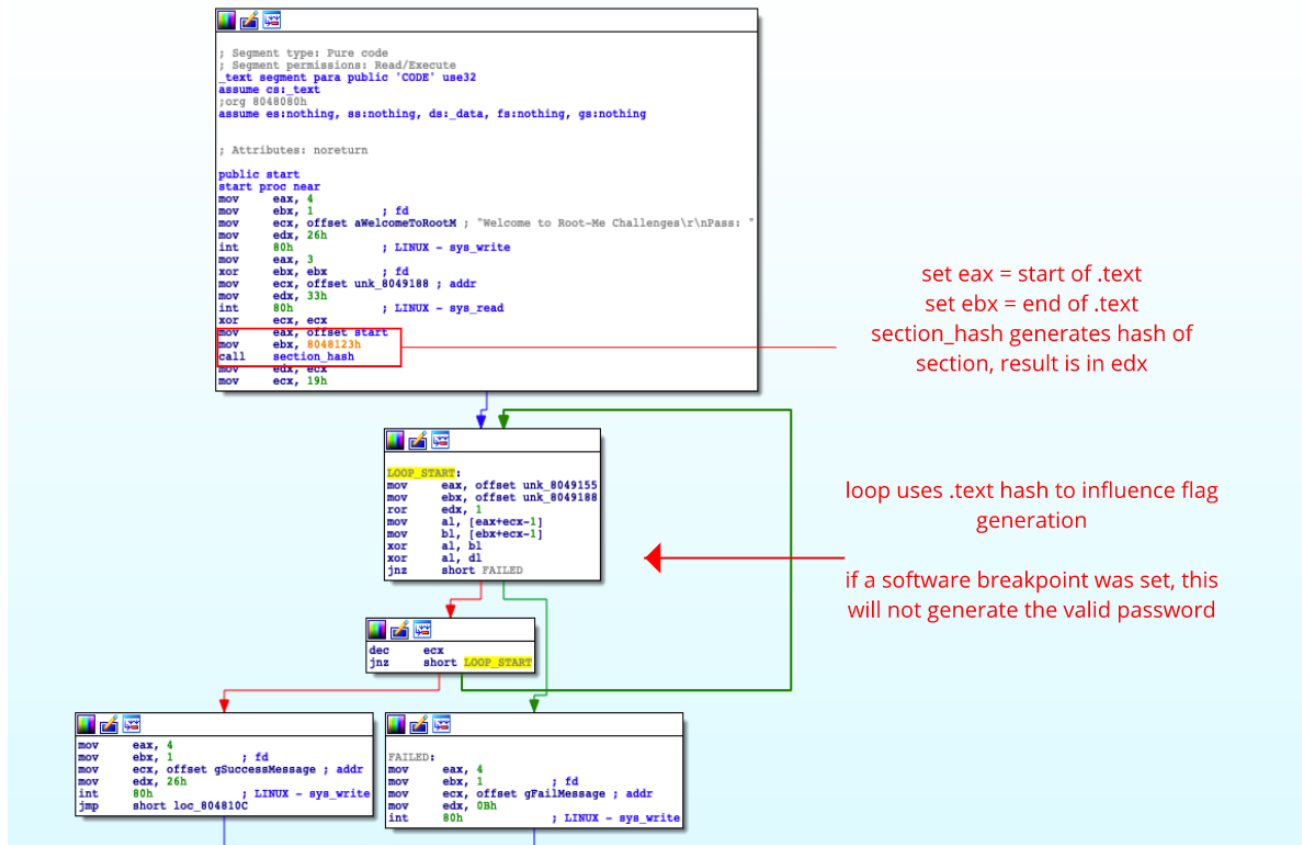


# Catching Debuggers with Section Hashing

malwareandstuff.com/catching-debuggers-with-section-hashing/

January 24, 2021



Published by [hackingump](#) on January 24, 2021

As a Reverse Engineer, you will always have to deal with various anti analysis measures. The amount of possibilities to hamper our work is endless. Not only you will have to deal with code obfuscation to hinder your static analysis, but also tricks to prevent you from debugging the software you want to dig deeper into. I want to present you [Section Hashing](#) today.

I will begin by explaining how software breakpoints work internally and then give you an example of a [Section Hashing](#) implementation.

## Debuggers – How software breakpoints work

When you set a breakpoint in your favourite debugger at a specific instruction, the debugger software will replace it temporarily with another instruction, which causes a fault or an interrupt. On x86, this is very often the `INT 3` instruction, which is the opcode `0xCC`. We can examine how this looks like in RAM.

We open `x32dbg.exe` and debug a 32 bit PE and set a breakpoint near the entry point.

```
55      PUSH EBP
8B EC   MOV EBP, ESP
83 EC 10 SUB ESP, 10
33 C0   XOR EAX, EAX
```

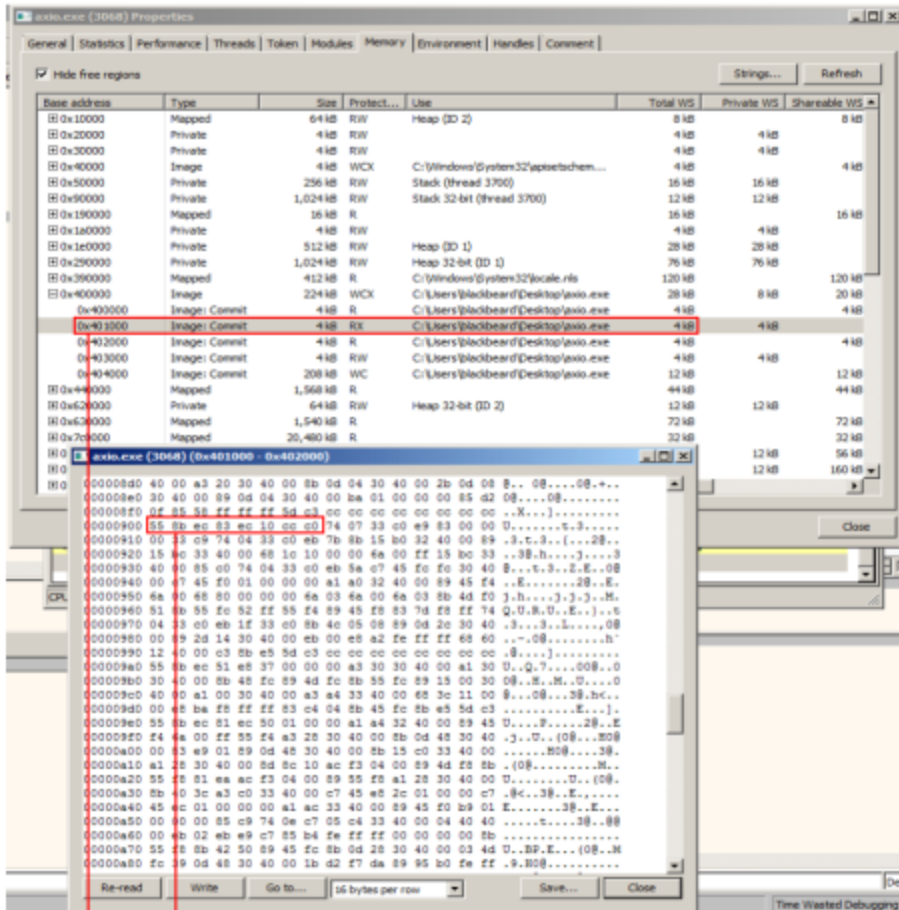
00401900	55	push ebp	EntryPoint
00401901	8B EC	mov ebp, esp	
00401902	83 EC 10	sub esp, 10	
00401906	33 C0	xor eax, eax	
00401908	74 07	je ax10.401911	
0040190A	33 C0	xor eax, eax	
0040190C	E9 83000000	jmp ax10.401994	
00401911	33 C9	xor ecx, ecx	
00401913	74 04	je ax10.401919	
00401915	33 C0	xor eax, eax	
00401917	E8 78	jmp ax10.401994	
00401919	8B 15 B0324000	mov edx, dword ptr ds:[<LoadCursorA>]	edx:EntryPoint
0040191F	89 15 BC334000	mov dword ptr ds:[40338C], edx	edx:EntryPoint
00401925	68 1C100000	push 101C	
0040192A	6A 00	push 0	
0040192C	FF 15 BC334000	call dword ptr ds:[40338C]	
00401932	85 C0	test eax, eax	
00401934	74 04	je ax10.40193A	
00401936	33 C0	xor eax, eax	
00401938	E8 5A	jmp ax10.401994	
0040193A	C7 45 FC FC304000	mov dword ptr ss:[ebp-4], ax10.4030FC	4030FC:L"\\mfc\\mfc.in1"
00401941	C7 45 F0 01000000	mov dword ptr ss:[ebp-10], 1	
00401948	A1 A0324000	mov eax, dword ptr ds:[<createfilew>]	
0040194D	89 45 F4	mov dword ptr ss:[ebp-C], eax	
00401950	6A 00	push 0	
00401952	68 80000000	push 80	
00401957	6A 03	push 3	
00401959	6A 00	push 0	
0040195B	6A 03	push 3	
0040195D	8B 4D F0	mov ecx, dword ptr ss:[ebp-10]	
00401960	51	push ecx	
00401961	8B 55 FC	mov edx, dword ptr ss:[ebp-4]	edx:EntryPoint

Disassembly

BREAKPOINT ON 0x401906

view of debugged program

When setting a breakpoint, you will see the original instruction instead of the patched one in the debugger. However, we can examine the same memory page in RAM with ProcessHacker.



Code

Examining executable  
executable memory

section in RAM during debug session

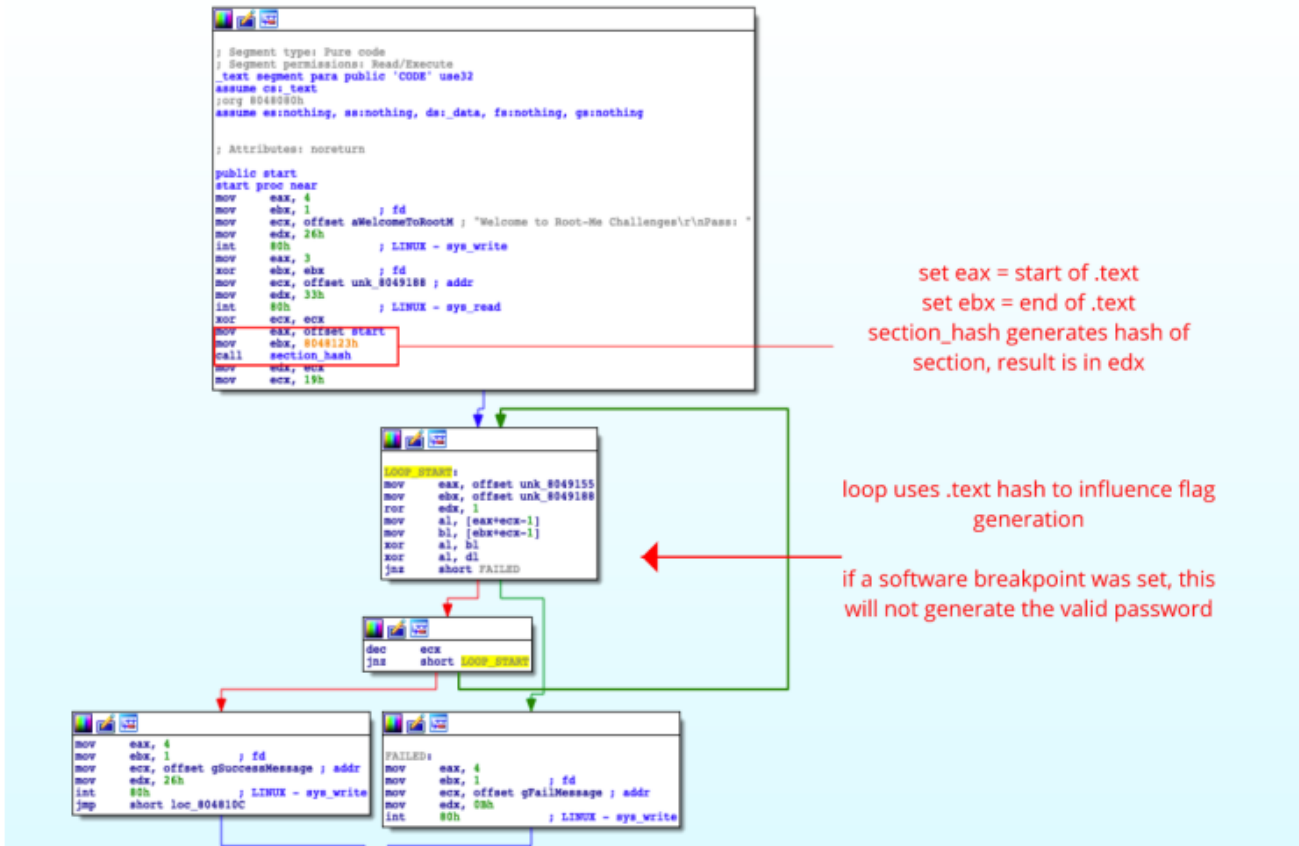
In volatile memory, the byte 33 changed to CC, which will cause the program to halt when reached. This software interrupt will then be handled by the debugger and the code will be replaced again.

### Catching Breakpoints with Section Hashing

After explaining how software breakpoints work, I'll get to the real topic of this article now. We will move to the Linux world now for this example.

A software breakpoint is actually nothing else than a code modification of the executable memory section in RAM. Once a breakpoint is set, the .text section will be modified. A very known technique to catch such breakpoints in RAM is called Section Hashing.

Authors can embed the hash of the .text section in the binary. Upon execution, they use the same algorithm to generate a new hash from the .text section. If a software breakpoint is set, the hash will differ from the embedded hash. An example implementation can look like this:



### Example implementation of Section Hashing

In this case, a hash of the .text section is generated. Afterwards it is used to influence the generation of the flag. If a software breakpoint is set during execution, a wrong hash will be generated.

This is a simple example of **Section Hashing**. In combination with code obfuscation and other anti analysis measurements, it can be very hard to spot this technique. It is also occasionally used by commercial packers.

## Defeating Section Hashing

There are multiple ways to defeat this technique, some of them could be:

- Patching instructions
- Using hardware breakpoints

Instead of modifying the code in Random Access Memory, in x86 hardware breakpoints use dedicated registers to halt the execution. Hardware Breakpoints are still detectable.

In Windows, the program can fetch the `CONTEXT` via `GetThreadContext` to see if the debugging registers are used. A great example on how this is implemented can be found here[1]. If you are interested in trying to defeat it by yourself, you can try to beat the `Section Hashing` technique by yourself at root-me.org[2].