Introduction to Software Security
Coursework 01

Introduction

This coursework introduces students to basic approaches to specify and verify security aspects concerning *confidentiality*, *integrity* and *availability* for software systems. In particular, this coursework provides theoretical and practical exercises to *(i)* identify and describe software vulnerabilities concerning memory safety in C programs; *(ii)* describe and evaluate typical examples of SQL injection, which an attacker can exploit; *(iii)* specify and verify race conditions in concurrent software; *(iv)* identity and describe cyber-threats in a simple embedded system, which can be remotely operated; and lastly *(v)* describe how (bounded) model checking techniques work.

Learning Objectives

By the end of this lab, you will be able to:

- Define standard notions of security and use them to evaluate the system's confidentiality, integrity, and availability.
- Explain standard software security problems in real-world applications.
- Use testing and verification techniques to reason about the system's safety and security.

1) **(Critical Software Vulnerabilities)** Considering the following C code, identify and describe the critical software vulnerabilities from the program statements a) to e). You can assume that each instruction is appended to the end of the C program, but they do not remain there when you consider the other statements. Note that you should describe the error and which CIA principles get violated.

```
int *zPtr;
int *aPtr = NULL;
void *sPtr = NULL;
int number, i;
int z[5] = {1, 2, 3, 4, 5};
sPtr = z;
```

a) ++zPtr;
b) number = zPtr;
c) number = *zPtr[2];
d) number = *sPtr;
e) ++z;

2) **(SQL injection)** SQL injection allows an attacker to interfere with the database queries to retrieve data. For example, a programmer can construct a SQL query to check name and password as

```
query = "select * from users where
name='" + name + "'" and pw = '" +
password + "'"
```

Figure 2: SQL Injection Example.

If an attacker provides the name string, the attacker can set the name to "John' –"; this would remove the password check from the query (note that -- starts a comment in SQL). Provide here other examples of SQL injection:

i. Retrieving hidden data;
ii. Subverting application logic;
iii. UNION attacks;
iv. Examining the database;
v. Blind SQL injection.

3) **(Race Condition)** Specify the three properties described below for the following mutual exclusion algorithm using linear-time temporal logic (LTL).

int flag[2], turn, x, i;

void *t1(void *arg) {
flag[0] = 1;
turn = 1;
while (flag[1] == 1 && turn == 1) {};
//critical section
if (i==1) x=1;
//end of critical section
flag[0] = 0;
return NULL;
}
	void *t2(void *arg) {
	flag[1] = 1;
	turn = 0;
	while (flag[0] == 1 && turn == 0) {};
	//critical section
	if (i==2) x=2;
	//end of critical section
	flag[1] = 0;
	return NULL;
	}

i. At most, one process is in the critical region at any time.
ii. Whenever a process tries to enter its critical region, it will eventually succeed.
iii. A process can eventually ask to enter its critical region.

4) **(Cyber-threats)** A typical embedded system consists of a *human-machine interface* (keyboard and display), *processing unit* (real-time computer system), and *instrumentation interface* (sensor, network, and actuator) that can be connected to some physical plant. For example, figure 1 illustrates a simple embedded system employed in a chemical process, which can be operated remotely via TCP/IP protocol. In particular, the overall objective of this simple embedded system is to keep the temperature and pressure of some chemical processes within well-defined limits by a remote operator.

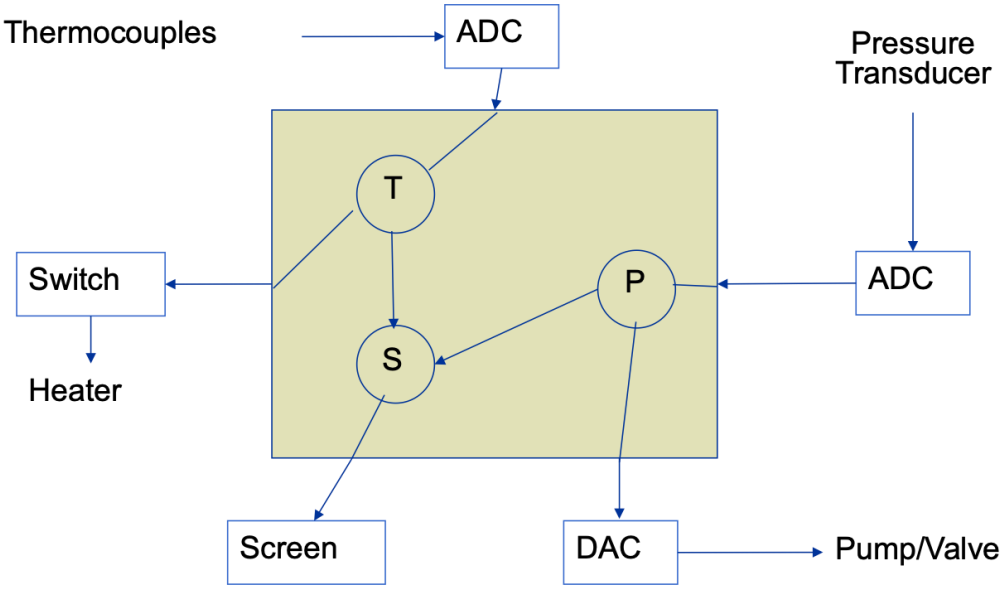

Figure 1: Simple Embedded System [1].

i. Specify the following properties of this simple embedded system using LTL:

a) A **T** process reads the measured values from a temperature sensor, turns the heating system on if the temperature is below 20°C, and turns off the heating system if the temperature is above 300°C.
b) Process **P** regulates pressure with a sensor opening the pump/valve when the pressure value is above 500bar and closing the pump/valve when the pressure value is below 100bar.
c) The measured values will be shown on a liquid crystal display (LCD) when the T and P processes transfer data to an S process.

ii. What are the **security objectives** of this simple embedded system?

iii. What are the sources of **security problems** that can arise when operating this simple embedded system remotely?

5) **(Bounded Model Checking)** Describe the main steps involved in checking programs using the BMC technique, from reading the program to generating the SMT formulas [2]. Consider the following example to describe each step in the technique. The **assume** directive can define constraints over (non-deterministic) variables, and the **assert** directive is used to check the system's correctness w.r.t. a given property.

```
#include <assert.h>
#define a 2
int main(int argc, char **argv) {
  long long int i=1, sn=0;
  unsigned int n=5;
  assume (n>=1);
  while (i<=n) {
    sn = sn + a;
    i++;
```

```
  }
  assert(sn==n*a);
}
```

Marking Scheme

Note that this may be refined to introduce extra cases reflecting special cases if required.

Question 1) Has the student Identified and described the critical software vulnerabilities for all program statements?	
The student was able to identify and describe the vulnerabilities according to the CIA principle.	(2)
The student was able to identify and describe some vulnerabilities according to the CIA principle.	(1.5)
The student was able to identify the vulnerabilities, but he/she cannot describe those vulnerabilities considering the CIA principle.	(1)
The student was able to identify some vulnerabilities, but he/she cannot describe those vulnerabilities considering the CIA principle.	(0.5)
No attempt has been made.	(0)

Question 2) Can the student provide all SQL injection examples as requested in the question?	
The student can provide all examples of SQL injections.	(2)
The student can provide four examples of SQL injections.	(1.5)
The student can provide two or three examples of SQL injections.	(1.0)
The student can provide one example of SQL injections.	(0.5)
No attempt has been made.	(0)

Question 3) Can the student specify the three LTL properties for the following mutual exclusion algorithm?	
The student can specify all three LTL properties.	(2)
The student can specify two LTL properties.	(1.5)
The student can specify one LTL property.	(1.0)
The student made some effort to specify some LTL properties, but all of them were wrong.	(0.5)
No attempt has been made.	(0)

Question 4) Can the student specify the LTL properties, security objective, and problems?	
The student can specify all three LTL properties together with the security objective and problems.	(2)
The student can specify all three LTL properties and some security objectives and problems.	(1.5)
The student can specify some security objectives and problems. The student can also specify some LTL properties.	(1.0)
The student can only specify the security objectives, security problems, or some LTL properties.	(0.5)
No attempt has been made.	(0)

Question 5) Can the student explain the BMC technique using the illustrative example?	
The student can explain all the steps of the BMC technique using the illustrative example.	(2)
The student can briefly explain all the steps of the BMC technique using the illustrative example.	(1.5)
The student can explain only parts of the BMC technique.	(1)
The student tried to explain the BMC technique, but his/her explanation was unclear.	(0.5)
No attempt has been made.	(0)

References

[1] Alan Burns, Andrew J. Wellings: *Real-Time Systems and Programming Languages - Ada, Real-Time Java and C / Real-Time POSIX*, Fourth Edition. International computer science series, Addison-Wesley 2009, ISBN 978-0-321-41745-9, pp. I-XVIII, 1-602.

[2] Lucas C. Cordeiro, Bernd Fischer, João Marques-Silva: *SMT-Based Bounded Model Checking for Embedded ANSI-C Software.* IEEE Trans. Software Eng. 38(4): 957-974 (2012).

[3] Christel Baier, Joost-Pieter Katoen: Principles of model checking. MIT Press 2008, ISBN 978-0-262-02649-9, pp. I-XVII, 1-975.