IOS客户端通用测试

IOS客户端运行时的安全

3.1 越狱环境监测

测试项描述

越狱的设备是不安全环境，各种恶意程序都容易运行在越狱环境之中。越狱环境中运行则难以保证程序使用过程中数据的安全性。检测设备是否越狱是一个比较有用的方法。

测试步骤

打开程序，检查程序是否会弹出越狱环境提示。

结果判定

安全。如右图所示

风险评级

低风险——未检测运行环境是否为越狱环境的

安全建议

方法一，检测文件与路径的方法

代码如下:

```
-(BOOL)isJailBroken{
    if ([[NSFileManager defaultManager] fileExistsAtPath:@"/Applications/Cydia.app"]){
        return YES;
    }else if([[NSFileManager defaultManager] fileExistsAtPath:@"/Library/MobileSubstrate/MobileSubstrate.dylib"]){
        return YES;
    }else if([[NSFileManager defaultManager] fileExistsAtPath:@"/bin/bash"]){
        return YES;
    }else if([[NSFileManager defaultManager] fileExistsAtPath:@"/usr/sbin/sshd"]){
        return YES;
    }else if([[NSFileManager defaultManager] fileExistsAtPath:@"/etc/apt"]){
        return YES;
    }
    return NO;
```

安全建议

方法二，可以检测 URL SCHEME 或者 SSH 链接

代码如下:

```
1  if([[UIApplication sharedApplication] canOpenURL:[NSURL URLWithString:@"cydi
   a://"]]){
2          return YES;
3  }
```

安全建议

方法三，检测沙箱环境 主要是检测读权限、检测写权限、检测可执行权限、已加载 dvlib、关键函数宿主:

```
1  if ([[NSFileManager defaultManager] fileExistsAtPath:@"/User/Applications/"]){
2      NSArray *applist = [[NSFileManager defaultManager]
           contentsOfDirectoryAtPath:@"/User/Applications/"
                               error:nil];
3      if (applist != nil || [applist count] > 0 ) {
4          return YES;
5      }
6  }// 可以是 NSFileManager
7
9  [@"test" writeToFile:@"/private/test.txt" atomically:YES encoding:NSUTF8StringEncoding erro
   r:nil];
10 if (error == nil) {
11     return YES;
12 }
13
14 int forkValue = system(-1); // 可以是 fork()
15 if (forkValue > 0) {
16     return YES;
17 }
```

注意事项:

此项是银行金融类APP合规测试项。

对于内部应用，完全可以检测到越狱环境之后终止程序运行；

对于互联网应用，则可能仅仅需要用户知道在越狱环境上使用是有风险的即可。

IOS客户端运行时的安全

3.2 动态环境监测

测试项描述

程序如果没有反动态调试，攻击者能够使用 LLDB对程序进行动态调试，修改运行时数 据，影响程序正常运行甚至获取程序逻辑。

测试步骤

配置debuaserver

1.将手机中的debugserver(/Developer/usr/bin/debugserver) 拷贝到电脑上。

2.利用命令进行处理 (lipo-thin armv7 debugserver-output debugserver)

3.添加task for pid 权限 (下ent.plist，将ent.plist和debugserver放到同一目录下进行签名， codesign-s - --entitlements ent.plist -f debugserver

4.将处理好的debuaserver拷贝回越狱ios设备

使用LLDB

1.ssh到手机，然后开启debuaserver监听 (debugserver*:12345-a“targetapp”)

2.使用命令:

Lldb

Process. Connnect connect://ip:port

结果判定

```
(lldb) breakpoint list
Current breakpoints:
1: address = 0x00000000000cc730, locations = 1
  1.1: address = 0x00000000000cc730, unresolved, hit count = 0
```

如右图所示，可以进行附加并调试

程序不安全

```
debugserver-@(#)PROGRAM:debugserver  PROJECT:debugserver-340.3.51.1
 for armv7.
Attaching to process DahuaLink_Enterprise...
Listening to port 12345 for a connection from *...
Waiting for debugger instructions for process 0.
```

风险评级

中风险

安全建议

使用ptrace阻止附加

```
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
#import <dlfcn.h>
#import <sys/types.h>

void anti_gdb_debug();

typedef int (*ptrace_ptr)(int request, pid_t pid, caddr_t addr, void *data);

#if !defined(PT_DENY_ATTACH)
#define PT_DENY_ATTACH 31
#endif

int main(int argc, char * argv[]) {

#ifndef _DEBUG
    anti_gdb_debug();
#endif

    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}

void anti_gdb_debug(){
    void *handle = dlopen(NULL, RTLD_GLOBAL | RTLD_NOW);
    ptrace_ptr p_ptr = dlsym(handle, "ptrace");
    p_ptr(PT_DENY_ATTACH,0,0,0);
    dlclose(handle);
}
```

IOS客户端运行时的安全

3.3 输入记录保护

测试项描述

iOS中安装了键盘记录程序(可以在非越狱状态运行)，则使用系统键盘输入的内容易 被键盘记录程序记录，导致内容泄露。

测试步骤

在苹果越狱机的Cydia软件中安装了ikevmonitor插件，keymonitor未过期可以在http://localhost: 8888 中查看记录)

结果判定

如下所示，其记录了 username 和 password:

```
com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {(
    )}; level 0.000000; active NO [wants NO]; suppression 0>] could not take process assertion
Aug 16 16:15:10 NSFOCUS-ZN ApexWhYdcrm[1134] <Warning> ***Keystrokes submitted Username
Aug 16 16:15:10 NSFOCUS-ZN SpringBoard[79] <Warning>: [<_UIKeyboardArbiterHandle: 0x17ca6cb0; PID 321:
com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {(
    )}; level 0.000000; active NO [wants NO]; suppression 0>] could not take process assertion
Aug 16 16:15:10 NSFOCUS-ZN SpringBoard[79] <Warning>: [<_UIKeyboardArbiterHandle: 0x17ca6cb0; PID 321:
com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {(
    )}; level 0.000000; active NO [wants NO]; suppression 0>] could not take process assertion
Aug 16 16:15:10 NSFOCUS-ZN SpringBoard[79] <Warning>: [<_UIKeyboardArbiterHandle: 0x17ca6cb0; PID 321:
com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {(
    )}; level 0.000000; active NO [wants NO]; suppression 0>] could not take process assertion
Aug 16 16:15:13 NSFOCUS-ZN ApexWhYdcrm[1134] <Warning> ***Keystrokes submitted password
Aug 16 16:15:13 NSFOCUS-ZN SpringBoard[79] <Warning>: [<_UIKeyboardArbiterHandle: 0x17ca6cb0; PID 321:
com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {(
```

风险评级

高风险——记录到交易密码级别内容

中风险——记录到用户名密码级别内容

低风险——其他信息

安全建议

使用自定义软键盘。

参考https://github.com/ibireme/YYKeyboardManager

IOS客户端运行时的安全

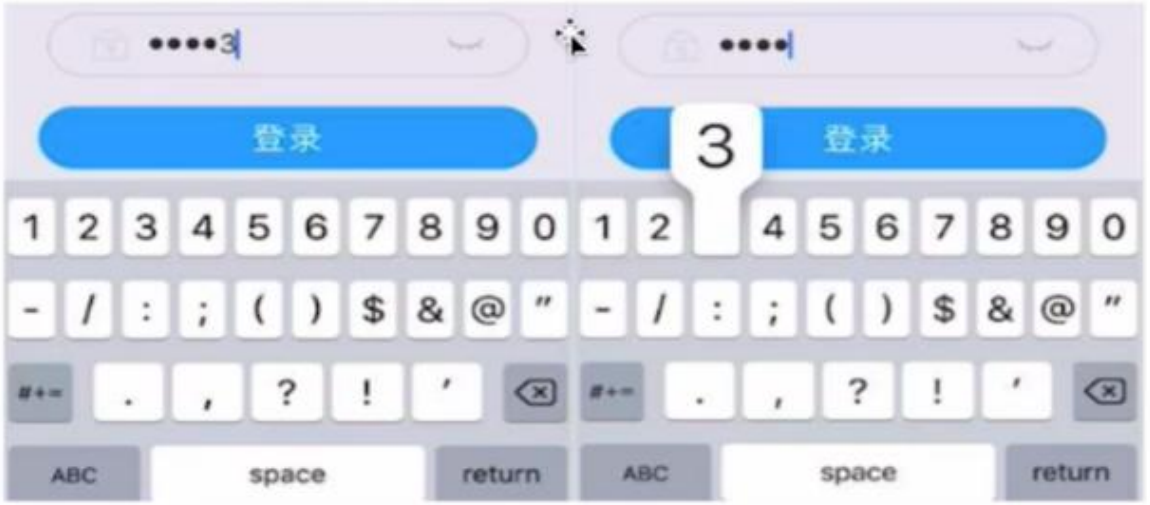
3.4 屏幕录像保护

测试项描述

测试通过截图工具，是否可以捕捉到用户密码输入框的密码。

结果判定

如图所示，输入时候动画变换被记录:



安全建议

重写输入框避免字符变化，并取消键盘动画回显。

IOS客户端运行时的安全

3.5 URL Scheme劫持检测

测试项描述

URLScheme是ios 中程序间进行信息传递的途径,但是苹果系统未禁止不同程序使用相同的URLScheme，导致URLScheme 能够被劫持，造成钓鱼风险。而且苹果并不会检查这样的 URLSchemes 重复问题。

什么是URL Scheme?

简单的说，由于苹果选择沙盒来保障用户的隐私和安全，App只能访问自己的沙盒但同时也阻碍了应用间合理的信息共享。所以苹果提供了一个可以在App之间跳转的方法URL Scheme。如果你的App需要提供一个供别的App访问的功能或者数据，那么你必须在你的App定义一个相对应的URL Scheme。当别的App使用一个URL Scheme进行访问时，系统会根据URLScheme进行匹配，执行相应的操作。

如何理解URL Scheme?

为了更加深入的理解上面的那那段话，首先我们清楚几个概念:

- URL(Uniform Resoure Locator:统一资源定位器):也就是我们熟悉的那个“网址”，通过他我们可以访问到我们想要的服务和资源，并且URL可以传递相应的参数;
- URL地址格式排列为:sheme://host;port/path，举个例子：http://wwwsohu.com/domain/HXWZ就是一个典型的URL，而这个网址对应的Scheme就是http表示的是一个 URL中的一个位置一最初始的位置。也可以理解为一种自定义的协议。
- 根据我们上面对URL Scheme 的理解，我们可以很轻易地理解，在以本地应用为主的ios 上，我们可以像定位一个网页一样，用一种特殊的 URL 来定位一个应用甚至应用里某个具体的功能。而定位这个应用的标识，也就是Scheme。比如微信的Scheme是weixin,打开微信扫一扫功能的URLScheme则是weixin://dl/scan。
- 这样一对比就容易很明白的理解出了URLScheme的真正含义，它是为了在iOS系统中定位对应的App然后执行对应的操作，复杂的URLScheme还可以传递参数。

URLScheme的应用场景？

为了更加深入的理解上面的那段话，首先我们清楚几个概念：

使用iOS系统预设的URL Scheme调用系统App:

ios系统内置的App，如mail，电话等等，都有相应的URL Scheme供其他的App调用。比如下面的代码就是使用系统的电话App给18888888888打电话。

[[UIApplication sharedApplication] openURL:INSURL URLWithStrina:@"tel/18888888888"]l;

关于其他系统支持的URL Scheme我在此不再赘述

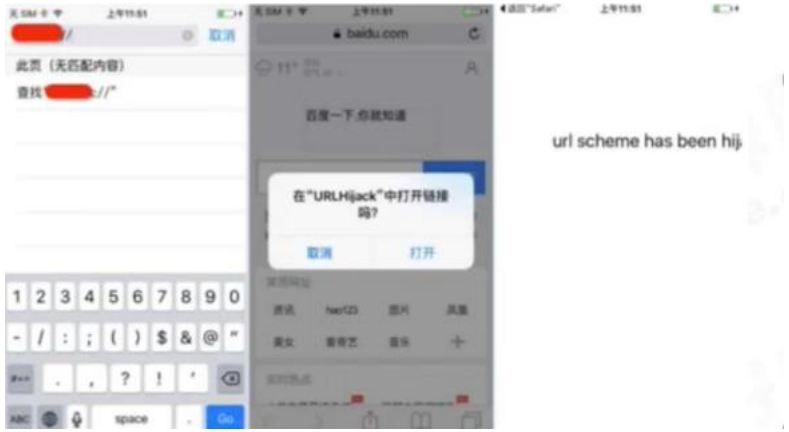
常用iOS URL Scheme

名称	URL Scheme	Bundle identifier
Safari：	http://	
maps：	http://maps.google.com	
Phone：	tel:	
SMS：	sms:	
Mail：	mailto:	
iBooks：	ibooks://	
App Store：	http://itunes.apple.com	
iTunes：	http:http://phobos.apple.com	
Music / iPod：	music:	
Videos：	videos:	

测试步骤

1.查看info.plist是否存在URLschemes

2.新建一个xCode程序，在infoplist中加入URLSchemes，item的值要和需要被劫持的值一致!



结果判定

在打开目标程序的时候，若目标程序未提醒用户URLScheme 可能被劫持，则存在风险。

风险评级

低风险

安全建议

方法一:

应用本身可以发送一条URLScheme 请求给自己如果自己可以接收到的话说明URLScheme 没有被劫持如果不能收到的话就说明被劫持了,这时候可以提醒用户卸载有冲突的 app

```
1   [[UIApplication sharedApplication] openURL:[NSURL URLWithString:@"<URL_S
    CHEME>://"]];
```

安全建议

方法二:

利用 MobileCoreServices 服务中的applicationsAvailableForHandlinaURLScheme0方法可以获取所有注册了该URL Schemes 的应用和处理顺序,随后应用就可以检测自己,或者别人的 URL Scheme 是否被劫持了，被谁劫持了

```
1  Class LSApplicationWorkspace_class = objc_getClass("LSApplicationWorkspace
   ");
2  NSObject * workspace = [LSApplicationWorkspace_class performSelector:@sel
   ector(defaultWorkspace)];
3  NSLog(@"openURL: %@", [workspace performSelector:@selector(applicationAv
   ailableForHandlingURLScheme:) withObject:@"<URL_SCHEME>"]);
```

谢谢观赏