Rowhammer Tutorial

Varnavas Papaioannou

September 24, 2017

Before starting, we should warn that experimenting with rowhammer may cause some issues with both the software and hardware of the testing environment. In our case, those issues manifested after 2 months of experimentation. To avoid possible software issues, we would suggest to either create a new partition with Ubuntu or just run Ubuntu directly from USB. In this tutorial, we assume that the latest Ubuntu distribution is used.

Now, for experimenting with the ”bug”, the most complete tool currently available is hammertime[1]. Even though it requires evelated privileges for its operation, it is the only tool capable of performing the attack under different microarchitectures and DRAM configurations. As such, this tutorial we will be primarily focused on that tool and at the end we will provide a brief description of the tools developed to run in user-space.

Experimenting with hammertime: We will begin by setting up the environment. First thing that has to be done is to install the dependencies for the program, in the latest Ubuntu distribution we just have to install `libpfm4`.

```
sudo apt-get install libpfm4-dev
```

Next we run:

```
make
```

Inside the `hammertime`'s root directory to build the tool.

After building the tool, we will have to generate the configuration file that describes the information about our system (such as processor's microarchitecture, number of DIMMS etc). To generate this configuration file, we run:

```
sudo ramses/tools/msys_detect.py
```

The first screen of the tool will ask about the type of the memory controller. To get that information we will have to find the microarchitecture of the system's processor. One way to achieve that is to first run:

```
cat /proc/cpuinfo | fgrep -i model
```

With this command we get the processor's model. Searching for that model on google will reveal its microarchitecture and accordingly we select the appropriate option in the `msys_detect`'s menu.

[1]https://github.com/vusec/hammertime/

The first option is the most important to get right. Wrongly identifying the type of the memory controller will most probably yield zero number of bit flips. Now the rest of options could be considered optional, but getting them right will result in an increased number of bit flips. The fastest way we found for setting them right is by heuristics.

After filling couple of options, the script will output the identified `DRAM` configuration of our system. Here we have to verify that everything that is listed is accurate. The total number of channels, DIMMs and ranks has to be correct because otherwise the test most likely will fail. After setting the proper memory configurations, the program will output all those information into a file, which by default is `mem.msys`

After getting done with generating the configuration file, we execute from the root directory of the tool:

```
sudo tools/profile/profile -s 256m mem.msys | py/prettyprofile.py -
```

With that command, we instruct the profiler to allocate `256 MB` of memory and use the `mem.msys` that we generated previously for the system's configuration to test for the rowhammer vulnerability. The test results of the profiler will be redirected to `prettyprofile` to format the output of the profiler into a more readable form. The profiler has a lot more options which are described in program's repository[2,3].

Now we will provide some information regarding the user space rowhammer testing tools[4]. The first tool is called `hprh` and is based on the Transparent Huge Pages feature for its operation. The current version is build for Sandy Bridge processors and can be parametrized to run for a wide range of memory configurations.

In order to run the tool, we first have to execute:

```
make
```

in its root directory to build the tool. Then we run it by:

```
./hprh -s 256
```

In this case we run the test on a `256 MB` buffer and using the default memory configuration which is a single channel, single dimm, dual rank system. The memory configuration can be adjusted by using the corresponding program arguments which are described in the tool's repository[5].

The rest of the tools (`tcrl`, `thrh`) are build and run in the exact same way with more specific information about their arguments described in their corresponding READMEs. Their operation is still based on the contiguous memory allocation, but this time we depend on the Linux memory allocator (buddy allocator) for that part.

Useful commands:

```
#If outputs [never], it means that THP is disabled
cat /sys/kernel/mm/transparent_hugepage/enabled
```

[2] `https://github.com/vusec/hammertime`

[3] `https://github.com/vusec/hammertime/tree/master/tools/profile`

[4] `https://github.com/vp777/Rowhammer`

[5] `https://github.com/vp777/Rowhammer/tree/master/1.THP%20Rowhammer`

```
#Temporarily (until reboot) enable/disable THP
su
echo always > /sys/kernel/mm/transparent_hugepage/enabled #enable
echo never > /sys/kernel/mm/transparent_hugepage/enabled #disable
```

For example, by disabling `THP`, `hprh`'s performance should drop significantly. In the contrary, for `tcrh` and `thrh` their performance should be more or less the same. To understand why `tcrh` and `thrh` could work even with THP disabled one should experiment with the tools provided in project zero repository[6]. In the case that the Linux allocates large contiguous chunks of physical memory then we know that both `tcrh` and `thrh` have the potential to operate smoothly even with THP disabled.

[6]`https://github.com/google/rowhammer-test/tree/master/physmem_alloc_analysis`