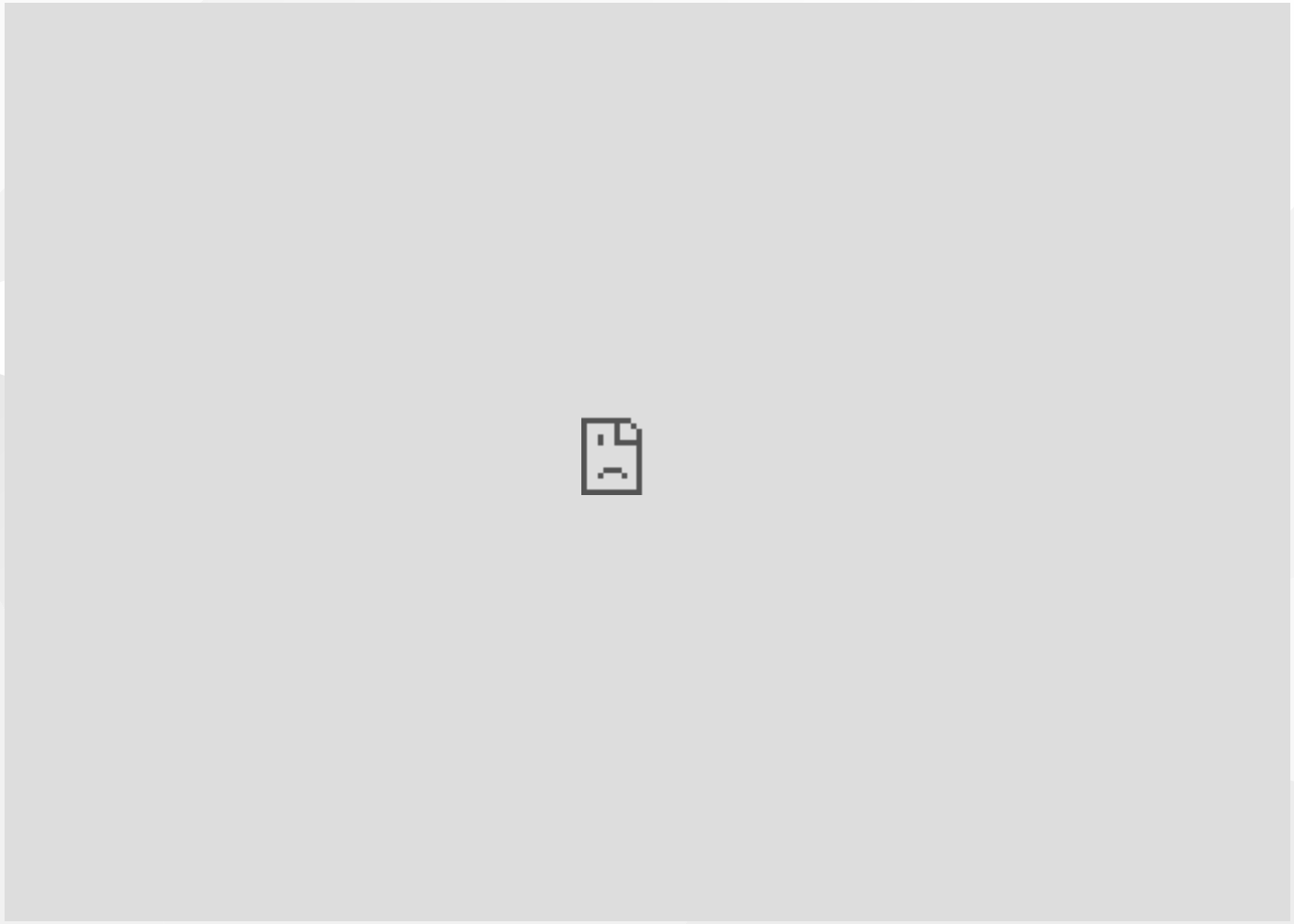
CEN429 Secure Programming

Week-7

Code Obfuscation and Diversification Techniques

Download

- PDF
- DOC
- SLIDE
- PPTX



Outline

- Code Obfuscation and Diversification Techniques
- Static and Dynamic Code Obfuscation
- Virtualization and Encryption

Week-7: Code Obfuscation and Diversification

Code obfuscation and diversification techniques involve making the source code and functions of a program more complex to enhance security. This week, we will explore these techniques and their applications. These methods are crucial for protecting software from reverse engineering and complicating attacks.

1. What is Tigress?

Theoretical Explanation: Tigress is a tool used to transform, obfuscate, and complicate programs. It provides obfuscation techniques that help protect software from reverse engineering. Tigress offers various methods to make code analysis harder.

2. Types of Obfuscation Techniques

Theoretical Explanation: Code obfuscation makes the code difficult to understand for both humans and tools. The following are basic methods of code obfuscation:

- **Abstraction Transformations:** Removing module structures, classes, functions, etc.
- **Data Transformations:** Changing data structures to new representations.
- **Control Transformations:** Eliminating control structures like if, while, repeat, etc.
- **Dynamic Transformations:** Modifying the program at runtime.

3. Static Obfuscation

Theoretical Explanation: Static obfuscation refers to obfuscation that remains fixed during the program's runtime. It alters the structure but does not change dynamically. The following techniques fall under this category:

- **Bogus Control Flow:** Introduces fake control structures, dead branches, and unnecessary branches to complicate the control flow.
- **Control Flow Flattening:** Breaks down control structures, flattening the code.

Application Examples:

1. Adding unnecessary branches and dead code to complicate control flow.
2. Injecting fake operations into functions.

4. Opaque Predicates and Breaking Them

Theoretical Explanation: Opaque Predicates are condition expressions that always evaluate to a fixed value but appear to change from an outsider's perspective. Creating complex mathematical or logical relations for these conditions makes code analysis difficult.

Application Examples:

1. Using **Opaque Predicates** to create fixed conditions.
2. Breaking opaque predicates by using mathematical analysis to resolve these structures.

5. Encoding Integer Arithmetic

Theoretical Explanation: Using complex mathematical transformations to hide original operations on numbers. For example, transforming simple addition into complex mathematical expressions makes reverse engineering more difficult.

Application Examples:

1. Hiding simple arithmetic operations like **x + y** by replacing them with more complex mathematical expressions.
2. Working with transformed numerical operations to reverse the original arithmetic structure.

6. Linear Transformation and Number-Theoretic Tricks

Theoretical Explanation: Linear transformations hide original data through complex mathematical operations. Although these transformations can be reversed, analyzing them is difficult.

Application Examples:

1. Using large modular arithmetic, like Mod 2^32, to apply linear transformations and hide numerical operations.
2. Reversing transformations using mathematical methods like Euclid's Extended Algorithm.

7. Virtualization

Theoretical Explanation: Virtualization involves executing code not directly on the CPU but within a virtual machine (interpreter). This method constantly converts the program at runtime, making reverse engineering much harder.

Application Examples:

1. Running all program commands through an interpreter to hide the original code.
2. Using interpreter-based virtualization to keep the code in a constantly changing state.

8. Diversity

Theoretical Explanation: Diversity involves creating different versions of the same program so that the code doesn't remain fixed. This makes it harder for viruses or malicious software to analyze the code.

Application Examples:

1. Generating different versions of code structures that perform the same function.
2. Making small structural changes in each version of the code to complicate analysis.

9. Encoding and Transforming

Theoretical Explanation: Certain parts of the code can be hidden using special encryption algorithms. This is another obfuscation technique that makes analyzing the code harder. Encoding and transformations can be applied especially to numbers.

Application Examples:

1. Encrypting the numbers used in the code to complicate their analysis.
2. Analyzing the encrypted numbers to reverse them back to their original values.

10. Opaque Expressions and Dynamic Obfuscation

Theoretical Explanation: Opaque expressions involve evaluating certain parts of the code under complex conditions. Dynamic obfuscation includes continuously transforming the code at runtime, keeping it in a constantly changing state.

Application Examples:

1. Applying continuous transformations during execution to make the code harder to analyze.
2. Restructuring the code dynamically during execution to prevent it from remaining fixed.

$$End of Week - 7$$