

# THE (NOT SO PROFITABLE) PATH TOWARDS AUTOMATED HEAP EXPLOITATION

Thaís Moreira Hamasaki

barbieAuglend

October 2018



---

# DISCLAIMER

---

*This research was accomplished by  
me in my personal capacity during  
my spare time.*

**DON'T BE TOO JUDGEMENTAL PLEASE! :)**



---

**full disclosure: I am NOT a vulnerability researcher!**

---



---

# ABOUT ME

---

```
echo 'Stare at binaries during the day';

echo 'Blackhoodie - Core Organizer and Board Member';
echo 'HackLu`s program`s committee';
echo 'Disobey`s Lead of Technical Content';
echo 'x86 Assembly & RE101 - Lead of both groups @chaosdorf';
echo 'Logical Programming, RE, static analysis, Mountaineering FTW';
echo 'Wannabe "Karaoke" singer';

echo 'Stare at binaries by night';
```

BlackHoodie



---

# WHAT AM I GOING TO TALK ABOUT?

---

- **constraint logic programming (CLP)**
- **solvers**
- **static analysis scalability**
- **the memory**
- **oh yeah, heaps...**



---

# S..A..T.. WHAT?

---



- **Solver!**
- **Satisfiability Modulo Theories (SMT)**









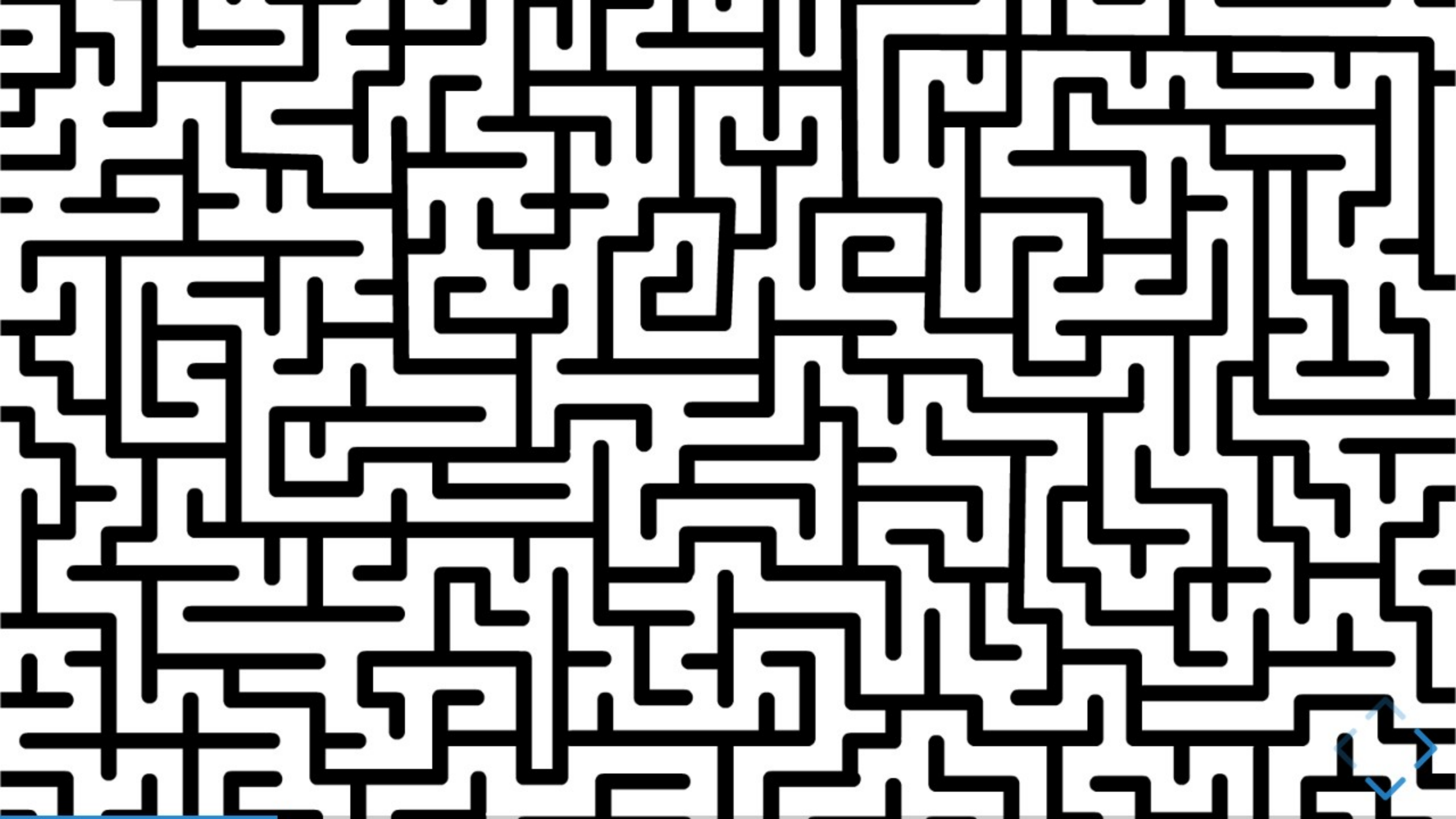
---

# CONSTRAINTS

---







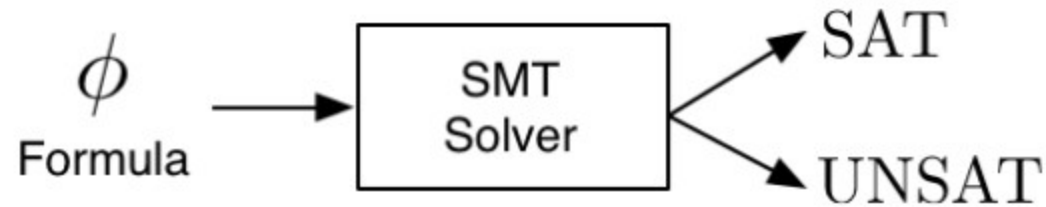
***"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." Eugene C. Freuder, Constraints, April 1997***



---

# AUTOMATED THEOREM PROVING

---



- **Hardware and Software → Large-scale verification**
- **Languages specification and Computing proof obligations**





---

# SYMBOLIC EXECUTION

---



# IT LOOKS LIKE THAT ...

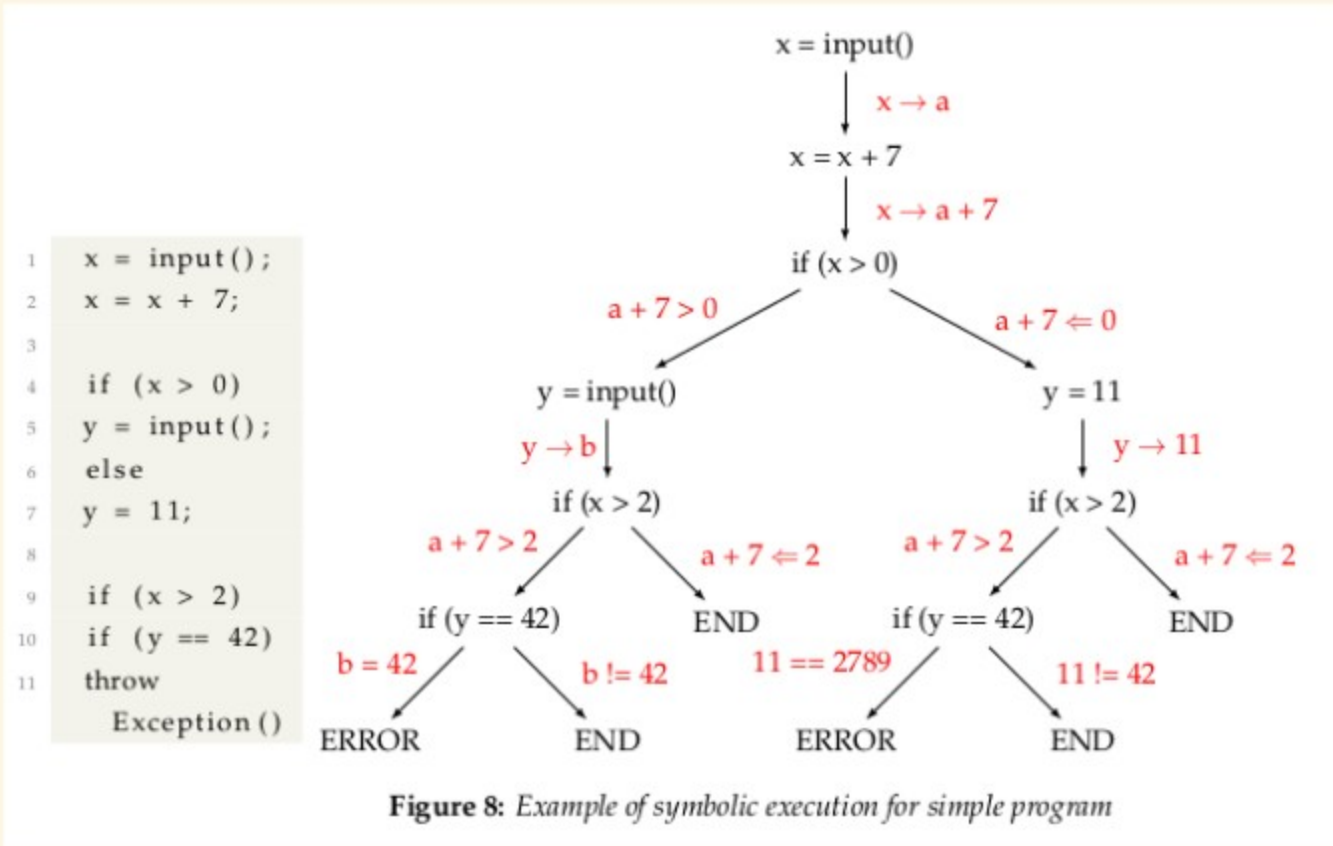


Figure 8: Example of symbolic execution for simple program



---

# HOW IT WORKS

---

- **Create a process ( $pc = 0$ ,  $state = []$ )**
- **Add the process ( $pc$ ,  $state$ ) to the domain system  $D$**
- **while  $D$  not empty:**
  - **Remove process ( $pc$ ,  $state$ ) from system**
  - **Execute it until the next branching point**
    - **If both paths are feasible, add both to  $D$**
    - **if just one is feasible, add the feasible path and the negation of the not feasible path to  $D$**

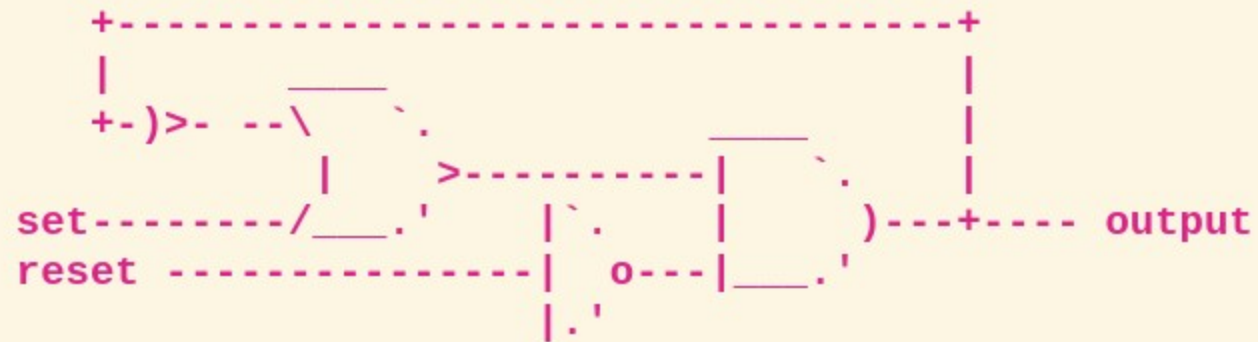




---

# THE LOGIC GATES OF THE MEMORY

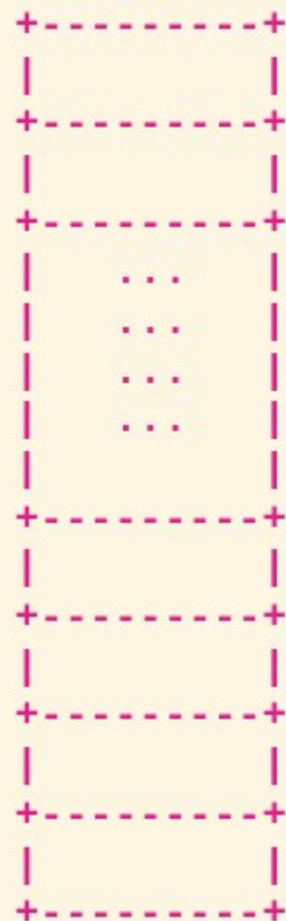
---



---

# 50 SHADES OF MEMORY

---



STACK

the IMPLEMENTATION

HEAP



---

# A HEAP OF INFORMATION

---





---

# APPLICATIONS

---



---

# MALWARE ANALYSIS

---



- **Obfuscation**
- **Compiler optimizations**
- **Crypto-analysis**



---

# BUG HUNTING

---



- **Fuzzing**
- **Code verification**
- **Binary Analysis**



---

# EXPLOITATION

---



- **PoC (Proof of Concept)**
- **REG (Automated Exploit Generation)**
- **APG (Automated Payload Generation)**





---

# WHAT WE ARE LOOKING FOR

---



- **Vulnerable**



---

# WHAT WE ARE LOOKING FOR

---



- **Vulnerable *AND* Exploitable**



---

# HOW TO CRASH!

---





---

# AUTOMATION OUT THERE

---



- **Exploratory testing**
- **Dynamic taint analysis**
- **Abstract interpretation**





---

# AUTOMATION OUT THERE

---



- **Klee**
  - **Open source symbolic executor**
  - **Runs on top of LLVM**
- **Manticore**
  - **Symbolic execution**
  - **Taint analysis**
  - **Binary instrumentation**



---

# TOOL OF CHOICE

---



**FORWARD**

**SYMBOLIC**

**EXECUTION**



---

# EXPLOIT GENERATION

---



Find a bug — easy right?

Def: Vulnerable Path for input  $\epsilon$  is  
 $\Pi_{\text{(vulnerability)}}(\epsilon)$



---

# PLAN

---



**Theorem: Given a program,  
automatically find vulnerabilities  
and generate exploits for them.**





---

# PLAN

---



**Theorem: Given a program,  
automatically find vulnerabilities  
and generate exploits for them.**

- **direct influence**



---

# PLAN

---



**Theorem: Given a program,  
automatically find vulnerabilities  
and generate exploits for them.**

- **indirect influence**

```
malloc(strlen(user_input));
```



---

# EXPLOIT GENERATION

---



Check if it is exploitable

Not that easy anymore...

$\Pi(\text{vulnerability})(\epsilon) \wedge \Pi(\text{exploit})(\epsilon) = \text{true}$

Where  $\Pi_{\text{exploit}}(\epsilon)$  is the attacker's logic



---

# EXPLOIT GENERATION

---



Implement  $\Pi_{\text{exploit}}(\epsilon)$

for a really special case  $\Pi_{\text{vulnerability}}(\epsilon) \wedge$   
 $\Pi_{\text{exploit}}(\epsilon)$

and then it works MOST of the times

is it really automated then?





---

# EXPLOIT GENERATION

---



## EXTRA: Evaluate

- Find the HEAP
- Exploit Verification
- State Space Explosion
- Environment Definition



---

# LIMITATIONS

---



# THEORETICAL #1

## Rice's Theorem

### Theorem

Let  $L$  be a subset of strings representing Turing machines, where

1. If  $M_1$  and  $M_2$  recognize the same language, then either  $\langle M_1 \rangle, \langle M_2 \rangle \in L$  or  $\langle M_1 \rangle, \langle M_2 \rangle \notin L$ .

2.  $\exists M_1, M_2$  s.t.  $\langle M_1 \rangle \in L$  and  $\langle M_2 \rangle \notin L$ .

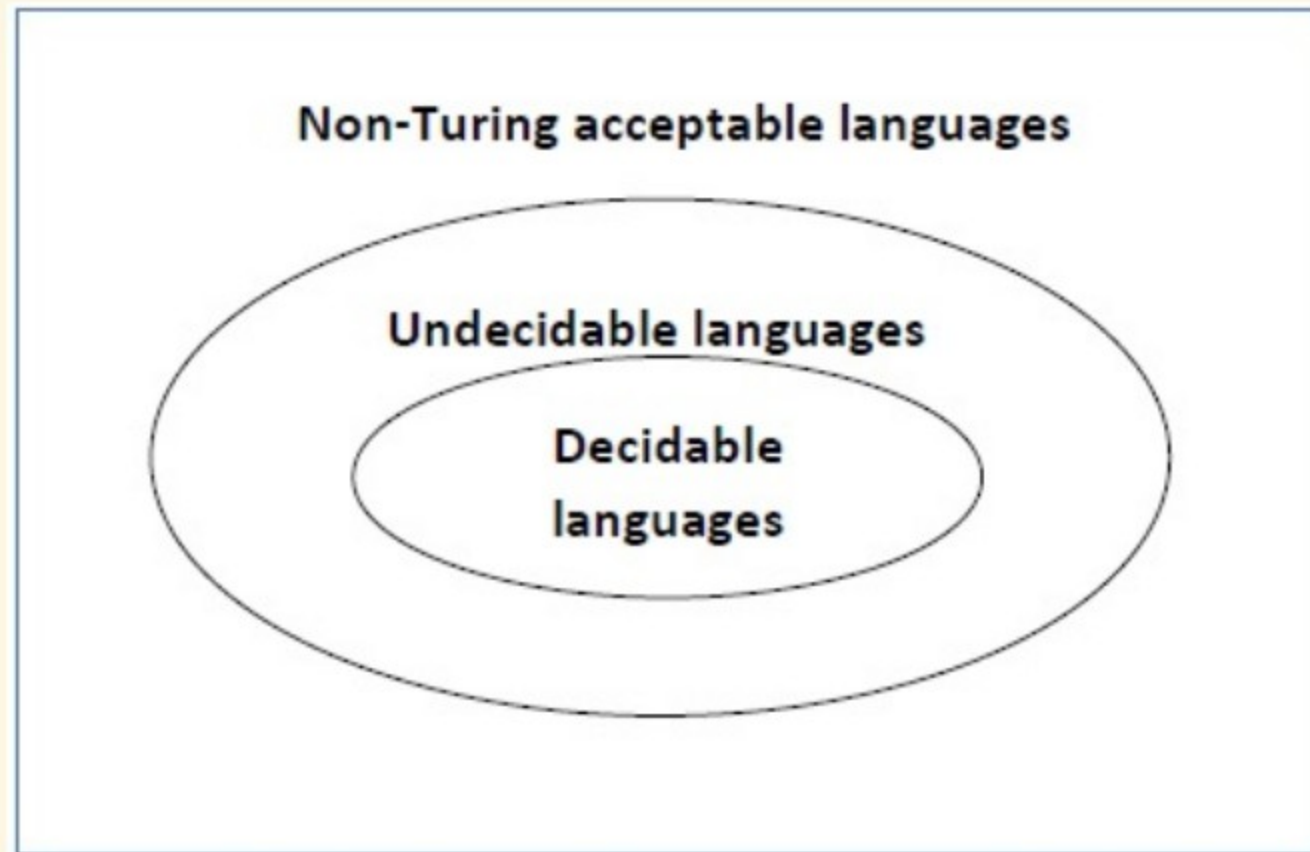
Then  $L$  is undecidable.



---

# THEORETICAL #2

---

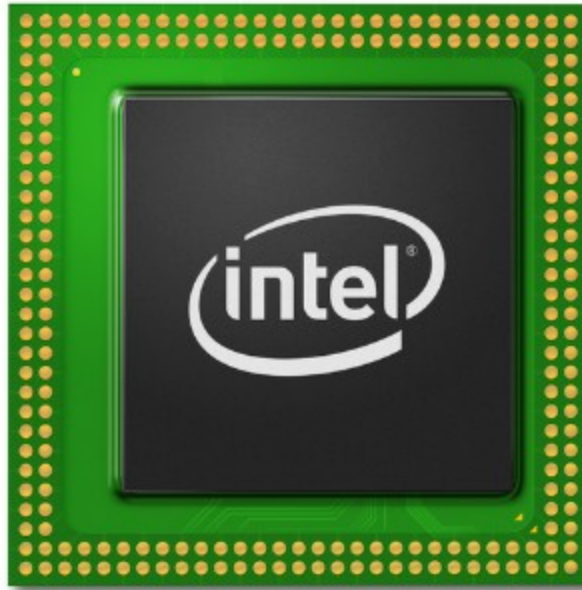




---

# PRACTICAL

---



Remember...



---

# CONCLUSION

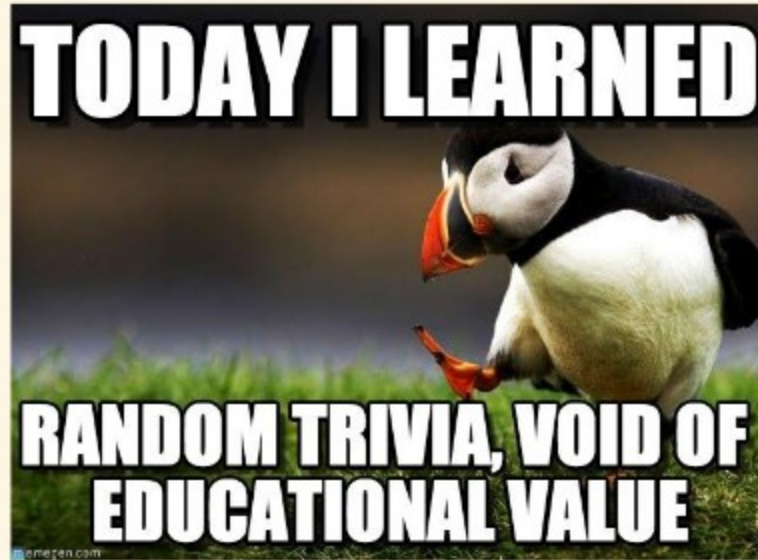
---



---

# LEARNINGS / TAKE AWAY

---



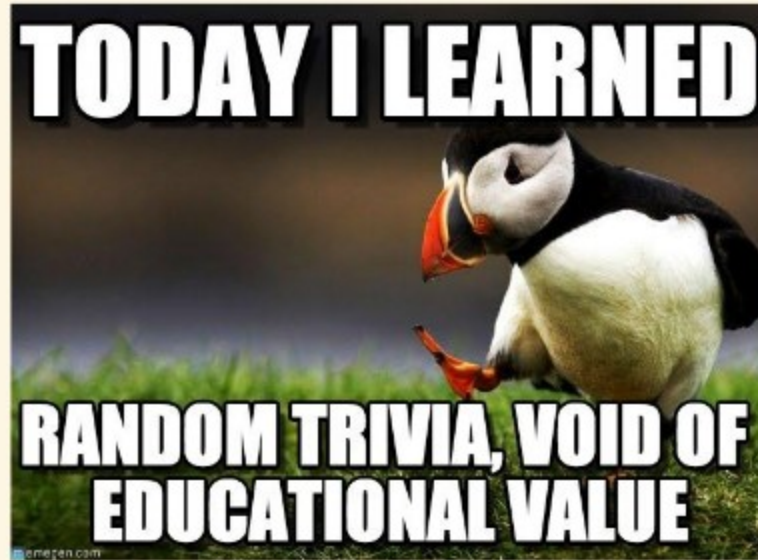
- Symbolic execution is a powerful tool while ~~analysing malware~~ for vulnerability research
- SMT solvers can reason and generate exploits



---

# LEARNINGS / TAKE AWAY

---



- Symbolic execution can be ~~is~~ a powerful tool for vulnerability research
- SMT solvers can reason and generate exploits





---

# WORK DONE:

---



- a binary garbage-code eliminator, a XOR search, some "cryptographic" algorithm breaker, a generic unpacker, a binary structure recognizer, a C++ class hierarchy reconstructor.



---

# WORKING ON ...

---



- **specialized constraint inference assistant for computer security problems**



---

# ACKNOWLEDGMENTS

---

- **Sean Heelan**
  - **Automated Heap Layout Manipulation for Exploitation (Heelan et al. to appear in Usenix Security 2010)**
  - **and his time;**
  - **and inspiration!**
- **Marion Marschalek**
- **Heap Models for Exploit Systems (Vanegue, Langsec 2015)**
- **and the Intel Documentation I think ... ?**



---

# QUESTIONS?

---



[@barbieAuglend](https://www.instagram.com/barbieAuglend)

[barbie@barbieauglend.re](mailto:barbie@barbieauglend.re)

