

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



```
#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);
```



fairplayd

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 Certificate, 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

text_crypter_create

```
/*  
 * Interface for text decryption family  
 */  
struct page_crypt_info {  
    /* Decrypt one page */  
    int (*page_decrypt)(const void *src_addr, void *dst_addr,  
                      unsigned long long src_offset, void *crypt_key);  
    /* Page using this crypter terminates - crypt module not needed anymore */  
    void (*crypt_end)(void *crypt_key);  
    /* Private data for the crypter */  
    void *crypt_key;  
    volatile int crypt_refcnt;  
};
```

IOTextCrypter::decryptPage

AppleFairplayTextCrypter::decryptPage

com_apple_driver_FairPlayIOKit::bvqhJ

EQIZPp

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(0x4000,0x40) => 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: 3ea01b294cf8107fc2646b7f6acde653, iv: 0888e773bb76ef698981d0733f1b2172
0xfb91c => IOUnlock(0x100206220)
0x10c094 => IOFree(0x100206c80,0x68)
0x10e608 => IOFreeAligned(0x100808800,0x4000)
>>> calling fairplay decrypt page : EQlZPp(handle=0xe421a923,off=0x36f000,src=0x101e6f000,dst=0x10080e800)
0x10e2f0 => IOMallocAligned(0x4000,0x40) => 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=0x101e6f000, iv=0x100206c92, n_blk=0x100, out=0x10080e800, ctx=0x16fdff1d0)
aes-128-cbc key: 660c99e61ac45c562986c89d350f2f67, iv: 97a8d8992c938907ffffd89239c665fce
0xfb91c => IOUnlock(0x100206220)
0x10c094 => IOFree(0x100206c80,0x68)
0x10e608 => IOFreeAligned(0x100808800,0x4000)
Process 81249 exited with status = 0 (0x00000000)

```

混淆

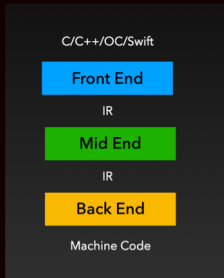
Fairplay – 混淆

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(+, -, *, &, |, ~)
makeOpaque(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[makeOpaque(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:000000000043888 E1 0F 00 F9      STR     X1, [SP,#0x90+var_78]
__text:00000000004388C A8 63 00 D1      SUB     X8, X29, #-var_18
__text:000000000043890 09 0D 88 52+    MOV     W9, #0x36724868
__text:000000000043894 49 CE A6 72      EOR     W8, W8, W9
__text:000000000043898 08 01 09 4A      MOV     W9, #0x1AA53365
__text:00000000004389C A9 6C 86 52+    MOV     W9, #0xD9548188
__text:0000000000438A4 08 7D 09 1B      MUL     W8, W8, W9
__text:0000000000438A8 09 31 96 52+    MOV     W9, #0xD9548188
__text:0000000000438B0 08 01 09 4A      EOR     W8, W8, W9
__text:0000000000438B4 A8 03 1F 88      STUR   W8, [X29,#var_10]
__text:0000000000438B8 E8 23 00 91      ADD     X8, SP, #0x90+var_88
__text:0000000000438BC A8 83 1E F8      STUR   X8, [X29,#var_18]
__text:0000000000438C0 68 07 00 80+    ADRL   X8, off_1303F0
__text:0000000000438C4 08 C1 0F 91      LDR     X9, [X8] ; _fp_dh_d2422d6a5f479457076ea15ed54da5a1
__text:0000000000438C8 09 01 40 F9      MOVK   X8, #0xB1FD,LSL#48
__text:0000000000438CC A8 3F F6 F2      SUB     X8, X29, #-var_18
__text:0000000000438D0 A0 63 00 D1      BLRAA  X9, X8
__text:0000000000438D4 28 09 3F D7      LDUR   W8, [X29,#var_C]
__text:0000000000438D8 A0 43 5F 88      LDUR   X8, [X29,#var_8]
__text:0000000000438DC A8 83 5F F8      ADRP   X9, #off_12B168@PAGE
__text:0000000000438E0 29 07 00 80      LDR     X9, [X9,#off_12B168@PAGEOFF]
__text:0000000000438E4 29 85 40 F9      LDR     X9, [X9]
__text:0000000000438E8 29 01 40 F9      CMP     X9, X8
__text:0000000000438EC 3F 01 08 EB      B.NE   loc_43C00
__text:0000000000438F0 81 00 00 54      LDP    X29, X30, [SP,#0x90+var_s0]
__text:0000000000438F4 FD 7B 49 A9      ADD     SP, SP, #0xA0
__text:0000000000438F8 FF 83 02 91

```

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

__text:0000000000043888 E1 0F 00 F9	STR	X1, [SP, #0x90+var_78]
__text:000000000004388C A8 63 00 D1	SUB	X8, X29, #-var_18
__text:0000000000043890 09 0D 88 52+	MOV	W9, #0x36724868
__text:0000000000043890 49 CE A6 72	EOR	W8, W8, W9
__text:0000000000043890 08 01 00 1A	MOV	W9, #0x1AA53365
0000000000043894 7F 23 03 56	STP	X28, X27, [SP, #-0x20+var_50]
0000000000043894 FC 6F 8A AD	STP	X26, X25, [SP, #-0x50+var_40]
0000000000043894 FA 67 81 AD	STP	X24, X23, [SP, #-0x50+var_30]
0000000000043894 F8 5F 81 AD	STP	X22, X21, [SP, #-0x50+var_20]
0000000000043894 F6 57 81 AD	STP	X20, X19, [SP, #-0x50+var_10]
0000000000043894 F4 4F 81 AD	STP	X29, X30, [SP, #-0x50+var_s0]
0000000000043894 F2 47 81 AD	ADD	X29, SP, #0x50
0000000000043894 F0 43 81 91	SUB	SP, SP, #0x100
0000000000043894 FF 03 97 D1	MOV	X19, X8
0000000000043894 F7 03 90 AA	ADD	X24, SP, #-0x210+var_170
0000000000043894 F6 03 82 91	ADRP	X8, #off_12B160@PAGE
0000000000043894 F4 03 80 90	LDR	X8, [X8, #off_12B160@PAGEOFF]
0000000000043894 F2 05 40 F9	LDR	X8, [X8]
0000000000043894 F0 01 40 F9	STUR	X8, [X29, #var_60]
0000000000043894 E8 03 1A F8	MOV	W25, #0x1AA53365
0000000000043894 E9 0C 86 52+	MOV	W23, #0x36724868
0000000000043894 E7 04 A3 72	MOV	W8, #0x36724868
0000000000043894 E7 05 A4 72	EOR	W8, W9, W8
0000000000043894 E8 00 80 52+	MUL	W8, W8, W25
0000000000043894 E8 CE A6 72	LDR	W9, [X8, #8]
0000000000043894 E8 82 80 4A	LDUR	W11, W9, W8
0000000000043894 E8 70 19 28	LDR	X13, [X8]
0000000000043894 E9 08 40 89	LDR	X8, [X13, #0x30]
0000000000043894 E8 01 80 4A	LDR	X20, [X13, #0x10]
0000000000043894 E0 08 40 F9	LDR	X9, [X13]
0000000000043894 A8 19 40 F9	ADD	W21, W11, W23
0000000000043894 04 40 40 F9	SUB	W10, W21, #0x40E
0000000000043894 A9 01 40 F9	ADRL	X12, _fp_dh_15d3f11e5574fba30266ee9e200f924
0000000000043894 75 01 17 00	LDR	X10, [X12, W10, SXTW#3]
0000000000043894 AA 3A 29 51	SUB	X22, X10, #0xC
0000000000043894 2C 04 00 90+	MOV	W10, #0x58C8
0000000000043894 8C 01 12 91	BRAA	X9, X10
0000000000043894 8A 09 6A F8	LDR	X9, [X8]
0000000000043894 54 39 00 D1	SUB	X29, X30, [SP, #0x90+var_s0]
0000000000043894 6A 19 80 52	LDP	X29, X30, [SP, #0x90+var_s0]
0000000000043894 2A 09 1F 07	ADD	SP, SP, #0xA0

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块具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本块之前目前仍然是孤立的，需要动态恢复

```
loc_FE434
MOV     W9, #0x43 : 'C'
MADD   W9, W9, W9, W13
ADD    W9, W9, W14
LDR    X9, [X12, W9, SXTW#3]
MOV    W10, #0x44DE7E2F
ADD    W25, W10, #0x16
MOV    W10, #0x16A3
BRAA   X9, X10

loc_FE458
MADD   W20, W8, W13, W14
SUB    W8, W20, #0xA8
ADD    X8, X26, W8, SXTW#3
LDR    X9, [X8]
MOV    W10, #0x4D7B
ADD    X10, X10, W20, UXTW
BFI    X8, X10, #0x30, #0x10 : '0'
MOV    W9, #0x68 : 'h'
MOV    X19, X4
BLRAA  X9, X8
LDUR   X13, [X29, #var_90]
LDUR   X12, [X29, #var_60]
STR    X0, [X28, #0x170]
CMP    X0, #0
CSET   W8, EQ
CSET   W9, NE
MOV    W10, #0xC3
MADD   W10, W9, W10, W20
LDR    X10, [X12, W10, SXTW#3]
MOV    W11, #0x16A3
MOV    W25, #0x44DE7E2F
BRAA   X10, X11
```

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块具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本块之前目前仍然是孤立的，需要动态恢复

The image shows two side-by-side windows of assembly code. The left window, titled 'loc_FE434', contains instructions for basic block 'loc_FE434'. The right window, titled 'loc_FE458', contains instructions for basic block 'loc_FE458'. Both windows show instructions with PAC modifiers (e.g., #0x43, #0x45) and branch instructions (BRAA) that jump to other locations. The modifiers are highlighted in red in the original image.

```
loc_FE434
MOV     W9, #0x43 : 'C'
MADD   W9, W9, W9, W13
ADD    W9, W9, W14
LDR    X9, [X12, W9, SXTW#3]
MOV    W10, #0x44DE7E2F
ADD    W25, W10, #0x16
MOV    W10, #0x16A3
BRAA   X9, X10

loc_FE458
MADD   W20, W8, W13, W14
SUB    W8, W20, #0xA8
ADD    X8, X26, W8, SXTW#3
LDR    X9, [X8]
MOV    W10, #0x4D7B
ADD    X10, X10, W20, UXTW
BFI    X8, X10, #0x30, #0x10 : '0'
MOV    W0, #0x68 : 'h'
MOV    X19, X4
BLRAA  X9, X8
LDUR   X13, [X29, #var_90]
LDUR   X12, [X29, #var_60]
STR    X0, [X28, #0x170]
CMP    X0, #0
CSET   W8, EQ
CSET   W9, NE
MOV    W10, #0xC3
MADD   W10, W9, W10, W20
LDR    X10, [X12, W10, SXTW#3]
MOV    W11, #0x16A3
MOV    W25, #0x44DE7E2F
BRAA   X10, X11
```


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

```

v40 = *(&fp_dh_5d6e44df68b4b624f58fc587dbb98d + v25 + 38) - 6;
v41 = *(&fp_dh_5d6e44df68b4b624f58fc587dbb98d + v25 + 104) - 10;
v230 = (*&v41[4 * (v34 * 0xE4)] - 2104345745) * *&v40[4 * (v24 * 0xA4)] * (v39 - ((2 * v39) & 0xFD56D20A) + 2125162757);
v221 = v22;
v216 = v226 * v230 * 0x654F64EA * 0x33A1F418;
v224 = v22 * v216 * 0x72C8BD2A;
v42 = v225 * v224 * 0x134F22CB;
v43 = (((v225 * v224) * 0xC49CAEA1) - ((2 * ((v225 * v224) * 0xC49CAEA1)) & 0xEC | 0x8290800) + 360154486) * *&v39[4 * (v24 * 0xA4)];
v44 = (v43 - ((2 * v43) & 0xFD56D20A) + 2125162757) * *&v40[4 * (BYTE2(v42) * 0xE5)];
v223 = (*&v41[4 * (BYTE1(v42) * 0x21)] - 2104345745) * *&v38[4 * (HIBYTE(v42) * 0x39)] * v230 * 0x654F64EA * ((HIBYTE(v42) * 0x39) + 0x3F98D223);
v45 = v22 * v223 * 0x3F98D223;
v222 = v45 * v42;
v46 = v45;
v218 = v45;
v47 = v45 * v42 * 0x94592E63;
v48 = *&v41[4 * (((v45 * v42 * 0x2E63) >> 8) * 0xDC)];
v49 = (v48 - ((2 * v48 + 86275806) & 0x7F31A446) - 1037365870) * *&v38[4 * (HIBYTE(v47) * 0xCB)] * v223 * 0x3F98D223;
v215 = v216 * v223 * 0x3F98D223;
v214 = *&v37[4 * (v47 * 0xEF)] * *&v40[4 * (BYTE2(v47) * 0xFE)] * ((v47 * 0xC49CAEB2) - ((2 * (v47 * 0xC49CAEB2)) & 0xEC | 0x8290800) + 360154486) * (v49 - ((2 * v49) & 0xFD56D20A) + 2125162757);
v50 = v215 * v214 * 0x258D9763;

```

Fairplay – 混淆

动态调试工具-穷人的“内核驱动”调试器

1. 把FairplayIOMKit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

```
0xfca0c => host_priv_self() => 0xc03
0xfca38 => host_get_special_port(priv=0xc03,node=-1,which=17,port=0x100808a10[0>
0xff1e0 => strlen("/Applications/▯▯▯▯/WrappedBundle/▯▯▯") => 0x31
0xff2b4 => kmem_alloc(map=0xa858000080000000,addrp=0x100809000,size=32,tag=0x100
0xff3b8 => vm_map_copyin(src_map=0xa858000080000000,src=0x100206930,len=32,src_c
Process 81249 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 2.1 3.1
   frame #0: 0x0000000100627adc FairPlayIOMKit`uf_setup_from_fp
FairPlayIOMKit`uf_setup_from_fp:
-> 0x100627adc <+0>: pacibsp
   0x100627ae0 <+4>: sub    sp, sp, #0x180           ; =0x180
   0x100627ae4 <+8>: stp   x24, x23, [sp, #0x140]
   0x100627ae8 <+12>: stp   x22, x21, [sp, #0x150]
Target 0: (uloader) stopped.
```

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 signum, 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

```
0xd2fd8: cset true
0xd2fdc: cset false
0xd3004: b 0xd3008
0xd3014: cset true
0xd3018: cset false
0xd3040: b 0xd3048
0xd3080: bl 0x3d4f0
0x3d504 => memcpy(dst=0x11c80a540,src=0x120008010,size=0x10)
0xd30a8: b 0xd3114
0xd3128: cset false
0xd3130: cset false
0xd313c: cset false
0xd3180: b 0xd3184
0xd31d0: bl 0x3d4f0
0x3d504 => memcpy(dst=0x11c80a090,src=0x120008000,size=0x20)
```

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

