LASIGE resilient distributed and networked systems

On the Path to Buffer Overflow Detection by Model Checking the Stack of Binary Programs

Luís Ferreirinha and Ibéria Medeiros

Faculty of Sciences of the University of Lisbon

ENASE 2024

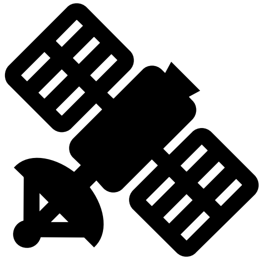
19th International Conference on Evaluation of Novel Approaches to Software Engineering

U LISBOA Ciências ULisboa FCT Fundação para a Ciência e a Tecnologia

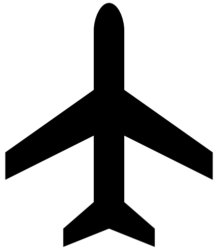
Contents

Introduction

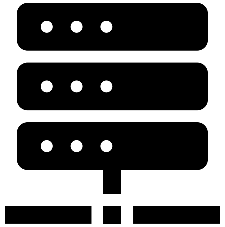
- Why does software need to be secure and reliable?



Mission Critical Systems



Safety Critical Systems



Security Critical Systems

- Software vulnerabilities are one of the main threats to the correct operation of these systems
- Buffer Overflows are classified as one the most dangerous vulnerabilities

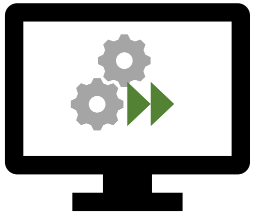
Introduction

LASIGE resilient distributed and networked systems

- Software vulnerabilities can be detected using the following methods

High Scalability
Low Precision

Static Analysis



High Precision
Low Scalability

Dynamic Analysis

- Despite advancements in compilers and operating systems security, vulnerabilities in C binaries persist

- Leading to the need to apply these methods directly in released software (binaries)

Introduction

LASIGE resilient distributed and networked systems

- The C programming language is the most vulnerable to these vulnerabilities due to the lack of safeguards when writing to arrays.

Example.c

```c
void copy(char *str) {
    char buffer_2[16];
    strcpy(buffer_2, str);
}

void main() {
    char buffer_1[256];

    for (int i = 0; i < 255; i++) {
        buffer_1[i] = 'x';
    }
    copy(buffer_1);
}
```

Introduction

LASIGE resilient distributed and networked systems

Example.c

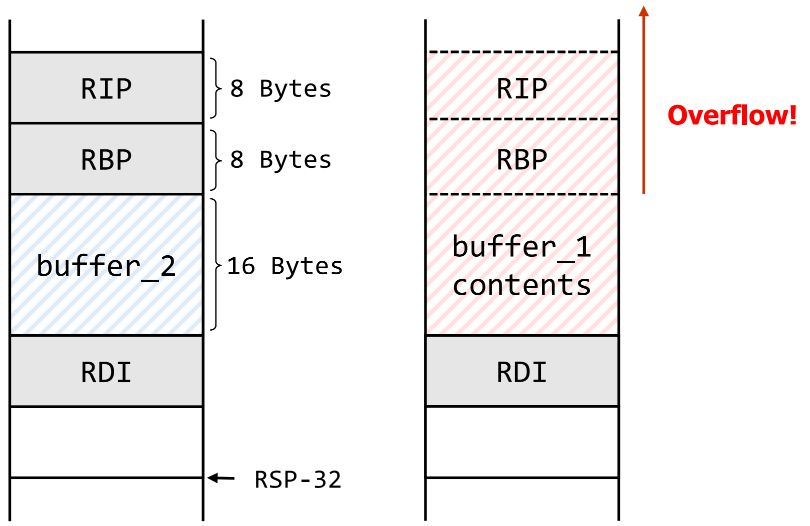
```c
void copy(char *str) {
    char buffer_2[16];
    strcpy(buffer_2, str);
}

void main() {
    char buffer_1[256];

    for (int i = 0; i < 255; i++) {
        buffer_1[i] = 'x';
    }
    copy(buffer_1);
}
```

Copy.asm

```asm
copy:
    push rbp
    mov rbp, rsp
    sub rsp, 32
    mov QWORD PTR [rbp-24], rdi
    mov rdx, QWORD PTR [rbp-24]
    lea rax, [rbp-16]
    mov rsi, rdx
    mov rdi, rax
    call strcpy
    nop
    leave
    ret
```



Before strcpy call

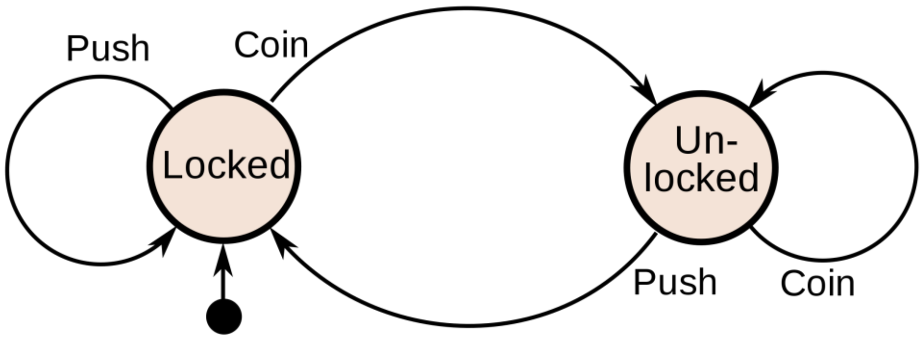
After strcpy call

Introduction

- This work aims to answer the question:

Can we devise a tool to accurately detect buffer overflows at scale?

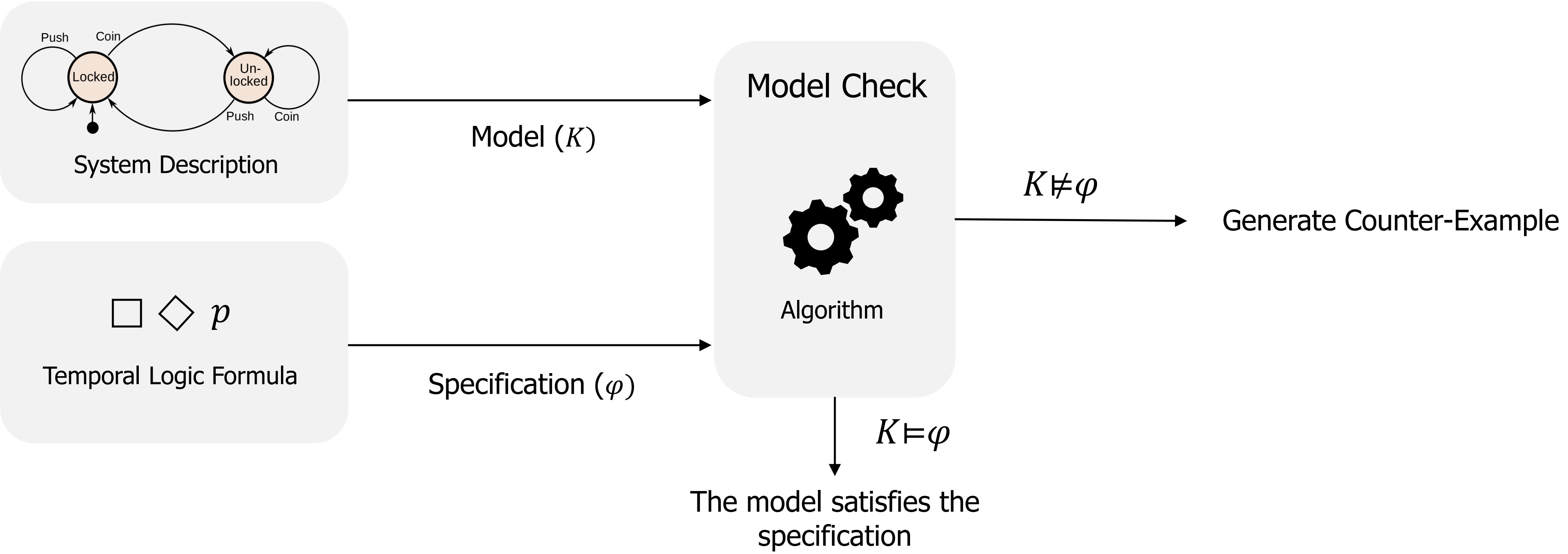
- We propose the use of the Model Checking for buffer overflow discovery in binary C code



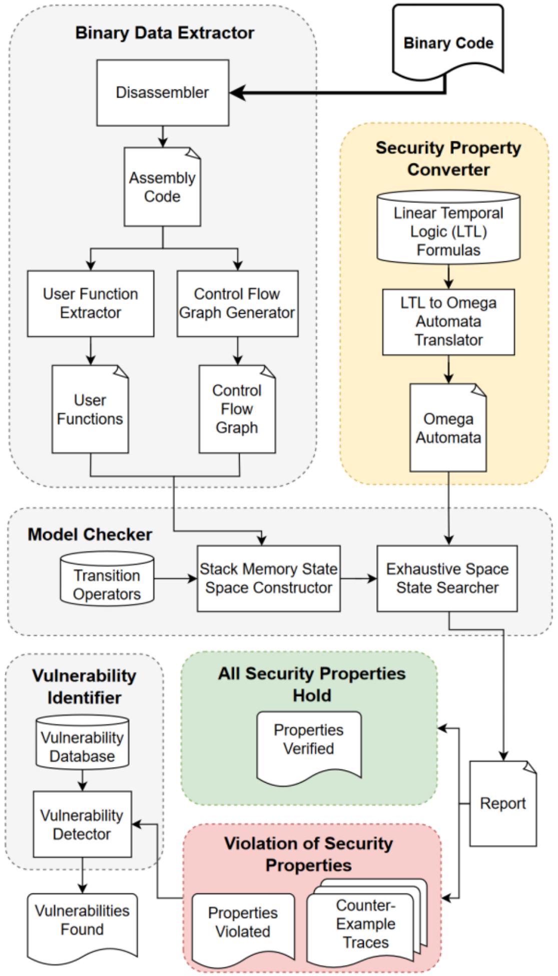
Model Checking

Introduction

- Model checking is a computational technique used to analyse the behaviours of dynamic systems



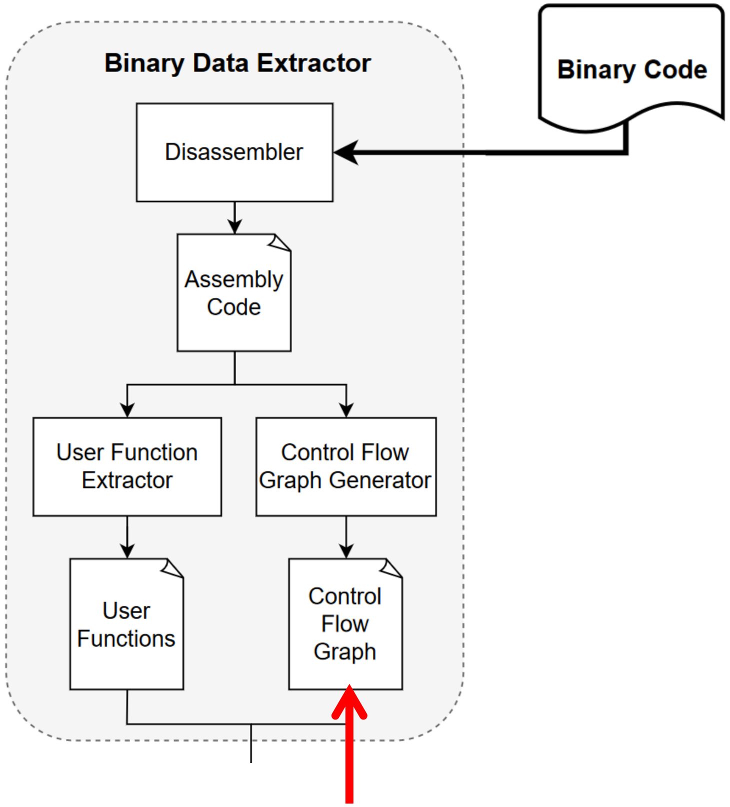
Stack Model Checking Approach

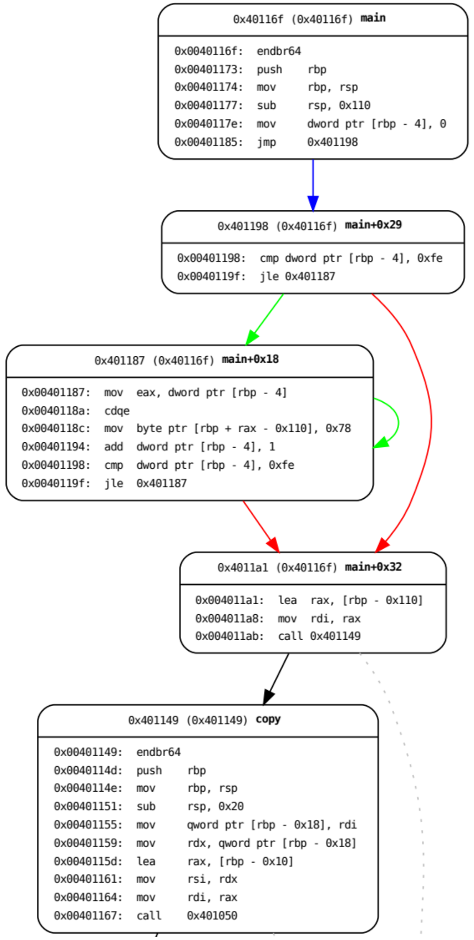


- Binary Data Extractor
- Security Property Converter
- Model Checker
- Vulnerability Identifier

Design Insights

Extracting Data from the Binary

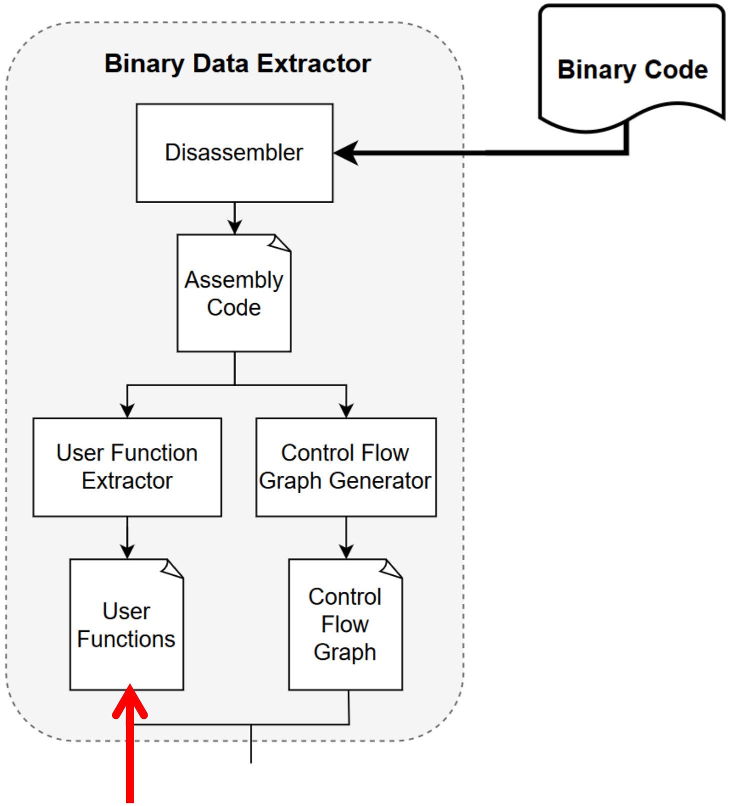




Control Flow Graph

Design Insights

Extracting Data from the Binary

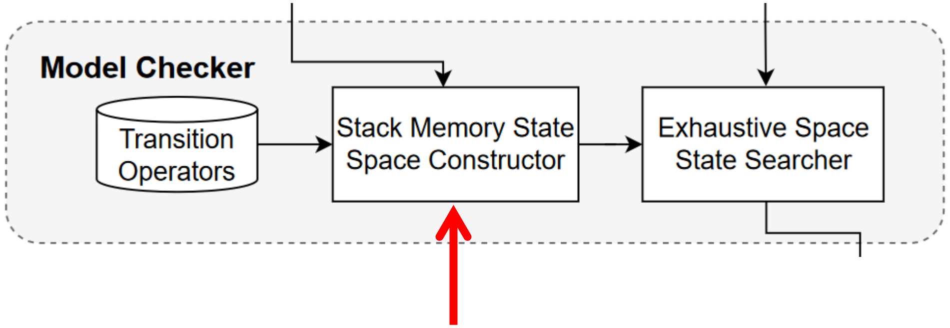


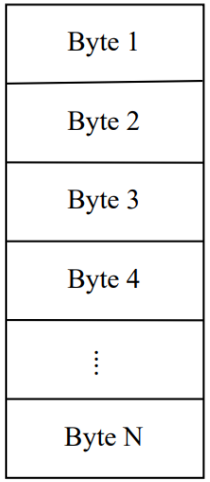

```
Function copy [4198729]
    Syscall: False
    SP difference: 0
    Has return: True
    Returning: True
    Alignment: False
    Arguments: reg: [], stack: []
    Blocks: [0x401149, 0x40116c]
    Calling convention: None
```

User Function Data

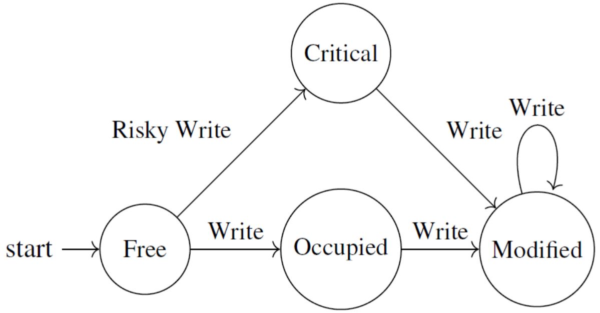
Design Insights

Building the Stack Memory State Space





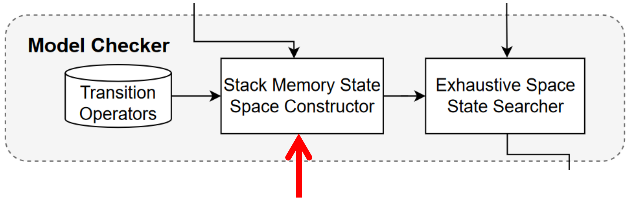
Stack Model

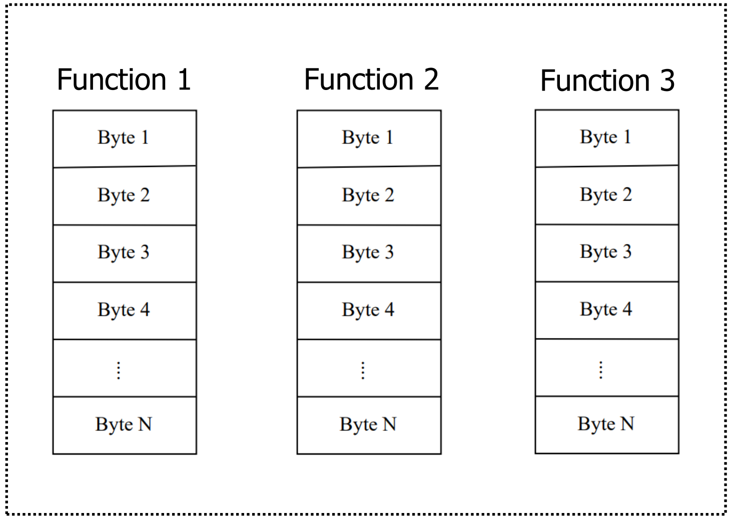


Byte States

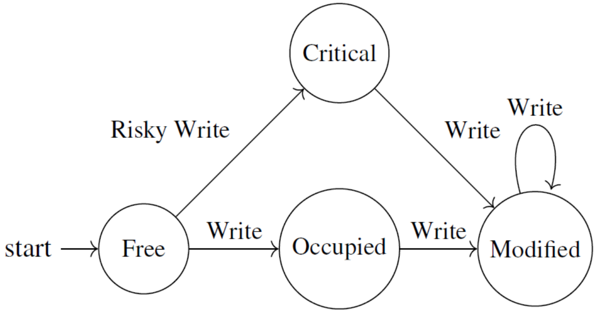
Design Insights

Building the Stack Memory State Space





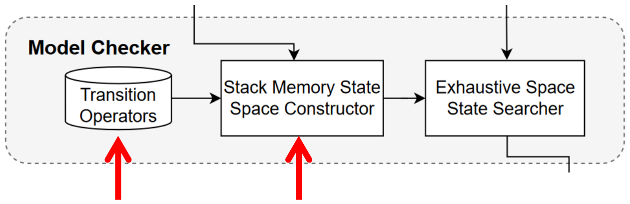
Memory State



Byte States

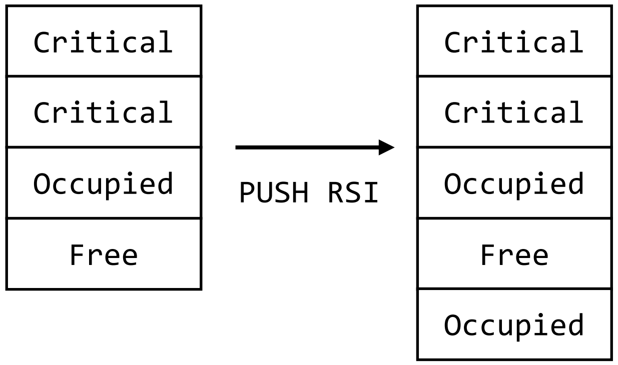
Design Insights

Building the Stack Memory State Space

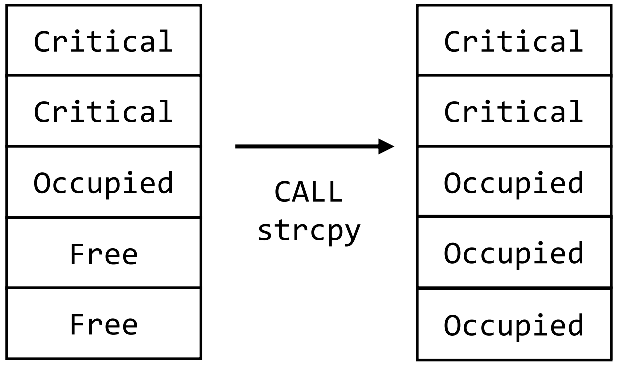


Type of Transition	Operation
Direct	MOV
Direct	PUSH
Direct	POP
Indirect	CALL (e.g., strcpy)

Memory Transition Operators



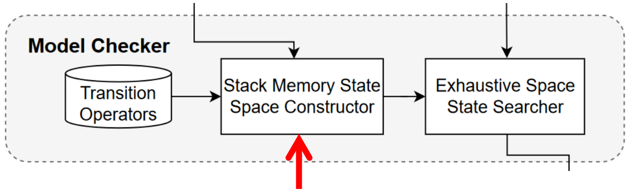
Direct Transition



Indirect Transition

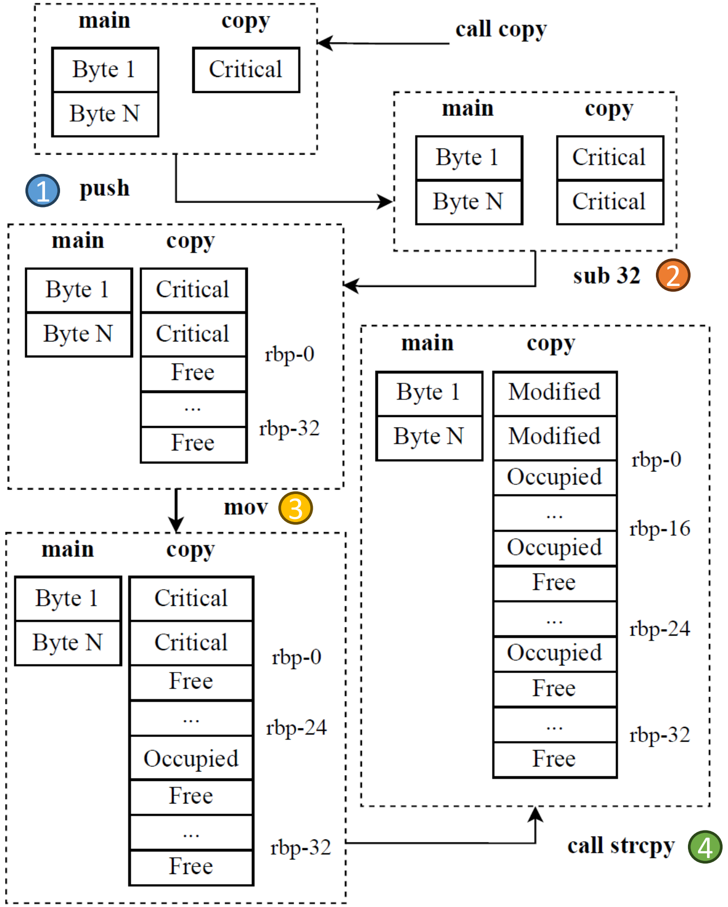
Design Insights

Constructing the State Space



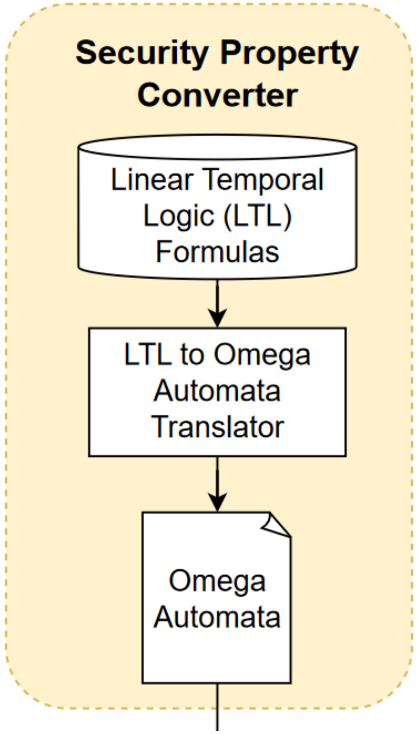
Copy.asm

```
① push rbp
  mov rbp, rsp
② sub rsp, 32
③ mov QWORD PTR [rbp-24], rdi
  mov rdx, QWORD PTR [rbp-24]
  lea rax, [rbp-16]
  mov rsi, rdx
  mov rdi, rax
④ call strcpy
  nop
  leave
  ret
```



Design Insights

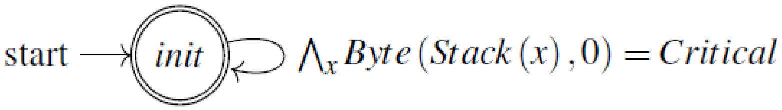
Specifying Security Properties



Security Property

$$\Box\left(\bigwedge_x Byte\left(Stack\left(x\right),0\right)=Critical\right)$$

Translated to Omega Automata

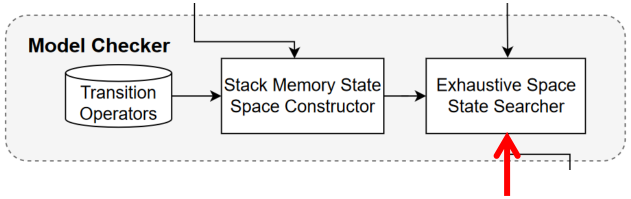


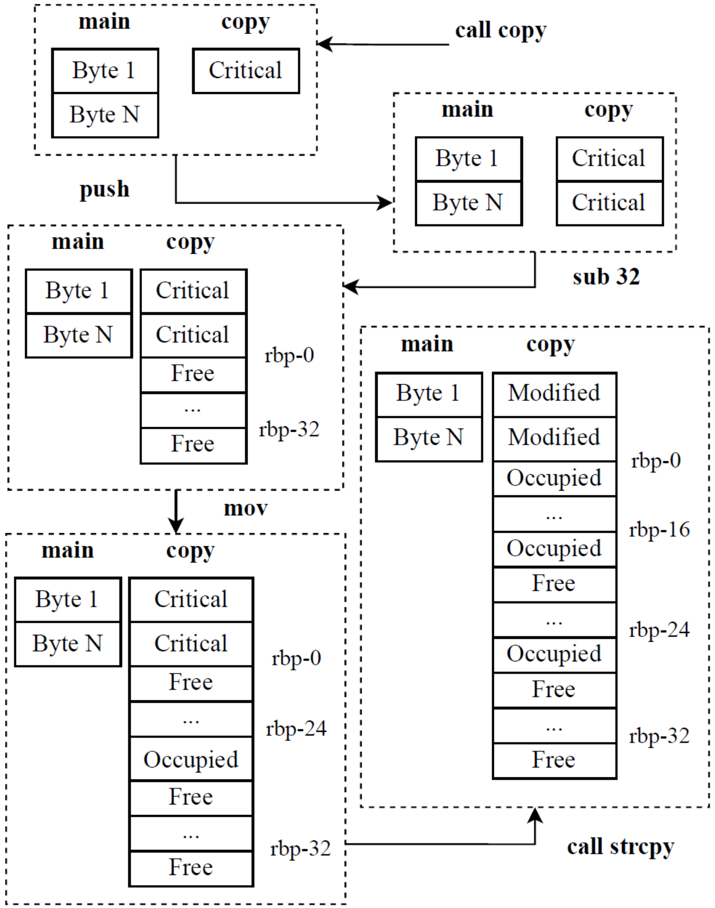
Stack

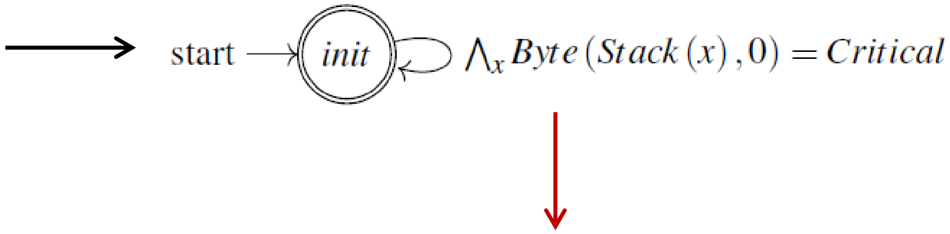
Critical
Critical
Occupied
Free
Occupied

Design Insights

Verifying Security Properties



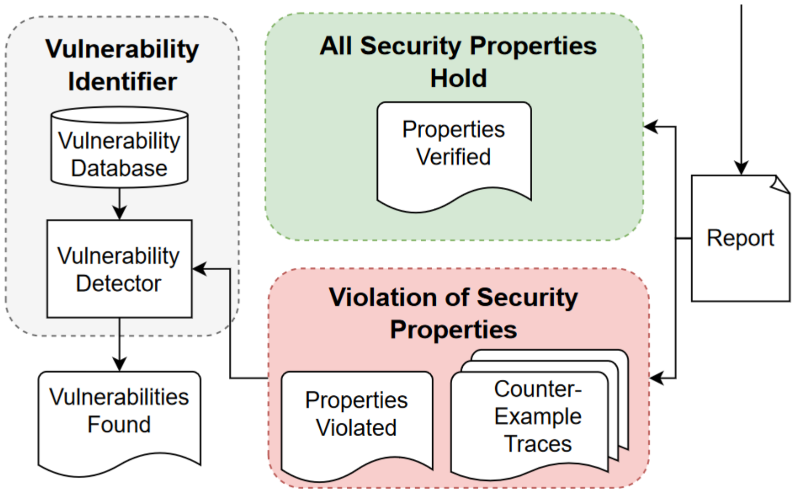




✗ No accepting run found

Design Insights

Identifying Vulnerabilities



Report:

Counter Example Trace

{call copy, push, sub 32, mov, call strcpy}

Sink

Violated Property

$$\Box\left(\bigwedge_{x} Byte\left(Stack\left(x\right),0\right) = Critical\right) \longrightarrow \text{CWE-787}$$

Corresponds

Preliminary Results

LASIGE resilient distributed and networked systems

Implemented a seminal prototype of the Model Checker and tested for 10 small C programs from NIST SARD

Security Property:

$$\neg\left(\Diamond\left(\bigvee_{x} Byte\left(Stack\left(x\right),0\right) = Modified \wedge PreviousTransition = \text{call strcpy}\right)\right)$$

↓

CWE-120

Program	Known Vulnerabilities	Output
Test case 1434	CWE-120, CWE-336	CWE-120
Test case 1430	CWE-120, CWE-336	CWE-120
Test case 1376	CWE-120, CWE-336	CWE-120
Test case 1330	CWE-120	CWE-120
Test case 103	CWE-120	CWE-120
Test case 149145	CWE-120	CWE-120
Test case 149137	CWE-120	CWE-120
Test case 149143	CWE-120	CWE-120
Test case 149139	CWE-120	CWE-120
Test case 149141	CWE-120	CWE-120

Conclusions and Future Work

- Introduced a model checking approach for the stack of binary programs
- Developed a framework for modelling the stack memory and formulating security properties
- Improve the accuracy of the memory state space
- Add new security properties to model more complex behaviors

Thank you!

<u>Luís Ferreirinha</u> and Ibéria Medeiros

Faculty of Sciences of the University of Lisbon