Minerva Labs

Don't Chase, PREVENT!

Joe’s Crypter – Fixed and PREVENTED!

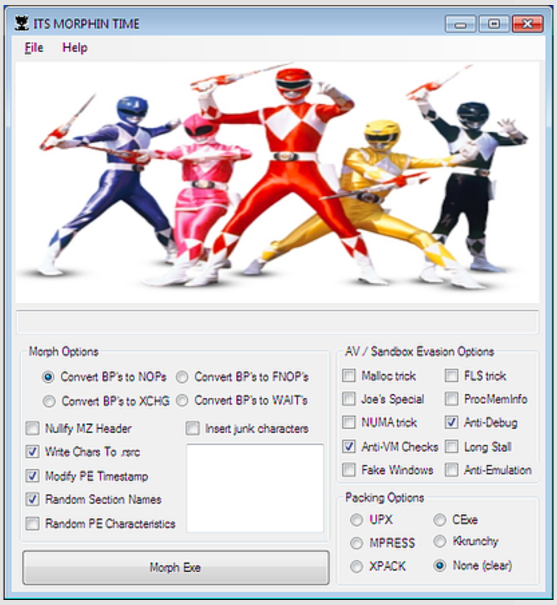
December 28, 2015 | Gal Bitensky

Intro

Crypters are tools used by attackers to change the binary signature of a malware while keeping its original functionality intact. The main reason for using them is to evade detection.

Last week Joe Giron, a security researcher, released a new crypter (joecrypter).

Joe chose the crypter theme to be the 90’s classic – Mighty Morphin’ Power Rangers:



Morph Options
- [✓ Convert BP's to NOPs] Convert BP's to FNOP's
- Convert BP's to XCHG Convert BP's to WAIT's
- Nullify MZ Header
- Insert junk characters
- [✓ Write Chars To .rsrc]
- [✓ Modify PE Timestamp]
- [✓ Random Section Names]
- Random PE Characteristics

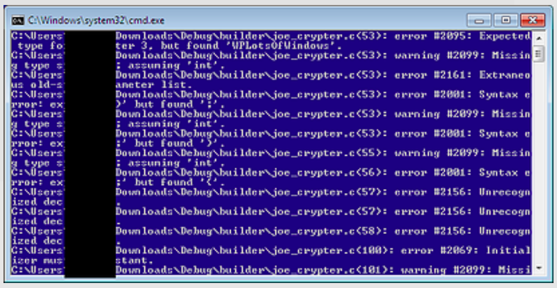
AV / Sandbox Evasion Options
- Malloc trick
- FLS trick
- Joe's Special
- ProcMemInfo
- NUMA trick
- [✓ Anti-Debug]
- [✓ Anti-VM Checks]
- Long Stall
- Fake Windows
- Anti-Emulation

Packing Options
- UPX CExe MPRESS Kkrunchy XPACK [✓ None (clear)]

This is how the author described his work:

The crypter I made modifies exes, packs them, and adds AV / VM / Sandbox / debugging evasions inside of a wrapper. I’m employing a basic process hollowing technique for the payload that is only run after all evasions are satisfied. The anti-debug modules include anti-single stepping as well as anti-tracing. I can even detect procmon without checking the process list.

We decided to check it out, but when we tried to pack a plain calc.exe – window with multiple errors popped up:



```
C:\Users[redacted]Downloads\Debug\builder\joe_crypter.c(53): error #2095: Expected
 type fo          ter 3, but found 'WPLotsOfWindows'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(53): warning #2099: Missin
g type s          ; assuming 'int'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(53): error #2161: Extraneo
us old-s          ameter list.
C:\Users          Downloads\Debug\builder\joe_crypter.c(53): error #2001: Syntax e
rror: ex          )' but found ';'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(53): warning #2099: Missin
g type s          ; assuming 'int'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(53): error #2001: Syntax e
rror: ex          ;' but found ')'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(55): warning #2099: Missin
g type s          ; assuming 'int'.
C:\Users          Downloads\Debug\builder\joe_crypter.c(56): error #2001: Syntax e
rror: ex          ;' but found '('.
C:\Users          Downloads\Debug\builder\joe_crypter.c(57): error #2156: Unrecogn
ized dec          .
C:\Users          Downloads\Debug\builder\joe_crypter.c(57): error #2156: Unrecogn
ized dec          .
C:\Users          Downloads\Debug\builder\joe_crypter.c(58): error #2156: Unrecogn
ized dec          .
C:\Users          Downloads\Debug\builder\joe_crypter.c(100): error #2069: Initial
izer mus          stant.
C:\Users          Downloads\Debug\builder\joe_crypter.c(101): warning #2099: Missi
```

Featured Posts

CopyKittens Attack Group
November 23, 2015

Recent Posts

Joe’s Crypter – Fixed and PREVENTED!
December 28, 2015

Duuzer - A Threat Prevented
November 30, 2015

Protecting ALL Browsers Against Binary Injection
November 24, 2015

CopyKittens Attack Group
November 23, 2015

Archive

December 2015 (1)

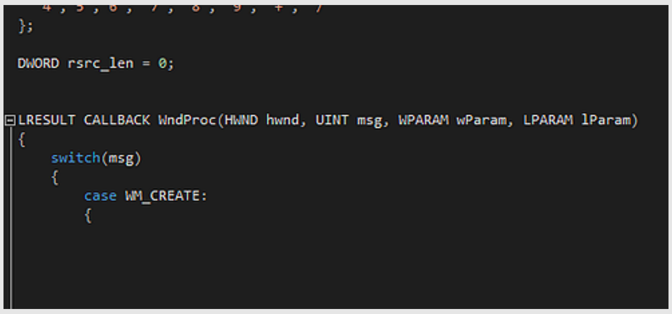
November 2015 (3)

Fixing The Cypter

This output indicated compilation errors, so we started to analyze the build process. This was not too difficult as Joe was kind enough to attach most of the source and the compiler used.

We found out that for each build a C source file named dont_touch_me.joe is modified as an intermediate part of the build process. This modified source is saved as joe_crypter.c, and is the cause for all the compile-time errors we experienced.

We took a closer look in the original dont_touch_me and noticed that its author prepared multiple blank lines in one of the functions for some reason:

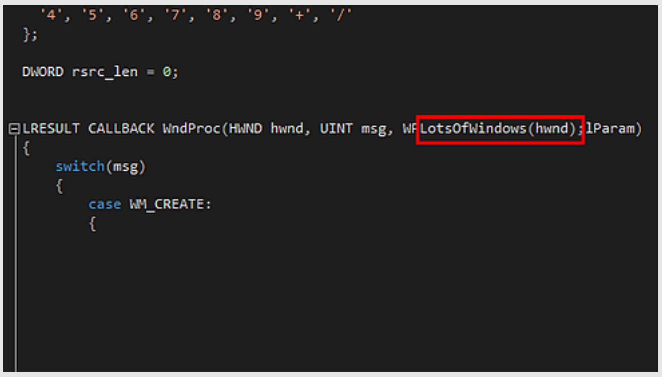


```
};

DWORD rsrc_len = 0;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
        case WM_CREATE:
        {
```

We also observed that the compilation errors begin in this function declaration – so we took a closer look in the modified joe_crypter.c in the exact same place:



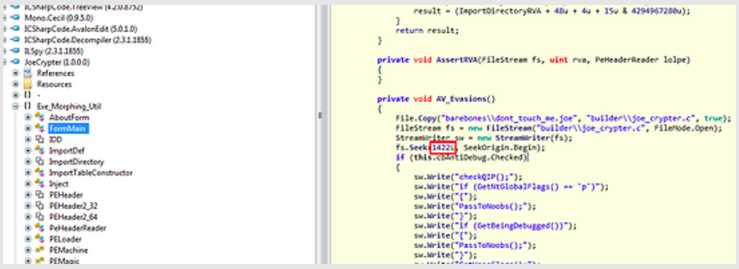
```
    '4', '5', '6', '7', '8', '9', '+', '/'
};

DWORD rsrc_len = 0;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPLotsOfWindows(hwnd);lParam)
{
    switch(msg)
    {
        case WM_CREATE:
        {
```

LotsOfWindows() is a function declared later in this file, implementing anti-analysis technique – it is quite clear that it should not be called in the middle of other function’s declaration...

Now we used ILSpy and took a closer look in the part of the .NET application in charge of morphing don’t_touch_me to joe_crypter:



```
            result = (ImportDirectoryRVA + 48u + 4u + 15u & 4294967280u);
        }
        return result;
    }

    private void AssertRVA(FileStream fs, uint rva, PeHeaderReader lolpe)
    {
    }

    private void AV_Evasions()
    {
        File.Copy("barebones\\dont_touch_me.joe", "builder\\joe_crypter.c", true);
        FileStream fs = new FileStream("builder\\joe_crypter.c", FileMode.Open);
        StreamWriter sw = new StreamWriter(fs);
        fs.Seek(1422L, SeekOrigin.Begin);
        if (this.cbAntiDebug.Checked)
        {
            sw.Write("checkQIP();");
            sw.Write("if (GetNtGlobalFlags() == 'p')");
            sw.Write("{");
            sw.Write("PassToNoobs();");
            sw.Write("}");
            sw.Write("if (GetBeingDebugged())");
            sw.Write("{");
            sw.Write("PassToNoobs();");
            sw.Write("}");
```

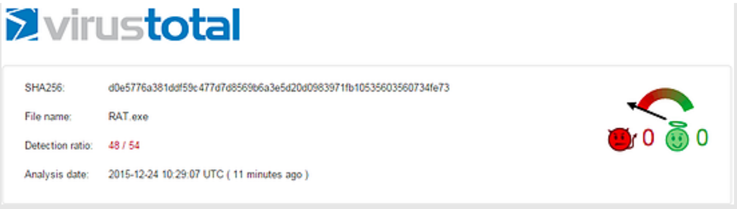
We suspected that the fs.seek offset calculation was wrong, and this was indeed the problem! The offset was supposed to send us straight to the blank lines prepared in advance but instead was about 80 bytes short.

We chose to modify don’t_touch_me so that the 1422L offset will match the blank lines in the function. Alternatively, we could have patched the compiled .NET with the right offset.

Voila! Now, we were able to pack binaries and asses the tool’s abilities.

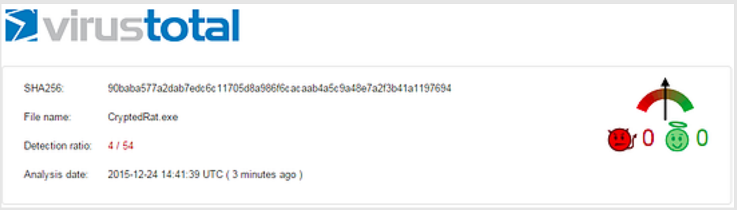
The Crypter

To have a sense of its power we compared the detection ratio of an extremely familiar piece of malware – packed and unpacked. The well-known Xtreme RAT was chosen for this mission, and in his undisguised form it was detected by 48/54 AV vendors:



```
SHA256:          d0e5776a381ddf59c477d7d8569b6a3e5d20d0983971fb10535603560734fe73
File name:       RAT.exe
Detection ratio: 48 / 54
Analysis date:   2015-12-24 10:29:07 UTC ( 11 minutes ago )
```

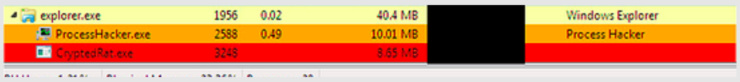
When we packed the same binary with Joecrypter it was detected only by 4/54 vendors:



```
SHA256:          90baba577a2dab7edc6c11705d8a986f6cacaab4a5c9a48e7a2f3b41a1197694
File name:       CryptedRat.exe
Detection ratio: 4 / 54
Analysis date:   2015-12-24 14:41:39 UTC ( 3 minutes ago )
```

PREVENTION

Minerva Labs Armor prevents infection by any kind of malware packed by this crypter. This is for example the packed Xtreme RAT, terminated on an workstation protected by Minerva Armor before it made any harm:



```
◢ explorer.exe          1956   0.02    40.4 MB   [redacted]   Windows Explorer
    ProcessHacker.exe   2588   0.49   10.01 MB                Process Hacker
    CryptedRat.exe      3248           8.65 MB
```

We would like to express our appreciation to Joe for sharing his original work publicly, it gives the community a chance to face challenges very similar to off-the-shelf crypters sold in underground hacking forums.

Want to see Minerva in action? Request a demo!

Minerva - don't chase, PREVENT!