Coursework 03
Detection of Software Vulnerabilities:
Static Analysis (Part I)

Introduction

This coursework introduces students to the basic concepts of software verification and validation concerning security aspects. In particular, this coursework provides five theoretical and practical exercises: (i) describe the syntax and semantics of the most common SMT background theories and their application to verify security properties in software systems; (ii) check the satisfiability of SMT equations using state-of-the-art solvers based on the theory of bit-vectors; (iii) analyze the main advantages and disadvantages of BMC techniques concerning soundness and completeness in order to verify security properties; (iv) build SMT formulas extracted from C programs to check for buffer overflow and memory leak; lastly, (v) develop a precise memory model for software verification by taking into account memory alignment.

Learning Objectives

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

- Introduce software verification and validation.
- Understand soundness and completeness concerning detection techniques.
- Emphasize the difference between static analysis, testing/simulation, and debugging.
- Explain bounded model checking of software.
- Explain precise memory model for software verification.

1) **(Satisfiability Modulo Theories)** Describe the syntax and semantics of the background theories used by SMT solvers, which are relevant for verifying security properties in software systems, as described by the SMT-LIB.[1]

2) **(Solving SMT equations)** Check the satisfiability of these propositional equations. You must use the theory of bit-vector (QF_BF) implemented in the Z3 SMT solver.[2]

a) $(\neg a \vee \neg b) \wedge (\neg b \vee c) \wedge b$

b) $(((d \wedge c) \vee (p \wedge \neg ((c \wedge \neg (d))))) \equiv ((c \wedge d) \vee (p \wedge c) \vee (p \wedge \neg (d))))$

c) $(((a) \wedge ((b) \rightarrow \neg (a)) \equiv \neg (b))$

3) **(Soundness and Completeness)** What are the main advantages and disadvantages of the BMC technique? How can we ensure the soundness and completeness of the BMC technique?

[1] http://smtlib.cs.uiowa.edu/

[2] https://github.com/Z3Prover/z3

4) **(C/C++ API of Z3)** Write a program using the C/C++ API of Z3 to verify the following C programs to check for buffer overflow and memory leak. Note that you must build two sets of formulas C and P and then check for the satisfiability of the resulting equation C ∧ ¬ P.

a) Buffer overflow.

```
int main() {
  int a[2], i, x, *p;
  p=a;
  if (x==0)
    a[i]=0;
  else
    a[i+1]=1;
  assert(*(p+2)==1);
}
```

b) Memory leak.

```
#include <stdlib.h>
int main() {
  char *p = malloc(5);
  char *q = malloc(5);
  p=q;
  free(p);
  p = malloc(5);
  free(p);
  return 0;
}
```

5) **(Aligned Memory Model)** Derive the equations C and P from the following program C. You must enforce the C alignment rules when writing the resulting formula C ∧ ¬ P.

```
#include <stdint.h>
struct foo {
  uint16_t bar[2];
  uint8_t baz;
};
int main() {
  struct foo qux;
  qux.bar[0] = 10;
  qux.bar[1] = 20;
  qux.baz = 'C';
  struct foo *quux = &qux;
  quux++;
  quux->baz = 'D';
  return 0;
}
```

References:

[1] 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).

[2] Jeremy Morse, Mikhail Ramalho, Lucas C. Cordeiro, Denis A. Nicole, Bernd Fischer: ESBMC 1.22 - (Competition Contribution). TACAS 2014: 405-407.