Kernel Sanders
CSAW ESC'2019

Grant Hernandez, Hunter Searle, Owen Flannagan, Claire Seiler, Kevin R.B. Butler

University of Florida

Dynamic & Symbolic Analysis with ANGR

- Running these challenges and being able to debug them would improve things

- We decided to use **ANGR**, a popular concolic execution engine and binary analysis framework

Solving A-lounge

```
import angr
proj = angr.Project("A/TeensyChallengeSetA.ino.elf")
st = proj.factory.blank_state()
                                                    NCONSTRAINED_MEMORY"])
```

Decompile: challenge_0_lounge - (Teens...

```
125   local_8c = 0;
126   if (b * a == 0x18af) {
127     i = 0;
128     while (i < 0x1f) {
129       challResult[i] = (&stack0xffffff54)[i];
130       i = i + 1;
131     }
132   }
133   else {
134     j = 0;
135     while (j < 0x1f) {
136       challResult[j] = -0x78;
137       j = j + 1;
138     }
139   }
```

0xc21

0xc51

```
:d_offsets)

.1challenge_06packet"]

ı_techniques.Explorer(
```

```
print_table(mgr.found[0])
```

print_card_offsets

```
WHITE_CARD_START_ADDR = 0x7fff0000-0xf
WHITE_CARD_SZ = 16*64
WHITE_CARD_END_ADDR = WHITE_CARD_START_ADDR + WHITE_CARD_SZ
BUTTON_OFFSET = WHITE_CARD_START_ADDR + \
   WHITE_CARD_SZ + 48

def print_card_offsets(state):
   expr = state.inspect.mem_read_address

   # the address could be symbolic, so get 'a' solution
   expr_val = state.solver.eval(expr)

   if expr_val >= WHITE_CARD_START_ADDR and expr_val <=
WHITE_CARD_END_ADDR:
      offset = expr_val - WHITE_CARD_START_ADDR
      print("CARD READ: %x (%s)" % (offset, str(expr))
   elif expr_val == BUTTON_OFFSET:
      print("!!!!!! BUTTON READ !!!!!!")
```

print_table

```
def print_table(state):
    table = state.solver.eval(
      state.memory.load(WHITE_CARD_START_ADDR, WHITE_CARD_SZ),
      cast_to=bytes)

    buttons = state.solver.eval(
      state.memory.load(BUTTON_OFFSET, 1), cast_to=int)
    arr = [], output = “”

    for i in range(64):
       arr += [[c for c in table[i*16:(i+1)*16]]]
    ...

    print("\n".join(output))
```

Getting Angry: Execution Troubles

- I/O (environment)
  - Calls to print, delay, etc. needed to be hooked and mocked to avoid I/O
- State Explosion
  - Some challenges had too much state to be feasible without additional constraints
- Unsat
  - Constraint solvers can't deal with cryptographic hash functions
- Slow Execution
  - ANGR lifts all basic blocks to VEX IR and executes that. This incurs >100x slow down in some cases
  - Solution: MORE CORES (used a 40-core server when needed)

How many challenges did we AutoSolve™ ?

- Set A
  - Lounge ✓
  - Closet *X* (symbolic load)
  - Café ✓
  - Stairs ✓
- Set B
  - Mobile X (state space)
  - Dance X (hash function)
  - Code X (hash function)
  - Blue X (hash function)
- Set C
  - Uno X (state space)
  - Game X (state space)
  - Break ✓
  - Recess ✓
- Set D
  - Bounce ✓
- Set E
  - Steel X (hash function)
  - Caeser X (error)
  - Spiral ✓
  - Tower X (static analysis, hash func.)
- Set F
  - Spire ✓

AutoSolve™: 8/18 (44%)

Hash Function: 5/18

Error/Timeout: 5/18

Our Results

- Set A
  - Lounge ✓
  - Closet ✓
  - Café ✓
  - Stairs ✓
- Set B
  - Mobile ✓
  - Dance ✓
  - Code ✓
  - Blue X (hash wouldn't crack!)
- *Set C*
  - Uno ✓
  - Game ✓
  - Break ✓
  - Recess ✓
- Set D
  - Bounce ✓
- Set E
  - Steel ✓
  - Caeser X (ran out of time)
  - Spiral ✓
  - Tower ✓
- Set F
  - Spire ✓

Solved: 16/18 (88.8%)
DNF: 2/18

Full Code Execution via D-Bounce

- Challenge summary: you are given a controlled stack overflow and need to redirect the saved LR to the **fillChallengeHash** function
- What about redirecting to some shellcode instead?
- Saved LR-> [0x1fff976d + 0x110] (global RFID array)

```
.section .text
.align 2
.syntax unified
adr r7, serial_putchar
ldrh r7, [r7]
adr r6, hacked

looper:
  ldrb r0, [r6]
  blx r7
  ldrb r0, [r6, #1]
  blx r7
b looper
```

```
serial_putchar:
.word 0x3dad

hacked:
.ascii "HACKED\n"
```

Conclusion

- Work smart, not hard
  - Static analysis is expensive. Dynamic analysis gets straight to the point
- Firmware without the hardware is just software
  - Emulate only what you need and mock away everything else
- Symbolic execution works great on smaller problems
  - Domain knowledge can alleviate state explosion, but this requires static analysis
- Firmware exploitation is like going back to the 90's
  - Processors powering many embedded devices don't support modern mitigations (or they are turned off)

Thank you!

Grant Hernandez
@digital_cold- https://hernan.de/z

Hunter Searle