Vicious Panda: The COVID Campaign

By lotemf

Published: 2020-03-12 · Archived: 2026-04-22 02:10:45 UTC

Introduction

Check Point Research discovered a new campaign against the **Mongolian** public sector, which takes advantage of the current Coronavirus scare, in order to deliver a previously unknown malware implant to the target.

A closer look at this campaign allowed us to tie it to other operations which were carried out by the same anonymous group, dating back to at least 2016. Over the years, these operations targeted different sectors in multiple countries, such as **Ukraine**, **Russia**, and **Belarus**.

In this report, we will provide a full analysis of the TTPs utilized throughout this campaign, the infrastructure, and the new tools we uncovered during our research, of what we believe to be a Chinese-based threat actor.

Lure Documents

The investigation started when we identified two suspicious RTF documents sent to the Mongolian public sector. The documents were written in the Mongolian language, with one of them allegedly from the Mongolian Ministry of Foreign Affairs:

Document 1: *Information about the prevalence of new Coronavirus infections*

Document 2: *Purchases for buildings in documentary projects*

These RTF files were weaponized using version `7.x` of a tool named RoyalRoad (aka `8.t`).

This tool, which is commonly used by various Chinese threat actors, allows the attacker to create customized documents with embedded objects that exploit the Equation Editor vulnerabilities of Microsoft Word.

Infection Chain

After the victim opens the specially crafted RTF document, and the Microsoft Word vulnerability is exploited, a file named `intel.wll` is dropped into the Word startup folder: `%APPDATA%\Microsoft\Word\STARTUP`.

This persistence technique is often used by newer versions of the so-called RoyalRoad. Every time that Microsoft Word application is launched, all the DLL files with a `WLL` extension in the Word Startup folder would launch as well, triggering the infection chain we describe below:

Infection Chain Diagram

This not only serves as a persistence technique, but also prevents the infection chain from fully "detonating" if run inside a sandbox, as a relaunch of Microsoft Word is required for the full execution of the malware.

After it's loaded, the malicious `intel.wll` DLL proceeds to download and decrypt the next stage of the infection chain, from one of the threat actor's servers: `95.179.242[.]6`.

The next stage downloaded is also a DLL file, and it serves as the main loader of the malware framework developed by the attackers. It is executed using `Rundll32`, and it communicates with another one of the threat actor's C&C servers (`95.179.242[.]27`) to receive additional functionality.

The threat actor operates the C&C server in a limited daily window, going online only for a few hours each day, making it harder to analyze and gain access to the advanced parts of the infection chain.

At the final stage of the infection chain, after the appropriate command is received, the malicious loader downloads and decrypts a RAT module, also in the form of a DLL file, and loads it into memory. This plug-in like architecture might hint at the existence of other modules, in addition to the payload we received.

The RAT module appears to be a custom and unique malware, though it also includes some rather common core capabilities, listed below:

- Take a screenshot
- List files and directories
- Create and delete directories
- Move and delete files
- Download a file
- Execute a new process
- Get a list of all services

Open Window

At the beginning of our research, one of the attacker's servers, which served the next stage malware, had directory listing enabled for a limited time. This allowed us to download all hosted files, as well as to gain some insight into the operation timeline and the working hours of the attackers.

Open directory at 95.179.242[.]6

Even though they were available for download, all the files on the server came encrypted.

Luckily, by utilizing the same encryption scheme seen in our infection chain, we were able to decrypt most of the files stored on the server.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

key = "VkvX7CK7X7*t$x&hssLR6fOyFSaKrFJKx&@#AK*Fnukj@J9J40f1mKaN$nsCNKPe"

def decrypt(enc,offset):

decrypted = ""

for i in range(len(enc)):

decrypted += chr((ord(enc[i]) ^ ord(key[(i + offset) & 0x3f])))

return decrypted

key = "VkvX7CK7X7*t$x&hssLR6fOyFSaKrFJKx&@#AK*Fnukj@J9J40f1mKaN$nsCNKPe" def decrypt(enc,offset): decrypted = "" for i in range(len(enc)): decrypted += chr((ord(enc[i]) ^ ord(key[(i + offset) & 0x3f]))) return decrypted

```
key = "VkvX7CK7X7*t$x&hssLR6fOyFSaKrFJKx&@#AK*Fnukj@J9J40f1mKaN$nsCNKPe"
def decrypt(enc,offset):
    decrypted = ""
    for i in range(len(enc)):
        decrypted += chr((ord(enc[i]) ^ ord(key[(i + offset) & 0x3f])))
    return decrypted
```

Decryption scheme derived from “intel.wll”

The dozen of files that we were able to decrypt can be divided into four main clusters of malware loader families. Their embedded internal names and core functionality are described below:

http_dll.dll (Intel.wll)	The first stage loader described above. Decrypts the C&C address, then downloads and decrypts the next stage DLL, and executes it via `Rundll32`.
ppdown.dll	Functions as downloader and decryptor for the .rar files stored on the attackers’ server. Reads an `access.txt` file from the server, decrypts it and splits the result into 3 parts: 1) The name of the next stage to download. 2) The next stage export function to call. 3) The decryption key for the next stage.
Rundll32Templete.dll	This variant serves as loader and decryptor for next stage payload. The payload is encrypted in .sect section.
Minisdllpub.dll	The second stage loader, fully described below. Loads additional DLL plugins. A similar version of this payload, called minisdllpublog.dll, contains some additional debug printing capabilities.

Payload types found on the server

Connection to other samples

After gaining access to the additional decrypted files, we were able to hunt for similar samples.

Searching for similar files by the internal names (`http_dll` , `Rundll32Templete` and `minisdllpub`), unique exported functions (`Engdic` , `WSSet` and `MSCheck`) and code similarities (decryption methods, communication patterns, etc.), allowed us to find more samples related to the attacker:

`5560644578a6bcf1ba79f380ca8bdb2f9a4b40b7`	`http_dll.dll`

207477076d069999533e0150be06a20ba74d5378	http_dll.dll
b942e1d1a0b5f0e66da3aa9bbd0fb46b8e16d71d	http_dll.dll
9ef97f90dcdfe123ccb7d9b45e6fa9eceb2446f0	hcc_dll.dll
cf5fb4017483cdf1d5eb659ebc9cd7d19588d935	Rundll32Templete.dll
92de0a807cfb1a332aa0d886a6981e7dee16d621	Rundll32Templete.dll
cde40c325fcf179242831a145fd918ca7288d9dc	minisdllpublog.dll
2426f9db2d962a444391aa3ddf75882faad0b67c	IrmonSvc.dll
9eda00aae384b2f9509fa48945ae820903912a90	IrmonSvc.dll
2e50c075343ab20228a8c0c094722bbff71c4a2a	IrmonSvc.dll
2f80f51188dc9aea697868864d88925d64c26abc	NWCWorkstation.dll

Newly discovered related samples

One of the samples found (`92de0a807cfb1a332aa0d886a6981e7dee16d621`) led us to an article covering a similar initial infection chain, which appears to be after **Ukrainian targets**.

Another sample (`9ef97f90dcdfe123ccb7d9b45e6fa9eceb2446f0`) was originally dropped by an RTF document which appears to be targeting entities in the **Russian Federation**, back in late 2018.

Infrastructure

Analyzing the newly discovered samples introduced us to a larger part of the infrastructure utilized by the threat actor, and a common TTP: All the C&C servers were hosted on **Vultr** servers and the domains were registered via the **GoDaddy** registrar.

Infrastructure overview

As we analyzed this campaign, in addition to the infrastructure used, we also noticed an interesting behavior by the attackers

At a certain point, the C&C server `95.179.242[.]6` stopped serving the open directory listing. A few days later `dw.adyboh[.]com` became an open directory:

Open directory listing at dw.adyboh[.]com

This might indicate that the attackers are enabling directory listing, when one of their payload delivery servers is in active use.

Attribution

From the malicious document perspective, we believe that the naming scheme for `intel.wll` – which is dropped by version `7.x` of `RoyalRoad` is not enough to make a clear cut attribution, as we observed the same name used by various threat actors dropping different malware families such as `Bisonal` and `Poison Ivy`.

From the payload perspective, on the other hand, once we found the additional related samples mentioned in the Hunting section above, we were able to connect it to a known threat group. In the `NWCWorkstation.dll` sample mentioned above, we observed a unique string as part of the logging functionality: "**V09SS0lO"**. This led us to an article from 2017 by Palo Alto Networks, titles **Threat Actors Target Government of Belarus**, which describes an attack that utilizes a RAT named `BYEBY`.

The article itself also connects to a previous article dating back to 2016, where the same tools were used in an attack targeting the Mongolian government. The article also explores the connections between these attacks and previous attacks related to the `Enfal` Trojan.

By comparing the IOCs from the 2017 attack to our campaign we observed several similarities:

Infrastructure Similarities

The servers from the 2017 publication were set on the same infrastructure as all the other samples found during our investigation, and utilize **Vultr** and **GoDaddy** services.

Code Similarities

When analyzing one of the files from the open directory (`bf9ef96b9dc8bdbc6996491d8167a8e1e63283fe`), we noticed that it decrypts and loads a DLL named `wincore.dll`. By investigating this dropped file, we were able to make several correlations to the `BYEBY` sample from 2017:

1. String similarity:

"BYEBY" strings

"wincore.dll" strings

2. Function similarity – Important functions in both `BYEBY` and `wincore.dll` have almost the same implementation. One such function is the payloads' main thread function.

Malware implementation similarities

3. Global Call-Graph and X-Ref Graph – Even though some obfuscation exists in both samples, we were able to verify that they have similar call and reference graphs, meaning that the core functionality of the executables is the same.

Payload – In Depth Analysis

To recap, the second stage payload in the attack chain, is an encrypted DLL file named `minisdllpub.dll`. The DLL, downloaded from `95.179.242[.]6`, is a downloader for an additional payload. In the following section, we go over its implementation and highlight the characteristics which are unique to this payload.

`Minisdllpub.dll` begins by creating a mutex with the name `Afx:DV3ControlHost`. This is a unique indicator that can later be used to hunt for more samples in the wild. It then defines a structure of size `0x5f8` to store system and environment information such as the name of the running computer, IP addresses, the username, and OS Version. Next, another structure of size `0x3FC` is created, this time to store pointers to loaded DLLs and API functions, as well as the command and control IP address (`95.179.242[.]27`) and port (`443`).

After setting up these structures, the flow continues and a new thread is created. First, it fetches several lists of API functions, and dynamically loads them. As can be seen in the following image, each list is comprised of the name of a library followed by a sequence of API functions to load from this library. Pointers to these functions are then added to the previous structure which are used to dynamically invoke them when needed.

Comma-separated lists of API functions, prepended with the library name

The second stage payload then sets up HTTP or HTTPS communication, depends on several checks, and starts communicating with its remote control in new threads. When the server replies, it sends XOR encoded DLL to the malware, with the key `0x51`. `Minisdllpub.dll` then decodes the given payload and dynamically loads the new PE to memory.

When loaded, it searches for an export function with the name e. The malware then keeps listening to commands from the server, and when those are received, it passes them to the `"e"` function of the newly loaded payload. By doing so, the second-stage is operating as a middle-man between the C&C and the final payload – a remote access tool.

The malware is searching for the export function “e”, in order to invoke it

At this point, we have a unique layout of modules loaded on the victim’s computer. First, is the `Minisdllpub.dll` that was initially loaded using `Rundll32` by `http_dll.dll` (`intel.wll`) when a Microsoft Office application was executed. Next, we have the RAT payload itself which receives its control commands not directly from the C&C, but through `Minisdllpub.dll` that acts as a mediator.

Loader execution flow

Interestingly, in addition to the commands to execute, `Minisdllpub.dll` also passes several structures to the final payload. The structures which were previously built and filled, are now used by the RAT to dynamically invoke API functions and deliver data to the C&C server. This unique approach of re-using function pointers that were loaded in the previous module makes the analyzing the RAT hardly possible without having the previous stage as well.

The supported functionalities of the final payload, as well as the respective commands it receives and sends, are described in the table in Appendix A.

Conclusion

In this campaign, we observed the latest iteration of what seems to be a long-running Chinese-based operation against a variety of governments and organizations worldwide. This specific campaign leverages the COVID-19 pandemic to lure victims to trigger the infection chain.

The attackers updated their toolset from documents with macros and older RTF exploits to the latest variation of the "RoyalRoad" RTF exploit-builder observed in the wild.

The full intention of this Chinese APT group is still a mystery, but it is clear they are here to stay and will update their tools and do whatever it takes to attract new victims to their network.

Check Point SandBlast Agent protects against this APT attack, and prevents it from the very first steps.

Appendix A: RAT Module – Supported Commands

Command ID (Sent from C&C)	Sub Command ID (Sent from C&C)	Description	Response ID (Sent from Bot)
0x21		Write a file to a specified path. Set the written file's timestamp to the timestamp of the local kernel32.dll.	0x22
0x23		Get contents of a file.	0x24
0x25		List files in a directory.	0x26
0x2E		Execute command in a new thread.	0x31
0x2F		Execute a command.	0x30
0x32	0x00	Create a directory of by a given path.	0x33
0x32	0x01	Remove a directory in a given path.	0x33
0x32	0x02	Move a file from a given path to a given directory.	0x33
0x32	0x03	Delete a file in a given path.	0x33
0x32	0x04	Move a file from a given path to a given directory. (Same as subcommand 0x02)	0x33
0x34	0x07	Get a list of all the services.	0x35
0x34	0x08	Execute a new process using WinExec.	0x35
0x34	0x09	Execute a new process. (Same as subcommand 0x08)	0x35
0x34	0x0A	Take a screenshot.	0x35
0x34	0x15	Set registry key values.	0x35
0x34	0x16	Download file from URL.	0x3A or 0x3B
0x34	0x17	Download file from URL. (Same as subcommand 0x16)	0x3A or 0x3B

0x34	0x18	Create Pipes and execute a new process.	0x3D or 0x3B
0x34	0x19	Create Pipes and execute a new process (same as 0x18).	0x3D or 0x3B
0x36		Copy the file of the current process with a ".t" extension and modify the registry.	0x37

Appendix B: Files on the server

Internal File Name	**SHA-1**	**Server Location**	**Exp**
http_dll.dll	dde7dd81eb9527b7ef99ebeefa821b11581b98e0	img\0115\WRql7X	Engc
http_dll.dll	fc9c38718e4d2c75a8ba894352fa2b3c9348c3d7	bin\0612wy3\KFuGrS-code	MSC
ppdown.dll	601a08e77ccb83ffcd4a3914286bb00e9b192cd6	bin\0612wy3\KFuGrS	MSC
ppdown.dll	27a029c864bb39910304d7ff2ca1396f22aa32a2	bin\0612wy3\KFuGrS-ppd-bak	MSC
Rundll32Templete.dll	8b121bc5bd9382dfdf1431987a5131576321aefb	img\0115\CYMi0Y-bak img\0115\R7pEFv	WSS
Rundll32Templete.dll (x64)	bf9ef96b9dc8bdbc6996491d8167a8e1e63283fe	bin\test0625\CmlN0i	MSC
minisdllpub.dll	fcf75e7cad45099bf977fe719a8a5fc245bd66b8	img\0115\CYMi0Y img\0120\VIdALQ img\1224\AF9i1i	WSS
minisdllpublog.dll	0bedd80bf62417760d25ce87dea0ce9a084c163c	bin\0612wy3\KFuGrS-www bin\0617wy3\LX5sG1	MSC
gg.dll	5eee7a65ae5b5171bf29c329683aacc7eb99ee0c	bin\0612wy3\TTXk1U.rar	MSC
minisdllpub.dll	3900054580bd4155b4b72ccf7144c6188987cd31	Dropped by 8b121bc5bd9382dfdf1431987a5131576321aefb	WSS
wincore.dll	e7826f5d9a9b08e758224ef34e2212d7a8f1b728	Dropped by bf9ef96b9dc8bdbc6996491d8167a8e1e63283fe	Load

Appendix C: Additional IOCs

Servers:

```
95.179.242[.]6
95.179.242[.]27
199.247.25[.]102
95.179.210[.]61
95.179.156[.]97
dw.adyboh[.]com
wy.adyboh[.]com
feb.kkooppt[.]com
compdate.my03[.]com
jocoly.esvnpe[.]com
bmy.hqoohoa[.]com
bur.vueleslie[.]com
wind.windmilldrops[.]com
```

RTFs:

```
234a10e432e0939820b2f40bf612eda9229db720
751155c42e01837f0b17e3b8615be2a9189c997a
ae042ec91ac661fdc0230bdddaafdc386fb442a3
d7f69f7bd7fc96d842fcac054e8768fd1ecaa88a
dba2fa756263549948fac6935911c3e0d4d1fa1f
```

DLLs:

```
0e0b006e85e905555c90dfc0c00b306bca062e7b
dde7dd81eb9527b7ef99ebeefa821b11581b98e0
fc9c38718e4d2c75a8ba894352fa2b3c9348c3d7
601a08e77ccb83ffcd4a3914286bb00e9b192cd6
27a029c864bb39910304d7ff2ca1396f22aa32a2
8b121bc5bd9382dfdf1431987a5131576321aefb
bf9ef96b9dc8bdbc6996491d8167a8e1e63283fe
fcf75e7cad45099bf977fe719a8a5fc245bd66b8
0bedd80bf62417760d25ce87dea0ce9a084c163c
5eee7a65ae5b5171bf29c329683aacc7eb99ee0c
3900054580bd4155b4b72ccf7144c6188987cd31
e7826f5d9a9b08e758224ef34e2212d7a8f1b728
a93ae61ce57db88be52593fc3f1565a442c34679
5ff9ecc1184c9952a16b9941b311d1a038fcab56
36e302e6751cc1a141d3a243ca19ec74bec9226a
080baf77c96ee71131b8ce4b057c126686c0c696
c945c9f4a56fd1057cac66fbc8b3e021974b1ec6
5560644578a6bcf1ba79f380ca8bdb2f9a4b40b7
207477076d069999533e0150be06a20ba74d5378
b942e1d1a0b5f0e66da3aa9bbd0fb46b8e16d71d
9ef97f90dcdfe123ccb7d9b45e6fa9eceb2446f0
cf5fb4017483cdf1d5eb659ebc9cd7d19588d935
92de0a807cfb1a332aa0d886a6981e7dee16d621
cde40c325fcf179242831a145fd918ca7288d9dc
2426f9db2d962a444391aa3ddf75882faad0b67c
9eda00aae384b2f9509fa48945ae820903912a90
2e50c075343ab20228a8c0c094722bbff71c4a2a
2f80f51188dc9aea697868864d88925d64c26abc
```

RAT:

```
238a1d2be44b684f5fe848081ba4c3e6ff821917
```

Source: https://research.checkpoint.com/2020/vicious-panda-the-covid-campaign/