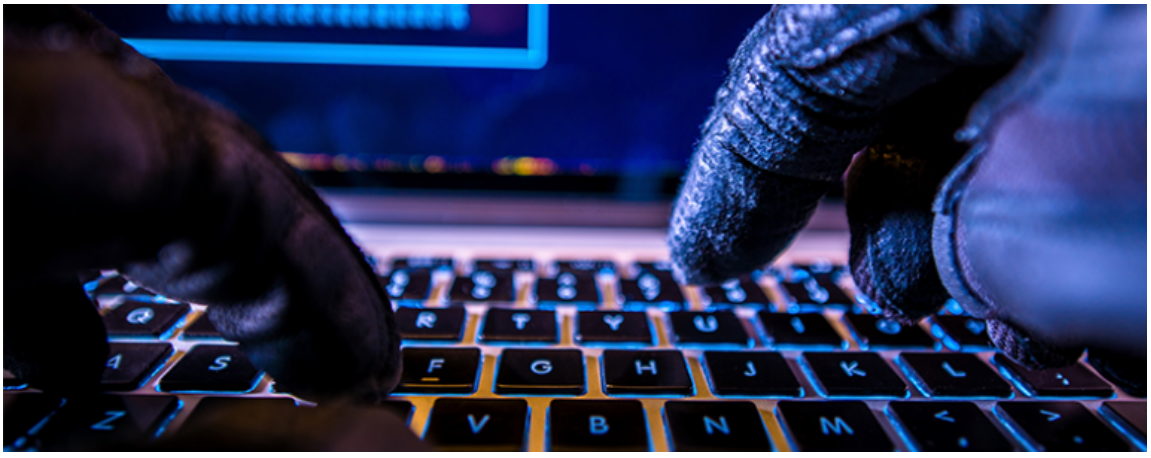
NimzaLoader: TA800's New Initial Access Malware

proofpoint.com/us/blog/threat-insight/nimzaloader-ta800s-new-initial-access-malware

March 9, 2021

Blog
Threat Insight
NimzaLoader: TA800's New Initial Access Malware



March 10, 2021 Dennis Schwarz, Matthew Mesa, and the Proofpoint Threat Research Team

Proofpoint researchers observed an interesting email campaign by a threat actor we track as TA800. This actor has predominantly used BazaLoader since April of 2020, but on February 3rd, 2021 they distributed a new malware we are calling NimzaLoader. One of NimzaLoader's distinguishing features is that it is written in the Nim programming language. Malware written in Nim are rare in the threat landscape. Malware developers may choose to use a rare programming language to avoid detection, as reverse engineers may not be familiar with Nim's implementation, or focused on developing detection for it, and therefore tools and sandboxes may struggle to analyze samples of it.

There has been some initial analysis of the malware on Twitter indicating that it may just be another variant of BazaLoader, of which there are many variants. On March 1st, Joshua Platt and Jason Reaves from Walmart put forth an excellent writeup on this malware that they call Nimar Loader. Our independent analysis corroborates their analysis and assertions that this malware is not a BazaLoader variant. Some of the major differences between NimzaLoader and the BazaLoader variants that we've analyzed include:

- Written in a completely different programming language
- Doesn't use the same code flattening obfuscator
- Doesn't use the same style of string decryption
- Doesn't use the same XOR/rotate based Windows API hashing algorithm
- Doesn't use the same RC4 using dates as the key command and control (C&C) response decryption
- Doesn't use a domain generation algorithm (DGA)
- Makes use of JSON in C&C communications

In this post we'll take a closer look at the email campaign and the malware.

Campaign Analysis

On February 3rd, 2021, Proofpoint observed a TA800 campaign distributing NimzaLoader. Consistent with previous activity, this campaign utilized personalized details in its lure, including, the recipient's name and/or the company's name. The messages contained links, which in some cases were shortened links, purporting to be a link to a PDF preview, but instead linked to GetResponse (an email marketing service) landing pages. The landing pages contained links to the "PDF" which was the NimzaLoader executable hosted on Slack and used a fake Adobe icon in an attempt to fool the user.

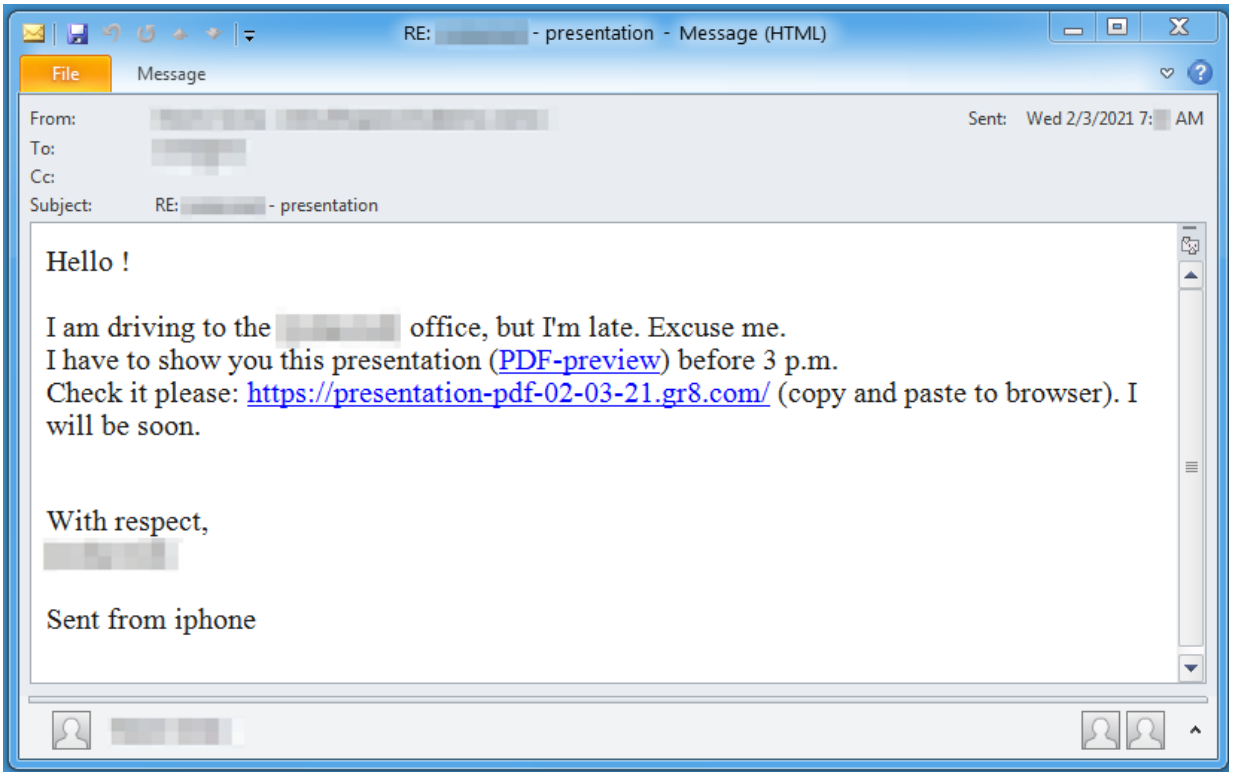


Figure 1: TA800 message linking to the GetResponse Landing Page

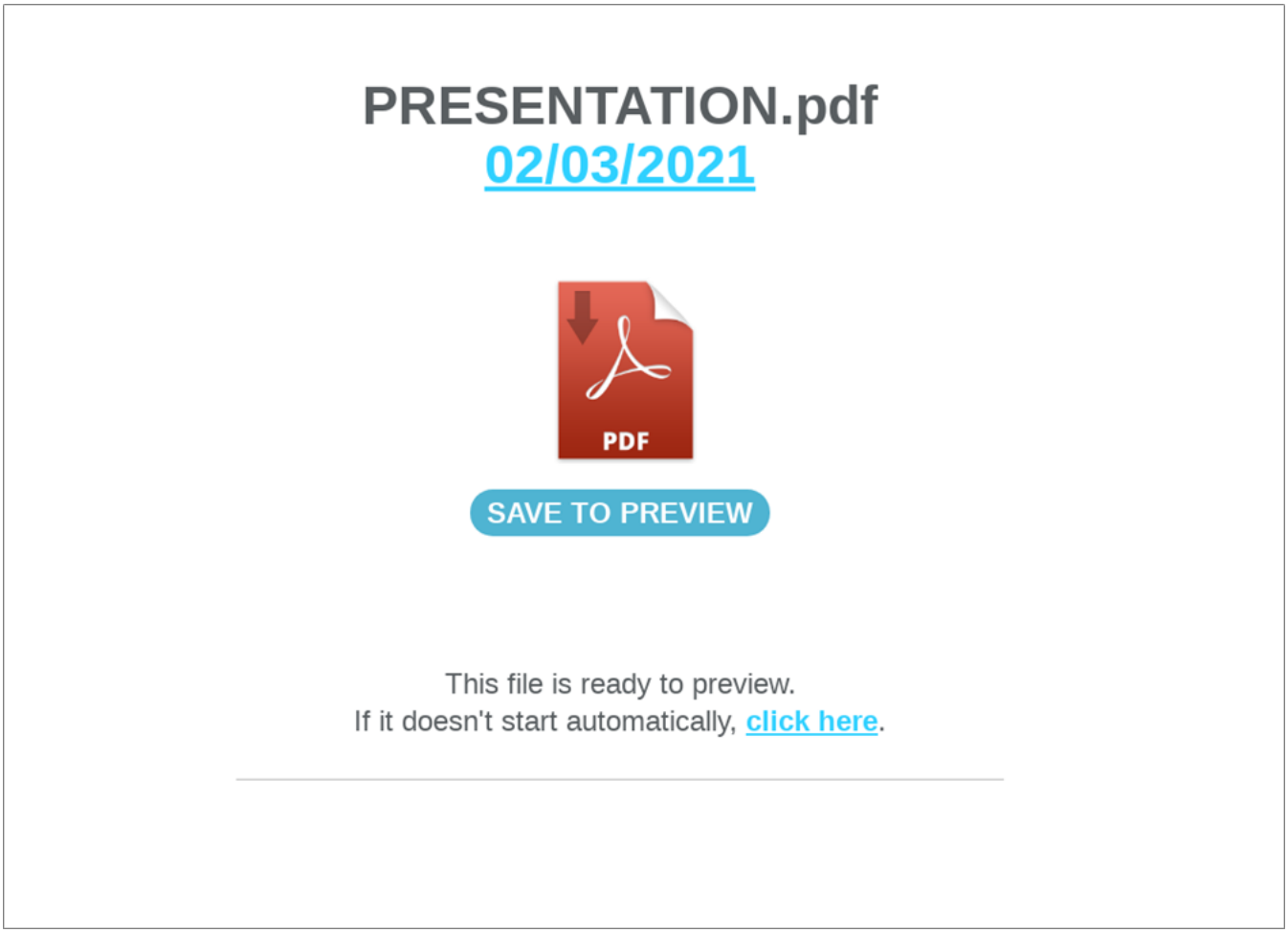


Figure 2: TA800 GetResponse Landing page linking to the download of NimzaLoader

Malware Analysis

The sample with a SHA-256 hash of 540c91d46a1aa2bb306f9cc15b93bdab6c4784047d64b95561cf2759368d3d1d was reverse engineered for this analysis. At the time of research, the C&C servers were already down, so we also made use of a PCAP file uploaded to VirusTotal.

Nim Programming Language

NimzaLoader was developed using the Nim programming language. This can be seen by the various "nim" related strings in the executable (Figure 3):

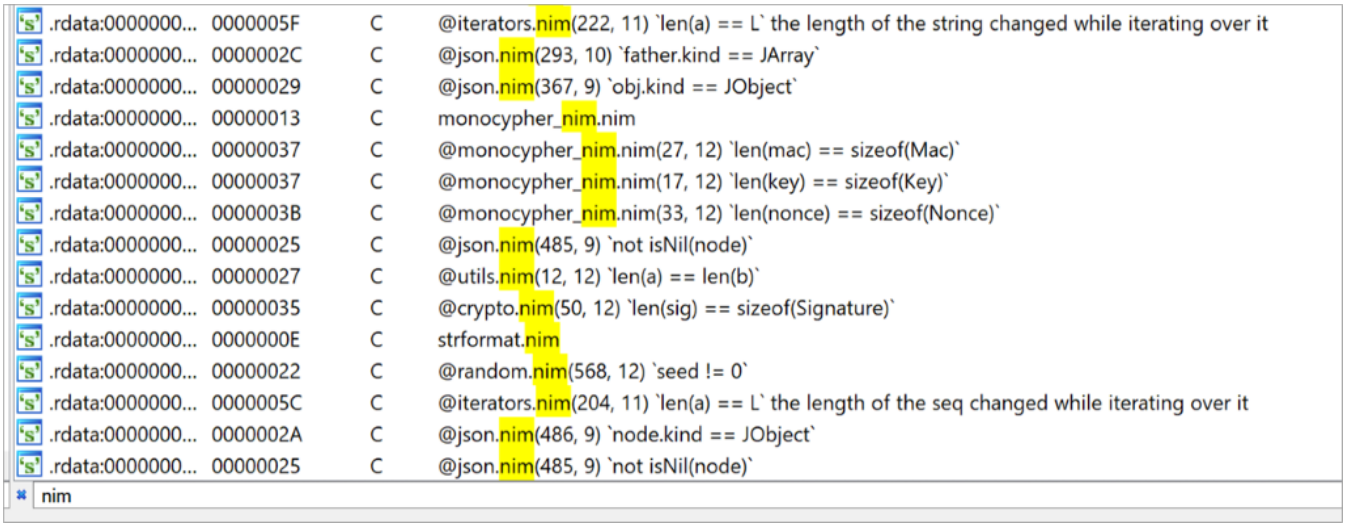


Figure 3: Example of Nim related strings

String Encryption

Most of the strings used by the malware are encrypted when stored by using an XOR-based algorithm and a single key per string. An IDA Python function of the algorithm will be available on our GitHub. Here is a listing of decrypted strings from the analyzed sample:

- 1612963255.0039535
- 1OcYomEX0BsbkWCzLHRggQ==
- ;
- ;\r\n
- =
- APISID
- CV54fakIvNL14Br0vFqSiw==
- Cookie:
- GET
- JSESSIONID
- SID
- WMCIf52ORF4UAztWoqpcAtAdZeysf2lWi0FvUE/L7Uc=
- \r\n
- about
- e8cbd40fda2500cd496b55df43402d8ed077b8cd965701a205c17f2b0389fce1
- hYLuwpX6qTSHW4zqip3prQ==
- hxxps://centralbancshares\.com
- hxxps://gariloy\.com
- hxxps://liqui-technik\.com
- job_id_header
- path_adj
- path_noun
- seq_num
- seq_total
- server_public_key
- user_agent

A few strings, mostly command names, are stored as stack strings instead of encrypted strings.

Expiration Date

One of the encrypted strings is a Unix epoch timestamp and is used as an expiration date for the malware. In the analyzed sample, the expiration date was set to “1612963255.0039535” (e.g., Wednesday, February 10, 2021 1:20:55.003 PM GMT) and the malware will not run after this date and time.

Configuration

The C&C URLs are stored as encrypted strings and in the analyzed sample were the following:

- hxxps://centralbancshares\.com
- hxxps://gariloy\.com
- hxxps://liqui-technik\.com

There is also an encrypted string that contains a C&C URL path component used in command requests. In the analyzed sample this component was “about”.

Command and Control

C&C is HTTPS based. The initial beacon is called a “handshake” by the malware and an example looks like Figure 4:

```
GET / HTTP/1.1
Connection: Keep-Alive
Cookie: SID=tQS+oDHQCwkaBsX+LGnzsGXebq2IEBbs86rWDC9Nl4rz2y9X9Z4aD8tAnIkp08FIB7bxeWH0s1Kn5oJyb/Cc7GFZLqME/
IJyfIRq8E6bggZnSSvWQEO8xqv9poEVx/F1rFOY5dedAJ5Fe71mVxchXtr5UnBRGrYDWSYtwf5j5nGSH233VjJ1jYzpXJJz2Y1ikzal51nEH5I=;\r\n
Host: centralbancshares.com

HTTP/1.1 200 OK
Date: Wed, 03 Feb 2021 17:34:15 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 400
Connection: keep-alive
Set-Cookie: __cfduid=d575157fb7337f44dcf0253a0fc69d6131612373655; expires=Fri, 05-Mar-21 17:34:15 GMT; path=/;
domain=.centralbancshares.com; HttpOnly; SameSite=Lax
CF-Cache-Status: DYNAMIC
cf-request-id: 080a8e98a90000281efa213000000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Report-To: {"max_age":604800,"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report?
s=4cmzx87u0jwWwdSMArhQH0In6RUsemLsyqPtHHbpYVVL4i3tkPOSTLI%2ByPB%2B3z%2Fvy4%2BbgxJp38fA9Z9YozbK
%2BZCkzUyujF4M7PPaDdoaNgS0L8BU9U8%3D"}],"group":"cf-nel"}
NEL: {"max_age":604800,"report_to":"cf-nel"}
Server: cloudflare
CF-RAY: 61bde6d44d4c281e-SLC

Nd3gdcvujq2cxjiWTW/3qpM1u702s86joImztBhTzV2cemIJRonlvcBcWyKPiVRSGQenvzEAJXzUqVTfWNgG0AiJPSAGC7xiI
+08ggPjn0w9CEdcQlANhk5Tj6ga9VjTqvkXV7akVgDfuC1P8MWEXBle+W+NfGPuNeyTuEPvDBT0CwBI/yFtf8ztYJXghIn3Inxvv
+faq5gSeRXXuUFXI0ZvhWKiQEE6fn7oVPk6ymSroZOi1XEQUG/HZ3G4Rozh4wdaCEcn+LONtHlrQh6wCy4isXASSz3qpSUTtQPmhy/l/
9eBmL4Kc7kJsnciFdZIJ1d4NaYqoedLj/yGLvWgbAdoj+ftcjlfELuzgyPNKlqMzVVN+n1VKYvDh87k5uXREKCGt+OZHKIT8jjz
```

Figure 4: Example handshake request and response

Encryption

The handshake is used to do a X2551 key exchange with the C&C server and retrieve some configuration items. The “SID” value in the “Cookie” header request is base64 encoded. Once decoded, it contains the malware’s generated public key for the key exchange and some additional data to decrypt the handshake response from the C&C server.

The C&C response can be decrypted by using this data along with some of the encrypted strings from the malware and some primitives from the Monocypher crypto library. A Python snippet using data from the above referenced PCAP and its sample showing this decryption process will be available on our GitHub. Once decrypted the response contains a JSON object that looks like Figure 5:

```
{
    "path_adj": "better-known",
    "path_noun": "disassociation",
    "seq_num": "bleached",
    "seq_total": "packer",
    "user_agent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1)",
    "job_id_header": "csrf-token",
    "server_public_key": "IL+Itub0tn8o+jtgRdtUB1c+fvad9TS/k5NawlHm8CQ="
}
```

Figure 5: Example handshake response JSON object

It contains the following pieces:

- path_adj – C&C URL component used in future C&C communications
- path_noun - C&C URL component used in future C&C communications
- seq_num – unknown (doesn’t seem to be used in the analyzed sample)
- seq_total – used as a JSON field name in command responses to the C&C server (blue highlight in Figure 7 below)
- user_agent – user agent used in future C&C communications
- job_id_header – unknown (doesn’t seem to be used in the analyzed sample)
- server_public_key – the C&C server’s public key used for the key exchange

The key shared between the malware and C&C server via the key exchange will be used for future C&C communications. Unfortunately, we were unable to derive the shared key used in the referenced PCAP to decrypt further communication examples.

Future C&C URLs are constructed using the configuration item mentioned above and the received “path_adj” and “path_noun” components. Here is an example C&C URL for the reference PCAP:

hxxp://liqui-technik\.com/about/disassociation/better-known

Updated “path_adj” and “path_noun” components are sent in successive C&C responses via a response header whose name is the previously set “path_noun”. This mechanism can be seen in the red highlights of Figures 6 and 7 below.

Once the handshake is completed, the remaining C&C communications are command requests and command responses. An example command request looks like Figure 6:

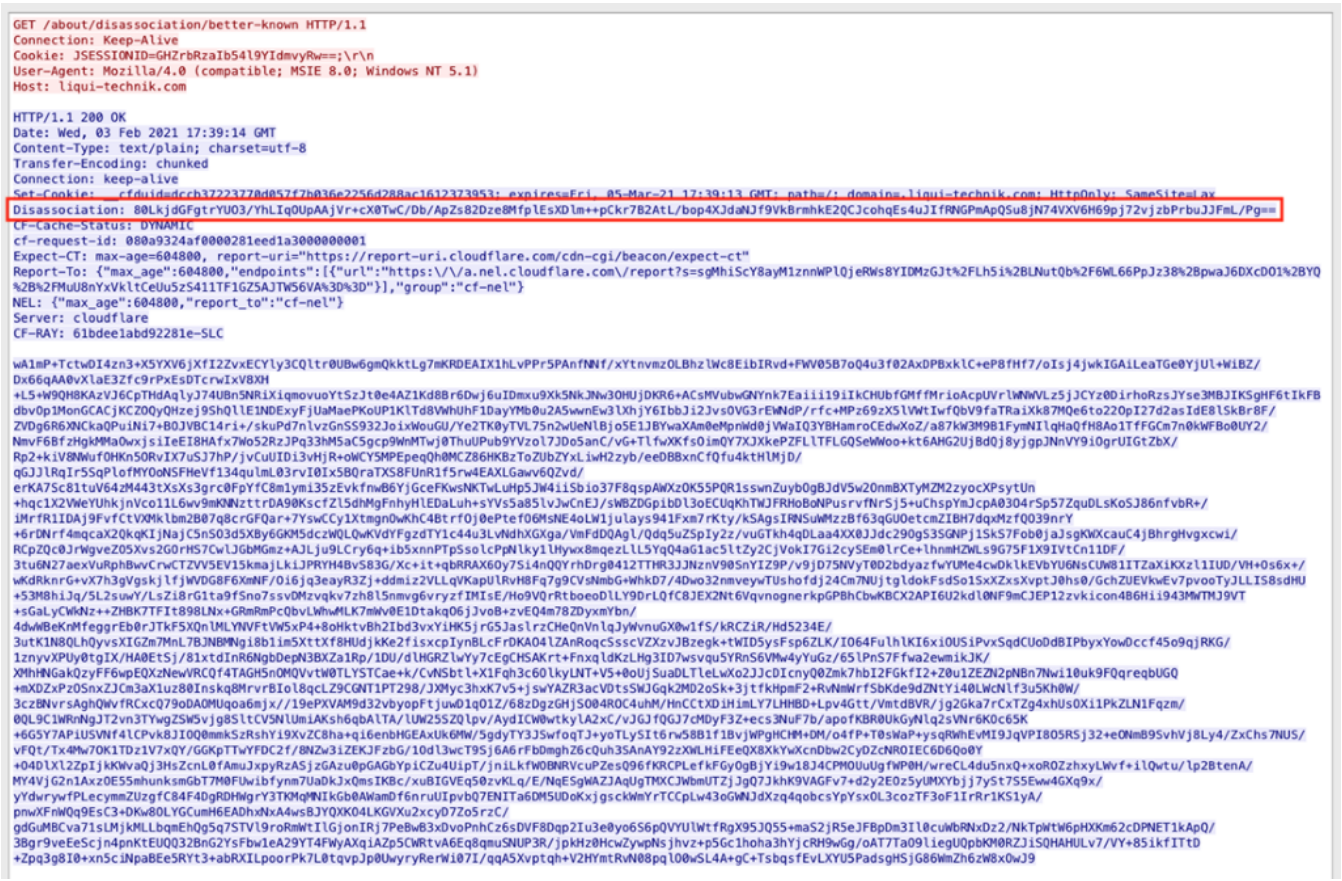


Figure 6: Example command request

In the command request, the Cookie header “SID” value changes to a bot identifier. The response is encrypted using the shared key and once decrypted contains a JSON object. It contains the following fields:

- job_id - identifier
- job – base64 encoded job details

Once decoded, “job” contains another JSON object containing:

- type – command
- args - command arguments

Commands will be detailed in the “Commands” section below.

An example command response looks like:

Figure 7: Example command response

Command responses are similar to requests and contain command outputs or error messages.

Commands

The following commands have been identified in NimzaLoader:

- cmd - execute cmd.exe command
- powershell - execute powershell.exe command
- handshake - redo handshake
- shellcode - inject shellcode into a process as a thread
- command arguments are a JSON object containing:
- sc – hex-encoded and compressed shellcode
- prog - program to inject shellcode into
- heartbeat - used to update expiration date of the malware in memory
- command arguments are a JSON object containing:
- heartbeat - new expiration time
- sig - used in a signature check with an encrypted string ("e8cbd40fda2500cd496b55df43402d8ed077b8cd965701a205c17f2b0389fce1" in the analyzed sample)

At the time of research, all known NimzaLoader C2s were already down, but a public malware sandbox run seems to show it receiving a “powershell” command that ultimately delivered a Cobalt Strike beacon. We are unable to validate or confirm this finding, but it does align with past TA800 tactics, techniques, and procedures (TTPs).

Conclusion

NimzaLoader is a new initial access malware being distributed and used by the TA800 threat actor. In 2020, we observed the shift from TA800 distributing the Trick, with intermittent shifts to Buer Loader, and a consistent distribution of Bazaloader since April 2020. There has been some research community analysis suggesting that NimzaLoader is just another variant of BazaLoader, but based on our observations of significant differences, we are tracking this as a distinct malware family. There has been some evidence suggesting NimzaLoader is being used to download and execute Cobalt Strike as its secondary payload, but it is unclear whether this is its primary purpose. It is also unclear if Nimzaloader is just a blip on the radar for TA800—and the wider threat landscape—or if Nimzaloader will be adopted by other threat actors in the same way BazaLaoder has gained wide adoption. TA800 continues to integrate different tactics into their campaigns, with the latest campaigns delivering Cobalt strike directly.

Indicators of Compromise

Indicator	Type	Notes

540c91d46a1aa2bb306f9cc15b93bdab6c4784047d64b95561cf2759368d3d1d	SHA-256	NimzaLoader Reverse Engineered
centralbancshares\.com	Domain	C&C
gariloy\.com	Domain	C&C
liqui-technik\.com	Domain	C&C
52bbe09c7150ea66269c71bac8d0237fb0e6b0cae4ca63ab19807c310d6a1a0b	SHA-256	NimzaLoader (PCAP)

Emerging Threats Signatures

ETPRO TROJAN NimzaLoader Initial CnC Host Checkin

ETPRO TROJAN NimzaLoader CnC Activity M1

ETPRO TROJAN NimzaLoader CnC Activity M2

Subscribe to the Proofpoint Blog