Introduction to Reversing with Z3

RPISEC

Avi Weinstock (aweinstock)

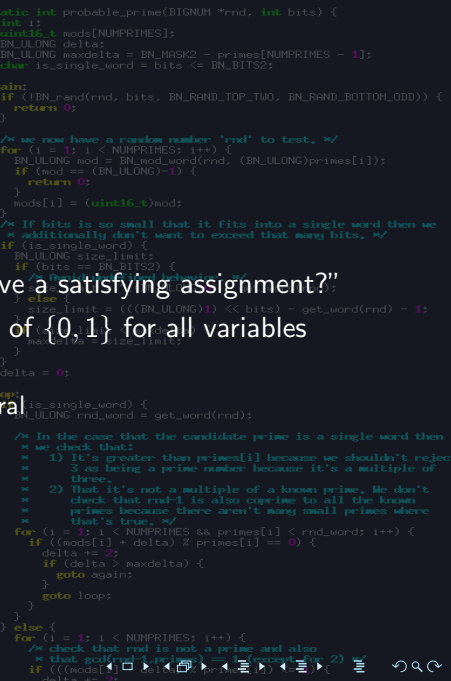
December 3, 2019

Overview

- What are SAT/SMT/Z3?
- Solving MBE lab1A with Z3
- Solving a Cyberseed RE challenge with Z3

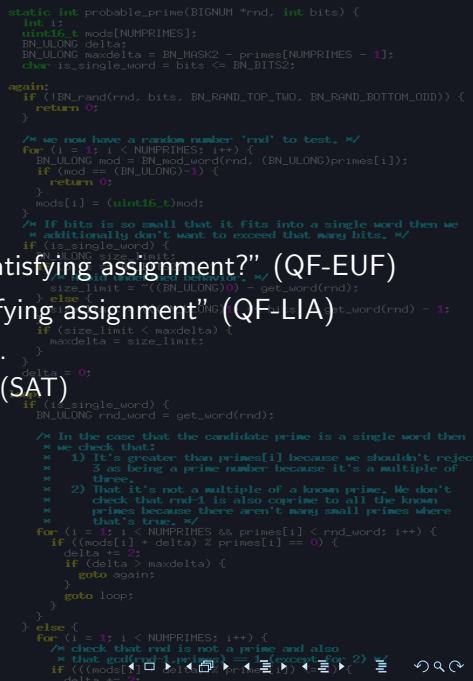
What is SAT?

- SAT is the boolean SATisfiability problem
- e.g. "Does the formula $(x \vee \neg y \vee z) \wedge (\neg x \vee y)$ have a satisfying assignment?"
- Brute forceable in $O(2^n)$ by trying all combinations of $\{0, 1\}$ for all variables
- NP-Complete
  - Not known to be subexponentially solvable in general
  - Many problems expressible as SAT



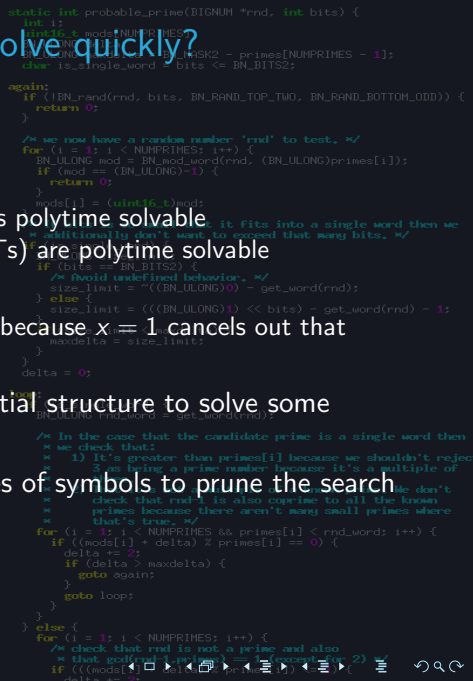
What is SMT?

- SMT is Satisfiability Modulo Theories
- "Does $(f(x,y) \vee z) \wedge (\neg g(x) = f(x,x))$ have a satisfying assignment?" (QF-EUF)
- "Does $2 * x + y \leq z \wedge x + 3 * y \geq z$ have a satisfying assignment" (QF-LIA)
- Allows more compact translation of problems, e.g.
  - $x = 1 \vee x = 2 \vee x = 3 \vee \ldots \vee x = 99 \vee x = 100$ (SAT)
  - $1 \leq x \wedge x \leq 100$ (SMT)
- Also NP-Complete



Why are SAT/SMT useful if they're hard to solve quickly?

- Not all problems are as hard as the hardest ones
  - 2-SAT (each clause having at most 2 variables) is polytime solvable
  - Monotone circuits (only ANDs and ORs, no NOTs) are polytime solvable
- It's often possible to prune the search space
  - e.g. $x \vee \varphi(a, b, c, \ldots)$ is solvable regardless of φ because $x = 1$ cancels out that subterm
- Algorithms like DPLL and CDCL make use of partial structure to solve some instances faster than others
- SMT can make use of the rules for the extra types of symbols to prune the search space at a higher level



What is Z3?

- SAT & SMT solver developed and maintained by Microsoft Research
- Libre and Open Source (MIT Licensed)
- C++, with python bindings (`pip install z3-solver`)
- Based on the CDCL algorithm

Using Z3 on small examples

- $(x \vee \neg y \vee z) \wedge (\neg x \vee y)$
- ```
  import z3
  solver = z3.Solver()
  x, y, z = z3.Bools('x y z')
  solver.add(z3.And(z3.Or(x, z3.Not(y), z), z3.Or(z3.Not(x), y)))
  if solver.check().r == 1:
      print(solver.model())
  ```
- $2 * x + y \leq z \wedge x + 3 * y \geq z \wedge z > 1$
- ```
  import z3
  solver = z3.Solver()
  x, y, z = z3.Ints('x y z')
  solver.add(2*x+y <= z)
  solver.add(x+3*y >= z)
  solver.add(z > 1)
  if solver.check().r == 1:
      print(solver.model())
  ```

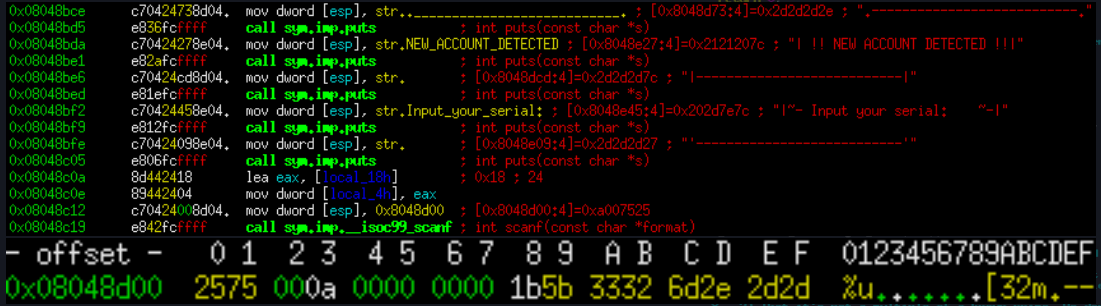
MBE Lab1A - Just Running It

```
avi@aweinstock-debian-ii:~/Documents/cloned-repos/MBE/src/lab01$ ./lab1A
.---------------------------.
|---------  RPISEC  --------|
|+ SECURE LOGIN SYS v. 3.0 +|
|---------------------------|
|~- Enter your Username:  ~-|
'---------------------------'
username
.---------------------------.
| !! NEW ACCOUNT DETECTED !!|
|---------------------------|
|~- Input your serial:    ~-|
'---------------------------'
password
avi@aweinstock-debian-ii:~/Documents/cloned-repos/MBE/src/lab01$ echo $?
1
```

MBE Lab1A - Username Entry

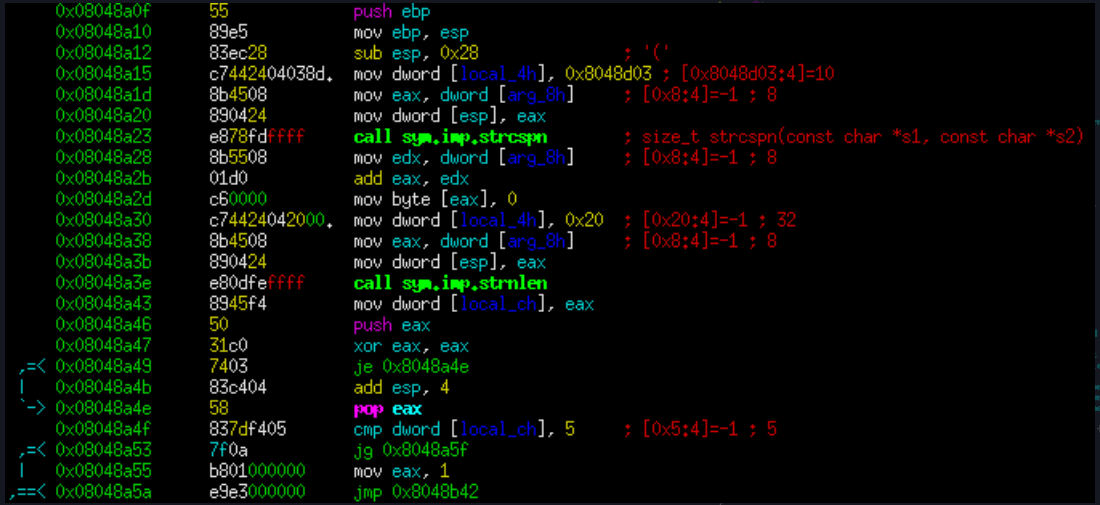
```
0x08048b69      c70424738d04.  mov dword [esp], str.+----------------------------+ ; [0x8048d73:4]=0x2d2d2d2e ; "+----------------------------+"
0x08048b70      e89bfcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048b75      c70424918d04.  mov dword [esp], str.RPISEC ; [0x8048d91:4]=0x2d2d2d7c ; "|---------  RPISEC  --------|"
0x08048b7c      e88ffcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048b81      c70424af8d04.  mov dword [esp], str.SECURE_LOGIN_SYS_v._3.0 ; [0x8048daf:4]=0x53202b7c ; "|+ SECURE LOGIN SYS v. 3.0 +|"
0x08048b88      e883fcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048b8d      c70424cd8d04.  mov dword [esp], str.       ; [0x8048dcd:4]=0x2d2d2d7c ; "|--------------------------|"
0x08048b94      e877fcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048b99      c70424eb8d04.  mov dword [esp], str.Enter_your_Username:_ ; [0x8048deb:4]=0x202d7e7c ; "|~ Enter your Username:  ~-|"
0x08048ba0      e86bfcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048ba5      c70424098e04.  mov dword [esp], str.       ; [0x8048e09:4]=0x2d2d2d27 ; "'--------------------------'"
0x08048bac      e85ffcffff     call sym.imp.puts          ; int puts(const char *s)
0x08048bb1      a160b00408     mov eax, dword [obj.stdin]  ; [0x804b060:4]=0
0x08048bb6      89442408       mov dword [local_8h], eax
0x08048bba      c74424042000.  mov dword [local_4h], 0x20  ; [0x20:4]=-1 ; 32
0x08048bc2      8d44241c       lea eax, [local_1ch]        ; 0x1c ; 28
0x08048bc6      890424         mov dword [esp], eax
0x08048bc9      e802fcffff     call sym.imp.fgets          ; char *fgets(char *s, int size, FILE *stream)
```

MBE Lab1A - Serial Entry



MBE Lab1A - Calling the authentication routine

MBE Lab1A - auth() 1/6: String processing and antidecomp



MBE Lab1A - auth() 2/6: Antidebugging with ptrace

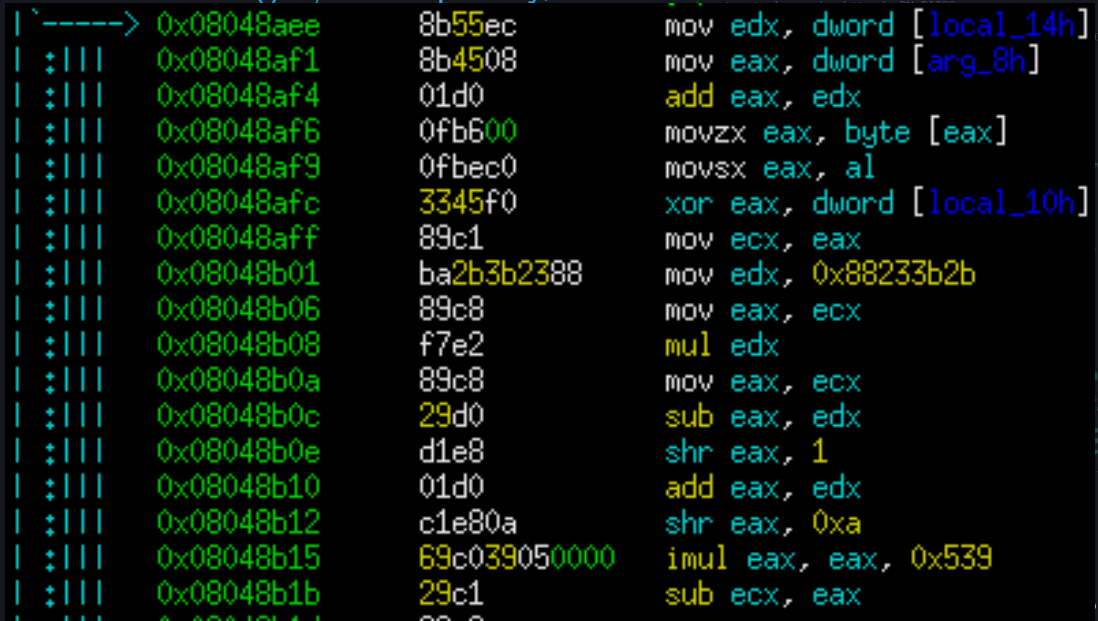
```
0x08048a5f      c744240c0000.   mov dword [local_ch_2], 0
0x08048a67      c74424080100.   mov dword [local_8h], 1
0x08048a6f      c7442404 0000.  mov dword [local_4h], 0
0x08048a77      c70424000000.   mov dword [esp], 0
0x08048a7e      e8edfdffff      call sym.imp.ptrace
0x08048a83      83f8ff          cmp eax, 0xffffffffffffffff
0x08048a86      752e            jne 0x8048ab6
0x08048a88      c70424088d04.   mov dword [esp], str.e_32m._____________________________
0x08048a8f      e87cfdffff      call sym.imp.puts           ; int puts(const char *s)
0x08048a94      c704242c8d04.   mov dword [esp], str.e_31m_____TAMPERING_DETECTED ; [0x
0x08048a9b      e870fdffff      call sym.imp.puts           ; int puts(const char *s)
0x08048aa0      c70424508d04.   mov dword [esp], str.e_32m   ; [0x8048d50:4]=0x32335b1b
0x08048aa7      e864fdffff      call sym.imp.puts           ; int puts(const char *s)
0x08048aac      b801000000      mov eax, 1
0x08048ab1      e98c000000      jmp 0x8048b42
```

MBE Lab1A - auth() 3/6: Pre-loop math

```
0x08048ab6      8b4508          mov eax, dword [arg_8h]
0x08048ab9      83c003          add eax, 3
0x08048abc      0fb600          movzx eax, byte [eax]
0x08048abf      0fbec0          movsx eax, al
0x08048ac2      3537130000      xor eax, 0x1337
0x08048ac7      05eded5e00      add eax, 0x5eeded
0x08048acc      8945f0          mov dword [local_10h], eax
```

MBE Lab1A - auth() 4/6: Loop header, restricting chars

MBE Lab1A - auth() 5/6: Loop body, much math



MBE Lab1A - auth() 6/6: Loop footer, return targets

MBE Lab1A - Z3ing auth() 1/?: Setting up variables

```
import z3
solver = z3.Solver()
wanted_length = 6
assert wanted_length > 5 # checked at 0x08048a4f
sym_username = [z3.BitVec('x{i}'.format(i=i), 8) for i in range(wanted_length)]
sym_serial = z3.BitVec('serial', 32)
```

- 8-bit entries for each character
- 32-bit serial number
- Concrete input length: `z3.Array` exists, but is more expensive
- Only use `z3.Array` if you need symbolic indexing

MBE Lab1A - Z3ing auth() 2/?: Translating the pre-loop math

```
eax = sym_username[3]
eax ^= z3.BitVecVal(0x1337, 32)
eax += z3.BitVecVal(0x5eeded, 32)
local_10h = eax
```

- We're wrapping concrete values in `z3.BitVecVal` so that wrapping/truncation happens the x86 way
- If we wre using python longs here, we'd have to manually mask them back into range

Resources

- https://github.com/Z3Prover/z3/
- https://pypi.org/project/z3-solver/
- https://rise4fun.com/Z3/tutorialcontent/guide
- https://en.wikipedia.org/wiki/Satisfiability_modulo_theories
- https://github.com/RPISEC/MBE