MARVEL: Repack Me If You Can

An Anti-Repackaging Solution Based on Android Virtualization

UNIVERSITÀ DEGLI STUDI DI GENOVA

Dibris

UNIVERSITÀ DEGLI STUDI DI PADOVA

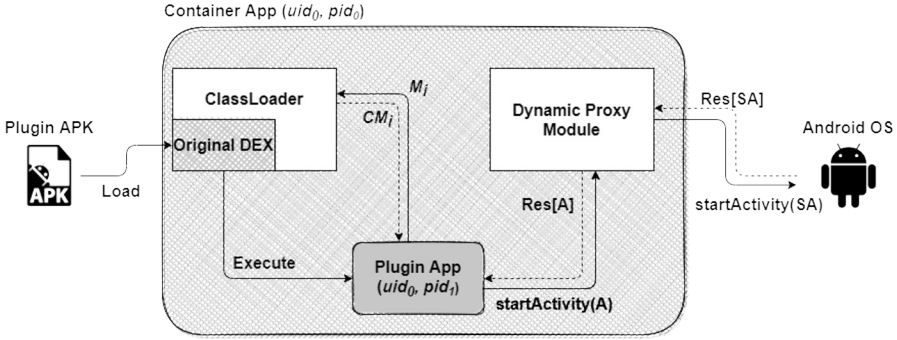
Alessio Merlo
Antonio Ruggia
Luca Verderame
name.surname@dibris.unige.it

Eleonora Losiouk
Mauro Conti
surname@math.unipd.it

Agenda

- Android Virtualization
- Android App-Repackaging
- MARVEL
- MARVELoid
- Experimental results
- Conclusion & Future work

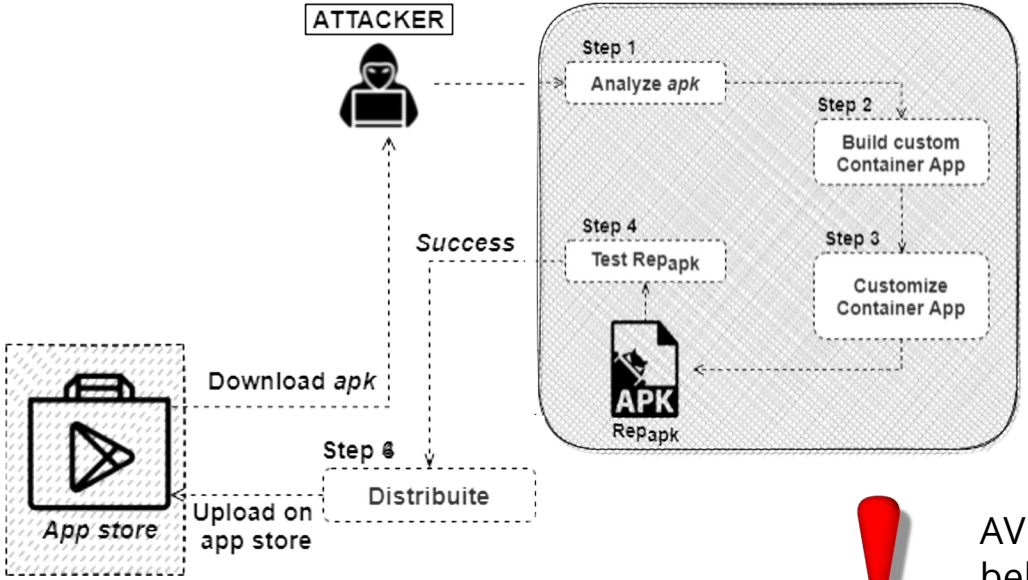
Android Virtualization (AV)



AV allows to execute an Android app (*plugin*) within the context of another app (*container*).

- **Dynamic Code Loading** allows the Java code that is not known about before a program starts
- **Java Reflection** allows a Java program to examine or "introspect" upon itself
- **Java Dynamic Proxy** creates a Proxy object to serve/handle multiple method calls

Android App-Repackaging



AV allows to modify the behavior of the app without repackaging it

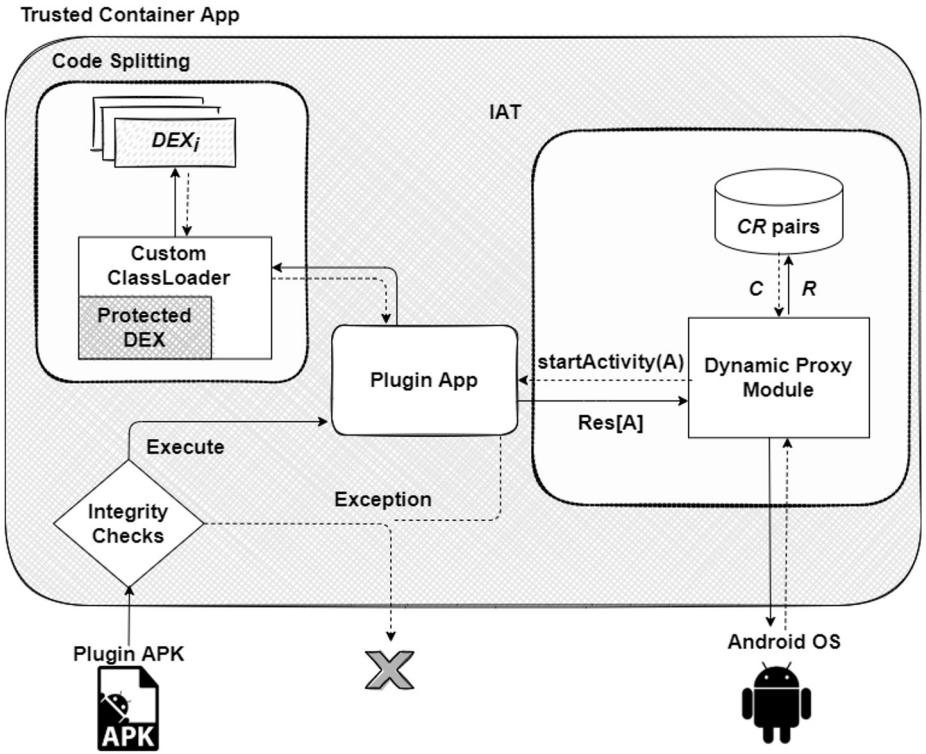
MARVEL ∘ Goal

Mobile-app Anti-Repackaging for Virtual Environments Locking

- (G1) Preventing the attacker from being able to statically analyze an app
- (G2) Preventing an app from being executed in a malicious container
- (G3) Detecting an intermediate malicious container executes a plugin

MARVEL ∘ Overview

- An app can be executed only by the Trusted Container (TC)
- Mutual verification between plugin and TC app
- Code splitting between plugin and TC app



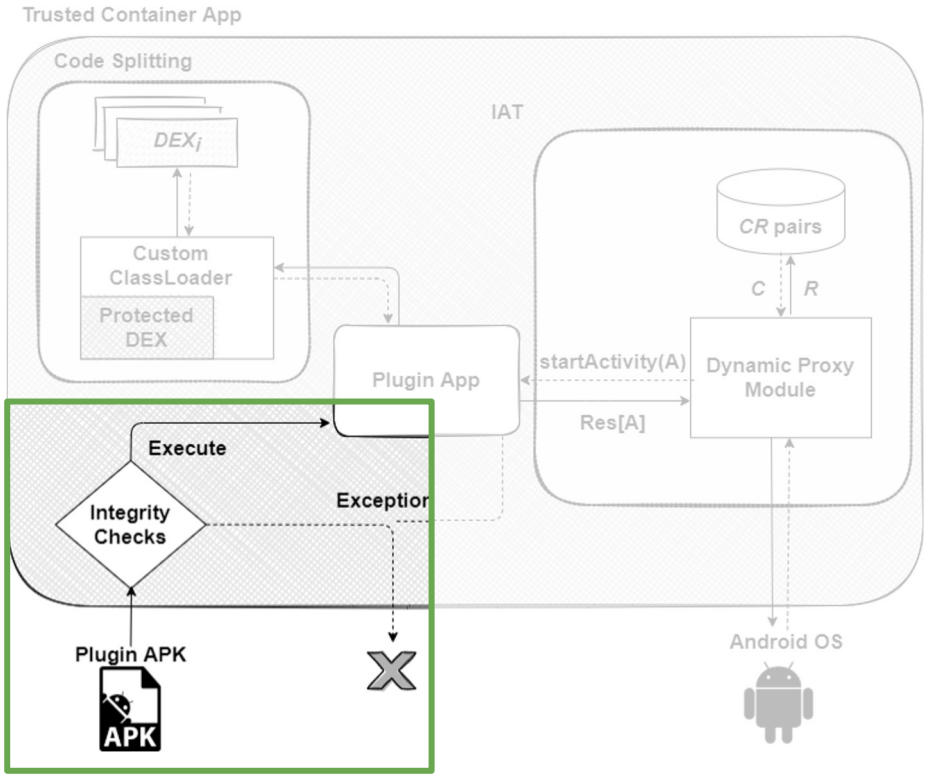
MARVEL ∘ General Purpose Controls

Trusted Container:

Verify the truthfulness and the integrity and the of the plugin app

Plugin App:

Verify the environment (TC)



MARVEL ∘ Code Splitting

Plugin App View

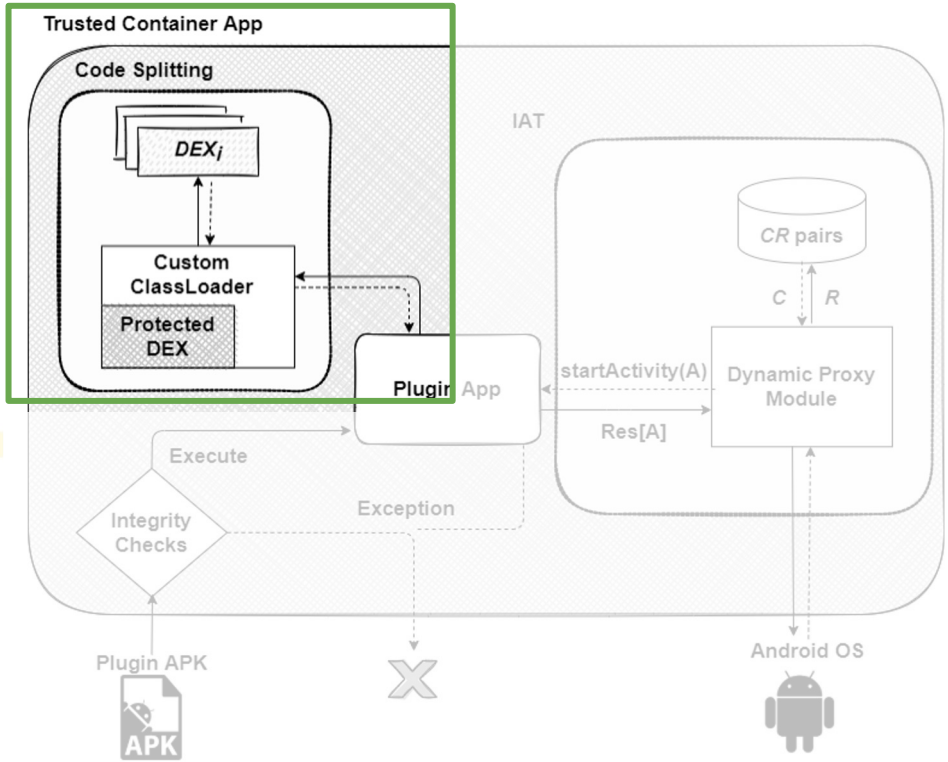
Trusted Container App View (At runtime)

```
public long exampleOfSplittedMethod(int p1, int p2) {
    // Compute p1^p2 in a dummy way!
    long result = p1;
    for (int i = 0; i < p2; i++) {
        result *= p1;
    }
    return result;
}
customHookingReplacingMethods.addMethod(key, hook);
}

if (hook == null) {
    Log.e(TAG, msg: "Cannot find method '"
            + hookMethodName + "' in class '" + hookClassName + "'.");
    throw new NoSuchMethodException("Cannot find method '"
```

Statically, before execution!

```
public long exampleOfSplittedMethod(int p1, int p2) {
    throw new RuntimeException(" Oh no :( ");
}
    targetClass, targetMethodName, targetMethodSignature, hook, backup: null);
return true;
```



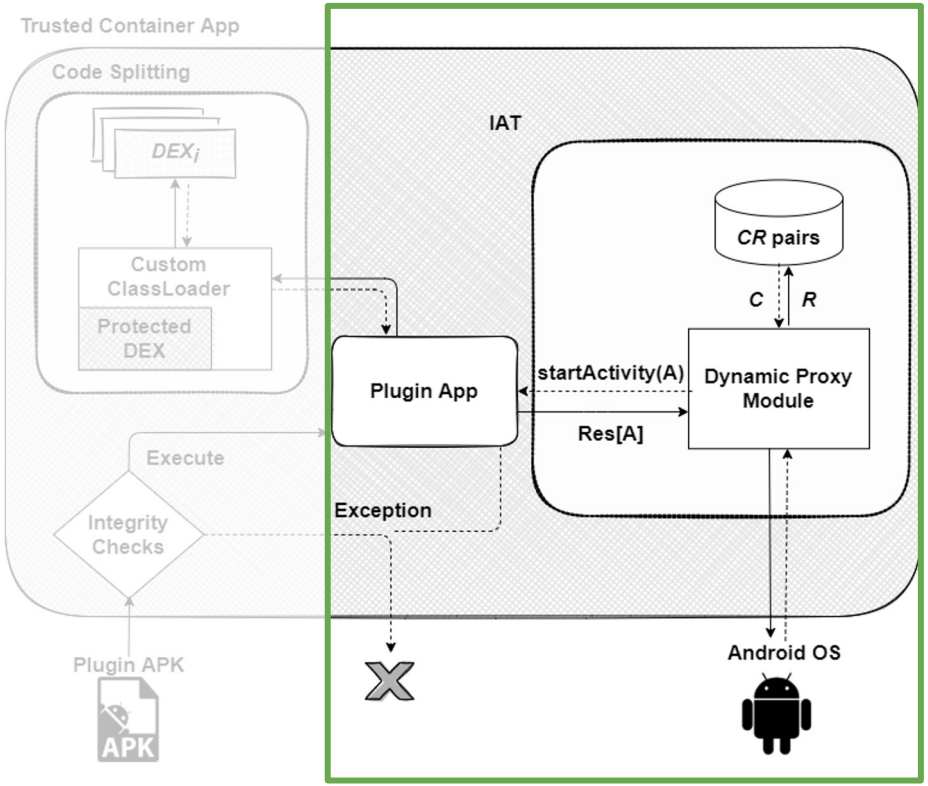
MARVEL ∘ IAT ∘ Interconnected AT

Plugin App View

Trusted Container App View

```
PackageInfo packageInfo = null;
try {
    packageInfo = context.getPackageManager()
            .getPackageInfo(packageName, PackageManager.GET_SIGNATURES);
} catch (PackageManager.NameNotFoundException e) {
    packageInfo = null;
}

// A Real signature check
if (packageInfo.signatures != null && packageInfo.signatures.length > 0) {
    Signature signature = packageInfo.signatures[0];
    byte[] signatureBytes = null;
    MessageDigest md = null;
    try {
        md = MessageDigest.getInstance("SHA");
        md.update(signature.toByteArray());
        signatureBytes = md.digest();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("Error creating the sha hash");
    }
    }
    return info;
}
```



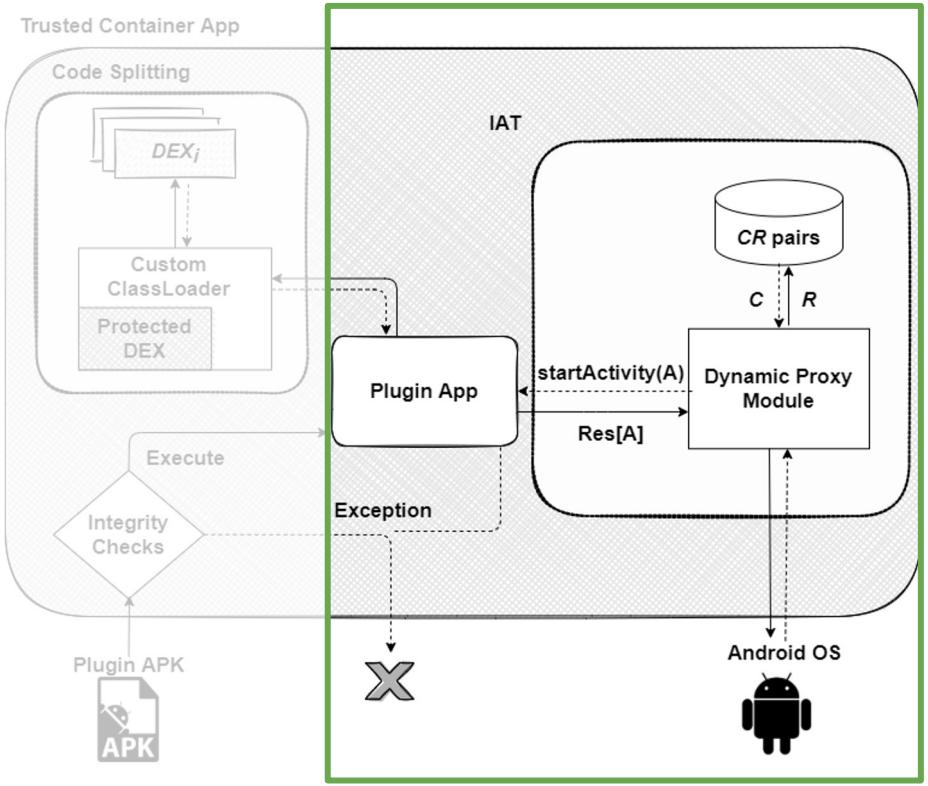
MARVEL ∘ IAT Types

Base IAT

```
public void m_with_baseIAT(String pack) throws Exception {
    PackageInfo pi = context.getPackageManager().getPackageInfo(pack, i: 0);
    if ("precomputed_value".equals(sha256(pi.packageName))) {
        // Execute original code
    }
}
```

IAT with encryption

```
public void m_with_IAT_encryption(String pack) throws Exception {
    PackageInfo pi = context.getPackageManager().getPackageInfo(pack, i: 0);
    String key = pi.sharedUserId;
    byte[] encr_code = "encrypted_original_code".getBytes();
    // try to decrypt and execute the encr_code
    execute(decrypt(encr_code, key));
}
```

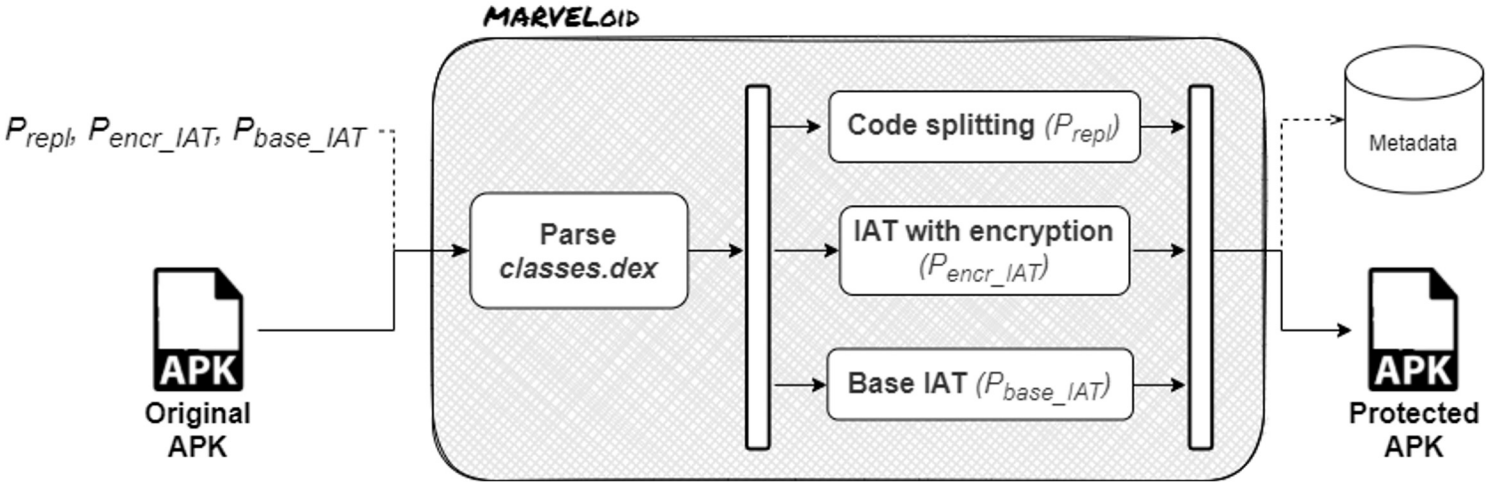


MARVEL ∘ Implementation

- MARVELoid
  - A Java tool to protect Android apps
  - Handles the code splitting and injections of Interconnected Anti-Tampering Controls (IAT).
- Trusted Container
  - A virtualization app that is built on top of the official *VirtualApp* framework
  - Responsible for the enforcement of the MARVEL runtime protection.

The source code is available at: https://github.com/totoR13/MARVEL

MARVELoid


MARVELOID
P_{repl}, P_{encr_IAT}, P_{base_IAT}
APK
Original APK
Parse classes.dex
Code splitting (P_{repl})
IAT with encryption (P_{encr_IAT})
Base IAT (P_{base_IAT})
Metadata
APK
Protected APK

TC ∘ Trusted Container

- IAT has been implemented extending the proxy objects of the container app

- The Java Dynamic Proxy cannot handle code splitting

  Limitation: Proxy objects, which class implements at least one interface

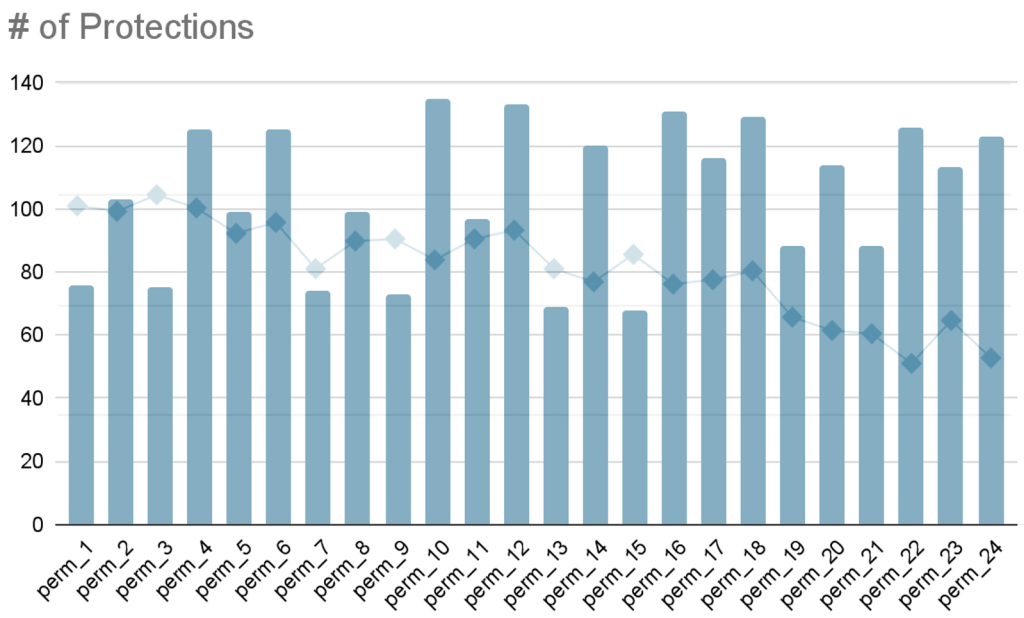
  Solution: ART Instrumentation

  We used: https://github.com/PAGalaxyLab/YAHFA

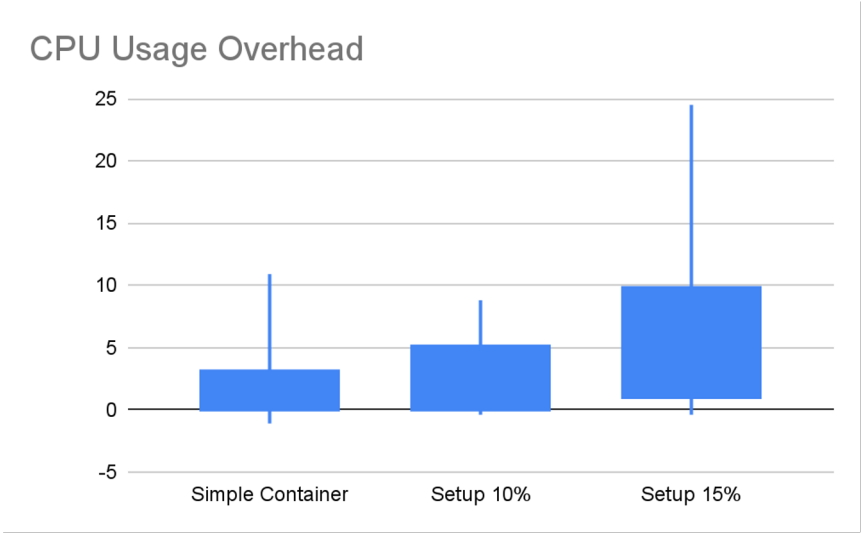
Experimental Campaign

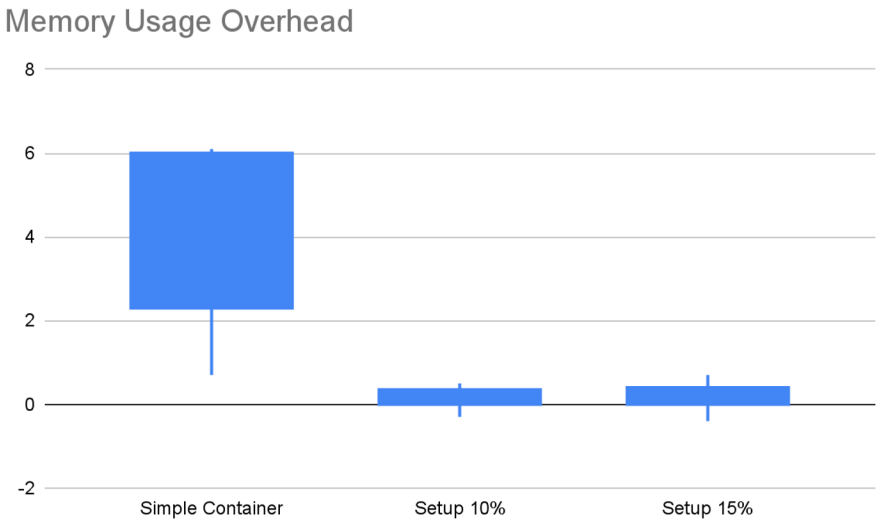
- **Static analysis**
  - *Dataset:* 4000 apps with 24 different combinations of the input percentages
  - *Goal:* Protection and overhead values (e.g., the number of injected protections)
- **Dynamic analysis**
  - *Dataset:* 45 random-selected apps with at least 5.000.000 downloads with 2 combinations of the input parameters
  - *Goal:* Feasibility and runtime overhead

Experimental Campaign ∘ Static Results



Experimental Campaign ∘ Dynamic Results





Limitation & Future Extensions

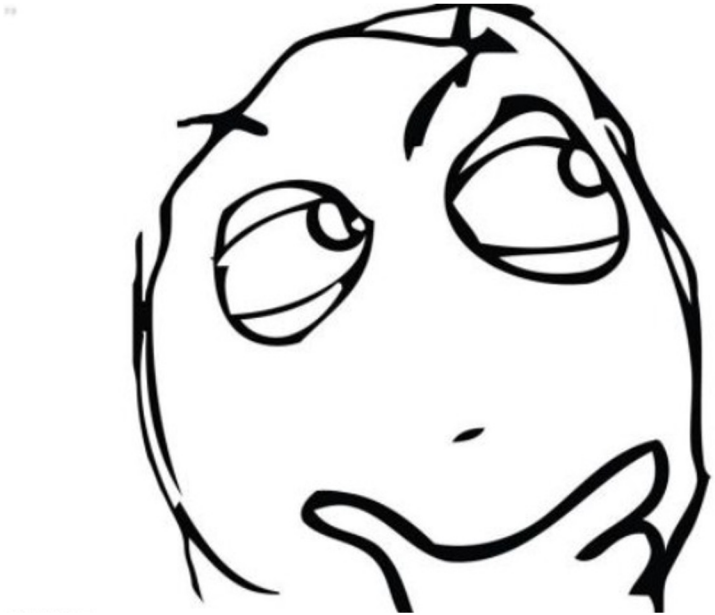
Limitation

- High CPU overhead (highly depends from Code Splitting)

Future Improvements

- Extend the communication mechanisms between TC and plugin app
- Include additional controls inside the IATs
- Enhance the code splitting technique to improve the runtime performances

Question & Answer



Thank you !!!