


North Korea's Lazarus: their initial access trade-craft using social media and social engineering

 research.nccgroup.com/2022/05/05/north-koreas-lazarus-and-their-initial-access-trade-craft-using-social-media-and-social-engineering/

May 5, 2022



Authored by: **Michael Matthews** and **Nikolaos Pantazopoulos**

tl;dr

This blog post documents some of the actions taken during the initial access phase for an attack attributed to Lazarus, along with analysis of the malware that was utilised during this phase.

The methods used in order to gain access to a victim network are widely reported however, nuances in post-exploitation provide a wealth of information on attack paths and threat hunting material that relate closely to TTP's of the Lazarus group.

In summary, we identified the following findings:

- Lazarus used LinkedIn profiles to impersonate employees of other legitimate companies
- Lazarus communicated with target employees through communication channels such as WhatsApp.
- Lazarus entices victims to download job adverts (zip files) containing malicious documents that lead to the execution of malware
- The identified malicious downloader appears to be a variant of LCPDOT
- Scheduled tasks are utilised as a form of persistence (`rund1132` execution from a scheduled task)

Initial Access

In line with what is publicly documented[1], the initial entry revolves heavily around social engineering, with recent efforts involving the impersonation of Lockheed Martin employees with LinkedIn profiles to persuade victims into following up with job opportunities that result in a malicious document being delivered.

In this instance, the domain hosting the document was `global-job[.]org`, likely attempting to impersonate `globaljobs[.]org`, a US based government/defence recruitment website. In order to subvert security controls in the recent changes made by Microsoft for Office macros, the website hosted a ZIP file which contained the malicious document.

The document had several characteristics comparable to other Lazarus samples however, due to unknown circumstances, the “shapes” containing the payloads were unavailable and could not be analysed.

Following the execution of the macro document, `rund1132.exe` is called to execute the DLL `C:\programdata\packages.mdb`, which then led to the initial command-and-control server call out. Unfortunately, the binary itself was no longer available for analysis however, it is believed that this component led to the LCPDot malware being placed on the victim’s host.

LCPDot

We were able to recover a malicious downloader that was executed as a scheduled task. The identified sample appears to be a variant of LCPDot, and it is attributed to the threat actor ‘Larazus’.

The file in question attempted to blend into the environment, leveraging the `ProgramData` directory once again `C:\ProgramData\Oracle\Java\JavaPackage.dll`. However, the file had characteristics that stand out whilst threat hunting:

- Large file size (60mb+) – likely to bypass anti-virus scanning
- Time stamping – timestamps copied from `CMD.exe`
- DLL owned by a user in the `ProgramData` directory (Not SYSTEM or Administrator)

To execute LCPDot, a scheduled task was created named “*Windows Java Vpn Interface*”, attempting to blend into the system with the Java theme. The scheduled task executed the binary but also allowed the threat actor to persist.

The scheduled task was set to run daily with the following parameter passed for execution, running:

```
<Exec>  
  <Command>c:\windows\system32\rundll32.exe</Command>
```

```
<Arguments>C:\ProgramData\Oracle\Java\JavaPackage.dll,VpnUserInterface</Arguments>  
</Exec>
```

LCPDot Binary Analysis

The downloader’s malicious core starts in a separate thread and the execution flow is determined based on Windows messages IDs (sent by the Windows API function `SendMessage`).

In the following sections we describe the most important features that we identified during our analysis.

Initialisation Phase

The initialisation phase takes place in a new thread and the following tasks are performed:

- Initialisation of class `MoscowTownList` . This class has the functionality to read/write the configuration.
- Creation of configuration file on disk. The configuration file is stored under the filename `VirtualStore.cab` in `%APPDATA%\Local` folder. The configuration includes various metadata along with the command-and-control servers URLs. The structure that it uses is:

```

struct Configuration
{
    DWORD Unknown; //Unknown, set to 0 by default. If higher than 20 then it
                  // can cause a 2-hour delay during the network
                  // communication process.
    SYSTEMTIME Variant_SystemTime; // Configuration timestamp created by
                                  // SystemTimeToVariantTime.
    SYSTEMTIME Host_SystemTime; // Configuration timestamp. Updated during
                               // network communication process.
    DWORD Logical_drives_find_flag; // Set to 0 by default
    DWORD Active_sessions_flag; // Set to 0 by default
    DWORD Boot_Time; // Milliseconds since boot time
    char *C2_Data;// Command-and-Control servers' domains
};

```

The configuration is encrypted by hashing (SHA-1) a random byte array (16 bytes) and then uses the hash output to derive (CryptDeriveKey) a RC4 key (16 bytes). Lastly it writes to the configuration file the random byte array followed by the encrypted configuration data.

Enumeration of logical drives and active logon sessions. The enumeration happens only if specified in the configuration. By default, this option is off. Furthermore, even if enabled, it does not appear to have any effect (e.g. sending them to the command-and-control server).

Once this phase is completed, the downloader starts the network communication with its command-and-control servers.

Network Communication

At this stage, the downloader registers the compromised host to the command-and-control server and then requests the payload to execute. In summary, the following steps are taken:

- Initialises the classes `Taxiroad` and `WashingtonRoad` .
- Creates a byte array (16 bytes), which is then encoded (base64), and a session ID. Both are sent to the server. The encoded byte array is used later to decrypt the received payload and is added to the body content of the request:
`redirect=Yes&idx=%d&num=%s` , where `idx` holds the compromised host's boot time value and `num` has the (BASE64) encoded byte array.
 In addition, the session ID is encoded (BASE64) and added to the following string:
`SESSIONID-%d-202110` , where `202110` is the network command ID.
 The above string is encoded again (BASE64) and then added to the `SESSIONID` header of the POST request.

After registering the compromised host, the server replies with one of the following messages:

- Validation Success – Bot registered without any issues.

- Validation Error – An error occurred.

Once the registration process has been completed, the downloader sends a GET request to download the second-stage payload. The received payload is decrypted by hashing (SHA-1) the previously created byte array and then use the resulting hash to derive (`CryptDeriveKey`) a RC4 key.

Lastly, the decrypted payload is loaded directly into memory and executed in a new thread.

In summary, we identified the following commands (Table 1).

Command ID	Description
202110	Register compromised host to the command-and-control server
202111	Request payload from the command-and-control server

Table 1 – Identified network commands

Unused Commands and Functions

One interesting observation is the presence of functions and network commands, which the downloader does not seem to use. Therefore, we concluded that the following network commands are not used by the downloader (at least in this variant) but we do believe that the operators may use them on the server-side (e.g. in the PHP scripts that the downloader sends data) or the loaded payload does use them (*Note: Commands 789020, 789021 and 789022 are by default disabled*):

- 202112 – Sends encrypted data in a POST request. Data context is unknown.
- 202114 – Sends a POST request with body content 'Cookie=Enable'.
- 789020 – Same functionality as command ID 202111.
- 789021 – Same functionality as command ID 202112.
- 789022 – Sends a POST request with body content 'Cookie=Enable'.

Indicators of Compromise

Domains

- ats[.]apvit[.]com – Legitimate Compromised website
- bugs-hpsm[.]mobitechnologies[.]com – Legitimate Compromised website
- global-job[.]org
- thefroster[.]co[.]uk – Legitimate Compromised website
- shoppingbagsdirect[.]com – Legitimate Compromised website

IP Address

13[.]88[.]245[.]250

Hashes

Javapackage.dll

MD5: AFBCB626B770B1F87FF9B5721D2F3235

SHA1: D25A4F20C0B9D982D63FC0135798384C17226B55

SHA256:

FD02E0F5FCF97022AC266A3E54888080F66760D731903FC32DF2E17E6E1E4C64

Virtualstore.cab

MD5: 49C2821A940846BDACB8A3457BE4663C

SHA1: 0A6F762A47557E369DB8655A0D14AB088926E05B

SHA256:

F4E314E8007104974681D92267673AC22721F756D8E1925142D9C26DC8A0FFB4

MITRE ATT&CK

Technique	ID
Phishing: Spearphishing via Service	T1566.003
Scheduled Task/Job: Scheduled Task	T1053.005
User Execution: Malicious File	T1204.002
Application Layer Protocol	T1071.001

MITRE ATT&CK

References

[1] <https://www.microsoft.com/security/blog/2021/01/28/zinc-attacks-against-security-researchers/>