

# All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)

**Matteo Di Pirro**

BSc in Computer Science  
Department of Mathematics

University of Padova

December 7, 2016



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



# Outline

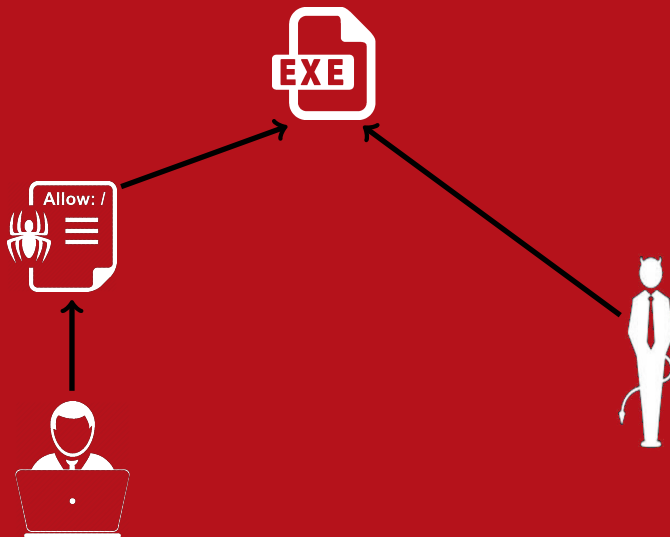
Introduction

The language

Dynamic Taint Analysis

Forward Symbolic Execution

# Of who or what do we trust?





# Input Analysis

There are two essential questions about the input analysis:



# Input Analysis

There are two essential questions about the input analysis:

## 1. Is the final value affected by user input?

- **Dynamic Taint Analysis!**
- Tracks information flow between sources and sinks

# Input Analysis

There are two essential questions about the input analysis:

1. **Is the final value affected by user input?**

- **Dynamic Taint Analysis!**
- Tracks information flow between sources and sinks

2. **What input will make execution reach this line of code?**

- **Forward Symbolic Execution**
- Allows us to reason about the behavior of a program on many different inputs



# Use cases

The number of security applications utilizing these two techniques is enormous:

# Use cases

The number of security applications utilizing these two techniques is enormous:

1. **Unknown Vulnerability Detection:** monitor whether user input is executed



# Use cases

The number of security applications utilizing these two techniques is enormous:

1. **Unknown Vulnerability Detection:** monitor whether user input is executed
2. **Automatic Input Filter Generation:** detect and remove exploits from the input stream

# Use cases

The number of security applications utilizing these two techniques is enormous:

1. **Unknown Vulnerability Detection:** monitor whether user input is executed
2. **Automatic Input Filter Generation:** detect and remove exploits from the input stream
3. **Forward Symbolic Execution:** analyze how information flows through a malware binary

# Use cases

The number of security applications utilizing these two techniques is enormous:

1. **Unknown Vulnerability Detection:** monitor whether user input is executed
2. **Automatic Input Filter Generation:** detect and remove exploits from the input stream
3. **Forward Symbolic Execution:** analyze how information flows through a malware binary
4. **Test Case Generation:** automatically generate inputs to test programs

# SimplL

Designed to demonstrate the critical aspects of this analysis.

*program* ::= *stmt*\*

*stmt s* ::= *var* := *exp* | store(*exp*, *exp*)  
| goto *exp* | assert *exp*  
| if *exp* then goto *exp*  
| else goto *exp*

*exp e* ::= load(*exp*) | *exp*  $\Diamond_b$  *exp* |  $\Diamond_u$  *exp*  
| *var* | get\_input(*src*) | *v*


$\Diamond_b$  ::= typical binary operators

$\Diamond_u$  ::= typical unary operators

*value v* ::= 32-bit unsigned integer

## SimplL Grammar

# Dynamic Taint Analysis

```
x := get_input()  
z := 42  
y := x + z  
goto y
```

# Dynamic Taint Analysis

Tainted

 `x` := get\_input()

`z` := 42

`y` := `x` + `z`

goto `y`

`x` is derived from  
a tainted source

# Dynamic Taint Analysis

Untainted `x := get_input()`

`z := 42`

`y := x + z`

`goto y`

`z` is a "static"  
constant

# Dynamic Taint Analysis

```
x := get_input()
```

```
z := 42
```

```
 y := x + z → Is y tainted?
```

```
goto y
```



# Dynamic Taint Analysis

```
x := get_input()
```

```
z := 42
```

```
 y := x + z → Is y tainted?
```

```
goto y
```

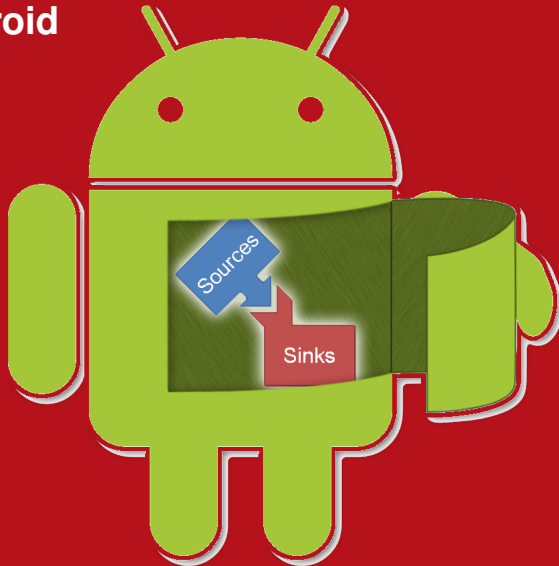
It depends on the  
selected policy

# What's a policy?

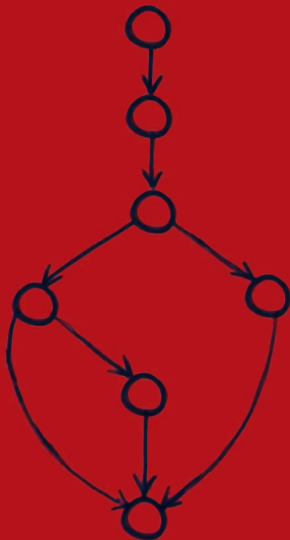
- ▶ A taint policy specifies three properties:
  - **Taint Introduction**
    - ▶ How is taint introduced into a system?
  - **Taint Propagation**
    - ▶ How does taint propagate into a system?
  - **Taint Checking**
    - ▶ Is the current operation secure?
  
- ▶ **Undertainting vs Overtainting**

# Can we go further?

## TaintDroid

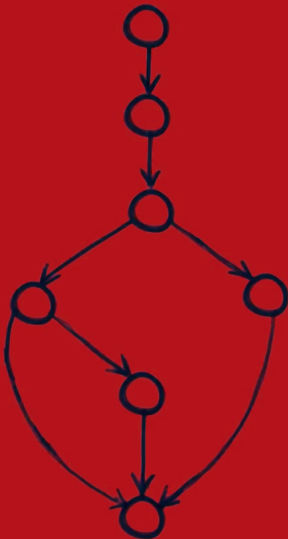


# Limitations



- ▶ Undertainting and overtainting are nearly **unavoidable!**
  - **Time of detection** vs **Time of attack**
- ▶ Pure dynamic taint analysis considers **data flows...**

# Limitations



- ▶ Undertainting and overtainting are nearly **unavoidable!**
  - **Time of detection** vs **Time of attack**
- ▶ Pure dynamic taint analysis considers **data flows...**
- ▶ ...but it ignores **control-flows**
  - What about different security policies for different I/O channels?  
→ **Static analysis**

# Forward Symbolic Execution

- ▶ We can reason about the behavior of a program using the logic...
- ▶ ... and it is **conceptually** a very simple process

```
x := 2 * get_input(src)
if x - 5 == 14 then goto 3 else goto 4
// line 3: catastrophic failure
// line 4: normal behaviour
```

# Forward Symbolic Execution

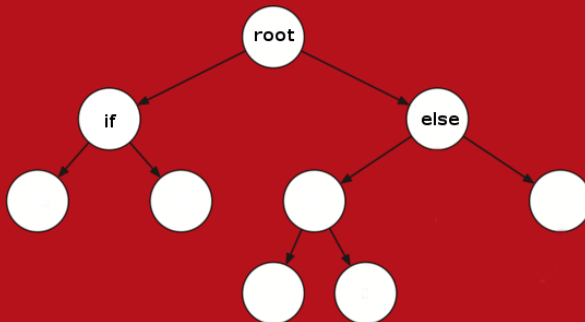
- ▶ We can reason about the behavior of a program using the logic...
- ▶ ... and it is **conceptually** a very simple process

```
x := 2 * get_input(src)
if x - 5 == 14 then goto 3 else goto 4
// line 3: catastrophic failure
// line 4: normal behaviour
```

- ▶ `get_input(src)` now returns a **symbol** instead of a concrete value
- ▶ But now expressions **cannot** be fully evaluated to a concrete value

# Path Selection and Performance

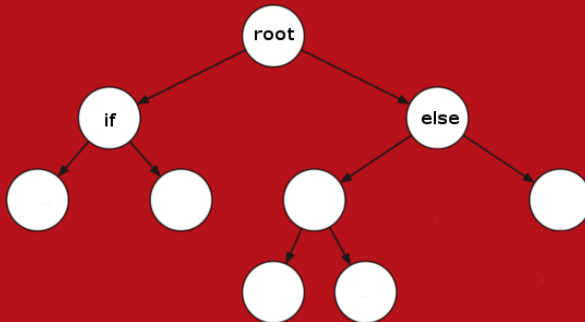
- ▶ Every conditional jump we must decide what path to follow first
  - But some path may never terminate
- ▶ Exponential blowup due to branches





# Path Selection and Performance

- ▶ Every conditional jump we must decide what path to follow first
  - But some path may never terminate
    - $\text{while } (3^n + 4^n == 5^n) \{n++; \dots\}$
- ▶ Exponential blowup due to branches



# Path Selection and Performance

- ▶ Every conditional jump we must decide what path to follow first
  - But some path may never terminate
$$\text{while } (3^n + 4^n == 5^n) \{n++; \dots\}$$
- ▶ Exponential blowup due to branches
- ▶ Solutions
  - **Path Selection Heuristic**
    - ▶ Concolic Testing
    - ▶ Depth-First or Random Search
  - More and faster **hardware**
  - Identify **redundancies** between formulas
  - Identify **independent subformulas**

# Memory Address Problems

- ▶ What are we supposed to do if a referenced address is derived from user input?
  - LOAD, STORE → **Symbolic Memory Address**
  - GOTO → **Symbolic Jumps**
- ▶ Solutions
  - Concolic testing
  - SMT (**S**atisfiability **M**odulo **T**heories) solvers
  - Static and alias analysis



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

Thank you for allowing me to  
taint your time!

Questions?



DIPARTIMENTO  
**MATEMATICA**