

iOS用户空间安全

Proteas



关于我

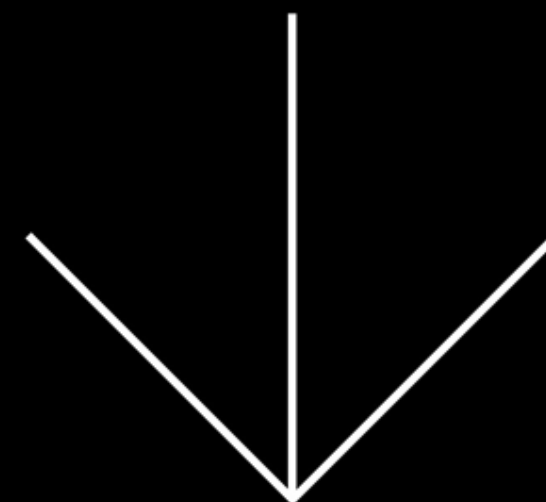
Proteas of 360 Nirvan Team

iOS and OS X Security Researching

weibo: @Proteas

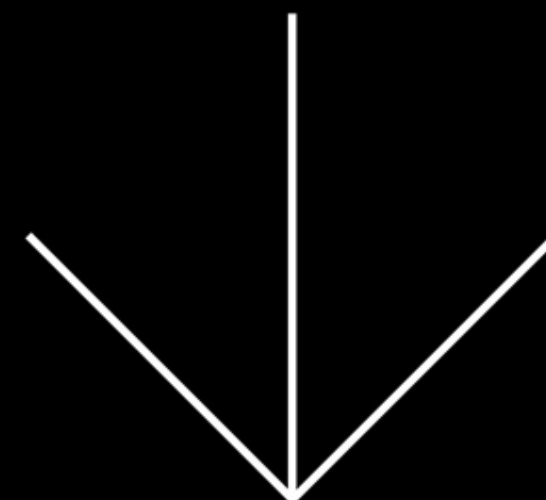
twitter: @ProteasWang

email: wangwei-it@360.cn, proteas.wang@gmail.com



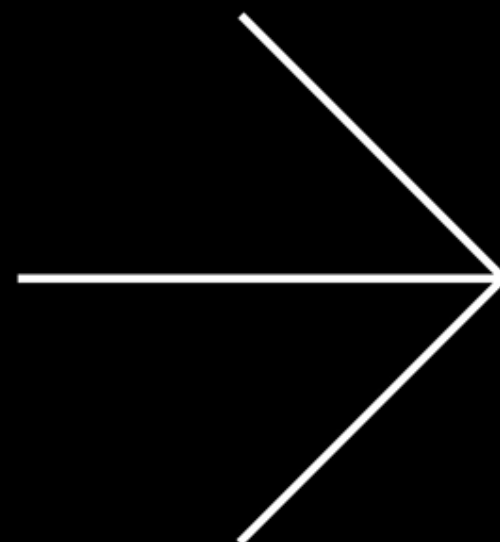
主要内容

- iOS 安全特性概览
- 沙盒的权限泄露
- 应用安全与攻击面
- 脚本型攻击
- 脚本型攻击的实施方案
- 脚本型攻击的防护建议

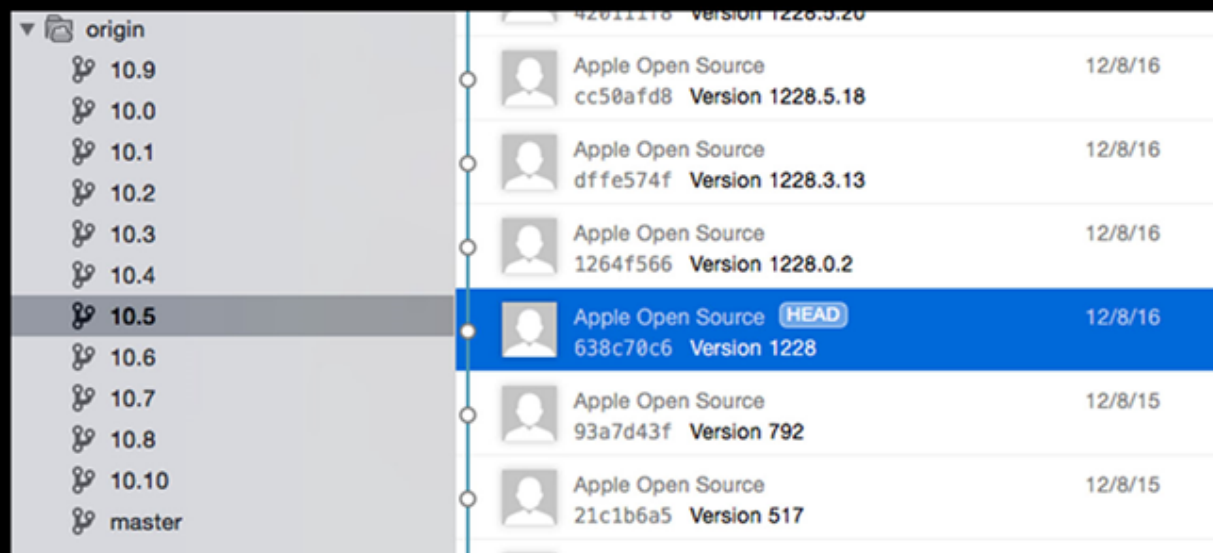


iOS 安全特性概览

- 01 Mandatory Access Control
- 02 Apple Mobile File Integrity
- 03 Sandbox

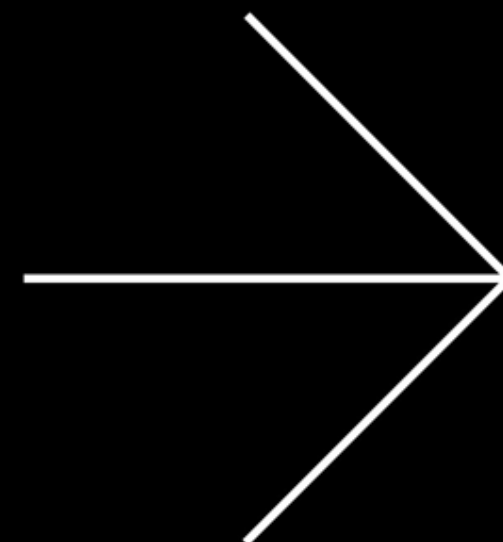


01 / iOS 安全特性—Mandatory Access Control



origin	42011170 Version 1228.5.20	
10.9	Apple Open Source cc50afd8 Version 1228.5.18	12/8/16
10.0	Apple Open Source dffe574f Version 1228.3.13	12/8/16
10.1	Apple Open Source 1264f566 Version 1228.0.2	12/8/16
10.2	Apple Open Source 638c70c6 Version 1228	12/8/16
10.3	Apple Open Source 93a7d43f Version 792	12/8/15
10.4	Apple Open Source 21c1b6a5 Version 517	12/8/15
10.5	Apple Open Source 638c70c6 Version 1228	12/8/16
10.6		
10.7		
10.8		
10.10		
master		

- 来自 FreeBSD , TrustedBSD
- xnu-1228 中加入 MAC 框架
- 代码位置 : xnu-src-root/security
- 对应的 Mac OS X 版本为 10.5.x
- 对应的 iOS 版本为 2.x



01 / iOS 安全特性—Mandatory Access Control

```
475 kern_return_t
476 task_for_pid(
477     struct task_for_pid_args *args)
478 {
479     mach_port_name_t target_tport = args->target_tport;
480     int pid = args->pid;
481     user_addr_t task_addr = args->t;
482     struct uthread *uthread;
483     proc_t p = PROC_NULL;
484     task_t t1 = TASK_NULL;
485     mach_port_name_t tret = MACH_PORT_NULL;
486     ipc_port_t tfpport;
487     void * sright;
488     int error = 0;
489
490     AUDIT_MACH_SYSCALL_ENTER(AUE_TASKFORPID);
491     AUDIT_ARG(pid, pid);
492     AUDIT_ARG(mach_port1, target_tport);
493
494     #if defined(SECURE_KERNEL) ==
495     #endif
496
497     #if CONFIG_MACF
498     error = mac_proc_check_get_task(kauth_cred_get(), p);
499     if (error) {
500         error = KERN_FAILURE;
501         goto tfpout;
502     }
503     #endif
504
505     }
506     error = KERN_SUCCESS;
507
508 tfpout:
509     task_deallocate(t1);
510     AUDIT_ARG(mach_port2, tret);
511     (void) copyout((char *) &tret, task_addr, sizeof(mach_port_name_t));
512     if (p != PROC_NULL)
513         proc_rele(p);
514     AUDIT_MACH_SYSCALL_EXIT(error);
515     return(error);
516 }
```

- 在内核函数中添加检查点
- 如上图：task_for_pid

01 / iOS 安全特性—Mandatory Access Control

```
6230 struct mac_policy_conf {
6231     const char *mpc_name; /* policy name */
6232     const char *mpc_fullname; /* full policy name */
6233     const char **mpc_labelnames; /* array of label names */
6234     unsigned int mpc_labelname_count; /* number of label names */
6235     struct mac_policy_ops *mpc_ops; /* policy operations */
6236     int mpc_loadtime_flags; /* load time flags */
6237     int *mpc_field_off; /* label field offsets */
6238     int mpc_runtime_flags; /* runtime flags */
6239     mpc_t mpc_list; /* List reference */
6240     void *mpc_data; /* module data */
6241 };
```

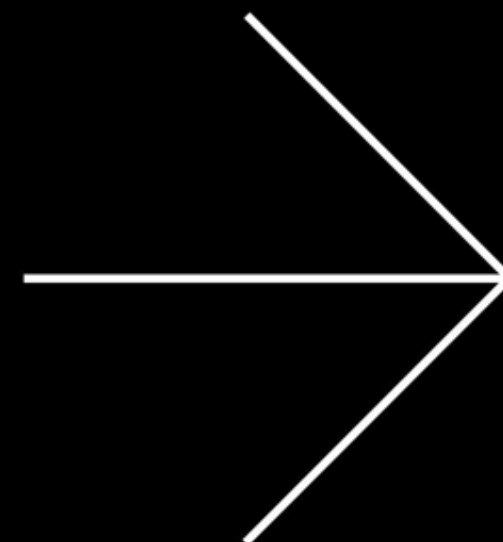
```
5815 #define MAC_POLICY_OPS_VERSION 31 /* inc when new reserved slots added */
5816 struct mac_policy_ops {
5817     mpo_audit_check_postselect_t *mpo_audit_check_postselect;
5818     mpo_audit_check_preselect_t *mpo_audit_check_preselect;
5819     mpo_bpfdesc_label_associate_t *mpo_bpfdesc_label_associate;
5820     mpo_bpfdesc_label_destroy_t *mpo_bpfdesc_label_destroy;
5821     mpo_bpfdesc_label_init_t *mpo_bpfdesc_label_init;
5822     mpo_bpfdesc_check_receive_t *mpo_bpfdesc_check_receive;
5823     ...
};
```

```
typedef int mpo_audit_check_postselect_t(
    kauth_cred_t cred,
    unsigned short syscode,
    void *args,
    int error,
    int retval
);
```

- 可动态注册或取消注册 MAC 策略
 - int mac_policy_register(struct mac_policy_conf *mpc, ...);
 - mac_policy_unregister
- struct mac_policy_conf
- struct mac_policy_ops
- 定义回调函数

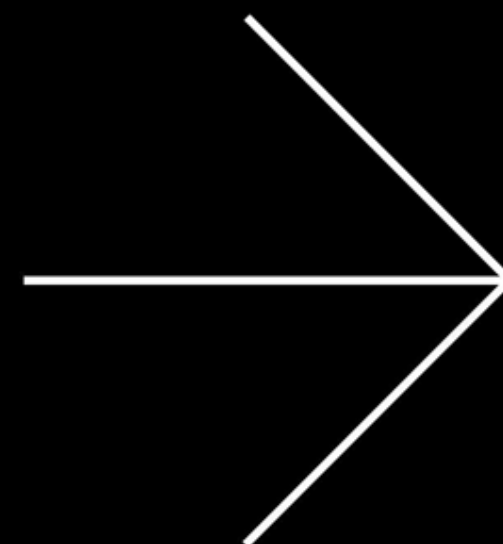
02 / iOS 安全特性—Apple Mobile File Integrity

- Check Executable
 - mpo_vnode_check_exec, check POSIX file permission
- Check Code Signature
 - mpo_vnode_check_signature
 - page with x must be signed



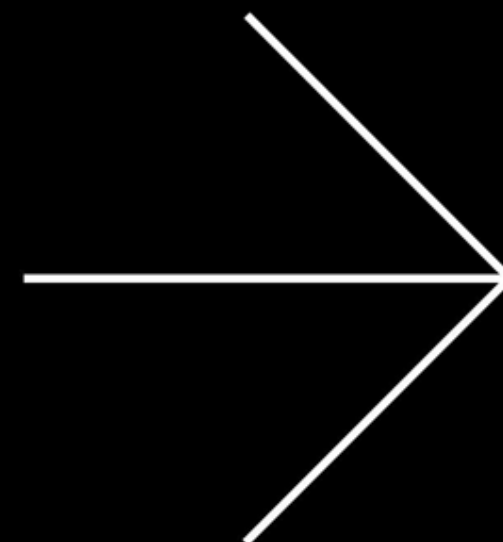
02 / iOS 安全特性—Apple Mobile File Integrity

- Library Validation
 - iOS 8
 - OS X 10 , disabled
- Hook 点 : mpo_file_check_mmap
- 目的 : 防止 dylib 注入
- 苹果的 dylib 可以被加载到任何进程
 - platform library
- 其他的 dylib 要与主程序的 Team ID 匹配
- iOS - 完全在内核中实现
- OS X - 依赖后台服务 amfid 做验证



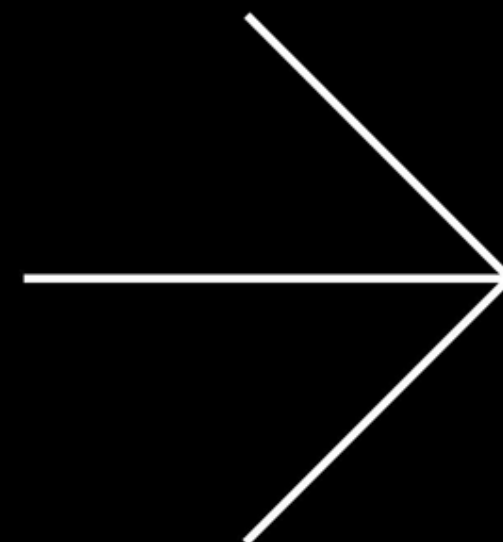
03 / iOS 安全特性—Sandbox

- 目的：限制程序的行为
- 手段-1：检查是否可以调用某些 API
- 手段-2：检查 API 的参数是否合法
 - AppStore 应用只能访问“沙盒”内目录



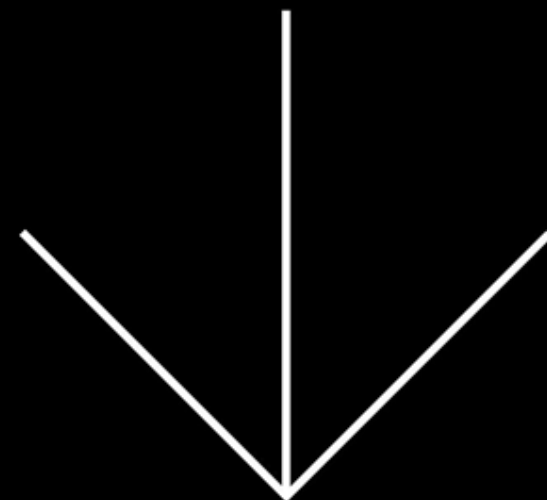
03 / iOS 安全特性—Sandbox

- Rootless
 - OS X 10.11 Beta 1
 - iOS 9, maybe
- 原因：root 的权限太强了，几乎可以任何事情
- 目的：保护系统的关键部分，保证系统的完整性
- 策略：禁止任何程序修改系统的关键部分，只对某些程序开放，比如：installer, updater
 - /bin, /sbin
 - /private/var/db/dyld
 - /System, /System/Library/CoreServices, /System/Library/Extensions
 - /usr
- 通过文件的扩展属性来控制：com.apple.rootless
 - xattr -l -r / 2> /dev/null | grep "com.apple.rootless"



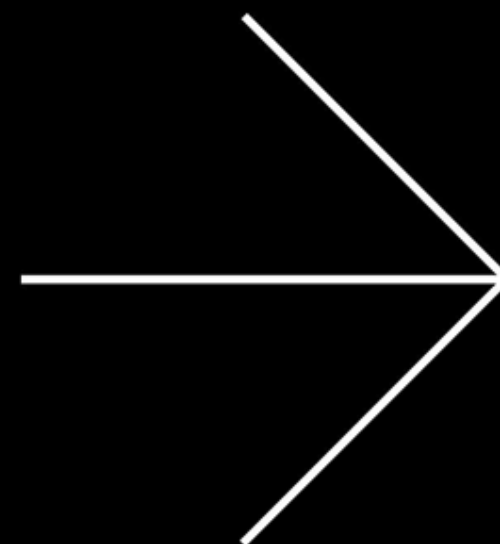
沙盒的权限泄露

- 01 沙盒与 Entitlements
- 02 Framework 相关的泄露
- 03 XPC 相关的泄露



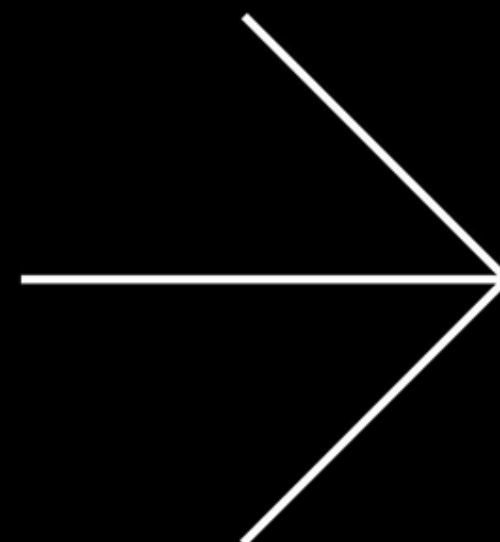
01 / 沙盒的权限泄露—沙盒与 Entitlements

- 沙盒可以限制的最小对象是进程
- 通过控制程序调用的系统 API 进而控制其行为
- 使用 Entitlements 控制例外情况
 - 比如：task_for_pid()
- 对于 AppStore 应用，Entitlements 由苹果控制
- 对于调试版的程序
 - MachO 中要嵌入相关的权限
 - app 目录中要包含 embedded.mobileprovision
 - embedded.mobileprovision 本身会被签名



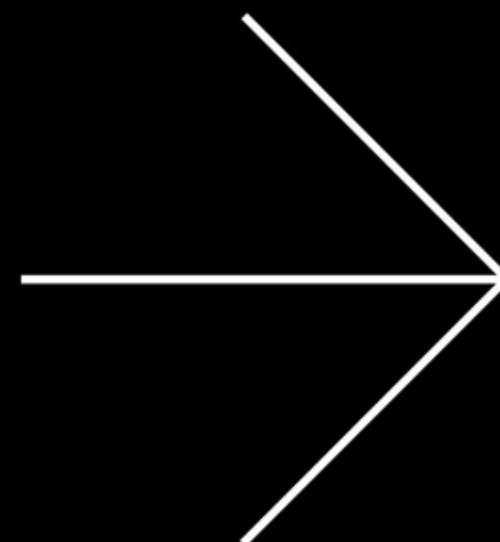
01 / 沙盒的权限泄露—沙盒与 Entitlements

- 用户空间也存在特权操作
 - 获取用户的账号信息
 - 管理用户安装的程序
- 这些特权操作通过 Framework 的 API 暴露出来
- 相关的 (Private)Framework 被 Daemon 与 App 共享
- 沙盒无法直接限制对库中 API 的调用



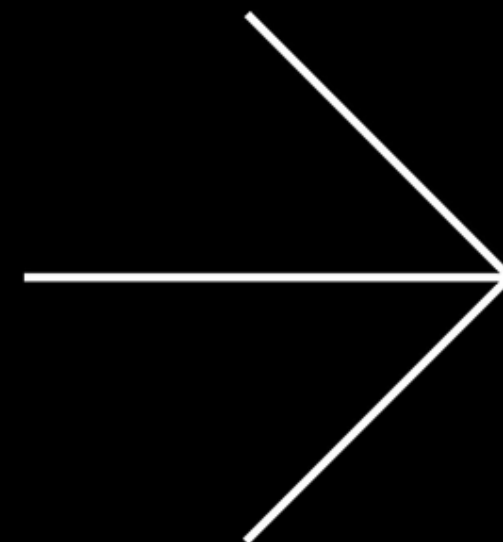
01 / 沙盒的权限泄露—沙盒与 Entitlements

- 解决方法：将特权操作封装到 Daemon 中，然后通过 Entitlements 进行限制
 - iOS 8 , Webkit & Nitro JavaScript Engine
- Daemon 需要显式检查调用者是否具有相应的 Entitlements
 - 如：com.apple.itunesstored.accounts，获取账号信息

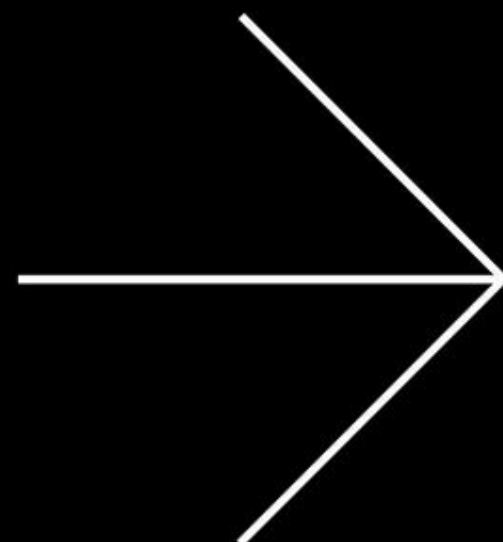
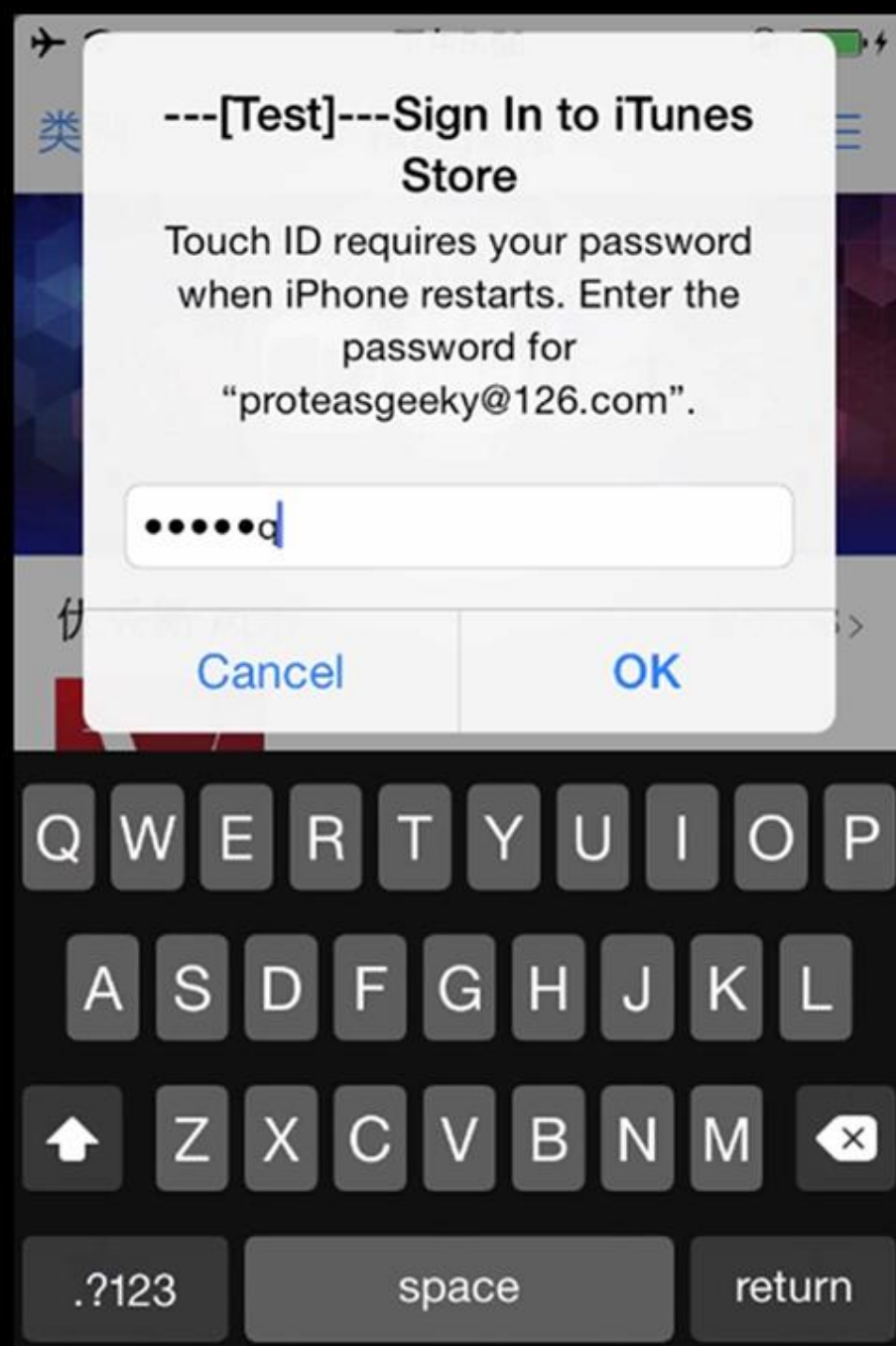


02 / 沙盒的权限泄露—Framework 相关的泄露

- 原因-1(Root Cause)：普通 App 可以使用 (Private) Framework , 可以调用私有 API
- 原因-2：遗漏了相关检查、没有正确处理异常输入等
- 例子
 - CVE-2014-4423：获取 iCloud 账号信息, Type: com.apple.account.Liverpool
 - 0 Day - LSAApplicationWorkspace：在非越狱手机上卸载 App, 已在 iOS 9 Beta 3 中修复
- 发现方法：跟踪、测试特权操作



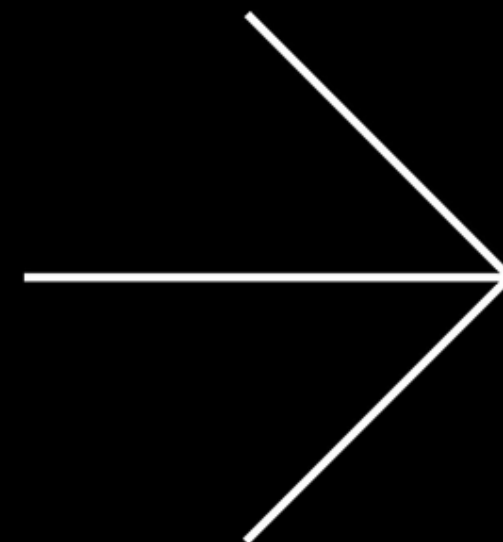
02 / 沙盒的权限泄露—Framework 相关的泄露



- 视频演示-CVE-2014-4423
- 视频演示 - 卸载 App

03 / 沙盒的权限泄露—XPC 相关的泄露

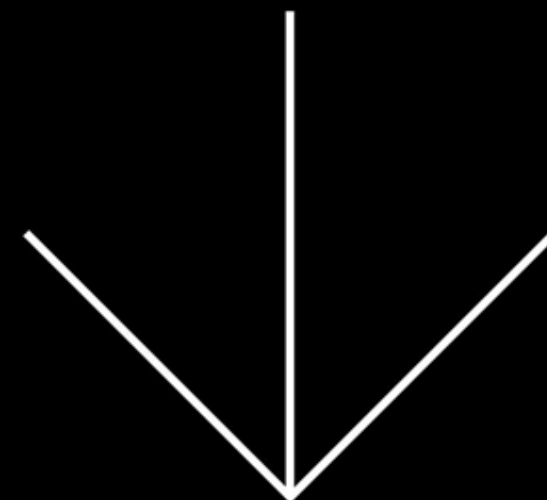
- iOS & OS X 中大量使用 XPC (IPC)
- 发现方法
 - 编写 Tweak 记录 XPC 及 Mach 服务
 - 测试是否可以与服务建立连接
 - 测试调用服务的接口：接口是否可以调用；接口能否处理异常数据；
- 例子
 - OS X - rootpipe: `-[WriteConfigClient authenticateUsingAuthorizationSync:nil]`
 - CVE-2014-4492 : iOS & OS X , lack of verification of the XPC dictionary data type



应用安全与攻击面

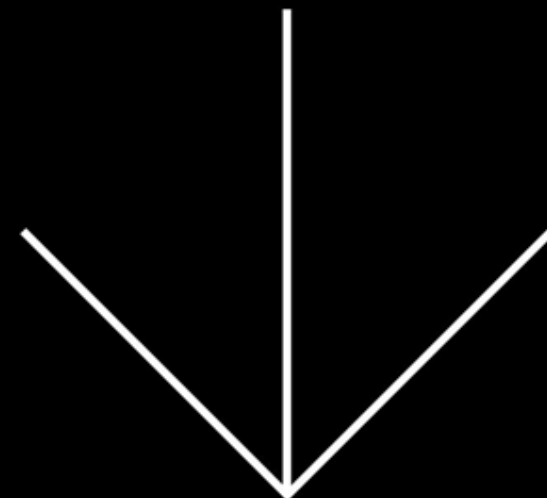
01 现状

02 攻击面



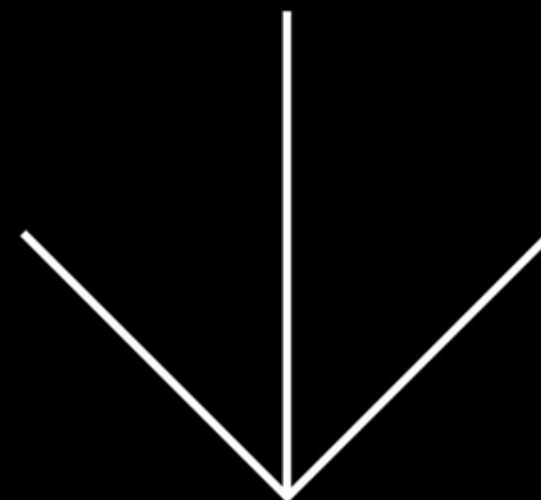
01 / 应用安全与攻击面—现状

- 应用无法获得 W+X 的内存页，无法加壳
- 大部分应用：
 - 没有做混淆
 - 没有反调试
 - 网络通信使用 HTTP
(iOS 9 要求使用 HTTPS)
 - 本地存储未加密
 - 敏感信息没有加密并存储到 Keychain



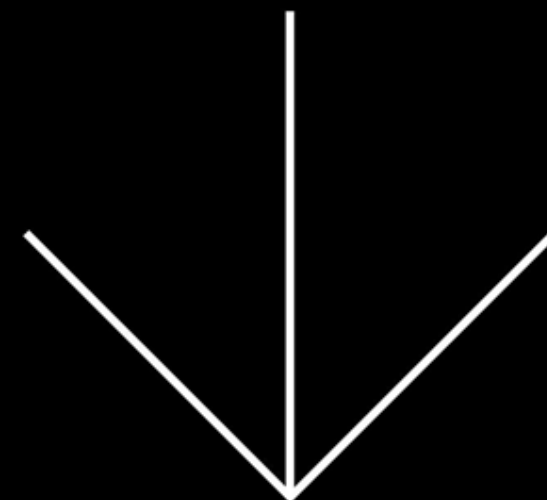
02 / 应用安全与攻击面—攻击面

- 几乎所有的移动应用都依赖网络来获取数据
- 针对特定应用的远程攻击需要结合“中间人”
 - iOS 9 要求使用 HTTPS
- 主要输入源
 - 自定义的数据流，socket
 - google protocol buffers
 - JSON，被大量使用，解析引擎相关问题（第三方库），业务数据相关问题
 - XML
 - 脚本，有些应用会内嵌脚本引擎并且下载、执行脚本



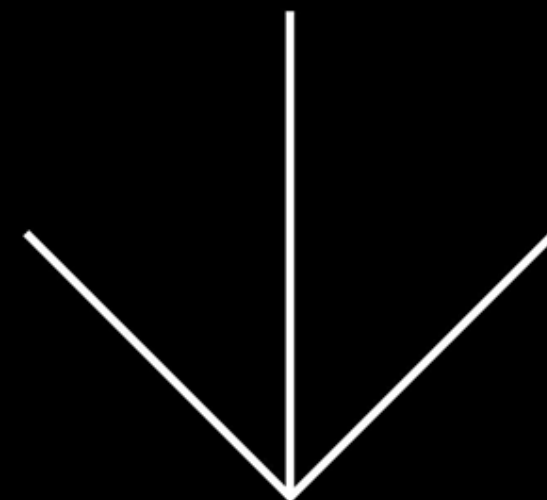
脚本型攻击

- 01 优势
- 02 如何选择目标



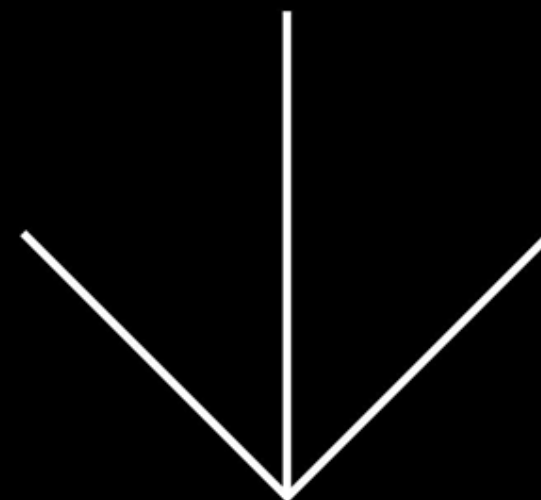
01 / 脚本型攻击—优势

- 脚本引擎破坏了 iOS 平台的基本保护机制
- 逻辑型漏洞
- 编写利用的难度与成本低
- 几乎可以完全控制 App



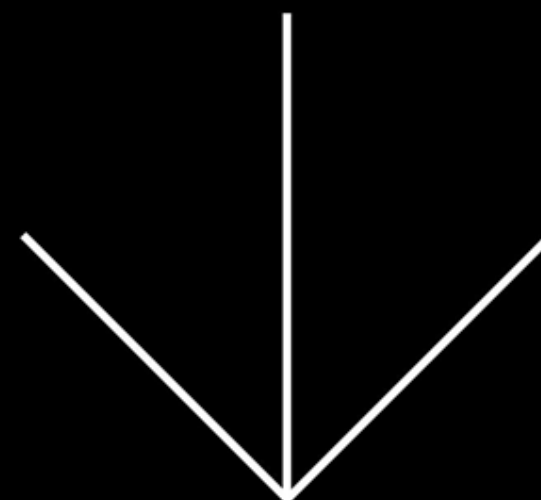
02 / 脚本型攻击—如何选择目标

- 将脚本语言与 ObjC 运行时做了绑定
- 可以覆盖现有类的方法
 - 支持 ObjC 分类
 - 支持 Patch 现有类方法
 - WaxPatch
 - JSPatch
 - nu-lisp
 - ...



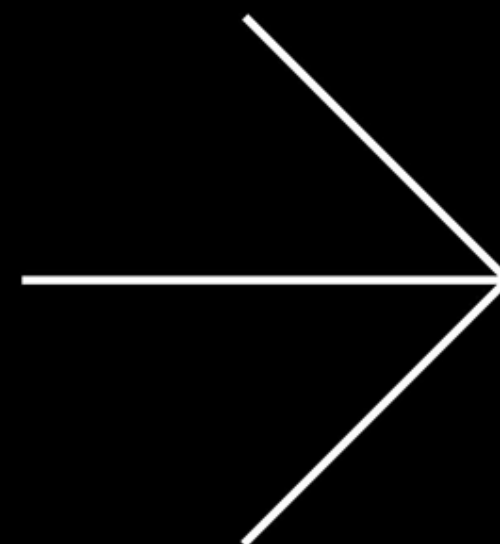
脚本型攻击实施方法

- ① 分析脚本的执行流程
- ② 分析网络通信
- ③ 构造 Payload
- ④ 结合“中间人”实施攻击

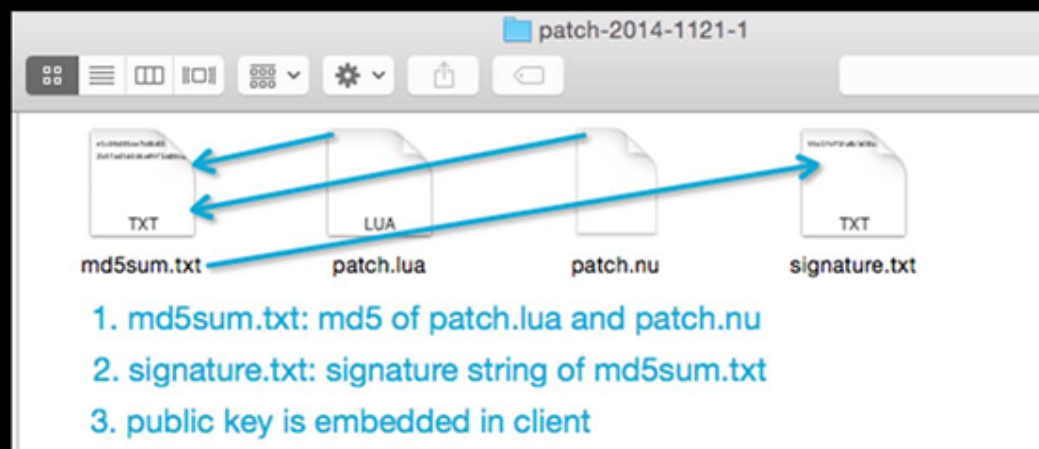


01 / 脚本型攻击实施方法—分析脚本的执行流程

- 确定是否下载脚本并执行
 - 程序可能在开源代码基础上做大量修改
- 确定如何校验脚本
 - 校验 MD5
 - 私有加密算法
 - 对称加密
 - 非对称加密
 - 密钥与具体设备相关（由于缺少设备唯一标识，这种方式几乎没有）



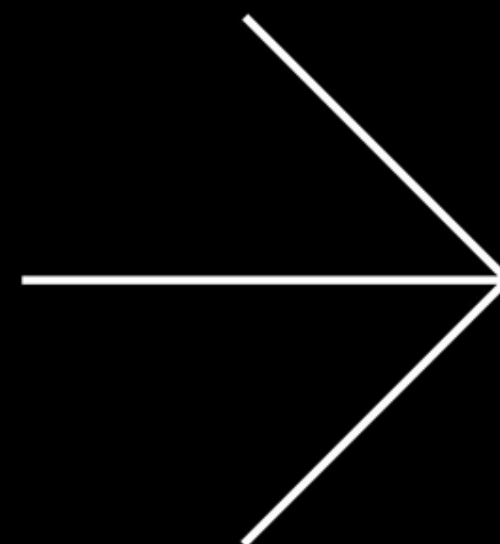
01 脚本型攻击实施方法—分析脚本的执行流程



- 确定脚本的加载时机
 - 在 didFinishLaunching 中加载
 - 从后台切到前台时加载
- 确定下载的包结构
 - 脚本一般会被 zip 压缩
 - zip 包会被校验 MD5
 - zip 包本身可能会被加密
 - zip 内的文件会被加密并校验 md5
 - 包中不只包含脚本，还可能包含相互校验的文件

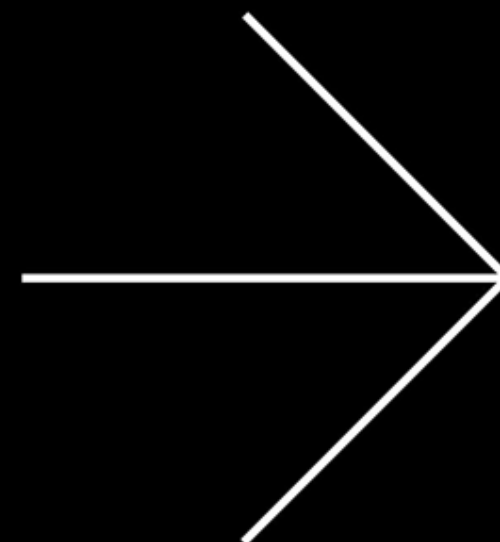
02 / 脚本型攻击实施方法—分析网络通信

- 一般不是从固定地址下载脚本：先从某个地址请求脚本的下载地址
- 先劫持控制脚本下发的请求
- 再劫持获取脚本的请求



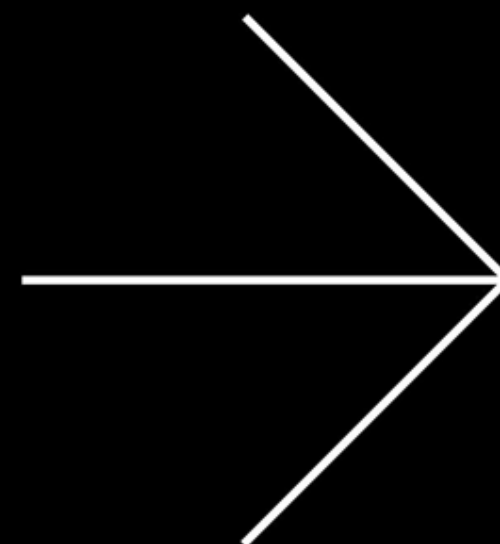
03 / 脚本型攻击实施方法—构造 Payload

- App as SDK : 利用程序本身的功能来实现 Payload 加密
- 对抗 : 私有加密算法、对称加密但密钥计算非常复杂
- LLDB Commands
 - \$0 = expression (id)[XXXSecretMgr sharedInstance]
 - \$1 = expression (id)[\$0 getAppKeyWithIos]
 - \$2 = expression (id)[NSData dataWithContentsOfFile:@"XXX"]
 - \$3 = expression (id)[XXXAlgrithm encryptByte:\$2 withAppKey:\$1]
 - expression (int)[\$3 writeToFile:@"XXX" atomically:0]

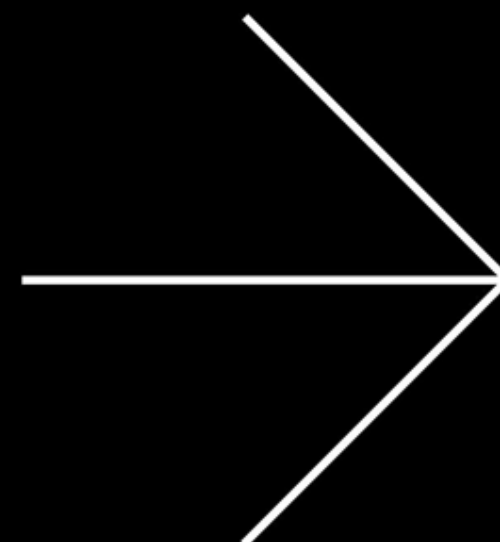
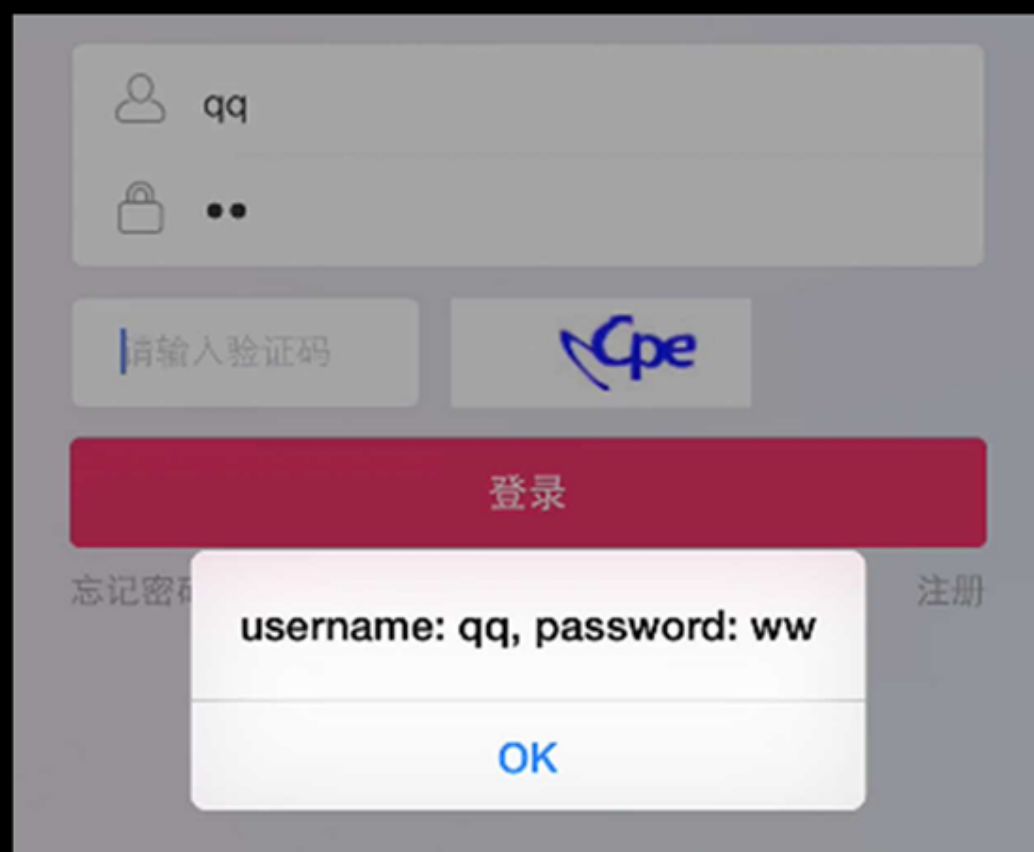


04 / 脚本型攻击实施方法—结合“中间人”实施攻击

- 调试：Charles
- 其他：DNS 投毒，恶意 WiFi 等



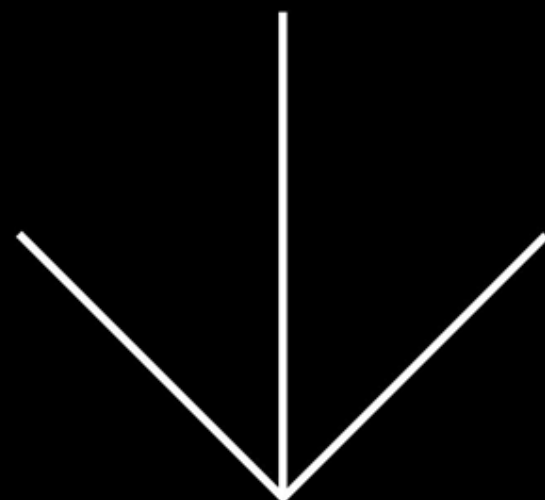
04 / 脚本型攻击实施方法—结合“中间人”实施攻击



- 视频演示-脚本型攻击-1
- 视频演示-脚本型攻击-2

脚本型攻击的防护建议

- ① 尽量不要使用可以修改现有类行为的脚本引擎
- ② 客户端不需“生产”脚本，只是“消费”脚本
- ③ 采用非对称的加密方法





THANKS!

