IoT Security Verification

Team Information - SDMay21-41

Client
Megan Ryan

Faculty Advisor
Dr. Kristin Rozier

Team Members

Joshua French	jkfrench@iastate.edu	CPR E
Vincent Johnson	vincentj@iastate.edu	S E
Jordan McKillip	jmck1998@iastate.edu	S E
Marcus Reecy	mjreecy@iastate.edu	CYB E

Problem Statement / Solution

Problem

The Internet of Things (IoT) is becoming more and more a part of people's everyday lives. Devices such as locks, cameras, and smart-speakers are just a very small view of all the ways our lives are going online. With all of these devices having important roles, being located in private places, and gathering loads of information, the security of them is much more prevalent as it would be problematic if it got into the wrong hands. There are already some ways that the security of the code behind these IoT devices is being tested. However, there are a lot of security properties that aren't being thoroughly checked. Our project will create a set of IoT benchmarks and utilize the top three tools and and security properties to verify said benchmarks. This will allow future competitions to validate IoT code.

Proposed Solution

SV-COMP is a security validation competition that aims to make model checking tools competitive with each other over many different software security properties. Benchexec, which is used in SV-COMP, helps to compare different software verification tools. These tools test a variety of common security failings to ensure that the software can be validated and secure for any developing needs regarding the compatible c software. Our project is to use and expand the SV-COMP technologies to focus on IoT device code and test different IoT libraries.

The final goal of this project is to create a set of IoT benchmarks using existing IoT libraries that we deem secure. SV-COMP may then use this secure set to aid in validating tools against IoT code. Additionally, future users may follow our documentation to expand / validate their own IoT libraries.

Design Requirements / Standards

Functional Requirements

- Written and tested within the C programming language
- Model-checking is done on 4 cores, 8 gb RAM, and 15 min. run time
- Should not require any special software on the competition machines

Non-Functional Requirements

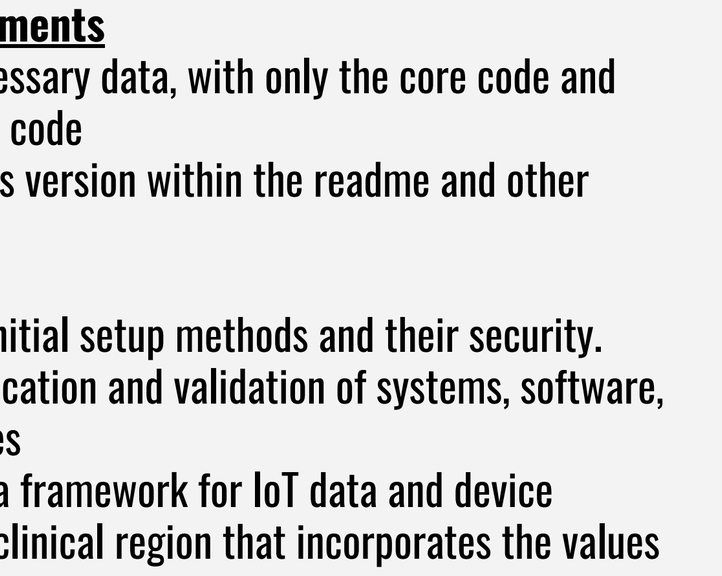
- Remains free of unnecessary data, with only the core code and descriptions within the code
- The software reports its version within the readme and other documentation

Relevant Standards

- IEEE 802.11ai-2016 - initial setup methods and their security.
- IEEE 1012-2016 - verification and validation of systems, software, and hardware life cycles
- IEEE P2933 - creating a framework for IoT data and device interoperability in the clinical region that incorporates the values of TIPPSS

Intended Users/Uses

The intended users of this software project will be developers and academia of IoT. Users will be able to create their own benchmarks following the detailed analysis that has been implemented within the project repository.



Design Approach

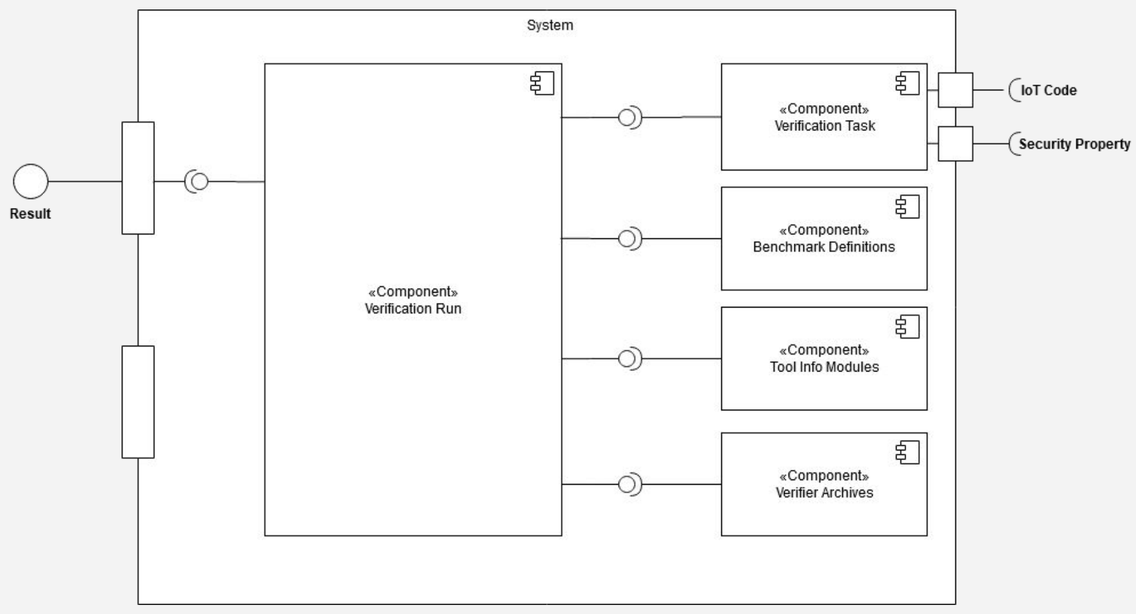


Figure 1: Concept Sketch

	T/F = Correct True/False				IT/IF = Incorrect True/False			U = Unknown
	CPA-SEQ				PredatorHP			
IoT BENCHMARKS	No Overflows	Mem Safety	Reach Safety	Mem Cleanup	No Overflows	Mem Safety	Reach Safety	Mem Cleanup
io_dig_patched.c	*T*	*T*	*F*	*T*	*T*	*T*	*U*	*T*
io_dig_vulnerable.c	T	T	F	T	F	F	U	T
clear_tempature_data_patched.c	T	T	T	T	T	T	U	T
clear_tempature_data_vulnerable.c	T	F	T	T	U	F	U	U
temptoa_patched.c	T	T	T	T	T	T	U	T
temptoa_vulnerable.c	T	T	T	T	T	T	U	T
count_lw_bus_error_patched.c	T	T	T	T	U	U	U	U
count_lw_bus_error_vulnerable.c	T	F	T	U	U	F	U	U
read_temperature_data_patched.c	U	F	T	U	T	T	U	T
read_temperature_data_vulnerable.c	U	F	T	U	U	U	U	U
select_lw_bus_patched.c	T	T	T	T	T	T	U	T
select_lw_bus_vulnerable.c	T	F	T	T	U	F	U	U
search_sensors_patched.c	U	T	T	T	U	U	U	U
search_sensors_vulnerable.c	U	T	T	T	U	U	U	U
fnet_ip6_ext_header_handler_result_t _fnet_ip6_ext_header_handler_options_patched	T	IF	U	U	U	IF	U	U
fnet_ip6_ext_header_handler_result_t _fnet_ip6_ext_header_handler_options_vulnerab	T	F	U	U	U	F	U	U

Figure 2: Results Spreadsheet

```
extern uint8_t __VERIFIER_nondet_uint8_t(void);
extern uint16_t __VERIFIER_nondet_uint16_t(void);
extern int16_t __VERIFIER_nondet_int16_t(void);
extern uint32_t __VERIFIER_nondet_uint32_t(void);
extern int32_t __VERIFIER_nondet_int32_t(void);
extern float __VERIFIER_nondet_float(void);

uint8_t mqtt_num_rem_len_bytes(const uint8_t* buf);

int main() {
        while (1) {
                buf = __VERIFIER_nondet_uint8_t();
                mqtt_num_rem_len_bytes(buf);
        }
}

uint8_t mqtt_num_rem_len_bytes(const uint8_t* buf) {
        uint8_t num_bytes = 1;

        //printf("mqtt_num_rem_len_bytes\n");

        if ((buf[1] & 0x80) == 0x80) {
                num_bytes++;
                if ((buf[2] & 0x80) == 0x80) {
                        num_bytes ++;
                        if ((buf[3] & 0x80) == 0x80) {
                                num_bytes ++;
                        }
                }
        }
        return num_bytes;
}
```

Figure 3: IoT Benchmark

```
format_version: '2.0'

input_files: 'mqtt_num_rem_len_bytes.i'

properties:
 - property_file: ../properties/no-overflow.prp
   expected_verdict: true
 - property_file: ../properties/valid-memsafety.prp
   expected_verdict: true
 - property_file: ../properties/valid-memcleanup.prp
   expected_verdict: true

options:
  language: C
  data_model: ILP32
```

Figure 4: .yml file for benchmark, listing the property and expected outcome

```
CHECK( init(Main.main()), LTL(G !deadlock) )
```

Figure 5: Security Property with LTL formula

Technical Details

SV-COMP: The software verification competition in which our project was derived from. A suite of software model checkers take input benchmarks (Java and C code), an expected result (.yml), and a security property (LTL formula) and output the results via Benchexec.

Benchexec: runs the model checkers - gives the ability to specify resource limits, measure memory usage of the tool, and verify the result with the expected value.

Benchmark: a piece of IoT software code that is refactored to run with the suite of model checkers. Our project focused mainly on C code.

Security Property: an LTL formula that the model checker follows in order to determine if the benchmark meets the property.

Model Checkers:

- **CpaChecker(CPA-SEQ):** The model checker with the best results for the no overflows category.
- **PredatorHP:** The model checker with the best results for the memory safety category.
- **VeriAbs:** The model checker with the best results for the reach safety category.

Testing

The testing environment that will be used is called Benchexec. It is used by SV-COMP to run their benchmarks on tools while measuring their performance. The testing for this project primarily involves model-checking the created IoT-benchmarks that have been created. The results of the testing will tell whether the IoT code that was run through the model-checkers is secure in various IoT-security properties. We recorded these results in our Results Spreadsheet (Figure 2).