Paper title: kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels
Paper authors: Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, Thorsten Holz
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The paper addresses the problem of memory safety vulnerabilities in kernel components of an operating system (OS). While fuzzing is a promising technique to identify software faults, most fuzzing efforts are limited to user space components of an OS. Bugs in kernel components are more severe as they allow an attacker to gain access to a system with full privileges, but kernel components are difficult to fuzz as feedback mechanisms cannot be easily applied. Additionally, non-determinism and kernel crashes pose problems for the fuzzer's performance.

Black-box fuzzers, such as **Radamsa** and **zzuf**, have no interaction beyond executing the program on newly generated inputs. White-box fuzzers, such as **SAGE**, **DART**, and **KLEE**, use program analysis techniques to find interesting inputs. Gray-box fuzzers, such as **AFL**, use coverage information to guide their search.

The authors also mention the first publicly available gray-box coverage-guided kernel fuzzer, **syzkaller**, and a modified version of AFL called **TriforceAFL**, which utilizes QEMU to measure fuzzing progress. Other specific previous works mentioned include learning input grammar from program traces, optimizing input selection and mutation rates, and using taint tracing to uncover new paths.

What are the key contributions of the paper?
The paper proposes a new hardware-assisted feedback fuzzing tool called kAFL for OS kernels.One of the primary contributions of the paper is the development of a custom hardware design for collecting feedback information from kernel code paths. The authors designed a set of hardware modules that monitor kernel execution and capture the execution trace, which is then used to guide the generation of new input test cases. The hardware design is optimized for performance and efficiency, and includes a range of features such as parallelization and caching to reduce the processing time required.

Another important contribution of the paper is the development of an efficient method for reducing the feedback data size. The authors developed a custom compression algorithm that is tailored to the specific characteristics of kernel code paths. The compression algorithm reduces the size of the feedback data by a factor of 20, which enables faster processing and reduces the overall hardware resources required.The paper also introduces a coverage-guided mutation strategy for generating input test cases. This approach involves repeatedly mutating the input test cases until the desired coverage criteria are met. The authors developed a set of novel mutation operators that are specific to kernel code paths, including register and memory operand mutations, to help maximize code coverage. The coverage-guided mutation strategy is highly effective at identifying previously unknown bugs and vulnerabilities in kernel code.

In addition to these contributions, the authors made several other design choices to optimize the performance and effectiveness of the tool. For example, they use a specialized hypervisor to provide a controlled testing environment and minimize the impact of the fuzzing process on the underlying system. They also implemented a range of performance optimizations, such as parallelization and caching, to help improve the efficiency of the fuzzing process.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The authors conducted extensive experiments to evaluate the effectiveness and performance of the kAFL tool in finding bugs and vulnerabilities in several popular OS kernels, including Linux, FreeBSD, and Windows. To evaluate the effectiveness of the tool, the authors compared the code coverage achieved by kAFL with other state-of-the-art fuzzing tools. The results show that kAFL achieves significantly higher code coverage than other tools, indicating that it is more effective at identifying previously unknown bugs and vulnerabilities. To evaluate the performance of the tool, the authors measured the processing time required to fuzz each kernel. The results show that kAFL is significantly faster than other fuzzing tools, despite the fact that it uses custom hardware to collect feedback information. The authors also conducted a thorough analysis of the bugs and vulnerabilities found by kAFL, which provides further support for the tool's effectiveness. The analysis shows that kAFL was able to identify a large number of previously unknown bugs and vulnerabilities, many of which were critical or high severity.

Write down one question you may want to ask the authors.
What are some potential limitations or challenges of deploying kAFL in a real-world setting, and how can these be addressed?