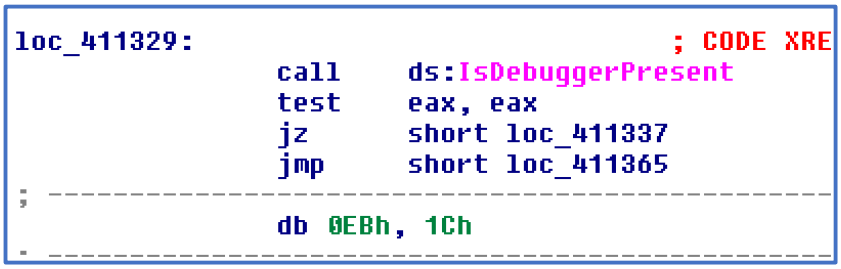
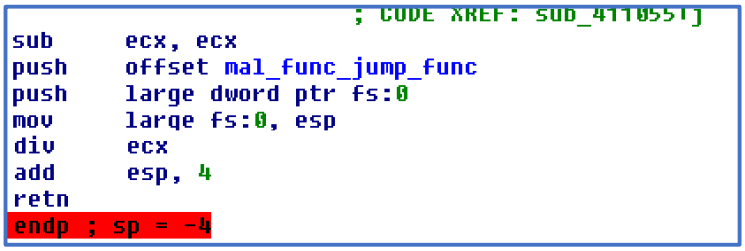
Lab 08-1.malware

1. **What anti-disassembly techniques are used in the binary?**

The binary uses rogue bytes after unconditional jumps which would cause errors in disassemblers that use the linear disassembly strategy. The disassemblers would erroneously try to read these bytes as instructions. Flow-oriented disassemblers like IDA would never reach these, thus never be confused by them. An example is shown below



Additionally, the malware uses several techniques that make analysis more difficult. Most functions in the malware have several misdirecting jumps before they can get called. Lastly, the malware hides its true functionality from plain sight by using SEH (Structured Exception Handling). The malware pushes the function it *really* wants to call into a designated exception handling spot on the stack (the first element of the Thread Information Block, [FS:0x00]), and then intentionally creates an exception by dividing by 0.

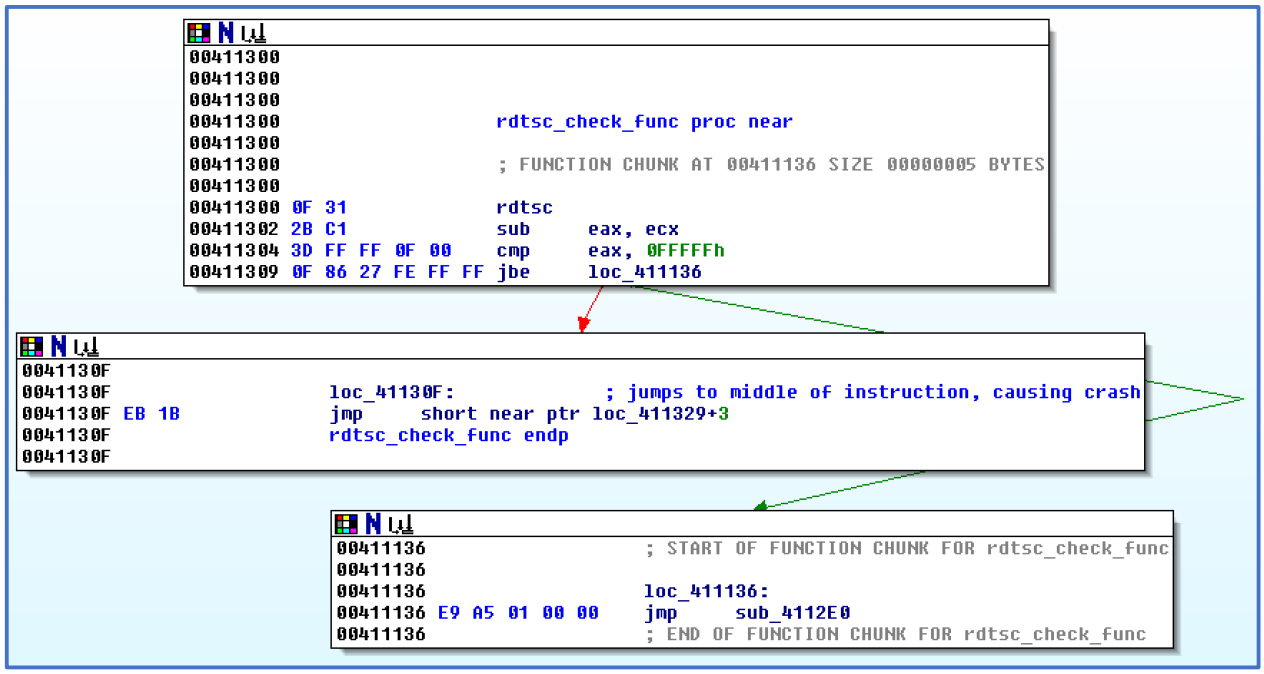


s

2. **What anti-debugging techniques are used in the binary?**

The malware makes a call to *IsDebuggerPresent*, and if it detects a debugger, it immediately exits. If it does not detect a debugger, it calls a function which calls *rdtsc*, which gets the number of cycles since the computer has started. It compares this with 0xFFFFF, and if the computer has had less cycles than this (1,048,575), it will continue successfully to the malicious function. If the computer has had more cycles than this, it will jump to the middle of an instruction, intentionally crashing the program. It is

unclear to me why and how the malware author intended to use the rdtsc instruction here.



This is an atypical use of the rdtsc instruction, as usually the instruction is called twice to calculate the cycle difference between two points in code. If a debugger was present, this cycle difference would be much greater than normal. Once successfully passing the rdtsc anti-debugging trick, the malware calls its malicious function.

3. **What does the sample do?**

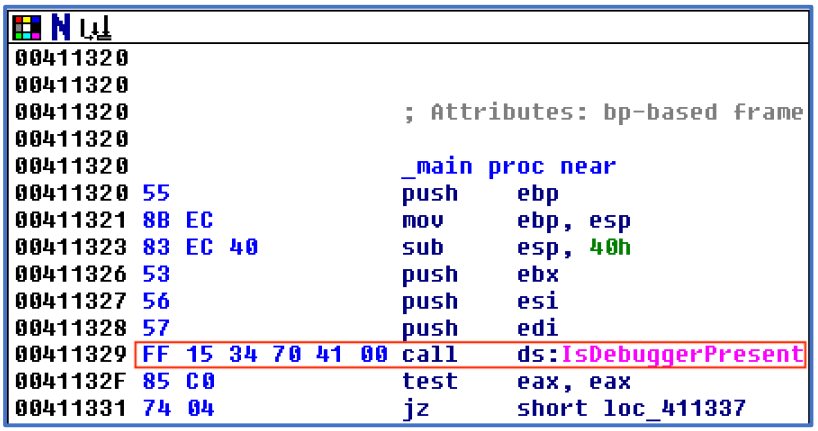
The malicious subroutine creates a file called *kingkai.bat* in the current users /AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup directory, and fills it with the contents:

```
@echo off
shutdown /s /f /t 0
```

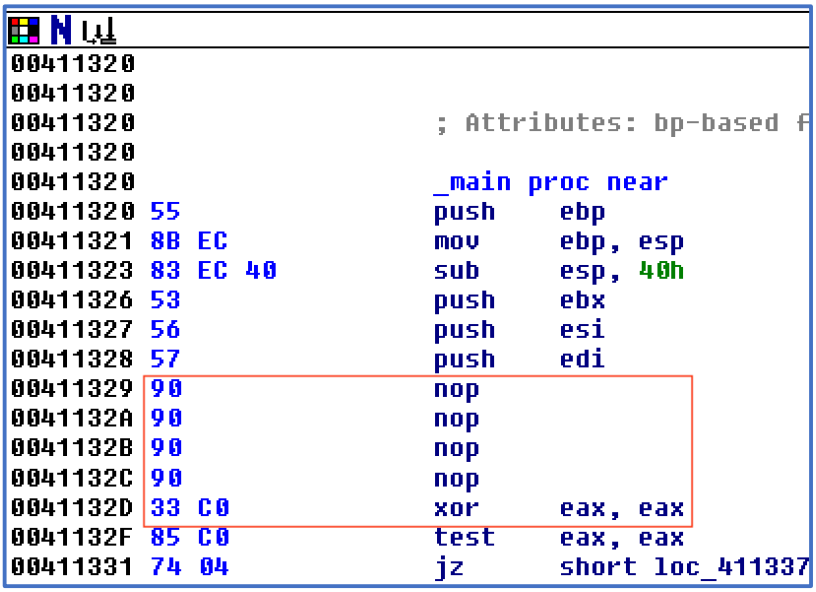
This attempts to shut down the computer every time it starts up.

4. **Patch the PE to bypass any techniques found.**

The following code patches show how to remove the anti-debugging features from this sample.



Before Figure



After/Patched Figure

The call to *IsDebuggerPresent* takes up 6 bytes, so I overwrote it with 4 NOPs and a **`xor eax, eax`**, which sets the eax register to 0. By doing so, the next instruction, **`test eax, eax`** always sets the zero flag, and the subsequent conditional jump-if-zero always occurs, which is what happens with the original malware at this point when it doesn't detect a debugger.

Additionally, since this patch address originally contained a function call, this address is in the relocation table, which will likely alter the patched bytes after being loaded into memory. As such, I used a tool called RelocEditor (http://www.woodmann.com/collaborative/tools/index.php/RelocEditor) to remove this entry from the table and ensure the **`xor eax, eax`** instruction didn't get altered.

The rdtsc patch was slightly easier. Instead of actually using the return value from the rdtsc instruction (which gets put into edx:eax), I just used the **`xor eax, eax`** instruction again to guarantee that the next jump instruction would jump to the 'true' path. The 'true' path leads to the malicious function, as shown in the last figure.

```
00411300 0F 31                rdtsc
00411302 2B C1                sub     eax, ecx
00411304 3D FF FF 0F 00       cmp     eax, 0FFFFFh
00411309 0F 86 27 FE FF FF    jbe     loc_411136
```

Before Figure

```
00411300 0F 31                rdtsc
00411302 33 C0                xor     eax, eax
00411304 3D FF FF 0F 00       cmp     eax, 0FFFFFh
00411309 0F 86 27 FE FF FF    jbe     loc_411136
```

After/Patched Figure