ROP (return oriented programming) based code reuse attack on Intel SGX

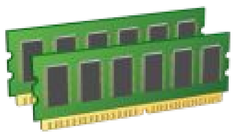
Disaiah Bennett :: Elizabeth City State University :: dlbennett365@students.ecsu.edu
Mentor: Dr. Xiaofeng Wang :: Assistant: Wenhao Wang :: Indiana University School of Informatics & Computing
Summer Research Opportunities in Computing Applications (SROC)

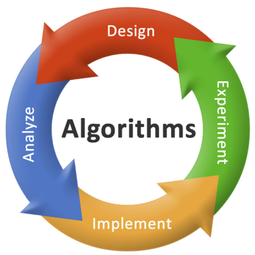
INTRODUCTION

As technology enhances, online security corporations such as Intel, is challenged with engineering systematic software that safeguards private structures. Developing its Software Guard Extension (SGX), Intel SGX is a set of X86 instruction enlargements. With the expansion, isolated domains known as enclaves are utilized to fortify user code and data under a strong adversary model. Even if the entire operating system is compromised, the attacker cannot read/write enclave data, unless they read/write the code running inside the enclave.

OBJECTIVE

The objective of this research project, was to outline an algorithm to construct a code that exploits unwitting software defects. With these inadequacies, Intel SGX can not secure memory from being emanated. Deducing that memory corruption exist, the flaws will be utilized to bypass the security and extract data.



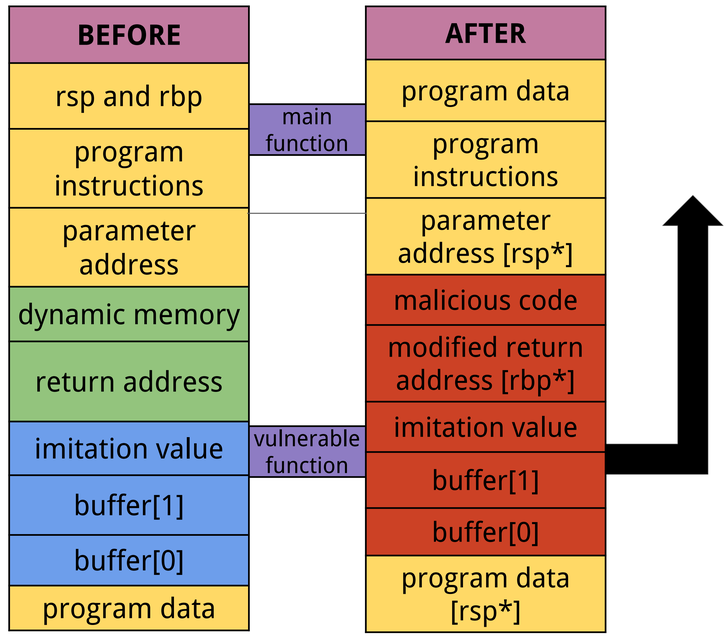


METHODOLOGY

Returned Oriented Programming (ROP)

In order to implement the ROP attack, it is critical to decipher the key elements of the invasion. By eluding the Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP), the attacker can manipulate the system to counteract these defensive software. While effectuating the code, the attacker situates a breaking point in the stack to affix overflowing amounts of data, this is known as a stack buffer overflow.

Utilizing this strategy, the attacker is able to exercise their command over the stack before the return function is achieved. Therefore, the code is transferred to another address in the program to be executed. Now in an executable environment, codes that are undisclosed or visible are easily more accessible for the user.



Intel Software Guard Extension (Intel SGX)

Intel SGX is an application for developers seeking to protect select code and data from disclosure or modification. Intel utilizes enclaves that are protected areas of execution. Application code can be put into an enclaves via special instructions and software made available to developers.

PROCESS

- **Code Blocks: Integrated Development Environment**
  - Open-source cross-platform IDE for C, C++, and Fortran.
- **Cygwin: Shell Programming**
  - Unix based environment, with a bash shell for Microsoft Windows.
- **Immunity Debugger**
  - Powerful way to write exploits, analyze malware, and reverse engineer binary files.

RESULTS

Due to implementing the ROP attack on a X64 system, the system functions were required to be passed through several registers, instead of being linked across the stack. After adjusting and detecting the respectable addresses, the payloads that contained the attack were dispatched at each stopping point that were marked in the general debugger. Upon reaching the break, the malicious code was entered and transported into the registers, but stopped due to the segmentation fault.

FUTURE WORK

By understanding the vulnerability in Intel SGX, future scholars will be able to reinforce and exercise the code to generate further hypotheses. This research encourages more security software to be analyzed and inspected before prototypes are finished. Without examinations, later surveillance productions could jeopardize the user's' systems and private/personal data. It is critical to explore these newer software that are designed for safeguarding memory.

CONCLUSION

The research has shown that the attack is capable of reaching the kernel. However, it is not acknowledged by the core to receive further admission. Before entering the system, an imitation value is noted within the kernel. Unless the two values from the code containing the malware and the system matches, the data that is trying be intercepted is moved around the system constantly to avoid detection.

Acknowledgements: I would like to think the National Science Foundation for their funding of this project. Also Dr. Xiaofang Wang for his mentorship, and Wenhao Wang for his assistance.

National Science Foundation
WHERE DISCOVERIES BEGIN