Side Channel Attacks on Compilers and Defensive Countermeasures

James Van Spyk - Montana State University, Mentors: Dr. Clemente Izurieta, Yvette Hastings

Side Channel Attacks

Side-channel attacks exploit indirect information from computer systems, requiring cyber attackers to infer data from seemingly innocuous sources. This sensitive information can be leaked through various forms, such as, power usage, timing, heat, and memory caches. Despite their subtlety, these attacks pose a significant threat to computer systems, especially to software used in critical infrastructure. This project investigates the vulnerability of computer compilers to side-channel attacks by answering how compilers inadvertently leak sensitive information and what countermeasures can be employed.

Compilers

Compilers typically take human code and converts it into machine readable instructions. The senior capstone course for computer science students “CSCI 468 - Compilers” tasks students with designing a recursive descent parser. The design of the compiler is broken down into 4 segments.

Tokenizer (Lexer)
Takes the source code, which is a stream of characters, and break it down into meaningful tokens. Tokens are the smallest units of syntax for the programming language.

Parser
The parser analyzes the tokens according to the rules of the programming language's grammar to ensure that they form valid expressions or statements.

Evaluation
Evaluation is the process of executing the code represented by the parser. For example the statement produced from the parser “ X = 1 + 1” would then produce the value 2 and would then need to be assigned to X.

Bytecode Generation
Bytecode is a low-level representation of the source code that is easier and faster to execute than the original source. In the below example (figure 1) the bytecode for “X = 1+ 1” is generated

```
iconst_1 //Push int 1
iconst_1 //Push int 1
iadd     //Pop two ints, add and push result
istore_1 //Pop and store as local var 1 (X)
```

Figure 1. Example of Java bytecode generation for the CSCI 468 compiler

Research Methods

The compiler used in the following tests is a recursive descent parser made in “CSCI 468 - Compilers”. The language used is called “Catscript”, a statically typed programing with a fairly small type system defined. More information on the compiler and programing language can be provided upon request.To research data leakage, an investigation into timing of instructions will be necessary. For data collection a variety of compiler actions were executed and timed (recorded in microseconds). 100 executions of the instruction were conducted and the average is compared to other compiler actions. If measurable differences in timing for instructions are found, it is concluded that attackers may be able to infer the design of a program.

Findings

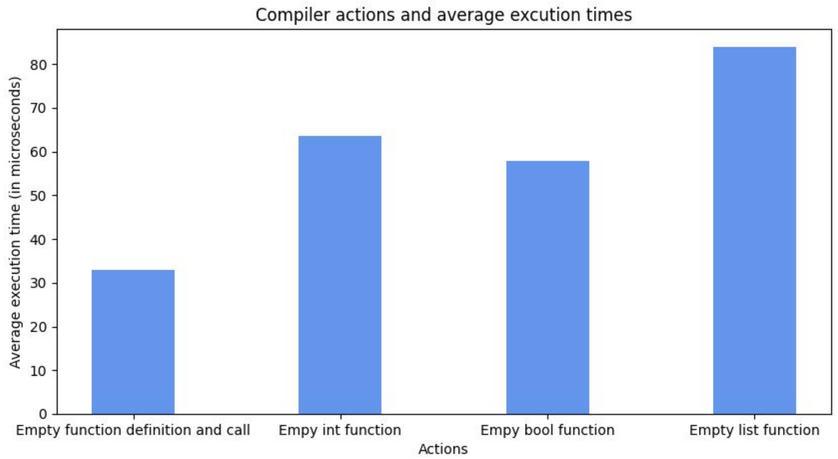

Figure 2. Multiple compiler actions were executed, the average time of 100 iterations were measured in microseconds.

Analysis revealed distinct timing signatures associated with different compiler actions (figure 2). While some actions exhibited relatively consistent execution times across multiple runs, the actions above illustrate how even the return type of a function can be revealed by its execution timing. Particularly, actions involving memory accesses and complex optimizations tended to exhibit more pronounced timing variations compared to simpler operations. Furthermore, certain compiler configurations and optimization levels exacerbated these timing differences, suggesting that program design and compilation design choices play a crucial role in shaping timing behavior.

Countermeasures

There are 3 major approaches to secure compilers to side channel attacks as mentioned by Agosta, Barenghi, and Pelosi in 2020 while studying compiler based techniques to secure embedded software from side channel attacks those were : Hiding, Masking, and Morphing [1].

- **Hiding:** Obscures sensitive information and/or operations to add difficulty to attackers to extract useful information from the side channel leak
- **Masking:** Splits data into multiple partitions and performs computations on these shares such that no single partition would reveal useful side channel information.
- **Morphing:** Changes the behavior of a program to act dynamically with its actions or with its data with changes in runtime conditions or inputs.

Intel also has published its suggestions for best practices for safe development specifically against side channel attacks and those include [2]:

- Using the latest software version
- Limit information in error and debug messages
- Not sharing resources when possible
- Writing constant timing code
- Being aware of compiler optimizations

Acknowledgments

This work is supported by funding for the VICEROY Northwest Institute for Cybersecurity Education and Research (CySER) provided by The Office of the Undersecretary of Defense for Research and Engineering, in collaboration with the Air Force Research Laboratory and Griffiss Institute.

References

[1] “Compiler-Based techniques to secure cryptographic embedded software against Side-Channel attacks,” IEEE Journals & Magazine | IEEE Xplore, Aug. 01, 2020. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8695829

[2] “Security Best Practices for Side Channel Resistance,” *Intel.com*. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/security-best-practices-side-channel-resistance.html#:~:text=Some%20of%20these%20best%20practices,threat%20model%20of%20one's%20applications.