MOBILE APP HARDENING

Against Reverse Engineering

Vikas Gupta
Gautam Arvind Pandian

Thales DIS

SINCON 2021, ~~Singapore~~ Virtual

November 2021

About Us

- **Vikas Gupta**
  - Security Researcher and Pentester at *Thales DIS*
  - M.Sc in Information Security, OSCP certified
  - Co-Author contributor for *OWASP Mobile Security Testing Guide (MSTG)*
  - Interests: Reverse Engineering, Obfuscation, Crypto
  - Github: @su-vikas

- **Gautam Arvind Pandian**
  - Security Researcher and Architect at *Thales DIS*
  - CTF creator in r2con2020 conference - R2Pay.
  - Speaker at various conferences - *Android Security Symposium 2020*, *Sincon 2020*
  - Interests: Crypto, Hardening Mobile Apps
  - Github: @darvincisec

Overview

- **Objective:** Harden mobile apps against RE and without using commercial tools
- What is Mobile App Hardening?
- Various app hardening techniques
  - Build Settings
  - Code Hardening
  - Data
  - Cryptography
- Case Study - R2Pay
- Discussion and Conclusion

1. App Hardening
2. Build Settings Techniques
3. Code Hardening Techniques
4. Data Hardening
5. Crypto Hardening
6. Discussion

How to improve the security skills of mobile app developers? Comparing and contrasting expert views

Charles Weir
Security Lancaster
Lancaster University, UK
+44-7876-027350
c.weir1@lancaster.ac.uk

Awais Rashid
Security Lancaster
Lancaster University, UK
+44-1524-510316
a.rashid@lancaster.ac.uk

James Noble
Victoria University
Wellington, NZ
+64-4-4635233
kjx@ecs.vuw.ac.nz

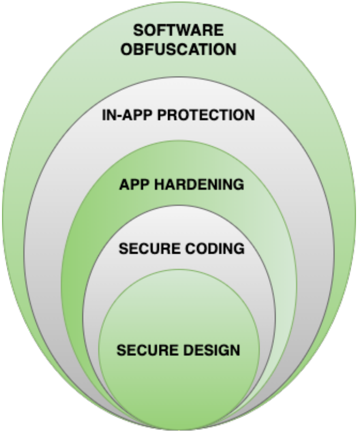
Programmers' lack of knowledge and ability in secure development threatens everyone who uses mobile apps. There's no consensus on how to empower app programmers to get that knowledge. Based on interviews with twelve industry experts we argue that the discipline of secure app development is still at an early stage. Only once industry and academia have produced effective app developer motivation and training approaches shall we begin to see the kinds of secure apps we need to combat crime and privacy invasions.

- "We found little existing work about how programmers learn application security."
- In summary, limited resources to motivate and train developers for application security.

What is App Hardening?

- *App hardening* is used in multiple contexts
- Hardening against exploitation
  - Stack Canaries, PIE code
- Self protection for a mobile app
  - RASP, Obfuscation
- Ensure the security of an app even on a hostile or breached OS/device.
- In this presentation, app hardening against reverse engineering is discussed.
  - Against both static and dynamic analysis.

- Mobile app security can be implemented in a multi-layered approach - onion layers.



- Absence of hardening doesn't make an app insecure.

Who Needs App Hardening?

- Threat modeling is necessary to determine depth of hardening needed.
  - If RE is not a risk, then hardening against RE is not required
- Having critical business assets.
  - Protection of IP
  - Sensitive data
- Providing sensitive services
  - Payment
  - Digital Banking
  - Govt services - Digital Identity (e.g: driving licenses)

- There are multitude of commercial tools
  - In-app protections, Code obfuscation, Symbol obfuscation
  - Arxan, Dexguard, Preemptive etc.
- Before using commercial tools, many techniques can be applied by developers.
- Techniques discussed not always present in commercial tools.
- Goal is to slow down RE attacks
- Most ideas are language agnostic, applicable for all platforms
  - Until mentioned otherwise
  - Many native code techniques applicable to ObjC and Swift code.

Static Analysis

- Understanding working of a binary without running it.
- Tools: Jadx, IDA Pro, Ghidra

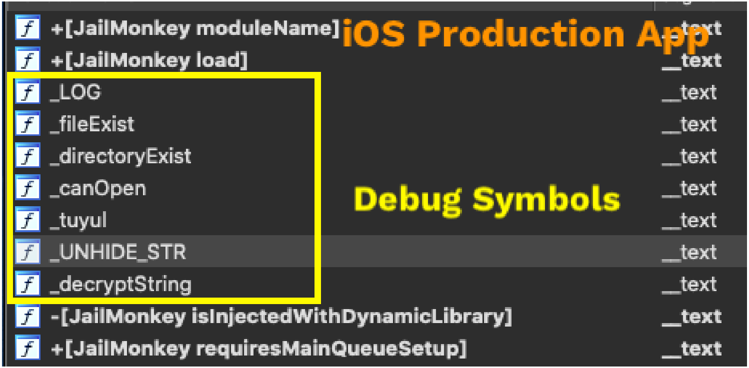
Dynamic Analysis

- Understanding workings of a binary by executing it.
- Tools: Debuggers, FRIDA, EdXposed

1. App Hardening
2. Build Settings Techniques
3. Code Hardening Techniques
4. Data Hardening
5. Crypto Hardening
6. Discussion

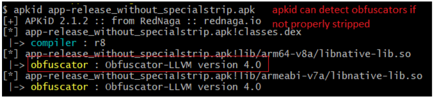
Striping Symbols

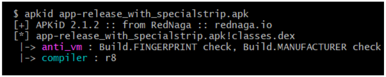
- Remove all the information not needed for execution of the binary
  - Debugging symbols



Striping Symbols

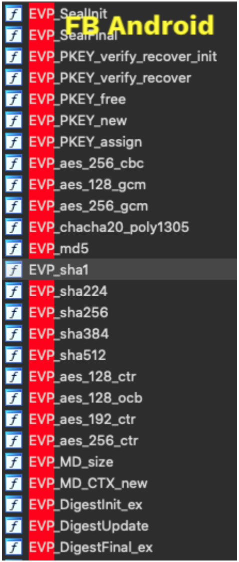
- ELF binary: strip *.comment*, *.strtab*, *.symtab* sections
  - *APKiD* uses information from these sections





Visibility Hidden Flag

- For native code
- Remove symbols that are private to shared library.
- *-fvisibility=hidden* make all symbols hidden by default.
  - Explicitly mark the symbols to be exported by setting visible attribute.



Static Linking Libraries

- Binary merging or having one monolith binary with minimal symbols

- Symbols will be statically linked.
  - Functions called directly using address
  - No exported symbols

- In Android, easier to obfuscate symbols with Progaurd

- Only works if a module’s source code is available.
  - Or compiled as static lib for native code.

Static Linking

- In Java if app is compiled with *BouncyCastle's* code

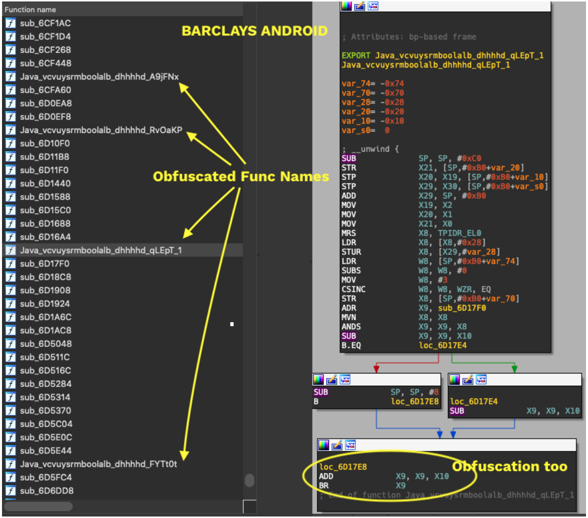
```java
import d.a.a.a.c.c;
import java.nio.ByteBuffer;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import me.zhanghai.android.materialprogressbar.BuildConfig;


 private static byte[] h(a aVar, byte[] bArr, byte[] bArr2) {
     String str = "Hmac" + aVar.toString();
     Mac instance = Mac.getInstance(str);
     instance.init(new SecretKeySpec(bArr, str));
     return instance.doFinal(bArr2);
 }
```

Without Static Linking BouncyCastle

```java
import d.b.a.d.c;
import d.b.a.d.d;
import d.b.a.d.e;
import d.b.a.d.f;
import java.nio.ByteBuffer;

private static byte[] h(b bVar, byte[] bArr, byte[] bArr2) {
    int i = a.f3790a[bVar.ordinal()];
    d.b.a.e.a aVar = i != 1 ? i != 2 ? i != 3 ? null : new d.b.a.e.a(new f()) : new d.b.a.e.a(new e()) : new d.b.a.e.a(new d());
    aVar.d(new d.b.a.f.a(bArr));
    aVar.e(bArr2, 0, bArr2.length);
    byte[] bArr3 = new byte[aVar.c()];
    aVar.a(bArr3, 0);
    return bArr3;
}
```

With Static Linking BouncyCastle

Symbols Obfuscation

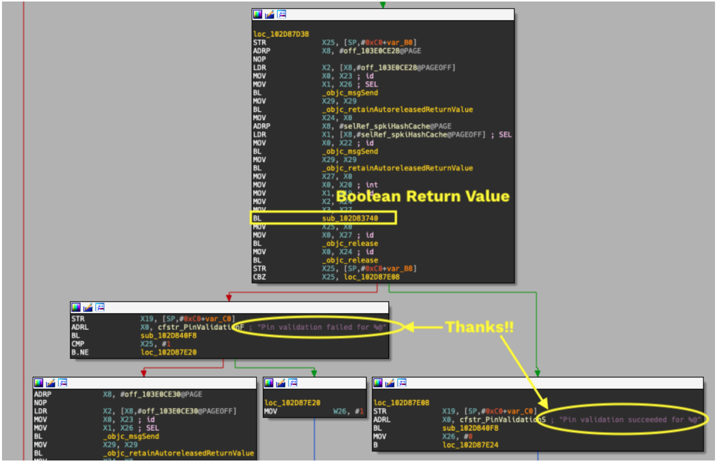


1. App Hardening
2. Build Settings Techniques
3. Code Hardening Techniques
4. Data Hardening
5. Crypto Hardening
6. Discussion

Code Hardening

Non-simple Return Types

- To prevent simple hooking or patching
  - Avoid *Boolean* return values
- Paypal iOS has simple boolean return value
  - For SSL pinning check



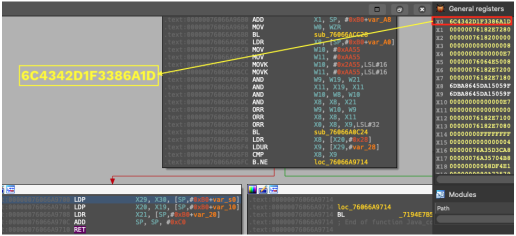
Non-simple Return Types

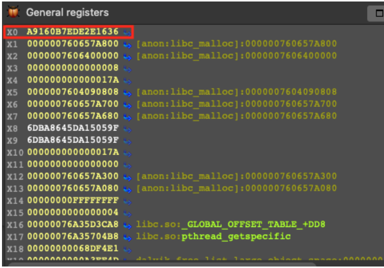
- Bypassing with FRIDA in couple of lines of JS

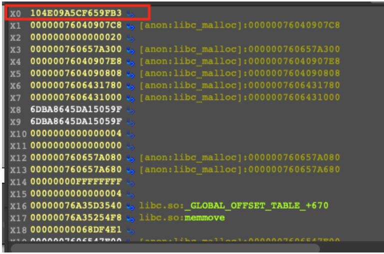
```
Interceptor.attach(
        ObjC.classes.IRoot['- jailbroken'].implementation,
        {
            onLeave: function(retval){
                retval.replace(0x0);
            }
        });
```

Anti-Replay Return Values

- Both, non-simple return type and anti-replay protection







System APIs

- Avoid using libc except for memory management
  - replace libc calls with open source memcpy, memset, string functions and inline them
  - replace file operations with syscalls for critical operations

Code Hardening

```c
__attribute__((always_inline))
static inline
void*  my_memset(void*  dst, int c, size_t n)
{
    char*  q   = (char*)dst;
    char*  end = q + n;
    for (;;) {
        if (q >= end) break; *q++ = (char) c;
        if (q >= end) break; *q++ = (char) c;
        if (q >= end) break; *q++ = (char) c;
        if (q >= end) break; *q++ = (char) c;
    }
    return dst;
}

__attribute__((always_inline))
static inline int
my_strcmp(const char *s1, const char *s2)
{
    while (*s1 == *s2++)
        if (*s1++ == 0)
            return (0);
    return (*(unsigned char *)s1 - *(unsigned char *)--s2);
}
```

Custom impl of libc

```c
__attribute__((always_inline))
static inline int  my_openat(int __dir_fd, const void* __path, int __flags, int __mode ){
    return (int)__syscall4(__NR_openat, __dir_fd, (long)__path, __flags, __mode);
}

__attribute__((always_inline))
static inline ssize_t my_read(int __fd, void* __buf, size_t __count){
    return __syscall3(__NR_read, __fd, (long)__buf, (long)__count);
}

__attribute__((always_inline))
static inline int my_close(int __fd){
    return (int)__syscall1(__NR_close, __fd);
}
```

syscall alternative for libc file operations

System APIs

- Use reflection for accessing Java System APIs such as Keystore.
  - further requires string obfuscation

String Data Structures

- Don't use immutable data structures for storing critical values
    - Memory cannot be zero'd
- Avoid `Strings` in Java, `NSData`/`NSString` in ObjC
    - Need to be dependent on GC or ref counting for variable to be destroyed
- Use byte arrays, as its easy to manipulate the lifetime.
    - XOR masking when kept in memory
    - Zero it when no more used

Reduce Variables Memory Lifetime

- Critical values wiped with 0s or random before *free*-ing the memory
- Example: passwords, keys

Mask Sensitive Values when in Memory

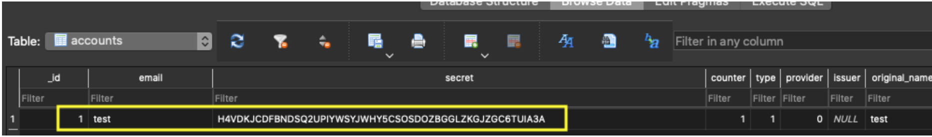
- XOR mask the values.
- Unmask when to be used and mask them back again.
- Example: crypto keys

Data at Rest

- Data at rest gives juicy information whether any sensitive or personal info is stored even if encrypted
- A malware or root application can dump the sandbox data and analyze the contents
- While protecting data at rest, assume sandbox is broken

Data at Rest

- Famous OTP apps such as Google Authenticator, Microsoft Authenticator stores sensitive data in clear.
- **Google Authenticator** SQLite DB.



- *counter* and *secret* for HOTP algo in clear.
- Column names very descriptive as well.

Obfuscated database

_id	_a	_b	_d	_e	_f	_g
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	33d1e99028fc25b7	3a0e3a911aa...	b54406d42ebaa455cdbabc1f723524b7	b54406d42ebaa455cdbab...	ea94f189599658...	da34c0e2-1692-4666-aa9f-...

- Security sensitive apps use certificate pinning to ensure the app is communicating with intended server
  - Easy to bypass certificate pinning in the absence of RASP
  - If Certificate pinning is bypassed, all data in transit is exposed

- Strengthening the obfuscation of data in transit
  - Obscure key names in JSON payload
  - Create a tunnel on top of TLS
    - Provisioning crypto keys, user sensitive information (credit card number) requires high protection

Data in Transit


```
X-Request-Id: c5d7f806-3c82-476c-8969-5649d05ff6c6
Content-Encoding: gzip
Accept-Language: en-US;q=1.0, en;q=0.9
User-Agent: Grab/5.170.0 (Android 10; Build 27844510)
Content-Type: application/json; charset=UTF-8
Content-Length: 1693
Accept-Encoding: gzip, deflate

{
  "adrID":"4490a70621735198",
  "adrIMEI":"4490a70621735198",
  "adrIMSI":"",
  "adrMEID":"",
  "adrSERIAL":"unknown",
  "adrUDID":"76d85db4-a9ce-42b9-8114-c0112b97260b",
  "advertisingID":"0979449c-e32f-49e1-86bd-2bb538703dca",
  "applicationVersion":"5.170.0(51700000) Build ; Build 27844510",
  "challengeID":"090eae8b-5c07-4db2-bd69-df9c667ec6fb",
  "cli":"",
  "countryCode":"in",
  "deviceManufacturer":"Google",
  "deviceModel":"Pixel",
  "iosUDID":"",
  "latitude":0.0,
  "longitude":0.0,
  "otp":"356512",
  "phoneNumber":"919845048726",
  "source":"android2",
  "sourceID":"",
  "tmSessionID":"",
  "tpToken":{
    "type":"Google",
    "value":"eyJhbGci0iJSUzI1NiIsImtpZCI6ImFkZDhjMGV1NjIz0TU0NGFmNTNm0TM3MTJhNTdiMmUyNmY5NDMzNTIiLCJ0eXAi0iJKV1QifQ.eyJpc3Mi0iJodHRwczovL2FjY291bnRzLmdvb2dsZS5jb20iLC
    YXV0YW0gQXJ2aW5kIiwicGljdHVyZSI6Imh0dHBz0i8vbGgzLmdvb2dsZXVzZXJjb250ZW50LmNvbS9hL0FBVFhBSndqU0dLSEVpS19VR0pBTFdLRmJYY1ZFbGpRdFNKZ1NXTW9kYmttPXM5NiljIiwiZ2l2ZW5fbm
    cFvsVd-IEpmhyBQPwlfs3ZP4gbRJzHQ4xcsXPGsj4NbYco17QFlJarfpo0aEDsu2FPaGAQFBhbVTlDLRPlvoZ5mkTIPHbnZV049Iir9h08e_MF5AnvnLGA"
  }
```


MITM can fetch sensitive data

Inline sensitive functions

- Inline sensitive functions
  - No separate function in final binary
  - Xref analysis won't work either
- Compiler directive `__attribute__((always_inline))` to inline sensitive functions in C

System APIs

- Avoid using system APIs for crypto
  - Leaks info on crypto mechanisms with simple static analysis
  - Use 3rd party libraries, and statically link them
- On iOS, use *OpenSSL* or other libs, build with source code.
  - Instead of using *CryptoKit*, *CCCrypt* APIs
  - Link statically

Crypto


```
loc_1EF24
ADRP    X1, #aMbedtls@PAGE ; "MBEDTLS"
ADRP    X2, #aIEncryptedInto@PAGE ; "[i] Encrypted into buffer:"
ADD     X1, X1, #aMbedtls@PAGEOFF ; "MBEDTLS"
ADD     X2, X2, #aIEncryptedInto@PAGEOFF ; "[i] Encrypted into buffer:"
MOV     W0, #2
BL      .__android_log_print
ADD     X0, SP, #0x2F0+var_1E8
BL      .mbedtls_gcm_init
ADD     X0, SP, #0x2F0+var_1E8
ADD     X2, SP, #0x2F0+var_200
MOV     W1, #2
MOV     W3, #0x80
BL      .mbedtls_gcm_setkey
ADD     X0, SP, #0x2F0+var_1E8 ; int
ADD     X2, SP, #0x2F0+var_210
MOV     W1, #1
MOV     W3, #0x10
MOV     X4, XZR
MOV     X5, XZR
BL      .mbedtls_gcm_starts
ADD     X0, SP, #0x2F0+var_1E8 ; int
ADD     X3, SP, #0x2F0+var_250
MOV     W1, #0x20 ; ' '
MOV     X2, X19
ADD     X24, SP, #0x2F0+var_250
BL      .mbedtls_gcm_update
ADD     X0, SP, #0x2F0+var_1E8
BL      .mbedtls_gcm_free
ADRL    X0, aHelloWorld ; "hello world"
MOV     W1, #0xC
BL      .__strlen_chk
CBZ     X0, loc_1F008
```

Symbols show up when crypto library is not built with visibility-hidden

```
loc_9374
ADRP    X1, #aMbedtls@PAGE ; "MBEDTLS"
ADRP    X2, #aIEncryptedInto@PAGE ; "[i] Encrypted into buffer:"
ADD     X1, X1, #aMbedtls@PAGEOFF ; "MBEDTLS"
ADD     X2, X2, #aIEncryptedInto@PAGEOFF ; "[i] Encrypted into buffer:"
MOV     W0, #2
BL      .__android_log_print
ADD     X0, SP, #0x2F0+var_1E8
BL      sub_28070
ADD     X0, SP, #0x2F0+var_1E8
ADD     X2, SP, #0x2F0+var_200
MOV     W1, #2
MOV     W3, #0x80
BL      sub_280AC
ADD     X0, SP, #0x2F0+var_1E8 ; int
ADD     X2, SP, #0x2F0+var_210
MOV     W1, #1
MOV     W3, #0x10
MOV     X4, XZR
MOV     X5, XZR
BL      sub_282B4
ADD     X0, SP, #0x2F0+var_1E8 ; int
ADD     X3, SP, #0x2F0+var_250
MOV     W1, #0x20 ; ' '
MOV     X2, X19
ADD     X24, SP, #0x2F0+var_250
BL      sub_285C0
ADD     X0, SP, #0x2F0+var_1E8
BL      sub_28AF0
ADRL    X0, aHelloWorld ; "hello world"
MOV     W1, #0xC
BL      .__strlen_chk
CBZ     X0, loc_9458
```

Crypto symbols are stripped out

System APIs

- On Android use BouncyCastle and build with source code
  - Instead of using *java.crypto* APIs
  - Call BC APIs directly, instead of via *Java Provider* route

Uses System APIs

```java
private static byte[] generateHash(HashAlgorithm algorithm, byte[] key,
        throws NoSuchAlgorithmException, InvalidKeyException {
    String algo = "Hmac" + algorithm.toString();

    Mac mac = Mac.getInstance(algo);
    mac.init(new SecretKeySpec(key, algo));

    return mac.doFinal(data);
}
```

Cannot be obfuscated

```java
import d.a.a.a.c.c;
import java.nio.ByteBuffer;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import me.zhanghai.android.materialprogressbar.BuildConfig;


 private static byte[] h(a aVar, byte[] bArr, byte[] bArr2) {
     String str = "Hmac" + aVar.toString();
     Mac instance = Mac.getInstance(str);
     instance.init(new SecretKeySpec(bArr, str));
     return instance.doFinal(bArr2);
 }
```

Without Static Linking BouncyCastle

Uses Bouncycastle APIs

```java
private static byte[] generateHash(HashAlgorithm algorithm, byte[] key, byte[] data) {
    HMac hMac = null;
    byte[] out;
    switch(algorithm) {
        case SHA1:
            hMac = new HMac(new SHA1Digest());
            break;
        case SHA256:
            hMac = new HMac(new SHA256Digest());
            break;
        case SHA512:
            hMac = new HMac(new SHA512Digest());
            break;
    }
    hMac.init(new KeyParameter(key));
    hMac.update(data, i: 0, data.length);
    out = new byte[hMac.getMacSize()];
    hMac.doFinal(out, i: 0);
    return out;
}
```

Can be obfuscated

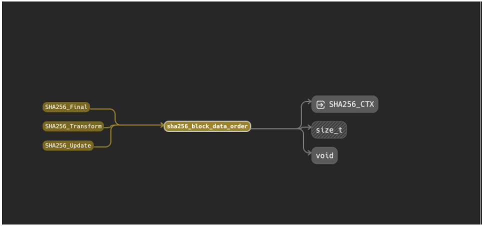
```java
import d.b.a.d.c;
import d.b.a.d.d;
import d.b.a.d.e;
import d.b.a.d.f;
import java.nio.ByteBuffer;

private static byte[] h(b bVar, byte[] bArr, byte[] bArr2) {
    int i = a.f3790a[bVar.ordinal()];
    d.b.a.e.a aVar = i != 1 ? i != 2 ? i != 3 ? null : new d.b.a.e.a(new f()) : new d.b.a.e.a(new e()) : new d.b.a.e.a(new d());
    aVar.d(new d.b.a.f.a(bArr));
    aVar.e(bArr2, 0, bArr2.length);
    byte[] bArr3 = new byte[aVar.c()];
    aVar.a(bArr3, 0);
    return bArr3;
}
```

With Static Linking BouncyCastle

Crypto Constants

- Constants are used by crypto algorithms.
- SHA256 uses the following as one of the constant

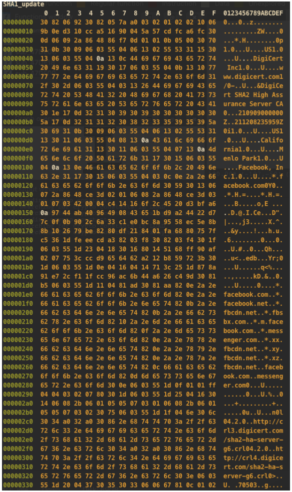
```
#ifndef SHA256_ASM
static const SHA_LONG K256[64] = {
    0x428a2f98UL, 0x71374491UL, 0xb5c0fbcfUL, 0xe9b5dba5UL,
    0x3956c25bUL, 0x59f111f1UL, 0x923f82a4UL, 0xab1c5ed5UL,
    0xd807aa98UL, 0x12835b01UL, 0x243185beUL, 0x550c7dc3UL,
    0x72be5d74UL, 0x80deb1feUL, 0x9bdc06a7UL, 0xc19bf174UL,
    0xe49b69c1UL, 0xefbe4786UL, 0x0fc19dc6UL, 0x240ca1ccUL,
    0x2de92c6fUL, 0x4a7484aaUL, 0x5cb0a9dcUL, 0x76f988daUL,
    0x983e5152UL, 0xa831c66dUL, 0xb00327c8UL, 0xbf597fc7UL,
    0xc6e00bf3UL, 0xd5a79147UL, 0x06ca6351UL, 0x14292967UL,
    0x27b70a85UL, 0x2e1b2138UL, 0x4d2c6dfcUL, 0x53380d13UL,
    0x650a7354UL, 0x766a0abbUL, 0x81c2c92eUL, 0x92722c85UL,
    0xa2bfe8a1UL, 0xa81a664bUL, 0xc24b8b70UL, 0xc76c51a3UL,
    0xd192e819UL, 0xd6990624UL, 0xf40e3585UL, 0x106aa070UL,
    0x19a4c116UL, 0x1e376c08UL, 0x2748774cUL, 0x34b0bcb5UL,
    0x391c0cb3UL, 0x4ed8aa4aUL, 0x5b9cca4fUL, 0x682e6ff3UL,
    0x748f82eeUL, 0x78a5636fUL, 0x84c87814UL, 0x8cc70208UL,
    0x90befffaUL, 0xa4506cebUL, 0xbef9a3f7UL, 0xc67178f2UL
};
```




```
static void sha256_block_data_order(SHA256_CTX *ctx, const void *in,
                                    size_t num)
{
    unsigned MD32_REG_T a, b, c, d, e, f, g, h, s0, s1, T1, T2;
    SHA_LONG X[16], l;
    int i;
    const unsigned char *data = in;

    while (num--) {

        a = ctx->h[0];
        b = ctx->h[1];
        c = ctx->h[2];
        d = ctx->h[3];
        e = ctx->h[4];
        f = ctx->h[5];
        g = ctx->h[6];
        h = ctx->h[7];

        for (i = 0; i < 16; i++) {
            (void)HOST_c2l(data, l);
            T1 = X[i] = l;
            T1 += h + Sigma1(e) + Ch(e, f, g) + K256[i];
            T2 = Sigma0(a) + Maj(a, b, c);
```

Crypto - Crypto Constants

- FB app uses SHA1 and SHA256 in libcoldstart.so
- Can help us in finding the API entry points, or critical sections of the algorithm



- Obfuscate crypto constants with simple XOR masks.

- Perform critical code in native, harder to reverse
- Use *Openssl* or *mbedtls*
  - Use *-fvisibility=hidden*
  - Use static linking
- Diffing or pattern matching can help in reversing popular crypto libraries
  - Crypto constants can easily indicate the matching functions
  - Use OLLVM to obfuscate the control flow and protect the strings

Crypto

Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary	Co	Algorithm	Matched B	Basic Block	Basic Block	Matched Inst	Instructions F	Instruct
1.00	0.99	-------	0003E304	mbedtls_gcm_finish	00028754	sub_00028754		Edges Flow Graph MD Index	15	15	15	117	117	117
0.97	0.99	G---E-C	0001C810	.mbedtls_gcm_auth_decrypt	000289B4	sub_000289B4		Call Reference	16	17	16	79	83	79
0.97	0.99	G---E--	0001D520	.mbedtls_gcm_starts	000282B4	sub_000282B4		Call Reference	23	24	23	125	129	125
0.97	0.99	G---E--	0001CFC0	.mbedtls_gcm_update	000285C0	sub_000285C0		Call Reference	24	25	24	101	105	101
0.96	0.99	G---E-C	0001D6A0	.mbedtls_gcm_setkey	000280AC	sub_000280AC		Call Reference	10	11	10	130	134	130
0.94	0.99	G---E--	0001CBB0	.mbedtls_gcm_crypt_and_tag	00028928	sub_00028928		Call Sequence (Sequence)	19	21	19	152	160	152
0.83	0.96	G---E--	0001D6B0	.mbedtls_gcm_free	00028AF0	sub_00028AF0		Call Reference	6	8	6	19	27	19
0.59	0.98	G---E--	0001D8C0	.mbedtls_gcm_init	00028070	sub_00028070		Call Reference	1	2	1	15	19	15

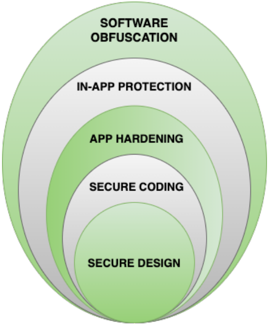
Is RASP enough?

- Often the mobile app hardening discussion starts and stops at RASP

- RASP is a good starting point.
  - But not necessarily always sufficient.
  - Depends on the threat model.

- RASP techniques are often standard and known
  - Without additional hardening, easy to defeat

- App hardening against RE will enhance RASP protection.



1. App Hardening
2. Build Settings Techniques
3. Code Hardening Techniques
4. Data Hardening
5. Crypto Hardening
6. Discussion

- Open source CTF code R2Pay incorporates the above discussed points
- Code obfuscation
    - proguard for Java
    - OLLVM obfuscator for C
- Code hardening
    - Inlining of sensitive code
        - RASP calls are inlined as part of the core logic
    - libc apis replaced with custom code and syscalls
    - byte arrays used for all sensitive data
    - memory wiped just after use

- Crypto obfuscation
  - Symbols removed
  - no traces of which crypto library being used
  - Crypto constants derived at runtime from arbitrary constants
  - Open source WBC to hide crypto key

Demo

- Discussed what is app hardening and its importance
- Discussed with hardening techniques
  - Build Settings
  - Code
  - Data
  - Crypto
- Case study on R2Pay
- One technique in itself will not be sufficient, use all of them for maximum benefit.
- It is an iterative process
- Determine your threat model and use accordingly.

Thank You

- Slides at: https://github.com/su-vikas/Presentations