Automated Rapid Verification and Validation of Software

Mentor: Steven Drager, AFRL Team: Hunter Beach, Samantha Brewer, Katie Brickner

Problem

Legacy codebases often contain undocumented languages and outdated security practices. Current methodology to maintain and harden these systems involves training and familiarizing employees with the system, as well as manual vulnerability detection.

Solution

A new pipeline strategy to automate vulnerability detection, system hardening, and behavioral model generation. The team presents a proof-of-concept built with C/C++, LLVM intermediate representation (IR), the chatbot ChatGPT, and the Mermaid modeling language.

Impact and Future Applications

LLVM allows for conversion of any supported language into IR bitcode or, via LLVM's LLC compiler, rudimentary C/C++ code. Further pipeline enhancements may include a bidirectional workflow and support for additional languages.

Step 1:

C/C++ Vulnerability Detection & Hardening

Scan project for vulnerabilities that result in buffer overflow. Bash script to compile with Address Sanitizer, harden then verify with Valgrind.

C/C++ Project
(CMake or Make-file)

Step 2:

LLVM IR Conversion

Whole Program LLVM (WLLVM) provides functionality to compile an entire C/C++ project to a single unified human-readable IR file.

LLVM IR
(Human-Readable .ll file)

Step 3:

AI-Assisted Model Generation

ChatGPT processes input IR, generating Mermaid code that can be turned into a high-level model of the code's functionality.

Figure 1. Pipeline components.

Use-Case Example

Figures 2, 3, and 4 show examples of each stage in the pipeline.

```
lab1_brewer.c: In function 'reload':
lab1_brewer.c:264:5: warning: implicit declaration of function
  264 |     gets(currentLine,200,stream);
      |     ^~~~
      |     fgets
```

Rerun post-hardening

```
For lists of detected and suppressed errors, rerun with: -s
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 2. Example of vulnerability detection (insecure 'gets' command), hardening (changed to secure 'fgets'), and then verification of hardening.

LLVM is static single-assignment, resulting in exponentially larger IR files compared to the original C/C++.

```
fgets(currentLine,
200,stream);
```

For our example use case, 572 lines of C code became 1704 lines of LLVM IR.

```
%15 = call i8* @fgets(i8*
noundef %13, i32 noundef
200, %struct._IO_FILE*
noundef %14)
```

Figure 3. Example of C code and resulting LLVM IR.

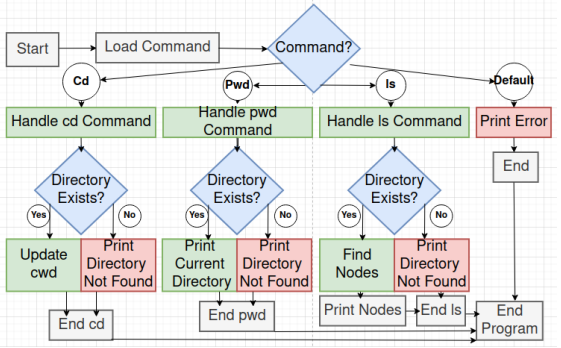

Figure 4. Example of generated model.