

# APT34: Jason project

marcoramilli.com/2019/06/06/apt34-jason-project/

View all posts by marcoramilli

June 6, 2019

June 3

لب دوختگان | Lab Dookhtegan | Read My Lips

ما امروز ابزار سایبری دیگری (Jason) که برای هک ایمیل و تخلیه اطلاعات آن توسط وزارت اطلاعات شیر ناپاک خورده استفاده می شود را افشا می کنیم. با سپاس از همکاری بی دریغ هموطنان زجر کشیده.

We are exposing today another cyber tool (Jason) being abused by the bastard Iranian Ministry of Intelligence for hacking emails and stealing information. We thank our suffering compatriots for their cooperation with no hesitation.

👁️ 1983 12:44

Today I want to share a quick analysis on a new leaked APT34 Tool in order to track similarities between APT34 public available toolsets. This time is the APT34 **Jason – Exchange Mail BF** project to be leaked by Lab Dookhtegan on June 3 2019.

June 3

لب دوختگان | Lab Dookhtegan | Read My Lips

| Thread Number | Current Username | Current Password | Status | Email Checked | Login Successful |
|---------------|------------------|------------------|--------|---------------|------------------|
|---------------|------------------|------------------|--------|---------------|------------------|

Original Leak

ما امروز ابزار سایبری دیگری (Jason) که برای هک ایمیل و تخلیه اطلاعات آن توسط وزارت اطلاعات شیر ناپاک خورده استفاده می شود را افشا می کنیم. با سپاس از همکاری بی دریغ هموطنان زجر کشیده.

We are exposing today another cyber tool (Jason) being abused by the bastard Iranian Ministry of Intelligence for hacking emails and stealing information. We thank our suffering compatriots for their cooperation with no hesitation.

👁 1983 12:44

## Context

According to FireEye, APT 34 has been active since 2014. APT 34, also referred to as “OilRig” or Helix Kitten, has been known to target regional corporations and industries. Although there was information about APT34 prior to 2019, a series of leaks on the website Telegram by an individual named “Lab Dookhtegan”, including Jason project, exposed many names and activities of the organization.

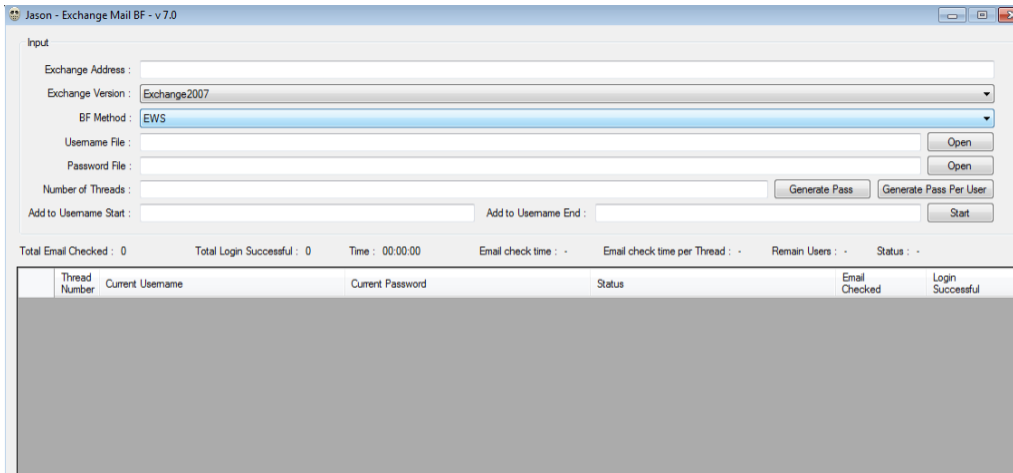
“APT34 conducts cyber espionage on behalf of Iran. Iran seeks to diminish the capabilities of other regional powers to create leverage and better establish itself. This strategy is especially important against nations it sees as a threat to its regional power such as Saudi Arabia and the United Arab Emirates.”

*Michael Lortz*

## Analysis

Jason is a graphic tool implemented to perform Microsoft exchange account brute-force in order to “harvest” the highest possible emails and accounts information. Distributed in a ZIP container (a copy is available [here](#)) the interface is quite intuitive: the Microsoft exchange

address and its version shall be provided (even if in the code a DNS-domain discovery mode function is available). Three brute-force methods could be selected: EWS (Exchange Web Service), OAB (Offline Address Book) or both (All). Username and password list can be selected (included in the distributed ZIP file) and threads number should be provided in order to optimize the attack balance.



Jason Project GUI

Deflating the ZIP container three artifacts are facing out. `Jason.exe` representing the graphic user interface and the main visible tool. `Microsoft.Exchange.WebService.dll` which includes the real functionalities used by `Jason.exe`, it's a Microsoft developed library, `PassSample` which includes some patterns implementation of possible Passwords (ie.[User@first]@[user@first]123) and a folder named `PasswordPatters` which includes building blocks for password guessing. For example it wraps up a file called `Year.txt` including numbers from 1900 to 2020, a file called `numspecial.txt` including special numbers patterns and special chars patterns, a file called `num4.txt` including numbers from 0 to 999 and from 0002 (why not 0001 or 0000?) to 9998 (why not 9999?) and finally a file called `num4special.txt` including special number patters like: 1234,7890,0707, and so on and so forth.

| Name                               | Date modified     | Type                  | Size     |
|------------------------------------|-------------------|-----------------------|----------|
| PasswordPatters                    | 6/4/2019 9:26 AM  | File folder           |          |
| Jason                              | 2/26/2019 2:21 PM | Application           | 47 KB    |
| Microsoft.Exchange.WebServices.dll | 2/26/2019 2:21 PM | Application extens... | 1,104 KB |
| PassSample                         | 2/26/2019 2:21 PM | Text Document         | 2 KB     |

Leaked ZIP content


Digging a little bit into the two Microsoft artifacts we might find out that both of them ( `Jason.exe` and `Microsoft.Exchange.WebService.dll` ) have been written using .NET framework. The used .dll provides a managed interface for developing .NET client applications that use EWS. By using the EWS Managed API, the developer can access almost all the information stored in an Office 365, Exchange Online, or Exchange Server mailbox. The attacker used an old version of `Microsoft.Exchange.WebService.dll` tagged as 15.0.0.0 which according to Microsoft [documentation](#) dates back to 2012.

```
12
13 [assembly: AssemblyVersion("15.0.0.0")]
14 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
15 [assembly: InternalsVisibleTo("Microsoft.Exchange.RpcClientAccess.Handler,
    PublicKey=0024000004800000940000000602000000240000525341310004000001000100b5fc90e7027f67871e773a8fde8938c81d
    d402ba65b9201d60593e96c492651e889cc13f1415ebb53fac1131ae0bd333c5ee6021672d9718ea31a8aebd0da0072f25d87dba6fc9
    0ffd598ed4da35e44c398c454307e8e33b8426143daec9f596836f97c8f74750e5975c64e2189f45def46b2a2b1247adc3652bf5c308
    055da9")]
```

WebService.dll

assembly version

The last available `Microsoft.Exchange.WebServices.dll` dates back to 2015, as shown in the following image, which might suggest a Jason dating period, even if it's not an irrefutable evidence.



## Microsoft.Exchange.WebServices 2.2.0

Microsoft Exchange WebServices

The Exchange Web Services (EWS) Managed API provides a .NET Framework interface to EWS in Exchange Online, Exchange Online as part of Office 365, and versions of Exchange starting with Exchange Server 2007 Service Pack 1 (SP1).

You can use this version of the EWS Managed API to evaluate the library for your application needs, to compare it to directly using XML or the auto generated proxy library, and to create production-ready applications.

Package Manager
.NET CLI
PackageReference
Paket CLI

```
PM> Install-Package Microsoft.Exchange.WebServices -Version 2.2.0
```

### Info

- last updated 15/01/2015
- [Project Site](#)
- [License Info](#)
- [Contact owners](#) Last Microsoft
- [Report](#)
- [Download package](#) (1.28 MB)

### Statistics

↓ 1.642.327 total

Exchange WebServices dll version dates to 2015

Analyzing the reversed byte-code a real eye catcher (at least in my persona point of view) is in the “exception securities” that have been placed. In other words, the developer used many checks such as: variable checks, Nullbytes avoidance, objects indexes and object key checks in order to reduce the probability of not managed software exceptions. These “exception protections” are usually adopted in two main scenarios: (i) the end-user is not a super “techy” guy, so he might end-up with some unexpected conditions or (ii) the attacker is a professional developer who is trained to write product oriented code and not simple working software (which is what attackers usually do). The following images show a couple of code snippets in where the developer decided to protect codes from unexpected user behavior.

```

    }) + "/";
    if (!Uri.TryCreate(this.txtExchangeAddress.Text, UriKind.Absolute, out exchangeUri))
    {
        MessageBox.Show("The entered Exchange Address is incorrect");
        return;
    }
    if (string.IsNullOrEmpty(this.txtUserPassFile.Text))
    {
        MessageBox.Show("Please enter Username File");
        return;
    }
    if (string.IsNullOrEmpty(this.txtPassword.Text))
    {
        MessageBox.Show("Please enter Password File");
        return;
    }
    if (string.IsNullOrEmpty(this.txtThreadCount.Text) || !int.TryParse(this.txtThreadCount.Text, out num))
    {
        MessageBox.Show("Please enter number of threads correctly");
        return;
    }
}

```

Basic exception

prevention 1

```

202     long num = arg_CC_0.Sum(arg_CC_1);
203     this.lblChecked.Text = num.ToString();
204     long num2 = 0L;
205     if (num > 0L)
206     {
207         num2 = (long)(timeSpan.TotalMilliseconds / (double)num);
208     }
209     this.lblEmailTime.Text = num2.ToString() + " ms";
210     this.lblEmailTimeThread.Text = ((int)(num2 * (long)MainConfig.ThreadCount / 1000L)).ToString() + " s";
211     Control arg_17A_0 = this.lblTotalUsers;
212     IEnumerable<UserClass> arg_16C_0 = MainConfig.Usernames;
213     Func<UserClass, bool> arg_16C_1;
214     if ((arg_16C_1 = Form1.<c.>9_14_2) == null)
215     {
216         arg_16C_1 = (Form1.<c.>9_14_2 = new Func<UserClass, bool>(Form1.<c.>9_14_2));
217     }
218     arg_17A_0.Text = arg_16C_0.Count(arg_16C_1).ToString();
219     IEnumerable<ThreadItem> arg_1A4_0 = this.ThreadItemList;
220     Func<ThreadItem, bool> arg_1A4_1;
221     if ((arg_1A4_1 = Form1.<c.>9_14_3) == null)
222     {
223         arg_1A4_1 = (Form1.<c.>9_14_3 = new Func<ThreadItem, bool>(Form1.<c.>9_14_3));
224     }
225     int num3 = arg_1A4_0.Where(arg_1A4_1).Count<ThreadItem>();
226     if (num3 == this.ThreadItemList.Count<ThreadItem>())
227     {
228         this.lblStatus.Text = "Complete";
229         for (int i = 0; i < MainConfig.ThreadCount; i++)
230         {
231             string path = string.Concat(new object[]
232

```

Basic exception

prevention 2

Comparing the code style with my previous analyses on APT34 (OilRig) which you might find [here](#) and [here](#), we might observe a similar code protection. Even if the code language is different the similarity in the basic exception prevention from Jason and -for example- the "ICAP.py script injection" function is very close. Another weak similarity is in the logging style. Jason and -for example- Glimpse project have a similar file logging function which includes string concatenation using special operators (no "flying casting" or "safe conversions", ie: "%s") and one line file logging into function focal points.

I am aware that these are weak similarities and there is no additional evidence or ties with previous leaked APT34 except for the trusted source (Lab Dookhtegan), so I am not giving any personal attribution since it gets very hard to attribute Jason directly to APT34 for what is known.

On the other hand Jason project doesn't share the main source code language with previous APT34 analyses, it doesn't include DNS tricks and or DNS usage evidences, it doesn't include distinguishing patterns or language mistakes, it have been recompiled on January 2019 but using older technology. As already discussed it shares just few code style similarities with Glimpse and WebMask.

IoC

- 9762444b94fa6cc5a25c79c487bbf97e007cb680118afeab0f5643d211fa3f78 (Jason.exe)
- 0cf66c68c265191d36fc9648b4ef879a80be0c3b6da289de5891ede1554de48d (Original ZIP File)

## YARA

---

```
rule _APT34_Jason {
  meta:
    description = "APT34 Jason"
    date = "2019-06-05"
    hash1 = "9762444b94fa6cc5a25c79c487bbf97e007cb680118afeab0f5643d211fa3f78"
  strings:
    $s1 = "System.Resources.ResourceReader, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089#System.Resources.R" ascii
    $s2 = "D:\\Project\\Jason\\obj\\Release\\Jason.pdb" fullword ascii
    $s3 = "Jason.exe" fullword wide
    $s4 = "get_PasswordPattern" fullword ascii
    $s5 = "get_PasswordFile" fullword ascii
    $s6 = "get_pCurrentPassword" fullword ascii
    $s7 = "Microsoft.Exchange.WebServices.Data" fullword ascii
    $s8 = "Total Login Successful :" fullword wide
    $s9 = "Login Successful" fullword wide
    $s10 = "<PasswordPattern>k__BackingField" fullword ascii
    $s11 = "<pCurrentPassword>k__BackingField" fullword ascii
    $s12 = "Jason - Exchange Mail BF - v 7.0" fullword wide
    $s13 = "Please enter Password File" fullword wide
    $s14 = "get_UsernameStart" fullword ascii
    $s15 = "get_UserPassFile" fullword ascii
    $s16 = "get_pCurrentUsername" fullword ascii
    $s17 = "set_pCurrentPassword" fullword ascii
    $s18 = "set_PasswordFile" fullword ascii
    $s19 = "set_PasswordPattern" fullword ascii
    $s20 = "connection was closed" fullword wide
  condition:
    uint16(0) == 0x5a4d and filesize < 100KB and
    8 of them
}
```