1
2 Hex dump of section '.rodata':
3   0xffffffff82000120 4c696e75 78207665 7273696f 6e20362e Linux version 6.
4   0xffffffff82000130 382e3131 2d616d64 36342028 64657665 8.11-amd64 (deve
```

Рисунок 11. Запись о версии ядра в секции .rodata

2.4.2.2. Проверка совместимости

После того как была получена информация об используемых символах ядра, независимо от способа ее получения, выполняется проверка совместимости этих символов с символами модуля, хранящимися в секции versions: поочередно каждая запись из списка системы сравнивается с соответствующей в секции модуля до тех пор, пока не будет найдено соответствие. Если хоть одна из них не была найдена в системе, было превышено количество итераций или CRC отличается — выполнение будет завершено.

В ином случае будет выполнена загрузка LKM-руткита в систему жертвы с помощью системного вызова init_module.

2.4.2.3. Отладочный режим

При детальном рассмотрении в загрузчике руткита был обнаружен отладочный режим, позволяющий получить расширенные сведения о процессе установки. Учитывая его сложность и множество зависимостей, мы предполагаем, что данный режим используется атакующей стороной в том случае, если при стандартном запуске дроппера (без дополнительных аргументов) не удается установить вредоносный модуль.

Для его активации злоумышленники выполняют два последовательных шага:

1. Запускают дроппер с предварительно установленной переменной окружения HUINDER и одним из следующих аргументов:
    - —extract-target или -et для извлечения tgt.bin;
    - —extract-weapon или -ew для извлечения wpn.bin.
2. После получения файла wpn.bin запускают его с аргументами -f, -v и -t (порядок аргументов не имеет значения) с правами суперпользователя.

После выполнения описанных выше действий загрузчик запускается в режиме отладки, в котором вместо непосредственной установки модуля осуществляется проверка его совместимости с системой. В этом режиме атакующему возвращается либо подтверждение возможности установки, либо подробное описание причин, по которым установка невозможна. Примеры возможных ошибок и их выводов показаны на рис. 12–14.

```
root@ubuntu:/tmp# ./wpn -t -f -v
[+] v2.404.25
[+] agent_id: 3670e7c7c4918f7577b045538cd4313b
[!!!] Secure boot enabled
root@ubuntu:/tmp#
```

Рисунок 12. Ошибка: используется Secure Boot

```
root@ubuntu:/tmp# ./wpn -t -f -v
[+] v2.404.25
[+] agent_id: 3670e7c7c4918f7577b045538cd4313b
[!] Wrong crc 30ff7695->2ca90791        module_layout
[!] Wrong crc e12e84cf->9a4105af        d_path
[!] Wrong crc ac1c4313->3d7b9a95        kmalloc_caches
[!] Wrong crc 97e2789a->7d0038a9        commit_creds
[!] Wrong crc 33a21a09->eca4aee1        pv_ops
[!] Wrong crc d10a0a5f->bf7cf078        kthread_create_on_node
[!] Wrong crc a22a96f7->4c9f47a5        current_task
[!] Wrong crc 8a23e3a5->dd786ec0        kthread_stop
[!] Wrong crc 6d95fde1->5e3d4b96        pid_task
[!] Wrong crc fcb49325->d67d4dbe        fput
[!] Wrong crc 11bab86->117a9d11 prepare_creds
[!] Wrong crc 25158a10->1956f9cd        wake_up_process
[!] Wrong crc 4f00afd3->a3859b3a        kmem_cache_alloc_trace
[!] Wrong crc 6788ede4->109525fb        find_get_pid
[!] Wrong crc 19e42a4f->2a0b8a0 vm_mmap
[!] Wrong crc 8b85d4c6->3cb66ed8        send_sig_info
[!] Wrong crc 54d65058->b5619408        fget
[!] This version can't be loaded to this kernel
root@ubuntu:/tmp#
```

Рисунок 13. Ошибка: несовпадение экспортируемых символов

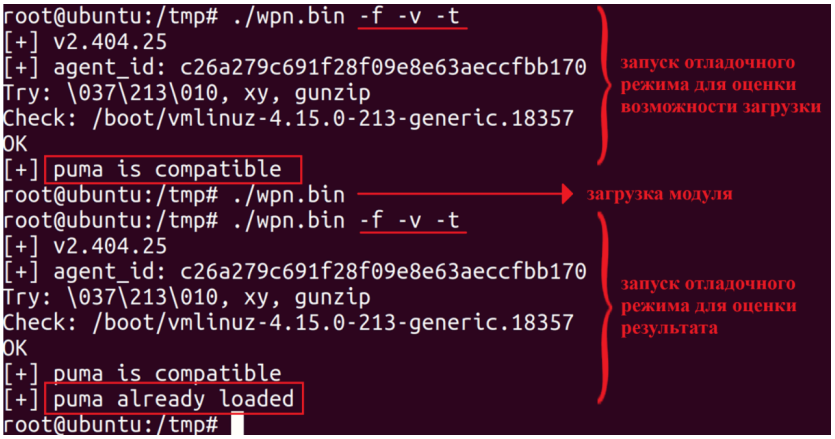

Рисунок 14. Предполагаемая последовательность действий злоумышленника

Кроме того, при запуске загрузчика в режиме отладки будет использоваться специальная версия скрипта для распаковки ядра, которая отличается от обычной добавленным журналированием, проверкой наличия утилит для распаковки (чтобы определить отсутствующие разархиваторы) и дополнительной проверкой на наличие данных в файле, что делает процесс более информативным (рис. 14).

```
 #!/bin/sh

c() {
  if [[ ! -s "$1" ]]; then
      return 1
  fi
  if file "$1" | grep -q "ELF"; then
    echo "OK"
    exit 0
  else
    echo "NOT ELF: $1"
    return 1
  fi
}

d() {
    echo "Try: $1, $2, $3"
    IFS=' ' read -r dcmd dargs <<< "$3"

    for p in tr "$1\n$2" "\n$2=" < "$i" | grep -abo "^$2"
    do
```

```
        p=${p%%:*}
        echo "Check: $i.$p"
        if ! command -v "$dcmd" &> /dev/null; then
            echo "Warning: Decompressor '$dcmd' not available. Skipping..."
            return 0
        fi
        tail -c+$p "$i" | $3 > $r 2>/dev/null
        c $r
    done
}

i=$1
r="/tmp/vmlinux"
[[ -z $vmlinuz_path ]] || exit 0
d '\037\213\010' xy gunzip
d '\3757zXZ\000' abcde unxz
d 'BZh' xy bunzip2
d '\135\0\0\0' xxx unlzma
d '\211\114\132' xy 'lzop -d'
d '\002!L\030' xxx 'lz4 -d'
d '(\265/\375' xxx unzstd
c $i
exit 1
```

Листинг 2. Расширенный скрипт для распаковки сжатого файла ядра

```
ubuntu@ubuntu:/tmp$ ./wpn.bin -f -t -v
[+] v2.404.25
[+] agent_id: c26a279c691f28f09e8e63aeccfbb170
Try: \037\213\010, xy, gunzip
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: \3757zXZ\000, abcde, unxz
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: BZh, xy, bunzip2
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: \135\0\0\0, xxx, unlzma
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: \211\114\132, xy, lzop -d
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: \002!L\030, xxx, lz4 -d
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
Try: (\265/\375, xxx, unzstd
/tmp/script.sh: line 19: /boot/vmlinuz-4.15.0-213-generic: Permission denied
NOT ELF: /boot/vmlinuz-4.15.0-213-generic
[!] Can't load symvers. No headers and kernel info
ubuntu@ubuntu:/tmp$
```

Рисунок 15. Работа расширенной версии скрипта

2.4.3. Третья стадия: LKM audit

При инициализации модуля выполняется несколько подготовительных шагов, необходимых для последующей работы руткита. Сначала выполняется регистрация и снятие kprobe, чтобы извлечь адрес kallsyms_lookup_name, обычно не экспортируемый напрямую. Полученный адрес сохраняется и используется для определения расположения таблицы системных вызовов.

После завершения подготовительных операций отключается защита записи в памяти ядра — для этого модифицируется содержимое регистра управления CR0: временно сбрасывается бит Write Protect, что позволяет записывать в ранее защищенные области памяти ядра (рис. 16).

```
unsigned __int64 disable_write_protect()
{
  CR0_reg_struct cr0_value; // [rsp+0h] [rbp-10h] BYREF

  *&cr0_value.prev_rsp_value = __readgsqword(0x28u);
  *&cr0_value.cr0_register_value = get_current_cr0();
  _InterlockedAnd8(&cr0_value.WriteProtect, 0xFEu);
  __writecr0(*&cr0_value.cr0_register_value);
  return *&cr0_value.prev_rsp_value - __readgsqword(0x28u);
}
```

Рисунок 16. Отключение защиты записи

После этого модуль способен напрямую изменять содержимое таблицы системных вызовов, обходя стандартные механизмы защиты, которые в обычных условиях запрещают подобные модификации.

Сразу после отключения защиты модуль заменяет стандартные системные вызовы своими обработчиками. Для этого сначала сохраняется оригинальный адрес каждого перехватываемого вызова, чтобы в дальнейшем можно было восстановить исходное поведение системы, после чего указатель в таблице системных вызовов обновляется (рис. 17).

```
disable_write_protect();

ptr_syscall_table = sys_call_table_addr;

original_recvfrom_addr = sys_call_table_addr->recvfrom;
sys_call_table_addr->recvfrom = hook_recvfrom_fun;
original_recvfrom_syscall = original_recvfrom_addr;

original_close_addr = ptr_syscall_table->close;
ptr_syscall_table->close = hook_close_fun;
original_close_syscall = original_close_addr;

original_execve_addr = ptr_syscall_table->execve;
ptr_syscall_table->execve = hook_execve_fun;
original_execve_syscall = original_execve_addr;

original_execveat_addr = ptr_syscall_table->execveat;
ptr_syscall_table->execveat = hook_execveat_fun;
original_execveat_syscall = original_execveat_addr;

original_newstat_addr = ptr_syscall_table->newstat;
ptr_syscall_table->newstat = hook_newstat_fun;
original_newstat_syscall = original_newstat_addr;

original_kill_addr = ptr_syscall_table->kill;
ptr_syscall_table->kill = hook_kill_fun;
original_kill_syscall = original_kill_addr;

original_getsid_addr = ptr_syscall_table->getsid;
ptr_syscall_table->getsid = hook_getsid_fun;
original_getsid_syscall = original_getsid_addr;

original_getpgid_syscall = ptr_syscall_table->getpgid;
ptr_syscall_table->getpgid = hook_getpgid_fun;
original_getpgid_handler = original_getpgid_syscall;

original_newlstat_syscall = ptr_syscall_table->newlstat;
ptr_syscall_table->newlstat = hook_newlstat_fun;
original_newlstat_handler = original_newlstat_syscall;
```

Рисунок 17. Переопределение системных вызовов

Полный перечень подменяемых системных вызовов: execveat, execve, newfstatat, mmap, openat, newlstat, getdents64, newfstat, getsid, newstat, getpgid, close, rmdir, open, getdents, write, kill, read.

Почти все системные вызовы, перехватываемые вредоносным модулем, следуют единой логике: обработчик сначала анализирует переданные аргументы, а затем принимает решение о модификации возвращаемого пользователю ответа.

Например, вызовы newlstat и newstat, предназначенные для получения информации о файлах, в обычных условиях возвращают полный набор данных, тогда как их обновленные версии определяют, к какому объекту файловой системы запрашивается доступ, и, если это требуется, скрывают файлы вредоноса, подменяя возвращаемые значения. Ниже приведен список скрываемых от пользователя объектов:

- /proc
- /sys/module/audit/initstate
- /sys/module/audit/holders
- /sys/module/audit/refcnt
- /sys/module/audit
- /usr/share

Отдельного внимания заслуживают переопределенные вызовы rmdir, read и write, содержащие ключевую для дальнейшего использования модуля логику.

2.4.3.1. Механизм стилера: перехват системных вызовов write и read

Для реализации механизма стилера осуществляется подмена системных вызовов write и read. Переопределенный write анализирует передаваемые на запись данные с целью обнаружения конфиденциальной информации, в частности строк, содержащих ключевые слова «password» или «passphrase». Переопределенный вызов read, в свою

очередь, предназначен для перехвата и сохранения закрытых криптографических ключей PEM-формата, считываемых процессом при установке защищенного соединения.

В обоих случаях модуль сохраняет в памяти сами перехваченные данные и их тип, а также данные о процессе, инициировавшем системный вызов.

Помимо таких общих фраз, как «password», «login», «passphrase» и «...PRIVATE KEY...», мы обнаружили, что стилер дополнительно перехватывает данные, содержащие разные варианты написания слов «пользователя:» и «пароль:».

Факт поиска строк на русском языке сигнализирует о фокусе атакующих на русскоязычном сегменте интернета.

Особого внимания заслуживает механизм внедрения SSH-ключа, обеспечивающий атакующему устойчивый удаленный доступ. Для этого руткит перехватывает вызовы open и openat, отслеживая обращения к файлу authorized_keys — стандартному компоненту OpenSSH, расположенному в каталоге ~/.ssh/ конкретного пользователя и определяющему, какие публичные ключи разрешают доступ к соответствующей учетной записи без ввода пароля. При попытке чтения файла (например, в процессе подключения или его проверки) руткит на лету модифицирует содержимое: к оригинальным данным дописывается заранее подготовленный публичный ключ. При этом сам файл на диске остается неизменным: подмена производится исключительно в памяти, в момент вызова read.

В типичном сценарии атаки злоумышленник, имея на руках приватный ключ, инициирует SSH-подключение. При проверке ~/.ssh/authorized_keys сервер читает подмененный в памяти список ключей, распознает внедренный ключ как доверенный и открывает сессию без запроса пароля. После этого атакующий получает интерактивный доступ от имени целевой учетной записи и может скрытно выполнять команды и разворачивать дополнительные инструменты. Скрытность сохраняется до тех пор, пока загружен модуль руткита.

Даже смена пароля и отключение парольной аутентификации не устраняют угрозу: доступ по внедряемому ключу сохраняется и остается активным.

Наибольшую угрозу представляет ситуация, при которой публичный ключ добавляется в authorized_keys пользователя root: это дает атакующим максимальные привилегии и полный контроль над целевым узлом.

Исходя из представленной функциональности, можно предположить, что основным назначением руткита является закрепление в системе, а также перехват учетных данных, обрабатываемых в процессе аутентификации при установлении SSH-соединений. Эти данные впоследствии будут использованы не только для закрепления на начальном скомпрометированном узле, но и для дальнейшего перемещения внутри инфраструктуры жертвы.

2.4.3.2. Механизм взаимодействия с руткитом через переопределение системного вызова rmdir

Ключевую роль в работе руткита играет переопределенный системный вызов rmdir, который выступает внутренним каналом взаимодействия между пользовательской частью бэкдора и установленным модулем ядра. Инициатором этих вызовов является сам бэкдор, передавая в качестве аргумента не путь к каталогу, а специально сформированную строку-аргумент.

В основе данного механизма лежит перехват системного вызова: при обращении к rmdir перед стандартной операцией удаления руткит проверяет переданный путь и, если он начинается с «zarya», интерпретирует последующую часть строки как управляющую команду. Данная команда должна строго соответствовать определенной структуре, которая представлена на рис. 18.

```
rmdir zarya_[command]_[ argument]
```

Полный список команд, доступных атакующему, представлен в табл. 1.

Таблица 1. Перечень команд для управления

Команда	Аргумент	Описание
v	***Атакующие используют символ-заглушку «0», но может быть и любой другой***	Отображает версию установленного модуля
d		Извлекает данные, собранные стилером, и выводит их в пространство пользователя. После чего удаляет эти данные из памяти
c		Копирует встроенную конфигурацию для установки соединения (данные из секции .puma-config) в пространство пользователя
t		Выполняет тестовый вызов без возврата ошибки

u		Восстанавливает отображение скрытого модуля, возвращая его в список загруженных модулей
9		Отображает в пространство пользователя таблицу соответствий PID ↔ IP-адрес, связывающую их между собой (рис. 19)
0		Повышает привилегии у вызвавшего процесса
1	***Идентификатор процесса (PID)***	Принимает строку с идентификатором процесса, проверяет наличие соответствующей записи во внутренней таблице и при ее отсутствии добавляет переданный PID в список скрываемых. Данная команда также проверяет актуальность текущих записей, удаляя сведения о завершившихся процессах и их IP-адресах. Используется совместно с командой «5» при установке соединения
5	***IP-адрес***	Команда принимает IP-адрес в виде строки, преобразовывает его в 32-битное число и, при отсутствии соответствующей записи, добавляет его во внутреннюю таблицу скрываемых подключений (см. команду «9»). Используется совместно с командой «1» при установке соединения
k	***Идентификатор процесса (PID)***	Устанавливает PID руткита

Было установлено, что, помимо перечисленных выше команд, при вызове rmdir с аргументом zarya или zarya_ (без указания конкретной команды) у инициировавшего вызов процесса повышаются привилегии.

```
PUMA <module_vers>      Kitsune PID <Rootkit_PID>
PID       |IP                 |
----------+-------------------+
<PID_1>   |<IP_1>             |
<PID_2>   |<IP_2>             |
<PID_3>   |<IP_3>             |
<PID_4>   |<IP_4>             |
```

Рисунок 19. Пример отображения команды «9»

Анализ множества семплов показал, что в более поздних версиях функциональность модуля была расширена. В частности, была добавлена логика сокрытия используемых портов — аналогичная той, которая была описана для IP-адресов. Также были добавлены команды для управления портами (табл. 2).

Таблица 2. Добавленные в новых версиях команды

Команда	Аргумент	Описание	Замечание
7	Локальный порт	Добавляет локальный порт в список скрываемых (см. рис. 20), если значение уникально	Вызов rmdir с данными командами выполняется при получении соответствующей команды с C2-сервера (см. описание полезной нагрузки бэкдора)
8	Локальный порт	Удаляет локальный порт из списка скрываемых	

```
PUMA <LKM_module_version>       Kitsune PID <Rootkit_PID>
PID       |IP               |Local Port    |
----------+-----------------+--------------+
<PID_1>   |<IP_1>           |<LOCAL_PORT_1>
<PID_2>   |<IP_2>           |<LOCAL_PORT_2>
<PID_3>   |<IP_3>           |<LOCAL_PORT_3>
----------+-----------------+--------------+
```

Рисунок 20. Пример отображения обновленной команды «9»

2.4.3.3. Использование ftrace-хуков

Помимо перехвата системных вызовов, данный руткит использует механизм ftrace для установки хуков на заранее определенный набор функций ядра Linux. Адреса этих функций вычисляются динамически с помощью ранее полученного указателя на kallsyms_lookup_name. Вычисленные адреса используются для корректной установки хуков.

Руткит дополнительно анализирует каждую целевую функцию, определяя в ее коде участок, наиболее подходящий для внедрения хука. После чего с помощью функций ftrace_set_filter_ip и register_ftrace_function регистрирует обработчик, перенаправляющий выполнение функций на их подмененную реализацию.

Полный перечень функций ядра, подвергающихся перехвату, можно разделить на две группы: полностью отключаемые руткитом и те, поведение которых переопределяется. В первую группу входят функции, связанные с механизмами контроля доступа: selinux_file_open, selinux_file_permission, avc_has_perm, file_map_prot_check, selinux_inode_setattr, selinux_inode_permission, selinux_socket_bind и selinux_socket_connect. Обработчики для них полностью заменяют оригинальную логику на простую заглушку, которая всегда завершает выполнение без ошибок (return 0). В итоге любые проверки или действия, которые должны были выполняться, фактически отключаются, поскольку система считает, что они завершились успешно.

Ко второй группе относятся функции sk_diag_fill, tpacket_rcv, tcp4_seq_show, kernel_clone или _do_fork (в зависимости от версии модуля PUMAKIT), а также nf_hook_slow. В более поздних версиях к этому списку добавляется функция inet_sk_diag_fill. Обработчики для каждой из них не блокируют выполнение оригинального кода полностью, а осуществляют предварительный анализ входных аргументов. Во всех перечисленных случаях обработчики извлекают IP-адрес из передаваемых в функцию аргументов и сравнивают их с внутренним списком IP-адресов, хранящимся в структуре руткита (см. рис. 19 и 20). Если среди переданных аргументов обнаруживается совпадение с одним из имеющихся в структуре адресов — оригинальная функция ядра не вызывается, а обработчик вместо этого сразу возвращает нулевой результат, тем самым подавляя выполнение исходной функции. Такой механизм переопределения позволяет эффективно скрывать нелегитимные подключения, процессы и сетевую активность, обеспечивая их невидимость для средств мониторинга и анализа.

Особого внимания заслуживает функция nf_hook_slow, поскольку перехват ее вызова позволяет злоумышленнику обойти работу файрвола на уровне ядра, исключая проверку пакетов механизмами фильтрации.

В стандартной реализации Netfilter функция nf_hook_slow выполняет последовательный вызов всех зарегистрированных в ядре хуков, включая обработчики iptables и nftables. Именно эти хуки принимают окончательное решение о том, пропустить, заблокировать или перенаправить пакет.

Однако в данном случае внедренный обработчик анализирует пакет до его передачи в оригинальную функцию, извлекая исходный и целевой IP-адреса и сравнивая их с внутренним списком, хранящимся в руките. Если хотя бы один из них совпадает — обработчик возвращает NF_ACCEPT (1), сразу принудительно разрешая прохождение пакета без его передачи в nf_hook_slow.

Такой подход позволяет атакующему полностью скрывать трафик от механизмов мониторинга и обходить политики файрвола, поскольку пакеты, принудительно принимаемые обработчиком, минуют стандартные процессы фильтрации и анализа. Таким образом, файрвол не имеет возможности зафиксировать, проанализировать или заблокировать такой трафик. В результате злоумышленник может беспрепятственно устанавливать скрытые соединения, не оставляющие следов в системах мониторинга и журналирования, полностью избегая наложенных ограничений безопасности.

2.4.3.4. Удаление из списка модулей и инъекция бэкдора libs.so

После завершения установки хуков и подмены адресов системных вызовов на собственные обработчики руткит принимает меры по сокрытию своего присутствия в системе. Для этого он удаляет себя из списка загруженных модулей ядра, модифицируя указатели в двусвязном списке, хранящим все активные модули. В результате стандартные инструменты мониторинга, такие как команда lsmod или просмотр файла /proc/modules, перестают отображать модуль руткита.

Далее руткит запускает отдельный поток, выполняющий функцию, отвечающую за инъекцию и дальнейшее поддержание бэкдора. В ней в бесконечном цикле производится проверка существования процесса бэкдора с помощью функций find_get_pid и pid_task. Если целевой процесс отсутствует или с момента последней проверки прошло более пяти секунд — проверка выполняется повторно и при необходимости руткит инициирует запуск бэкдора, обеспечивая его постоянное присутствие в системе. Для этого с помощью механизма запуска пользовательских процессов из ядра call_usermodehelper через оболочку /bin/sh выполняются две команды:

1. truncate -s 0 /usr/share/zov_f/zov_latest;
2. cat /dev/null 1>/dev/null.

При этом, помимо самих команд, call_usermodehelper также получает указатель на массив переменных окружения:

- SHELL=sh
- HOME=/
- LD_PRELOAD=/lib64/libs.so
- PATH=/sbin:/bin:/usr/sbin:/usr/bin

Среди этих переменных особое значение имеет LD_PRELOAD=/lib64/libs.so, обеспечивающая инъекцию бэкдора.

В результате первая команда обнуляет определенный файл, используемый бэкдором (его работа будет рассмотрена далее), фактически скрывая его содержимое, пока он находится в неактивном состоянии. Вторая команда, по сути, не выполняет никаких значимых действий: она просто считывает пустой файл и перенаправляет вывод в /dev/null. Однако ее запуск необходим для выполнения с заданным окружением, которое позволяет активировать инъекцию бэкдора без заметных следов в системе.

Итак, описанный выше механизм обеспечивает автоматическое восстановление и постоянное присутствие в системе.

2.4.4. Четвертая стадия: libs.so

После того как LKM-руткит вызывает call_usermodehelper с модифицированным окружением (переменная LD_PRELOAD указывает на встроенный в рутките файл libs.so), бэкдор немедленно загружается в адресное пространство созданного процесса и начинает выполнение. Поскольку вызванная команда завершается практически сразу, бэкдор дополнительно предпринимает шаги, направленные на закрепление в системе и обеспечение возможности длительной автономной работы.

Для этого выполняется превращение процесса в демон, позволяя ему работать в фоновом режиме без привязки к терминалу и управляющей сессии. Достигается это двумя последовательными вызовами функции fork: первый создает дочерний процесс и сразу завершает родительский, разрывая исходную связь с запускающим процессом. Затем дочерний процесс вновь вызывает fork, а следом за ним и setsid, чтобы создать новую сессию, полностью отсоединенную от управляющего терминала. Эти действия окончательно разрывают связь бэкдора с его первоначальным контекстом запуска, гарантируя, что процесс перестает зависеть от сигналов или жизненного цикла родительского процесса.

Вслед за этим бэкдор изменяет текущий рабочий каталог на корневой (chdir("/")), чтобы исключить зависимость от исходного пути, устанавливает маску прав доступа в значение umask(0), предотвращая возможные ограничения при создании файлов, и закрывает все открытые файловые дескрипторы (включая стандартные потоки ввода, вывода и ошибок), предотвращая случайный вывод данных в консоль.

Для получившегося в результате процесса с помощью вызова функции getpid определяется PID, а после, с помощью полученного значения, выполняется вызов rmdir, переопределенного руткитом, с параметром zarya_k_<PID> — для связывания между собой бэкдора и руткита.

После этого бэкдор переходит к следующему этапу своей работы: он проверяет наличие конфигурационных данных, необходимых для установки соединения с C2-сервером, в секции .konfig, расположенной в памяти процесса. В случае отсутствия этих данных бэкдор извлекает необходимую конфигурацию из секции .puma-config, расположенной в памяти ранее загруженного LKM-модуля. Для извлечения и последующего заполнения собственной конфигурационной секции используется механизм, основанный на вызове функции rmdir, но уже с аргументом zarya_c_0. Особого внимания заслуживает структура извлекаемой конфигурации, представленная на рис. 21.

```
struct intEntry {
    _BYTE fieldType; // int (0x10)
    char fieldName[];
    _DWORD data;
}

struct dataEntry {
    _BYTE fieldType; // string (0x2), bytes (0x5)
    char fieldName[];
    _DWORD dataSize;
    _BYTE data[dataSize];
}
```

Рисунок 21. Структура конфигурационной секции модуля .puma-config

В конфигурации могут быть указаны следующие данные.

Таблица 3. Перечень команд для управления

Название записи	**Тип структуры записи**	**Описание**	**Значение по умолчанию**
ping_interval_s	Int32	Интервал между попытками опроса C2-сервера при отсутствии установленного соединения	5 секунд
hibernate_s	Int32	Время сна при неудачном подключении к C2-серверу (после использования должно быть повторно получено)	0 секунд
session_timeout_s	Int32	Время жизни сессии	3 секунды
c2_timeout_s	Int32	Максимальное время, в течение которого бэкдор подключен к одному серверу	43 200 секунд (12 часов)
jitter_s	Int32	Величина случайного отклонения от ping	50
tag	String	Тег жертвы	«x»
cert	Binary	Отпечаток сертификата (SHA-256), используемого для подключения к серверу	Отсутствует
sni	String	Доменное имя в сертификате, используемом для подключения к C2-серверу	
dns_s	Int32	Время жизни кэшированных DNS-записей	3600 секунд
gw_p	String	Порт, на котором бэкдор будет работать в режиме прокси-сервера	Отсутствует
s_a<№>	String	Адрес C2-сервера	
s_p<№>	String	Порт C2-сервера	
s_c<№>	String	Протокол C2-сервера	

Важно отметить, что конфигурация бэкдора способна включать в себя до 32 записей об управляющих серверах (s_*<№>).

Получив необходимые данные конфигурации, бэкдор формирует уникальный идентификатор зараженного узла (agent_id). Алгоритм его вычисления аналогичен тому, который был описан в загрузчике. Сформированное значение служит для однозначного определения устройства при последующем взаимодействии с C2-сервером.

2.4.4.1. Сохранение данных

Следующим шагом становятся извлечение и обработка данных, ранее полученных стилером. Для этого бэкдор выполняет вызов rmdir с аргументом zarya_d_0. Данный вызов выполняется циклически — не более восьми раз подряд либо до момента, пока его ответы не перестанут содержать информацию. Следует отметить, что при каждом обращении с таким аргументом получаемые данные удаляются из памяти модуля, благодаря чему на каждом следующем шаге возвращается новая, еще не обработанная информация.

В ранних версиях модуля структура перехваченных данных содержит такие поля, как тип перехваченных данных, UID процесса (данные которого были перехвачены) и непосредственно сама конфиденциальная информация (см. рис. 22).

```
struct oldDumpVersion {
    _BYTE returnMarker;       // тип перехваченных данных
    _DWORD hookedProcUID;
    _BYTE hookedData[0x8000]; // состоит из имени процесса и перехваченных данных
}
```

Рисунок 22. Структура данных, возвращаемых в пространство пользователя при вызове rmdir с аргументом zarya_d_0 (в ранних версиях)

Сохраненный и переданный в бэкдор UID используется в качестве ключа для получения полной информации о соответствующей учетной записи системы: сначала бэкдор пытается обратиться к системному файлу /etc/passwd, чтобы, используя полученный идентификатор, извлечь информацию о зарегистрированном в системе пользователе. Если доступ к файлу невозможен или нужная запись не обнаружена — UID преобразуется в строковый формат и используется для формирования запроса к демону кэширования учетных данных — Name Service Cache Daemon. На основе полученных данных бэкдор формирует строку с информацией о пользователе, используя нулевой байт (0×00) в качестве разделителя.

<username> <password> <UID> <GID> <GECOS> <home_directory> <login_shell>

Рисунок 23. Сохраняемая информация о пользователе (usrData)

В более поздних версиях модуля, помимо UID, злоумышленники также сохраняют EUID и SUID процесса.

```
struct newDumpVersion {
    _BYTE returnMarker;       // тип перехваченных данных
    _DWORD hookedProcUID;
    _DWORD hookedProcEUID;
    _DWORD hookedProcSUID;
    _BYTE hookedData[0x8000]; // состоит из имени процесса и перехваченных данных
}
```

Рисунок 24. Структура данных, возвращаемых в пространство пользователя при вызове rmdir с аргументом zarya_d_0 (в поздних версиях)

Добавленные идентификаторы также используются в качестве ключей для извлечения информации о пользователе из файла /etc/passwd. Однако в отличие от более ранних версий, бэкдор сохраняет не всю запись целиком, а только имя пользователя, соответствующее каждому из указанных идентификаторов (так как эти идентификаторы могут указывать на разных пользователей). Полученные значения формируются в строку в специальном формате.

```
uid=<uid_value>(<uid_username>) euid=<euid_value>(<euid_username>) suid=<suid_value>(<suid_username>)
```

Рисунок 25. Сохраняемая информация о пользователе (usrData)

Следует отметить, что более поздние версии бэкдора имеют обратную совместимость со старыми версиями модуля (сохраняющими исключительно UID). Такая возможность была добавлена вследствие того, что злоумышленники не предусмотрели обновление модуля ядра, в отличие от бэкдора (описание полезной нагрузки бэкдора представлено в табл. 5). Для таких случаев применяется специальный формат строки (рис. 26).

```
uid=<uid_value>(<uid_username>)
```

Рисунок 26. Поддержка предыдущих версий модуля (usrData)

В более поздних версиях модуля, перед тем как полученные от руткита данные будут записаны в журнал, бэкдор хеширует их с помощью алгоритма FNV-1: если такой хеш уже присутствует в памяти — увеличивается счетчик обращений и запись пропускается, а если его нет, то он добавляется в таблицу (или замещает наименее используемый при переполнении). Таким образом злоумышленники исключают запись повторяющихся данных.

Если же запись является уникальной — сведения о пользователе, перехваченные данные и информация о процессе-источнике кодируются в Base64 и записываются в журнал в порядке, определяемом типом перехваченных данных. Каждая запись начинается с временной метки, за которой следует сформированная CSV-строка, элементы которой разделены запятыми.

```
TIMESTAMP,BASE64(usrData),BASE64(hookedProcName),,,BASE64(hookedData)     // перехваченная информация - парольная фраза
TIMESTAMP,BASE64(usrData),BASE64(hookedProcName),,BASE64(hookedData),     // перехваченная информация - криптографический ключ
TIMESTAMP,BASE64(usrData),BASE64(hookedProcName),BASE64(hookedData),,     // перехваченная информация - пароли или логин
```

Рисунок 27. Данные для записи в zov_logs.txt

Получившаяся запись добавляется в конец файла /usr/share/zov_f/zov_logs.txt. Перед записью бэкдор обязательно проверяет источник, с которым связаны перехваченные данные. Если оказывается, что обработке подвергается собственный процесс ведения журнала — запись немедленно прекращается, что исключает саможурналирование и зацикливание записей.

2.4.4.2. Взаимодействие с C2-сервером

После того как перехваченные данные были записаны в журнал, процесс предпринимает попытку установить соединение с C2-сервером для последующего взаимодействия. Для этого, при отсутствии активного соединения, бэкдор сравнивает текущее время с сохраненной временной меткой последнего успешного взаимодействия с C2-сервером:

1. При отсутствии ранее установленных соединений бэкдор фиксирует текущее время в качестве отправной точки и инициирует цикл перебора доступных конфигураций подключения, полученных из ранее заполненной секции .konfig. Для каждой записи перед попыткой установить соединение выполняется DNS-разрешение, по результатам которого в случае успеха сохраняется временная метка. Цикл ограничен 100 итерациями и продолжается до тех пор, пока хотя бы одна из записей не будет успешно разрешена.
2. Если метка последнего успешного взаимодействия с сервером установлена — вычисляет время, прошедшее с момента предыдущего сеанса связи. Если это значение не превышает c2_timeout_s, процесс пытается установить соединение, используя последнюю использованную конфигурацию. В противном случае выбирается следующая запись из секции .konfig, для которой проводится проверка актуальности DNS-разрешения: если с момента последнего запроса прошло менее dns_s секунд — используется ранее полученный адрес. Иначе инициируется новое разрешение, по завершении которого обновляется временная метка.
3. В обоих случаях перед инициализацией соединения по очереди вызываются команды:

- rmdir zarya_1_<PID> — для скрытия текущего PID процесса;
- rmdir zarya_5_<IP> — для скрытия IP-адреса разрешенного C2-сервера.

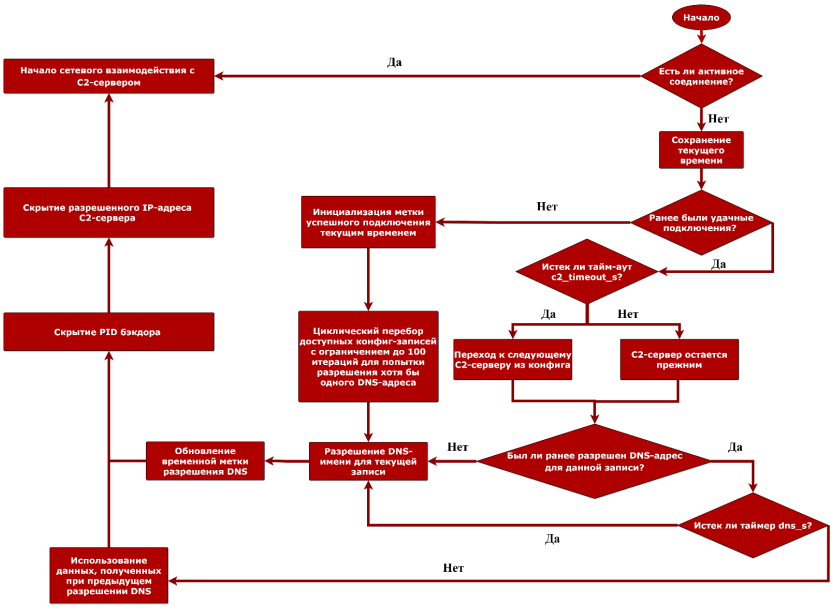

Рисунок 28. Алгоритм соединения с C2-сервером

Для обеспечения регулярного взаимодействия с сервером в бэкдоре предусмотрен механизм внутренних таймеров, контролирующих периодичность обращений (см. структуру конфигурации). При инициализации соединения формируется пустое сообщение с заголовком, содержащим основную информацию о зараженной системе (команда 4097).

Таблица 4. Базовый заголовок для всех сообщений

Поле структуры сообщения	**Добавляемые данные**
agent_id	Уникальный идентификатор жертвы
v	Версия бэкдора
pv	Версия модуля
p	PID процесса
log_t	Время, прошедшее с момента последней записи в zov_logs.txt
uptime	Общее время работы руткита
cmd_id	Заголовок отправленных данных
cmd_type	Номер выполненной команды
jitter_s	Верхняя граница величины случайной задержки, добавляемой к основному интервалу

tag	Тег жертвы
dpi	Уникальный идентификатор сообщения, представляющий собой случайное значение из диапазона [0, 999]

Важно отметить, что каждое сообщение, отправляемое на C2-сервер, начинается с этого заголовка и дополняется результатами выполнения соответствующих команд (см. табл. 5).

После формирования и отправки стартового сообщения бэкдор переходит в режим ожидания ответа от C2-сервера, включающего тип команды (cmd_type), идентификатор передаваемых данных (cmd_id) и непосредственно полезную нагрузку. Форматы отправляемых и получаемых сообщений, а также описание функциональных возможностей бэкдора приведены в табл. 5.

Таблица 5. Возможности бэкдора

Тип команды (cmd_type)	**Описание команды**	**Комментарий**	**Структура ответного сообщения**
4097	Отправляет на C2-сервер блок данных с основной информацией о зараженной системе (heartbeating)	Каждое сообщение начинается с этого блока	Только заголовок (см. табл. 4), в котором будет указан тип выполненной команды
4100	Получает и записывает новую версию бэкдора в /usr/share/zov_f/zov_latest	—	
4101	Запускает реверс-шелл	Используется псевдотерминал (pty)	
4103	Удаляет или добавляет (см. табл. 2) локальный порт для работы в режиме прокси	Номер порта и выбранная команда поступают с C2-сервера	
4098	Отправляет содержимое файла /usr/share/zov_f/zov_logs.txt	При превышении порога в 8 МБ файл очищается	«logs» + <содержимое zov_logs.txt>
4096	Собирает и отправляет системную информацию	Вызов команд с помощью /bin/sh: • «cat /etc/*-release»; • «uname -a»; • «ip a»; «ifconfig»; • «hostname»; • «users»	Полученные в результате данные группируются: • «os» + <data> • «uname» + <data> • «ip1» + <data> • «ip2» + <data> • «hostname» + <data> • «tag» + <data>
4099	Выполняет произвольную шелл-команду и отправляет ее результат	Команды выполняются с помощью /bin/sh	«result» + <stdout> «output» + <stderr>
4102	Обновляет внутренние таймеры	Обновляемые параметры: • «c2_timeout_s»; • «ping_interval_s»; • «session_timeout_s»; • «jitter_s»; • «dns_s»; • «hibernate_s»	Значения конфигурационных параметров: • «c2_timeout_s»; • «ping_interval_s»; • «jitter_s»; • «session_timeout_s»; • «dns_s»; • «tag»; • «cert»;

			• «sni»; • «s_a<№>»; • «s_p<№>»; • «s_t<№>»;

2.4.4.3. Используемые сертификаты

В ходе анализа доменов, использовавшихся для C2-серверов, было установлено, что ряд из них применяет весьма примечательный SSL-сертификат (рис. 29). Он интересен тем, что в качестве Issuer и Subject злоумышленники указали организацию «FSB» и адрес «2 Bolshaya Lubyanka Street».

Ранее, в ходе одного из расследований, мы встречали использование данного сертификата группировкой (Ex)Cobalt. Поэтому мы связываем руткит PUMAKIT с данной группировкой и рассматриваем его как часть ее инструментария.

Рисунок 29. Используемый сертификат

Данный сертификат был использован следующими C2-серверами:

- cckitsfrp1.n3x1lo.pro
- qdkitsorp2.n3x1lo.pro
- cddcvesfhfp1.wris.monster
- deefveskiip2.wris.monster
- cumfpo90sing.agddns.net
- procdia42ecte.agddns.net
- viedeu98.agddns.net
- laipros50.agddns.net
- fira24sonstablee.agddns.net
- chronback49in.duckdns.org

Злоумышленники маскируют сертификат, используемый для шифрования взаимодействия бэкдора с C2-сервером, при помощи Server Name Indication (SNI), расширения протокола TLS. Настоящий сертификат выдается по имени, указанному в поле sni конфигурации бэкдора. Таким образом, они могут избежать обнаружения других C2-серверов по сертификату и получения других сертификатов без знания имени сервера из конфигурации. Сертификат является

заглушкой, которая возвращается сервером в том случае, если не указано имя сервера в запросе. Она не используется при взаимодействии с C2-сервером.

Отпечаток сертификата (SHA-256) и соответствующие им SNI, использовавшиеся злоумышленниками и обнаруженные нами в конфигурационной секции, представлены в таблице 6.

Таблица 6. Информация о сертификатах

SHA-256	**SNI**
29AD1A06DCA85041E793A8BF2F966B6A7CF3F35904FB3E8648A7F97D9A211F8D	run.sssddd.org
0B3B6E06CB7B6C25066B0DAEA6CF2D6EEA57D33FB58EF66EBAEF2107BA2A92B0	zfs.wefwe.net

2.4.5. Эволюция

Помимо руткита PUMAKIT, группировка (Ex)Cobalt регулярно использует и другой инструмент — Facefish, частично повторяющий его функциональность. В ходе расследований инцидентов наши специалисты неоднократно фиксировали случаи одновременного применения двух этих инструментов внутри одной и той же инфраструктуры.

Ситуация дополнительно усложняется тем, что ранее злоумышленники использовали еще один руткит — Kitsune, который был подробно изучен нашими коллегами из BI.ZONE.

Сопоставив возможности всех трех инструментов и временные метки их появления, можно четко проследить, как эволюционировал инструментарий группировки с течением времени.

Таблица 7. Сравнение инструментов

	Facefish	Kitsune	PUMAKIT
Функции	• Кража учетных данных SSH-пользователей через перехват функций ssh/sshd • Сбор и отправка сведений об узле • Исполнение произвольных команд • Реверс-шелл • В новых версиях был также реализован перехват функции getdelim: при чтении конфигурационного файла /etc/ssh/sshd_config будет добавляться дополнительный порт, на котором сервис SSH принимает входящие соединения	• Кража различных учетных данных, включая данные SSH-пользователей • Сбор и отправка сведений об узле • Исполнение произвольных команд • Реверс-шелл • Получение информации путем выполнения заготовленных команд	• Кража различных учетных данных, включая данные SSH-пользователей • Сбор и отправка сведений об узле • Исполнение произвольных команд • Реверс-шелл • Получение информации путем выполнения заготовленных команд • Бэкдор отделен от руткита, появилась возможность обновление бэкдора • Проксирование трафика • Добавление собственных ключей в файл authorized_keys
Первое обнаружение	Февраль 2021 г.	Февраль 2022 г.	Март 2024 г.
Тип руткита	Userland	Userland	Kernel
Компоненты	• Дроппер	• Userland-руткит	• Дроппер

	• Userland-руткит с функциями бэкдора и стилера	с функциями бэкдора и стилера	• Загрузчик LKM-руткита • LKM-руткит с функциями стилера • Бэкдор
Способ закрепления в системе	Дроппер сохраняет userland-руткит в файл /lib64/libs.so и прописывает его в /etc/ld.so.preload	Userland-руткит сохраняется злоумышленниками в файл /lib64/libselinux.so (как в рутките Azazel) и прописывается в /etc/ld.so.preload	Подмена системного сервиса cron
Способ активации	При вызове функции bind() процессом sshd	При вызове функции bind() процессом sshd	Активируется сразу при старте подмененного cron-сервиса
Формат пакетов для общения с C2-сервером	Собственный формат пакетов	BSON	BSON
Хранение конфигурации	Хранится в «хвосте» дроппера	Хранится в файле /etc/config__hhide	Хранится в виде ELF-секции руткита, в некоторых версиях в файле /usr/share/zov_f/zov_config
Механизмы скрытности	Отсутствуют	Реализован механизм скрытия записи в /etc/ld.so.preload. Скрывает файлы, процессы и сетевые соединения, связанные с ними, на уровне пользователя через перехват функций libc	Полноценный руткит уровня ядра. Скрывает: • процессы, • файлы и каталоги, • сетевые соединения

При этом мы считаем, что Facefish не был разработан группировкой (Ex)Cobalt, а был приобретен на черном рынке или получен в результате утечки. В пользу данного предположения говорит тот факт, что используемая в Fasefish конфигурация расположена в конце файла и не упаковывается UPX, а просто зашифровывается (рис. 30). Таким образом, не имея исходного кода, злоумышленники могли беспрепятственно менять параметры конфигурации.

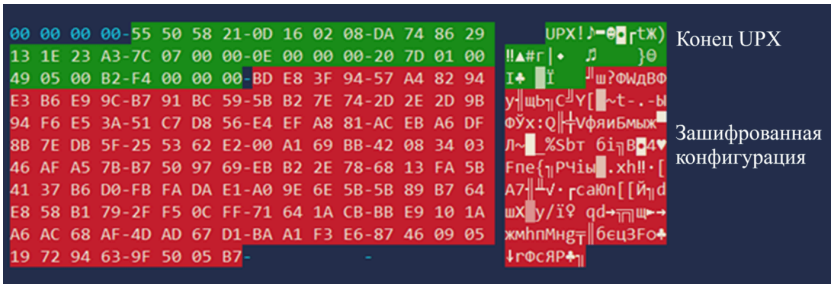

Рисунок 30. Расположение конфигурации в Facefish

Однако предоставляемой Facefish функциональности, по всей видимости, оказалось недостаточно, вследствие чего группировка разработала собственный руткит — Kitsune, во многом наследующий возможности своего предшественника. Примечательно, что в Facefish и в Kitsune запуск бэкдора привязан к вызову функции bind(), что указывает на прямую преемственность архитектурных решений.

В то же время имеются все основания предполагать, что Kitsune был создан на основе исходного кода руткита Azazel: помимо схожих архитектурных решений, шифрование строк, заложенное в Azazel (XOR с байтом 0xFE), сохранилось в Kitsune и в дальнейшем перешло в его следующую версию (рис. 31 и 32).

Рисунок 31. Механизм шифрования строк в Azazel

```
else
{
  decrypted_str = strdup(encrypted_str);
  Idx = 0LL;
  if ( strlen(decrypted_str) )
  {
    do
      decrypted_str[Idx++] ^= 0xFEu;
    while ( strlen(decrypted_str) > Idx );
  }
```

Рисунок 32. Механизм расшифровки строк в бэкдоре PUMAKIT

Переход от руткита Kitsune к PUMAKIT не был резким: между ними существовала промежуточная версия, известная как Megatsune. И хотя Megatsune по-прежнему оставался руткитом уровня пользователя, а его архитектура и логика напрямую наследовали решения Kitsune — именно на Megatsune злоумышленники начали тестировать бэкдор, позднее реализованный в PUMAKIT, сохранив ту же логику работы, набор команд и структуру конфигурации.

На основании проведенного анализа мы объединяем эти три инструмента в одно семейство — PUMA, предполагаемая цепочка эволюции которого представлена на рис. 33.

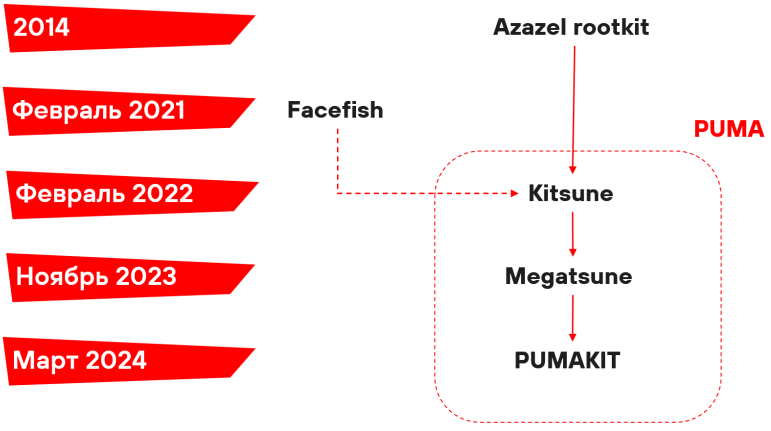

Рисунок 33. Эволюция семейства PUMA

2.4.6. Методы обнаружения и борьбы с руткитом в своей системе

В ходе анализа мы выявили в LKM-модуле важную уязвимость, проэксплуатировав которую можно обнаружить установленный модуль PUMAKIT через его же интерфейс: руткит не проверяет, какой именно процесс вызывает переопределенный rmdir для выполнения команд. Таким образом, вызвав rmdir с определенными аргументами, можно однозначно определить присутствие руткита в системе.

Вы можете легко проверить свою систему на наличие руткит-модуля, воспользовавшись нашим скриптом для детекта.

```
 import ctypes
import subprocess
import os

SYS_rmdir = 84

buffer_size = 16
path_buf = ctypes.create_string_buffer(buffer_size)
ctypes.memmove(path_buf, b"zarya.u\0", 7)

libc = ctypes.CDLL("libc.so.6", use_errno=True)
ret = libc.syscall(SYS_rmdir, path_buf)

try:
    proc = subprocess.Popen(
        "lsmod | grep audit",
        shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        universal_newlines=True
)
    output, _ = proc.communicate()

    if output.strip():
        path_buf2 = ctypes.create_string_buffer(buffer_size)
        ctypes.memmove(path_buf2, b"zarya.t.0\0", 9)
        ret2 = libc.syscall(SYS_rmdir, path_buf2)

        if ret2 == 0:
            print(f"Puma detected on this machine, module info:\n{output.strip()}")
        else:
            print("Puma wasn't detected at this machine")
    else:
        print("Puma wasn't detected at this machine")

except Exception as e:
    print("Error:", e)
```

Листинг 3. Скрипт для обнаружения скрытого модуля руткита

Уязвимость трудноустранима, поскольку злоумышленники, используя возможности бэкдора, могут обновлять лишь его код, но не код загруженного в систему модуля ядра. Для этого потребуется значительно больше усилий.

Кроме того, чтобы убедиться в отсутствии вмешательства руткита в файл authorized_keys, выполните в своей системе следующий скрипт.

```
 import difflib
import os
import shutil
import subprocess
import sys
import tempfile
from pathlib import Path

def read_with_binary(binary_path: str, target_file: str) -> bytes:
    try:
        res = subprocess.run(
            [binary_path, target_file],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            check=True
        )
        return res.stdout
    except subprocess.CalledProcessError:
        print(f"Ошибка: не удалось запустить {binary_path}", file=sys.stderr)
        sys.exit(1)

def detect_in_memory_modification(authorized_keys: Path):
```

```
    clean = read_with_binary("/bin/cat", str(authorized_keys))

    with tempfile.TemporaryDirectory() as td:
        fake_ssh = Path(td) / "ssh"
        shutil.copy2("/bin/cat", fake_ssh)
        fake_ssh.chmod(0o755)
        modified = read_with_binary(str(fake_ssh), str(authorized_keys))

    if clean != modified:
        print("Обнаружено вмешательство руткита!")
        clean_lines = clean.decode(errors="ignore").splitlines()
        mod_lines = modified.decode(errors="ignore").splitlines()
        diff = difflib.ndiff(clean_lines, mod_lines)
        added = [
            line[2:]
            for line in diff
            if line.startswith('+ ') and line[2:].strip() != ''
        ]
        if added:
            print("\nДобавленные строки:")
            for l in added:
                print(f"  + {l}")
        else:
            print("Изменения обнаружены, но добавленных строк не найдено.")
    else:
        print("Не обнаружено подмены содержимого authorized_keys при запуске ssh-процесса.")

if __name__ == "__main__":
    path = sys.argv[1] if len(sys.argv) > 1 else os.path.expanduser("~/.ssh/authorized_keys")
    ak = Path(path)
    if not ak.is_file():
        print(f"Ошибка: файл {ak} не найден.", file=sys.stderr)
        sys.exit(1)
    detect_in_memory_modification(ak)
```

Листинг 4. Скрипт для проверки возможной подмены содержимого файла authorized_keys

Скрипт последовательно читает файл authorized_keys через настоящий /bin/cat и через его копию, переименованную в ssh, сравнивает оба вывода и при обнаружении различий выводит только те строки (ключи), которые руткит «подмешивает» на лету.

Отключение локального файрвола, реализованное в рутките, наглядно демонстрирует, насколько легко злоумышленники могут обойти встроенные средства защиты, если те функционируют исключительно внутри целевой системы. Чтобы противостоять подобным атакам, критически важно выносить фильтрацию трафика за пределы потенциально скомпрометированной среды. Одним из надежных решений может стать использование внешнего межсетевого экрана нового поколения, например PT NGFW — продукта, который не только отслеживает и блокирует подозрительный трафик на уровне приложений, но и позволяет своевременно выявлять сложные сетевые атаки, включая попытки обхода традиционных механизмов защиты.

2.5. Octopus и его щупальца

Еще одним заслуживающим внимания элементом инструментария группы является реализованный на Rust инструмент Octopus, который может применяться для повышения привилегий и закрепления в скомпрометированной Linux-системе.

2.5.1. Локальные крейты

В реализации инструмента применялся ряд недоступных публично крейтов[1]. Приведем их краткое описание.

[1] Crate, контейнер — модуль, обособленная единица компиляции в языке Rust.

Таблица 8. Локальные крейты Octopus

Крейт	Назначение
octopus	Обертка для управления функциональностью, предоставляемой крейтами, приведенными ниже

octorepl	Реализация read–eval–print loop (REPL) на основе крейта clap
octosys	Реализация взаимодействия с операционной системой
octoproc	Реализация взаимодействия с оболочкой во время запуска эксплойтов
octolog	Реализация журналирования
leech	Реализация библиотеки патчинга на основе крейта gimli, также присутствует возможность сетевого взаимодействия
baron	Реализация CVE-2021-3156 (уязвимость heap overflow в утилите sudo), переписанная на Rust
route4-filter	Реализация CVE-2022-2588 (уязвимость double free в функции route4_change), переписанная на Rust
looney_tunables	Реализация CVE-2023-4911 (уязвимость buffer overflow в ld.so), переписанная на Rust
infect	Реализация заражения исполняемых файлов
gtfo	Реализация поиска GTFOBins

2.5.2. Поток управления

Поток управления базируется на локальном крейте octorepl, который основан на крейте clap и реализует собственный REPL. Если Octopus будет запущен без аргументов, откроется интерактивная оболочка — REPL. При вызове команды help из оболочки будет выведен список команд и их описание.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./tmp.CGVX9dKVLX
[11840]> help
octopus 16.7.0


Usage: app [COMMAND]

Commands:
  detect    Detect CVEs that should be working on this system
  do        Run inidividual exploit
  su        Detect and run exploits to get root shell
  glue      Patch binary to start binary
  netpatch  Patch binaries to hijack connections and also to restore itself
  help      Print this message or the help of the given subcommand(s)

Options:
  -h, --help  Print help

[11840]>
```

Рисунок 34. Список команд, доступных в REPL

В режиме REPL нет возможности запускать часть команд, которые доступны только в режиме запуска через интерфейс командной строки.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./tmp.CGVX9dKVLX help
Usage: tmp.CGVX9dKVLX [OPTIONS] [COMMAND]

Commands:
  detect           Detect CVEs that should be working on this system
  system-id        Print system id and exit
  do               Run inidividual exploit
  su               Detect and run exploits to get root shell
  glue             Patch binary to start binary
  netpatch         Patch binaries to hijack connections and also to restore itself
  netpatch-update  Update binaries previously patched by netpatch
  check-bin        Check a binary
  gtfo             GTFO command group
  help             Print this message or the help of the given subcommand(s)

Options:
  -d, --delete-on-success
          Delete itself when operation succeeds

  -D, --delete-on-start
          Delete itself on start

          This means binary will not be there if you need to run it again. Works in interacive mode too. Works even if exploit crashes.

  -h, --help
          Print help (see a summary with '-h')

  -V, --version
          Print version
```

Рисунок 35. Список команд, доступных в командной строке

Если в режиме REPL попытаться ввести команду, которая доступна только в командной строке, оболочка подсветит такую команду красным цветом, в ином случае зеленым. Стоит отметить, что в оболочке есть автодополнение

команд, а также что за дополнительной информацией можно обратиться на внутреннюю вики.

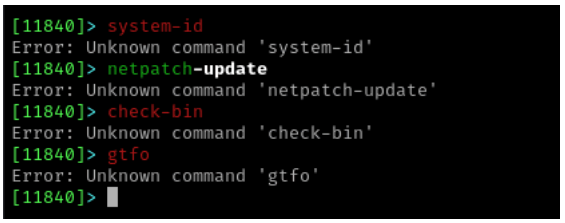

Рисунок 36. Отображение недоступных команд в REPL

2.5.3. Команды

Инструмент Octopus содержит следующий набор команд.

Таблица 9. Команды Octopus

Команда	Описание
detect	Проверяет, какие эксплойты повышения привилегий работают в системе. Эксплойты берутся из команды do
system-id	Генерирует идентификатор жертвы посредством сбора информации о системе и хеширования ее алгоритмом MD5
do	Запуск набора эксплойтов для повышения привилегий: • pwnkit — запуск эксплойта CVE-2021-4034; • looney-tunables — запуск эксплойта CVE-2023-4911; • baron — запуск эксплойта CVE-2021-3156; • route4-filter — запуск эксплойта CVE-2022-2588
su	Сканирование системы для обнаружения компонентов, содержащих уязвимости, позволяющие повысить привилегии. Использует команду detect
glue	Модификация легитимных утилит, установленных в системе, для обеспечения запуска вшитой полезной нагрузки
netpatch	Патчинг бинарных файлов, для перехвата соединения и восстановления себя
netpatch-update	Обновление бинарных файлов, ранее пропатченных с помощью команды netpatch
check-bin	Проверка бинарного файла на роль кандидата для заражения
gtfo	Поиск GTFOBins
help	Вывод информации о командах

2.5.4. Полезная нагрузка

Octopus несет в себе 4 полезных нагрузки:

- Руткит libzst.so.0 (мы дали название ему Spawner). Руткит для межпроцессорного взаимодействия.
- Руткит libsockopt.so.1 (мы дали название ему TransMarker). Руткит для межпроцессорного взаимодействия.
- Руткит libsockopt.so.2 (mosquito). Руткит для межпроцессорного взаимодействия.
- Бэкдор mycelium. Инструмент для удаленного управления зараженной системой.

Полезная нагрузка находятся в секции .rodata в сериализованном (CBOR) и сжатом (zstd) виде.

```
.rodata:00000000003B3D8E modules         db 28h, 0B5h, 2Fh, 0FDh, 0, 68h, 14h, 12h, 0, 86h, 54h
.rodata:00000000003B3D8E                                         ; DATA XREF: octopus__get_modules+1B↑o
.rodata:00000000003B3D99                 db 4Fh, 3Dh, 0E0h, 98h, 69h, 0Ch, 24h, 21h, 64h, 0Ah, 21h
.rodata:00000000003B3DA4                 db 0DCh, 30h, 95h, 5Ch, 0C0h, 48h, 10h, 6Ah, 70h, 80h
.rodata:00000000003B3DAE                 db 0Dh, 43h, 28h, 0BBh, 94h, 0F8h, 10h, 0E0h, 47h, 57h
.rodata:00000000003B3DB8                 db 0ACh, 31h, 0B9h, 0E6h, 73h, 0B4h, 0Dh, 0EBh, 0BCh, 0E4h
.rodata:00000000003B3DC2                 db 0B9h, 0C8h, 15h, 67h, 50h, 0D9h, 42h, 6, 0CFh, 8Ah
.rodata:00000000003B3DCC                 db 5Eh, 0FDh, 3Dh, 2Eh, 0D4h, 43h, 64h, 6Fh, 29h, 0A5h
.rodata:00000000003B3DD6                 db 4Ch, 1, 3Bh, 0, 40h, 0, 42h, 0, 0C9h, 0FDh, 0BFh, 9Eh
.rodata:00000000003B3DE2                 db 1Fh, 0Eh, 4Ch, 0C6h, 34h, 92h, 19h, 0E6h, 5Fh, 0FEh
.rodata:00000000003B3DEC                 db 57h, 0A2h, 40h, 14h, 0D1h, 0CDh, 10h, 4Dh, 87h, 41h
.rodata:00000000003B3DF6                 db 37h, 80h, 0CCh, 0C4h, 71h, 2Bh, 3Bh, 0BEh, 0D4h, 0F7h
.rodata:00000000003B3E00                 db 0C1h, 6Fh, 0B0h, 4Bh, 3Ch, 56h, 0B4h, 0FEh, 19h, 9Fh
.rodata:00000000003B3E0A                 db 0FDh, 0E2h, 0FAh, 0C7h, 38h, 0FBh, 0DEh, 9Eh, 8Bh, 14h
.rodata:00000000003B3E14                 db 19h, 0BBh, 0C8h, 0A3h, 1, 45h, 9Eh, 9Fh, 0E4h, 2Dh
.rodata:00000000003B3E1E                 db 0CEh, 37h, 0B3h, 0EAh, 99h, 68h, 0A7h, 74h, 8Dh, 0A4h
.rodata:00000000003B3E28                 db 15h, 61h, 0A4h, 97h, 67h, 0CBh, 51h, 0A1h, 6Ch, 67h
.rodata:00000000003B3E32                 db 0B0h, 5Eh, 85h, 0B0h, 0E8h, 0A9h, 49h, 0B4h, 46h, 0EBh
.rodata:00000000003B3E3C                 db 0C1h, 8Eh, 0Dh, 5Dh, 0C3h, 57h, 42h, 81h, 0B2h, 0B8h
.rodata:00000000003B3E46                 db 0BFh, 0FDh, 0DDh, 0F0h, 1, 36h, 0C7h, 7Dh, 4Ch, 0A1h
.rodata:00000000003B3E50                 db 0FAh, 78h, 0CDh, 0FCh, 0DBh, 0B5h, 7Fh, 0DDh, 3, 0BDh
.rodata:00000000003B3E5A                 db 0BFh, 0E5h, 0F6h, 52h, 7Dh, 75h, 0D7h, 0CBh, 67h, 13h
.rodata:00000000003B3E64                 db 88h, 9, 61h, 0A0h, 69h, 0F1h, 0Fh, 0BFh, 3Dh, 9Eh, 0AAh
.rodata:00000000003B3E6F                 db 0Ah, 0BEh, 0ACh, 9Fh, 0Dh, 7Fh, 0CEh, 8Fh, 0F1h, 95h
```

Рисунок 37. Вшитая полезная нагрузка

```
 import zstandard
import cbor2
import sys
import json

dctx = zstandard.ZstdDecompressor()

with open(sys.argv[1], 'rb') as ifh, open(sys.argv[1] + '.dec', 'wb') as ofh:
       dctx.copy_stream(ifh, ofh)

with open(sys.argv[1] + '.dec', 'rb') as fp, open(sys.argv[1] + '.des', 'w') as fw:
       obj = cbor2.load(fp)
       json.dump(obj, fw, indent = 4, sort_keys = True, default = repr)
```

Листинг 5. Скрипт на Python для декодирования полезной нагрузки

После десериализации будет получен JSON-объект, хранящий тело полезной нагрузки, а также индекс и название (при наличии).

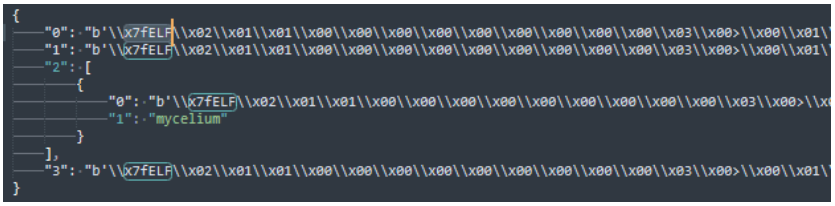

Рисунок 38. Разархивированная и десериализованная полезная нагрузка

2.5.5. Команда glue

Команда отвечает за модификацию легитимных утилит, установленных в системе, для обеспечения запуска вшитой полезной нагрузки.

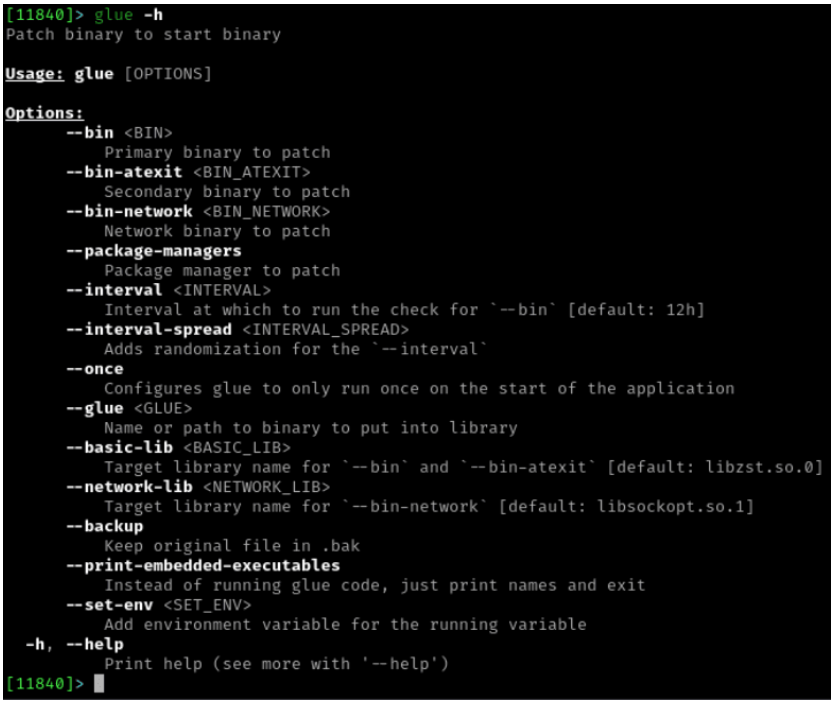

Рисунок 39. Параметры команды glue

Для использования команды должна быть указана как минимум одна из опций --bin, --bin-network или --package-managers. В противном случае будет выведено сообщение об ошибке.

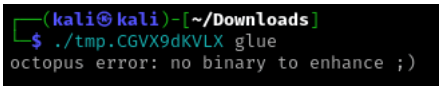

Рисунок 40. Запуск команды glue без параметров

Для каждого переданного команде glue файла будет выполнена проверка на наличие библиотеки libc.so.6 в зависимостях. Более того, Octopus должен иметь доступ к атрибуту security.selinux (если такой имеется у библиотеки).

В случае успешных проверок, в зависимости от того, передавался ли аргумент --glue, Octopus попытается либо считать файл (если в glue передавался путь до библиотеки, при этом важно, чтобы путь начинался с символа /), либо найти его по имени среди встроенных библиотек.

Если glue не удалось по какой-либо причине найти, Octopus выдаст сообщение об ошибке.

Далее Octopus сериализует конфигурацию, необходимую для работы руткитов, используя последовательно алгоритм CBOR и сжатие с помощью zstd. Сериализацию проходят параметры, переданные команде в качестве аргументов. Затем Octopus записывает необходимые руткиты в каталог, в котором находится libc.so.6. В каждом записываемом рутките дополнительно создается секция с типом SHT_NOTE, в которую помещается сериализованная ранее конфигурация.

Крейт, который непосредственно отвечает за патчинг, назван leech.

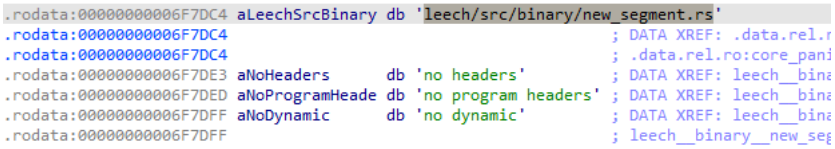

Рисунок 41. Название крейта, в котором реализована функция патчинга

Суть используемого метода патчинга состоит в следующем. В выбранный легитимный исполняемый файл внедряется дополнительная зависимость от вредоносного модуля, который является библиотекой .so.

Опишем механику внедрения зависимости. Leech пересобирает секции исходного бинарного файла, добавляя в секцию .dynamic (содержит сведения о динамически загружаемых модулях) ссылку на строку с именем внедряемого вредоносного модуля (дописывается в секцию .dynstr). Вначале leech парсит исходный бинарный файл, собирая информацию о его сегментах и секциях. В первую очередь это секция .interp (специальная секция, которая

содержит путь к нужному компоновщику), а также секции с типом SHT_NOTE. Секции SHT_NOTE, как правило, несут дополнительную информацию о файле, однако существует такая секция с именем .note.ABI-tag, которая, согласно спецификации, должна присутствовать в каждом ELF-файле.

```
while ( 1 )
{
  v17 = v10 < v83;
  v18 = v10 - v83;
  v19 = 0LL;
  if ( !v17 )
    v19 = v18;
  if ( v19 >= v82 )
    break;
  v10 = *leech::get_n_section(v84, v16, &core_panic_location___leech_src_binary_new_segment_rs_59_31, v12);
  v21 = *(leech::get_n_section(v84, v16 - 1, &core_panic_location___leech_src_binary_new_segment_rs_60_25, v20) + 8);
  v87 = v21;
  leech::collect_section(&v76, v21);
  n_section_ptr = leech::get_n_section_ptr(v65, v21, &core_panic_location___leech_src_binary_new_segment_rs_62_44);
  v23 = *(n_section_ptr + 32);
  if ( !leech::check_section(n_section_ptr, a2) )
  {
    v88[0] = &v87;
    // cannot move section #{} to free space for new program header
    v70 = &off_7DF7F0;
    v71 = 2LL;
    v74 = 0LL;
    v88[1] = core::fmt::num::imp::<impl core::fmt::Display for usize>::fmt;
    v72 = v88;
    v73 = 1LL;
    alloc::fmt::format::format_inner(v89, &v70);
    v35 = sub_ABC28(v89);
    goto LABEL_53;
  }
  v69 += v23;
  v14 -= 16LL;
  v15 += 16LL;
  ++v16;
}
v24 = leech::get_n_section(v84, v16 - 1, &core_panic_location___leech_src_binary_new_segment_rs_73_38, v12);
if ( *(leech::get_n_section_ptr(v65, *(v24 + 8), &core_panic_location___leech_src_binary_new_segment_rs_73_31) + 4) == 7 )
{
```

Рисунок 42. Сбор информации о бинарном файле

Далее leech ищет индекс секции, содержащей таблицу символов (данная секция называется .dynstr и имеет тип SHT_STRTAB), и индекс секции, в которой находится информация, необходимая для динамического связывания (секция имеет имя .dynamic и тип SHT_DYNAMIC).

Затем, в соответствии с собранной информацией, создается новый сегмент с типом PT_LOAD, куда помещаются собранные секции. А в секции .dynamic дополнительно создается поле с типом DT_NEEDED и указателем на строку с именем руткита, которое предварительно дописывается в секцию .dynstr.

Таким образом, при запуске пропатченного файла компоновщик автоматически подгрузит руткит (так как это необходимая зависимость). В результате модификаций создается временный файл с именем <original_parent_dir>/.~<original_name>.tmp, куда записывается пропатченный ELF с учетом перестроения.

В случае корректного запуска данный временный файл переименовывается в оригинальный (с помощью системного вызова rename), тем самым замещая его.

2.5.6. Команда netpatch

Команда netpatch внедряет полезную нагрузку в один из сервисов, функционирующих в системе.

```
[11840]> netpatch -h
Patch binaries to hijack connections and also to restore itself

Usage: netpatch [OPTIONS]

Options:
      --dry-run
          Do not actually patch, just detect
  -n, --filter <FILTER>
          A comma filter of what to patch. See wiki for more info
  -B, --backup
          Keep original file in .bak
  -R, --no-restart
          Do not restart services, just print what to do
      --connect-to <CONNECT_TO>
          Ask mycelium to connect to the specified host
      --ping-interval <PING_INTERVAL>
          Set ping interval for connect-to
      --success-reconnect-interval <SUCCESS_RECONNECT_INTERVAL>
          Set reconnect interval on success for connect-to
      --failure-reconnect-interval <FAILURE_RECONNECT_INTERVAL>
          Set reconnect interval on failure for connect-to
      --mosquito
          Do a smaller patch via mosquito library (does not support all the features)
  -h, --help
          Print help (see more with '--help')
[11840]>
```

Рисунок 43. Параметры команды netpatch

Внедрение происходит через модификацию исполняемого файла целевого сервиса. В качестве полезной нагрузки по умолчанию применяется бэкдор mycelium. Также есть возможность использовать в качестве полезной нагрузки

встроенный бэкдор mosquito (libsockopt.so.2). Для его применения необходимо указать опцию --mosquito при выполнении команды netpatch.

При использовании опции --mosquito значительно уменьшаются возможности Octopus: нельзя использовать опцию --connect-to и не поддерживаются спавнеры (например, сервис cron) и пакетные менеджеры.

В опцию --connect-to передается значение в формате host:port, к этому порту будет подключаться полезная нагрузка (в данном случае mycelium).

Слово «спавнер» в контексте Octopus означает программу-демон, пропатченную библиотекой libzst.so.0, которая с определенной периодичностью (согласно справке к команде glue, в интервале от --interval до --interval-spread) будет запускать нагрузку (в данном случае это mycelium).

Стоит отметить, что бэкдор mycelium работает на дистрибутивах, версия ядра которых не ниже 2.6.27.

```
.rodata:00000000003B297E aMinimumKernelV db 'Minimum kernel version that tokio(mycelium) supports is 2.6.27 bu'
.rodata:00000000003B297E                                        ; DATA XREF: .data.rel.ro:off_7D7E70↓o
.rodata:00000000003B29BF                 db 't this system is '
.rodata:00000000003B29D0 unk_3B29D0      db  2Eh ; .            ; DATA XREF: .data.rel.ro:00000000007D7E80↓o
.rodata:00000000003B29D1 aErrorCannotChe db 'Error cannot check kernel: '
.rodata:00000000003B29D1                                        ; DATA XREF: .data.rel.ro:off_7D7E90↓o
.rodata:00000000003B29EC aGettingUname   db 'getting uname'     ; DATA XREF: octopus__cmd__netpatch+B1↑o
.rodata:00000000003B29F9 aReleaseIsAStri db 'release is a string'
.rodata:00000000003B29F9                                        ; DATA XREF: octopus__cmd__netpatch+159↑o
.rodata:00000000003B2A0C aInvalidRelease db 'invalid release'   ; DATA XREF: octopus__cmd__netpatch+240↑o
```

Рисунок 44. Строка о минимальной версии ядра для запуска mycelium

При запуске команды netpatch будет выведена информация о доступных в системе сетевых интерфейсах, перебор которых осуществляется с помощью инструмента Netlink.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo ./tmp.CGVX9dKVLX netpatch
[*] Note: IP addresses: 127.0.0.1/8, ::1/128
```

Рисунок 45. Доступные сетевые интерфейсы

Выводится также информация об известных и неизвестных сервисах, а также о названиях пакетов, которые их поставляют.

```
[*] Note: "/usr/sbin/mariadbd" on 127.0.0.1:3306 is unknown
Warning: non-existing binary "/memfd:kernel"

Patchable files:
  1. /usr/sbin/cron (spawns mycelium)
     package: cron
     cron (systemd):
  2. /usr/sbin/nginx (network daemon)
     package: nginx
     nginx (systemd): 0.0.0.0:80, [::]:80
  3. /usr/sbin/sshd (network daemon)
     package: openssh-server
     ssh (systemd): 0.0.0.0:22, [::]:22
  4. /usr/bin/dpkg (package manager)
     package: dpkg
```


Рисунок 46. Запущенные сервисы

Под известными подразумеваются следующие сервисы:

- cron (crond),
- sshd,
- postgres,
- nginx,
- apache,
- lighttpd,
- mariadb.

Контроль над тем, что необходимо пропатчить, осуществляется либо с помощью опции —filter, либо с помощью непосредственного считывания строки во время запуска команды.

Если команда netpatch запущена с полезной нагрузкой по умолчанию (mycelium), то при указании опции —connect-to для данной нагрузки кодируется конфигурация и генерируется TOKEN (см. рис. 49).

Начальная конфигурация для mycelium представляет собой набор аргументов, с которыми он запускается.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./mycelium.bin -h
Usage: mycelium.bin [OPTIONS] [COMMAND]

Commands:
  debug  Debugging mode
  help   Print this message or the help of the given subcommand(s)

Options:
  -f, --foreground  Do not daemonize
  -h, --help        Print help (see more with '--help')

┌──(kali㉿kali)-[~/Downloads]
└─$ ./mycelium.bin -f help
Usage: mycelium.bin [OPTIONS] [COMMAND]

Commands:
  debug  Debugging mode
  help   Print this message or the help of the given subcommand(s)

Options:
  -f, --foreground
          Do not daemonize

          Also: 1. Does not check existence of the daemon 2. Allows running as non-root

  -h, --help
          Print help (see a summary with '-h')
┌──(kali㉿kali)-[~/Downloads]
└─$ ./mycelium.bin debug --help
Debugging mode

Listen on TCP socket instead of SCM_RIGHTS over unix socket

Usage: mycelium.bin debug [OPTIONS]

Options:
  -H, --ip <IP>
          Bind the IP address instead of unix socket

          This can be used to test connectivity without patching anything

          [default: 127.0.0.1]

  -P, --port <PORT>
          Ip to connect to

          [default: 7000]

  -v, --verbose


  -w, --wait
          Wait for handshake instead of relying on early data

      --connect-to <CONNECT_TO>
          Host port to connect to

  -h, --help
          Print help (see a summary with '-h')
```

Рисунок 47. Команды mycelium и их описание

Конфигурация сериализуется алгоритмом CBOR. Затем генерируется случайная гамма размером 5 байт и сериализованная конфигурация гаммируется. К зашифрованной конфигурации дописывается сгенерированная гамма, и все это кодируется алгоритмом Base64.

```
cbor::serialize(&v10, a2);
sub_144DB8(&v6, &v10, &aInvalidRelease[15], 13LL);
if ( v6 == 0x8000000000000000LL )
{
  *a1 = __PAIR128__(*(&v6 + 1), 0x8000000000000000LL);
}
else
{
  v2 = v7;
  v8 = v6;
  v9 = v7;
  ::generate_key(&v8, 0LL, 5LL);
  ::split_key_data(&v10, *(&v8 + 1), v9, v2, &core_panic_location___infect_src_netpatch_rs_252_30);
  v12 += v11;
  v13 = v11;
  v14 = v12;
  v16 = 0LL;
  v15 = 0LL;
  *(&v10 + 1) += v10;
  while ( 1 )
  {
    v3 = ::get_next_key_data_pair(&v10);
    if ( !v3 )
      break;
    *v3 ^= *v4;
  }
  v10 = v8;
  v11 = v9;
  ::base64(a1, &aInvalidRelease[28], *(&v8 + 1));
  drop_17(&v10);
}
return a1;
```

Рисунок 48. Шифрование конфигурации для mycelium

Закодированная конфигурация mycelium будет дописана в секцию с конфигурацией руткита, который записывается командой glue.

```
0 :  43200
1 :  14400
3 :  []
4 :  []
5 :  ['/usr/sbin/sshd']
6 :  libzst.so.0
7 :  libsockopt.so.1
8 :  [['TOKEN', 'MJOeYu2gowtMtL-hElOFicGRkTxihA']]
9 :  [1, {}]
10 :  ['openssh-server']
```

Рисунок 49. Передаваемая конфигурация для mycelium

После того как все переданные сервисы были пропатчены, Octopus заменяет MD5-хеш-суммы пропатченных сервисов. Например, для сервиса openssh в системе с dpkg изменится сумма в файле /var/lib/dpkg/info/openssh-server.md5sum.

Для запуска mycelium Octopus вызывает fork и в процессе-потомке использует вызов execveat.

Стоит заметить, что вшитая полезная нагрузка на диск не записывается. Она хранятся в скрытом файле с именем kernel, который создается с помощью системного вызова memfd_create.

Матрица MITRE ATT&CK

ID	ТЕХНИКА	ОПИСАНИЕ
PERSISTENCE		
T1554	Compromise Host Software Binary	(Ex)Cobalt подменяет системный бинарный файл cron на вредоносный загрузчик руткита PUMAKIT, сохраняя внешний вид и функции штатного демона. (Ex)Cobalt модифицирует легитимные исполняемые файлы на скомпрометированных системах, внедряя код запуска руткита с помощью утилиты Octopus
T1547.006	Boot or Logon Autostart Execution: Kernel Modules and Extensions	(Ex)Cobalt устанавливает модуль ядра (audit) после проверки необходимых условий — отключенного Secure Boot и совместимости символов ядра, используя wpn
T1574.006	Hijack Execution Flow: Dynamic Linker Hijacking	(Ex)Cobalt использует загруженный модуль ядра (audit) для загрузки бэкдора, устанавливая в LD_PRELOAD значение /lib64/libs.so
T1098.004	Account Manipulation: SSH Authorized Keys	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата системных вызовов open и read, чтобы при обращении легитимных процессов (например, sshd) к файлу ~/.ssh/authorized_keys добавлять в возвращаемый буфер собственный SSH-ключ. Сам файл на диске остается неизменным, но с точки зрения sshd у учетной записи появляется еще один авторизованный ключ, что создает скрытую и устойчивую точку входа по SSH
EXECUTION		
T1059.004	Command and Scripting Interpreter: Unix Shell	(Ex)Cobalt использует бэкдор (libs.so) для запуска и выполнения произвольных команд через /bin/sh
PRIVILEGE ESCALATION		
T1068	Exploitation for Privilege Escalation	(Ex)Cobalt применяет эксплойты для уязвимостей CVE-2021-3156 (heap overflow в утилите sudo), CVE-2022-2588 (double free в функции route4_change), CVE-2023-4911 (buffer overflow в ld.so) для повышения привилегий

ID	ТЕХНИКА	ОПИСАНИЕ
T1548.001	Abuse Elevation Control Mechanism: Setuid and Setgid	(Ex)Cobalt использует загруженный модуль ядра (audit) для модификации структуры cred вызывающего rmdir zarya_0_0 процесса, устанавливая UID/GID/EUID/EGID в 0 и, тем самым, повышая его привилегии до root
DEFENSE EVASION		
T1014	Rootkit	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата и модификации системных вызовов и ключевых функций ядра. Это позволяет скрывать собственные файлы, процессы, сетевые соединения и установленный модуль, маскируя присутствие в системе
T1036.005	Masquerading: Match Legitimate Name or Location	(Ex)Cobalt подменяет легитимный cron (/usr/sbin/cron) на дроппер PUMAKIT, сохраняя ожидаемое поведение сервиса
T1027.009	Obfuscated Files or Information: Embedded Payloads	(Ex)Cobalt использует утилиту Octopus, которая хранит полезную нагрузку в секции данных своего исполняемого файла
T1027.011	Obfuscated Files or Information: Fileless Storage	(Ex)Cobalt использует анонимные файловые дескрипторы (memfd) для запуска ключевых компонентов ВПО PUMAKIT (например, /memfd:tgt и /memfd:wpn). Полезная нагрузка, устанавливаемая инструментом Octopus, не записывается на диск и хранится в скрытом файле с именем kernel, который создается с помощью системного вызова memfd_create
T1027.015	Obfuscated Files or Information: Compression	(Ex)Cobalt использует утилиту Octopus, которая сжимает полезную нагрузку алгоритмом zstd
T1036	Masquerading	(Ex)Cobalt использует утилиту Octopus, которая сохраняет экземпляры полезной нагрузки в файлы, имена которых имитируют названия легитимных компонентов: libzst.so.0, libsockopt.so.1, libsockopt.so.2
T1218	System Binary Proxy Execution	(Ex)Cobalt использует утилиту Octopus, которая имеет функцию поиска установленных в системе утилит GTFOBins
T1564.001	Hide Artifacts: Hidden Files and Directories	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата вызова getdents и последующей модификации его результатов, скрывая артефакты PUMAKIT из результатов чтения каталогов, делая их невидимыми для ls и аналогичных утилит
T1564	Hide Artifacts	(Ex)Cobalt использует загруженный модуль ядра (audit) для фильтрации вывода информации о процессах, модулях и сетевых соединениях, скрывая их артефакты от стандартных средств мониторинга
T1562.004	Impair Defenses: Disable or Modify System Firewall	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата nf_hook_slow, что позволяет PUMAKIT обходить фильтрацию netfilter/iptables и сохранять связь с C2-сервером, нивелируя работу встроенного файрвола для собственных соединений
T1562.001	Impair Defenses: Disable or Modify Tools	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата функций SELinux, препятствуя применению политик к важным для руткита процессам и соединениям, фактически отключая механизмы контроля доступа и аудита

ID	ТЕХНИКА	ОПИСАНИЕ
CREDENTIAL ACCESS		
T1056.004	Input Capture: Credential API Hooking	(Ex)Cobalt использует загруженный модуль ядра (audit) для перехвата системных вызовов read и write, анализируя передаваемые буферы на наличие искомых подстрок
DISCOVERY		
T1082	System Information Discovery	(Ex)Cobalt используют бэкдор (libs.so) для сбора сведений об операционной системе, ядре, архитектуре и окружении, передавая их оператору для оценки узла
LATERAL MOVEMENT		
T1021.004	Remote Services: SSH	(Ex)Cobalt использует загруженный модуль ядра (audit) для скрытного добавления SSH-ключа, что позволяет злоумышленникам входить по SSH под видом легитимного пользователя и использовать узел в качестве точки дальнейшего продвижения в инфраструктуре жертвы
COLLECTION		
T1074.001	Data Staged: Local Data Staging	(Ex)Cobalt агрегирует украденные пароли, ключи и другие данные в локальном файле (zov_logs.txt), служащим буфером перед эксфильтрацией
COMMAND AND CONTROL		
T1071.001	Application Layer Protocol: Web Protocols	(Ex)Cobalt использует бэкдор (libs.so) для установки TLS-соединения с C2-сервером и последующей передачи бинарных сообщений в формате BSON
T1571	Non-Standard Port	(Ex)Cobalt использует для C2-серверов нетипичные порты (например, 8443)
T1573	Encrypted Channel	(Ex)Cobalt использует TLS для взаимодействия с C2-серверами
T1105	Ingress Tool Transfer	(Ex)Cobalt использует бэкдор (libs.so) для получения с C2-сервера своей обновленной версии (zov_latest)
T1090.001	Proxy: Internal Proxy	(Ex)Cobalt использует бэкдор (libs.so) для запуска на узле SOCKS-прокси
EXFILTRATION		
T1041	Exfiltration Over C2 Channel	(Ex)Cobalt использует бэкдор (libs.so) для передачи содержимого файла zov_logs.txt (перехваченные пароли и ключи), файл отправляется на C2-сервер по тому же зашифрованному каналу, что и команды управления
T1030	Data Transfer Size Limits	(Ex)Cobalt ограничивает размер файла zov_logs.txt до 8 МБ, при превышении этого порога файл автоматически очищается

Индикаторы компрометации

PUMA		
Имя	SHA-256	SHA-1
cron	0eafb9dbaf9f72e61ee65926a61b3b07a3a5d1ab1a47b0bb6e3e8bc3c9f9f9f2	7fbe0a0dbeb8d55ad7764242fd4f8d6a
	0f37d45c70d5fae0c76b9eeed454b4a0ef58609ce99534d1ca9365fac5155fc9	c1633991f910735949380c416b2ffd9e
	5cef48c73704040bea45f85601cbc0fe40b5dea84721344b1cfd67c62650e996	528585a5c921ddf7027a4d5d3096d99

PUMA		
	8ce769d07989f59ece5f137afce84e4075fcc1010426bf1a77378d7e2a628995	aae68641de0d64f3d2ee71129819168
	8d3a411ea9225d8bc0503235e0832025e7a3260b9bcdb85119ccb508c4547107	b75165c8713f4f28df8f36694f988467
	9f5a79d9c5100de0af25bec6f640264e4db367d1a6cd1804e1e2a1772637c30a	2d9607bdc928cdd057c7c03732235e7
	37a5bd6361e48586749594f5fb853c4b79a0931f979f03bdb31dc91c526924ec	cb6c56e43c3db089d140bf54047bd1a
	49d4cd2bf3b65db2ecbaaa46944b2b76786de51724fc73c44d0d8e485fa1f3b5	3ab09a089574545e2824cd5549715af
	9959a16f1fd19191de2d40cad5dbf4e63c62eb55487068de0a6f6b4696bccd72	c095b4e41bcab55fc0cbb29889fbd43c
	87290ce0d3bd3a680d092fe038a9e80b37b3358502d511236ec5df50e4bfe6ee	acb36b770d5d5322abd2a2f5712e34e
	df9b0d259a869c2bf71cbcdd90643750ffad067acdf8480e2981531f47ebf930	c6468621aa3021b46190777b903c634
	30b26707d5fb407ef39ebee37ded7edeea2890fb5ec1ebfa09a3b3edfc80db1f	810f4b422b9c0a6718e9461de3a2edd
	13c9c6cc9b7a180949e29baa38efde81e06e7e0a29be3b4ed98eb10af4ab60ef	30b3241f662d2c17945c7ce19bb4666
wpn	7b1da4c6c9b44444e9a6c4d58f7676af52ecd504fe2657fce9ce1adc2b711055	d3910322b006a398011dc22b831d709
	be44c606c2114534222f1be794a37a7d113d849376328149d60d6b67c1318946	329254021a8679a603bf0f29f97195b0
	f71873f37fbd5a5870a51db75d8b99ff83ac2d03f0e5682e943d8cb9c84b6129	8ba2aeb62c043b2f35e56e1e1633031b
	650bd3fec6160759878f58962d2cffe9cf0074b59b01e569621335470da1e005	798b5cc955c7d39cb9d40d011b5f598
	28dbae86db73868758301c140f2b2c7bc19ce3862e0cd21966bfe9d3e7476e4a	d05c83c339d8d1224e5992eae04d8eb
	809e41c3a700ccaaaf2602b62e162a4ae59315ef86ee37daca8c7064d01bf086	be0da40d7dc3ea8bcd2a72821d223f2
	65e35f68c12938f133fb120565f3c8221c53d3b841dbddb98e0d3c6054f2c7be	fe25ed6b992af2d37fb3876ef48d4431
	fb1c815365622a37328570c4b3703654fe7e5c9e9eae335542bac6090e2f9a2e	b342fc7e1655fabd5a9232f8fe85f677
	1444b49c6e6183f856a4f18ea6a657ae3033b47a45c1152dcf60dc361a7826d7	b11fdff95db2e06f64151600772b7e1d
	8fc01417832dd08559609ef0667149a24897380113294e10b44694071afd22f2	06ab28718573365dfe47238e89372c8
	2d7328d0623b36c31e22b1b12ece8094aa232ccdb7ea92ec860ff6b7030fbe93	4eb2e95b7e1079d53610acc7ea910af4
	bcc0cda02a11acbd0923db43099f854038caaf03ddbf59e1d53452641119d42a	b8f5d35e407090e800b1686a0a3d550
	17d9359625b47fcf65ccdb59bd0dba6c44f8e140d8eedc37a8923baa8aefd481	9ec003870a9cfe15ea7530197c95241c
audit	d291f0bfa4a82cad1c95d11ebabb1539b9d9d5837534a8bd5b659462dd2eeabd	6bce7d2a1df368edd56f0f6a126299f8
	9a0d5e90444d94778775180b6373a6d4f0ca293764f51b4371f4ab5452f202e3	ec6e3309e371977ba840b3e1240653a
	873c4fbf47b8036238b52fd02bf7846d86df4c87f6dbf302db9a959371d464ff	2ac766bab601a2c172db98c7c9e7805
	58cb73a09b0cb497119bf6450f20d8343f01ea8c8ace72cc71e9414920f77342	0d49e3c8ec9f02a262e29ce90d66b58c
	13299152ead4b1592843b9eddfbfd7b320a5fdec09c568a11ec7f9834cd7a610	26824e4f248fbd9e162c522962972a9a
	fb0d79602dfac97f5e36f5f0287d167271d04294dccfdccfa15a88ff6dba11b7	cbe1e249142b25fe3bc11ad7f57d4777
	e49eb564c061b132c411859e79e8026f46059f4be759d812e2043956400d32ad	88f5e5832ec4a2641792e3448653ed9
	ee33385b20be0d457f4acccc5d222eb11adb7683c96b4dfd30d2772361f2346b	113a9d00dbe3d5db956463d4675b939
	ef8305ea0a53676a04da70abb9d18ea36acba67404e5bfdeddc201d95ec1c22a	e0c5d82debbea11bc46357ccbbd0371
	1cc3edb1758fcadd97cd835f6b516db3aa8c3cbc563776080ab8ebf378536c71	b1c5f389de9de271681ac277ef7fe4c8
	ec2fc991758ae4b28c1691a00470c3723171db69f901627c8e07b7f8dce221ca	d78b2ba08f5ed03bcd48019efab92052
	f5635d9afd0116846cf447e218c561cc111c50e6a994b5c719556ce28c4e64d6	e9ed4d162edce0bc6be286cbfadb6216
	fb1811eaa27caeb84077900e5cfe7000bc1130a4dc51d3a2f93340080e16776f	0179608320b26e25f94ba41b1e9366e
/lib64/libs.so (zov_latest)	e7aa94e76acf181b4fd6fbfdcf599ce677c5f95675b50e68fe2d43ea72202bf7	d752651d0bcfc2fc7768c60409dbd858
	d6763d0d0c2b12ea9124ccaea3e6497bfc2cb4a3a9dfe6239a734afeaf372013	a071b503940a34c8779cf2d7fe466f12
	77b9a372ee97c0c832a618def71940e65a3c3d1eca1f8cb34ff395fd39cd23a2	56f7874b9f89c0299cc3656d4f5d48f4

PUMA		
	0002afc2839af82725499ed50f75a193b01a32111253ba3d8647189d8f0f2c28	8a9b9d23fc25fe963e64d692fed09da4
	cad9677ad4198b1710d390747e98b6b5e29ceab18f16e8ab12be5b6b8aab4021	f971af8387ac3e11f5feb2728978e37e
	d597eee02fc30b3bc1c523e8afad63f75912c4c2a47d3e981c5950a00bd76a6d	f7db8a88992966917179cc22d833bf8
	3ee7ae78964f35ec56a1d5d6026482e40e6ca9a59f9f474fff475e32231e2ff4	c1ec4ed9c5c6244fc0a4b0856bb5190
	e7e52e15636cb32ac227f4833adf75a0bacca3d8b9081f3e3bdcb8243a8b68aa	07ce236a5b5edfa0b071cd9338f28edc
	d984d1de0c18d0290450b990b572a7e8454b095295810bdea6ced9151973a350	9fdc6f48ea3100cebe038d11c4c6669d
	c1949fe4bdf693ae4e62f52bc728bf3b455b47862298fdfcd61fefa7e304d7cf	624ba4531c4099eceae27c284cced7f5
Octopus		
tmp.CGVX9dKVLX	8b43430136fcc9ef43f201febddbdd608cfebd25c611c13a146a66883b0f201e	05245a5aff54c598adef5e67d8dccd8b
libsockopt.so.1	87b34520d791933733b8f9f27088a68d9b4e67b2db869be50de2a6e01a92077f	fc27af1478a427fdbeae8cc2b7cc7a40
	82581c8043ac12ef3e5c35572759fb494775aff210b060ebd7492d3134b25024	3f5d0bc0c7b8acacc614763179f810f2
libsockopt.so.2	46a4f04b3a88d2ecfc98c3488c5efe485eb0e48d4b1701123319dec2ceea96fc	8a8797238494808840a724ece404551
libzst.so.0	a9e316ba471b1bf606ee3c77ae8bf4b6552e139928d6071b7dca3f847c571c92	8422dad0a89a3665d502eb2203f0509
-	36d9db678b6895cd08e8673438b3de44a2cd589329c68cbf214067a38cd47d0e	1bc50e17273ff2414d099403ea39590

Сетевые индикаторы	
PUMAKIT	rhel.opsecurity1.art
	sec.opsecurity1.art
	89.23.113.204
	qxpngendvvp1.wris.monster
	cnhgenfhfp2.wris.monster
	ccdertsfrp1.wris.monster
	qdprtsorp2.wris.monster
	cckitsfrp1.n3x1lo.pro
	qdkitsorp2.n3x1lo.pro
	cddcvesfhfp1.wris.monster
	deefveskiip2.wris.monster
	cumfpo90sing.agddns.net
	procdia42ecte.agddns.net
	viedeu98.agddns.net
	laipros50.agddns.net
	fira24sonstablee.agddns.net
	chronback49in.duckdns.org

Вердикты продуктов Positive Technologies

PT Sandbox
Create.Process.Grep.SecureBoot
Trojan-Dropper.Linux.PumaKit.a
Create.Process.Memory.ReflectiveCodeLoading
Read.Process.Pkexec.DetectPwnKit

PT Sandbox
Create.Process.ExploitProbe.TcacheCheck
Read.Process.Sudo.PrivilegeCheck
Write.File.Binary.MyceliumPatch
apt_linux_UA_Excobalt__Backdoor__Octopus
apt_linux_UA_Excobalt__Rootkit__TransMarker
apt_linux_UA_Excobalt__Rootkit__Mosquito
apt_linux_UA_Excobalt__Rootkit__Spawner
apt_linux_UA_Excobalt__Backdoor__Mycelium

MaxPatrol SIEM
Suspicious_Read_System_Distro_Reconnaissance
Unix_Recon_Tools_And_Commands
Suspicious_Create_Process_DPKG_CheckInstalledSoftware
Unix_System_Information_Discovery
Unix_Software_Discovery
Suspicious_Create_File_Attribute_Hidden
Unix_Suspicious_Hidden_Files
Unix_Lib_Modify
Unix_Bin_Modify
Suspicious_Read_File_Shell_History
Suspicious_Read_File_Passwd_CredentialsEnumeration
Malware_Exploit_Elf_CVE_2021_4034_a
Subrule_CVE_2023_4911_GLIBC_Buffer_Overflow
