The (not so profitable) path towards automated heap exploitation

Thaís Moreira Hamasaki
barbieauglend
barbie@barbieauglend.re

December / 2018 - iSecCon - Portland, US

DISCLAIMER

This research was accomplished by me in my personal capacity during my spare time. The views expressed are those of the speaker (me) and not, necessarily, of my employee.

full disclosure: I am NOT a vulnerability researcher!

About me

- BlackHoodie Core Organizer and Board Member
- HackLu`s program`s committee
- Disobey`s Lead of Technical Content
- x86 Assembly & RE101 - Lead of both groups @chaosdorf

- Logical Programming, RE, static analysis, Mountaineering FTW
- Wannabe "Karaoke" singer
- Currently staring at binaries @F-Secure

What am I going to talk about?

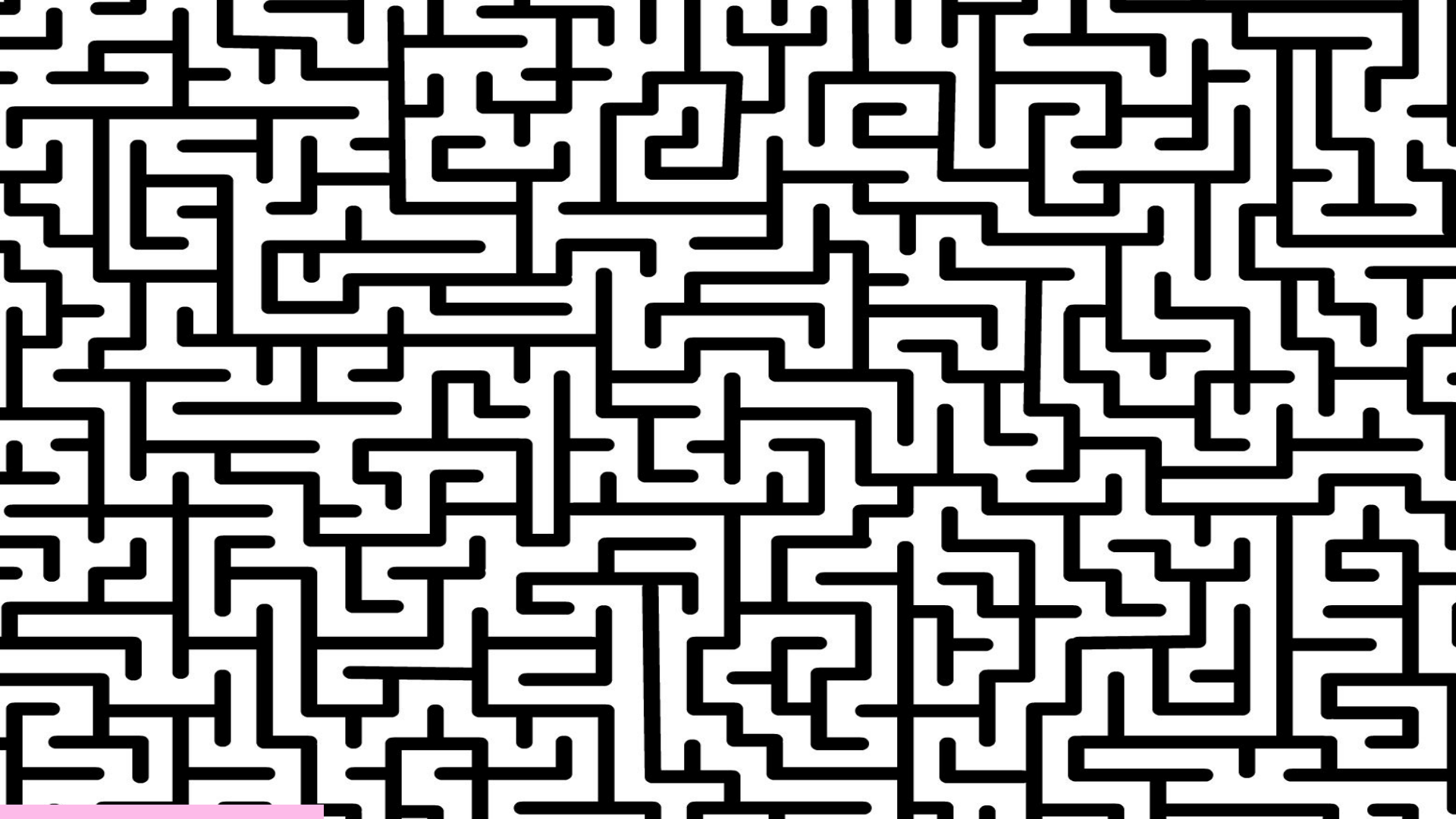
- constraint logic programming (CLP)
- solvers
- static analysis scalability
- the memory
- oh yeah, heaps...

S...A...T... What?

SOLVER!

Satisfiability Modulo Theories (SMT)

Boolean <-> arithmetic

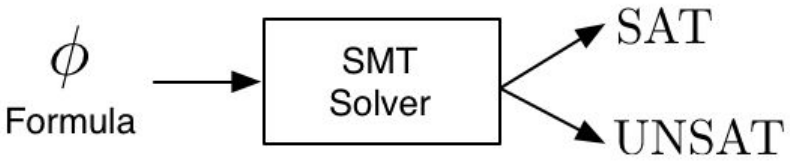


Constraints

"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

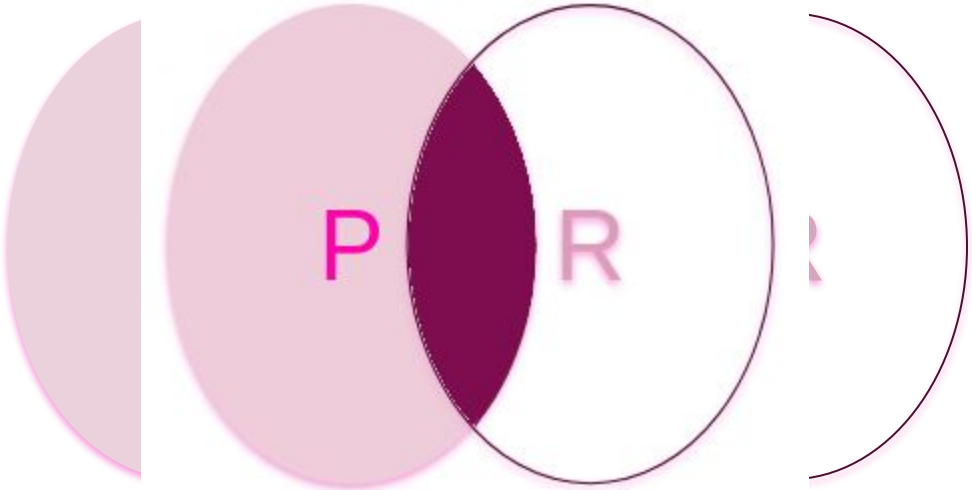
Automated Theorem Proving



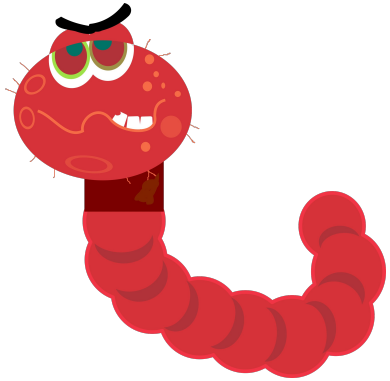
- Hardware and Software -> Large-scale verification
- Languages specification and Computing proof obligations
- Applied Math ~ Computer Science

Model checking

Property P:
SOMETHING BAD NEVER HAPPENS in my perfect code



Applications



- Malware analysis
  - Obfuscation
  - Compiler optimizations
  - Crypto-analysis

- Bug Hunting
  - Fuzzing
  - Code verification
  - Binary Analysis
- Exploitation
  - PoC
  - AEG
  - APG

Symbolic Execution

```
1   x = input();
2   x = x + 7;
3
4   if (x > 0)
5   y = input();
6   else
7   y = 11;
8
9   if (x > 2)
10  if (y == 42)
11  throw
      Exception()
```

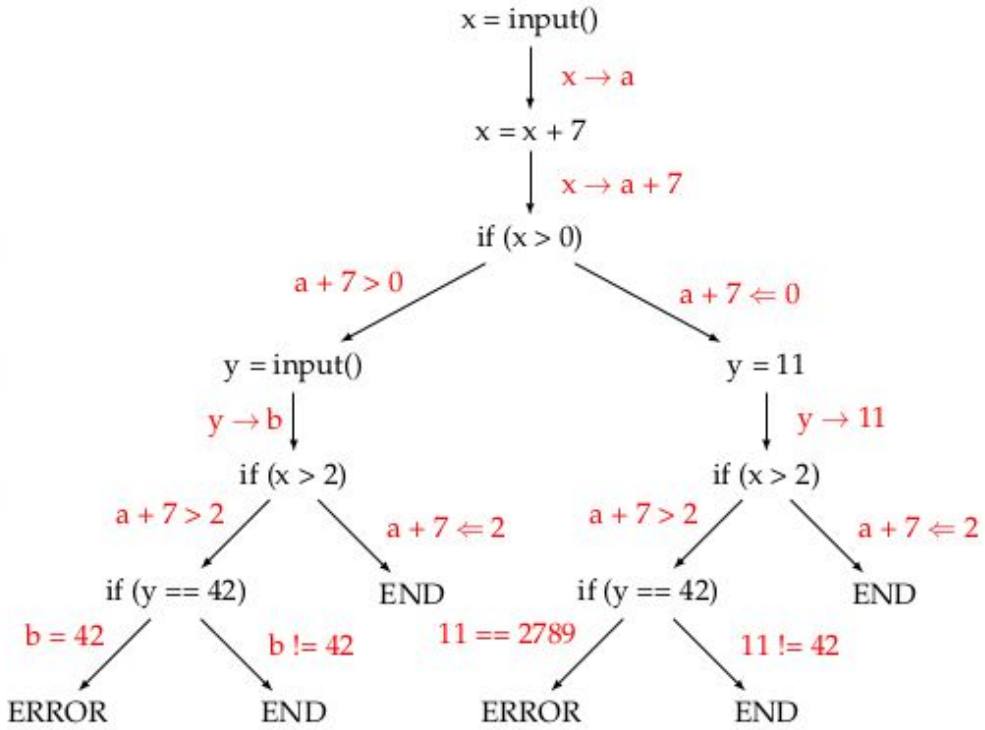
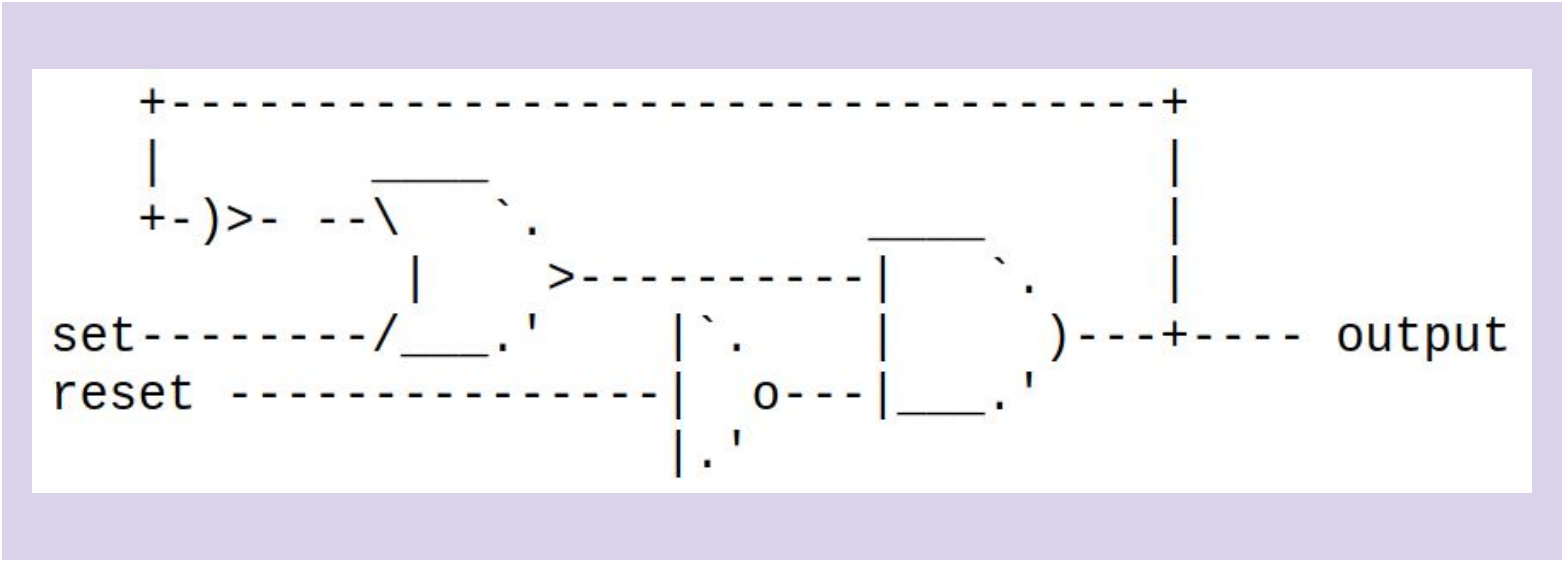


Figure 8: *Example of symbolic execution for simple program*

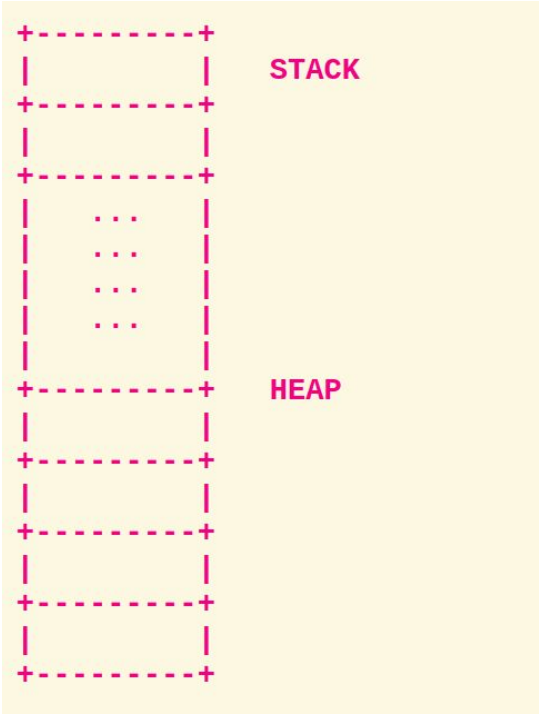
The Algorithm

- 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

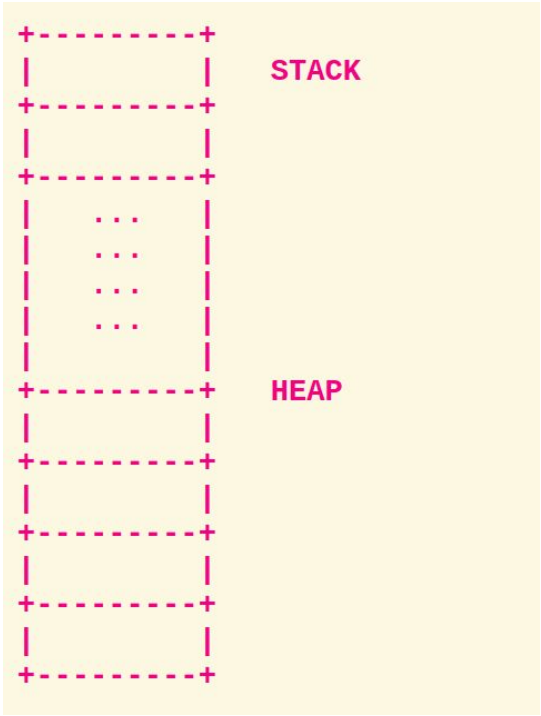
The logic gates of the Memory



50 shades of Memory



50 shades of Memory



A heap of information

A heap is a managed memory region that allows for the dynamic allocation of variable-sized blocks of memory in runtime. A program simply requests a block of a certain size and receives a pointer to the newly allocated block (assuming that enough memory is available). Heaps are managed either by software libraries that are shipped alongside programs or by the operating system. Heaps are typically used for variable-sized objects that are used by the program or for objects that are too big to be placed on the stack. “ Eldad Eilam

Heap allocation

How-to: void *malloc(size_t size);

- Call heap_allocator
- Check for unused heap space
- Return a heap block
- Create a pointer to the block
- Return the pointer

Unused != Clean

How-to: void free(void *pointer);

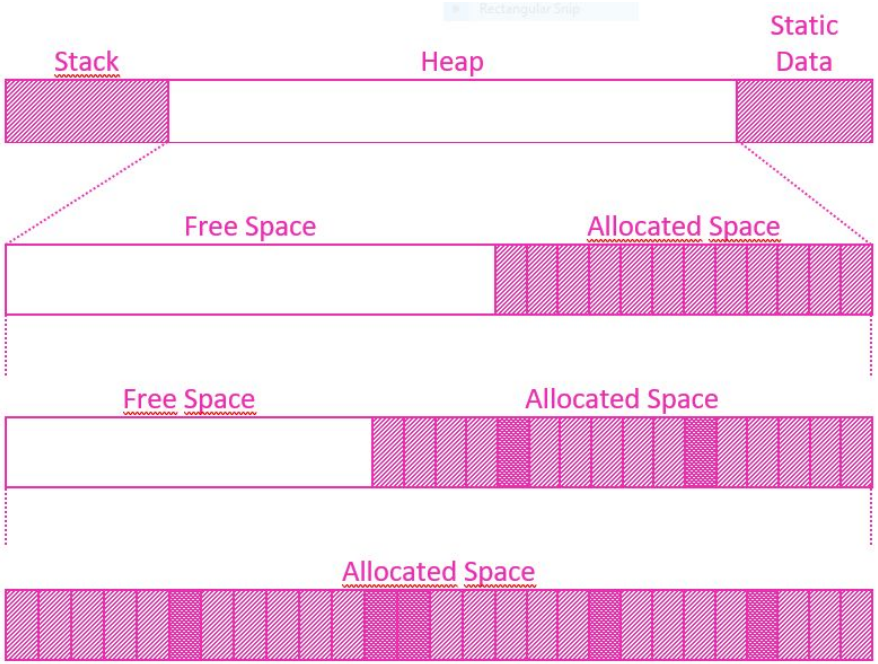
- The object is added to the free space list
- The programmer is responsible
- The data is not deleted

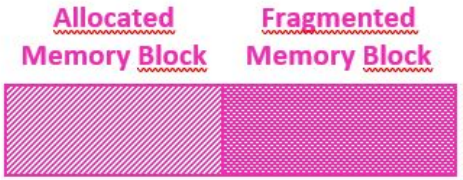
Pointer errors

Infoleaks

For consistency: other functions can be used like realloc() and malloc_trim() - not in the scope.

Memory Fragmentation





What I was looking for

VULNERABLE

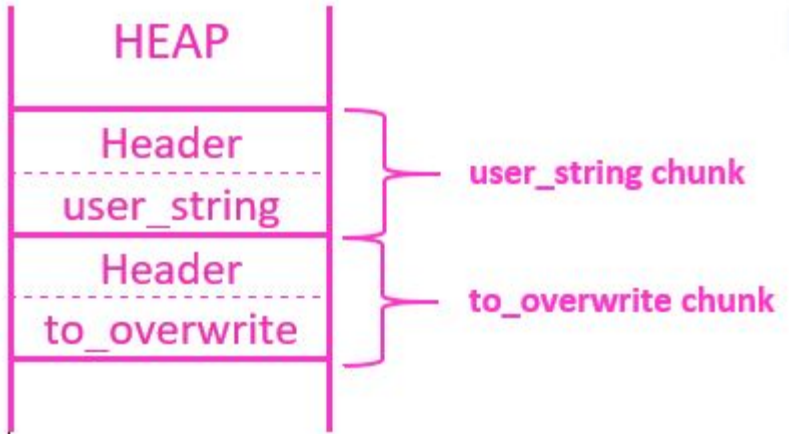
&

EXPLOITABLE

Heap bugs

```
/* code ... */
char *user_string =
malloc(8);
int *to_overwrite =
malloc(4);

*to_overwrite = 1234;
gets(user_string);
/* ... */
```



Heap bugs

```
/* code ... */
q = p = malloc(1337);
free(p);
/* more code containing
malloc's */
q[100] = 1234;
/* ... */
```

```
/* code ... */
q = p = malloc(1337);
free(p);
p = NULL;
/* more code containing
malloc's */
q[100] = 1234;
/* ... */
```

Automation out there

- Exploratory testing
- Dynamic taint analysis
- Abstract interpretation

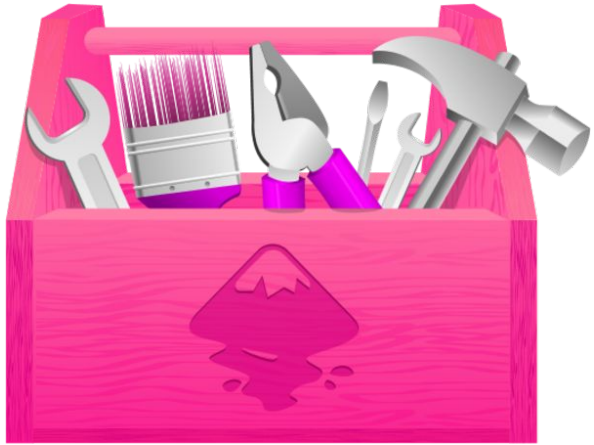
- Klee
  - Open source symbolic executor
  - Runs on top of LLVM
- Manticore
  - Symbolic execution
  - Taint analysis
  - Binary instrumentation
- MAYHEM
  - Automated binary exploitable vulnerabilities discover

Tool of choice

FORWARD

SYMBOLIC

EXECUTION



Exploit Generation

1 - Find a bug!

Easy right?

Def: Vulnerable Path for input ε is Π_(vulnerability)(ε)

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

- Direct influence
- Indirect influence “malloc(sizeof(struct hey *) * user_input); “

Exploit Generation

1 - Find a bug!

2 - Check if it is exploitable

Not that easy anymore...

Def: Π_(vulnerability)(ε) ⋀ Π_(exploit)(ε) = true

Where Π_(exploit)(ε) is the attacker's logic

Exploit Generation

1 - Find a bug!

2 - Check if it is exploitable

3 - Implement Π_(exploit)(ε)

is it really automated then?

Which works for a really special case Π_(vulnerability)(ε) ∧ Π_(exploit)(ε)

and then it works MOST of the times

Exploit Generation

1 - Find a bug!	Find the HEAP
2 - Check if it is exploitable	Exploit Verification
3 - Implement Π_(exploit)(ε)	State Space Explosion
4 - Evaluate	Environment Definition

Limitations

Rice's Theorem

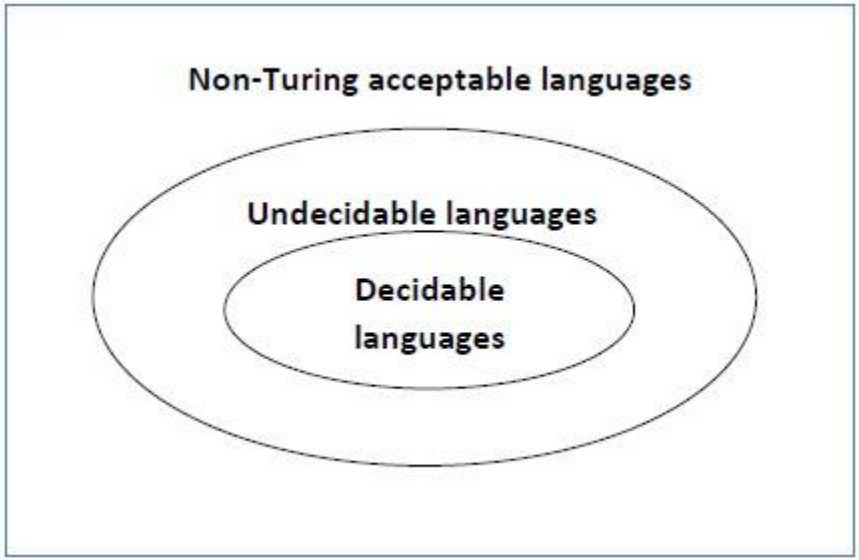
Theorem

Let L be a subset of Strings representing Turing machines, where

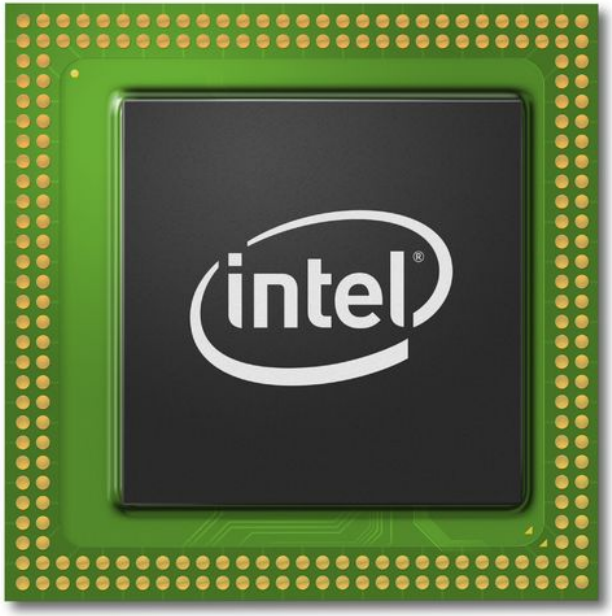
1. If M_1 and M_2 recognize the same language, then either $\langle M_1 \rangle, \langle M_2 \rangle \in L$ or $\langle M_1 \rangle, \langle M_2 \rangle \notin L$.
2. $\exists\ M_1, M_2$ s.t $\langle M_1 \rangle \in L$ and $\langle M_2 \rangle \notin L$.

Then L is undecidable.

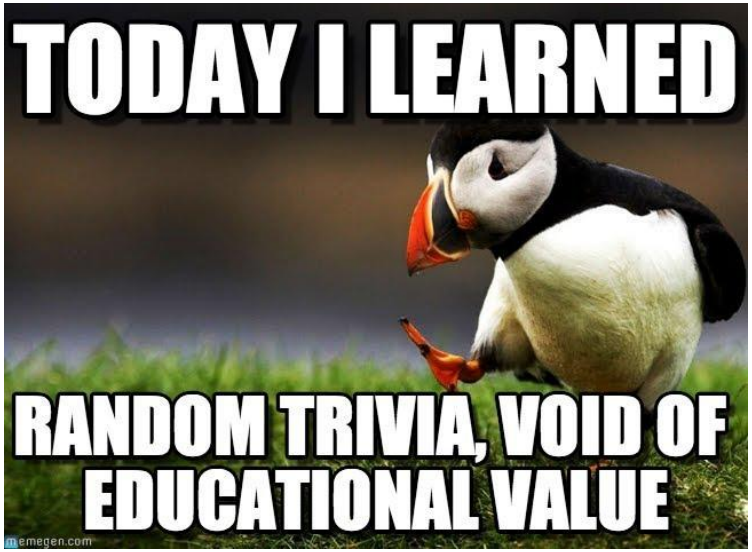
Limitations



Limitations

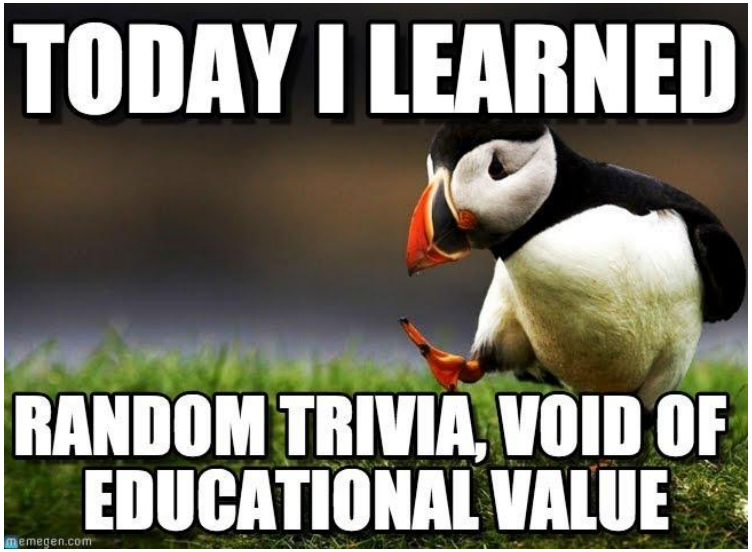


Conclusion : Take away



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

Conclusion : Take away



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

Conclusion : 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

And what now?

- FULLY automating tasks is not always the best choice, especially for non-deterministic systems;
- Check which subtasks can be automated;
- Let it go and don’t feel afraid of failing;
- Create assistants and advisors to accelerate and facilitate the work of experts.

Conclusion : Working on

A 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 2018)
- and his time
- and inspiration!

Marion Marschalek

Rodrigo Branco

- Heap Models for Exploit Systems (Vanegue, Langsec 2015)
- and the Intel Documentation I think ... ?

Wanna contribute?

A specialized constraint inference assistant for computer security problems.

Thaís Moreira Hamasaki
barbieauglend
barbie@barbieauglend.re

December / 2018 - iSecCon - Portland, US

For sources and more detailed information, please ask for the paper, which is going to be published soon (I hope!)