第一周总结

本周主要阅读了几篇攻击处理器的文献。对处理器进行EMFI攻击测试与分析，流程主要为设计测试代码，找到合适的注入位置和注入时间，根据结果进行分析；或者进一步完成特定目标的分析，如绕过鉴权等。

Electromagnetic Fault Injection as a New Forensic Approach for SoCs阅读笔记

文章讲述了一种通过电磁故障注入技术对Linux及衍生系统内部敏感信息司法调查取证的手段。文章所使用的SoC是一个4核心的CortexA53，采用ARMv8-A构架，兼容ARMv7指令。使用的注入设备由树莓派控制，EMFI所使用的脉冲发生器可以提供一个400 V的上升时间为2 ns的脉冲。最终目标则是通过EMFI获取Linux系统下的Root权限。

文章先介绍了一些常用的攻击手段以及EMFI在1GHz 以上的SoC上注入的难点、使用的SoC和EMFI注入设备。

之后描述了EMFI实现的思路。对于故障的注入，首先要弄清楚何时，在哪产生干扰。

为了确定EMFI时机，论文给出了一个测试代码。

```
mov x28, #0170
sub x19, x28, #0x1
sub x20, x19, #0x1
sub x21, x20, #0x1
...
sub x28, x27, #0x1
```

首先将0x170赋值给x28寄存器，减1赋给x19，减1赋给x20，一直到x28，重复运行32次。执行后x19到x28对应的值为0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30。同时还需要通过SEMA来分析SoC何时在执行NOP何时在处理数据（执行测试代码）。

经过验证，只有一颗核心对电磁注入敏感，因此需要在特定的位置注入。

EMFI的结果包括无效注入、注入成功和freeze、reboot，论文中推测freeze是由UARTS导致的、reboot则是PC计数器异常导致的。在注入成功和无效注入的结果里

REF.	Occurrences	Result (x19, ..., x28)	Proportion
A	27287	39,38,37,36,35,34,33,32,31,30	71.0%
B	5314	连接丢失	13.8%
C	4899	43,42,41,40,3F,3E,3D,3C,3B,3A	12.7%
D	48	39,38,37,36,35,3E,3D,3C,3B,3A	0.1%
E	28	39,42,41,40,3F,3E,3D,3C,3B,3A	0.1%

A为无效注入，B为连接丢失，主要出现在注入时机过早的情况（相较于C的时机），C推测为跳过了某一条sub指令。DE主要出现在注入时机过晚的情况（相对于C），同样是指令跳过，只是发生在最后10条。

由此可以分析出，可以通过EMFI实现指令跳过从而提权。

最后一部分是对Linux鉴权机制的分析。Linux在校验密码时，先将用户输入的密码Hash，而后strcmp，因此可以从strcmp入手。

EM Fault Injection on ARM and RISC-V

本篇文章主要介绍了在ARM和RISC-V两种不同架构上的EMFI实现。

ARM

```
_MAIN:
  Assert trigger
  NOPs
  LDR R3, =(0xFEDCBA98);
  LDR R4, =(0xFEDCBA98);
  CMP R3, R4;
  BNE _FAULT;
  Deassert trigger
  B MAIN
```

基于上述代码进行了三次实验：第一次实验中，BNE 和后续指令LDR R0的OpCode被赋给了R4。R4本应取到PC+0x30处的数据，但是取到了PC+0处的数据，也就是说EMFI导致错误的数据被读入寄存器中。

第二、三次实验则是为了排除EMFI修改了LDR R4的可能性。第二次在LDR R4后加入了多行NOP，第三次实验则是将第二次实验中LDRR4后的第一个NOP改成了MOVS。论文结果表明，是PC没有正确偏移，PC+0xAMN变成了PC+0xA1N。个人认为，这种说法并不妥当，因为当时PC所处的Address恰好是0xA1X，因此跳转到0xA1N有可能是因为此时PC的Address造成的，而且文章并没有证明是否与X值有关。应该说是在PC=0xAX8时偏移计算时高位被忽略了更为准确妥当，同时猜测这种高位被忽略的现象与PC值无关。应该在LDRR4前加入NOP或其他指令来改变LDR R4的Address来继续研究。

除此之外，联系其他文章，不难发现，EMFI的位置、电压、芯片的封装、布局、时钟周期、Jitter均需要考虑在内。

RISC-V

在RISCV上进行了4个实验：1) 确定芯片上可注入故障的最佳位置；2) 确定可注入故障的时间；3) 分析一次可故障多少条指令；4) 观察改变时钟频率和供电电压的影响

对于实验1，测试代码对某一寄存器连续多次相减，对芯片不同位置进行注入，通过最终相减结果来判断是否发生故障。

对于实验2，通过改变加法指令在很多行NOP中的位置，来判断何时进行注入。

对于实验3，测试代码将加法结果分别赋值给多个寄存器，最终通过寄存器中的数据来判断哪些指令被跳过了。

对于实验4，则是基于电荷模型进行了注入分析。在相同的注入时机，位置，低供电电压，高频率，更容易引起故障。

In-depth Analysis of the Effects of Electromagnetic Fault Injection Attack on a 32-bit MCU

本文中使用的MCU是stm32f103zet6 enhanced MCU，LQFP144 封装，包含一颗32bit的Cortex-M.

（进展报告字数限制，这里删掉了论文中所使用的代码）

实验中，MCU被分为60*60的小格子，探针按照0.33cm的步长依次进行注入，最终得到注入结果和对应的位置。

论文中的结果可以很明显的看出故障类型和注入位置有很大关系。

而注入结果大致可分为3类：错误字符输出、程序重启以及计算错误。

错误字符输出比较容易理解，可以认为是printf导致的，0-p 00110000-01110000，6-v 00110110-01110110，5-u 00110101-01110101，space-$ 00100000-00100100

程序重启是由于EM干扰了Vdd，出现了毛刺导致芯片复位。

计算错误可以细分为65024 65026 >65026 <65024四种情况。

65024：指令跳过或+1时的立即数由1翻到了0，或j由+1变成了+2。由于指令跳过是通过影响PC计数器来实现的，且PC寄存器为奇数时会出现异常，但实验中又没有出现异常，因此排除是PC错误。由1变2翻转了两位，而由1变0只翻转了1位，更容易实现。因此推断是adds a,a,#1变成了adds,a,a,#0

65026:同理，adds j,j.#1变成了adds j,j,#0

<65024(<64770)个人认为介于二者之间，是多次指令跳过导致的，也有可能是a+=0，或内循环BLT跳转指令被跳过导致的。

>65026 MOV指令没有被正确执行，立即数不是0，导致a初值不是0。