Use1ess Pwnie
-- COMPSCI 702 G5

Sean Zeng, Tim Koo, Yujun Zhang, Ken Fang

Introduction

01 the app is to encrypt and decrypt messages

02 a backdoor password is needed to decrypt any encrypted messages due to some countries' regulations

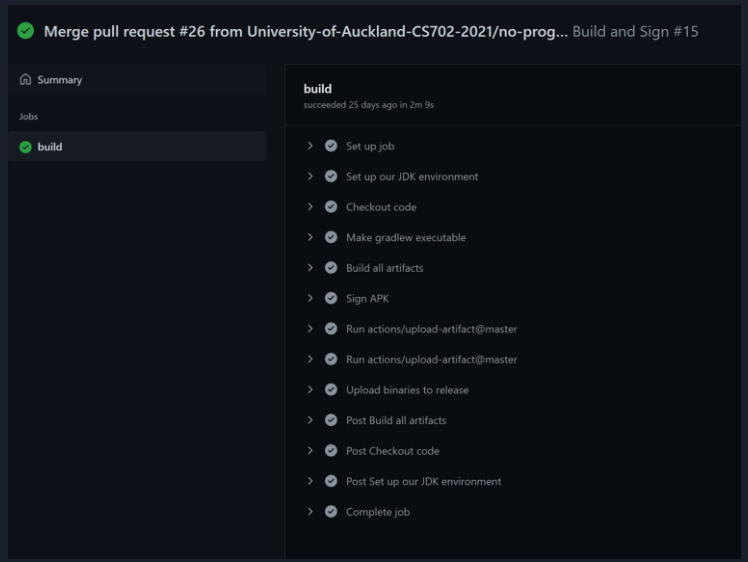
03 a solution is needed to hide the backdoor password

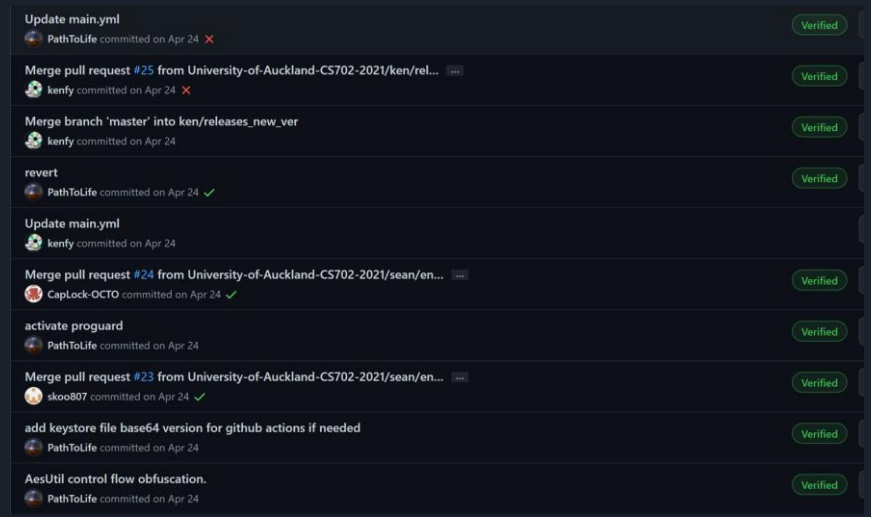
The App

1. Encryption
2. Decryption
3. Share the result of Encryption and Decryption (QR, clipboard, etc)
4. Cryptography libraries

Code together

1. We use github and zoom for collaboration.
2. Using Github Actions, so we can have new Apk to install for every commit.





App Demo - Use1esspwnie

Obfuscation

01 Encryption at rest, Base64, Cesar Cipher

02 Complicated Control Flow Logic

03 Proguard

Obfuscation

01 Encryption at rest, Base64, Cesar Cipher

```
protected static final String BASE_ALGORITHM = "AES";
protected static final String GCM_ALGORITHM =  "AES/GCM/NoPadding";
protected static final String PBE_ALGORITHM = "PBKDF2WithHmacSHA1";
protected static final String ENCODING_CHARSET = "UTF-8";
protected static final int IV_LENGTH_BYTE = 12;
protected static final int TAG_LENGTH_BIT = 128;
protected static final int SALT_LENGTH_BYTE = 16;
```

Before

```
protected static final String _BASE_ALGORITHM = "a299";
protected static final String _GCM_ALGORITHM = "a299WXFtd1l4GXoLDg4TGBE=";
protected static final String _PBE_ALGORITHM = "emx1bnBcARMeEnIXCw19cmtb";
protected static final String _ENCODING_CHARSET = "f35wV2I=";
protected static final String _IV_LENGTH_BYTE = "W1w=";
protected static final String _TAG_LENGTH_BIT = "W1xi";
protected static final String _SALT_LENGTH_BYTE = "W2A=";
```

After

Obfuscation

02 Complicated Control Flow Logic

OVERVIEW

$fn\ C(x) = ASCII + Int,\ Base64\ Cesar\ Cipher$
$fn\ PBE(password,\ salt,\ algorithm) = PBE\ Secret\ Key\ Hashing\ function,\ (10000\ rounds,\ 256\ bits)$
$fn\ E(payload,\ pbeKey,\ iv,\ algorithm) = Encryption\ function,\ AES/GCM/NoPadding$
$fn\ SecureRandom() = Android\ System\ random\ byte\ generation$
$fn\ Base64 = turns\ byte\ array\ into\ base64\ string$

$C(A_{PBE}) = "emx1bnBcARMeEnIXCw19cmtb"$
$C(A_E) = "a299WXFtd1l4GXoLDg4TGBE="$

$A_{PBE} = !C(A_{PBE})$
$A_E = !C(A_E)$
$A_{PBE} = Password\ Based\ Encryption\ Algorithm = "PBKDF2WithHmacSHA1"$
$A_E = Encryption\ Algorithm = "AES/GCM/NoPadding"$

$salt = SecureRandom()$
$iv = SecureRandom()$

$P_{plain} = plain\ text\ password$
$D_{plain} = plain\ text\ payload$
$P_{hash} = PBE(P_{plain},\ salt,\ A_{PBE})$

$B_{plain} = backdoor\ plain\ text\ password,\ not\ stored\ in\ codebase$
$B_{salt} = backdoor\ password\ salt$
$B_{hash} = PBE(B_{plain},\ B_{salt},\ A_{PBE})$

$BackdoorPayload = E(P_{Plain},\ B_{hash},\ iv,\ A_E)$
$EncryptedPayload = E(D_{plain},\ P_{hash},\ iv,\ A_E)$

$Result_{bytes} = \{salt,\ iv,\ BackdoorPayload_{length},\ BackdoorPayload,\ EncryptedPayload\}$
$Result_{string} = Base64(R_{bytes})$

```java
package nz.ac.auckland.cs702.use1esspwnie.lib;

import android.os.Build;
import android.util.Base64;

import java.nio.ByteBuffer;
import java.security.SecureRandom;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.KeySpec;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
```

Obfuscation

02 Complicated Control Flow Logic

HASHING

$$salt = SecureRandom()$$
$$iv = SecureRandom()$$

$$P_{plain} = plain\ text\ password$$
$$D_{plain} = plain\ text\ payload$$
$$P_{hash} = PBE(P_{plain}, salt, A_{PBE})$$

$$B_{plain} = backdoor\ plain\ text\ password,\ not\ stored\ in\ codebase$$
$$B_{salt} = backdoor\ password\ salt$$
$$B_{hash} = PBE(B_{plain}, B_{salt}, A_{PBE})$$

Obfuscation

02

Complicated Control Flow Logic

Proper ENCRYPTION

$$BackdoorPayload = E(P_{Plain}, B_{hash}, iv, A_E)$$
$$EncryptedPayload = E(D_{plain}, P_{hash}, iv, A_E)$$

Obfuscation

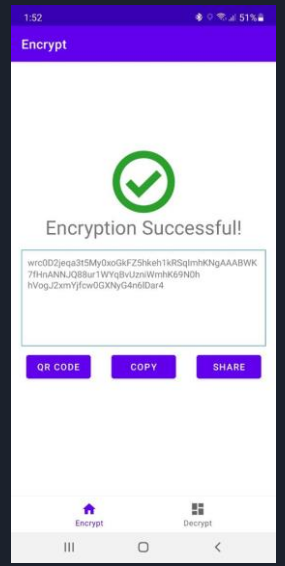
02 Complicated Control Flow Logic

RESULT

$$Result_{bytes} = \{salt,\ iv,\ BackdoorPayload_{length},\ BackdoorPayload,\ EncryptedPayload\}$$
$$Result_{string} = Base64(R_{bytes})$$

```
c2 b7 34 0f 68 de a9 ad ed e4 cc b4 c6 81 a4 15 9e 61 91
e8 75 91 14 aa 22 68 4a 36 00 00 00 15 8a ed f1 e7 00 d3
49 43 cf 2e af 55 98 a8 1b d4 ce 78 96 9a 12 ba f4 dd 21
85 5a 20 27 6c 66 62 37 dc c3 41 97 37 21 b8 9f a9 43 6a
be
```

wrc0D2jeqa3t5My0xoGkFZ5hkeh1kRSqImhKNg
AAABWK7fHnANNJQ88ur1WYqBvUzniWmhK6
9N0hhVogJ2xmYjfcw0GXNyG4n6lDar4



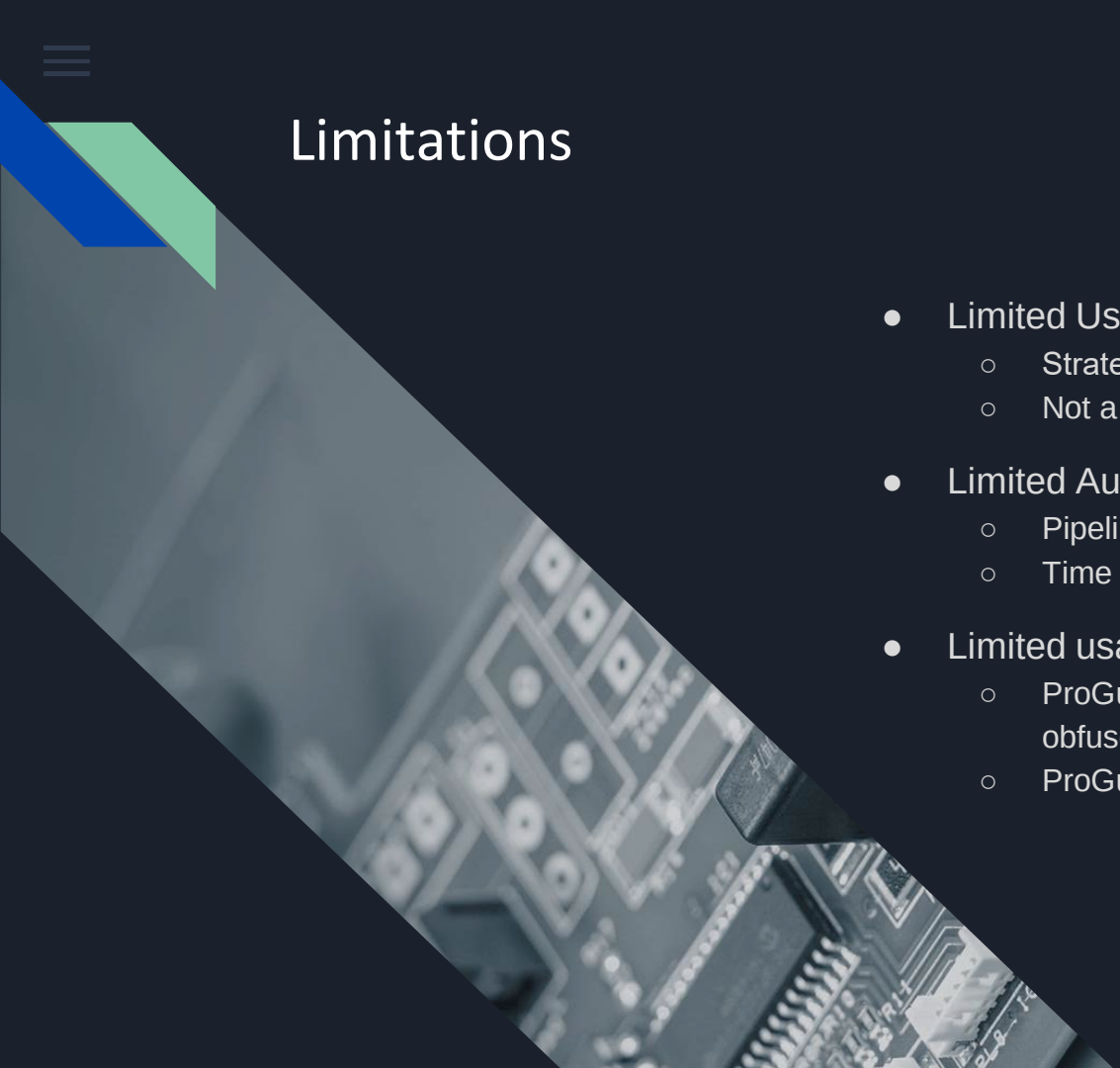
Obfuscation - Final Result

03 Proguard

```
public static void d() {
    if (!a) {
        f1940e = b("a299");
        f1941f = b("a299WXFtd1l4GXoLDg4TGBE=");
        f1942g = b("emx1bnBcARMeEnIXCw19cmtb");
        f1943h = b("f35wV2I=");
        b = c("W1w=");
        f1938c = c("W1xi");
        f1939d = c("W2A=");
        a = true;
    }
}
```

Goal: Hide variable names. Control flow logic is complicated already

Limitations



- Limited Using Scenario
  - Strategy focus on hiding the “payload” and algorithm
  - Not a general solution for other applications
- Limited Automated Solution
  - Pipeline is hard to replicate
  - Time constraint on developing automated solution
- Limited usage of ProGuard
  - ProGuard is only used for automated rename obfuscation
  - ProGuard did provided compile-time obfuscation

Future Approach

- Discover a more generic solution
  - Real-world implementation for mobile utilities Apps
- Automatic obfuscation solution
  - Using parsers and modifiers