**Upgrading AutoFuzzer for Hypertext Transfer Protocol Secure (HTTPS) Protocol Vulnerability Detection**

Shuoyuan Chen 20683201 [s487chen@uwaterloo.ca](mailto:s487chen@uwaterloo.ca)

Jiahao Shi

Xinyi Ma

**Abstract**

Fuzzing is a widely used method to stress-test servers to expose security vulnerabilities before serious damage happens. In this project, we aim to upgrade the prototype automatic fuzzing tool named AutoFuzz and expand its usage over the Hypertext Transfer Protocol Secure (HTTPs). The results against various benchmark suites will be compared with current-day fuzzing tools. The fuzzing algorithm in the AutoFuzz framework will be modified and the abstraction function will be optimized to suit HTTPs protocols. The AutoFuzz combines Finite State Automation (FSA) and Generic Message Sequence (GMS) to automatically extract network protocol traces for testing, which improves coverage with reduced human interference. The ultimate goal of this project is to develop a fast and easy-to-use fuzzing tool for beginners and small website holders, who seeks to detect major protocol flaws with limited budget.

**Introduction**

HTTP is widely utilized to transfer information on the internet, but it is vulnerable against eavesdropping since the communication is not encrypted. Nowadays, a more secure HTTPS protocol has become mainstream, but the security challenges around the protocol remain and weak spots in protocol could still lead to serious consequences, such as the server crash and information leakage. Fuzzing test can generate random inputs, which are often unexpected by the developers, and thus expose vulnerabilities. However, to generate meaningful input and penetrate deep into the codes, a fuzzer needs to ‘know’ the grammar of the protocol, which requires a large amount of programming. Therefore, an automated fuzzer with the ability to learn the specifications of HTTPs on its own is desired. We propose to modify such an automated fuzzer, called AutoFuzz, to bring automated fuzzing to HTTPS, easing the labor of developers and introducing a new point of view to protocol fuzzing. Finally, we want to develop a handy HTTPS fuzzer for people with low budget and low programming skills, protecting their web applications from errors and attacks.

**Approach and algorithms**

In this project, a model-based fuzzing approach, which needs modification and customization for each new protocol to apply to, is selected. Specifically, we will build new FSA based on the traces recorded in the client - proxy - server transition process. Meanwhile, GMS will be associated with the FSA state. Finally, the fuzzing algorithm which is improved to apply to HTTP protocol is started. The fuzzing algorithm checks whether the current state has a transition for the abstract representation of the input and updates the current state in FSA, then sends the input message to the server end.

**Autofuzz**

**Benchmarks**

The performance of the fuzzer can be measured by its coverage of the target website, the number of bugs exposed, execution speed and stability. Our fuzzer is expected to be tested using Google’s Fuzzbench package, which supports all OSS-Fuzz benchmarks and customized benchmark too. It is a powerful tool to test fuzzers as it enables running multiple fuzzers on a variety of benchmarks at the same time. The edge coverage against different benchmarks over time will be plotted. Finally, the overall performance will be compared with other fuzzers on the same benchmarks. By comparing the performance between different benchmarks, a clear image of the fuzzer’s capability can be obtained. Moreover, the comparison with existing popular fuzzers really puts the real-world usability of our fuzzer to a test.

**Plans**  
We plan to first do literature research

Timeline

1. Build a server - autofuzz - client system, running https protocol; (1 week)

a. autofuzz acts as a man-in-the-middle proxy server and records protocol traces

2. Learn protocol specifications through recorded traces (3 weeks)

a. Construct finite state automaton (consisting of key states and transitions of a protocol);

b. Modify abstraction functions to identify messages and distribute them into clusters (similar messages in the same cluster)

c. Apply sequence alignment algorithms to generate generic message sequence (GMS) for each cluster.

3. Modify fuzzing algorithms (2 weeks)

a. Modify deterministic functions

i. Each function is directive, and it is to induce a kind of software vulnerability (such as inserting maximum / minimum integers to cause integer overflow)

b. Modify non-deterministic functions

i. Randomly modify messages or GMS

4. Construct a methodology to analyse the performance of the fuzzer (2 week)

a. For example: Test fuzzer with Fuzzbench Compare with other tools

1. There are many papers providing several methods or fuzzers to detect vulnerabilities of software.

2. It is feasible for us to fuzz client or server protocol implementations based on ongoing learning about Computer Networks.

3. We have evaluated the workload of modifying the AutoFuzz tool and the construction of the server - fuzzer - client framework in detail.

4. The tasks that need to be done are as follows - framework construction, function modification (fuzzing algorithms, deterministic and non-deterministic functions), protocols running and traces recording, and comparison methodology construction. Based on the tasks above, the project is divided into parts that could be pulled off in 10 weeks step by step.