Threat Actor “Cold River”: Network Traffic Analysis and a Deep Dive on Agent Drable

lastline.com/labsblog/threat-actor-cold-river-network-traffic-analysis-and-a-deep-dive-on-agent-drable/

January 11, 2019

Posted by Quentin Fois and Labs Team ON JAN 11, 2019

Executive Summary

While reviewing some network anomalies, we recently uncovered Cold River, a sophisticated threat actor making malicious use of DNS tunneling for command and control activities. We have been able to decode the raw traffic in command and control, find sophisticated lure documents used in the campaign, connect other previously unknown samples, and associate a number of legitimate organizations whose infrastructure is referenced and used in the campaign.

The campaign targets Middle Eastern organizations largely from the Lebanon and United Arab Emirates, though, Indian and Canadian companies with interests in those Middle Eastern countries are also targeted. There are new TTPs used in this attack – for example Agent_Drable is leveraging the Django python framework for command and control infrastructure, the technical details of which are outlined later in the blog.

We are not sure which threat actor or proxy of a threat actor is behind the campaign. This campaign is using previously undiscovered toolcraft and we speculate that right-to-left languages used has influenced the hardcoded string “Agent_Drable” name into the implant

used in the campaign. It references a 2007 conflict of the Lebanese army at the “Nahr Elbard” Palestinian Refugee camp, which is a transliteration of Nahr el bared. The English translation of Nahr Elbard is “Cold River.”

In short, “Cold River” is a sophisticated threat that utilizes DNS subdomain hijacking, certificate spoofing, and covert tunneled command and control traffic in combination with complex and convincing lure documents and custom implants.

***Note**: the campaign described in this blog post has been also covered by Talos and CERT-OPMD, whereas the underpinning DNS hijacking attacks have been recently described in detail by FireEye in this article.*

MalDoc Droppers

Two malicious word documents were found, differing only in the decoy content (same VBA macro, same payload). The first sample is an empty document but is weaponized (Figure 1).

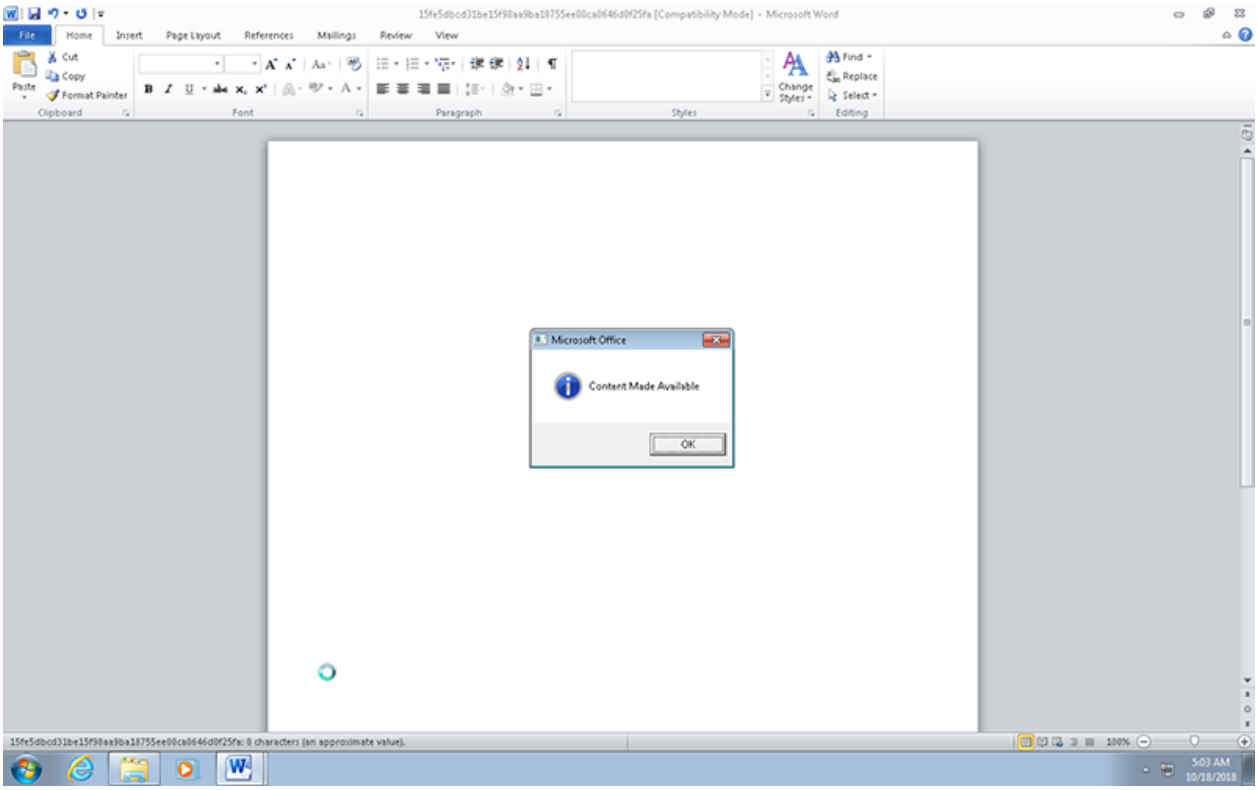


Figure 1: Screenshot of the weaponized empty document, sha1: 1f007ab17b62cca88a5681f02089ab33adc10eec

The second one is a legitimate HR document from the SUNCOR company to which they added the malicious payload and VBA macro (Figure 2).

Figure 2: Screenshot of the HR document from Suncor, sha1: 9ea865e000e3e15cec15efc466801bb181ba40a1

While gathering open intelligence about the callback domain `0ffice36o[.]com` we found a reference to a potential linked document from Twitter (see Figure 3); although that document did not contain the same payload. The person behind this Twitter account may have attached the wrong document.

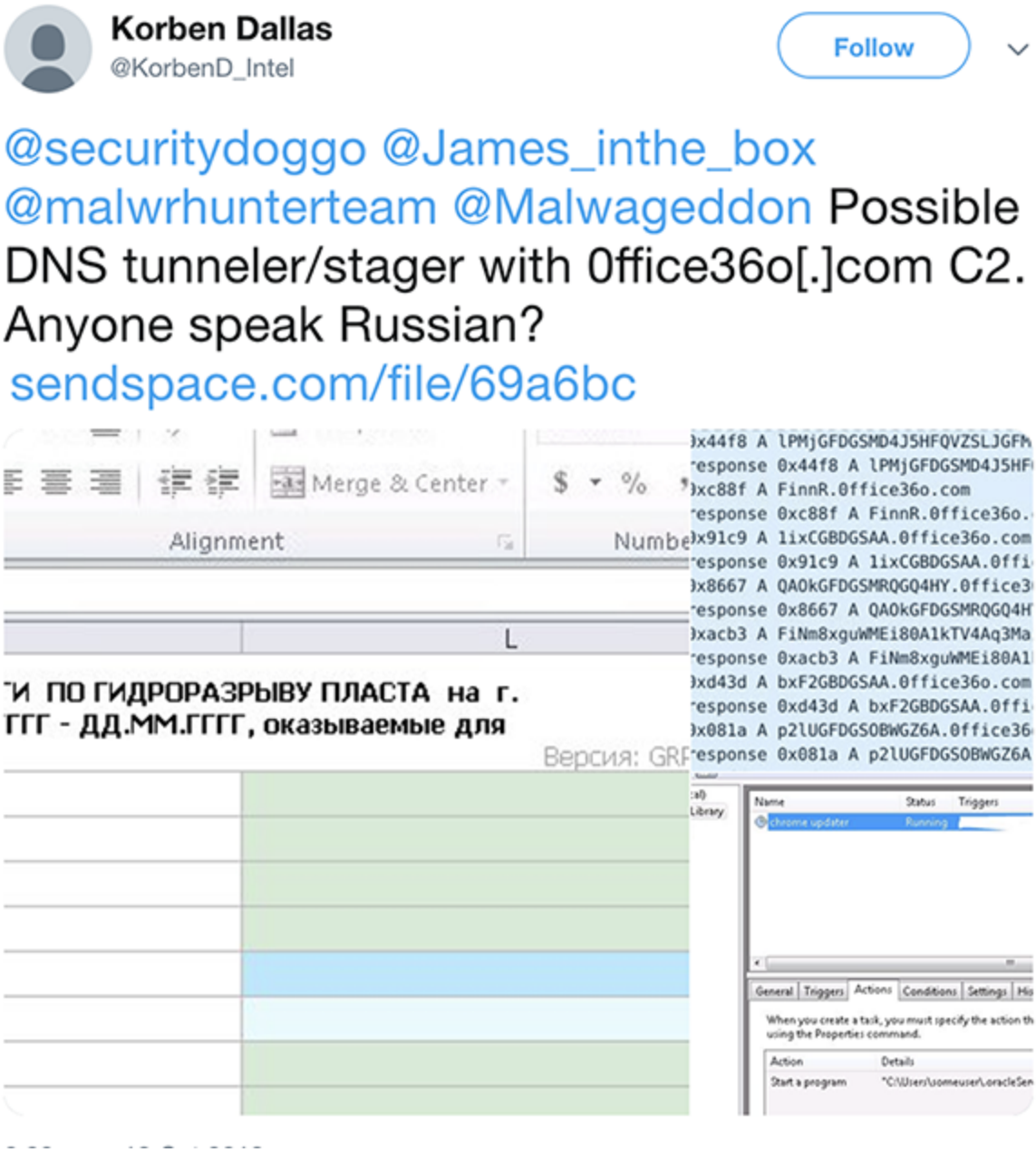


Figure 3: Tweet referencing a third document: https://twitter.com/KorbenD_Intel/status/1053037793012781061

The timestamps listed in Table 1 tend to confirm the hypothesis that the Suncor document is a legitimate document which was weaponized: the creation date is old enough, and the last save matches the timeframe of the campaign. The empty document is most likely the one used to test the macro or to deliver the payload in an environment not related to Suncor.

SHA1	Description	Creation Time	Last Saved Time
`1f007ab17b62cca88a5681f02089ab33adc10eec`	Empty doc	2018-10-05 07:10:00	2018-10-15 02:59:00

`9ea865e000e3e15cec15efc466801bb181ba40a1`	Suncor decoy	2012-06-07 18:25:00	2018-10-15 22:22:00

Table 1: Malicious documents and related metadata.

For a more global timeline of the document and their payload, please refer to Figure 4.

Behavior Analysis

Regarding the VBA macro, it stays basic but efficient. The macro is split into two components, one executing when the document is opened and the other at document close. The actual payload is not stored directly into the VBA code, but instead hidden in a form within the document.

When opening the Suncor document, macro execution must be enabled by the user to actually see its content. This makes the macro activation part appear legitimate to an average user. The only additional obfuscation taking place is the use of string concatenation, such as " `t` " & " `mp` ", " `Microsoft.XML` " & " `DOM` ", " `userp` " & " `rofile` ", etc.

The malicious macro contains some basic anti-sandboxing code, checking to see if a mouse is available on the computer using the API `Application.MouseAvailable`. Overall, the logic of the macro is the following:

At document opening:

- Check if `Environ("userprofile")\.oracleServices\svshost_serv.exe` exists.
- If yes, stop. If no, continue.
- Create the directory `Environ("userprofile")\.oracleServices` if it does not exist.
- Fetch the base64 encoded payload stored in `UserForm1.Label1.Caption`.
- Decode and write it into `Environ("userprofile")\.oracleServices\svshost_serv.doc`.
- Reveal the document content.

At document close:

- Rename the dropped " `svshost_serv.doc` " file as " `svshost_serv.exe` ".
- Create a scheduled task that runs the EXE file every minute, named " `chrome updater` ".

A last interesting thing to note is that the part of the code setting up the scheduled task is copied from an online resource[1].

Payloads and CnC Communication

We found two related payloads, shown in Table 2. The main difference between the two payloads is that one of them has some event logging capabilities, making it easier to determine the actual intention of the implant; most likely it was an early development or debug version. The sample actually packaged inside the Suncor documents was stripped of this functionality.

SHA1	Description	Compilation Timestamp
`1c1fbda6ffc4d19be63a630bd2483f3d2f7aa1f5`	Payload with logs information	2018-09-03 16:57:26 UTC
`1022620da25db2497dc237adedb53755e6b859e3`	Payload without logs information	2018-09-15 02:31:15 UTC

Table 2: the Agent_Drable payloads.

One interesting string found inside the binary is " `AgentDrable.exe` ". This name is written in the DLL Name entry of the Export directory inside the PE header. It will reappear in different parts of this campaign, such as the infrastructure configuration. We can assume with confidence that this is the name given to this implant by the threat actor. There is very little evidence referencing AgentDrable outside of recent submissions to a few analysis portals. One hypothesis is that it would be the name "Elbard" reversed.

Compilation timestamps of the two samples are interesting as well. One has to be fully aware that timestamps can easily be falsified, however, these can be found in multiple places across the binaries (Debug directory, File header) and are coherent with the other events of the campaign. We placed all the dropper and payloads timestamps in Figure 4.

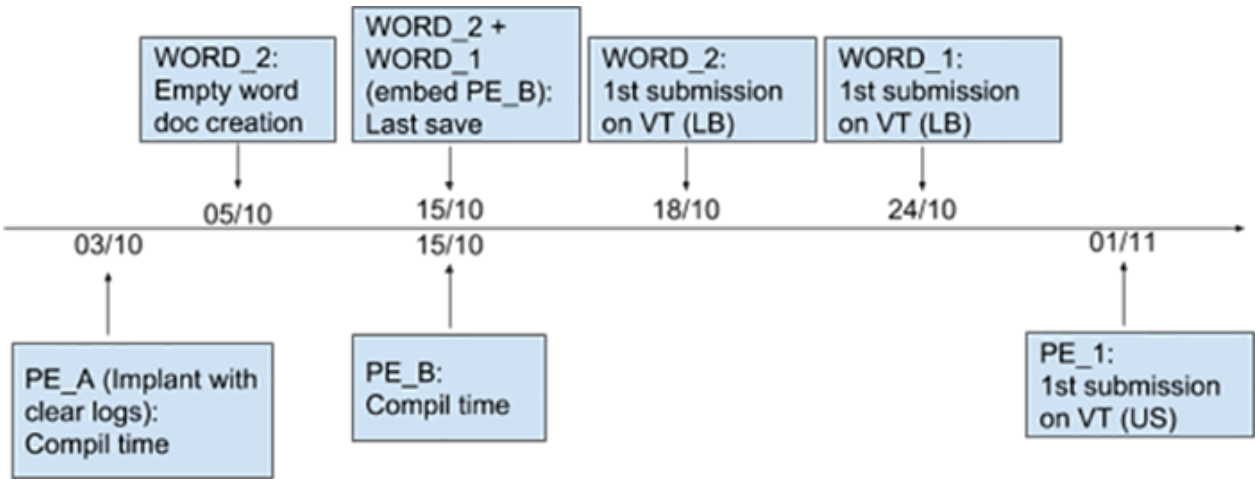


Figure 4: Note that the creation timestamp of WORD_1 is omitted, being way further back in time (2012).

One interesting fact is the compilation timestamp of the dropped sample without logs, which matches the last save time of the two word documents in which the dropped file was embedded. Meaning that they likely compiled the last version of their implant and directly weaponized the document for delivery.

Both malicious documents were submitted to VirusTotal from Lebanon just a few days later. Overall this timeline provides a coherent story and suggests that none of the timestamps were altered by the attackers. This completes the overview of the campaign deployment; we will provide additional insight as we compare this with the evolution of the attackers command and control infrastructure.

Dropped Executable – Behavior Analysis

The primary function of the dropped payload is to operate as a reconnaissance tool. There are no advanced functionalities implemented inside the binary (no screen capture or keylogger, for example). This file's main functions are:

- Running commands from CnC and returning the output
- File download and execution
- File exfiltration

One IP and one domain name are hardcoded inside the binary, as well as a user agent:

- `0ffice36o[.]com` (clearly mimicking the legitimate `office360[.]com`)
- `185.161.211[.]72`
- `Mozilla/5.0 (Windows NT 6.1; Trident/7.0;` rv `:11.0) like Gecko`

The implant has two main ways of communicating with a CnC: (1) DNS requests, and (2) HTTP(S) GET/POST. The first execution defaults to DNS communication then, based on the commands received, it may switch to HTTP.

The binary is executed every minute thanks to a scheduled task created by the malicious document. At the beginning of every run, it creates the following subdirectories if they do not exist:

- `.\Apps`
- `.\Uploads`
- `.\Downloads`

Directories "`Uploads`" and "`Downloads`" act exactly according to their name. Any executable located in the "`Apps`" directory will run each time the dropper implant is executed.

All the configuration data is handled using JSON and the cJSON library. Key names are generic, using one or two letters ('a', 'm', 'ul', …) but we managed to get a comprehensive listing as shown in Table 3. The configuration is stored in the file "`Configure.txt`" and

retrieved at the beginning of every execution.

Parameter Name	Comment
`a`	`Execution mode (DNS/HTTP)`
`m`	`Max query length, used to split long DNS queries into multiple shorter ones`
`f`	`Phase`
`c`	`DNS counter`
`h`	`Home path, where the subdirectories and config file are created`
`u`	`HTTP CnC resource path`
`s`	`HTTP CnC IP address`
`d`	`DNS CnC domain`
`p`	`HTTP CnC port number`
`l`	`Connection type, HTTP or HTTPS`
`i`	`Victim ID (2 chars)`
`k`	`Custom base64 alphabet`

Table 3: JSON configuration parameters (list not exhaustive).

In order to communicate with the DNS CnC, the sample performs DNS queries of specially crafted subdomains. For example, here are some DNS queries from different victims:

- `crzugfdhsmrqgq4hy000.0ffice36o[.]com`
- `gyc3gfmhomrqgq4hy.0ffice36o[.]com`
- `svg4gf2ugmrqgq4hy.0ffice36o[.]com`
- `Hnahgfmg4mrqgq4hy.0ffice36o[.]com`
- `6ghzGF2UGMD4JI2VOR2TGVKEUTKF.0ffice36o[.]com`

The subdomains follow a specific schema: they are made of 4 random alphadecimal chars and a base32 encoded payload. When applied to the domains listed above we get:

Subdomain	Plain text
`crzugfdhsmrqgq4hy000`	`1Fy2048|`
`gyc3gfmhomrqgq4hy`	`1Xw2048|`

`svg4gf2ugmrqgq4hy`	`1uC2048|`
`6ghzGF2UGMD4JI2VOR2TGVKEUTKF`	`1uC0|J5WGS5TJME`

These three first plain texts differ only by two letters: Fy / Xw / uC. It is an ID generated by the sample, that allows the CnC to identify the source of a request. It is generated from the username and/or hostname, and thus stays consistent between implant executions. The same ID is used during HTTP communications.

While in DNS mode, the implant communicates with the CnC exclusively through these crafted subdomains and gets its command by interpreting the IP addresses returned. The HTTP communication mode is a bit more advanced: requests and answers from the implant respectively use GET and POST methods. By default, the sample builds the URL http `://[CNC_IP]/[RESOURCE_PATH]?id=[ID]` where:

Parameter	Default Value	Note
`CNC_IP`	`185.161.211[.]72`	This IP can be updated
`RESOURCE_PATH`	`/index.html`	This path can be updated
`ID`	`Fy`	This ID is constant for a given infection

The hardcoded CnC IP stored in the binaries was offline at the time of the analysis. We were able to find another active CnC hosted at `185.20.184[.]138`. Figure 5 shows what the page looks like when accessed through a web browser.

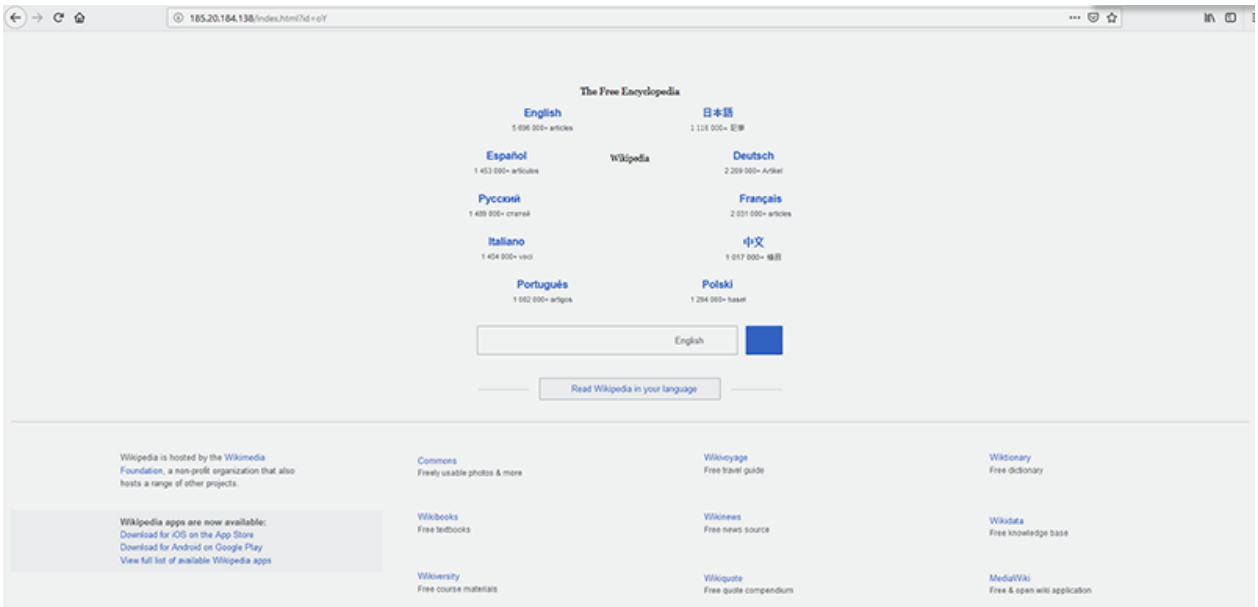


Figure 5: The fake Wikipedia page.

The CnC commands are hidden inside HTML comments or within specific tags and encoded using a custom base64 alphabet. Below is an excerpt of the source code of the page, showing the encoded data.

```
<head>
        <!--eyJ0IjogLTEsICJpIjogIi00MDAwIiwgImsiOiAwLCAiYyI6ICJlY2hvICV1c2VybmFtZSUifQ==-->
        <!--eyJ0IjogLTEsICJpIjogIi01MDAwIiwgImsiOiAwLCAiYyI6ICJob3N0bmFtZSJ9-->
        <!--
eyJ0IjogLTEsICJpIjogIi02MDAwIiwgImsiOiAwLCAiYyI6ICJzeXN0ZW1pbmZvIHwgZmluZHN0ciAvQiAvQzpcIkRvbWFpblwiIn0
->
    <meta charset="utf-8">
    <title>Wikipedia</title>
```

Once decoded, they give the following JSON object from which the commands are extracted:

```
{"t": -1, "i": "-4000", "k": 0, "c": "echo %username%"}
{"t": -1, "i": "-5000", "k": 0, "c": "hostname"}
{"t": -1, "i": "-6000", "k": 0, "c": "systeminfo | findstr /B /C:\"Domain\""}
```

These commands show the typical steps that an attacker would take to perform host reconnaissance before proceeding with the intrusion. The full list of tags containing instructions or commands is in Table 4.

Tag	Description
`<!--[DATA]-->`	Base64 encoded JSON content
`<link href="[DATA]">`	Resource path from which a download must occur
`<form action="[DATA]"`	Resource path on which the POST answers should be performed
`<style>/* [DATA]*/</style>`	
`<script>/* [DATA]*/</script>`	

Table 4: List of the tags that are extracted from the page.

The HTTP CnC is powered by a Django framework with debug mode activated. Thanks to that misconfiguration, it is possible to gather some additional pieces of information that can be used to map their whole infrastructure. Table 5 lists all the endpoints available.

Path	Description
`/index.html (GET)`	Retrieves commands and generic conf params

`/Client/Login (GET)`	Retrieves the custom b64 alphabet used to encode data
`/Client/Upload (POST)`	Upload exfiltrated data or command results
`/Client/Download/<str:url>`	
`/DnsClient/Register`	
`/DnsClient/GetCommand`	
`/DnsClient/SendResult`	
`/DnsClient/SendNotification`	
`/static/`	
`^\.well\-known\/acme\-challenge\/(?P<path>.*)$`	Used to generate let's encrypt certificates

Table 5: List of all available endpoints.

Besides all the resource paths, the debug mode leaked all of the environment variables and some Django internal settings. The most interesting values are listed in Tables 6 and 7 (the full list is available upon request):

Var Name	Value	Comment
`PWD`	`/root/relayHttps`	Interesting directory name
`PATH_INFO`	`/static/backup.zip`	Password protected backup of the database
`SERVER_NAME`	`debian`	
`SERVER_SOFTWARE`	`WSGIServer/0.2`	
`SHELL`	`/usr/bin/zsh`	
`SSH_CLIENT`	`194.9.177[.]22 53190 22`	Leaked IP of their VPN server

Table 6: Environment variables leaked due to a misconfigured Django instance.

Var Name	Value	Comment
`LOGIN_URL`	`/accounts/login/`	
`MAGIC_WORD`	`microsoft`	Unknown
`PANEL_PATH`	`/Th!sIsP@NeL`	

PANEL_PORT	:7070	
PANEL_USER_NAME	admin	
DATABASES	/root/relayHttps/ db .sqlite3	
SERVER_PORT	:8083	
SERVER_URL	https://185.20.184[.]157	Leaked IP, unknown usage

Table 7: Settings leaked due to a misconfigured Django instance.

Once again we can find a mention to the “drable” monicker, this time as part of one of the queries used to fetch data from the underlying database:

```
SELECT COUNT(*) AS "__count" FROM "Client_drable"
WHERE "Client_drable"."relay_id" = %s
```

Infrastructure

Thanks to the data leaked by the CnC and additional passive DNS data, we were able to identify with high confidence, multiple hosts which belong to the campaign infrastructure. One interesting fact is they are all part of the same autonomous system, Serverius N (AS 50673), and hosted by Deltahost. Furthermore, all the domain names were registered through NameSilo.

IP	Description
185.161.211[.]72	Hardcoded HTTP CnC, not used at the time of the analysis.
185.20.187[.]8	Mostly used to generate Let’s Encrypt certificates. Port 443 still answers with memail.mea.com[.]lb. Port 444 has a “GlobalSign” certificate of memail.mea.com[.]lb.
185.20.184[.]138	Live HTTP CnC. Ports 80 and 443 return interesting Django debug info.
185.20.184[.]157	Unknown usage. Basic authentication protected page on port 7070 with https, cert CN is ” kerteros “. Port 8083 hosts a webserver , but only returns a blank page.
185.161.211[.]79	Hosted the HR phishing domains hr-suncor[.]com and hr-wipro[.]com, now redirect to the legitimate website.
194.9.177[.]22	Openconnect VPN used to reach the HTTP CnC.

By correlating these IP addresses with DNS resolutions (See timeline in Appendix A), we identified three domains that were most likely used to deliver the weaponized first stage documents:

- `hr-suncor[.]com`
- `hr-wipro[.]com`
- `files-sender[.]com`

These similar looking domains names match well with the Suncor document template used in the attack. We have not found any specific document linked to Wipro yet. We also found suspicious DNS resolution from government AE and LB domain names pointing towards 185.20.187[.]8 for a short amount of time (~ 1 day each).

By cross-referencing this data with certificate generation records available on https://crt.sh, we conclude that the attackers managed to take over the DNS entries of these domains and generated multiple "Let's encrypt" certificates allowing them to transparently intercept any TLS exchange.

Domain	Certificate	Redirection Dates
`memail.mea.com[.]lb`	`https://crt.sh/?id=923463758`	`2018-11-06`
`webmail.finance.gov[.]lb`	`https://crt.sh/?id=922787406`	`2018-11-06`
`mail.apc.gov[.]ae`	`https://crt.sh/?id=782678542`	`2018-09-23`
`mail.mgov[.]ae`	`https://crt.sh/?id=750443611`	`2018-09-15`
`adpvpn.adpolice.gov[.]ae`	`https://crt.sh/?id=741047630`	`2018-09-12`

Conclusion

In summary, Cold River is a sophisticated threat actor making malicious use of DNS tunneling for command and control activities, compelling lure documents, and previously unknown implants. The campaign targets Middle Eastern organizations largely from the Lebanon and United Arab Emirates, though, Indian and Canadian companies with interests in those Middle Eastern countries may have also been targeted.

Cold River highlights the importance of detection diversity and contextualized threat intelligence. Without correlating Behavioral Intelligence and Network Traffic Analysis, the full scope of Cold River's capabilities would go unseen, exposing victims to additional risk.

Indicators of Compromise

Droppers (maldocs)

`9ea865e000e3e15cec15efc466801bb181ba40a1 (Suncor document)`
`678ea06ebf058f33fffa1237d40b89b47f0e45e1`

Payloads

`1022620da25db2497dc237adedb53755e6b859e3 (Document Payload)`
`1c1fbda6ffc4d19be63a630bd2483f3d2f7aa1f5 (Writes logs)`

IP addresses

`185.161.211[.]72`
`185.20.184[.]138`
`185.20.187[.]8`
`185.20.184[.]15`
`185.161.211[.]79`
`194.9.177[.]22`
`104.148.109[.]193`

Domain names

`0ffice36o[.]com`
`hr-suncor[.]com`
`hr-wipro[.]com`
`files-sender[.]com`
`microsoftonedrive[.]org`

Certificates domain names

`memail.mea.com[.]lb`
`webmail.finance.gov[.]lb`
`mail.mgov[.]ae`
`adpvpn.adpolice.gov[.]ae`
`Mail.apc.gov[.]ae`

Generated certificates

`https://crt.sh/?id=923463758`
`https://crt.sh/?id=922787406`
`https://crt.sh/?id=782678542`
`https://crt.sh/?id=750443611`
`https://crt.sh/?id=741047630`

User agent

`Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko`

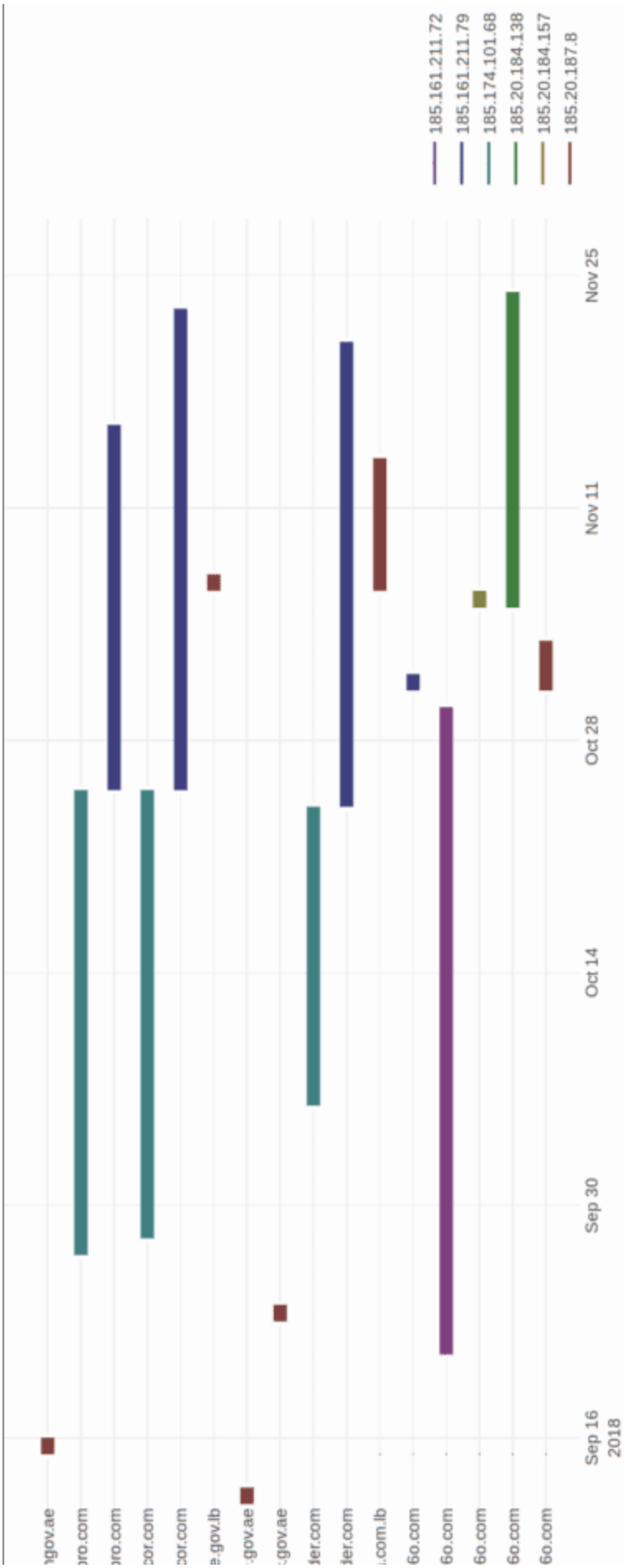
Filesystem artifacts

`%userprofile%\.oracleServices\Apps\`
`%userprofile%\.oracleServices\Configure.txt`
`%userprofile%\.oracleServices\Downloads\`
`%userprofile%\.oracleServices\log.txt`
`%userprofile%\.oracleServices\svshost_serv.doc`
`%userprofile%\.oracleServices\svshost_serv.exe`
`%userprofile%\.oracleServices\Uploads\`

Scheduled task

```
Name: “chrome updater”
Description: “chromium updater v 37.5.0”
Interval: 1 minute
Execution: “%userprofile%\.oracleServices\svshost_serv.exe”
```

Appendix A: DNS Resolution Timeline


185.161.211.72
185.161.211.79
185.174.101.68
185.20.184.138
185.20.184.157
185.20.187.8
Sep 16
2018
Sep 30
Oct 14
Oct 28
Nov 11
Nov 25

Footnotes

[1] https://www.experts-exchange.com/articles/11591/VBScript-and-Task-Scheduler-2-0-Creating-Scheduled-Tasks.html

Quentin Fois

Quentin Fois is a Malware Reverse Engineer at Lastline. A casual CTF player, he also enjoys new technical challenges and deep diving into unknown mechanisms of OS internals. Prior to joining Lastline, Quentin worked at Airbus Cybersecurity as part of their Threat Intelligence team, actively tracking APT groups.

Latest posts by Quentin Fois (see all)

- Threat Research Report: Infostealers and self-compiling droppers set loose by an unusual spam campaign - January 30, 2020
- Reporting from Security Analyst Summit 2019 - April 18, 2019
- Threat Actor “Cold River”: Network Traffic Analysis and a Deep Dive on Agent Drable - January 11, 2019

- About
- Latest Posts

Labs Team

Lastline Labs is where some of the most brilliant minds in the threat prevention community collaborate to develop advanced cyber security solutions. Our research team tracks the evolution, proliferation, and impact of advanced malware. The Lastline Labs Team continually monitors the threat landscape and analyzes new security threats and vulnerabilities.

Latest posts by Labs Team (see all)

- Threat Actor “Cold River”: Network Traffic Analysis and a Deep Dive on Agent Drable - January 11, 2019
- Tales From the Field: The Surge of Agent Tesla - August 28, 2018
- From Russia(?) with Code - March 9, 2018

Tags:

Agent Drable, certificate spoofing, command and control, Django python framework, DNS subdomain hijacking, DNS Tunneling, Nahr Elbard, Quentin Fois