KCon

纵横

FAIRPLAY DRM和混淆实现

演讲者：

卢俊志（@pwn0rz）、王馨笛（@1dnix）、朱学文（Ju Zhu）

DRM

Fairplay - DRM

介绍

数字版权保护
应用于电子书籍/音乐/视频
App DRM自2013年引入
私有代码，高度混淆

Fairplay - DRM

Load Command

```
$ otool -l target  | grep -i crypt
          cmd LC_ENCRYPTION_INFO_64
      cryptoff 16384
     cryptsize 4177920
       cryptid 1
```

Fairplay - DRM

Fairplay Open - From Kernel

Fairplay - DRM

Fairplay Open - MIG

FairplayIOKit  fairplayd 

```
#include <mach/std_types.defs>
#include <mach/mach_types.defs>

subsystem KernelUser unfreed 502;

type unk1_t = struct[136] of char;
type unk2_t = struct[84] of char;

routine fairplay_open(
    fairplay_port   : mach_port_t;
    executable_path : pointer_t;
    cpu_type        : uint32_t;
    cpu_subtype     : uint32_t;
    out supf        : pointer_t;
    out unk_ool2    : pointer_t;
    out unk1        : unk1_t;
    out unk2        : unk2_t;
    out supf_size   : uint32_t;
    out ool2_size   : uint32_t;
    out ukn3        : uint32_t);
```

Fairplay - DRM

Fairplay Open - fairplayd

```
$ tree
.
├── SC_Info
│   ├── target.sinf
│   └── target.supf
└── target
```

Fairplay - DRM

Fairplay Open - SINF

```
$ sinf_view.py SC_Info/target.sinf
sinf.frma: game
sinf.schm: itun
sinf.schi.user: 0xdeadbeef
sinf.schi.key : 0x00000002
sinf.schi.iviv: <***16 bytes IV***>
sinf.schi.righ.veID: 0x00012345
sinf.schi.righ.plat: 0x00000000
sinf.schi.righ.aver: 0x11223344
sinf.schi.righ.tran: 0x11223344
sinf.schi.righ.sing: 0x00000000
sinf.schi.righ.song: 0x11223344
sinf.schi.righ.tool: P550
sinf.schi.righ.medi: 0x00000080
sinf.schi.righ.mode: 0x00000000
sinf.schi.righ.hi32: 0x00000002
sinf.schi.name:<***null terminated username, 256 bytes***>
sinf.schi.priv: <***432 bytes encrypted data***>
sinf.sign: <***128 bytes signature***>
```

Fairplay - DRM

Fairplay Open - SUPF

```
$ supf_view.py SC_Info/target.supf
KeyPair Segments:
        Segment 0x0: arm_v7, Keys: 0x3d0/4k, sha1sum = <code_sig>
        Segment 0x1: arm64, Keys: 0x3fc/4k, sha1sum = <code_sig>

Fairplay Certificate: <RSA 1024 Ceritificate, valid since 2008, expire at 2013>

RSA Signature: <128 bytes>
```

Fairplay - DRM

Fairplay Open - QA时间

1. 使用了不安全的RSA密钥长度, 没有校验RSA证书的有效期
3. SINF中明文存储了用户身份标识信息(但是沙盒内无法读取)
4. 可以通过调用MIG + Hook来稳定获取Fairplayd运行中间过程
5. 可通过回归测试确定最终和DRM相关/无关的字段
6. SINF文件中sinf.sign字段不校验（仅在安装时通过installd校验）

Fairplay - DRM

Fairplay Decrypt - Kernel

Fairplay - DRM

Fairplay Decrypt - 一些细节

1. Fairplay 以page为单位解密，尺寸是4096 bytes
2. aes-128-cbc解密，密钥通过Fairplay Open的结果计算得出
3. 至少解密过程中没有涉及到HW AES(S8000)

Fairplay - DRM

Fairplay Decrypt - Demo

```
>>> calling fairplay_decrypt_page : EQlZPp(handle=0xe421a923,off=0x36e000,src=0x101e6e000,dst=0x10080e800)
0x10e2f0 ==> IOMallocAligned(0x4000,0x4b) => 0x100808800
0xfea5c ==> IOMalloc(0x68) => 0x100206c80
0xfa7a4 ==> IOLockLock(0x1002061d0)
0xfb91c ==> IOUnlock(0x1002061d0)
0xfa7a4 ==> IOLockLock(0x100206220)
0x40924 ==> aes_decrypt_key(key=0x100206c82,len=0x10,ctx=0x16fdff1d0)
0x4093c ==> aes_decrypt_cbc(in=0x101e6e000,iv=0x100206c92,n_blk=0x100,out=0x10080e800,ctx=0x16fdff1d0)
        aes-128-cbc key: 3ea01b29ecf8107fc2646b7f4acde653, iv: 0888e773bb76ef6989814f0733f1b2172
0xfb91c ==> IOUnlock(0x100206220)
0x10c894 ==> IOFree(0x100206c80,0x68)
0x10e608 ==> IOFreeAligned(0x100808800,0x4000)
>>> calling fairplay_decrypt_page : EQlZPp(handle=0xe421a923,off=0x36f000,src=0x101e6f000,dst=0x10080e800)
0x10e2f0 ==> IOMallocAligned(0x4000,0x4b) => 0x100808800
0xfea5c ==> IOMalloc(0x68) => 0x100206cc0
0xfa7a4 ==> IOLockLock(0x1002061d0)
0xfb91c ==> IOUnlock(0x1002061d0)
0xfa7a4 ==> IOLockLock(0x100206220)
0x40924 ==> aes_decrypt_key(key=0x100206cc2,len=0x10,ctx=0x16fdff1d0)
0x4093c ==> aes_decrypt_cbc(in=0x101e6f000,iv=0x100206cd2,n_blk=0x100,out=0x10080e800,ctx=0x16fdff1d0)
        aes-128-cbc key: 660c99e61ac45c562986c89d350f2f67, iv: 97a8d8992c938907f7fd89239c665fce
0xfb91c ==> IOUnlock(0x100206220)
0x10c894 ==> IOFree(0x100206c80,0x68)
0x10e608 ==> IOFreeAligned(0x100808800,0x4000)
Process_8124O9 exited with status = 0 (0x00000000)
```

混淆

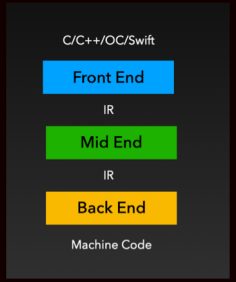
Fairplay – 混淆

编译优化 vs makeOpaque

编译优化：Constant Folding, Common Subexpression Elimination, Dead Code Elimination....

makeOpaque: 绕过编译优化

```
Expression* makeOpaque(Expression* in)
```



Fairplay – 混淆

makeOpaque: 不透明谓词

```
makeOpaque(true)

=>

uint32_t x = random();
( (x * x % 4) == 0  ||  (x * x % 4) ==1)
```

Fairplay – 混淆

makeOpaque: 不透明谓词之
BogusCFG

```
if(makeOpaque(true)){
    real_block();
}else{
    fake_block();
}
```

Fairplay – 混淆

makeOpaque: 不透明常量之可逆变换

```
//对于互为模反元素的a: 4872655123和ra: 3980501275， 取
uint32_t x = random();
uint32_t c = 0xbeefbeef;
//则 -ra * c = 0x57f38dcb, 满足
((x * 4872655123) + 0xbeefbeef ) * 3980501275 + 0x57f38dcb == x
```

Fairplay – 混淆

makeOpaque:不透明常量之MBA表达式

```
//OperationSet(+, - , * , & , | , ~)
makeOpauqe(x – c) => (x ^ ~c)  + ( (2 * x) & ~(2 * c + 1) ) + 1;
```

Fairplay – 混淆

makeOpaque:不透明常量应用-
IndirectBranch

```
//OperationSet(+, - , * , & , | , ~)
jmp branch;
=>
jmp global_branch_lut[index];
=>
jmp global_branch_lut[makeOpauqe(index)];
```

Fairplay – 混淆

静态恢复实战 – Call Graph 恢复

Indirect Branch + Call Convention混淆的同时对参数进行了混淆（父函数加密，子函数解密，利用LLVM不进行Inter-procedure分析的特性）

```
__text:00000000000043B88 E1 0F 00 F9          STR             X1, [SP,#0x90+var_78]
__text:00000000000043B8C A8 63 00 D1          SUB             X8, X29, #-var_18
__text:00000000000043B90 09 0D B8 52+         MOV             W9, #0x36724068
__text:00000000000043B94 09 CE A6 72          MOVK            W9, #0x3672,LSL#16
__text:00000000000043B98 08 01 09 4A          EOR             W8, W8, W9
__text:00000000000043B9C A9 6C B6 52+         MOV             W9, #0x1AA53365
__text:00000000000043BA0 A9 54 A1 72          MOVK            W9, #0xAA53,LSL#16
__text:00000000000043BA4 08 7D 09 1B          MUL             W8, W8, W9
__text:00000000000043BA8 09 31 96 52+         MOV             W9, #0xB9548B88
__text:00000000000043BAC 89 2A B8 72          MOVK            W9, #0xC154,LSL#16
__text:00000000000043BB0 08 01 09 4A          EOR             W8, W8, W9
__text:00000000000043BB4 A8 83 1F B8          STUR            W8, [X29,#var_10]
__text:00000000000043BB8 E8 23 00 91          ADD             X8, SP, #0x90+var_88
__text:00000000000043BBC A8 83 1E F8          STUR            X8, [X29,#var_18]
__text:00000000000043BC0 68 07 00 B0+         ADRL            X8, off_1383F0  ; _fp_dh_d2422d6a5f4779457076ea15ed54da5a1
__text:00000000000043BC8 08 C1 0F 91          LDR             X9, [X8]  ; __fp_dh_d2422d6a5f4779457076ea15ed54da5a1
__text:00000000000043BCC 09 01 40 F9          MOVK            X8, #0x81FD,LSL#48
__text:00000000000043BD0 A0 63 00 D1          SUB             X0, X29, #-var_18
__text:00000000000043BD4 28 09 3F D7          BLRAA           X9, X8
__text:00000000000043BD8 A0 43 5F B8          LDUR            W0, [X29,#var_C]
__text:00000000000043BDC A8 83 5F F8          LDUR            X8, [X29,#var_8]
__text:00000000000043BE0 29 07 00 B0          ADRP            X9, #off_1281680@PAGE
__text:00000000000043BE4 29 B5 40 F9          LDR             X9, [X9,#off_1281680@PAGEOFF]
__text:00000000000043BE8 29 01 40 F9          LDR             X9, [X9]
__text:00000000000043BEC 3F 01 08 EB          CMP             X9, X8
__text:00000000000043BF0 81 00 00 54          B.NE            loc_43C00
__text:00000000000043BF4 FD 7B 49 A9          LDP             X29, X30, [SP,#0x90+var_s0]
__text:00000000000043BF8 FF 83 02 91          ADD             SP, SP, #0xA0
```

Fairplay – 混淆

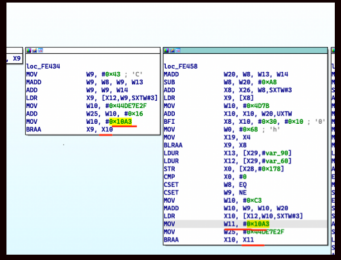
静态恢复实战 – Call Graph 恢复

参数混淆恰巧在父子函数中引入相同随机数，让我们得以根据这一特性恢复出调用关系

Fairplay – 混淆

静态恢复实战 – 尝试恢复CFG

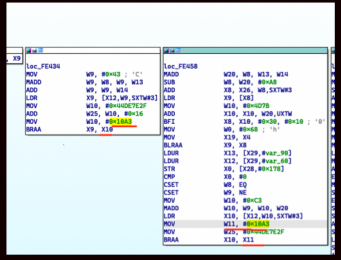
1. 使用了Indirect Branch混淆机制
2. 同一个函数的每个基本快具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本快之前目前仍然是孤立的，需要动态恢复



Fairplay – 混淆

静态恢复实战 – 尝试恢复CFG

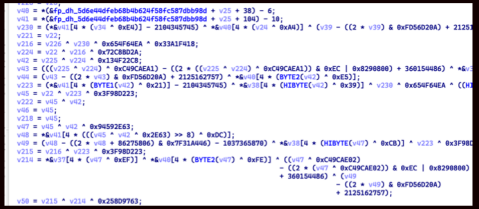
1. 使用了Indirect Branch混淆机制
2. 同一个函数的每个基本快具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本快之前目前仍然是孤立的，需要动态恢复



Fairplay – 混淆

静态恢复实战 – 其他未解决的

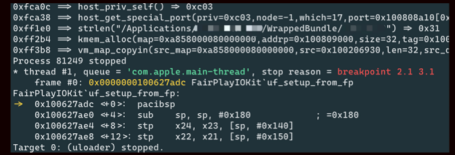
1. 基于不透明常量的数据流混淆, 目前未找到其生成规则



Fairplay – 混淆

动态调试工具-穷人的”内核驱动”调试器

1. 把FairplayIOKit内核驱动加载到用户态
2. 通过dyld的机制通知调试器新加载的内核扩展
3. 开始调试



Fairplay – 混淆

动态调试工具-执行流跟踪

可以记录自己的执行路径(trapfuzz类似)

可以记录很多次非直接跳转的结果(trapfuzz不支持)

不能single-step自身

从DTrace中获取灵感: exception-emulation-recover

```
static void breakpoint_handler(int signnum, siginfo_t* info, void *context){
    // get breakpoint address and context
    uint32_t *pc = (uint32_t*)info->si_addr;
    ucontext_t *ctx = (ucontext_t*)context;

    uint32_t *saddr = shadow_addr(pc);
    uint32_t opcode = *saddr;

    uint64_t rn = 0;
    uint64_t ctx_data = 0;
    uint64_t ncov = 0;

    //now let's interpret arm64 instructions lol
    if((opcode & 0xfffffc00) == 0xd73f0800){ /*blraa*/
        // lr = pc + 4
        // br rn

        rn = (opcode >> 5) & 0x1f;
        ctx_data = ctx->uc_mcontext->__ss.__x[rn]; //blr target

        ctx->uc_mcontext->__ss.__lr = (uintptr_t)pc + 4;
        ctx->uc_mcontext->__ss.__pc = ctx_data;
```

Fairplay – 混淆

动态调试工具-Demo

```
0×d2fd8: cset true
0×d2fdc: cset false
0×d3004: b 0×d3008
0×d3014: cset true
0×d3018: cset false
0×d3040: b 0×d3048
0×d3080: bl 0×3d4f0
0×3d504 ⟹ memcpy(dst=0×11c80a540,src=0×120008010,size=0×10)
0×d30a8: b 0×d3114
0×d3128: cset false
0×d3130: cset false
0×d313c: cset false
0×d3180: b 0×d3184
0×d31d0: bl 0×3d4f0
0×3d504 ⟹ memcpy(dst=0×11c80a090,src=0×120008000,size=0×20)
```

Fairplay – 混淆

动态调试工具-更多可能

WIP 反射式macho注入
WIP 无源代码的macho二进制Profiler工具

Update @ https://github.com/pwn0rz/fairplay_research

感谢观看！

KCon 汇聚黑客的智慧

知道创宇 | KCon