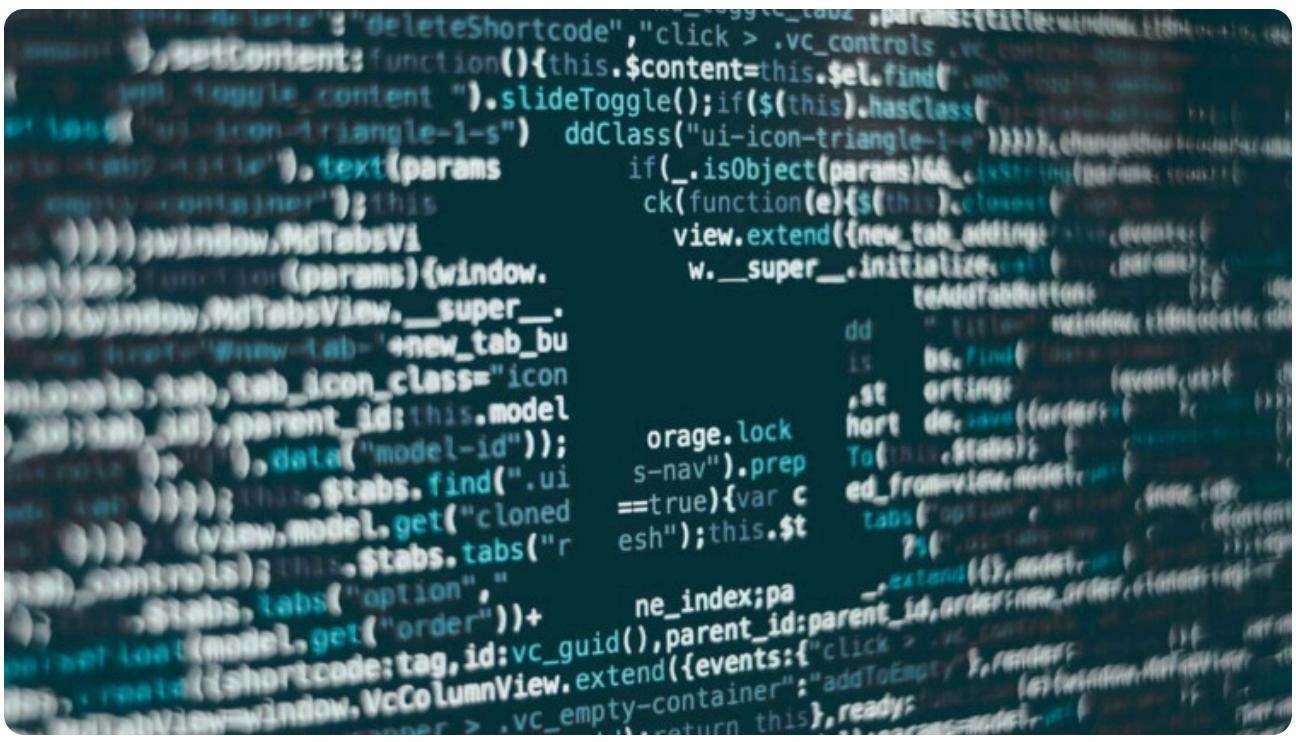
ESET RESEARCH

Lazarus luring employees with trojanized coding challenges: The case of a Spanish aerospace company

While analyzing a Lazarus attack luring employees of an aerospace company, ESET researchers discovered a publicly undocumented backdoor

Peter Kálnai

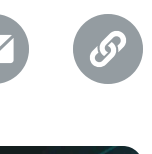
29 Sep 2023 • 28 min. read



Similar Articles

ESET RESEARCH
Lazarus supply-chain attack in South Korea

ESET RESEARCH
(Are you) afreight of the dark? Watch out for Vyveva, new Lazarus backdoor



ESET RESEARCH
Amazon-themed campaigns of Lazarus in the Netherlands and Belgium

ESET RESEARCH
Linux malware strengthens links between Lazarus and the 3CX supply-chain attack

ESET RESEARCH
WinorDLL64: A backdoor from the vast Lazarus arsenal?

Share Article

ESET researchers have uncovered a Lazarus attack against an aerospace company in Spain, where the group deployed several tools, most notably a publicly undocumented backdoor we named LightlessCan. Lazarus operators obtained initial access to the company's network last year after a successful spearphishing campaign, masquerading as a recruiter for Meta – the company behind Facebook, Instagram, and WhatsApp.

The fake recruiter contacted the victim via LinkedIn Messaging, a feature within the LinkedIn professional social networking platform, and sent two coding challenges required as part of a hiring process, which the victim downloaded and executed on a company device. The first challenge is a very basic project that displays the text "Hello, World!", the second one prints a Fibonacci sequence – a series of numbers in which each number is the sum of the two preceding ones. ESET Research was able to reconstruct the initial access steps and analyze the toolset used by Lazarus thanks to cooperation with the affected aerospace company.

In this blogpost, we describe the method of infiltration and the tools deployed during this Lazarus attack. We will also present some of our findings about this attack at the Virus Bulletin conference on October 4, 2023.

Key points of the blogpost:

- *Employees of the targeted company were contacted by a fake recruiter via LinkedIn and tricked into opening a malicious executable presenting itself as a coding challenge or quiz.*
- *We identified four different execution chains, delivering three types of payloads via DLL side-loading .*
- *The most notable payload is the LightlessCan backdoor, implementing techniques to hinder detection by real-time security monitoring software and analysis by cybersecurity professionals; this presents a major shift in comparison with its predecessor BlindingCan, a flagship HTTP(S) Lazarus RAT.*
- *We attribute this activity with a high level of confidence to Lazarus, particularly to its campaigns related to Operation DreamJob.*
- *The final goal of the attack was cyberespionage.*

the most notable is a
ojan (RAT) that we
ement compared to its
alities of a wide range
within the RAT itself
ances stealthiness,
re challenging.

oyment of execution
ypted on the intended
e protocols and
onfidentiality of the
payload during its deployment and execution, effectively preventing unauthorized

decryption on unintended machines, such as those of security researchers. We describe the implementation of this mechanism in the Execution chain 3: LightlessCan (complex version) section.

Attribution to the Lazarus group

The Lazarus group (also known as HIDDEN COBRA) is a cyberespionage group linked to North Korea that has been active since at least 2009. It is responsible for high-profile incidents such as both the Sony Pictures Entertainment hack and tens-of-millions-of-dollar cyberheists in 2016, the WannaCryptor (aka WannaCry) outbreak in 2017, the 3CX and X_TRADER supply-chain attacks, and a long history of disruptive attacks against South Korean public and critical infrastructure since at least 2011. The diversity, number, and eccentricity in implementation of Lazarus campaigns define this group, as well as that it performs all three pillars of cybercriminal activities: cyberespionage, cybersabotage, and pursuit of financial gain.

Aerospace companies are not an unusual target for North Korea-aligned advanced persistent threat (APT) groups. The country has conducted multiple nuclear tests and launched intercontinental ballistic missiles, which violate United Nations (UN) Security Council resolutions. The UN monitors North Korea's nuclear activities to prevent further development and proliferation of nuclear weapons or weapons of mass destruction, and publishes biannual reports tracking such activities. According to these reports, North Korea-aligned APT groups attack aerospace companies in attempts to access sensitive technology and aerospace know-how, as intercontinental ballistic missiles spend their midcourse phase in the space outside of Earth's atmosphere. These reports also claim that money gained from cyberattacks accounts for a portion of North Korea's missile development costs.

We attribute the attack in Spain to the Lazarus group, specifically to Operation DreamJob, with a high level of confidence. The name for Operation DreamJob was coined in a blogpost by ClearSky from August 2020, describing a Lazarus campaign targeting defense and aerospace companies, with the objective of cyberespionage. Since then, we have loosely used the term to denote various Lazarus operations leveraging job-offering lures but not deploying tools clearly similar to those involved in its other activities, such as Operation In(ter)ception. For example, the campaign involving tools signed with 2 TOY GUYS certificates (see ESET Threat Report T1 2021, page 11), and the case of Amazon-themed lures in the Netherlands and Belgium published in September 2022.

w a relationship
mpaign:

then convincing the
eed in a hiring process.
reamJob.

entified in the Dutch case
an backdoor linked with

s of this Lazarus
campaign – AES-128 and RC6 with a 256-bit key – that were also used in the Amazon-

campaign – AES-128 and RC6 with a 256-bit key – that were also used in the Amazon-themed campaign.

2. Infrastructure:

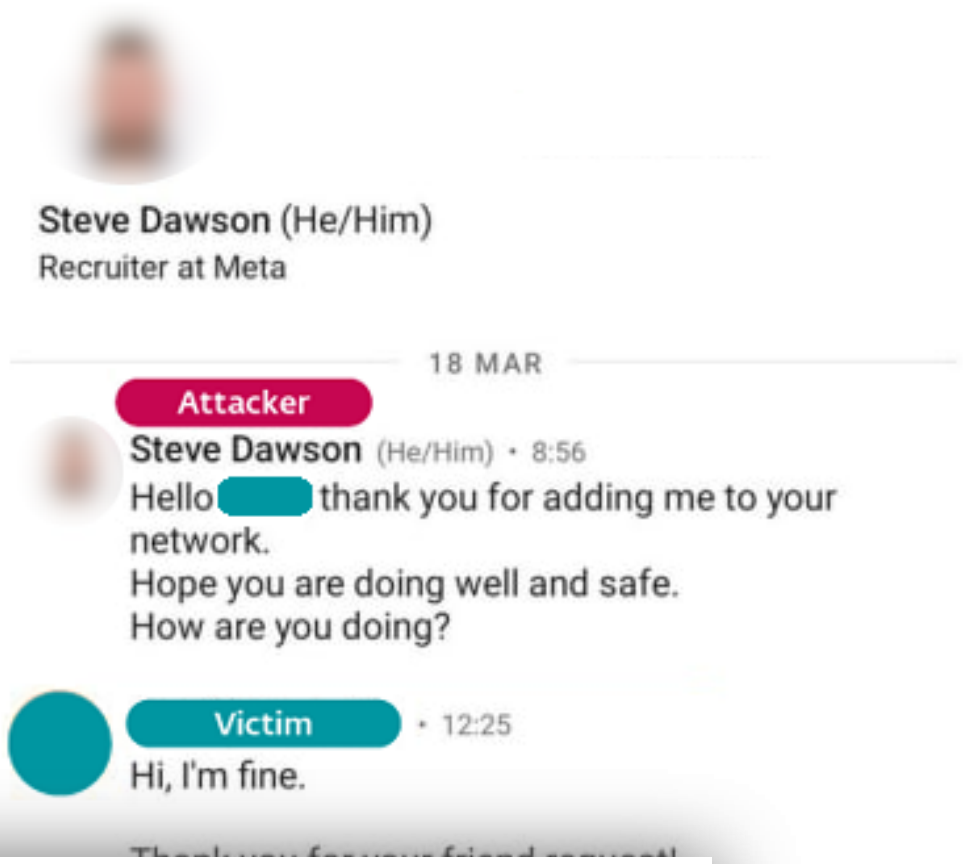
- For the first-level C&C servers (listed in the Network section at the end of this blogpost), the attackers do not set up their own servers, but compromise existing ones, usually those having poor security and that host sites with neglected maintenance. This is a typical, yet weak-confidence behavior, of Lazarus.

3. *Cui bono*:

- Pilfering the know-how of an aerospace company is aligned with long-term goals manifested by Lazarus.

Initial access

The group targeted multiple company employees via LinkedIn Messaging. Masquerading as a Meta recruiter, the attacker used a job offer lure to attract the target's attention and trust; a screenshot of this conversation, which we obtained during our cooperation with the Spanish aerospace company, is depicted in Figure 1.



recruiter from

e usually convinced to the attackers employ ute an attacker-provided (and trojanized) PDF viewer to see the full content of a job offer.

Alternately, the target is encouraged to connect with a trojanized SSL/VPN client, being provided with an IP address and login details. Both scenarios are described in a Microsoft blogpost published in September 2022. The narrative in this case was the scammer's request to prove the victim's proficiency in the C++ programming language.

Two malicious executables, `Quiz1.exe` and `Quiz2.exe`, were provided for that purpose and delivered via the `Quiz1.iso` and `Quiz2.iso` images hosted on a third-party cloud storage platform. Both executables are very simple command line applications asking for input.

The first one is a Hello World project, which is a very basic program, often consisting of just a single line of code, that displays the text "Hello, World!" when executed. The second prints a Fibonacci sequence up to the largest element smaller than the number entered as input. A Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones, typically starting with 0 and 1; however, in this malicious challenge, the sequence starts with 1 and 2. Figure 2 displays example output from the Fibonacci sequence challenge. After the output is printed, both executables trigger the malicious action of installing additional payloads from the ISO images onto the target's system. The task for a targeted developer is to understand the logic of the program and rewrite it in the C++ programming language.

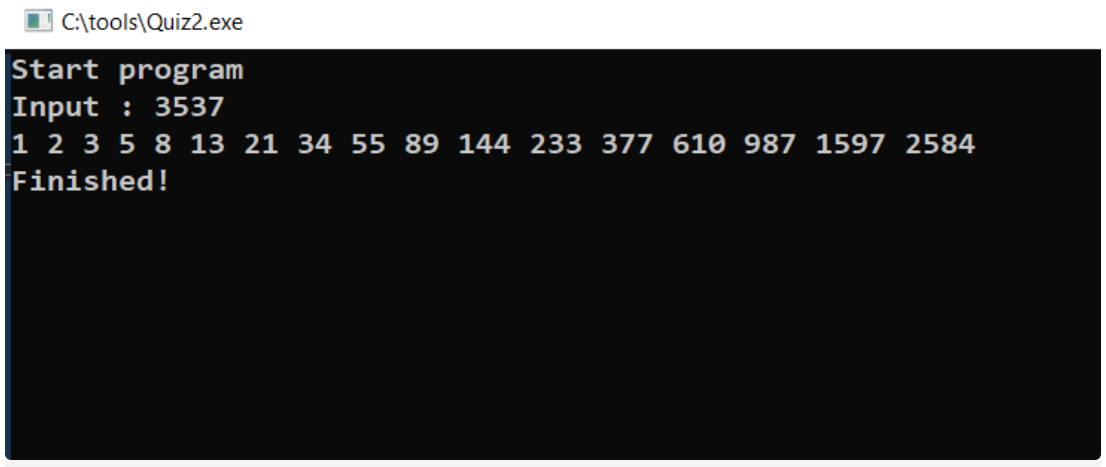
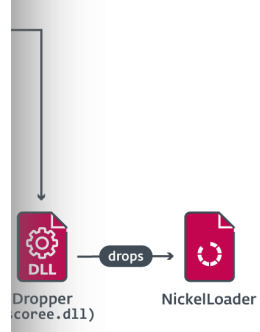


Figure 2. The output of the decoy program Quiz2.exe

The chain of events that led to the initial compromise is sketched in Figure 3. The first payload delivered to the target's system is an HTTP(S) downloader that we have named NickelLoader. The tool allows the attackers to deploy any desired program into the memory of the victim's computer.



access

Once NickelLoader is running on the target's system, the attackers use it to deliver two types of RATs. One of these RATs is already known to be part of the Lazarus toolkit, specifically a variant of the BlindingCan backdoor with limited functionality but identical command processing logic. To distinguish it, we put the prefix mini- in front of the variant's name. Additionally, the attackers introduced a RAT not previously undocumented publicly, which we have named LightlessCan.

The RATs are deployed as the final step of chains of stages with varying levels of complexity and are preceded by helper executables, like droppers and loaders. We denote an executable as a dropper if it contains an embedded payload, even if it's not dropped onto the file system but instead loaded directly into memory and executed. Malware that doesn't have an encrypted embedded data array, but that loads a payload from the file system, we denote as a loader.

Besides the initial quiz-related lures, Table 1 summarizes the executable files (EXEs) and dynamic link libraries (DLLs) delivered to the victim's system. All the malware samples in the third column are trojanized open-source applications (see the fourth column for the underlying project), with a legitimate executable side-loading a malicious DLL. For example, the malicious `mscoree.dll` is a trojanized version of the legitimate NppyPluginDll; the DLL contains an embedded NickelLoader and is loaded by a legitimate `PresentationHost.exe`, both located in the `C:\ProgramShared` directory.

Table 1. Summary of binaries involved in the attack

Location directory	Legitimate parent process	Malicious side-loaded DLL	Trojani project (payloa
`C:\ProgramShared\`	`PresentationHost.exe`	`mscoree.dll`	NppyPlug (NickelLo
`C:\ProgramData\Adobe\`	`colorcpl.exe`	`colorui.dll`	LibreSSL (miniBlinc
		`apistub.dll`	Lua plugi Notepad- 1.4.0.0 (Lightless
		`ID.dll`	MZC8051 Notepad- (Lightless

The most interesting payload used in this campaign is LightlessCan, a successor of

the group's flagship HTTP(S) Lazarus RAT named BlindingCan. LightlessCan is a new complex RAT that has support for up to 68 distinct commands, indexed in a custom function table, but in the current version, 1.0, only 43 of those commands are implemented with some functionality. The remaining commands are present but have a formal implementation in the form of placeholders, lacking actual functionality. The project behind the RAT is definitely based on the BlindingCan source code, as the order of the shared commands is preserved significantly, even though there may be differences in their indexing.

The most significant update is mimicked functionality of many native Windows commands like `ping`, `ipconfig`, `systeminfo`, `sc`, `net`, etc. The hardcoded string "The operation completed successfully.", the standard system message for the ERROR_SUCCESS result, brought us to that idea. Table 2 contains a list of those commands that are implemented in LightlessCan. In previously reported Lazarus attacks, as documented in blogposts by Positive Technologies in April 2021 and HvS Consulting in December 2020, these native commands are often executed in many instances after the attackers have gotten a foothold in the target's system. However, in this case, these commands are executed discreetly within the RAT itself, rather than being executed visibly in the system console. This approach offers a significant advantage in terms of stealthiness, both in evading real-time monitoring solutions like EDRs, and postmortem digital forensic tools. The internal version number (1.0) indicates that this represents a new development effort by the attackers.

As the core utilities of Windows are proprietary and not open-source, the developers of LightlessCan faced a choice: either to reverse engineer the closed-source system binaries or to get inspired by the code available via the Wine project, where many programs are rewritten in order to mimic their execution on other platforms like Linux, macOS, or ChromeOS. We are inclined to believe the developers chose the first option, as the corresponding Wine programs they mimicked in LightlessCan were implemented a little bit differently or not at all (e.g., `netsh`).

Interestingly, in one of the cases we analyzed, the LightlessCan payload is stored in an encrypted file on the compromised machine, which can only be decrypted using an environment-dependent key. More details about this can be found in the *Execution chain 3: LightlessCan (complex version)* section. This is to ensure that the payload can only be decrypted on the computer of the intended victim and not, for example, on a device of a security researcher.

for Windows prompt

mand prompt; see Figure

Figure 5.

n the Windows prompt;

36	Mimic the `netstat` command from the Windows prompt.
37	Mimic the `ping -6` command from the Windows prompt.
38	Mimic the `reg` command from the Windows prompt; see Figure 7.
39	Mimic the `sc` command from the Windows prompt; see Figure 8.
40	Mimic the `ping` command from the Windows prompt.
41	Mimic the `tasklist` command from the Windows prompt.
42	Mimic the `wmic process call create` command from the Windows prompt; see Figure 9.
43	Mimic the `nslookup` command from the Windows Server prompt.
44	Mimic the `schstasks` command from the Windows prompt; see Figure 10.
45	Mimic the `systeminfo` command from the Windows prompt.
46	Mimic the `arp` command from the Windows prompt.
47	Mimic the `mkdir` command from the Windows prompt.

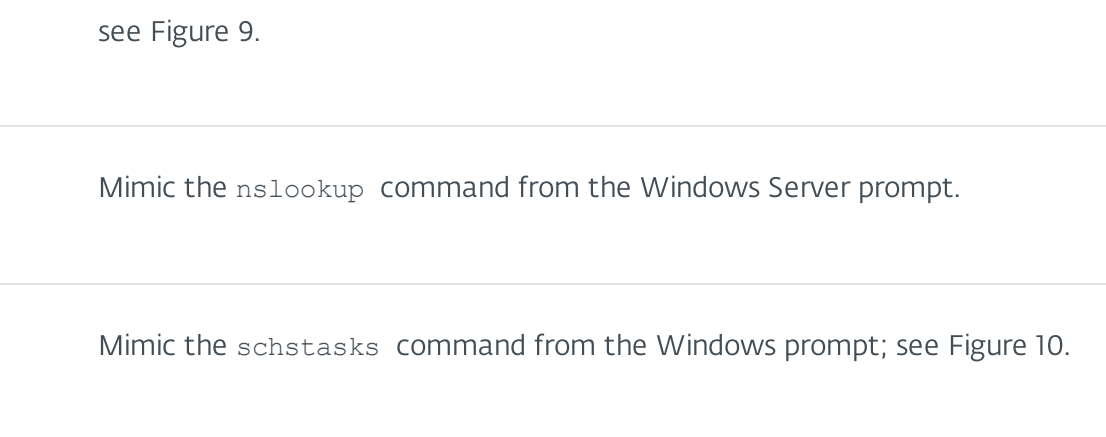


fig functionality

```
.rdata:000000018005F238                 wide_char_260 <'/delete'>
.rdata:000000018005F440                 wide_char_260 <'/del'>
.rdata:000000018005F648                 wide_char_260 <'/user'>
.rdata:000000018005F850                 wide_char_260 <'/u'>
.rdata:000000018005FA58                 wide_char_260 <'/add'>
.rdata:000000018005FC60                 wide_char_260 <'/domain'>
```

Figure 5. Hardcoded strings revealing the subset of the net functionality

```
.rdata:000000018005CBA0 netsh_params    wide_char_260 <'unknown'>
.rdata:000000018005CBA0                                       ; DATA XREF: Command__netsh+56↑o
.rdata:000000018005CBA0                                       ; Command__netsh:loc_18000ACC0↑o
.rdata:000000018005CDA8                 wide_char_260 <'add'>
.rdata:000000018005CFB0                 wide_char_260 <'show'>
.rdata:000000018005D1B8                 wide_char_260 <'delete'>
.rdata:000000018005D3C0                 wide_char_260 <'name='>
.rdata:000000018005D5C8                 wide_char_260 <'dir='>
.rdata:000000018005D7D0                 wide_char_260 <'action='>
.rdata:000000018005D9D8                 wide_char_260 <'protocol='>
.rdata:000000018005DBE0                 wide_char_260 <'localport='>
.rdata:000000018005DDE8                 wide_char_260 <'remoteport='>
.rdata:000000018005DFF0                 wide_char_260 <'description'>
.rdata:000000018005E1F8                 wide_char_260 <'program='>
```

Figure 6. Hardcoded strings revealing the netsh firewall functionality

```
.rdata:000000018005AD20 reg_params      wide_char_260 <'unknown'>
.rdata:000000018005AD20                                       ; DATA XREF: Command__reg+53↑o
.rdata:000000018005AD20                                       ; sub_18000EC10:loc_18000ED10↑o
.rdata:000000018005AF28                 wide_char_260 <'query'>
.rdata:000000018005B130                 wide_char_260 <'add'>
.rdata:000000018005B338                 wide_char_260 <'delete'>
.rdata:000000018005B540                 wide_char_260 <'save'>
.rdata:000000018005B748                 wide_char_260 <'/v'>
.rdata:000000018005B950                 wide_char_260 <'/ve'>
.rdata:000000018005BB58                 wide_char_260 <'/s'>
.rdata:000000018005BD60                 wide_char_260 <'/se'>
.rdata:000000018005BF68                 wide_char_260 <'/f'>
.rdata:000000018005C170                 wide_char_260 <'/k'>
.rdata:000000018005C378                 wide_char_260 <'/d'>
.rdata:000000018005C580                 wide_char_260 <'/t'>
.rdata:000000018005C788                 wide_char_260 <'/va'>
.rdata:000000018005C990                 wide_char_260 <'/y'>
```

Figure 7. Hardcoded strings revealing the (partial) reg functionality

```
.rdata:0000000180592B0 sc_params        wide_char_260 <'unknown'>
.rdata:0000000180592B0                                ; DATA XREF: Command__sc:loc_180010BA0↑o
.rdata:0000000180592B0                                ; sub_180010D00:loc_180010D92↑o
.rdata:00000001800594B8                 wide_char_260 <'create'>
.rdata:00000001800596C0                 wide_char_260 <'delete'>
.rdata:00000001800598C8                 wide_char_260 <'stop'>
.rdata:0000000180059AD0                 wide_char_260 <'query'>
.rdata:0000000180059CD8                 wide_char_260 <'start'>
.rdata:0000000180059EE0                 wide_char_260 <'binpath='>
.rdata:000000018005A0E8                 wide_char_260 <'type='>
.rdata:000000018005A2F0                 wide_char_260 <'start='>
.rdata:000000018005A4F8                 wide_char_260 <'error='>
.rdata:000000018005A700                 wide_char_260 <'displayname='>
.rdata:000000018005A908                 wide_char_260 <'obj='>
.rdata:000000018005AB10                 wide_char_260 <'password='>
```

Figure 8. Hardcoded strings revealing the (partial) sc functionality

```
.rdata:00000001800520F0 wmic_params     wide_char_260 <'unknown'>
.rdata:00000001800520F0                                ; DATA XREF: Command__wmic+4E↑o
.rdata:00000001800522F8                 wide_char_260 <'process'>
```

call create

```
                                        Command__schtasks+46↑o
                                        schtasks:loc_1800122A0↑o
                                        wide_char_260 <'/np'>
                                        wide_char_260 <'/z'>
                                        wide_char_260 <'/xml'>
                                        wide_char_260 <'/v1'>
                                        wide_char_260 <'/f'>
                                        wide_char_260 <'/rl'>
                                        wide_char_260 <'/delay'>
                                        wide_char_260 <'/author'>
                                        wide_char_260 <'minute'>
                                        wide_char_260 <'hourly'>
                                        wide_char_260 <'daily'>
                                        wide_char_260 <'weekly'>
                                        wide_char_260 <'monthly'>
.rdata:0000000180054DA0                 wide_char_260 <'/d'>
                                        wide_char_260 <'once'>
```

```
.rdata:0000000180054FA8    wide_char_260 <'/m'>        wide_char_260 <'onstart'>
.rdata:00000001800551B0    wide_char_260 <'/i'>        wide_char_260 <'onlogon'>
.rdata:00000001800553B8    wide_char_260 <'/tn'>       wide_char_260 <'onidle'>
.rdata:00000001800555C0    wide_char_260 <'/tr'>       wide_char_260 <'onevent'>
.rdata:00000001800557C8    wide_char_260 <'/args'>     wide_char_260 <'onregist'>
```

Figure 10. Hardcoded strings revealing the (partial) schtasks functionality

Furthermore, an examination of the RAT's internal configuration suggests that, in comparison to BlindingCan, Lazarus increased the code sophistication in LightlessCan.

Technical analysis

In this section, we provide technical details about the compromise chain that delivers the NickelLoader downloader, and the three execution chains Lazarus used to deliver its payloads on the compromised system.

Compromise chain: NickelLoader

NickelLoader is an HTTP(S) downloader executed on the compromised system via DLL side-loading, which is later used to deliver other Lazarus payloads.

The process of delivering NickelLoader unfolds in a series of stages, commencing with the execution of `PresentationHost.exe`, which is triggered automatically after the target manually executes the initial quiz challenges; the `Quiz1` case is depicted in Figure 3. A malicious dynamically linked library, `mscoree.dll`, is then side-loaded by the legitimate `PresentationHost.exe` – both located in `C:\ProgramShared\`. This DLL is a trojanized `NppyPluginDll.dll`, from the inactive General Python Plugins DLL for Notepad++ project from 2011. It serves as a dropper and has various exports: all the exports copied from the original `NppyPluginDll.dll` plus all the exports from the legitimate `mscoree.dll`. One of these legitimate exports, `CorExitProcess`, contains the malicious code responsible for the decryption and execution of the next malware stage.

To successfully decrypt an encrypted data array embedded in the dropper, three 16-character-long keywords are required by the dropper. These keywords are as follows:

1. the name of the parent process (`PresentationHost`),

`6o78753qg8`), and

`mbeddingObject`),

`tionHost.exe`, being

ns the AES-128

commands, all five

ommands, we chose to

the colloquial term for

ands are `avdrq` and

ds data received from

the C&C server as a DLL. For this purpose, the attackers probably used

MemoryModule, a library that can be used to load a DLL completely from memory.

Table 3. The list of magic keywords recognized in received buffers

Keyword	Description
`abcde`	Requests another immediate command without the usual long sleep delay that separates the execution of the commands.
`avdrq`	Loads a DLL contained in the received buffer and executes its hardcoded export info.
`gabnc`	Loads a DLL contained in the received buffer.
`dcrqv`	Terminates itself.

Execution chain 1: miniBlindingCan

One of the payloads downloaded and executed by NickelLoader is miniBlindingCan, a simplified version of the group's flagship BlindingCan RAT. It was reported for the first time by Mandiant in September 2022, under the name AIRDRY.V2.

To load miniBlindingCan, a 64-bit malicious dynamically linked library `colorui.dll` is side-loaded by a legitimate `colorcpl.exe` executed from `C:\ProgramData\Adobe\` and serves as a dropper. The DLL is obfuscated using VMProtect and contains thousands of exports from which LaunchColorCpl is the most important, as it handles the execution of the next stage. There's an encrypted data array in the DLL's dumped body, together with multiple debug symbols revealing the root directory and the project from which it was built:

```
W:\Develop\aTool\ShellCodeLoader\App\libressl-2.6.5\
```

As the name `ShellCodeLoader` suggests, the main purpose of this initial stage is
tains shellcode. At the
i-debugging
the Process
scrutinized or
niques to avoid
urity analysis. The
`colorcpl.exe`; if not,

an intermediate blob
ata array, which
be produced by an
cular, the
ting the Python script

ConvertToShellcode.py from this project on a payload DLL acting as a source for reflective DLL injection.

The final payload is extracted and decrypted using XOR with a long key, which is a string built by concatenating the name of the parent process (`colorcpl.exe`), the filename of the dropper (`colorui.dll`), and the external command line parameter – in this case resulting in `COLORCPL.EXECOLORUI.DLL669498484488D3F22712CC5BACA6B7A7`. This process is akin to what we observed with BlindingCan backdoor in the Dutch case we previously described in this WeLiveSecurity blogpost. The decryption reveals an executable with download-and-execute functionality, whose internal logic of sending and parsing commands is strongly reminiscent of BlindingCan, a flagship HTTP(S) Lazarus RAT. Unlike the case in the Netherlands, it is not VMProtect-ed and it supports only a small subset of commands available previously: compare Table 4in this blogpost and Table 3 in the blogpost on the Dutch case from September 2022. Because the features of this RAT are notably scaled down compared to those in BlindingCan, and yet they seem to share the same server-side infrastructure, we have chosen to distinguish it by appending the prefix "mini-" to its name, highlighting its reduced functionality compared to its fully-featured RAT counterpart.

Table 4. Commands of miniBlindingCan

Command ID	Description
8201	Send system information like computer name, Windows version, and code page.
8232	Update the current communication interval with a value provided by the C&C server.
8233	Discontinue the command execution.
8241	Send the current configuration of size 9,392 bytes to the C&C server.

8279	Execute shellcode passed as a parameter.

Figure 11 shows the decrypted state of a 9,392-byte-long configuration embedded in the RAT. It contains five URLs, in this case compromised websites, each limited by a maximum size of 260 wide characters.

```
00000000:  00 00 05 00-00 00 68 00-74 00 74 00-70 00 73 00    ♣   h t t p s
00000010:  3A 00 2F 00-2F 00 74 00-75 00 72 00-6E 00 73 00  : / / t u r n s
00000020:  63 00 6F 00-72 00 2E 00-63 00 6F 00-6D 00 2F 00  c o r . c o m /
00000030:  77 00 70 00-2D 00 69 00-6E 00 63 00-6C 00 75 00  w p - i n c l u
00000040:  64 00 65 00-73 00 2F 00-63 00 6F 00-6E 00 74 00  d e s / c o n t
00000050:  61 00 63 00-74 00 73 00-2E 00 70 00-68 00 70 00  a c t s . p h p
00000820:  00 00 00 00-00 00 68 00-74 00 74 00-70 00 73 00        h t t p s
00000830:  3A 00 2F 00-2F 00 77 00-77 00 77 00-2E 00 65 00  : / / w w w . e
00000840:  6C 00 69 00-74 00 65 00-34 00 70 00-72 00 69 00  l i t e 4 p r i
00000850:  6E 00 74 00-2E 00 63 00-6F 00 6D 00-2F 00 73 00  n t . c o m / s
00000860:  75 00 70 00-70 00 6F 00-72 00 74 00-2F 00 73 00  u p p o r t / s
00000870:  75 00 70 00-70 00 6F 00-72 00 74 00-2E 00 61 00  u p p o r t . a
00000880:  73 00 70 00-00 00 00 00-00 00 00 00-00 00 00 00  s p
000018A0:  63 00 3A 00-5C 00 77 00-69 00 6E 00-64 00 6F 00  c : \ w i n d o
000018B0:  77 00 73 00-5C 00 73 00-79 00 73 00-74 00 65 00  w s \ s y s t e
000018C0:  6D 00 33 00-32 00 5C 00-63 00 6D 00-64 00 2E 00  m 3 2 \ c m d .
000018D0:  65 00 78 00-65 00 00 00-00 00 00 00-00 00 00 00  e x e
00001AA0:  00 00 00 00-00 00 00 00-25 00 74 00-65 00 6D 00          % t e m
00001AB0:  70 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  p
00002490:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000024A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

Figure 11. A configuration of the miniBlindingCan backdoor. The highlighted value is the count of URLs, but only the first and the last of the five URLs are shown here. The purpose of the last two wide strings is not known

Execution chain 2: LightlessCan (simple version)

Another payload we have seen executed by NickelLoader is LightlessCan, a new Lazarus backdoor. We have observed two different chains loading this backdoor.

In the simple version of the chain, the dropper of this payload is the malicious dynamically linked library mapistub.dll that is side-loaded by the legitimate fixmapi.exe executed from `C:\ProgramData\Oracle\Java\`. The DLL is a trojanized Lua plugin, version 1.4, with all the exports copied from the legitimate Windows `mapi32.dll`. The export FixMAPI contains malicious code responsible for decrypting and loading the next stage; all the other exports contain benign code sourced from a publicly available MineSweeper sample project. This

duled task.

cept that its parent

`-k netsvcs -p -s`

per needs three

`NfE9uMz63n`), and

The keywords are XOR-ed byte by byte and the output forms a 128-bit AES key to be used for decryption. Note that the length of the keywords are not all exactly 16 bytes, but the decryption process will still work if the oversized string is truncated to a 16-byte length (for instance, `AudioEndpointBuilder` to `AudioEndpointBui`), and the undersized string, `fixmapi.exe`, is treated as `fixmapi.exe\x00\x00\x00\x00\x00`, because the string was initialized as 260 instances of the NUL character.

Execution chain 3: LightlessCan (complex version)

The most complex chain we observed on the compromised system also delivers LightlessCan, with various components involved in the complete chain of installation stages: a legitimate application, an initial dropper, a complete dropper (which contains the configuration), an intermediate dropper, a configuration file, a file with system information (for the decryption of encrypted payloads on the file system), an intermediate loader and the final step, the LightlessCan RAT. The connections and relationships among these files are illustrated in Figure 12.

th payload

The initial dropper of the fourth chain is a malicious dynamically linked library `HID.dll` that is side-loaded by a legitimate executable, `tabcal.exe`, executed from `C:\ProgramData\Adobe\ARM\`. The DLL is a trojanized version of `MZC8051.dll`, a legitimate file from the 8051 C compiler plugin project for Notepad++. It contains all the exports from the original project, but also the necessary exports from the legitimate Hid User Library by Microsoft, so that the side-loading by tabcal.exe will be successful. The export `HidD_GetHidGuid` contains the malicious code responsible for dropping the next stage and, as in the case of the dropper of the previous chain (Execution chain 2), all the other exports contain the benign MineSweeper code.

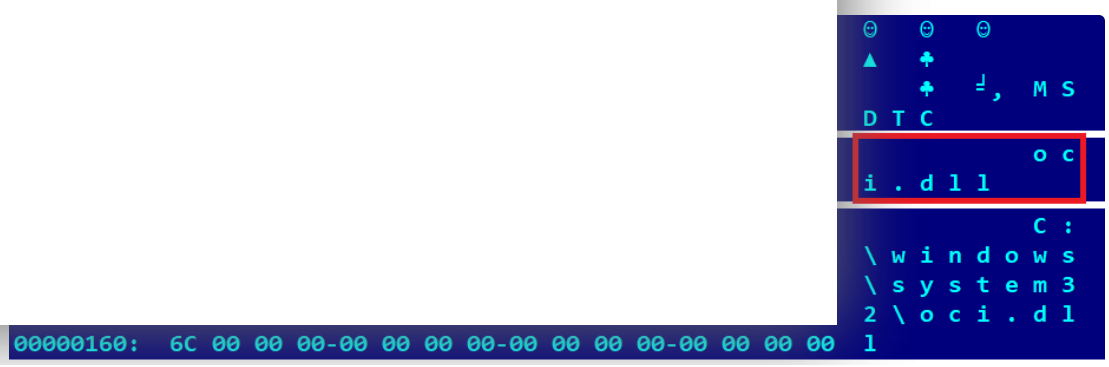
As in the previous cases, three long keywords must be provided to decrypt the embedded payload:

1. the name of the parent process (`tabcal.exe`),
2. the internal parameter hardcoded in the binary (`9zCnQP6o78753qg8`), and
3. the external parameter (`LocalServiceNetworkRestricted`) – this time not expressed as a command line parameter, but instead as the content of a file located at `%WINDOWS%\system32\thumbs.db`.

Again, the keywords are XOR-ed byte by byte and the output forms a 128-bit AES key to be used for the decryption. As in the previous case, the lengths of the keywords are not all exactly 16 bytes, but the decryption will still work if the oversized string is truncated (for instance, to `LocalServiceNetw`) and the undersized string is extended with nulls (for instance, to `tabcal.exe\x00\x00\x00\x00\x00\x00`).

The executable produced by the above recipe is the complete dropper from Figure 12 and has the InternalName resource `AppResolver.dll` (found in the VERSIONINFO resource). It contains two encrypted data arrays: a small one of 126 bytes, and a large one of 1,807,464 bytes (which contains three subparts). First, it decrypts the small array using the RC6 algorithm with the hardcoded 256-bit key `DA 48 A3 14 8D BF E2 D2 EF 91 12 11 FF 75 59 A3 E1 6E A0 64 B8 78 89 77 A0 37 91 58 5A FF FF 07`. The output represents paths to which the first two subparts of the large blob are dropped (i.e., LightlessCan and the intermediate dropper), and yields the strings `C:\windows\system32\oci.dll` and `C:\windows\system32\grpedit.dat`.

he large blob – using
l blob containing three
an), a DLL
a 14,948 byte
.cpl (configuration);



```
00000930:  43 00 3A 00-5C 00 77 00-69 00 6E 00-64 00 6F 00  C : \ w i n d o
00000940:  77 00 73 00-5C 00 73 00-79 00 73 00-74 00 65 00  w s \ s y s t e
00000950:  6D 00 33 00-32 00 5C 00-67 00 72 00-70 00 65 00  m 3 2 \ g r p e
00000960:  64 00 69 00-74 00 2E 00-64 00 61 00-74 00 00 00  d i t . d a t

00001130:  00 00 00 00-68 00 74 00-74 00 70 00-73 00 3A 00      h t t p s :
00001140:  2F 00 2F 00-6B 00 61 00-70 00 61 00-74 00 61 00  / / k a p a t a
00001150:  2D 00 61 00-72 00 6B 00-65 00 6F 00-6C 00 6F 00  - a r k e o l o
00001160:  67 00 69 00-2E 00 6B 00-65 00 6D 00-64 00 69 00  g i . k e m d i
00001170:  6B 00 62 00-75 00 64 00-2E 00 67 00-6F 00 2E 00  k b u d . g o .
00001180:  69 00 64 00-2F 00 70 00-61 00 67 00-65 00 73 00  i d / p a g e s
00001190:  2F 00 70 00-61 00 79 00-6D 00 65 00-6E 00 74 00  / p a y m e n t
000011A0:  2F 00 70 00-61 00 79 00-6D 00 65 00-6E 00 74 00  / p a y m e n t
000011B0:  2E 00 70 00-68 00 70 00-00 00 00 00-00 00 00 00  . p h p
000011C0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00

00003A30:  00 00 00 00-00 00 31 00-2E 00 30 00-00 00 00 00        1 . 0
00003A40:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00003A50:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00003A60:  00 00 00 00-           -           -
```

Figure 13. The decrypted configuration stored in wlansvc.cpl

Moreover, the complete dropper also stores several characteristics identifying the compromised system in the file `%WINDOWS%\System32\4F59FB87DF2F`, whose name is hardcoded in the binary. These characteristics are primarily retrieved from the `Computer\HKLM\HARDWARE\DESCRIPTION\System\BIOS` registry path. Here are the specific values of these characteristics, along with a PowerShell command provided in brackets that can be used to display the corresponding value on any Windows machine:

- SystemBIOSDate (`Get-ItemProperty "HKLM:\HARDWARE\Description\System\BIOS" -Name BIOSReleaseDate | Select-Object -Property BIOSReleaseDate`)
- SystemBIOSVersion (`Get-CimInstance -ClassName Win32_Bios | Select-Object -Property Version`)
- SystemManufacturer (`Get-CimInstance -ClassName Win32_ComputerSystem | Select-Object -Property Manufacturer`)
- SystemProductName (`Get-CimInstance -ClassName Win32_ComputerSystemProduct | Select-Object -Property Name`)
- Identifier in `Computer\HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\MultifunctionAdapter\0\DiskController\0\DiskPeripheral\0`

The concatenation of the values is required for decryption of the encrypted
ing an image of
ual
te dropper that drops
e described in the
an open-source
onger available online.
provided in order to

2. the internal parameter hardcoded in the binary (`fb5XPNCr8v83Y85P`).

Both keywords are XOR-ed byte by byte (the parent process name is truncated, or padded with NULLs, as necessary to fill 16 bytes). The product of the decryption is the intermediate loader (`LLTMapperAPI.dll`). It uses the system information (same as the values stored in `4F59FB87DF2F`) to decrypt the configuration file `wlansvc.cpl` and to locate, decrypt, and load the encrypted `grpedit.dat`, which is LightlessCan, the new full-featured RAT.

Conclusion

We have described a new Lazarus attack that originated on LinkedIn where fake recruiters approached their potential victims, who were using corporate computers for personal purposes. Even though public awareness of these types of attacks should be high, the success rates of these campaigns have still not dropped to zero.

The most worrying aspect of the attack is the new type of payload, LightlessCan, a complex and possibly evolving tool that exhibits a high level of sophistication in its design and operation, representing a significant advancement in malicious capabilities compared to its predecessor, BlindingCan.

The attackers can now significantly limit the execution traces of their favorite Windows command line programs that are heavily used in their post-compromise activity. This maneuver has far-reaching implications, impacting the effectiveness of both real-time monitoring solutions and of post-mortem digital forensic tools.

IoCs

Files

SHA-1	Filename
C273B244EA7DFF20B1D6B1C7FD97F343201984B3	%TEMP%\7zOC35416EE\Quiz1.exe
	3CC96D\Quiz2.exe
	FILE%\Adobe\colorui.dll
	FILE%\Oracle\Java\mapis

C136DD71F45EAEF3206BF5C03412195227D15F38	C:\ProgramShared\mscoree.dll
E61672B23DBD03FE3B97EE469FA0895ED1F9185D	N/A
E18B9743EC203AB49D3B57FED6DF5A99061F80E0	%ALLUSERSPROFILE%\Adobe\ARM\HID.dl
10BD3E6BA6A48D3F2E056C4F974D90549AED1B96	N/A
3007DDA05CA8C7DE85CD169F3773D43B1A009318	%WINDIR%\system32\grpedit.dat
247C5F59CFFBAF099203F5BA3680F82A95C51E6E	%WINDIR%\system32\oci.dll
EBD3EF268C71A0ED11AE103AA745F1D8A63DDF13	N/A

Network

IP	Domain	Hosting provider	First s
46.105.57[.]169	bug.restoroad[.]com	OVH SAS	2021-10-
50.192.28[.]29	hurricanepub[.]com	Comcast Cable Communications, LLC	2020-01
		Web, L.L.C	2020-01
		SA	2021-03
		Ltd	2020-10
118.98.221[.]114	kapata-[illegible]	Pustekkom	2020-01

118.98.221[.]14	arkeologi.kemdikbud.go[.]id	Pustekkom	2020-01
160.153.33[.]195	barsaji.com[.]mx	GoDaddy.com, LLC	2020-03
175.207.13[.]231	www.keewoom.co[.]kr	Korea Telecom	2021-01-
178.251.26[.]65	kerstpakketten.horesca-meppel[.]nl	InterRacks B.V.	2020-11-
185.51.65[.]233	kittimasszazs[.]hu	DoclerNet Operations, ORG-DHK1-RIPE	2020-02
199.188.206[.]75	nrfm[.]lk	Namecheap, Inc.	2021-03-

MITRE ATT&CK techniques

This table was built using version 13 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
			zarus attackers used kedIn to identify and ntact specific employees a company of interest.
			mpromised servers were ed by the Lazarus HTTP(S) ckdoors and the wnloader for C&C.
		Social Media Accounts	zarus attackers created a e LinkedIn identity of a headhunter from Meta.

Resource Development	*T1585.003*	Establish Accounts: Cloud Accounts	Lazarus attackers had to create an account on a third-party cloud storage in order to deliver the initial ISO images.
	T1587.001	Develop Capabilities: Malware	Custom tools from the attack are likely developed by the attackers. Some exhibit highly specific kernel development capacities seen earlier in Lazarus tools.
	T1608.001	Stage Capabilities: Upload Malware	Lazarus attackers uploaded the initial ISO images to a cloud storage.
Initial Access	*T1566.002*	Phishing: Spearphishing Link	The target received a link to a third-party remote storage with malicious ISO images.
	T1566.003	Phishing: Spearphishing via Service	The target was contacted via LinkedIn Messaging.
	T1106	Native API	Windows APIs are essential for miniBlindingCan and LightlessCan to function and are resolved dynamically at runtime.
			sed on the parent ocess, a scheduled task s probably created to gger thesimple chain of LightlessCan execution.
			kelLoader can load and ecute an arbitrary DLL hin memory.
		Malicious File	zarus attackers relied on execution of `Quiz1.exe` and `Quiz2.exe` from the

			ISO files.
	T1047	Windows Management Instrumentation	One of the LightlessCan commands allows creation of a new process via WMI.
Persistence	*T1053*	Scheduled Task/Job	Based on the parent process, a scheduled task was probably created to trigger the simple chain of the LightlessCan execution. Moreover, LightlessCan can mimic the `schtasks` command.
	T1134.002	Access Token Manipulation: Create Process with Token	LightlessCan can create a new process in the security context of the user represented by the specified token and collect the output.
	T1622	Debugger Evasion	There's an anti-debug check in the dropper of miniBlindingCan.
	T1480	Execution Guardrails	There's a parent process check in the miniBlindingCan dropper. The concatenation of the values is required for decryption of the encrypted LightlessCan from the file system.
			ny of these Lazarus tools d configurations are crypted on the file tem, e.g., LightlessCan in `edit.dat` and its figuration in `nsvc.cpl`.
			ny of the Lazarus ppers and loaders use a itimate program for their ding.

Defense Evasion	*T1027.002*	Obfuscated Files or Information: Software Packing	Lazarus obfuscated several executables by VMProtect in this attack, e.g., `colorui.dll`
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution	Both LightlessCan and miniBlindingCan resolve Windows APIs dynamically.
	T1027.009	Obfuscated Files or Information: Embedded Payloads	The droppers of all malicious chains contain an embedded data array with an additional stage.
	T1562.003	Impair Defenses: Impair Command History Logging	New features of LightlessCan mimic the most useful Windows command line utilities, to avoid executing the original console utilities.
	T1562.004	Impair Defenses: Disable or Modify System Firewall	LightlessCan can mimic the `netsh` command and interact with firewall rules.
	T1070.004	Indicator Removal: File Deletion	LightlessCan has the ability to delete files securely.
	T1070.006	Indicator Removal: Timestomp	LightlessCan can alter the modification timestamps of files.
			LightlessCan bypasses mmand execution by plementing their ctionality.
			htlessCan and niBlindingCan use various es of process injection.
			e miniBlindingCan dropper s an intentional initial ecution delay.

	T1620	Reflective Code Loading	Most of the droppers use reflective DLL injection.
	T1083	File and Directory Discovery	LightlessCan can locate a file by its name.
	T1135	Network Share Discovery	LightlessCan can mimic the `net share` command.
	T1057	Process Discovery	LightlessCan identifies processes by name.
	T1012	Query Registry	LightlessCan queries the registry for various system information it uses for encryption.
Discovery	*T1018*	Remote System Discovery	LightlessCan can mimic the `net view` command.
	T1016	System Network Configuration Discovery	LightlessCan can mimic the `arp` and `ipconfig` commands.
	T1049	System Network Connections Discovery	LightlessCan can mimic the `netstat` command.
	T1007	System Service Discovery	LightlessCan can mimic the `sc query` and `tasklist` commands.
		Application Layer	NickelLoader, LightlessCan, d miniBlindingCan use TP and HTTPS for C&C.
			htlessCan and iBlindingCan encrypt C traffic using the AES- algorithm.
			htlessCan and iBlindingCan encode C&C ffic using base64.

Exfiltration	*T1041*	Exfiltration Over C2 Channel	LightlessCan can exfiltrate data to its C&C server.

Related Articles

ESET RESEARCH
CloudScout: Evasive Panda scouting cloud services

ESET RESEARCH
ESET Research Podcast: CosmicBeetle

ESET RESEARCH
Embargo ransomware: Rock'n'Rust

Discussion

What do you think?

7 Responses

Upvote Funny Love Surprised Angry Sad

0 Comments Login

Start the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

Share Best Newest Oldest