Blueliv.

Vawtrak Banking Trojan Technical Report

Raashid Bhat
BlueLiv Labs

Blueliv.

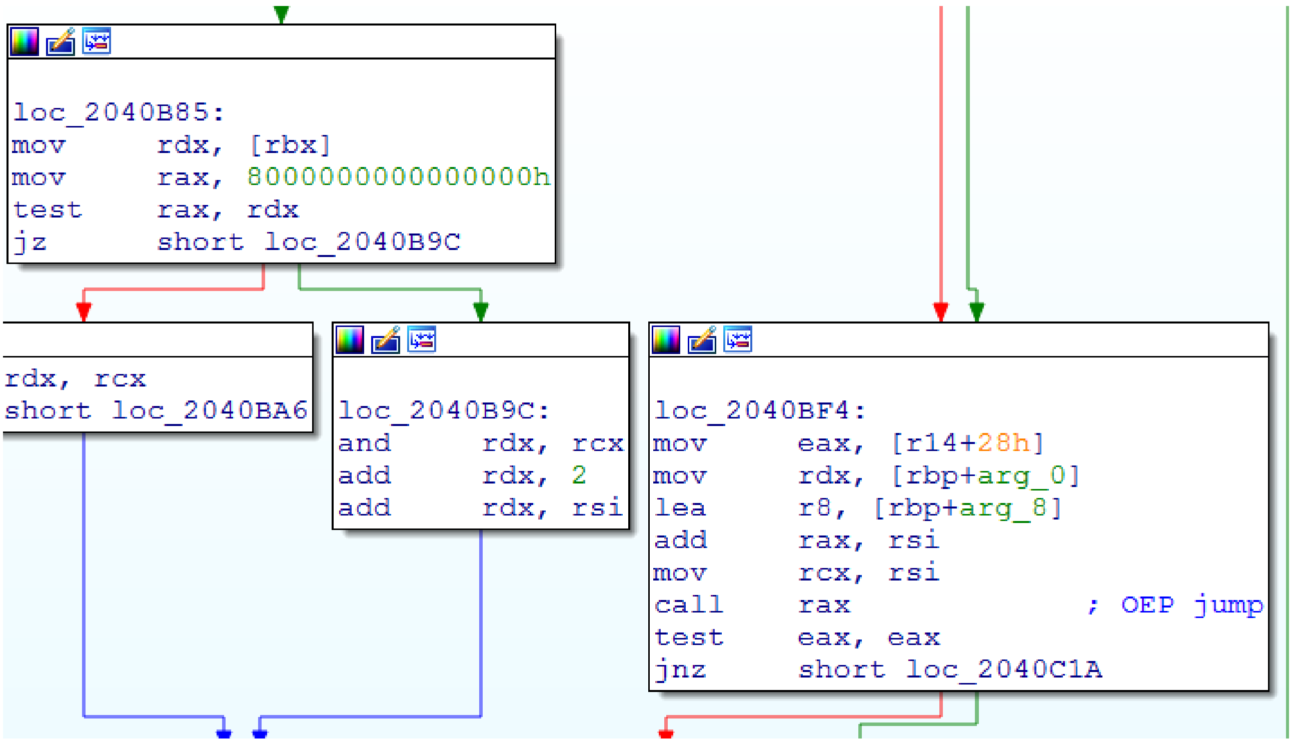
Vawtrak Trojan has been an enduring banking trojan for a long time . It is one of the most prevavalent banking Trojan in the wild today . It also known as neverquest or snifula .

Default packer embeds a resource section which consists an encoded and compressed data buffer. Which consists a static configuration buffer and main binaries (32 bit and 64bit)

DECIMAL	HEXADECIMAL	DESCRIPTION
3970	0xF82	Microsoft portable executable
71554	0x11782	Microsoft portable executable

this data is compressed using LZMAT compression(previously was using APLIB) and is LCG encoded.

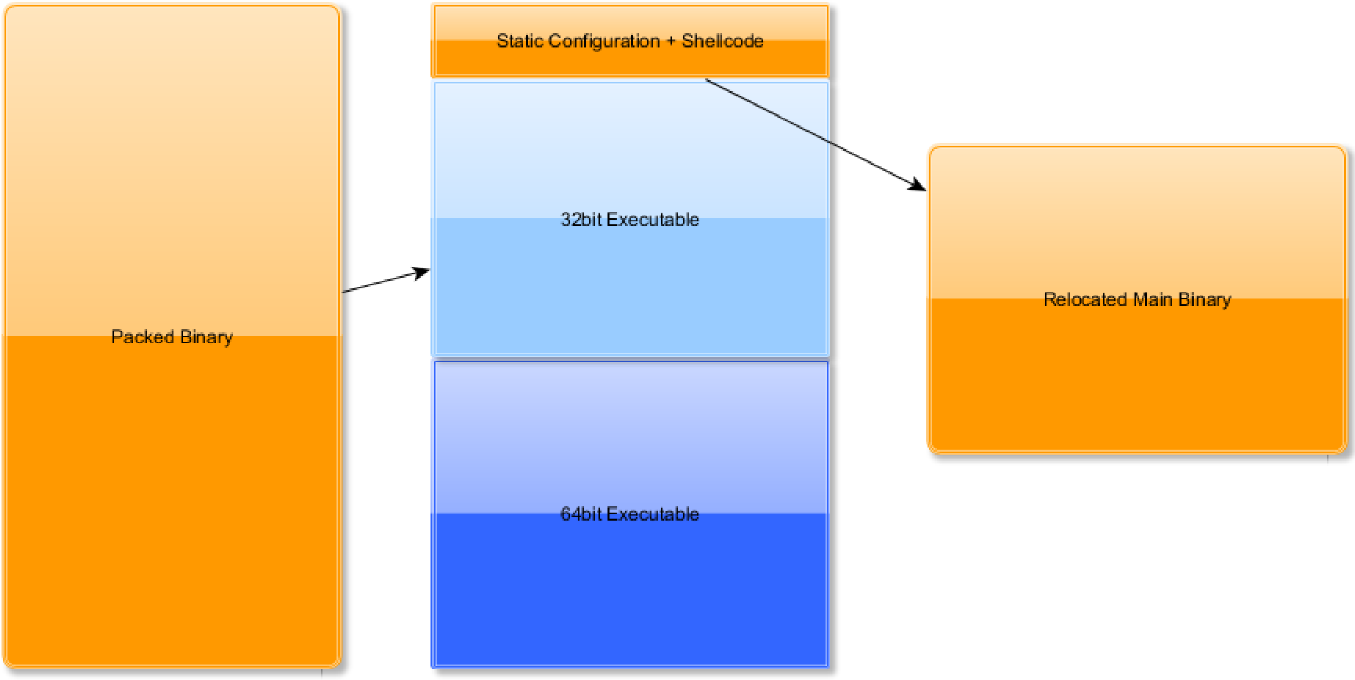
A static configuration buffer consists of important parameters for the main bot to function and an executable code to load the main binary inmemory.



-

This code implements a small loader which relocates one of the main binaries on an allocated executable region and jumps to entrypoint .

Following diagram explains the flow better



Base of this static configuration buffer is passed in as a second parameter to decoded main binary .

Following is a tentative list of items which are present in static configuration

1 : Command and control servers
2 : Botnet Configuration eg ProjectID , UpdateVersion , TorAddress, etc etc
3 : URL resources
4 : Keys to verify module integrity

A simple python script can be used to extract the second stage buffer from the resource object dump

```
import struct
import os
from ctypes import *


_LZMATPATH = os.path.join(os.path.dirname(__file__), './lzmat.so')
print _LZMATPATH
lib = cdll.LoadLibrary(_LZMATPATH)

print _LZMATPATH
def encode(data):
    size = len(data)
```

```
    outlen = (size)+((size+7) >> 3)+0x21
    out = create_string_buffer(outlen+sizeof(c_int()))
    outlen = c_int(outlen)
    ret = lib.lzmat_encode(byref(out), byref(outlen), data, len(data))
    print str(bytearray(out)[:outlen.value]).encode('hex'), outlen
    if ret == 0:
        return out[:outlen.value]
    else:
        raise Exception('Return error: %d' % ret)


def decode(data, size):
    if size:
        outlen = size

    else:
        outlen = len(data)*10000
    out = create_string_buffer(outlen)
    outlen = c_int(outlen)
    ret = lib.lzmat_decode(byref(out), byref(outlen), str(data), len(data))
    if ret == 0:
        return out[:outlen.value]
    else:
        raise Exception('Return error: %d' % ret)

def LCG(seed):
    seed = ((seed * 0x41C64E6D) + 0x3039 ) & 0xFFFFFFFF
    return seed

def LCGDecode(buff, seed, packLen):
    dst = ""

    for p in range(0, packLen):
        seed = LCG(seed)

        dst = dst + chr( ( ord(buff[p]) - (seed & 0xff) ) & 0xff )

    return dst

fp = open("data.mem", "rb")
data = fp.read()

seed = struct.unpack("<L", data[0:4])[0]
print "[] seed = %x" % seed
data = LCGDecode(data[4:], seed, len(data[4:]))
DecompressLen = struct.unpack("<L", data[0:4])[0]

open("final.bin", "wb").write(decode(data[4:], DecompressLen))
```

Blueliv.

Main binary

Vawtrak begins by generating an event name for each running process which is of the following format
{%0.8X-%0.4X-%0.4X-%0.4X-%0.4X%0.8X} in which each of the format specifier are generated by successive pseudo random values generated out from an initial seed. Initial seed in this case is the process ID . This event is used to prevent code injection twice in a same process.

Strings are encoded using the LCG algorithm and can be effectively decoded using an IDA script

```
unsigned int StringDecode(const char *a1, ...)
{
  unsigned int v2; // [sp+0h] [bp-Ch]@1
  int v3; // [sp+4h] [bp-8h]@1
  unsigned int i; // [sp+8h] [bp-4h]@1
  const char *v5; // [sp+14h] [bp+8h]@1
  va_list va; // [sp+18h] [bp+Ch]@1

  va_start(va, a1);
  v3 = *(_DWORD *)a1;
  v2 = (unsigned int)(*(_DWORD *)a1 ^ *((_DWORD *)a1 + 1)) >> 16;
  v5 = a1 + 8;
  for ( i = 0; i < v2; ++i )
  {
    v3 = 1103515245 * v3 + 12345;
    *(_BYTE *)(i + *(int *)va) = v5[i] - v3;
  }
  return v2;
}
```

Depending upon the stage of binary (injected or standalone) a main function of the binary would be either executed as a thread or directly called .

Upon the execution of main thread it again create an event which has a format **OLE%0.8X%0.2X %0.2X%0.8X%0.8X** with initial seed as 13 . This event will be used to trigger down a shutdown event call . Which basically unmaps the code from an executing image

```
call    GenerateRandom
mov     esi, eax
lea     eax, [esp+84h+var_60]
push    eax
push    offset aOle0_8x0_2x0_2 ; "OLE%0.8X%0.2X%0.2X%0.8X%0.8X"
call    StringDecode
push    esi
push    edi
push    0Dh
movzx   eax, bl
push    eax
push    ebp
lea     eax, [esp+0A0h+var_60]
push    eax             ; LPCSTR
lea     eax, [esp+0A4h+Name]
push    eax             ; LPSTR
call    ds:wsprintfA
add     esp, 34h
lea     eax, [esp+74h+Name]
push    eax             ; lpName
push    0               ; bInheritHandle
push    2               ; dwDesiredAccess
call    ds:OpenEventA
```

In the first run it will try to locate and inject into explorer.exe . A byte at an offset 0x0C in static configuration buffer is used as a marker to keep track of the injection process .

Meanwhile if executed inside explorer.exe it starts to inject inside all processes except a blacklist

Vawtrak generates namspaces for multiple objects like filepaths , registry storage and other UID 's which are used for specific purposes thought out the run time of the malware (explained later) .

All these name-spaces are generated using an initial number as a seed to pseudo number generator. Following is a tentative list of name spaces and their corresponding numbers (init Seed)

UID namsspace

#1 :	Registry Run persiatancekey
#3	DebugLog FileMapping
#5	Update Internal Static Config
#6	CommStructEvent
#8	IPC InternalStruct Eventname
#9	IPC InternalStruct FileMapping Name
#10	IPC InternalStruct Mutex Name
#13	OLEID for Shutdown bot event listener
#16	OLEID for installer subroutine event

Registry Path Namespace

#1	Saved Webinject Configuration
#2	Set At restartPC enevnt
#4	BotnetID
#5	Updated StaticConf Reg value
#6	Random Hashed Value
#7	1 (const) Process list delivered

```
call    Generate16byteRandom ; EDX = dest, ECX = SeedMultiplier
lea     eax, [ebp+var_50]
push    eax
push    offset a0_8x0_4x0_4x0_ ; "{%0.8X-%0.4X-%0.4X-%0.4X-%0.4X%0.8X}"
call    StringDecode
push    [ebp+var_4]
movzx   eax, [ebp+var_6]
push    eax
movzx   eax, [ebp+var_8]
push    eax
movzx   eax, [ebp+var_A]
push    eax
movzx   eax, [ebp+var_C]
push    eax
push    [ebp+var_10]
lea     eax, [ebp+var_50]
push    eax             ; LPCSTR
push    esi             ; LPSTR
```

After this step binary starts its initialization phase . It sets up necessary priviledges necessary for the binary and retrieves the VolumerSerialDrive number (which is used to generate folderpath and regitry path). Depending upon the configuration in static buffer it might also setup a UnhandledExceptionFilter to log crash dumps

```
push    offset aDbghelp_dll ; "dbghelp.dll"
call    ds:LoadLibraryA
mov     [ebp+hModule], eax
cmp     [ebp+hModule], 0
jnz     short loc_1000C335
```

```
loc_1000C335:           ; "MiniDumpWriteDump"
push    offset aMinidumpwrited
push    [ebp+hModule]   ; hModule
call    ds:GetProcAddress
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 0
jnz     short loc_1000C350
```

certains parameter which are required globally for the identification of bot are retrieved/generated some of which inlucde

1 : Bot ID
2 : Project ID
3: Update Version
4 : commHttps? (is comunication http or https)
5 : A sign Key to verify downloaded modules and updates

```
mov     ds:BotId, eax
mov     eax, [esi+10h]
mov     ds:ESI_0x10h_ProjectID, eax
mov     ax, [esi+14h]
mov     ds:ESI_0x14h_word_UPDATEVERSION, ax
mov     al, [esi+16h]
mov     ds:Https_ESI_16h?, al
mov     ds:ShellCodeBase, esi
mov     al, [esi+17h]
mov     ds:_shellcode_0x17h, al
lea     eax, [esi+18h]
push    98h                ; len
push    eax                ; source
push    offset BinarySigatureC2 ; dst
call    memcpyX
```

BotID is generated from a combination of data from GetAdaptersInfo() and VolumeDriveSerial and this value is immediately stored in in register storage number #3

```
  if ( GetAdaptersInfo((PIP_ADAPTER_INFO)v1, &SizePointer) )
    HeapFree_0(v1);
  else
    v0 = *((_BYTE *)v1 + 409) | (*((_BYTE *)v1 + 408) << 8) | ((*((_BYTE *)v1 + 407)
                                                              + ((*((_BYTE *)v1 + 407)
                                                               + *((_BYTE *)v1 + 406)
                                                               + *((_BYTE *)v1 + 405)) << 8)) << 16);
  }
}
v2 = GetVolumeSerial() ^ v0;
```

It sets up a pointer to a chunk of data in the static configuration buffer which will be later on use to retrieve c2, URI and other parameters .

If Registry configuration number #5 is if , which consists of an updated configuration buffer retrived from a tor connected web address (explained later) this pointer is rather pointed to that buffer .

A type of process variable is set which basically corresponds to shell if explorer is the running process or 'browser' if it is running in one of the target browsers

Vawtrak also logs its event in a shared filemapping . Most of the important events are logged thought a Shared File Mapping. The name of this file map is generated using namespace number #3. It has been intentionally put in by the author to assist him/her during the developmental stage of this trojan . It provides a function to print a formatted debug log .

```
va_start(va, a2);
if ( DebugFilemapingGUID )
{
  v9 = (LPSTR)HeapAllocate(0x1000u);
  if ( v9 )
  {
    if ( printf? )
    {
      GetLocalTime(&SystemTime);
      v2 = SystemTime.wSecond;
      v3 = SystemTime.wMinute;
      v4 = SystemTime.wHour;
      v5 = GetCurrentProcessId();
      wsprintfA(v9, "PID: %u [%0.2u:%0.2u:%0.2u] ", v5, v4, v3, v2);
      v8 = sub_1000F6EF((int)DebugFilemapingGUID, v9);
```

These debug logs can be captured using the following C Program

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>


void dump(char buf[])
{
        int i = 0;

        for (i = 0; i < 0x10000; i++)
        {
                printf("%c", buf[i]);
        }
}

int main (int argc, char **argv)
{
  HANDLE hMapFile;
  LPCTSTR pBuf;

  hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS,FALSE,argv[1]);

  if (hMapFile == NULL)
  {
    printf("Error opening FileMap");

    return -1;
  }

  pBuf = (LPTSTR) MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS,  0,0,0x10000);
```

```
  dump(pBuf);
  UnmapViewOfFile(pBuf);

  CloseHandle(hMapFile);

  return 0;
}
```

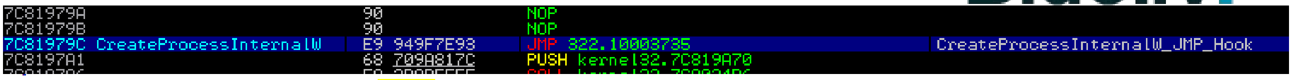
```
PID: 1076 [01:44:15] SHELL START
PID: 1812 [01:44:15] BROWSER START
PID: 1812 [01:45:15] CALL TO=0 FROM=0 CMD=14 PCount=0
PID: 1812 [01:45:15] CALL Status=1 Size=579
PID: 1812 [01:45:15] CALL TO=0 FROM=0 CMD=7 PCount=2
PID: 1812 [01:45:15] CALL Status=1 Size=0
```

Output shows a debug log consisting of some internal calls and their return status

Based on the embedded static configuration vawtrak also sets up a ring-3 rootkit

```
push    offset Trampoline
push    offset CreateProcessInternalHook
push    esi
push    [ebp+var_4]
call    SetAPIHook
add     esp, 10h
mov     dword_100150A8, eax
cmp     eax, 0FFFFFFFFh
jnz     short loc_10003899
```

It basically sets up a hook at CreateProcessInternalW to inject inside any new process launched in the current running executable




```
v10 = AllocMove((int)ShellCodeBase, *(_DWORD *)ShellCodeBase);
v11 = (void *)v10;
if ( v10 )
{
  *(_DWORD *)(v10 + 192) |= 0x10000000u;
  *(_BYTE *)(v10 + 196) = a5;
  Handles = (HANDLE)InjectAndExecute(
                      v6,
                      (LPCVOID)v10,
                      *(_DWORD *)ShellCodeBase,
                      *((_DWORD *)ShellCodeBase + 1),
                      0,
                      0);
```

Most of the major tasks and communication with the command and control server is done though the code injected inside a browser . Some inbuilt commands or the ones received from c2 are supposed to be executed globally by all injected processes.

For this purpose vawtrak sets up a IPC communication mechanism again using a memory mapped File object . It sets up this IPC mechanism . It is internally represented using a structure

struct CommStruct
{
    HANDLE CLSIDEvent; // CLSID ID 6
    *void *Filemap; // 0x04 : SEE FileMap Struct*
    *void *PFilemap_0x16; // : Filemap + 16*
    *void *PFilemap_0x12_Sz1_; // 0x0c: FILEMAP + 0x_Sz1_400h + 0x10*
    DWORD FileMapLowSize; // : Sz1_400h
    DWORD FileMapHighSize; // :Sz2_5000h (if map size is small , data buffer is written to a file)
    HANDLE FnStubThreadhandle; // :Handle for Stub Function
    *void *StubFnPtr; // : Stub Pointer*
    DWORD Const1; // : Function Argument
    DWORD waitTimeoutSeed; // : Seed to be fed to waitForSingleObject Timeout() 0 INFINITE
    DWORD Const3 ;
};

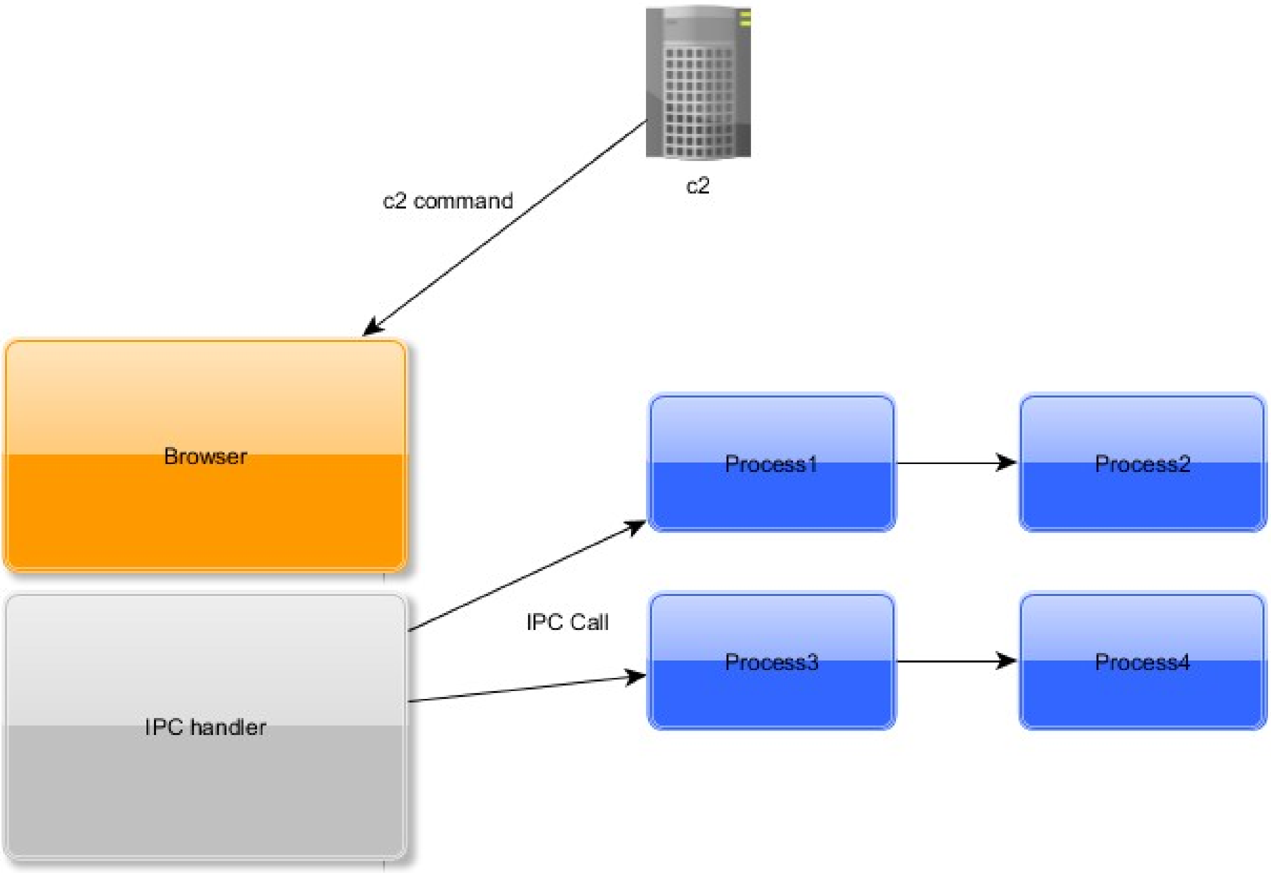
A memory mapped file object is used to pass on the input data though an IPC call and an event which is used to trigger an IPC call

IPC call handler has the following definition

int __stdcall IPCHandler(unsigned __int16 ProcID, int Const, const void *buff, DWORD bufflen)

ProcID : *Process ID of caller*
Const : *Represents type of call to be performed*
buff : *Input Buffer*
bufflen : *input length*

for example when a bot recieves a PluginUnmap command from c2 , it has to unmap from all injected processes .Though an IPC call it is instructed to perform so .




```
def VawtrakCrc32(buff):
   initMask = 0xffffffff
   for i in range(0, len(buff)):
      bt = ord(buff[i])
      for j in range(0, 8):
         if (bt ^ initMask) & 1:
            initMask = ((initMask >> 1) & 0xffffffff) ^ 0xEDB88320
         else:
```

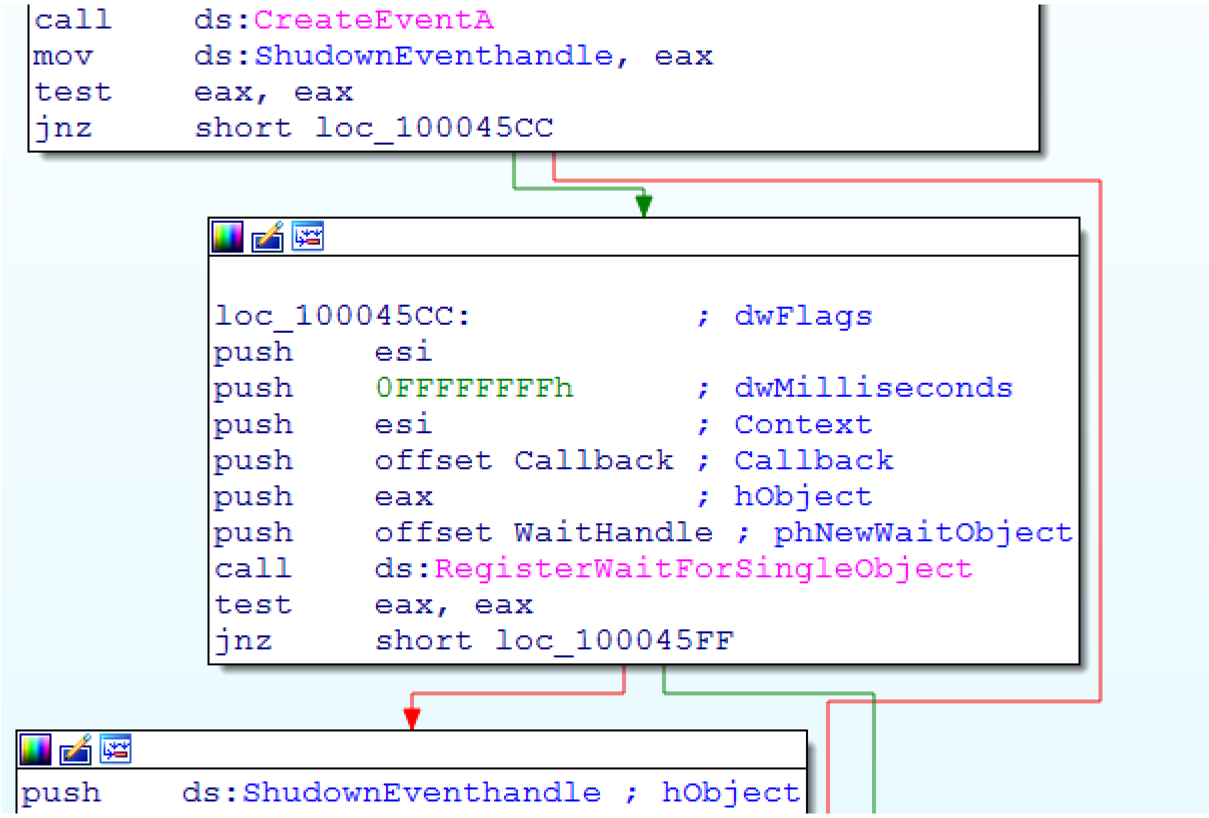
```
        initMask = (initMask >> 1 ) & 0xffffffff
      bt = (bt >> 1) & 0xff
  return initMask
```

Saved Web injects are encoded and compressed using the save LCG encoding and LZMAT compression .

First 4 bytes are used a seed and next 4 bytes represent actual length of the compressed web inject file

```
push    ebp
push    esi
push    edi                 ; dst
push    ebx                 ; source
call    LCgDecodeSavedregconfig
mov     ebp, [esp+1Ch+dst]
lea     ecx, [esp+1Ch+verifycrc32]
push    ecx                 ; DestLen
push    ebp                 ; Dest
push    eax                 ; Len
push    edi                 ; Source
call    LZMatDecodePayload
mov     esi, [esp+2Ch+arg_4]
push    edi                 ; lpMem
mov     [esi], eax
call    HeapFree_0
add     esp, 1Ch
cmp     dword ptr [esi], 0
jz      short loc_100033B6
```

Another namspace of value of #13 is generated to set up an shutdown event . If this event is signaled vawtrak unmaps the binary from the target image .



If the executing process is explorer.exe then an installer thread is created which copies the file to s specified path and sets up the registry run keys. And if the executing process is a browser then it calls a subroutine which is responsible for communicating with the command and control center

This function looks for IBM rapport which is used for protection against MITM attacks in browsers and if found it disables the functionality of it by suspending the thread of rapport module

image > rapport

vawtrak implements a structure for heap management and certain functions are provided for manipulating it

Structure is represented as

struct MemStruct
{
*void *Memory;*
CRITIALSECTION Crti;
DWORD roundedtotalsize;
DWORD datasize;
DWORD Const; // (block section?)
DWORD pagesize; // = 0x1000
}

Following structure is prepared which is later fed to another subroutine and some parameters are generated from it

```
struct InfoEntry
{
        BYTE TypeID;
        WORD Len;
        BYTE Data[Len];
};

struct
{
        struct botInfo
        {
                DWORD botID;
                DWORD projectID;
                WORD  updateVersion;
                WORD  buildVersion;
                WORD  Const0;
                BYTE  Const0;
                BYTE  isInstalled;
        }

        struct InfoEntry Injectcrc32_rand = {0, 8, [injecthash + randomDword]}
        struct InfoEntry ProxyServer          = {1, strlen(proxyserver), proxyserver}
        struct InfoEntry CompName             = {2, len(CompInfo), CompInfo}
        struct InfoEntry LangGroup            = {3, len(LangGroup), LangGroup}
        struct InfoEntry VersoinInfo    = {4, len(VersoinInfo), VersoinInfo }
        struct InfoEntry InstalledPlugins = {5, len(plugins), [WORD Plugins[i]]}


};
```

botInfo is used to generate a 32byte session ID , which is used as a cookie *phpsessionid* rest of the bytes in the struct are encrypted by rc4 using 4 byte botID.

```
def GenerateSessionID(data):
   chars = "0123456789ABCDEF"
   output = ""
   InitSeed = struct.unpack("<B", data[0:1])[0]
   output = output + chr(InitSeed & 0xff)
   for i in range(1, len(data)):
      output = output + chr( ord(data[i]) ^ (InitSeed & 0xff))
      InitSeed = InitSeed + i
   PhpSess = ""
   #print output.encode("hex")
   for i in range(0 , 16):
      PhpSess = PhpSess +  chars[ord(output[i]) >> 4]
```

PhpSess = PhpSess + chars[ord(output[i]) & 0xf]
print "Session ID generated [] = %s" % PhpSess
return PhpSess

This data is sent to command and control server and based on the configuration setting communication is either http or https .

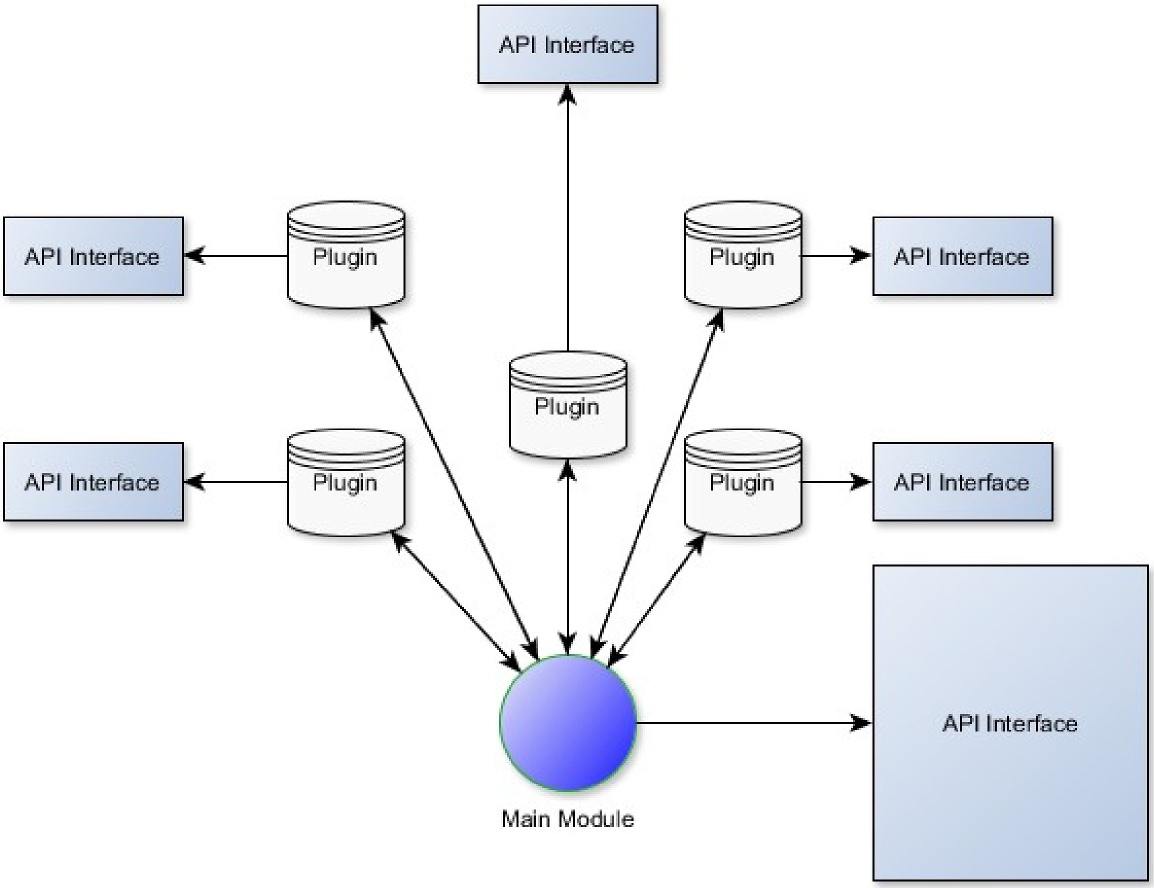
List of c2's are present in encoded form inside static configuration buffer and and retirived using the following subroutine

```
  *a3 = Urlindex;
  *a1 = v5;
  v6 = *(_DWORD *)&v3[0x44 * v5 + 0x206];
  v7 = 0;
  v10 = v6;
  do
  {
    *(_BYTE *)(v7 + Dest) = *((_BYTE *)C2EncodedBuffer + 0x44 * v5 + v7 + 0x20A) ^ generateRandom(&v10);
    ++v7;
  }
  while ( v7 < 0x40 );
}
```

before sending this data to c2 registry configuration key #7 is read and is not found then list of running processes are retrieved in a buffer in format of “%u(%u)\t%s\r\n “ representing processID, parent ProcessID and ImageName . This call is made though the opcode 14 provided by the main module . Vawtrak is a modular trojan it implments a plugin based architecture in which each module is a segregated component of the system including the main module . Each of these modules expose a method to call any API (opcodes) provided by that particular module . That is why we saw in the debug call log some of the function called initially

```
PID: 1076 [01:44:15] SHELL START
PID: 1812 [01:44:15] BROWSER START
PID: 1812 [01:45:15] CALL TO=0 FROM=0 CMD=14 PCount=0
PID: 1812 [01:45:15] CALL Status=1 Size=579
PID: 1812 [01:45:15] CALL TO=0 FROM=0 CMD=7 PCount=2
PID: 1812 [01:45:15] CALL Status=1 Size=0
```

those two functions CMD 14 and CMD 7 are two function to retrive processlist and to send raw data to c2



Main base module implements around **41 Command or API's** . These function can be called from any plugin or even by the request of c2

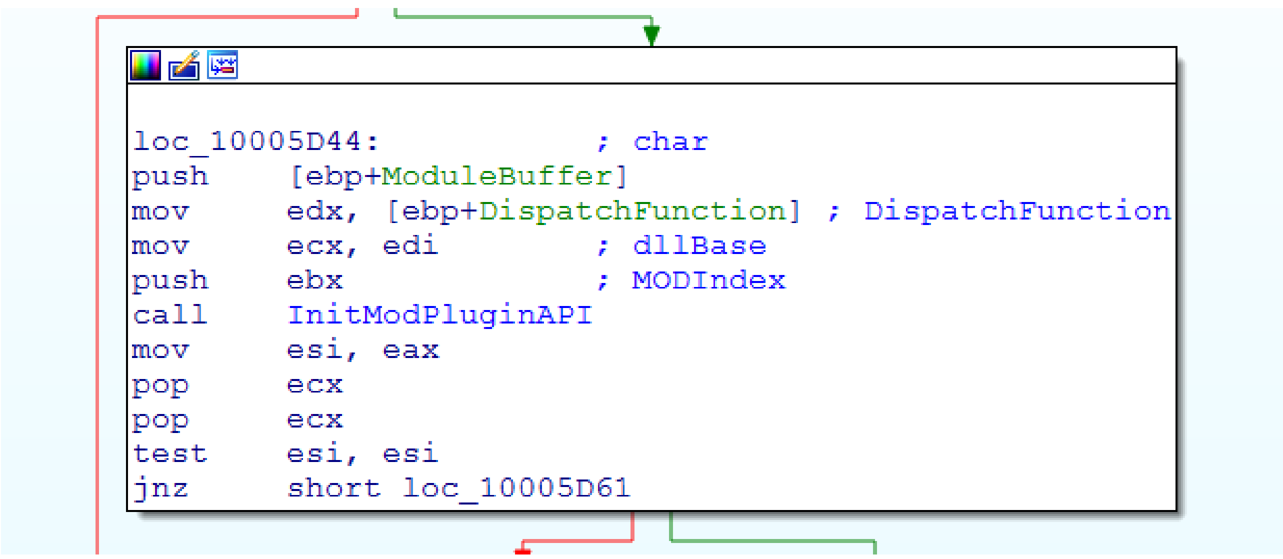
Following table list some of the main functions provided

NullSubroutine
GetsavedConfSectionData
DecodeC2DatapacketWebinject
DownloadAndLoadModules
DeletePluginOp
RetriveModNVerList
InjectBotInProcess
sendData2c2
SendandRecvC2
SetMainC2Eventhandle
RetriveRegconfvalue
SetConfigurationkeyValue
DelConfigurationkeyValue
RemoveAllConfigurationKeys

RetriveProcessList
ExecCommand
DownloadAndExecuteExeFile
RetriveDebugLog
OutpuDebugLog
ResetDebugLog
RestartPC
RestartPCSetConf2key
ShellExecuteExFile
ClearHostsFile
WriteHostsFileUnicode
AppendHostsFile
uploadDnsHostsFilesBuffer
DownloadUpdateAndShutdown
OverRideUserAgent
RetriveFolderPath
RetriveFilePath
RetriveRegPath
RetriveOLECLSID
RetriveRandom16
RetriveCLSID
RetriveShellCodeBase
SetC2TimeWait
send0x4000009Const

Any of these API'S can be called though a Dispatch Handler subroutine which acts as a focal point when there is an call to one of these API's . API's can be called by their position in the API array and module index.

During the initilization phase dispach function and plugin structures are initialized for the main modules and as well as the saved Plugin. A list of modindex's and modversion's array is sent to c2 during initial packet for the c2 to determine which plug-ins are installed on the system



struct basePacket
{
DWORD Size;
BYTE numsections;
struct SectionData
{
BYTE moduleIndex;
BYTE callType;
BYTE NumSec;
// next section
struct _Data{
DWORD lenSection;
BYTE Buffer[lenSection]
}Data[NumSec];
}SectionData[numsections];
}

It contains a header and multiple sections which have own headers specifying to which API and plug-in that particular section is designated for.

Data present in each SectionData entry has its own format , For example when a call opcode 3 == DownloadAndLoadPlugins() is received it has the following hex data

```
0000h: 52 01 00 00 01 00 03 05 3E 00 00 00 68 74 74 70  R.......>...http
0010h: 3A 2F 2F 39 35 2E 32 31 33 2E 31 33 39 2E 31 31  ://95.213.139.11
0020h: 36 2F 6D 6F 64 75 6C 65 2F 32 37 32 61 35 61 64  6/module/272a5ad
0030h: 34 61 31 62 39 37 61 32 61 63 38 37 34 64 36 64  4a1b97a2ac874d6d
0040h: 33 65 35 66 66 66 30 31 64 00 3E 00 00 00 68 74  3e5fff01d.>...ht
0050h: 74 70 3A 2F 2F 39 35 2E 32 31 33 2E 31 33 39 2E  tp://95.213.139.
0060h: 31 31 36 2F 6D 6F 64 75 6C 65 2F 64 31 39 36 37  116/module/d1967
0070h: 63 39 39 63 30 63 37 66 39 62 34 36 38 66 32 65  c99c0c7f9b468f2e
0080h: 30 38 65 35 39 65 34 31 66 66 65 00 3E 00 00 00  08e59e41ffe.>...
0090h: 68 74 74 70 3A 2F 2F 39 35 2E 32 31 33 2E 31 33  http://95.213.13
00A0h: 39 2E 31 31 36 2F 6D 6F 64 75 6C 65 2F 39 36 64  9.116/module/96d
00B0h: 66 31 63 38 34 63 37 66 62 31 33 65 38 38 30 65  f1c84c7fb13e880e
00C0h: 33 39 39 66 39 36 32 37 65 30 64 62 30 00 3E 00  399f9627e0db0.>.
00D0h: 00 00 68 74 74 70 3A 2F 2F 39 35 2E 32 31 33 2E  ..http://95.213.
00E0h: 31 33 39 2E 31 31 36 2F 6D 6F 64 75 6C 65 2F 36  139.116/module/6
00F0h: 66 61 36 38 64 34 34 31 63 38 37 65 36 37 65 36  fa68d441c87e67e6
0100h: 38 31 34 39 35 36 65 30 31 32 38 33 37 36 66 00  814956e0128376f.
0110h: 3E 00 00 00 68 74 74 70 3A 2F 2F 39 35 2E 32 31  >...http://95.21
0120h: 33 2E 31 33 39 2E 31 31 36 2F 6D 6F 64 75 6C 65  3.139.116/module
0130h: 2F 34 32 66 63 65 38 62 35 36 35 35 33 35 35 63  /42fce8b5655355c
0140h: 63 61 61 65 30 64 34 64 38 62 31 30 66 33 63 61  caae0d4d8b10f3ca
0150h: 35 00                                            5.
```

first four bytes represent the total size next bytes represents subsection and call type and mod index

In this case it is

MODINDEX = 0 (main module)

API CALL = 3 (DownloadAndLoadPlugins)

These plugins are encoded and compressed . Plugins are again encoded using LCG with the first 4 bytes of the file as seed. Other information like MODID and BUILDVERSION of plugin is provided for the loader . Plugins are saved locally and path is generated using #MODID namespace.

Following is a tentative list of plug ins which are usually downloaded

Web inject Plugin
A stealer plugin (based on modified version of pony)
Back Connect Module to Tunnel traffic though victims module
Keylogger
Certificate ,history Stealer and A fileManger

input to any plugin takes the following paramaters

```
struct Arg0
{
	BYTE modID;
	const BYTE CallingMod ;
	WORD CallOpcode;
	BYTE numsection;
	struct SecinfoArr[numsection]
	{
		DWORD buffertype; // {1 == integer, 2 == pointer, 0 MOD Number call}
		DWORD SectionLength; { = 0x24}
		DWORD InfostructPtr;
		BYTE compressed? ; //
	}
};

struct InfostructPtr{
	DWORD  SubCallID
	DWORD  BotID;
	DWORD  ProjectID;
	DWORD  VolumeSerialNumber;
	DWORD  CallDispatcherfn;
	DWORD  TLSIndex;
	DWORD  ProcessTypeOPT;
	DWORD  Const1;
	DWORD  ProcessType;
};
```

Usually a webinject section of call type 2 is also supplied .Webinject file of Vawtrak is comprehensive and is similarity to UrlZone inject file . It further consists of encoded subsection which are numerically named which further consist of other sections.

For example web inject section #3 which corresponds to list of URL which are subjected to POST , HTTP AUTH and cookie theft has the following structure

```
struct WebConfigSection
{

	DWORD TotalLen; // including header
	WORD NumOfSubSections;
	DWORD SboxSeed;

	struct SubSection [NumOfSubSection]
	{
		DWORD SectionSize; // including header
		DOWRD SboxSeed;
		WORD SectionIdentifier;
		WORD NumOfEntries;
		struct EntryCharacterictics
		{
			DWORD TotalEntrySize;
			BYTE TypeOfEntry; // Regex , Plain
			BYTE SupportedURLTypes; // Mask {1,2,3}
			BYTE Size; // Excluding Header
		}

		BYTE unknown;
		BYTE BrowserType;
		BYTE UNKNOWN;
		BYTE UNKNOWN;

		struct Sectionvalue{BYTE Size, BYTE Data[Size]};
	};
};
```

```
0070h: 00 00 61 6D 61 7A 6F 6E 61 77 73 2E 63 6F 6D 00  ..amazonaws.com.
0080h: 16 00 00 00 01 03 13 61 16 00 00 00 61 6D 65 72  .......a....amer
0090h: 69 63 61 6E 66 61 6D 69 6C 79 2E 63 6F 6D 00 1D  icanfamily.com..
00A0h: 00 00 00 01 03 0A 61 1D 00 00 00 61 6E 61 6C 79  ......a....analy
00B0h: 74 69 63 73 2E 71 75 65 72 79 2E 79 61 68 6F 6F  tics.query.yahoo
00C0h: 2E 63 6F 6D 00 17 00 00 00 01 03 44 67 17 00 00  .com.......Dg...
00D0h: 00 67 65 6F 2E 71 75 65 72 79 2E 79 61 68 6F 6F  .geo.query.yahoo
00E0h: 2E 63 6F 6D 00 11 00 00 00 01 03 06 61 11 00 00  .com........a...
00F0h: 00 61 6E 73 77 65 72 73 2E 79 61 68 6F 6F 00 13  .answers.yahoo..
0100h: 00 00 00 01 03 64 61 13 00 00 00 61 70 69 2E 6C  .....da....api.l
0110h: 6F 67 69 6E 2E 79 61 68 6F 6F 00 13 00 00 00 01  ogin.yahoo......
0120h: 03 64 61 13 00 00 00 61 70 69 73 2E 67 6F 6F 67  .da....apis.goog
0130h: 6C 65 2E 63 6F 6D 00 14 00 00 00 01 03 51 61 14  le.com.......Qa.
0140h: 00 00 00 61 70 69 2E 76 6B 6F 6E 74 61 6B 74 65  ...api.vkontakte
0150h: 2E 72 75 00 17 00 00 00 01 03 44 61 17 00 00 00  .ru.......Da....
0160h: 61 70 70 6C 69 63 61 74 69 6F 6E 73 74 61 74 2E  applicationstat.
0170h: 63 6F 6D 00 2E 00 00 00 01 03 2B 61 2E 00 00 00  com.......+a....
0180h: 61 70 70 2E 6D 62 67 61 2D 70 6C 61 74 66 6F 72  app.mbga-platfor
0190h: 6D 2E 6A 70 2F 73 6F 63 69 61 6C 2F 61 70 69 2F  m.jp/social/api/
01A0h: 6A 73 6F 6E 72 70 63 2F 76 32 00 24 00 00 00 01  jsonrpc/v2.$....
01B0h: 03 A1 61 24 00 00 00 61 70 70 72 65 70 2E 73 6D  .¡a$...apprep.sm
01C0h: 61 72 74 73 63 72 65 65 6E 2E 6D 69 63 72 6F 73  artscreen.micros
01D0h: 6F 66 74 2E 63 6F 6D 00 1A 00 00 00 01 03 17 61  oft.com........a
```