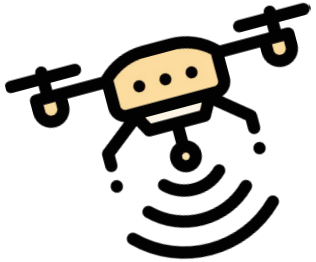
ENGINEERING COLLEGE AJMER
BIKANER TECHNICAL UNIVERSITY, BIKANER

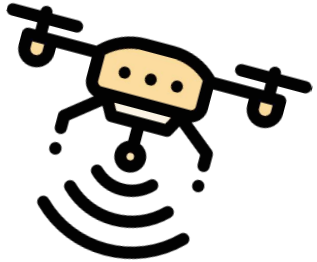
A SEMINAR PRESENTATION ON

Deobfuscating Drone Android Applications through Deep Learning

SUBMITTED IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE OF BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

Deobfuscating Drone Android Applications through Deep Learning





INTRODUCTION

Vulnerabilities In Drone Android Applications In Absence Of Security Measures

01 **Data Theft and Privacy Breach:** In hostile environments, drone applications are susceptible to data theft, enabling malicious actors to intercept and exploit sensitive information such as the drone's location path or acquired data.

02 **Data Tampering and Manipulation**: Malicious entities can tamper with transmitted data, posing risks of altering critical telemetry or mission commands, potentially leading to safety hazards or mission failure in drone operations.

03 **Reverse Engineering and Exploitation:** Hostile actors may reverse engineer drone applications to uncover vulnerabilities, exploiting weaknesses in the source code to gain unauthorized access or control over the drone, compromising its security.

04 **Exposure of Weaknesses of the Application:** Discovery of vulnerabilities in the application's codebase exposes it to potential attacks, enabling adversaries to exploit weaknesses for targeted attacks, jeopardizing the confidentiality and integrity of drone operations.

Safeguarding Techniques For Drone Android Applications

ENCRYPTION

OBFUSCATION

SECURE COMMUNICATION PROTOCOLS

AUTHENTICATION

What Is OBFUSCATION ?

- Obfuscation is a security technique that makes the source code of an application intentionally complex and difficult to understand.
- Obfuscation transforms the code into a complex and unreadable format, which still functions correctly but is hard for humans to interpret.

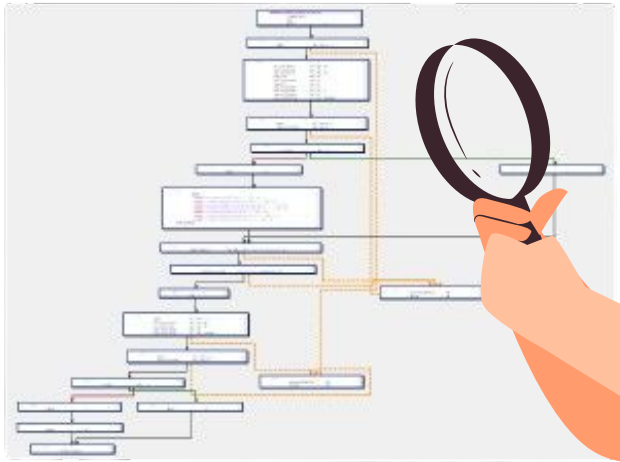
Types Of Obfuscation

01 CONTROL FLOW OBFUSCATION:

Alters the logical flow of the application to make it difficult for attackers to understand the code structure and logic. This involves changing the sequence of operations and making the control flow less predictable.

- Tool Examples: ProGuard, Allatori, DexGuard

Original control flow graph



Obfuscated control flow graph

02 ADDITION OF DEAD CODE:

Inserts redundant or non-functional code into the application. This extra code confuses anyone trying to analyze the application, making it harder to identify the actual logic.

- Tool Examples: Allatori, DexGuard

Adding dead code is like putting extra pieces in a puzzle that don't belong, making it confusing to figure out the whole picture.

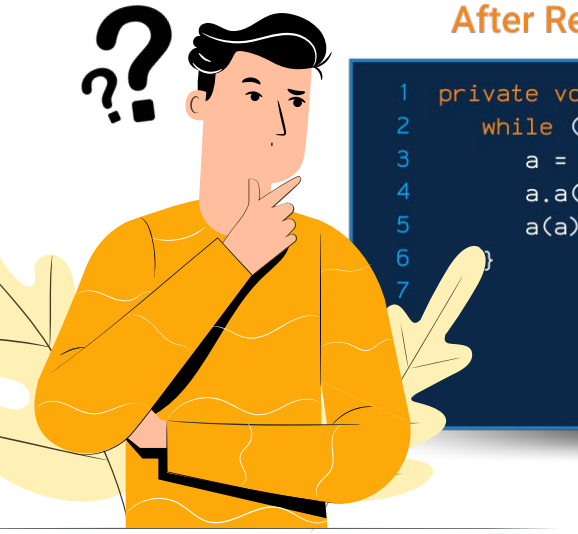
03 IDENTIFIER RENAMING:

Changes the names of variables, methods, and classes to meaningless or random names. This makes it difficult to understand the purpose and functionality of different parts of the code.

- Tool Examples: ProGuard, Allatori, DexGuard

Before Rename Obfuscation:

```
private void
CalcPayroll (SpecialList employee-
Group) {
    while(employeeGroup.HasMore()) {
        employee =
employeeGroup.GetNext(true);
        employee.UpdateSalary();
        DistributeCheck(employee);
    }
}
```



After Rename Obfuscation:

```
private void a(a b) {
    while (b.a()) {
        a = b.a(true);
        a.a();
        a(a);
    }
}
```

There are many obfuscation techniques but the main ones are
Control flow obfuscation
Addition of dead code
Identifier renaming.

Some other techniques include data obfuscation, string encryption, and metadata obfuscation etc.

How Obfuscation Protects From Attackers ?

Obfuscation protects drone Android applications by making the underlying code difficult to understand, preventing unauthorized access, and thwarting reverse engineering attempts. This safeguarding measure enhances security, deters attackers, and preserves the integrity of the application's functionality and proprietary algorithms.

How It Also Make The Drone Android Application Vulnerable To Malwares

- **Exploitation Risk:** Obfuscation can be exploited by attackers to insert malicious code or malware into the application, leveraging the complexity to evade detection.
- **Concealed Malware Insertion**: Attackers can insert malware that retrieves location data, path, or trajectory of the drone, or steal or tamper with data collected by the drone, taking advantage of the obfuscation applied to the Android application.

Deobfuscation

To analyze malware inserted by an attacker, it is necessary to undo the obfuscation and make the source code readable and understandable again. This process, known as Deobfuscation, allows security analysts to thoroughly examine the underlying source code or audit the application's activity logs.

By doing so, they can identify and understand any malicious components that have been concealed through obfuscation, thereby detecting, mitigating, and addressing security threats to ensure the integrity and security of the application.

Objective Of This Project

Analyze and Observe Different Tools of Deobfuscation:

- Study various deobfuscation tools available in the market.
- Evaluate their effectiveness in reversing obfuscated code.

Understand Limitations and Constraints of Each Deobfuscator:

- Identify the strengths and weaknesses of each deobfuscation tool.
- Assess how different tools handle various types of obfuscation techniques.

Raise Awareness About Deobfuscation for Safeguarding Drone Android Applications:

- Educate developers and security professionals on the importance of deobfuscation.
- Promote best practices for using deobfuscation to protect against malware and malicious content.

Deobfuscation Tools Used In This Project

DEGUARD

Statistical Deobfuscation of Android Applications

KATALINA

an open-source Android string deobfuscator

MACNETO

Deobfuscating Android Applications through Deep Learning

DEOPTFUSCATOR

Defeating Advanced Control-Flow Obfuscation Using Android Runtime (ART)

SIMBA

Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions

Assessing Deobfuscation Tools:

A Thorough Examination Of Their Limitations, Advantages, And Disadvantages

DEGUARD Statistical Deobfuscation of Android Applications

Approach :-

The approach to deguard an application obfuscated by ProGuard involves several steps. First, the application is analyzed to generate graphs, including dependency graphs, which illustrate the relationships between different components. Various features and constraints are identified within these graphs to understand the application's structure and behavior. Using this information, unidentified identifiers are predicted, allowing for the deobfuscation process to commence. Finally, these identifiers are renamed appropriately to make the code more readable and understandable.

Limitations of DeGuard

1. **Sophisticated Obfuscation Techniques**:
    - Struggles with advanced methods like control flow obfuscation and string encryption.
    - Challenges in reconstructing complex obfuscated structures.
2. **Scalability and Learning**:
    - Difficulty scaling to larger datasets for probabilistic modeling.
    - Ensuring feature quality and relevance becomes complex as datasets grow.

Pros And Cons Of Deguard

Pros of DeGuard

1. **Effective Reversal of ProGuard Obfuscation**:
   - Accurately predicts many obfuscated elements.
   - Improves understanding of application structure and functionality.
2. **Precision in Library Identification**:
   - High precision and recall in identifying third-party libraries post-obfuscation.
   - Essential for developers and security analysts in understanding codebases and malware.
3. **Utility in Security Analysis**:
   - Reveals sensitive data usage and hidden functionalities in malware.
   - Helps uncover vulnerabilities and understand obfuscated behavior.

Cons of DeGuard

1. **Challenges with Sophisticated Obfuscation**:
   - Limited effectiveness against advanced techniques like control flow obfuscation and string encryption.
2. **Scalability Concerns**:
   - Computational challenges with large datasets.
   - Increased complexity in maintaining accuracy and performance as data grows.

DEOPTFUSCATOR: Defeating Advanced Control-Flow Obfuscation Using Android Runtime

Approach :-

The deobfuscator approach begins by creating a control flow graph (CFG) from the application's DEX file to detect obfuscated variables. This involves profiling to identify these obscure variables accurately. Once detected, the bytecode is modified to clarify these variables, resulting in a modified DEX file. Further optimization is performed using tools like Redex, enhancing the efficiency and readability of the code. The final step is to generate a deobfuscated APK with a clearer control flow, making the application easier to analyze and understand.

Limitations of Deoptfuscator

1. **Narrow Focus**:
   - Primarily effective against DexGuard's control-flow obfuscation.
   - Limited applicability to other obfuscation techniques.
2. **Risk of Errors**:
   - Arbitrary removal of opaque variables can cause errors, especially with anti-tampering mechanisms.
3. **Limited Validation**:
   - Experimentation with a small range of apps and sample sizes may affect result generalizability.

Pros And Cons Of Deoptfuscator

Pros of Deoptfuscator

1. **Effective Control-Flow Handling**:
   - Reduces complexity and restores code structure affected by DexGuard.
2. **Detailed Insights**:
   - Method-by-method similarity analysis facilitates targeted improvements.
3. **Optimization Benefits**:
   - Substantial reduction in obfuscation and bytecode size with ReDex, enhancing performance.
4. **Flexible Deobfuscation**:
   - Adjustable aggressiveness allows customization to user needs.

Cons of Deoptfuscator

1. **Limited Tool Scope**:
   - Ineffective against obfuscation techniques from other tools like DashO or Allatori.
2. **Variable Effectiveness**:
   - Performance may vary with different obfuscation complexities, needing broader validation.
3. **Anti-Tampering Concerns**:
   - Uncertainty about executing deobfuscated apps with anti-tampering protections.

SiMBA : Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions

Approach :-

SiMBASiMBA is a tool for the simplification of linear mixed Boolean-arithmetic expressions (MBAs). Like MBA-Blast and MBA-Solver, it uses a fully algebraic approach based on the idea that a linear MBA is fully determined by its values on the set of zeros and ones, but leveraging the new insights that a transformation to the 1-bit-space is not necessary for this.

Limitations of SiMBA

1. **Complexity for Higher Variables**:
   - Struggles with high-variable expressions, affecting runtime and solution simplicity.
2. **Incomplete Simplification**:
   - May not always find the simplest expression due to refinement limitations.
3. **Verification Challenges**:
   - Computationally intensive to verify simplified expressions, impacting scalability for large datasets.

Pros And Cons Of SiMBA

Pros of SiMBA

1. **Immediate Evaluation**:
   - Faster processing by evaluating the entire linear MBA at once.
2. **Generic Approach**:
   - Applicable to a wide range of scenarios without specific input requirements.
3. **Competitive Runtimes**:
   - Performs well compared to other tools, especially for moderate-variable expressions.

Cons of SiMBA

1. **Limited Scalability**:
   - Challenges with highly complex expressions or large datasets, leading to performance degradation.
2. **Complexity Handling**:
   - May produce suboptimal solutions or longer runtimes for high-variable expressions.
3. **Verification Overhead**:
   - Additional computational overhead for verifying correctness, impacting overall performance and efficiency.

KATALINA: an open-source Android string deobfuscator

Approach :-

Katlalina is a powerful new tool designed to execute Android bytecode, facilitating the deobfuscation of strings in Android malware. Written in Python, it emulates each individual bytecode instruction used in earlier versions of Android. The primary purpose of Katlalina is to emulate functions that conceal strings behind complex logic, significantly reducing the time and effort analysts spend on deobfuscation. The tool is effective against most modern obfuscation techniques.

To use Katlalina, first convert the APK file to a ZIP file and extract all the DEX files. Then, Katlalina can execute the Android bytecode, allowing users to deobfuscate hidden string

Limitations of Katalina

1. **Lack of Multidex Support**:
   - Not suitable for larger Android apps with multiple dex files.
2. **Performance Overhead**:
   - Complex malware can introduce significant performance overhead due to bytecode emulation reliance.
3. **Evolving Obfuscation Techniques**:
   - May struggle with new obfuscation methods in Android malware over time.

Pros And Cons Of Katalina

Pros of Katalina

1. **Accessibility**:
    - Free and open-source, making malware analysis more accessible.
2. **Modular and Flexible**:
    - Customizable deobfuscation process to meet specific needs.
3. **Efficient Malware Analysis**:
    - Emulates most instructions, same-class method invocations, static fields/methods, and string APIs.

Cons of Katalina

1. **Limited Functionality Support**:
    - Issues with multidex files, iterator APIs, array APIs, and cross-class non-static method invocations.
2. **I/O and Windows Support Issues**:
    - May require Windows Subsystem for Linux (WSL) for proper functionality on Windows.

MACNETO: Deobfuscating Android Applications through Deep Learning

Approach :-

The system architecture of MACNETO deobfuscates Android apps through a structured four-stage process. Initially, the instruction distribution stage segments the app's instructions into manageable parts. Following this, the classification stage organizes these segments into predefined categories to streamline subsequent analysis. In the deep learning on machine topics stage, advanced deep learning techniques are applied to these categorized segments, identifying patterns and relationships within the instruction data. Finally, the online scoring stage evaluates the analyzed data in real-time, effectively deobfuscating the app by revealing its original structure and identifying obfuscated elements.

Limitations of MACNETO

1. **Callgraph Dependency**:
    - Ineffective at deobfuscating methods not included in the callgraph.
2. **Graph Diff Module Limitations**:
    - Struggles with obfuscation techniques involving randomly generated methods.

Pros And Cons Of MACNETO

Pros of MACNETO

1. **High Precision Rates**:
   - Achieves high precision, outperforming tools like DeGuard.
2. **Versatility**:
   - Effective against various obfuscation techniques, including lexical obfuscation and control/data transformation.
3. **Advanced Techniques**:
   - Utilizes deep learning and graph diff algorithms to enhance deobfuscation.

Cons of MACNETO

1. **Callgraph Dependency**:
   - Effectiveness is limited when methods are not part of the callgraph.
2. **Handling Certain Obfuscation Techniques**:
   - Challenges with randomly generated methods and complex control flow transformations.

Overall Analysis Of Tools

HOME

1. **DeGuard**:
   - Proficient in reversing ProGuard obfuscation.
   - Accurately predicts third-party libraries.
   - Faces challenges with sophisticated obfuscation and scalability.
2. **Deoptfuscator**:
   - Effectively handles control-flow obfuscation.
   - Optimizes app performance with ReDex.
   - Limited applicability and potential errors require caution.
3. **SiMBA**:
   - Efficient linear MBA computation.
   - Struggles with complexity and verification.
   - Hinders scalability for larger datasets.
4. **Katalina**:
   - Valuable free alternative for malware analysis.
   - Lacks multidex support.
   - May introduce performance overheads.
5. **MACNETO**:
   - High precision in deobfuscating obfuscated programs.
   - Relies on callgraph and faces challenges with certain obfuscation techniques.

CONCLUSION

1. No single deobfuscation method is universally superior.

Each technique has specific strengths and weaknesses tailored to different obfuscation types. Effective deobfuscation often requires combining multiple methods to handle various obfuscation strategies comprehensively.

2. Not every application is obfuscated

Not every application is obfuscated. Many are distributed in a clear, readable format, especially in open-source projects or when transparency and ease of debugging are prioritized. The level of obfuscation varies based on the developer's goals and security concerns.

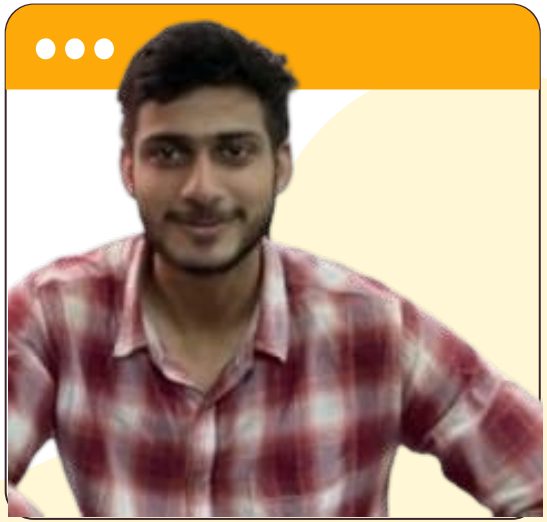
3. Deobfuscators are vital in detecting malware within drone applications

They reverse obfuscation techniques, revealing the original code. This allows analysts to identify and analyze malicious components, ensuring the safety and security of drone operations by uncovering suspicious code patterns and behaviors.

OUR TEAM

Sparsh Bansal
[20IT57]

Rajat Mishra (20IT43)

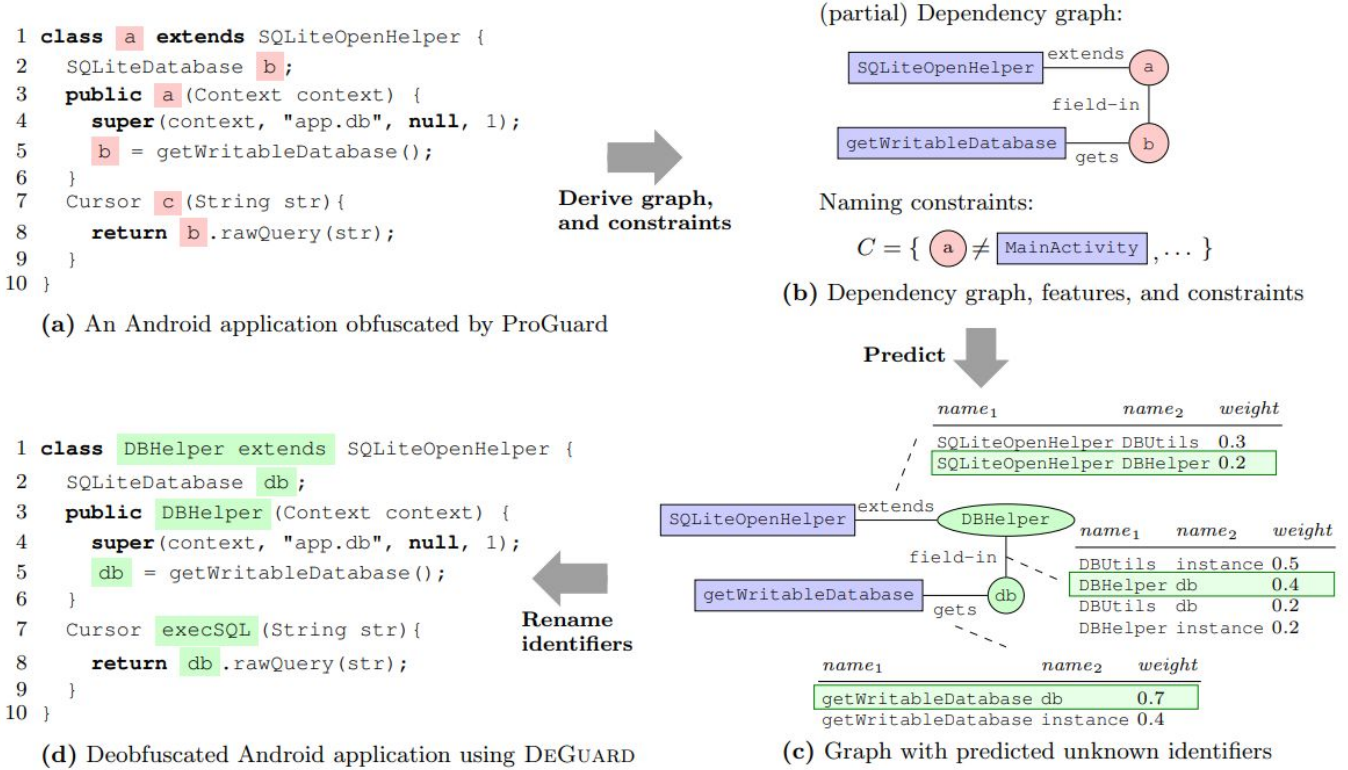


Harsh Yadav (20IT17)

THANK YOU

BACK

Statistical deobfuscation of Android applications using DeGuard. The red color indicates the elements whose names are to be renamed (in the input), the green color are the same elements with the new names (in the output), and the purple color denotes the elements whose names are known and remain the same.



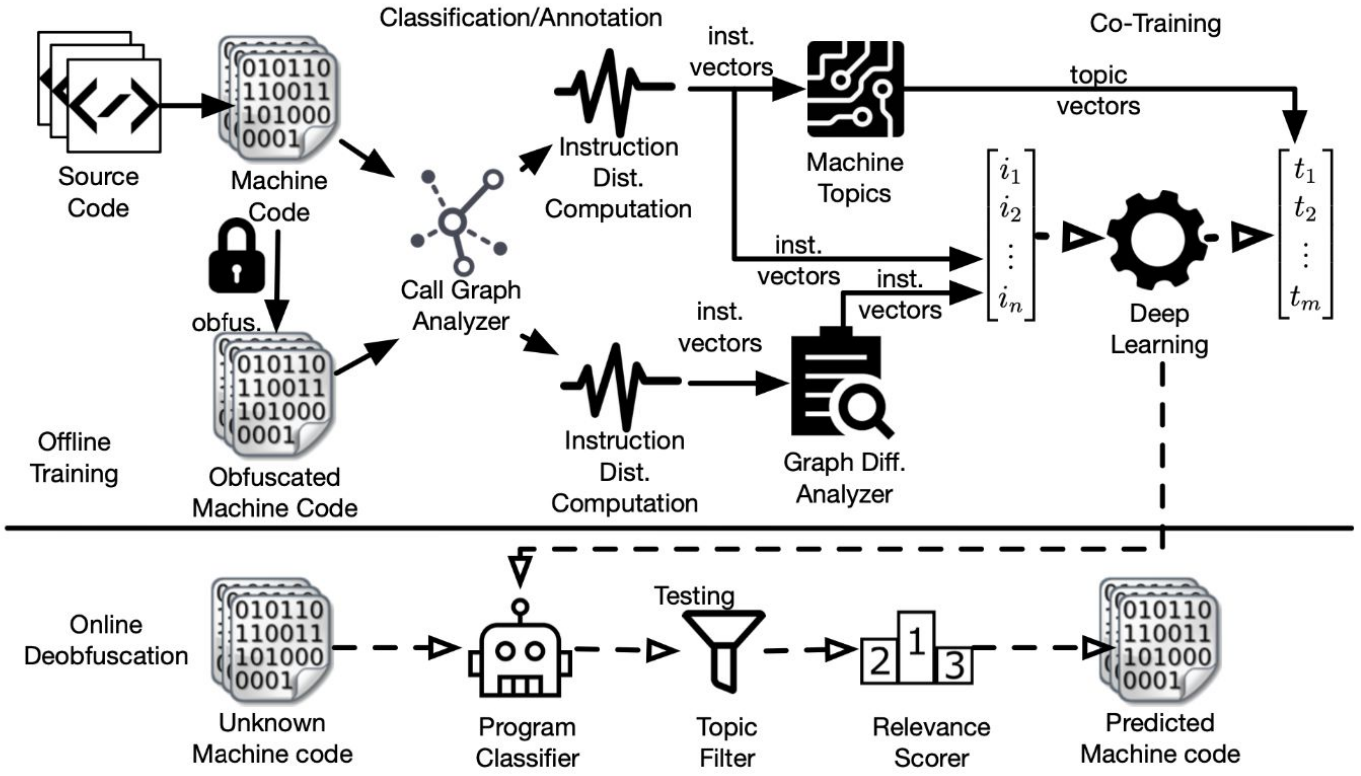
(a) An Android application obfuscated by ProGuard

(b) Dependency graph, features, and constraints

(c) Graph with predicted unknown identifiers

(d) Deobfuscated Android application using DeGuard

The system architecture of MACNETO, which consists of four stages: instruction distribution, classification, deep-learning on machine topics and online scoring, to deobfuscate Android apps.



BACK

Deobfuscation process of deoptfuscator