**Project Proposal** **Abhishek Srivastava**
CS260-001 Computer Security Seminar Student Id: 861307778
Prof. Heng Yin February 23, 2017

---

## Introduction

With the ever increasing use of digital devices and softwares, security risks among them also is rising. Manually analyzing bugs in not a scalable method for finding vulnerability and fixing them. Thats why currently there is a very large research focus on developing tools which are automated and are capable of finding and patching the vulnerability without human assistance such as providing initial seeding for fuzzing. Even with the human assistance existing methods are only able to find shallow bugs and struggle to find bugs in the deeper paths.

Driller is a hybrid vulnerability excavation mechanism to find the bugs deeper in the execution paths. It is a hybrid method because it mixes two vulnerability mechanism *Fuzzing* and *Selective Concolic Analysis* in complementary way. By using both mechanism together it overcomes the shortcomings of both methods by avoiding **path explosion** in concolic analysis and **incompleteness** or **finding shallow bugs** using fuzzing when they are applied on the problem on their own.

## Background

Automated vulnerability analysis systems have been designed actively by researchers . Existing approaches mainly falls into three main categories: **static**, **dynamic**, and **concolic analysis systems**. Each approaches have their advantages and disadvantages.

Static analysis systems can be used to show with certainty, that a given binary code is secure. But, it suffers with two fundamental drawbacks: a large amount of false positives results because it it imprecise, and they lack the capability to provide specific input which can trigger a detected vulnerability.

Dynamic analysis systems, such as "fuzzers", are used to identify flaws by monitoring the native execution of an application. It provides an actionable inputs to trigger the flaws when detected. However, "test cases" are needed to input in these systems for its execution. Generating test cases requires manual effort which limits usability of such systems.

Concolic execution engines utilize program interpretation and constraint solving techniques to generate inputs which helps to explore the state space of the binary to trigger deeper vulnerabilities. However, it needs to trigger a large number of paths in the binary which can cause "path explosion", limiting its scalability.

Driller is composed of following components:

- **Input test cases** : Driller can operate without input test cases. However, the presence of test cases can speed up the initial fuzzing step by pre-guiding the fuzzer toward certain paths.
- **Fuzzing** : When Driller is invoked, it begins by launching its fuzzing engine. The fuzzing engine explores the firs compartment of the application until it reaches the first complex

check on specific input. At this point, the fuzzing engine gets "stuck" and is unable to identify inputs to search new paths in the program.

- **Concolic execution** : When the fuzzing engine gets stuck, Driller invokes its selective concolic execution component. This component analyzes the application, pre-constraining the user input with the unique inputs discovered by the prior fuzzing step to prevent a path explosion. After tracing the inputs discovered by the fuzzer, the concolic execution component utilizes its constraint-solving engine to identify inputs that would force execution down previously unexplored paths.

- **Repeat** : Once the concolic execution component identifies new inputs, they are passed back to the fuzzing component, which continues mutation on these inputs to fuzz the new compartments. Driller continues to cycle between fuzzing and concolic execution until a crashing input is discovered for the application.

Driller functions by combining the speed of fuzzing with the input reasoning ability of concolic execution. This allows Driller to quickly explore portions of binaries that do not impose complex requirements on user input while also being able to handle, without the scalability issues of pure concolic execution, complex checks on specific input. "Complex" checks as those checks which are too specific to be satisfied by input from an input-mutating fuzzer.

American Fuzzy Lop (AFL) is being used in the Driller with my source code version. AFL relies on instrumentation to make informed decisions on which paths are interesting.

Important AFL features which are used by Driller:

- **Genetic fuzzing**
- **State transition tracking**
- **Loop "bucketization"**
- **Derandomization**

These features allow AFL to rapidly discover unique paths through an application, performing path discovery within a given compartment/code block of the application. Driller overcomes the weakness of fuzzing by determining specific user input required to pass complex checks for fuzzing to proceed forward, by taking advantage of its concolic execution component.

Angr, an open-sourced symbolic execution engine is used for Drillers as a concolic execution engine. This engine is based on Mayhem and S2E. Angr translates binary code into Valgrinds VEX intermediate representation, which is interpreted to determine the effects of program code on a symbolic state. The analysis engine tracks all concrete and symbolic values in memory and registers throughout execution. At any point in the program that the engine reaches, a constraint resolution can be performed to determine a possible input that satisfies the constraints on all symbolic variables in the state. Such an input, when passed to a normal execution of the application, would drive the application to that point. The advantage of concolic execution is that it can explore and find inputs for any path that the constraint solver can satisfy. This makes it useful for identifying solutions to complex comparisons that a fuzzer would be unlikely to ever brute force.

## Approach

I will evaluated Driller on multiple binaries which will vary in terms of different vulnerabilities. Such as Stack-based Buffer Overflow, Improper Access of Indexable Resource and many other. The main aim is to crash the binaries using different tools which implies that the binaries are vulnerable.

The experiments will be done using following tools:

- AFL
- Angr
- Driller

I will present the number of vulnerabilities that were discovered by these three experiments and possibly the reason as well.

## Challenges

The main challenges involved in this project will be:

- Collecting multiple variation of vulnerable binaries for evaluation.
- Setting up correct environment where these tolls do not interfere with the evaluation.
- Showing that driller considerably expands the code coverage.
- Showing Increased number of discovered vulnerabilities because of increased coverage because it highly depends on the type of vulnerability.

## Expected Results

Following Results are expected:

- Poor performance of Symbolic execution with respect to fuzzer and Driller.
- Most crashes discovered in the Driller experiment were also found with the baseline fuzzer.
- Driller finds the more number of crashes with compared to fuzzer and symbolic execution mechanism .

These results will demonstrate that enhancing a fuzzer with selective concolic execution improves its performance in finding crashes. Driller is able to crash more applications than the union of those found by fuzzing and by symbolic execution separately.

## References

[1] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna. 2016. *Driller: Augmenting Fuzzing Through Selective Symbolic Execution* Department of Computer Science, UC Santa Barbara.