CEN429 Secure Programming Week-13

Tigress and Diversification Techniques

Author: Dr. UÄŸur CORUH

Contents

List of Figures

List of Tables

1 CEN429 Secure Programming

1.1 Week-13

1.1.0.1 Tigress and Diversification Techniques Download

- PDF[1]
- DOC[2]
- SLIDE[3]
- PPTX[4]

1.1.1 Outline

- Tigress and Diversification Techniques
- Obfuscation Methods
- Defense Against Attacks

1.1.2 Week-13: Tigress and Diversification Techniques

This week, we will examine diversification techniques that make code analysis more difficult and make programs more resistant to attacks, as well as obfuscation tools like Tigress. These techniques ensure that each time the program runs, it behaves differently, making it harder for attackers to analyze the program with the same methods.

1.1.2.1 1. Tigress Diversification Theoretical Explanation:

Tigress is a powerful obfuscation tool that transforms a program in various ways to make it more resistant to attacks. Each output of a program creates a unique interpreter, randomizing the program's behavior and making it harder to analyze.

- **Methods Used in Tigress:**

[1] pandoc_cen429-week-13.pdf
[2] pandoc_cen429-week-13.docx
[3] cen429-week-13.pdf
[4] cen429-week-13.pptx

- **Instruction Dispatch Types:**
  - Switch, direct, indirect, call, if-else, linear, binary, interpolation.
- **Operand Types:**
  - Stack, registers.
- **Randomized Operators:**
  - Complex combinations of operands and operators to obfuscate the code.
- **Various Transformations:**
  - **Code Flattening:** Flattening the control flow of the program.
  - **Merge/Split Functions:** Merging or splitting functions.
  - **Opaque Predicates:** Adding hidden and unchangeable conditional statements in the code.

Example:

```
tigress --Transform=Virtualize --Functions=fib --VirtualizeDispatch=switch --out=v1.c test1.c
gcc -o v1 v1.c
```

This transforms the "fib" function into a switch-based virtual machine.

2. Implementing Diversification in Code Theoretical Explanation: Diversification involves randomizing code in different ways to make it harder to analyze. This method makes it difficult for an attacker to reverse-engineer the program. With Tigress, a program can create a unique virtual machine every time it runs.

Techniques for Implementing Code Diversification: Flattening: Mixing the control flow of the program by placing everything in a loop. Function Merging: Combining multiple functions into one. Random Numbers for Diversification: Using random numbers to obfuscate functions and operands. Instruction Dispatch Types: Switch-based Dispatching: Using a switch-case to dispatch virtual machine instructions. Indirect Dispatching: Redirecting branching points indirectly. Example:

```
tigress --Transform=Virtualize --Functions=fib --VirtualizeDispatch=indirect --out=v2.c test1.c
gcc -o v2 v2.c
```

1.1.2.2 3. Attacks and Countermeasures Theoretical Explanation:

An attacker may try to reverse-engineer the virtual instruction set of the program to understand how it operates. There are various attack methods developed for this, but Tigress provides countermeasures against these attacks.

- **Attack Types:**
  - **Attack 1:** Reverse-engineering instructions by interpreting the program.
  - **Attack 2:** Using dynamic attacks, executing the program and observing the virtual program counter (PC) to decipher instructions.
- **Countermeasures:**
  - **Using Complex Semantic Instructions:** Making instruction contents more complex to complicate reverse-engineering.
  - **Using Multiple Program Counters:** Creating multiple PCs in the program, making it harder for attackers to figure out which PC to monitor.

Example:

1. **Attack Scenario:** Reverse-engineering the instruction set of a virtual machine.
2. **Countermeasure:** Implementing multiple virtual machine program counters to complicate reverse-engineering.

1.1.2.3 4. Algorithmic Methods and Implementing Diversification Theoretical Explanation:

Diversification algorithms can be applied at various levels to complicate the program's execution. These methods are used to reduce the likelihood of an attacker successfully reverse-engineering the program.

- **Algorithmic Methods:**
  - **Build-and-Execute:** Building and executing parts of the code during runtime.
  - **Self-Modifying Code:** Algorithms that modify code during runtime.
  - **Encryption:** Encrypting parts of the code and decrypting them at runtime.

- **Moving Code Around:** Shuffling the code to different locations each time it runs.

- **Granularity Levels:**
  - **File Level:** Encrypting or moving the entire file.
  - **Function Level:** Dynamically changing specific functions.
  - **Basic Block Level:** Shuffling the basic blocks of the program.

Example:

```
void makeCodeWritable(caddr_t first, caddr_t last) {
  // Modify code before execution.
}
```

1.1.2.4 5. Self-Modifying Code

Theoretical Explanation: Self-modifying code allows a program to modify itself during execution. This method is used to make it harder for attackers to analyze the code.

- **Kanzaki Algorithm:** Replaces real instructions with fake ones and swaps them back before execution.
- **Madou Algorithm:** Dynamically merges functions and ensures the code changes continuously.

Example:

```
gcc -o v5 v5.c
```

1.1.2.5 Conclusion

This week, we learned about advanced code obfuscation techniques like diversification and self-modifying code. These techniques make programs more resistant to attacks and harder for attackers to reverse-engineer. Tools like Tigress randomize code, creating a different structure every time, making code analysis and reverse-engineering much more difficult.

$$End - Of - Week - 13$$