
ISC
2015
中国互联网安全大会
360互联网安全中心
ISC
2015
数据驱动安全
2015 中国互联网安全大会
China Internet Security Conference
ios安全体系演进过程
及漏洞分析

从XcodeGhost说起

- 非官方的XCode在编译生成App的同时植入额外的代码。新增代码可收集和上传用户部分隐私数据，也具备伪造弹窗、分发广告等功能

部分厂商和用户反应

当然，审核并不是 ios 最后的安全手段，最关键还是 ios 自身的沙盒隔离。审核只是多做一层防护，相当于开个白名单，减少沙盒的安全漏洞被利用的可能性。至于沙盒这最后一道(也是最重要的)防线，就要求用户你不要去搞什么越狱啦。

#XcodeGhost#来问我的人太多了，统一回复下：就目前已经披露的信息和现在的状态来说，本次安全事件现在对于一般终端用户没什么危害，尤其是非越狱用户。控制端服务器已经关闭（他们不想死的话估计不会再开），iOS沙盒机制保证了app即使作恶也能力有限。本事件对从业人员意义重大，但一般用户不必惊慌。

部分厂商和用户反应

iOS沙盒 ＋上架前审计 ＝ 安全

沙盒能完全保证设备安全么

iOS沙盒 ＋上架前审计 ≠ 安全

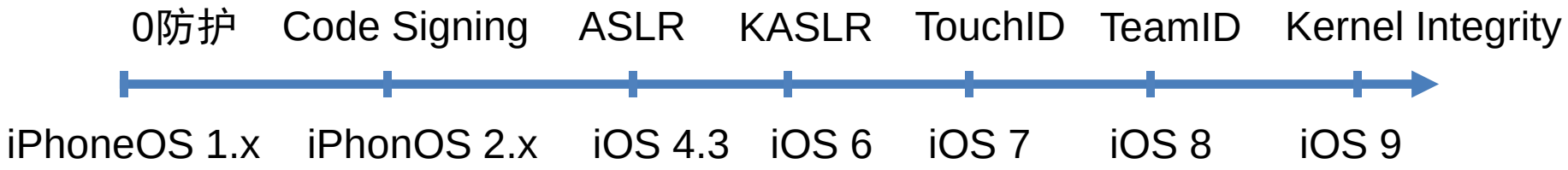
看个视频

完全信赖沙盒和审计 = 自废武功

回到主题

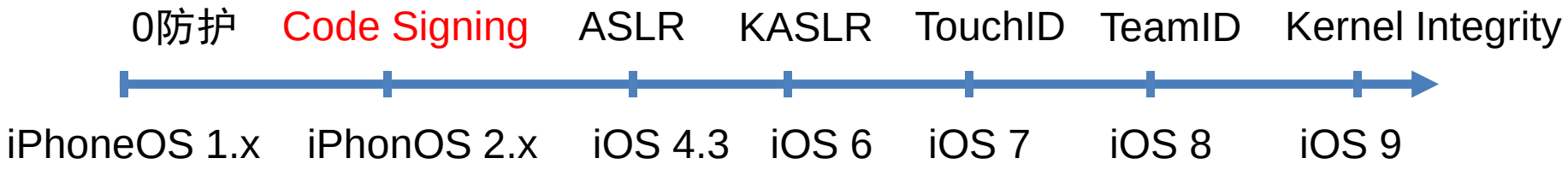
- iOS安全架构发展时间轴
- 审视iOS代码签名策略：从进程创建到第一条指令执行
- 总结

iOS安全架构发展时间轴



iOS安全体系经历了从无到有、从弱到强的发展

iOS安全架构发展时间轴



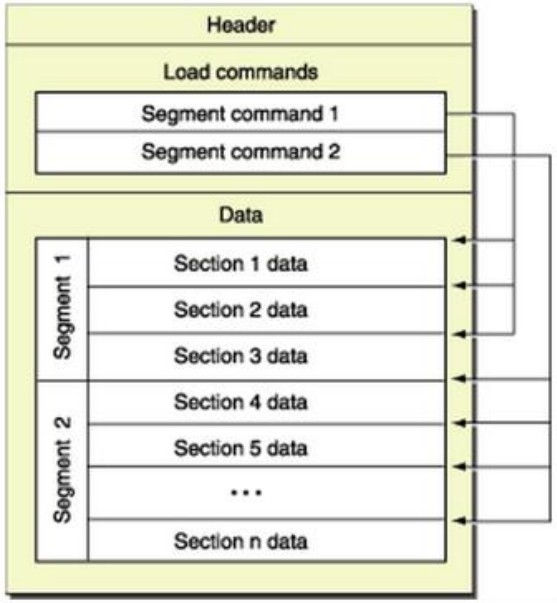
签名绕过漏洞对完美越狱非常关键

内容大纲

- iOS安全架构发展时间轴
- **审视iOS代码签名策略：从进程创建到第一条指令执行**
- 总结

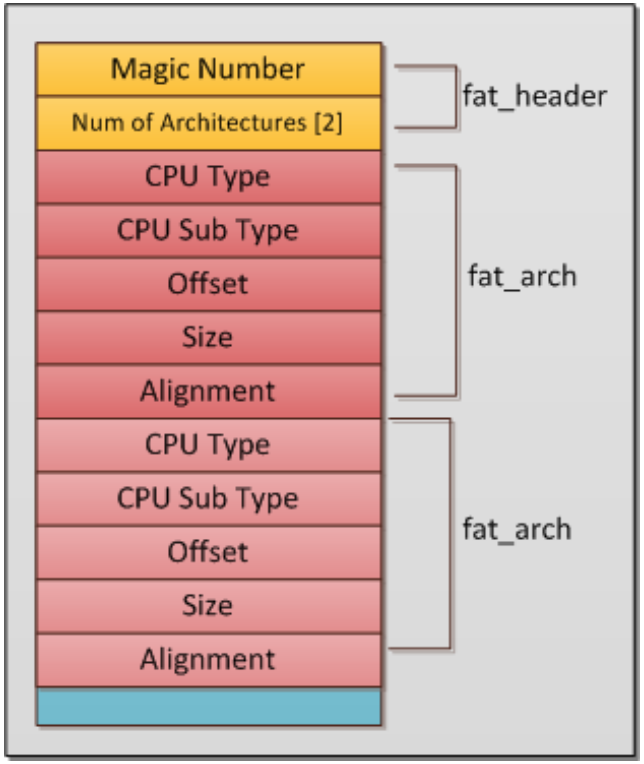
可执行文件格式

- Mach-O: iOS/OS X上的可执行文件格式



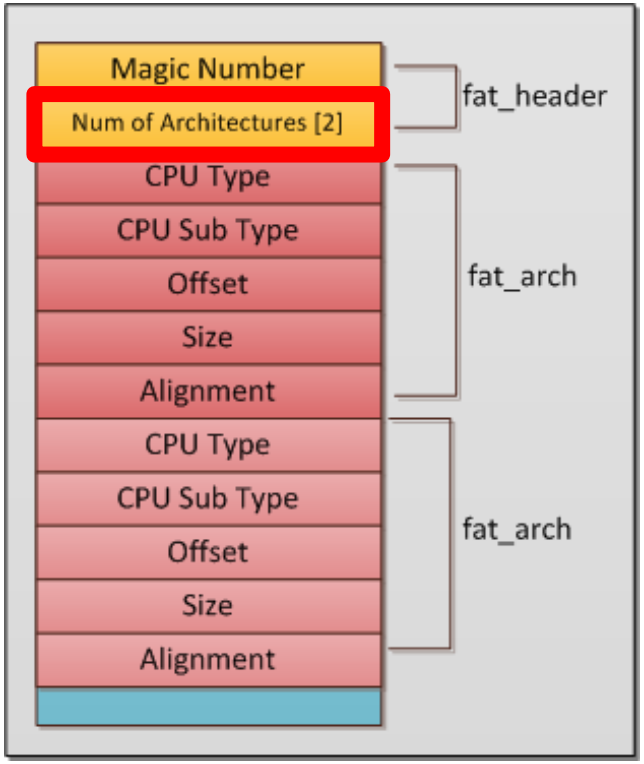
可执行文件格式

- FAT Binary: 多个不同CPU架构的Mach-O



内核负责加载主程序

- FAT Binary解析漏洞
  - 通过posix_spawn陷入内核时，内核会试图从FAT binary中选择最匹配的CPU架构
  - 内核没有严格验证FAT header中的arch数目



漏洞代码

```
static int
exec_fat_imgact(struct image_params *imgp)
{
...
        nfat_arch = OSSwapBigToHostInt32(fat_header->nfat_arch);

        /* make sure bogus nfat_arch doesn't cause chaos - 19376072 */
        if ( (sizeof(struct fat_header) + (nfat_arch * sizeof(struct fat_arch))) > PAGE_SIZE ) {
            error = EBADEXEC;
            goto bad;
        }

        /* Check each preference listed against all arches in header */
        for (pr = 0; pr < NBINPREFS; pr++) {
...
            for (f = 0; f < nfat_arch; f++) {
```

正常执行流程

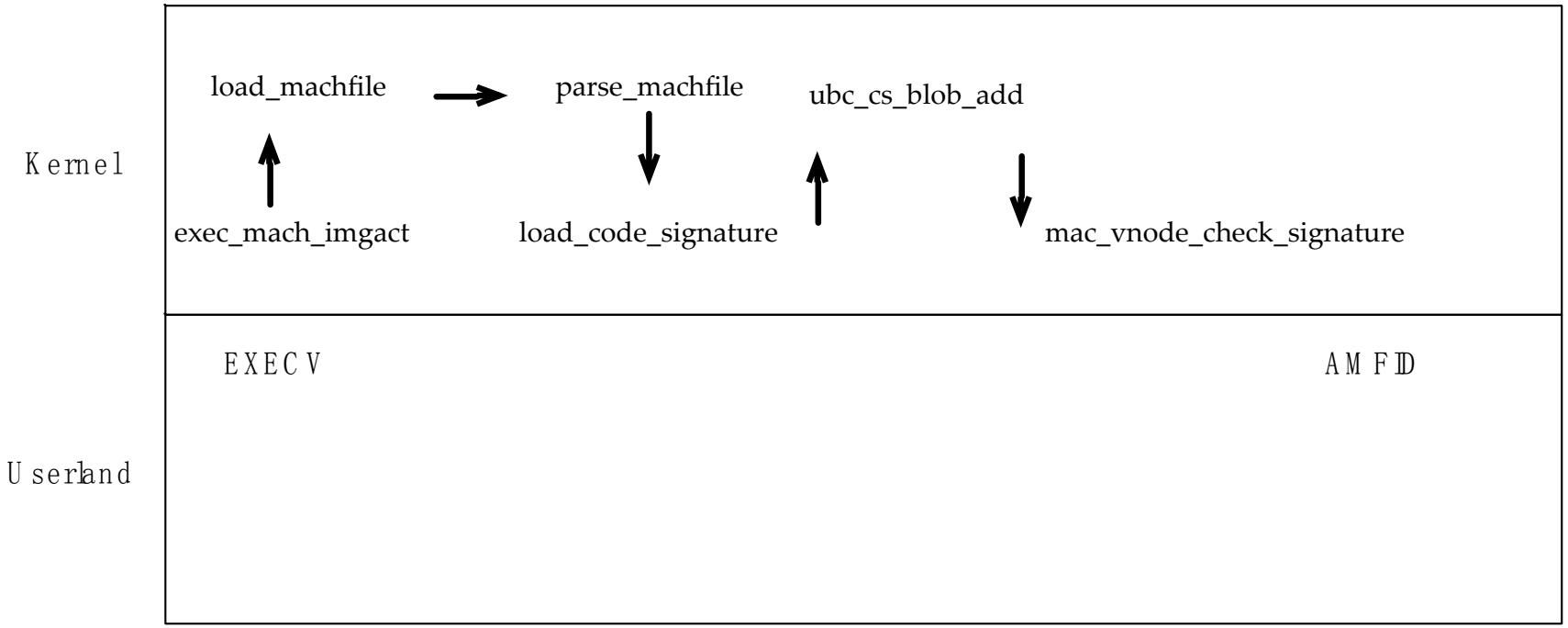
- 内核加载主程序
  - 主程序通过 LC_MAIN 命令指明主程序入口
- 内核加载dyld
  - dyld通过LC_UNIXTHREAD命令指明dyld的入口
- 内核完成加载后，将控制流交回dyld（用户态）
- dyld加载dylib后，将控制流交回主程序

iOS 8.2中的一个漏洞

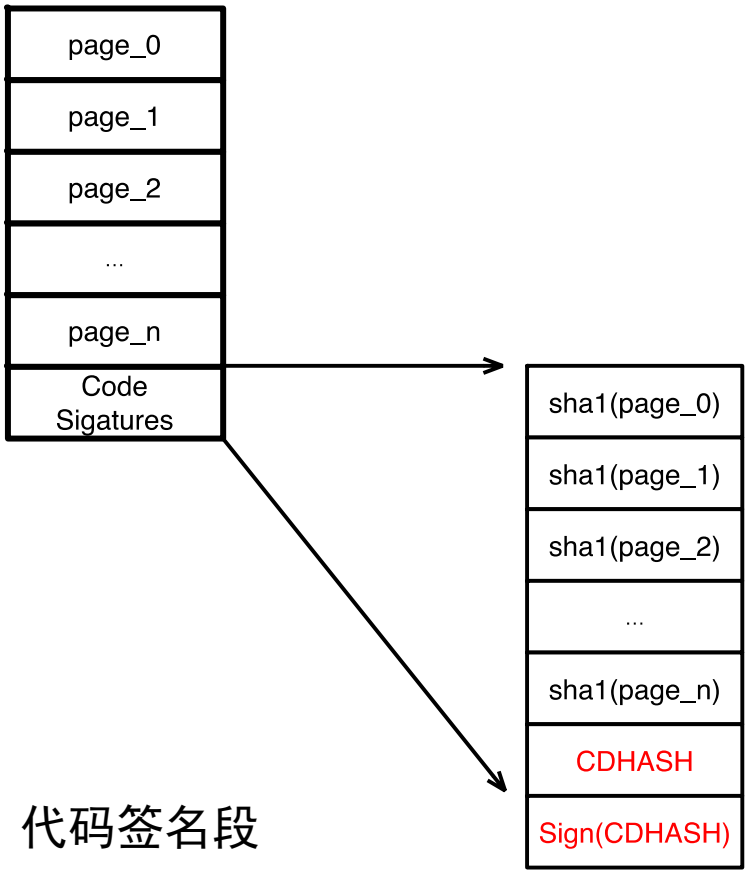
- 内核并未主动验证主程序MachO头的签名
  - 只有当dyld在用户态执行后才会去访问主程序MachO头，触发page fault而引起签名验证
- 攻击思路
  - 修改主程序的MachO头，强制其不加载dyld，同时设置LC_UNIXTHREAD命令，将用户态的第一条指令指向已有代码段，从而利用ROP的手段获得任意代码执行

iOS 8.3中的修复

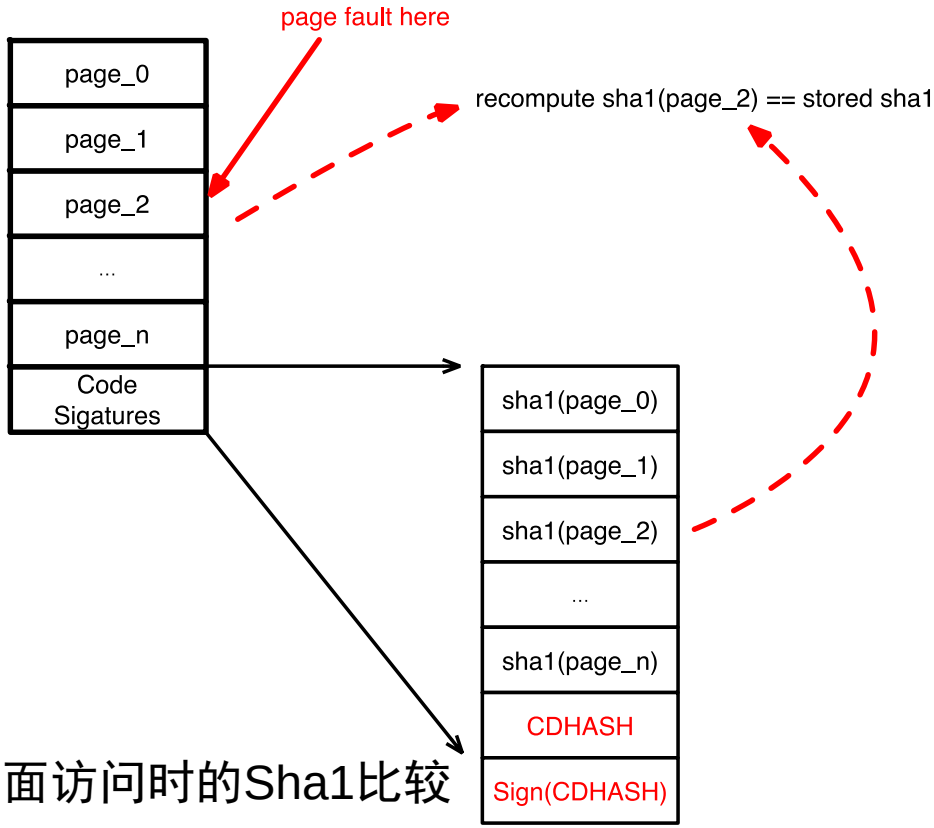
- 内核在注册主程序的代码签名段后就会访问MachO头，进而触发签名检查



签名的实现



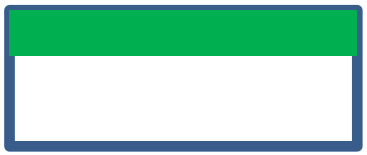
代码签名段



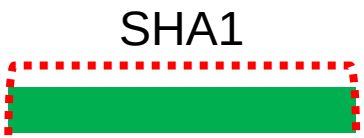
页面访问时的Sha1比较

iOS 8.3-8.4中的一个缺陷

- 代码签名实际校验是通过比对内存页面的SHA1值和预存在代码签名段中的SHA1值
- iOS以内存页面大小（pagesize）为单位，预先存储程序各部分的SHA1值
- 如果程序的大小不能被pagesize整除，在计算最后一个SHA1值时，iOS并没有进行填充，而是计算实际内容的SHA1



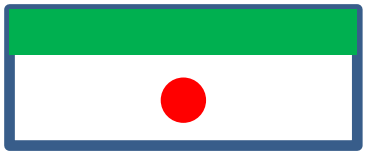
程序的最后一个page
绿色代表程序中原有内容

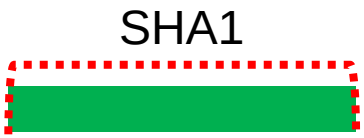


签名段中保存的SHA1值

iOS 8.3-8.4中的一个缺陷

- 在真实内容之外的区域，实际上并未被保护
- 如果对程序最后一个page修改，并不会违背签名验证





总结

- iOS的各种安全威胁一直存在，不能完全信赖iOS沙盒和苹果的审计
- 代码签名中的TOCTOU (Time of check to time of use)问题仍有诸多隐患
- iOS的从0安全防护到最强系统防护之路值得借鉴学习

ISC 2015 中国互联网安全大会

360互联网安全中心

谢谢
Q&A

ISC 2015 中国互联网安全大会

360互联网安全中心

谢谢
Q&A

ISC 2015 中国互联网安全大会

360互联网安全中心

谢谢
Q&A

ISC 2015 中国互联网安全大会

360互联网安全中心

谢谢
Q&A