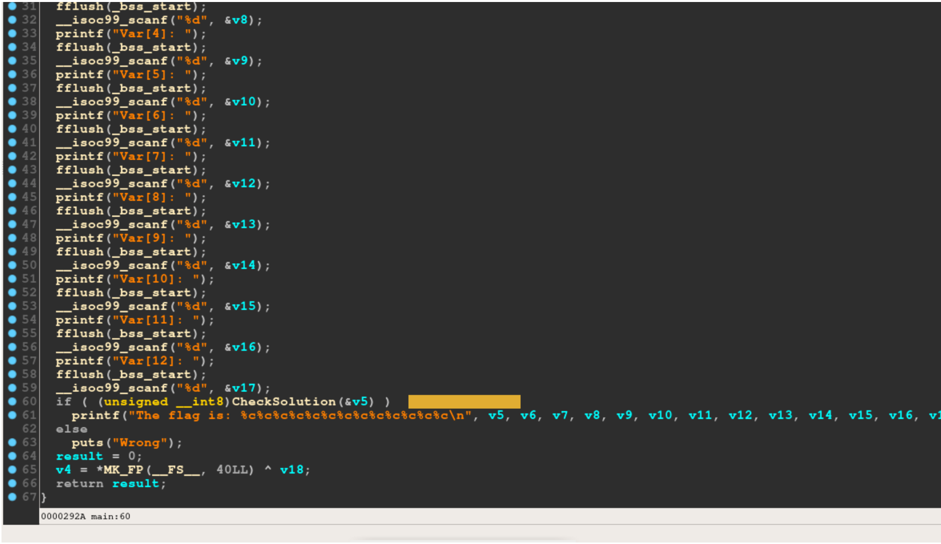
Session 0x0E
Advanced Topics: Intro to Symbolic Execution

Security Summer School

ACS/Ixia/Hexcellents

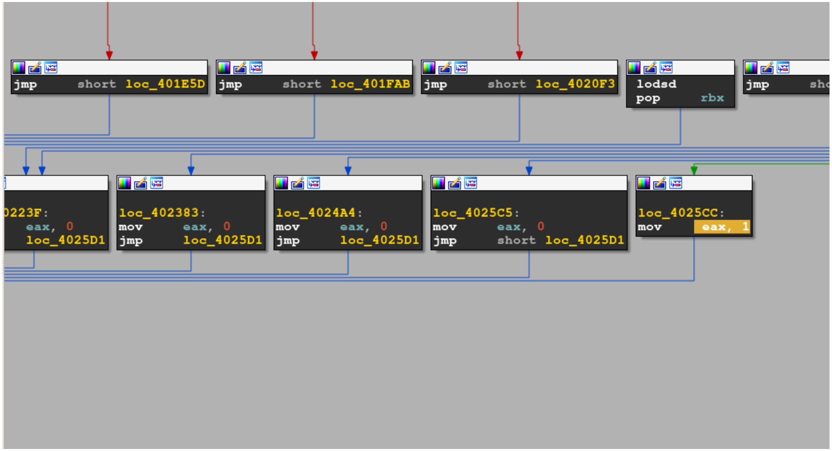
Motivation



```
  fflush(_bss_start);
  __isoc99_scanf("%d", &v8);
  printf("Var[4]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v9);
  printf("Var[5]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v10);
  printf("Var[6]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v11);
  printf("Var[7]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v12);
  printf("Var[8]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v13);
  printf("Var[9]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v14);
  printf("Var[10]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v15);
  printf("Var[11]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v16);
  printf("Var[12]: ");
  fflush(_bss_start);
  __isoc99_scanf("%d", &v17);
  if ( (unsigned __int8)CheckSolution(&v5) )  [redacted]
    printf("The flag is: %c%c%c%c%c%c%c%c%c%c%c%c%c\n", v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v1
  else
    puts("Wrong");
  result = 0;
  v4 = *MK_FP(__FS__, 40LL) ^ v18;
  return result;
}
```

0000292A main:60

Motivation



```
jmp     short loc_401E5D
```

```
jmp     short loc_401FAB
```

```
jmp     short loc_4020F3
```

```
lodsd
pop     rbx
```

```
jmp     sho
```

```
0223F:
        eax, 0
        loc_4025D1
```

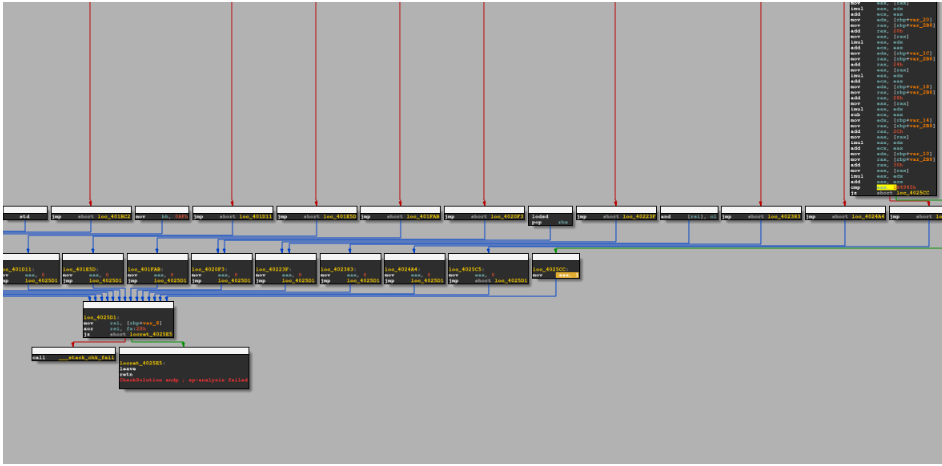
```
loc_402383:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_4024A4:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_4025C5:
mov     eax, 0
jmp     short loc_4025D1
```

```
loc_4025CC:
mov     eax, 1
```

Motivation



```
mov     eax, [rax]
imul    eax, edx
add     ecx, eax
mov     edx, [rbp+var_20]
mov     rax, [rbp+var_2B8]
add     rax, 20h
mov     eax, [rax]
imul    eax, edx
add     ecx, eax
mov     edx, [rbp+var_1C]
mov     rax, [rbp+var_2B8]
add     rax, 24h
mov     eax, [rax]
imul    eax, edx
add     ecx, eax
mov     edx, [rbp+var_18]
mov     rax, [rbp+var_2B8]
add     rax, 28h
mov     eax, [rax]
imul    eax, edx
sub     ecx, eax
mov     edx, [rbp+var_14]
mov     rax, [rbp+var_2B8]
add     rax, 2Ch
mov     eax, [rax]
imul    eax, edx
add     ecx, eax
mov     edx, [rbp+var_10]
mov     rax, [rbp+var_2B8]
add     rax, 30h
mov     eax, [rax]
imul    eax, edx
add     eax, ecx
cmp     eax, 1B4945h
jz      short loc_4025CC
```

```
std
```

```
jmp     short loc_401BC2
```

```
mov     bh, 0AFh
```

```
jmp     short loc_401D11
```

```
jmp     short loc_401E5D
```

```
jmp     short loc_401FAB
```

```
jmp     short loc_4020F3
```

```
lodsd
pop     rbx
```

```
jmp     short loc_40223F
```

```
and     [rsi], cl
```

```
jmp     short loc_402383
```

```
jmp     short loc_4024A4
```

```
jmp     short lo
```

```
oc_401D11:
ov      eax, 0
mp      loc_4025D1
```

```
loc_401E5D:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_401FAB:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_4020F3:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_40223F:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_402383:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_4024A4:
mov     eax, 0
jmp     loc_4025D1
```

```
loc_4025C5:
mov     eax, 0
jmp     short loc_4025D1
```

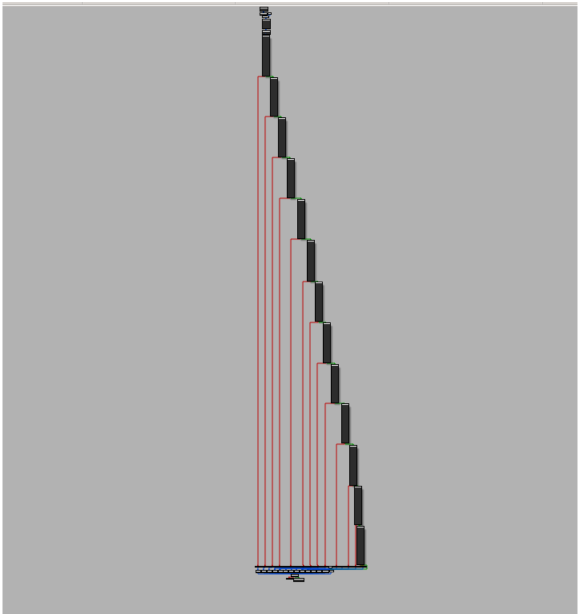
```
loc_4025CC:
mov     eax, 1
```

```
loc_4025D1:
mov     rsi, [rbp+var_8]
xor     rsi, fs:28h
jz      short locret_4025E5
```

```
call    ___stack_chk_fail
```

```
locret_4025E5:
leave
retn
CheckSolution endp ; sp-analysis failed
```

Motivation



Motivation

- How fast do you think you can solve this challenge?
- Can you solve it at all?

Motivation

- How fast do you think you can solve this challenge?
- Can you solve it at all?
- How about throwing some CPU power at it?
- Objective: creating input to reach deep inside CFGs

Intro

- In practice, vulnerability triggering requires chains of input
- Targetting long code paths and corner cases
- Manually crafting input becomes tedious
- What can be automated and what cannot?

Main idea

```
int a = α, b = β, c = γ;
                    // symbolic
int x = 0, y = 0, z = 0;
if (a) {
  x = -2;
}
if (b < 5) {
  if (!a && c)  { y = 1; }
  z = 2;
}
assert(x+y+z!=3)
```

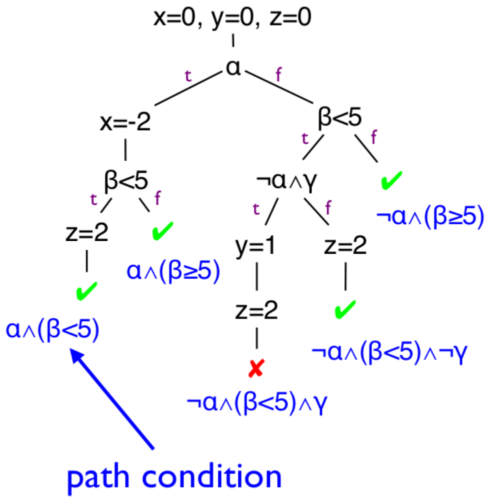


image source: www.cs.umd.edu

Main idea

- Execution is split into multiple code runs (for each code branching)
- This may lead to path explosion
- It's better if you know exactly where you want the execution to go
- Behind the scenes, it uses Constraint Solvers (SAT-SMT solvers)

References

- DEF CON 23 - Shoshitaishvili and Wang - Angry Hacking: The next gen of binary analysis `https://www.youtube.com/watch?v=oznsT-ptAbk`
- `https://github.com/angr/angr-doc/tree/master/examples`