**yFuzz:**

1. Are there any consequences of using an older version of AFL, e.g., breadth of supported input?

All updated files in the AFL 2020 version compared to the 2018 version can be found at the link below. We found that most of the changes are related to bug fixes, android support, extension of metrics for output layout, Minor documentation improvements, specifying the CPU core id to execute the tool etc. <https://github.com/google/AFL/blob/master/docs/ChangeLog>

2. Memory cost of ASAN:

It is possible that ASAN will consume lots of memory space. In the worst case, it may consume 20 Tb memory space, depending on different machine configurations and programs. Please refer to GitHub discussion:<https://github.com/google/sanitizers/issues/731>.

However, during the test of yFuzz on OpenSSL, we didn’t observe much differences in memory usage as ASAN was switched on/off. Dynamic memory usage stays around about 0.4% monitored by using *top* command in another terminal, and it does not vary much.

3. Whether ASAN is required for yFuzz?

ASAN aims to detect memory-based bugs. Without ASAN, some bugs cannot be detected by AFL and yFuzz, such as a classic off-by-one-error. For more details, please refer to: <https://fuzzing-project.org/tutorial2.html>

P.S. To turn ASAN off, delete the configuration “AFL\_USE\_ASAN = 1” in the ***build*** file, which is located in the program directory (in VM it is ./yfuzz/testcase/openssl101).

4. Please provide feedback as desired on any of the issues/TODOs in the README.

We have spoken with Yurong Chen, the student who developed yFuzz and now works at FB Research. He mentioned the readme is a deprecated note during the development of yFuzz and can be safely ignored at this point. All issues noted there have already been resolved in the current yFuzz version shared at this point.

**CustomPro:**

1. How old is TEMU? 2009?

Temu was last tested in 2009. Any questions regarding the BitBlaze project can be sent to [bitblaze@gmail.com](http://bitblaze.cs.berkeley.edu/temu.html). More info on the source code updates can be found at <http://bitblaze.cs.berkeley.edu/temu.html>.

1. [Low, requires test] What is the impact of TEMU's age on support for newer versions of input software?

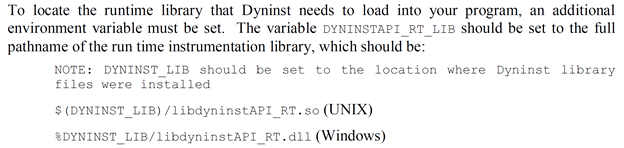
While we have not not used DECAF in our project so far, we performed a preliminary study to compare TEMU’s and DECAF’s capabilities.

Capabilities of TEMU: **TEMU** is able to perform whole-system dynamic taint analysis. Marking certain information sources (e.g., keystrokes, network inputs, reads for certain memory locations, and function call outputs) as tainted, **TEMU** keeps track of the tainted information propagating in the system.

Capabilities of DECAF: **DECAF** ensures precise tainting by maintaining bit-level precision for CPU registers and memory, and inlining precise tainting rules in the translated code blocks. Thus, the taint status for each CPU register and memory location is processed and updated synchronously during the code execution of the virtual machine.

1. [low] Why do generated binaries have a dependency on libdyninstaAPI\_RT.so?

We were able to find only limited information about Dyninst’s design, since it is not an open-source project. Based on the information provided in the Dyninst manual, Page 63, it appears that this library must be added to the environment before using the Dyninst API: <https://github.com/dyninst/dyninst/blob/master/dyninstAPI/doc/dyninstAPI.pdf>.



**Interactions between PJR Corps and Research group for yFuzz & CustomPro:**

**By Froilan:**

1. The ssss ppt slides mention (slide 20) a "taint\_analyze.py" python script should be used to extract tainted instructions from the TEMU generated trace file; however, I haven't been able to locate the python script within the VM provided. Where might I find a copy?

(From what I've gathered extracting the tainted instructions is a necessary step to move forward in the CustomPro's process)

The work after extracting the tainted instructions is present in the following link: <https://docs.google.com/presentation/d/1TYZKaan2WzhqH4ypkvfMwDpUsI3__x0V/edit?usp=sharing&ouid=111313426431583858222&rtpof=true&sd=true>

2. Once the tainted instruction set is created, what are the next steps?

From what I've understood so far (hopefully accurately), the next steps would be symbolic execution and binary rewriting using Angr and DynInst tools respectively?

It also seems that these 'next steps' may not fall within the scope of CustomPro and a user would have to "move" the given trace file through those tools to generate a binary?

Yes, the next steps would be symbolic execution and binary rewriting using Angr and DynInst tools respectively. I think the process works like, we need to remove a feature for ex: Encrypt then MAC, let's say, our tool removes the feature from both client and server sides to produce the customized binary. This is explained clearly, in the custom pro paper page 13, section 7 (evaluation).