MOBILE APPLICATIONS

@domchell
@mdseclabs

MDSec

AGENDA

- Background
- The Attack Surface
- Case Studies
- Binary Protections
- Bypasses
- Conclusions

BACKGROUND

- **Mobile apps for everything == lots of interesting data**
  - Banking – financial
  - Social networking – PII
  - Privacy driven applications – variety of data
  - Enterprise applications – integrated in to internal networks

- **It's reasonable to assume that mobile applications are a target, but many applications are described as "secure"**

- **So how secure are the secure applications?**

THE ATTACK SURFACE

- **Traditional mobile application vulnerabilities are well understood and documented**
- **But do they account for all the attack surfaces?**
- **Less emphasis placed on certain types of attacks:**
  - Attacks from malware
  - Exploitation by targeted or casual attack (remote & physical access)
  - Piracy/App Imitation attacks
- **Why should we care about these?**

THE ATTACK SURFACE

- **Malware is a problem and it does target applications**
  - Unflod baby panda – Apple credentials
  - iKee.B – ING Direct customers
  - HijackRAT – various banking applications
  - Xsser mRAT – Tencent app + other device data
- **Currently only a problem for jailbroken devices on iOS**
- **Much more of a problem in the Android ecosystem**
- **Does your secure application detect this?**

THE ATTACK SURFACE

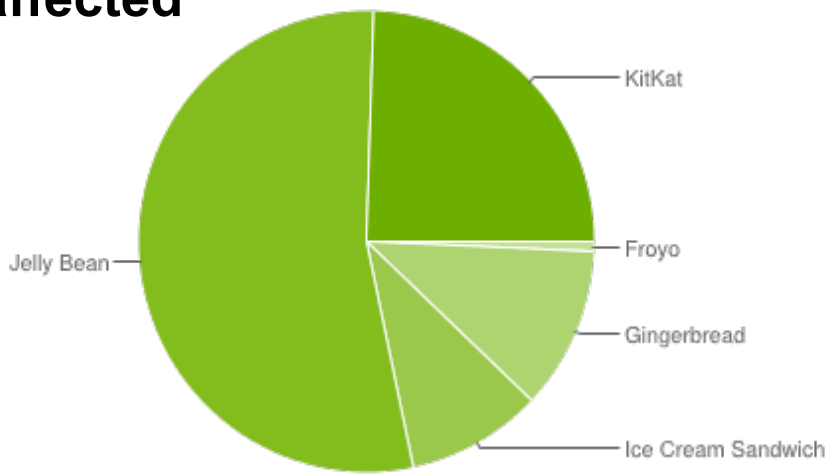
- **Exploitation from targeted or casual attack may be** *easier than you think*…..
- **Jailbreaks via physical access (shoulder surfed passcode or stolen while unlocked)**
- **Accuvant demonstrated a remote Over-The-Air jailbreak at BlackHat 2014 (iOS < 7.0.3)**
  - http://www.accuvant.com/blog/cellular-exploitation-on-a-global-scale
- **Remote weaponised exploit for MobileSafari available in the wild (CVE-2014-4377 – iOS 7.1.x)**
  - https://github.com/feliam/CVE-2014-4377

THE ATTACK SURFACE

- **Research by Google shows only 24.5% of devices are on Kitkat (Sept 9th 2014)**
  - At least 67.5% of devices using < API 17
  - Many devices vulnerable to CVE-2012-6636 (addJavaScriptInterface) including ~34.7% of devices with the AOSP browser (CVE-2014-1939)!

- **Likely many more applications affected**

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	11.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	9.6%
4.1.x	Jelly Bean	16	25.1%
4.2.x		17	20.7%
4.3		18	8.0%
4.4	KitKat	19	24.5%



CASE STUDIES

- **It's reasonable to assume that on-device attacks and malware are a credible threat to mobile applications**

- **Certain types of applications require more protection:**
  - BYOD
  - MDM
  - Enterprise applications
  - Banking
  - Privacy-driven

- **Many of the current solutions make *bold* statements on security, but have these been challenged?**

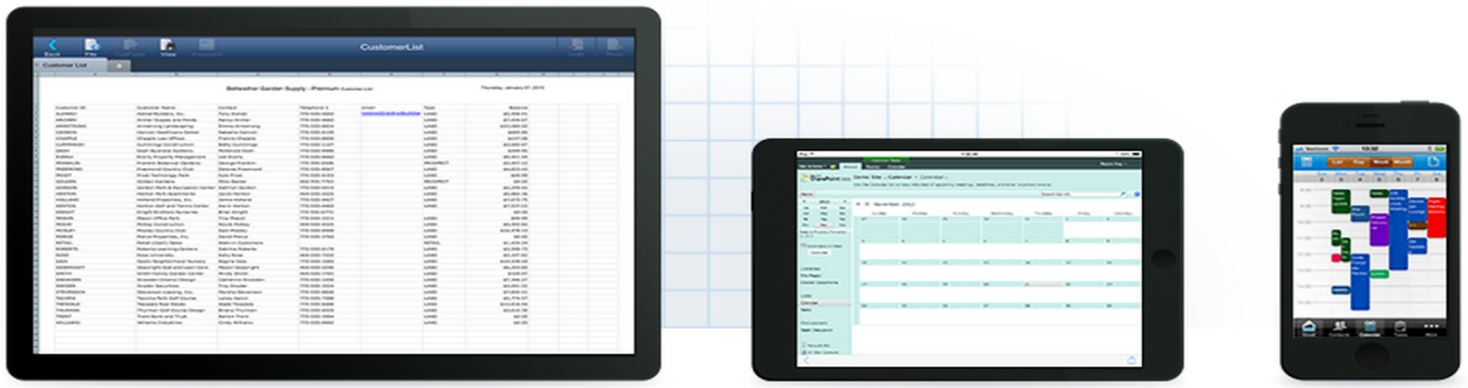
CASE STUDIES

KASEYA BYOD

- **Best described by their marketing….**

Bring Your Own Device (BYOD) Management

With the Kaseya BYOD Suite, organizations can give employees the freedom to work on their personal devices, while ensuring the security of enterprise data and applications on those devices. The solution provides employees with access to corporate applications, email, and documents from their personal devices , while providing IT with peace of mind through unprecedented BYOD security and ease of deployment and administration.



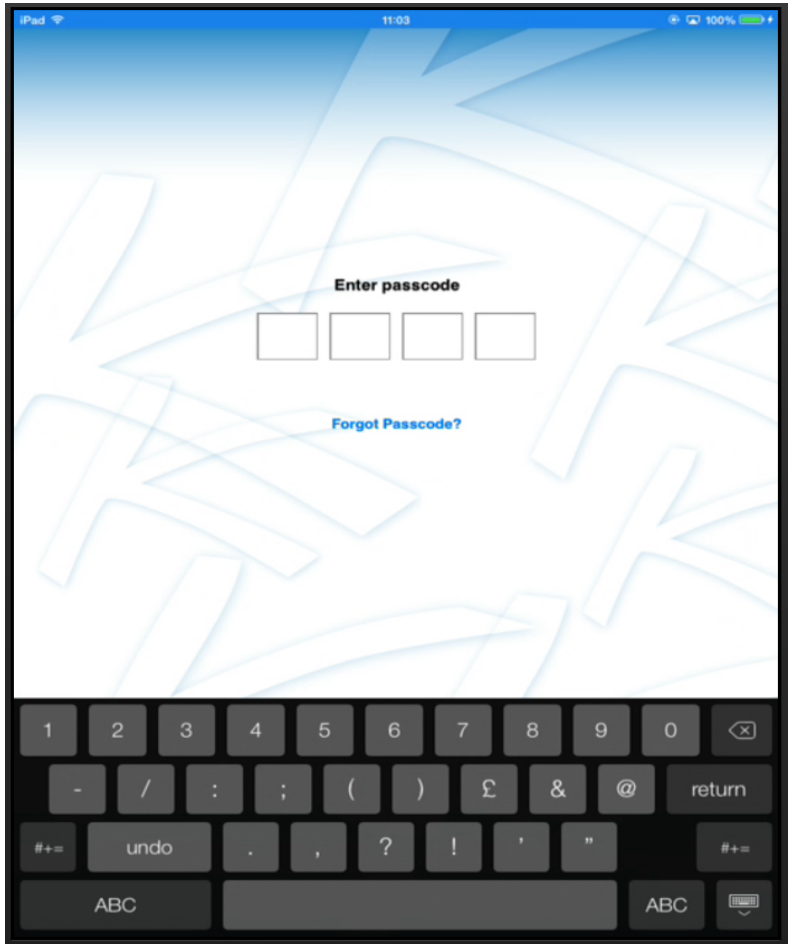
Kaseya BYOD Suite running on different device types

KASEYA BYOD

- Integrates to your internal network via relay servers

KASEYA BYOD

- **Physical access is protected via PIN**



KASEYA BYOD

DEMO

WICKR

- **Came to my attention with a bug bounty announcement in Jan 2014**

Wickr will pay as much as US $100,000 for a vulnerability that substantially affects the confidentiality or integrity of user data. We will also consider paying the same amount for defense techniques and novel approaches to eliminating the vulnerability that are submitted at the same time. Our goal is to make this the most generous and successful bounty program in the world.

Beyond making lots of money, you can feel good about helping Wickr because we were founded to protect the basic human right of private correspondence. Private correspondence is extremely important to a free society. People all over the world depend on Wickr. Please help us with this mission.

To submit a bug, please contact us via email at bugbounty@mywickr.com. The program specifics are on the following pages.

Engaging Hackers

Beyond the Bug Bounty Program, Wickr engages with the best security firms in the world for code review and penetration testing. Veracode gave Wickr a perfect score on its first review. Furthermore, Wickr had the honor to be the target of a presentation at DEF CON 21 conducted by experts from Stroz Friedberg, one of the largest forensics companies in the world. The researchers analyzed Wickr, Snapchat and Facebook Poke to determine that while Snapchat and Facebook revealed personal information, Wickr indeed left no trace. We expect finding critical vulnerabilities in Wickr to be difficult and are honored to work with those that do.

WICKR

- **“Top-Secret Messenger”**
  - Privacy driven instant messaging service
- **“Leave no trace”**
  - Supposedly forensically sound
- **“Messages are secured with military-grade encryption”**
- **“They can only be read by you and the recipients on the devices you authorize”**

ESCAPE THE INTERNET™

WICKR

DEMO

GO!ENTERPRISE

- **BYOD application similar to Kaseya (integrates with on-premises or via cloud)**

- **Makes some bold claims about security…**
  - "The GO!Enterprise mobility platform was designed from the ground up with security in mind. Thus GO!Enterprise solutions inherit a wealth of security features that minimize the risk of unauthorized access, data leakage and security breaches."

GO!ENTERPRISE

- **A cursory analysis revealed that most of the apps content is encrypted**

- **Further analysis showed SQLCipher was being used to encrypt the storage database**

- **But how is the encryption key derived?**

GO!Enterprise Security features include:
- FIPS 140-2 validated cryptography on device and Server
- AES 256 bit Encryption on Device and Over the Air (OTA) traffic
- 3DES 192 bit Encryption on Server and Cloud

GO!ENTERPRISE

- **A password is generated from the IMEI and the path to the database (static) then used by a key derivation function**

```
public String generateKey()
{
  Object[] arrayOfObject = new Object[2];
  arrayOfObject[0] = getDeviceIMEI();
  arrayOfObject[1] = this.mContext.getDatabasePath(this.mName);
  String str1 = String.format("%s:%s", arrayOfObject);
  byte[] arrayOfByte = getDeviceIMEI().getBytes();
  try
  {
    PBEKeySpec localPBEKeySpec = new PBEKeySpec(str1.toCharArray(), arrayOfByte, 300, 192);
    str2 = new String(encodeHex(new SecretKeySpec(SecretKeyFactory.getInstance("PBEWithSHA256And256BitAES-CBC-BC")
      .generateSecret(localPBEKeySpec).getEncoded(), "AES").getEncoded()));
    return str2;
  }
```

GO!ENTERPRISE

DEMO

SUMMARY

- **Security controls within mobile applications, such as client-side authentication, can often be trivially bypassed using instrumentation**

- **Integrating a BYOD or MDM app in to your network can widen your exposure and allow exploitation from the client**

- **If you're using crypto then use some input from the user or a split server key**

BINARY PROTECTIONS

- **Introduced to the OWASP Mobile Top Ten at OWASP AppSec California in January 2014**

- **Creating self-defending mobile applications**

- **Attempts to achieve the following goals:**
  - Prevent software operating in an untrusted environment
  - Thwart or increase the complexity of reverse engineering
  - Thwart or increase the complexity of modification or tampering attacks
  - Detect/Prevent attacks from on-device malware

- **How common are these protections?**
  - 2013 study by HP : “86 percent of applications tested *lacked* binary hardening”

BINARY PROTECTIONS

- **So what are binary protections?**

- **You're likely to have encountered some of the following:**
  - Jailbreak/Root detection
  - Resource integrity validation
  - Anti-debugging
  - Runtime-tamper detection
  - Obfuscation

- **A continual arms race**

- **Not a silver bullet!**

JAILBREAK/ROOT DETECTION

- **Perhaps the most commonly implemented binary protection**
- **Attempts to detect if the application is running on a jailbroken/rooted device**
- **If a compromise is detected the app usually does one or more of the following:**
  - Warn the user
  - Wipe any sensitive data
  - Report back to a management server
  - Exit or crash

JAILBREAK/ROOT DETECTION

- **Usually implemented by checking for one or more of the following:**
  - Jailbreak/root artifacts
  - Non standard open ports
  - Weakening of the sandbox
  - Evidence of system modifications (e.g. build keys)

- **Often trivial to bypass unless other protections are in place**

RESOURCE INTEGRITY VALIDATION

- **Attempts to detect changes to application resources or internal code structures:**
  - Images, javascript or HTML files
  - Modifications to APK
  - Patching of internal code or shared libraries

- **Typically implemented using checksums that are embedded in to the application at compile time**
  - Only useful if implemented as “web” – what’s checking the checksumming routines?
  - CRC32 is commonly used, so scan for polynomials
  - LLVM can be used to make the process relatively seamless

ANTI-DEBUGGING

- With the ability to debug an application an attacker can trivially modify it's behavior
- Anti-debugging protections attempt to detect and prevent a debugger being attached to your application
- Unlikely to thwart an advanced adversary
- There's a handful of tricks that are relatively easy to implement but may slow down your attacker….

ANTI-DEBUGGING

- On iOS a debugger your process status flag can be queried using sysctl:

```
int checkDebugger()
{
    int name[4];
    struct kinfo_proc info;
    size_t info_size = sizeof(info);

    info.kp_proc.p_flag = 0;

    name[0] = CTL_KERN;
    name[1] = KERN_PROC;
    name[2] = KERN_PROC_PID;
    name[3] = getpid();

    if (sysctl(name, 4, &info, &info_size, NULL, 0) == -1) {
        return 1;
    }
    return ((info.kp_proc.p_flag & P_TRACED) != 0);
}
```

- The PT_DENY_ATTACH flag can also be set to prevent tracing

ANTI-DEBUGGING

- **On Android detection is most commonly implemented using the `Debug.isDebuggerConnected` method in the DVM**

- **If you don't trust the API you can also query this value using JNI and the gDvm symbol exported by the Dalvik VM**

- **There's also scope to get creative using tricks such as thread timing:**

```
boolean checkDebugger3()
{
    long currentCpuTime = Debug.threadCpuTimeNanos();

    for(int i=0; i<1000000; ++ i)
        continue;

    long endCpuTime = Debug.threadCpuTimeNanos();

    if((endCpuTime - currentCpuTime) < 30000000)
        return false;
    else return true;
}
```

RUNTIME TAMPER DETECTION

- **Frameworks like Xposed, Frida and Cydia Substrate making instrumentation of the Objective-C and Java runtimes trivial**
- **This can be used by your attacker or malware to invoke or modify internal methods:**
  - Bypass security controls
  - Leak/steal sensitive data
- **This leads to a fairly unique situation where a developer cannot necessarily always trust their own runtime**
- **Detection can be complex but there's a number of tricks you can use on iOS – unaware of anything on Android outside of protecting native code**

RUNTIME TAMPER DETECTION

- **Check #1 : Validating the source image location**

- **The locations for dylibs with the SDK methods is a finite set of directories:**
  - /System/Library/TextInput
  - /System/Library/Accessibility
  - /System/Library/PrivateFrameworks/
  - /System/Library/Frameworks/
  - /usr/lib/

- `dladdr` **takes a function pointer and returns details on the source image – can be used to check if the function lives in the right location**

RUNTIME TAMPER DETECTION

- Iterate the methods in a class and check the source image

```
int checkClassHooked(char * class_name)
{
    char imagepath[512];

    int nummeths;
    Dl_info info;
    id c = objc_lookUpClass(class_name);
    Method * m = class_copyMethodList(c, &nummeths);

    for (int i=0; i<nummeths; i++)
    {
        char * methodname = sel_getName(method_getName(m[i]));
        void * methodimp = (void *) method_getImplementation(m[i]);

        int d = dladdr((const void*) methodimp, &info);
        if (!d) return YES;

        memset(imagepath, 0x00, sizeof(imagepath));
        memcpy(imagepath, info.dli_fname, 9);
        if (strcmp(imagepath, "/usr/lib/") == 0) continue;
        // add additional paths

        return YES;
    }
    return NO;
}
```

RUNTIME TAMPER DETECTION

- **Check #2: Scan for malicious libraries**

- **Cydia Substrate and Cycript will inject a dylib in to the process when it launches**

- **Scanning for unknown libraries and signatures from malicious libraries can give you some confidence of hooking**

RUNTIME TAMPER DETECTION

- **Retrieve a list of loaded libraries and scan their names**

```
void scanForInjection()
{
    uint32_t count = _dyld_image_count();

    char* evilLibs[] =
    {
        "Substrate", "cycript"
    };

    for(uint32_t i = 0; i < count; i++)
    {
        const char *dyld = _dyld_get_image_name(i);
        int slength = strlen(dyld);
        int j;
        for(j = slength - 1; j>= 0; --j)
            if(dyld[j] == '/') break;

        char *name = strndup(dyld + ++j, slength - j);

        for(int x=0; x < sizeof(evilLibs) / sizeof(char*); x++)
        {
            if(strstr(name, evilLibs[x]) || strstr(dyld, evilLibs[x]))
                fprintf(stderr,"Found injected library matching string: %s", evilLibs[x]);
        }

        free(name);
    }
}
```

RUNTIME TAMPER DETECTION

- Check #3: Searching for trampolines

- Under the hood many of the hooking frameworks simply insert a trampoline in the function prologue

- Looking at a normal (left) and hooked (right) function you can see the changes:

```
(lldb) disassemble -a 0x3900b7a5
libsystem_c.dylib`fork:
    0x3900b7a4:  push   {r4, r5, r7, lr}
    0x3900b7a6:  movw   r5, #0xe86c
    0x3900b7aa:  add    r7, sp, #0x8
    0x3900b7ac:  movt   r5, #0x1d0
    0x3900b7b0:  add    r5, pc
    0x3900b7b2:  ldr    r0, [r5]
    0x3900b7b4:  blx    r0
```

```
(lldb) disassemble -a 0x3900b7a5
libsystem_c.dylib`fork:
    0x3900b7a4:  bx     pc
    0x3900b7a6:  mov    r8, r8
    0x3900b7a8:  .long  0xe51ff004
    0x3900b7ac:  bkpt   #0x79
    0x3900b7ae:  lsls   r5, r1, #0x6
    0x3900b7b0:  add    r5, pc
    0x3900b7b2:  ldr    r0, [r5]
    0x3900b7b4:  blx    r0
```

RUNTIME TAMPER DETECTION

- **Tracing this back to the Substrate code you can see exactly what it's doing:**

```
buffer[start+0] = A$ldr_rd_$rn_im$(A$pc, A$pc, 4 - 8);
```

- **Substrate simply inserts a jump to next word after $pc (ldr pc, [pc-4])**

```
int checkFunctionHook(void * funcptr)
{
    unsigned int * funcaddr = (unsigned int *) funcptr;
    if (funcptr) {
        if (funcaddr[0] == 0xe51ff004) return 1;
    }
    return 0;
}
```

OBFUSCATION

- **Attempts to complicate reverse engineering by making compiled code complex to understand**

- **Some of the techniques include:**
  - Obscuring class, field and method names,
  - Inserting bogus code,
  - Modifying the control flow,
  - String encryption,
  - Substitution of code to make it appear more complex, for example using reflection,
  - Control flow flattening.

OBFUSCATION

- **There are a number of obfuscators available for Android, including Proguard, Dexguard, DashO, APKProtect, Allatori to name but a few**

- **Good comparison of the different obfuscation solutions for Android was made by jcase & diff @ defcon22**
  - https://www.defcon.org/images/defcon-22/dc-22-presentations/Strazzere-Sawyer/DEFCON-22-Strazzere-and-Sawyer-Android-Hacker-Protection-Level-UPDATED.pdf

- **Freely available options for obfuscating native code are limited but obfuscator-llvm and iOS-class-guard are useful (**https://github.com/Polidea/ios-class-guard**)**

BYPASSING BINARY PROTECTIONS

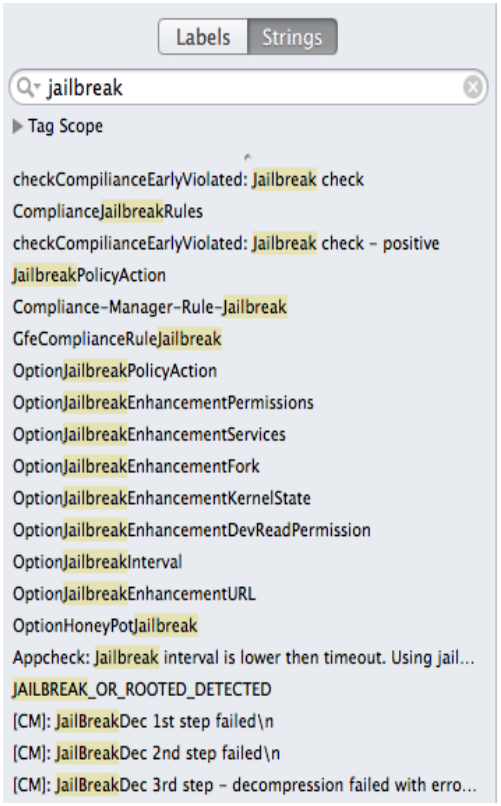
- As previously noted binary protections are not a silver bullet
- In most cases they can be trivially bypassed if not layered together in an intelligent manner
- To demonstrate how easy this can be, we looked at how binary protections were used in two popular MDM products

GOOD FOR ENTERPRISE

- GFE is a popular MDM product
- It communications with the Good NOC to customer hosted equipment to integrate in to internal systems
- Policy is driven via a management server and enforced on the client
- The only evident binary protection was jailbreak detection, how complex would this be to bypass?

GOOD FOR ENTERPRISE

- …..actually quite easy!
- 5 binary patches can be applied to defeat the detection



Location	Original instruction	Modified Instruction
0x190e2	bl sub_d900	movw r0, 0x0
0x1ab10	bl sub_d900	movw r0, 0x0
0x22f6e	bl sub_d900	movw r0, 0x0
0x596b56	ldrbw r0, [sp]	movs r0, 0x0
0x596b5a	movs r4, 0x1	movw r0, 0x0

MOBILEIRON

- Direct competitor to GFE
- Works in much the same way though clients can host their own VSP & Sentry devices
- Again the only evident binary protection was jailbreak detection, how complex is it to bypass?
- Thanks to @hackerfantastic for this research!

MOBILEIRON

- …..again, not that complex!
- 12 hooks

```
%hook HomeViewController
-(void)processCompliance:(int)compliance { %log; %orig(1);}
%end
%hook AppConnectAppStatus
-(void)setAuthState:(id)state { %log; }
%end
%hook MILog
+(void)setLogLevel:(int)level { %log; %orig(3); }
%end
%hook MIConfig
- (void)setBlocked:(unsigned long )blocked { %log; %orig(0); }
-(void)updateJustSpecial:(id)special andSaveToDisk:(BOOL)disk { %log; }
%end
%hook DeviceDetailManager
-(id)getChangedDetails:(BOOL)details { %log; id r = 0; return r; }
-(BOOL)hasDetailChanged:(id)changed { %log; BOOL r = false; return r; }
-(void)markAsUpToDate { %log;}
-(void)updateDeviceDetails { %log; }
%end
%hook AppDataManager
-(void)wipeDataForApp:(id)app { %log; }
%end
%hook DocManager
-(void)wipeFileDataAndRecentFiles { %log; }
-(void)wipe { %log; }
%end
```

CONCLUSIONS

- On-device and malware are a realistic attack surface to consider when developing mobile applications
- Building self-defending mobile apps can give you some assurances and thwart skilled adversaries
- Binary protections aren't a silver bullet and can be trivially bypassed if not layered in an intelligent way

Q&A

@domchell
@mdseclabs