STAGE 1 INVESTIGATION (http://www.joineset.com/download/EsetCrackme2013.exe)

Analyzing file

My PEID 0.95 tells that `EsetCrackme2013.exe` is packed with UPX, but UPX itself can not unpack this file. I didn't investigate why – it is faster for me to unpack it manually. OEP is `0x40463C`.

At the Original Entry Point

So, we are at the original entry point. First goes a huge sequence of `MOV BYTE PTR DS:[EBX+<I>],<BYTE>` and some `MOV BYTE PTR DS:[ESI+<I>],<BYTE>` commands. Obviously, some strings are generated here. To see them, we need to know where `ESI` and `EBX` point to.

EBX and ESI registers get initialized

```
00404661        MOV EBX, 00409770    ; ebx = &arr1[0]
00404666        MOV ESI, 00409670    ; esi = &arr2[0]
```

We can set breakpoint after the last `MOV` command at `0x407A67` and run debugger. After breakpoint is hit, we should look at memory dump at, for example, `0x409670`.

Memory dump

```
00409670  11 45 73 65 74 20 43 72 61 63 6B 6D 65 20 32 30  #Eset Crackme 20
00409680  31 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00  13..............
```

Actual string data is preceded with `0x11` byte (decimal 17) – the string length. This is an attribute of Pascal strings.

The strings are aligned in memory to `0x100` bound. There are 25 of them. We can make a conclusion that this is a `char array[25][256]`.

Anti-debugging tricks

The first strange call we meet is `OutputDebugString()` with string parameter `"%s%s%s…"`. I've found info that this is a trick to crash OllyDbg 1.10.

Trying to crash OllyDbg

```
00407B9C        PUSH EAX    ; ASCII "%s%s%s…"
00407B9D        CALL <JMP.&kernel32.OutputDebugStringA>
```

Next goes IsDebuggerPresent() call.

IsDebuggerPresent() call via ESI

```
00407C35        PUSH 00407F8C   ; ASCII "IsDebuggerPresent"
00407C3A        PUSH EBX        ; kernel32.dll handle
00407C3B        CALL <JMP.&kernel32.GetProcAddress>
00407C40        MOV EDI,EAX
00407C42        MOV ESI,EDI     ; ESI = IsDebuggerPresent
```

```
00407C44         TEST EDI,EDI
00407C46         JE SHORT 00407C55
00407C48         CALL ESI    ; IsDebuggerPresent())
00407C4A         CMP AL,1
00407C4C         JNZ SHORT 00407C55
00407C4E         PUSH 0
00407C50         CALL <JMP.&kernel32.ExitProcess>
```

And right after `IsDebuggerPresent()` call there goes two checks of `BeingDebugged` field of the PEB structure.

Checking BeingDebugged #1

```
00407C55          MOV EAX,DWORD PTR FS:[18]
00407C5C          MOV EAX,DWORD PTR DS:[EAX+30]
00407C5F          MOVZX EAX,BYTE PTR DS:[EAX+2]    ; BeingDebugged
00407C63          CMP EAX,1
00407C66          JNE SHORT 00407C6F
00407C68          PUSH 0
00407C6A          CALL <JMP.&kernel32.ExitProcess>
```

Checking BeingDebugged #2

```
00407CAA          MOV EAX,DWORD PTR FS:[18]
00407CB1          NOP
00407CB2          MOV EAX,DWORD PTR DS:[EAX+30]
00407CB5          PUSH EAX
00407CB6          POP EAX
00407CB7          ADD EAX,0
00407CBA          MOVZX EAX,BYTE PTR DS:[EAX+2]    ; BeingDebugged
00407CBE          CMP EAX,0
00407CC1          JE SHORT 00407CD0
```

The last anti-debugging trick is `GetTickCount()` calls throughout the code: if the time difference between two calls is greater than 1000 milliseconds, program exists.

Magic checks

There is a couple of magic number checks that prevents us from getting to the code that decrypts hidden parts of text.

Checking for magic constants #1

```
00407CD0         CMP WORD PTR DS:[40B078],832
00407CD9         JNE 00407E0C
00407CDF         CMP WORD PTR DS:[40B07A],63
00407CE7         JNE 00407E0C
00407CED         CMP WORD PTR DS:[40B07E],0C
00407CF5         JNE 00407E0C
```

Checking for magic constants #2

```
00407D4A         CMP WORD PTR DS:[40B080],6
00407D52         JNE 00407E0C
```

Checking for magic constants #3

```
00407DA7        CMP WORD PTR DS:[40B082],0B
00407DAF        JNE SHORT 00407E0C
00407DB1        CMP DWORD PTR DS:[40B088],36EE80
00407DBB        JBE SHORT 00407E0C
```

We should either modify data at specific locations to contain proper values or just `NOP` several `JXX 00407E0C` commands.

Hidden parts of text decryption

Encrypted strings are located at:

```
0x403BEC, length 0x143, key "kernel32.dll";
0x403D38, length 0x3FF, key "GetProcAddress";
0x404140, length 0x080, key "ExitProcess";
```

The decryption algorithm is XOR-based.

The decryption algorithm in C

```
typedef struct {
    int size;
    char *data;
} pstr_t;

pstr_t crypted, decrypted, key;
int i;
/* Initialize pstr_t with actual data … */

for (i = 0; i < crypted.size; i++){
    decrypted.data[i] = crypted.data[i] ^ key.data[i%key.size];
}
```

After decryption of each hidden part of text, bytes at `0x4080F4`, `0x4080F8` and `0x4080FC` are set to 1.

Setting flag #1

```
00407D29        CALL 00403A94                   ; Decryption procedure call
00407D2E        MOV BYTE PTR DS:[4080F4],1
```

Setting flag #2

```
00407D86        CALL 00403A94
00407D8B        MOV BYTE PTR DS:[4080F8],1
```

Setting flag #3

```
00407DEB        CALL 00403A94
00407DF0        MOV BYTE PTR DS:[4080FC],1
```

Results

If everything was done properly (debugger detection and magic checks passed), then we should see all 3 hidden parts.

Console output

```
* Hidden part #1. Text picked from the following URL:
* http://www.virusradar.com/en/Win32_Virut.E/description

O noon of life! O time to celebrate!
O summer garden!
Relentlessly happy and expectant, standing: -
Watching all day and night, for friends I wait:
Where are you, friends? Come! It is time! It's late!

* Hidden part #2. Text picked from the following URL:
* http://www.virusradar.com/en/Win32_Ridnu.NAA/description

DEAR MY PRINCESS
WHEN THE STARS FILL THE SKY I WILL MEET YOU MY LOVELY PRINCESS
I MISS YOU SO MUCH MY PRINCESS
IN MY DEAREST MEMORY I SEE YOU REACHING OUT TO ME
I WILL REMEMBER YOU AS LONG AS YOU REMEMBER ME
IN YOUR DEAREST MEMORY DO YOU REMEMBER LOVING ME
PLEASE DO NOT FORGET OUR PAST
DID YOU KNOW THAT I HAD MIND ON YOU
I NEVER WISH TO LOSE YOU AGAIN
SHALL I BE THE ONE FOR YOU
I WANNA TAKE YOU TO MY PALACE
I WILL TAKE YOU TO OUR UTOPIA
I AM FALLING IN LOVE WITH YOU
I WILL BE WAITING FOR YOU
I DO NOT WANT TO SAY GOOD BYE TO YOU
PLEASE DO NOT FORGET YOUR PRINCE
I SAW YOU SMILING AT ME WAS IT REAL OR JUST MY FANTASY
YOU WILL ALWAYS IN MY HEART
YOU ALWAYS IN MY DREAMS
I ALWAYS SEE YOU IN MY DREAMS
I HAVE BEEN POISONED BY YOUR LOVE
I MISS YOU I AM STILL LOOKING FOR YOU
I WILL BE THERE I WILL BE WAITING FOR YOU
PLEASE COME BACK TO OUR BEAUTY ISLAND
I MISS YOUR CUTE SMILE

* Hidden part #3.

Continue with the next ESET crackme here:
http://www.joineset.com/download/bmV4dF9maWxl/crack_me_2.zip
```

STAGE 2 INVESTIGATION (http://www.joineset.com/download/bmV4dF9maWxl/crack_me_2.zip)

The downloaded crack_me_2.zip is not a .zip archive. It starts with MZ signature, so we should change extension to .exe.

Unpacking

Unpacking code is located in the `UPX1` section and unpacks data to `UPX0`. We can make a conclusion, that the first `JMP` or `CALL` to `UPX0` section is the jump to `OEP`. Such jump is located at `0x4B330E`.

Jumping to OEP

```
004B3308        CALL 004B34A8
004B330D        POPAD
004B330E        JMP 00408043    ; OEP = 0x408043
004B3313        NOP
004B3314        POPAD
004B3315        RETN
```

At the Original Entry Point

Let's now see how the crack-me's code gets decrypted.

The decryption procedure is located at `0x404B20`. Note, that this procedure uses its own body as a key for decryption, so if to place a software breakpoint within it, decryption will not be correct. Procedure decrypts code from `0x403B00` to `0x404B1F` and then makes a call to `0x4047F0`. There is where all the easter-eggs gathered.

Chicken or the egg?

All checks are done before crack-me dialog is shown.

Hamburger message

Crack-me is trying to open named pipe `\\.\pipe\Vincent_Vega` and read "Royale with Cheese" string from there (this string is stored base64-encoded as "Um95YWxlIHdpdGggQ2hlZXNl" at `0x42600C` and gets decoded before comparison). If succeeded, then a message box with title "Hamburger message" and text "Hamburgers. The cornerstone of any nutrition breakfast!" is shown.

Resource 131

Crack-me decrypts resource 131 using key "T2ggbWFuLEkgc2hvdCBNYXJ2aW4gaW4gdGhlIGZhY2U=" (base64-encoded "Oh man,I shot Marvin in the face" at `0x425FD0`) and stores it as a temporary file at temporary directory. The stored file gets locked by `CreateFile()` call at `0x4049AC`.

MALCHO.DLL

Crack-me is trying to load library called `MALCHO.DLL` (stored base64-encoded as "TUFMQ0hPLkRMTA==" at `0x4260AC`) and get an address of procedure

"Z2V0UGFzc3dvcmQ=" (base64-encoded "getPassword" at `0x4260C0`) and then call it. Procedure must return a pointer to `char[0x20]`. The returned string is used for further decryption of resource 133.

Resource 133

If `MALCHO.DLL` loaded and corresponding procedure address got, then crack-me executes the procedure and stores the returned value (`char[0x20]` type). Then crack-me decrypts resource 133 using returned string and stores the resource as a temporary file at temporary directory.

DialogBoxParamW

Dialog box initialization performed at `0x403A52`.

Dialog from template

```
00403A46        PUSH 0
00403A48        PUSH 004042C0    ; DlgProc = 004042C0
00403A4D        PUSH 0
00403A4F        PUSH 67
00403A51        PUSH EAX
00403A52        CALL DWORD PTR DS:[<&user32.DialogBoxParamW>]
```

`0x4042C0` is an address of window messages handler. Besides handling window messages, that procedure also performs passwords checking.

Anti-debugging tricks

When running crack-me under debugger, we get numerous "division by zero" and "single-step event" exceptions. These exceptions must be handled by crack-me itself. We should configure our debugger to pass these exceptions to the application.

There also exist a check for `SeDebugPriveledge`. The check is done once as `WM_INITDIALOG` message is received by crack-me dialog.

LookupPrivilegeValueW() call

```
00403DCF        PUSH ECX
00403DD0        PUSH 00425F58    ; Privilege = "SeDebugPrivilege"
00403DD5        PUSH 0
00403DD7        CALL DWORD PTR DS:[<&advapi32.LookupPrivilegeValueW>]
00403DDD        TEST EAX,EAX    ; Must return zero
00403DDF        JNZ SHORT 00403DF9
```

Passphrases

Password checking in the message handler of the dialog box (`0x4042C0`) is very simple. It is based on two `lstrcmpA()` calls.

We can just set breakpoints on each call and see what is passed to a function. But ...

I did it! - Nope you didn't

... if we enter passphrases "Pigs are filthy animals" and "I did it!" for the 1st and 2nd stages relatively, we will see a message box with text "Nope you didn't". Let's see what else we can do here.

STAGE 3 INVESTIGATION (injected procedure)

The window procedure of the edit control with ID `0x03EA` is overridden to be `0x405F50` (I used WinSpy++ to see that). When `WM_GETTEXT` message is received by this control, check for another passphrase is done.

After analysis of the handler, we can see that the needed passphrase is stored encrypted at `0x4227EC`.

Memory dump
```
004227EC  3F 93 0E 05 A6 44 6C F8 2D 47 40 21 C5 85 1F 19  ?▯##ŠDlø-G@!Å▯##
```

The xor-mask for decryption is obtained from a byte array, but it is constant and there is no need to investigate the algorithm.

The mask
```
unsigned char xor_mask[] = {
	0x65, 0xF6, 0x6A, 0x22, 0xD5, 0x64, 0x08, 0x9D,
	0x4C, 0x23, 0x6C, 0x01, 0xA7, 0xE4, 0x7D, 0x60
};
```

The correct passphrase for the second stage is "Zed's dead, baby".

Injection to userinit.exe

The next thing what the handler does is process injection. The procedure body (address `0x403E70` length `0x440`) and some data get copied to `%systemroot%/system32/userinit.exe` process memory and then the injected procedure is called via `CreateRemoteThread()`.

Investigation shows that the injected procedure makes `POST` request like `http://localhost:8080/index.php?key=XXXXXXXX` and decrypts the string that script returned. If the decrypted string is "Jules", then a message box "THE END. THE MOVIE IS OVER" is shown. After that, another one `POST` request performed. It looks like `http://localhost:8080/index.php?r=XXX...` with xor-ed "Thats all. Congratulations!" string as value of `r` variable.

Sample application and PHP script sources that do the stuff provided below.

Sample index.php source
```
<?php
$key = $_POST['key'];
$r = $_POST['r'];

if ($key != ""){
	file_put_contents('key.txt', $key);
	file_put_contents('value.txt', 'Jules');
	system('app.exe key.txt value.txt');
}
else if ($r != ""){
      file_put_contents('value.txt', $r);
      system('app.exe key.txt value.txt > thats.all');
} ?>
```

Sample app.exe source

```
#include <stdio.h>
#include <string.h>

void swap(unsigned char *val_1, unsigned char *val_2)
{
    unsigned char tmp_val_1;

    if (!val_1 || !val_2){
        return;
    }
    tmp_val_1 = *val_1;
    *val_1 = *val_2;
    *val_2 = tmp_val_1;
}

void tbl_init(unsigned char tbl[256], char *key)
{
    size_t key_len = strlen(key);
    unsigned char acc_i = 0;
    int i;

    for (i = 0 ; i < 256; i++){
        tbl[i] = i;
    }

    for (i = 0; i < 256; i++){
        acc_i += key[i%key_len] + tbl[i];
        swap(&tbl[i], &tbl[acc_i]);
    }
}

void tbl_mod_value(unsigned char tbl[256], char *str)
{
    unsigned i = 0;
    unsigned str_len = strlen(str);
    unsigned char acc_i = 0;

    for (i = 1; i <= str_len; i++){
        acc_i += tbl[i];
        swap(&tbl[i], &tbl[acc_i]);
        str[i-1] ^= tbl[ (unsigned char)(tbl[i] + tbl[acc_i])];
    }
}

int read_content(char *filename, unsigned char *buf, unsigned max)
{
    FILE *f;
    if (!filename || !buf){
        return -1;
    }
    f = fopen(filename, "r");
    if (!f){
        return -1;
    }
    fread(buf, 1, max, f);
    fclose(f);
    return 0;
}

int main(int argc, char * argv[])
{
    unsigned char tbl[256] = {0};
    unsigned char value[100] = {0};
    char key[9] = {0};

    if (argc < 3){
        printf("usage: %s <key-file> <input-file>", argv[0]);
        return -1;
    } else {
        if (read_content(argv[1], &key[0], 8)) { return -1; }
        if (read_content(argv[2], &value[0], 100)) { return -1; }
        tbl_init(tbl, key);
        tbl_mod_value(tbl, value);
        fputs((const char *)value, stdout);
    }
    return 0;
}
```

Results

Finally, after we set up our own `http://localhost:8080` site and place correct `index.php` script there, we should enter "Pigs are filthy animals" and "Zed's dead, baby" passphrases for the stage 2 crack-me and see "THE END" message box. Also, there will appear `thats.all` file near `index.php` containing "Thats all. Congratulations!" text.

EASTER EGG 1 INVESTIGATION (unpacked resource 131 - .net assembly)

(This is my first experience in .net reversing, so I may be talking crap somewhen).

What to start with?

When I was advised to use .net reflector to reverse-engineer this assembly and I when saw how it automatically restored full hierarchy of classes, I was very happy. I just exported the assembly into C# project, then tried to compile it and ... then compilation failed. This assembly is obfuscated so I was unable to get working C# project that I wanted. You better try to do it yourself to understand why.

The only way I found to get rid of obfuscation is to decompile the assembly into msil, rename classes, methods and variables manually and then compile it back.

The payload

After analyzing fully restored and refactored source code of the assembly, we should have find out, that this easter egg performs two related tasks: 1. it enumerates child windows of `iexplore.exe` process sending `WM_GETTEXT` message to them and looking for some substring in their titles and 2. it opens a named pipe `\\.\pipe\Vincent_Vega` and, depending on whether a window containing needed substring in title found or not, writes one of "Royale with Cheese" or "Le Big Mac" strings there relatively.

Which window is searched?

Enumeration procedure that is passed as a parameter to `EnumChildWindows()` function calls a method which body is dynamically generated via a sequence of `Emit()` calls. ILVisualizer for Visual Studio was very helpful in restoring the body of that method. The method performs the search of a substring and its logic is rather simple.

Substring decryption and searching in C
```
unsigned char encrypted_buf[35] = {
        0x89, 0xC1, 0xCB, 0xD4, 0xCC, 0xCD, 0xCF, 0xFE,
        0xFA, 0xB2, 0xFE, 0xF6, 0xCC, 0xAD, 0xFC, 0xD1,
        0xAF, 0xB8, 0xF5, 0xC4, 0xEA, 0xFD, 0xBF, 0xEC,
        0xC9, 0xC8, 0xE7, 0xC1, 0xC2, 0xED, 0xE3, 0xA9,
        0xF2, 0xFD, 0xF0
};

char *buf_decrypt(char *encrypted_buf)
{
      int i, key;
      char decrypted[35];

      for(i = 0; i < 35; i++){
          decrypted[i] = encrypted_buf[i] ^ (key - i);
      }
}
```

```
bool dyn_generated_func(char *window_title, char *encrypted_buf)
{
      return (strstr(window_title, buf_decrypt(encrypted_buf)) != NULL);
}
```

Bruteforced key is `0xA6`, so the substring that is encrypted is "/download/bmV4dF9maWxl/cGEkJHdk.txt". We can now open link `http://www.joineset.com/download/bmV4dF9maWxl/cGEkJHdk.txt` in Internet Explorer, then run stage 2 crack-me (keep easter-egg running) and see the "Hamburger message".

EASTER EGG 2 INVESTIGATION (resource 133)

The file at `http://www.joineset.com/download/bmV4dF9maWxl/cGEkJHdk.txt` contains a single line of text: "Jules:You read the bible, Ringo?". This is a key for decryption of resource 133.

We are now to create MALCHO.DLL. As I was unable to create a dynamic library with an exported symbol "TUFMQ0hPLkRMTA==", I just modified text at `0x4260C0` of stage 2 crack-me executable to be "getPassword".

Exported library function

```
extern "C" __declspec(dllexport) getPassword()
{
      return "Jules:You read the bible, Ringo?";
}
```

Now, if we put our .dll together with stage 2 crack-me executable, then resource 133 will be unpacked and stored in temporary directory.

It is a Java .class file (as it starts with `0xCAFEBABE` signature) with "Bible" class inside. Let's run it.

Shell commands

```
cd <class-file-location>
java -cp . Bible
```

Results

```
Ezekiel 25:17
----------------------------------------------------------------------------------
-------------------------------
The path of the righteous man is beset on all sides by the inequities of the
selfish and the tyranny of evil men.
Blessed is he, who in the name of charity and good will, shepherds the weak
through the valley of darkness,
for he is truly his brother's keeper and the finder of lost children.
And I will strike down upon thee with great vengeance and furious anger those who
would attempt to poison
and destroy my brothers.And you will know my name is the Lord when I lay my
vengeance upon thee.
----------------------------------------------------------------------------------
-------------------------------
Great! You found all easter eggs!
```

Alexander Makarov. September 4, 2013
seems.deviant@gmail.com