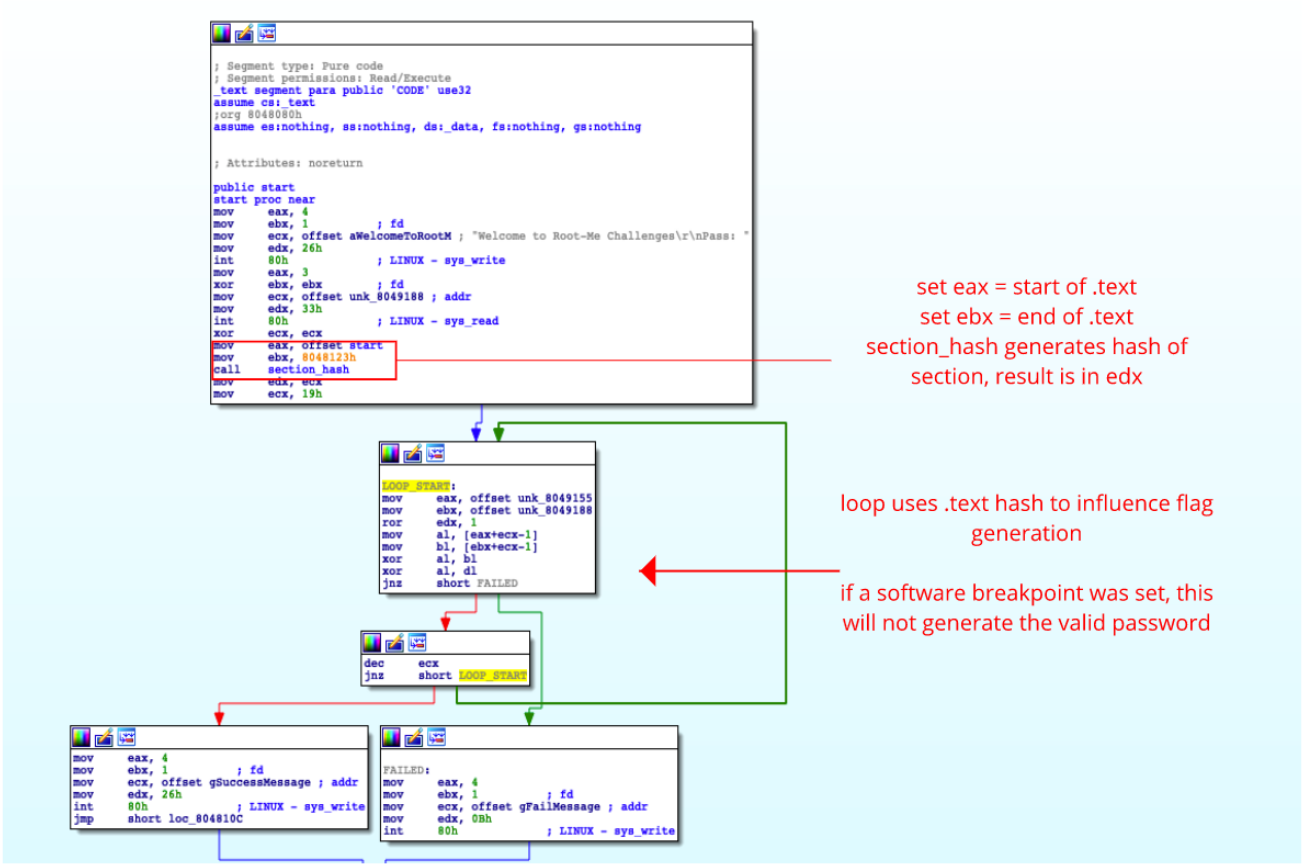
Catching Debuggers with Section Hashing

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.

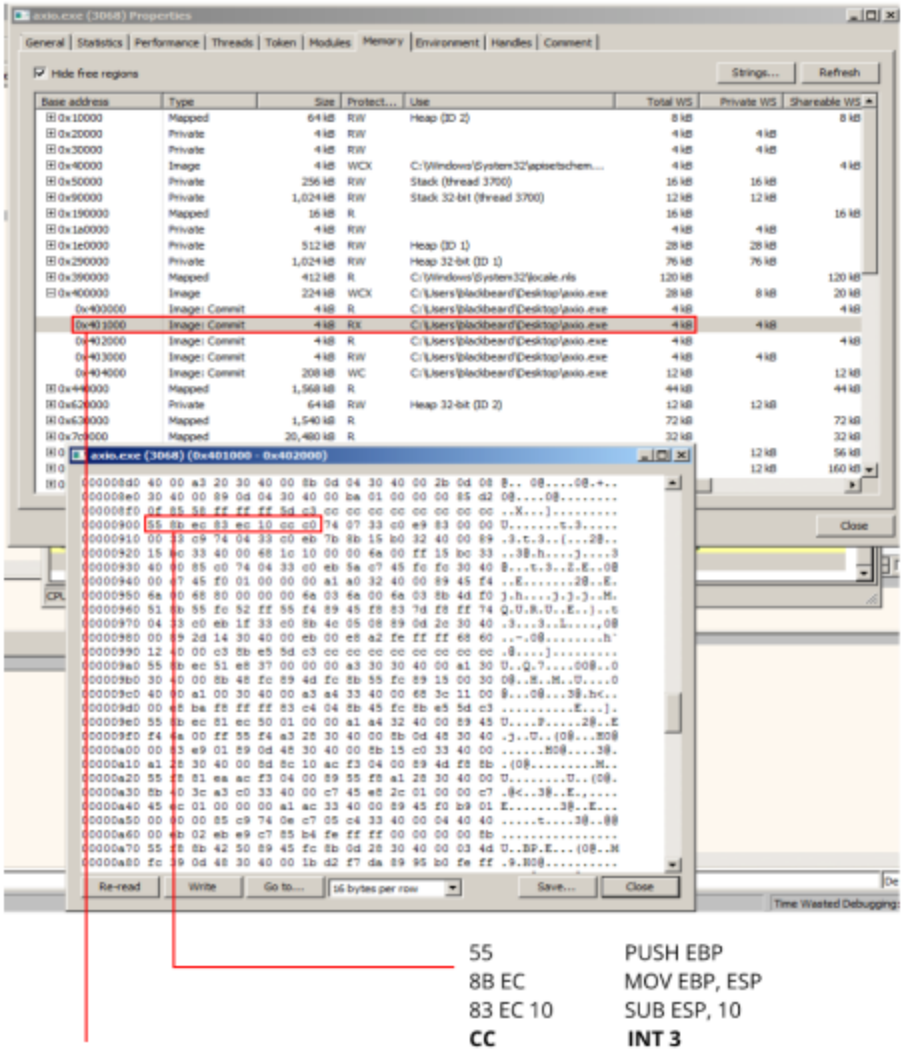
Debuggers – How software breakpoints work

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.

Disassembly 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.



Examining executable executable memory

Code

section in RAM during debug session

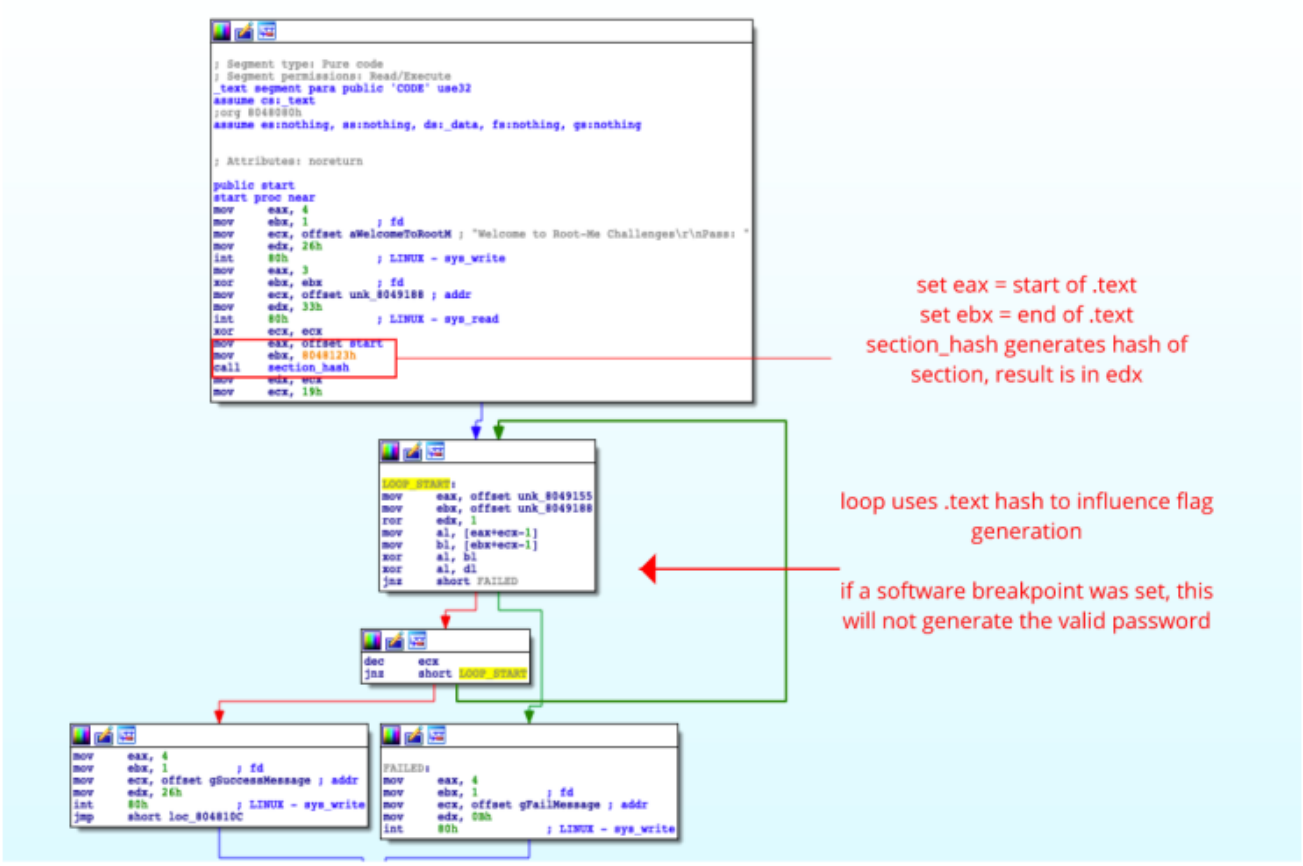
In volatile memory, the byte `33` changed to `CC`, which will cause the program to halt when reached. This software interrupt will then be handled by the debugger and the code will be replaced again.

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].