EXERT Team Project Retrospective

Written by Aidan Frantz, Aria Seiler, Stephanie Morehart, and Evan Bestic

Project Goals

The rough idea of Exercising Runtime Triggers for Guest OS Analysis (EXERT) was to create a program that could gather OSI information dynamically from a running Linux kernel image.

These are the goals that we finalized with the client:

- The project should determine all OSI values except for kernel version and architecture.
- The project should generate its OSI information independent of symbol tables such as kallsyms.
- The project should be able to support the following four architectures for analyzing Linux kernels: Arm, AArch64, i386, and x86_64.
- The project should boot a live ramdisk with programs that it can run to determine OSI information.
- The project should utilize callbacks from PANDA.re to gather information from the running kernel when running usermode programs.
- The program should be extensible enough to support additional architectures and Linux kernel versions in the future.
- The program should have a reasonable runtime.

Project Achievements

- Developed a custom C parser in Python in order to generate rules for different systems regardless of how much we know about a given system.
  - The parser functions as follows:
    - First, the parser tokenizes C source code files.
    - It applies the preprocessor to expand macros and preprocessor directives present in the file, even if they are ambiguous.
    - Finally, the parser runs over the C source code and attempts to generate types and value information.
  - The parser was created to help with the creation of different rules systems, but it rapidly grew in complexity since we were unable to make assumptions based on the kernel's runtime environment.
- Created a tool which generates a very limited subset of OSI information. The current tool is limited, but easily extensible. It functions as follows:
  - First, it runs PyPanda with callbacks to generate individual pieces of OSI info.
  - It then uses Python pickle to write the binary results to a file which the main process can read.
  - When it finishes running, it writes its findings and some defaults to an OSI file.
- Create a sophisticated rules system to dynamically test values in memory.
  - The system works as follows:
    - Given sets of potential addresses, it applies many structure, union, and primitive rules to the in-memory data to check for validity.
    - Upon completion, the rules system returns sets of addresses which could contain a data structure.
  - Currently, the rules system is only able to support kernel version 4.4.100, since it takes a long time to write rules by hand.
- Designed a custom PyPlugin interface that made running callbacks easier
  - It works with Arm, AArch64, i386, and x86_64 kernels.
  - The interface works as follows:
    - First, the interface accepts a callback function taking a Panda object and a CPUState object as arguments.
    - This can trigger that callback upon the execution of system blocks or during hypervisor calls.
- Built a custom ramdisk generator which includes arbitrary files and BusyBox versions for many different architectures, allowing us to generate functional Linux ramdisks for any architecture.

The whole system has a long runtime, though we expect the runtime to increase upon completion. Currently, it takes around 30 minutes for the parser and five minutes for the OSI generator when starting from a clean setup. However, the average user can be feasibly expected to wait during an initial run. The system also caches its results, resulting in quicker subsequent runs.

Things We Could Have Done Better (What would we do differently if we started again?)

Enough Design Up Front (EDUF) was applicable to this project in the beginning. However, for a project of this magnitude, there should have been more time devoted to developing a design before making an implementation. For example, learning about the OSI values and their usage would have saved time when designing approaches to find each value. For our current overall design, we became hesitant to try a new implementation since there was already so much work that we had to contribute, and not enough time to feasibly try a new implementation. We could have been able to make our design more robust by designing a strong first implementation instead of one that was good enough for the goals we had at the time. Designing the program up front would have also allowed us to find major design flaws and other issues that would have been ignored or worsened under an EDUF approach.

Using a compiled language would have greatly improved our program’s runtime. Python is slower than languages like Rust or C++, but this language was initially chosen due to its ease of use. If we were designing this project again, we would probably use C++ for its speed, established compatibility with PANDA.re, and easy interoperability with most tools.

Extracting OSI information at runtime is a deceptively complicated problem. If we were to redo this project, we would have looked into disassembly and using code interception to supplement raw memory analysis. While this would have required more research, it would also allow us to extract more information that would have been unattainable otherwise.

Finally, we would spend more time learning how PANDA.re works internally. Our method of treating the system as a black box ended up creating more confusion than what was necessary. We had hoped that treating it as a black box would help keep us from becoming overwhelmed and keep our project scope from getting unmanageable. However, being familiar with its internal workings is required to fully understand its capabilities, and to understand how we can utilize it as a tool. PANDA.re has many live analysis features which have the potential to make our project easier, and if we were able to spend more time learning and understanding these features, we could have had a clearer idea of if our solution was optimal.

Conclusion

Despite its difficulty, we had a lot of fun on this project and it was very interesting. In the future, aggressive optimization will be necessary to generate OSI information in a reasonable amount of time given our constraints, and the problem will likely need to be attacked from multiple angles at once in order to obtain all of the information needed for PANDA.re OSI files. As Dr. Estrada's example here provided inspiration to us, hopefully our project here can provide inspiration to other people that attempt to tackle this problem in the future.