February 2020

Obfuscating and Deobfuscating Code Jump Addresses via Readable Memory

Aaron Vaage

Obfuscating and Deobfuscating Code Jump Addresses via Readable Memory

ABSTRACT

For enhanced security, code execution can be obfuscated to mask the specific memory locations to which the code flow jumps. In such cases, the traditional approach is to obfuscate the corresponding instructions using writeable-executable memory and deobfuscate them at the time of execution. However, writable-executable memory can allow execution of arbitrary code, thus posing a security risk. Therefore, many application environments forbid the use of writable-executable memory, thus hampering code obfuscation via the traditional mechanism. This disclosure describes techniques to perform obfuscation of connections between instructions within executable code without the use of writable-executable memory.

KEYWORDS

- Code obfuscation
- Code deobfuscation
- Software execution
- Execution flow
- Jump/call instruction
- Memory addresses
- Static code analysis
- Writable-executable memory
- General purpose register

BACKGROUND

Computer code or software often involves execution flows that include jumping to another location within the code and continuing execution with the instructions present at that

location. For enhanced security, such code execution can be obfuscated to mask the specific memory locations to which the code flow jumps. For instance, code obfuscation can help in combating static code analysis and/or unauthorized copying. In such cases, the traditional approach is to obfuscate the corresponding instructions using writeable-executable memory and deobfuscate them at the time of execution. However, writable-executable memory can allow execution of arbitrary code, thus posing a security risk. Therefore, many application environments forbid the use of writable-executable memory, thus hampering code obfuscation via the traditional mechanism.

DESCRIPTION

This disclosure describes techniques to perform obfuscation of connections between instructions within executable code without the use of writable-executable memory. Specifically, the techniques involve obfuscating the specific memory address of each non-sequential jump within code from one instruction to the next. Such code flow changes are located by identifying each jump/call instruction that results in the execution flow jumping to the code at the address specified within the instruction. The absolute memory address specified in the instruction is resolved and stored in readable memory in an obfuscated format. The corresponding instruction is then replaced with an indirect jump/call instruction that uses the respective obfuscated value of the address. Instructions to deobfuscate the address are added before execution reaches the jump/call instruction so that the code flow can proceed appropriately to the correct memory address specified in the original unobfuscated instruction.

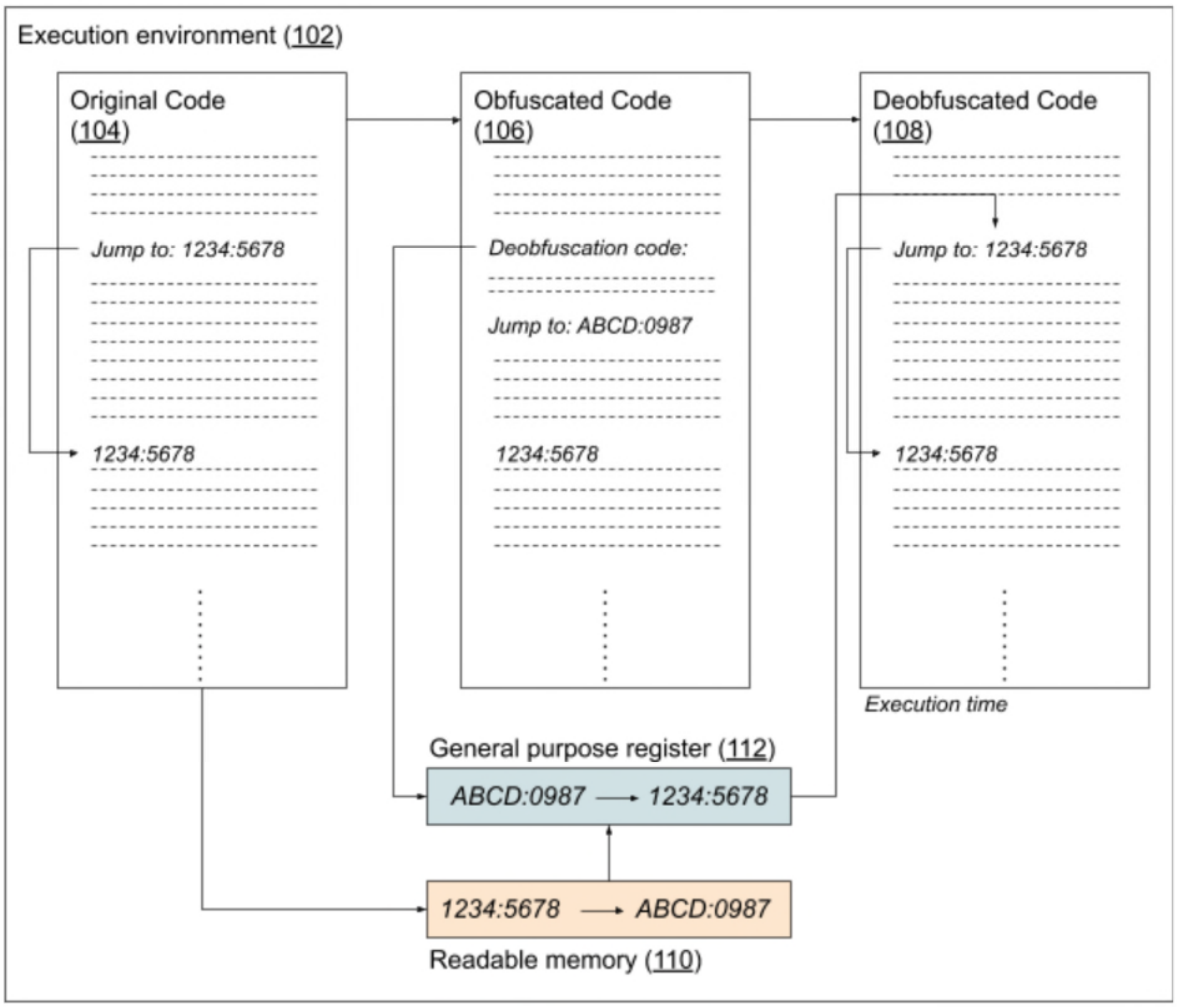

Fig. 1: Obfuscation and deobfuscation of jump/call instruction via readable memory

Fig. 1 shows an operational implementation of the techniques described in this disclosure. Original code (104) designed to run within an execution environment (102) includes an instruction for the code flow to jump to address 1234:5678. The address is obfuscated to ABCD:0987 via readable memory (110), resulting in obfuscated code (106) that contains corresponding deobfuscation code inserted prior to the jump/call instruction. At execution time, the code is deobfuscated (108) via a general purpose register (112) that is isolated from the code.

Operational implementation of the deobfuscation techniques involves copying the obfuscated address to a register and performing the deobfuscation within the register. Such an operation avoids storing the deobfuscated address in long-term memory and makes it unnecessary to reobfuscate the address during execution. Importantly, the deobfuscation is performed in isolation from the corresponding jump/call instruction to ensure that the relationship between the data source and usage remains unexposed. Alternatively, or in addition, exposure of the deobfuscation can be avoided by implementing it as multiple partial operations spread across multiple memory locations. Moreover, to limit the effect of one value being revealed or deduced, each jump/call instruction can be coded with a unique obfuscated value even if several instructions reference the same code address for the jump location.

In operational implementations, resolving the absolute address of the jump location can require computing the relative virtual address depending on the specifics of the underlying operating system and/or hardware. Similarly, storing the obfuscated value in readable memory can require the use of specifically marked memory sections based on the constraints imposed by the application and/or the operating system and/or the hardware.

The techniques described in this disclosure can be utilized in any system architecture that supports the execution of code that involves indirect jump/call instructions that provide the address of the jump via a general purpose hardware register. The techniques can be used by the developers as well as by providers that offer software protection (code-protection) solutions.

CONCLUSION

This disclosure describes techniques to perform obfuscation of connections between instructions within executable code without the use of writable-executable memory. To this end, the absolute memory address specified in each jump/call instruction is resolved and stored in

readable memory in an obfuscated format. The corresponding instruction is then replaced with an indirect jump/call instruction that uses the respective obfuscated value of the address. Instructions to deobfuscate the address are added before the execution reaches the jump/call instruction so that the code flow can proceed appropriately to the correct memory address specified in the original unobfuscated instruction. The techniques described in this disclosure can apply to any system architecture that supports the execution of code that involves indirect jump/call instructions that provide the address of the jump via a general purpose hardware register. The techniques can be used by the code developers as well as by third party services that offer code-protection solutions.