符号执行及其在CTF中的应用

解培岱

国防科技大学计算机学院

2019年12月28号

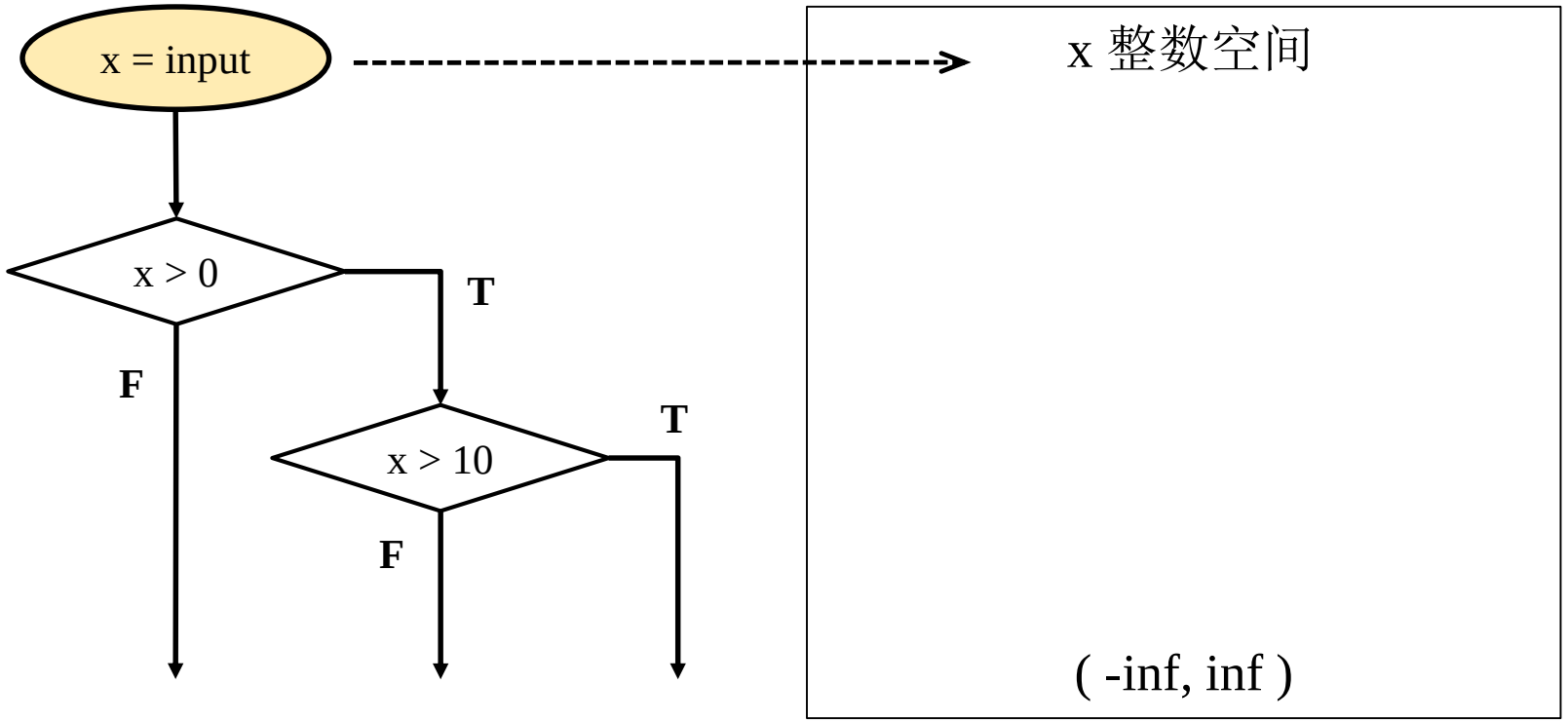
代码执行

- 一个例子
  - 实际执行

```
int main (){
    int x;

    read(0, &x, 4);

    if (x > 0){
        if (x < 10){
            //Error
        }
        else{
            printf(“ok”);
        }
    }

    return 0;
}
```

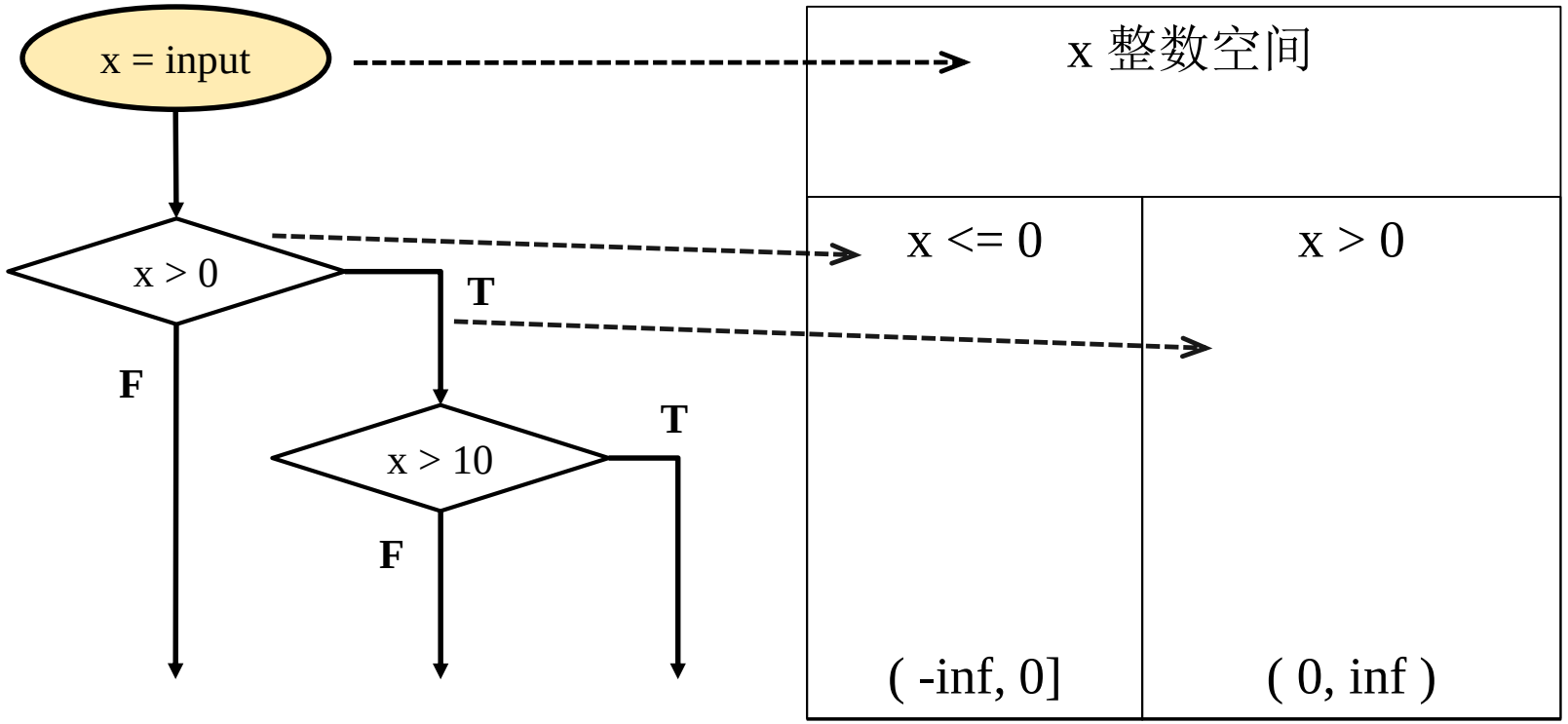
符号执行

- 输入变量的取值空间对应程序路径



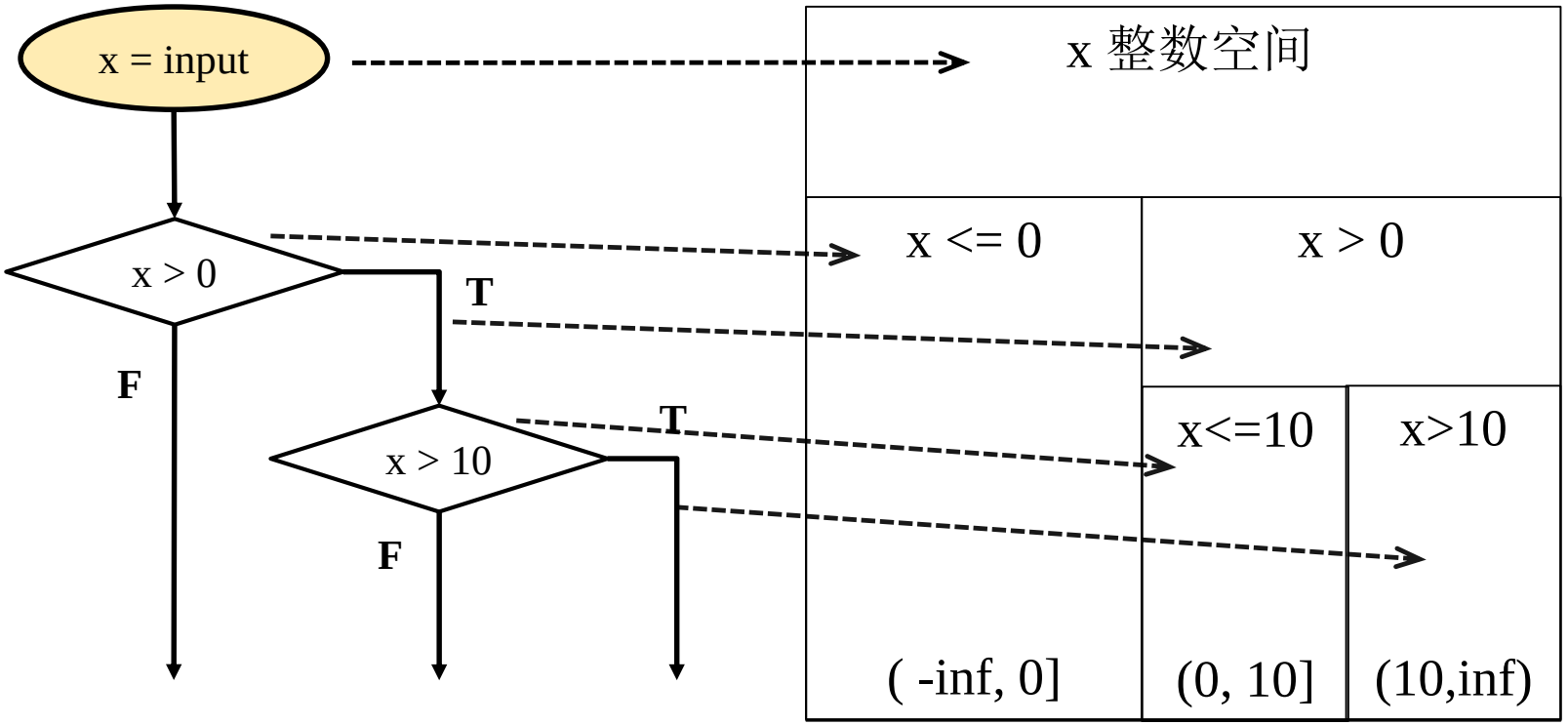
符号执行

- 输入变量的取值空间对应程序路径



符号执行

- 输入变量的取值空间对应程序路径



符号执行

- 提出
  - Robert S. Boyer, ICRS, page 234-245, 1975
    - SELECT - a formal system for testing and debugging programs by symbolic execution
  - James C. King, CACM, page 385-394, 1976
    - Symbolic execution and program testing
  - …

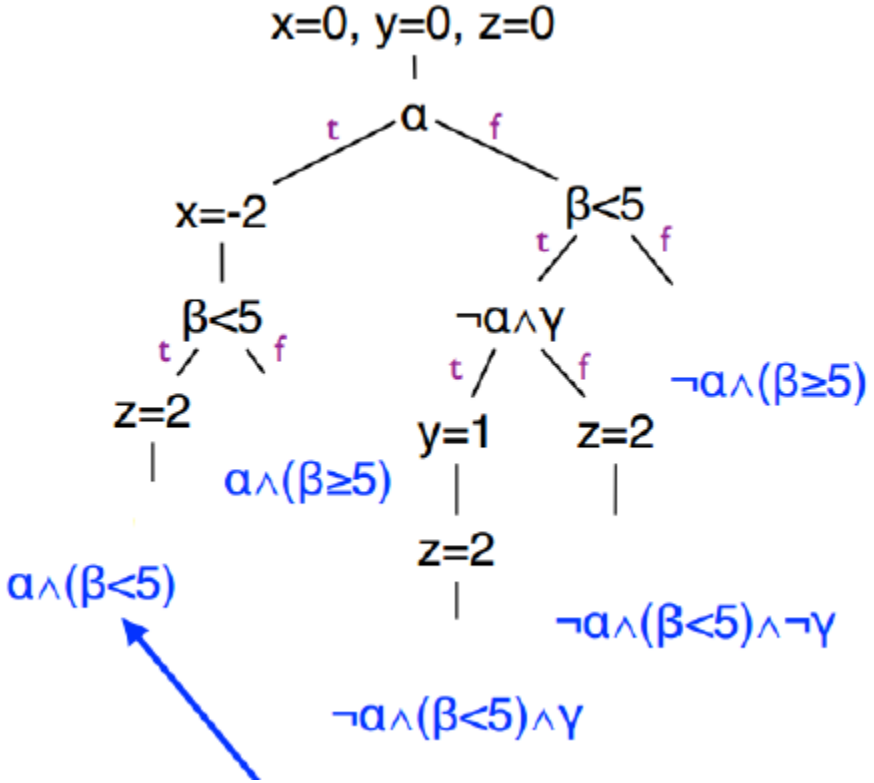
符号执行

- 使用符号变量执行程序
  - 变量不再是一个具体的实际值，而是一个符号
- 同时执行多条路径
  - 在路径分支处fork一个新进程
  - 使用路径约束表示一条路径

路径约束

```
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)
```

执行树



路径约束
可使用SMT求解器进行求解

路径约束求解器

- SMT/SAT求解器
  - SAT = Satisfiability
  - SMT = Satisfiability modulo theory = SAT++
  - 求解器实现：z3, Yices, STP
- Hoare逻辑
  - 结构

assertions	$P, Q \in \mathbf{Assn}$	$P ::= \mathbf{true} \mid \mathbf{false} \mid a_1 < a_2 \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \mid P_1 \Rightarrow P_2 \mid \neg P \mid \forall i.\ P \mid \exists i.\ P$
arithmetic expressions	$a \in \mathbf{Aexp}$	$a ::= \ldots$
logical variables	$i, j \in \mathbf{LVar}$	

  - 规则

$$\textsc{Skip}\ \frac{}{\{P\}\ \mathbf{skip}\ \{P\}} \qquad \textsc{Assign}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

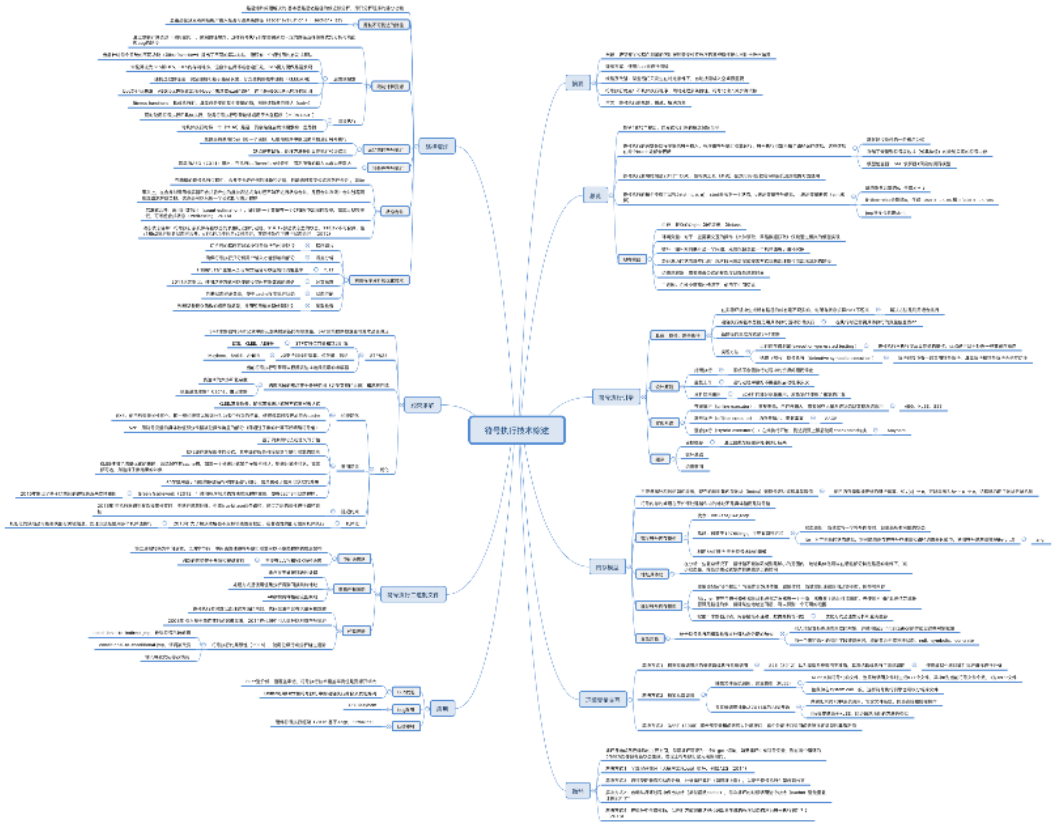
$$\textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}} \qquad \textsc{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \{Q\}}$$

$$\textsc{Consequence}\ \frac{\vDash (P \Rightarrow P') \qquad \{P'\}\ c\ \{Q'\} \qquad \vDash (Q' \Rightarrow Q)}{\{P\}\ c\ \{Q\}}$$

$$\textsc{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{P \wedge \neg b\}}$$

符号执行

- 发展
  - Roberto Baldoni, ACM Computing Surveys, No 50, 2018
    - A Survey of Symbolic Execution Techniques
    - 符号执行引擎、内存模型、循环、路径爆炸、约束求解



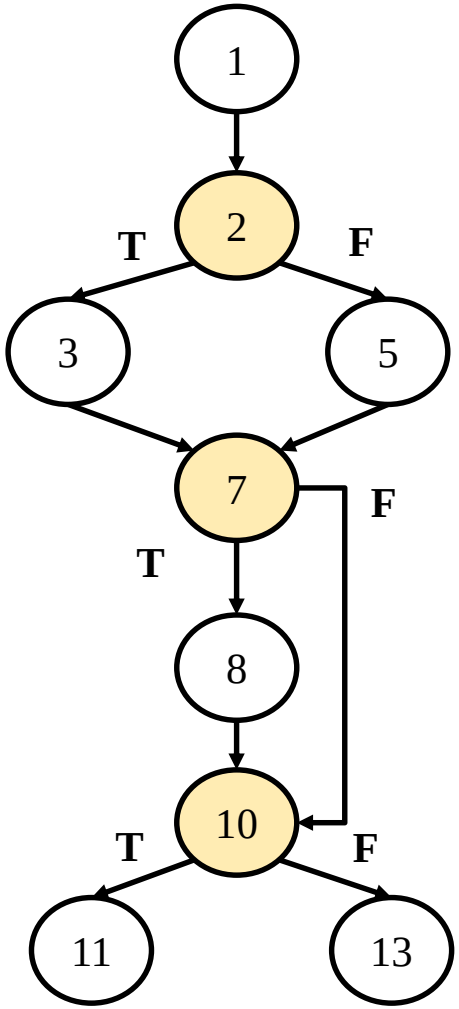
符号执行的一般实现

- 使用符号变量代替具体的输入变量
- 以程序代码的基本块为执行单位
- 在符号执行引擎中执行，记录符号状态
  - 在每个基本块执行结束时，更新符号状态，包括：
    - 程序在当前基本块的变量赋值，包括符号变量和本地变量
    - 能够执行到达当前位置的路径条件表达式，即路径约束
  - 当遭遇分支判断时，为真和为假的两个分支都被执行
    - 对于真分支，将判断条件加入路径约束
    - 对于假分支，将判断条件取反后加入路径约束
  - 若某分支路径不可达，则当前路径执行终止

符号执行

- 实例

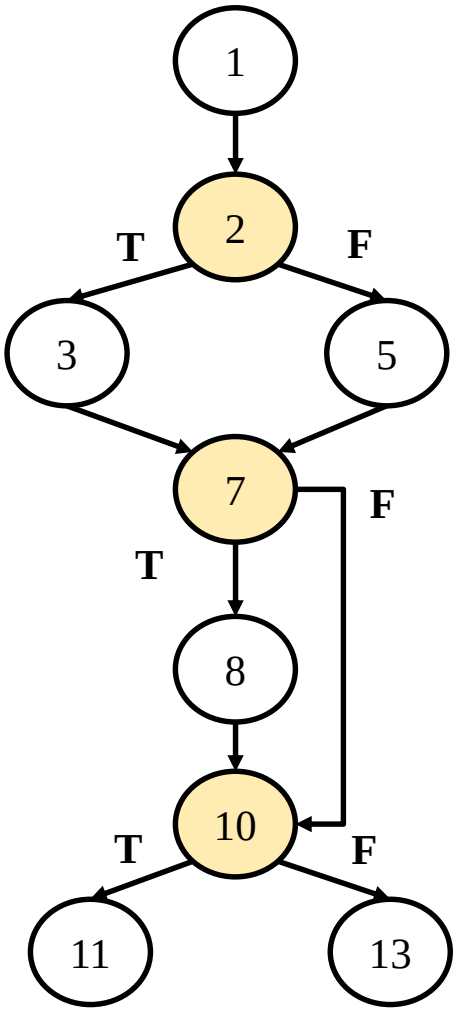
```
Function foo (int x, int y):
1.  Read x, y
2.  if x > 0:
3.      y = 2 * x
4.  else:
5.      y = x
6.  endif
7.  if y ≥ 0:
8.      y = y + 1
9.  endif
10. if x * y > 0:
11.     return x
12. else:
13.     return y
```



符号执行

- 输入变量: x, y
  - 符号名字: X, Y
- 路径共8条

1, 2, 3, 7, 8, 10, 11
1, 2, 5, 7, 8, 10, 11
1, 2, 3, 7, 10, 11
1, 2, 5, 7, 10, 11
1, 2, 3, 7, 8, 10, 13
1, 2, 5, 7, 8, 10, 13
1, 2, 3, 7, 10, 13
1, 2, 5, 7, 10, 13



执行路径：1, [2, 3,] [7, 8,] [10, 11]

赋值：x=X,y=Y
路径表达式：True

赋值：y=2*X
路径表达式：True ∧ X>0

赋值：y=2*X+1
路径表达式：True ∧ X>0 ∧ 2*X>=0

赋值：*无*
路径表达式：True ∧ X>0 ∧ 2*X>=0 ∧ X*(2*X+1)>0

```
Function foo (int x, int y):
1.  Read x, y
2.  if x > 0:
3.      y = 2 * x
4.  else:
5.      y = x
6.  endif
7.  if y ≥ 0:
8.      y = y + 1
9.  endif
10. if x * y > 0:
11.     return x
12. else:
13.     return y
```

执行路径：1, [2, 3,] [7, 8,] [10, 11]

行号	赋值	路径约束（真值表达式）
1	x ← X, y ← Y	True
2, 3	y ← 2 * X	True ∧ X>0
7, 8	y ← 2*X + 1	True ∧ X>0 ∧ 2*X>=0
10, 11		True ∧ X>0 ∧ 2*X>=0 ∧ X * (2*X+1) > 0

使用求解器对路径约束求解。
求解可得：X=1，Y=1，路径可达。

执行路径：1, [2, 3,] 7, [10, 13]

赋值：x=X,y=Y
路径表达式：True

赋值：y=2*X
路径表达式：True ∧ X>0

赋值：*无*
路径表达式：True ∧ X>0 ∧ 2*X<0

赋值：*无*
路径表达式：True ∧ X>0 ∧ 2*X<0 ∧ X*(2*X)<=0

```
Function foo (int x, int y):
1.  Read x, y
2.  if x > 0:
3.      y = 2 * x
4.  else:
5.      y = x
6.  endif
7.  if y ≥ 0:
8.      y = y + 1
9.  endif
10. if x * y > 0:
11.     return x
12. else:
13.     return y
```

执行路径：1, [2, 3,] 7, [10, 13]

行号	赋值	路径约束（真值表达式）
1	x ← X, y ← Y	True
2, 3	y ← 2 * X	True ∧ X > 0
7	无	True ∧ X > 0 ∧ 2*X < 0
10, 13	无	True ∧ X > 0 ∧ 2*X < 0 ∧ X * (2 * X) <= 0

使用求解器对路径约束求解。
求解无解，路径不可达。

符号执行

- 可见，当执行某一条路径时，实际上可实现对多个实际输入值的执行的效果。

符号执行

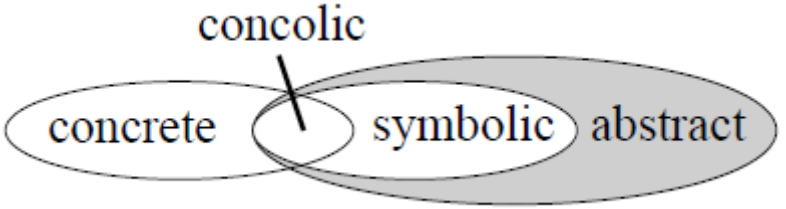
- 经典符号执行
  - 执行路径
    - 真值表达式true和false的序列seq={p0,p1,…,pn}
  - 执行树
    - 表征所有执行路径
  - 符号状态
    - 符号变量/符号内存，符号表达式，及其映射关系
  - 符号路径约束
    - 执行路径的符号表达式

符号执行的挑战

- 路径爆炸
  - 路径表达式项是分支数的2的幂次方
  - 符号化的控制变量
    - 如循环控制变量是符号变量
  - 解决方法
    - 裁剪不可达路径、函数和循环摘要、路径归并与等价、约束下的符号执行、状态合并、预置条件与输入特性等
- 约束求解
  - 路径约束表达式求解是NP-Complete问题，是主要挑战
  - 约束求解优化是重要技术研究方向，近年来取得进展
  - 解决方法
    - 限制约束、重用约束的解、懒惰约束、处理不确定的约束等
- 其他
  - 符号执行引擎、内存建模

符号执行的发展

- 动态符号执行Concolic Execution
  - 符号执行(Symbolic)与实际执行(Concrete)相结合
  - 两种状态
    - 精确状态、符号状态
  - 执行方法
    - 随机产生输入数据
    - 启动具体执行，记录执行状态，同时收集路径约束
    - 在分支点将路径约束的分支路径条件取反，求解出新的输入数据
    - 当前执行继续直到到达执行终止条件
    - 从记录的输入数据集合中挑选出一个新输入，启动下一次执行



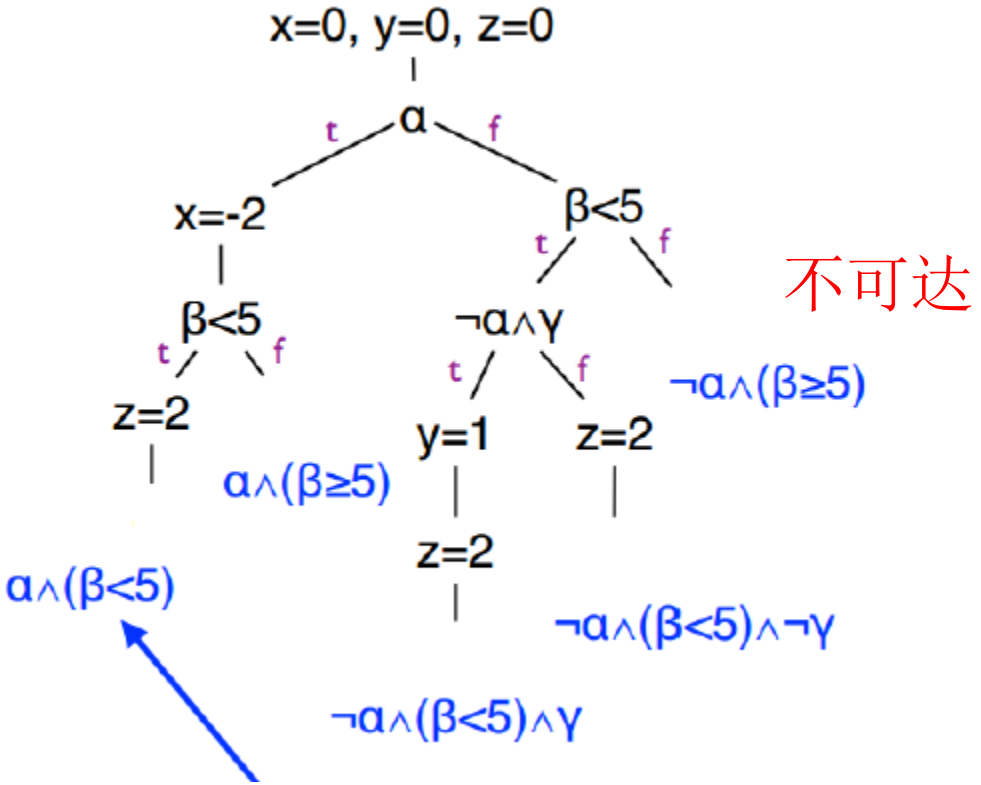
符号执行的应用

- 基本目标是通过遍历路径实现对程序语义的理解
- 基础应用
  - 判断路径的可达性
  - 生成程序的测试用例
  - 检测程序安全缺陷/漏洞

符号执行的应用

- 判断路径的可达性
  - 假设增加约束条件 α = β

```
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)
```

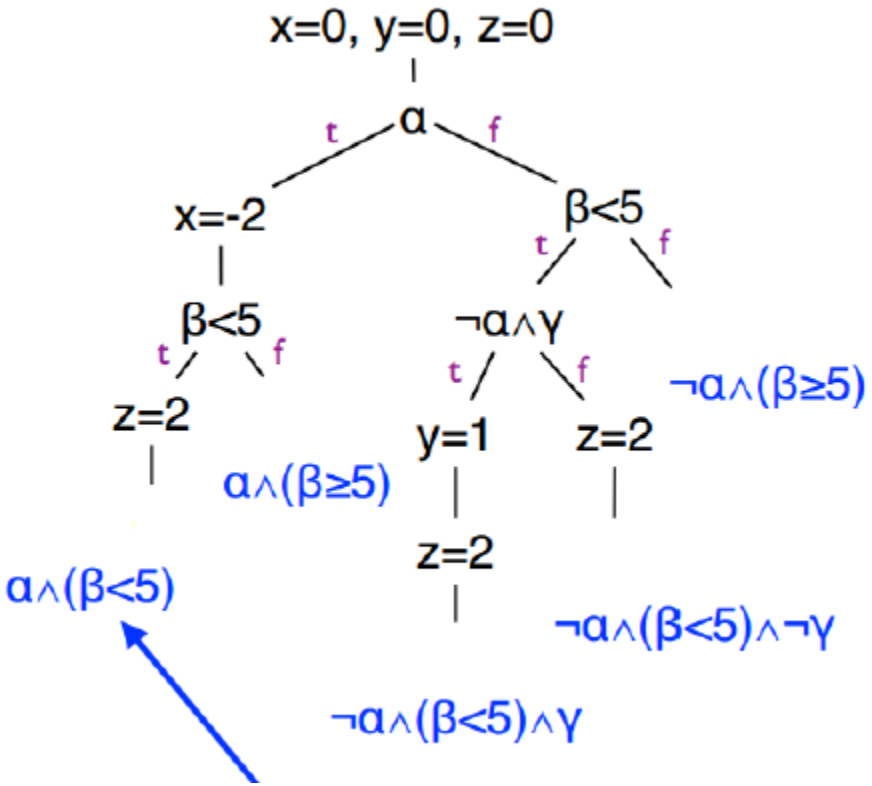


路径约束

符号执行的应用

- 生成程序的测试用例遍历路径
  - 求解各路径的路径约束

```
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)
```



路径1：α = 1；β = 1
路径2：α = 1；β = 6
路径3：…

符号执行的应用

- 检测程序安全缺陷/漏洞

```
int foo (int i){
    int j = 2*i;

    i = i++;
    i = i*j;
    if(i < 1)
        i = -i;
    i = j/i;

    return i;
}
```

i: 符号变量I

真分支：

$2*I \wedge 2 + i*I < 0$

$i = -2*I \wedge 2 - i*I$

$i == 0$

I求得i=-1时触发漏洞

假分支：

$2*I \wedge 2 + i*I >= 0$

$i = 2*I \wedge 2 + i*I$

$i == 0$

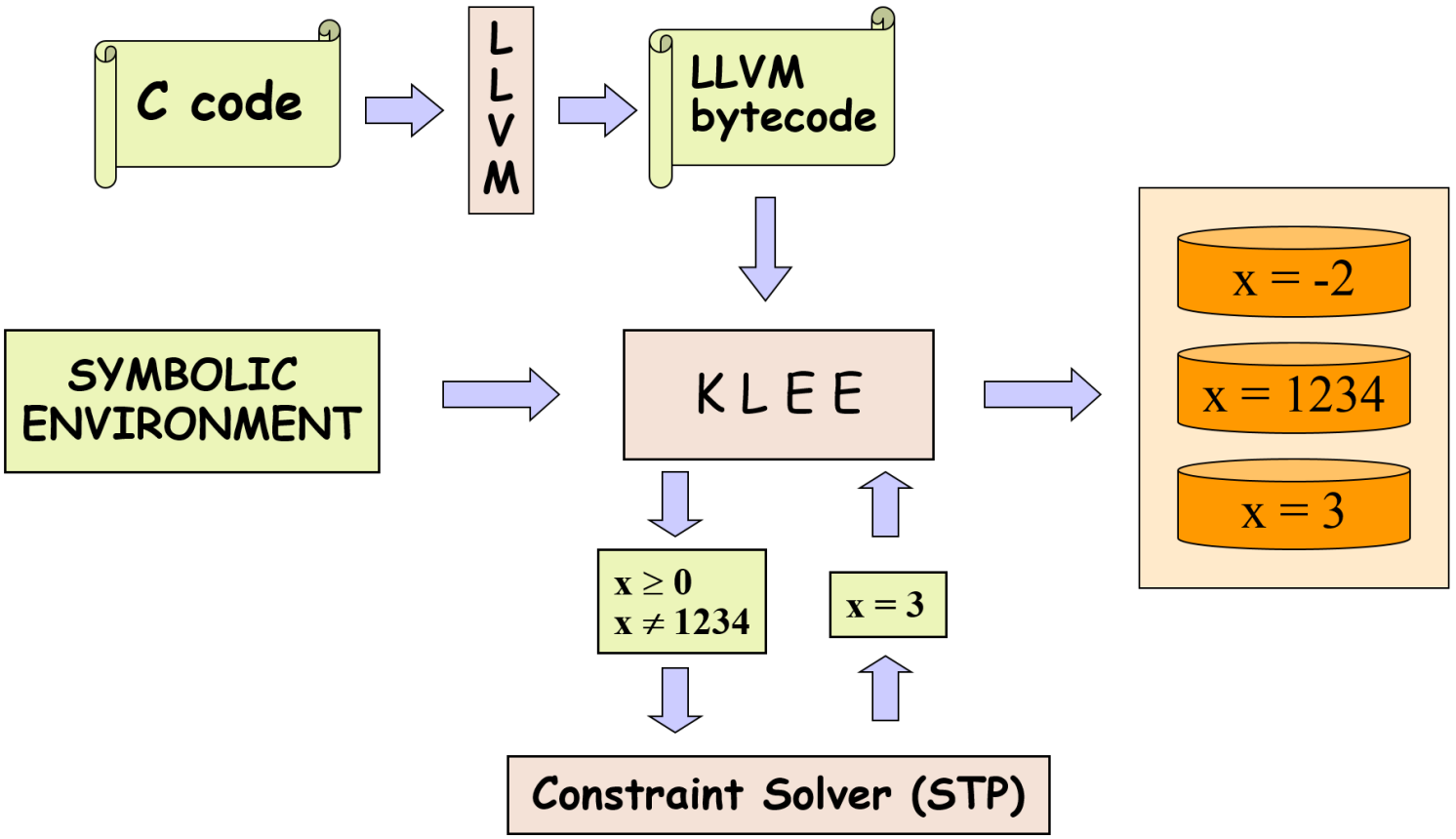
不会触发漏洞

符号执行的应用

- 具体应用
  - 可达路径的输入数据求解
  - 混淆代码的对抗/语义无效代码消除/...
  - 路径覆盖
  - 多版本函数语义的对比
  - 功能代码片段的语义背离检测
  - 基于测试用例生成的漏洞挖掘
  - 恶意代码分析
  - 基于崩溃路径的漏洞利用代码自动生成 / AEG
  - 自动漏洞挖掘/利用、自动攻防、智能攻防

符号执行平台

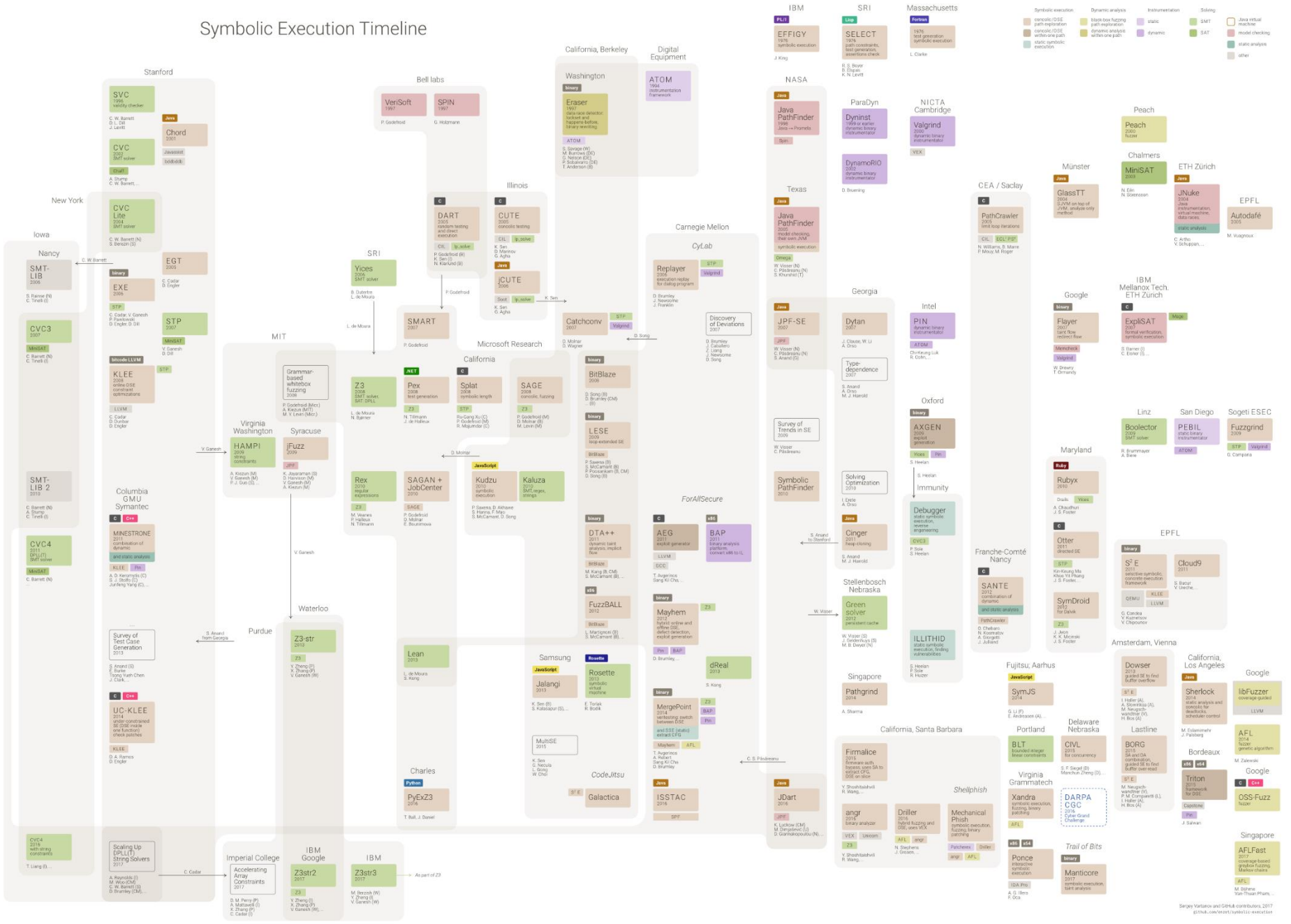
- KLEE
  - 基于LLVM的符号执行引擎



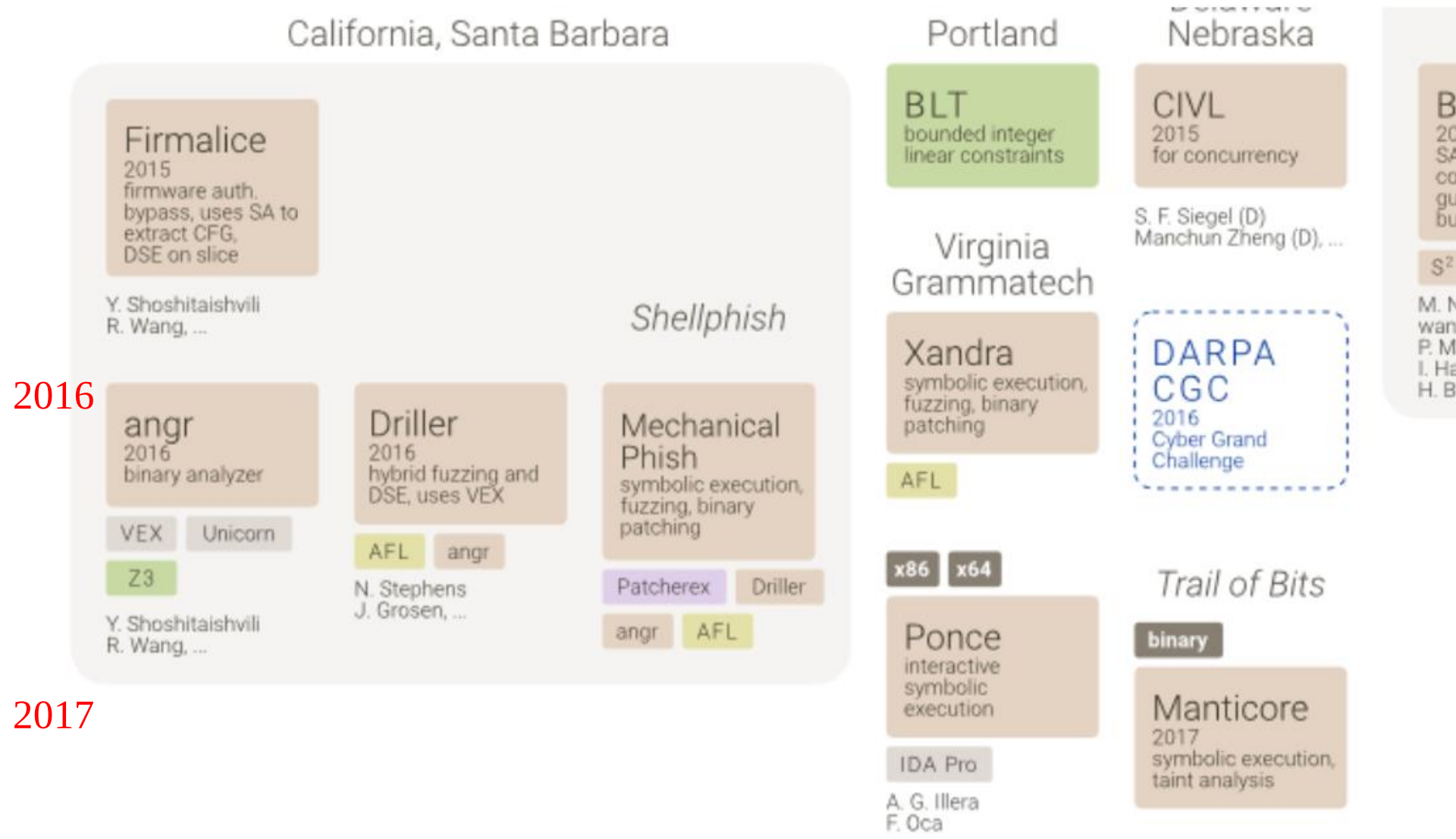
符号执行平台

- 源码级
  - jCUTE
  - Cloud9
  - Kite
  - PyExZ3
  - SymDroid
  - SymJS
  - …
- 二进制代码级
  - Mayhem
  - SAGE
  - DART
  - BitBlaze
  - PathGrind
  - FuzzBALL
  - S2E
  - BAP
  - Triton
  - Angr
  - Driller
  - miasm

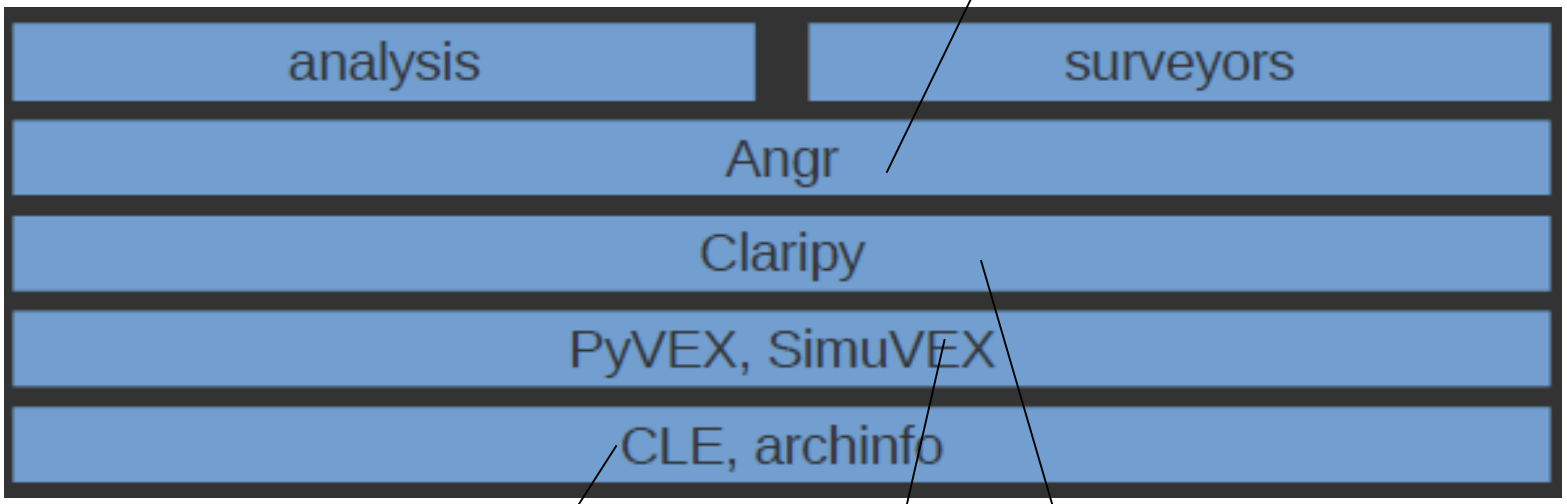
Symbolic Execution Timeline



符号执行平台


California, Santa Barbara
Firmalice
2015
firmware auth. bypass, uses SA to extract CFG, DSE on slice
Y. Shoshitaishvili
R. Wang, ...
Shellphish
2016
angr
2016
binary analyzer
VEX
Unicorn
Z3
Y. Shoshitaishvili
R. Wang, ...
Driller
2016
hybrid fuzzing and DSE, uses VEX
AFL
angr
N. Stephens
J. Grosen, ...
Mechanical Phish
symbolic execution, fuzzing, binary patching
Patcherex
Driller
angr
AFL
2017
Portland
BLT
bounded integer linear constraints
Virginia Grammatech
Xandra
symbolic execution, fuzzing, binary patching
AFL
x86
x64
Ponce
interactive symbolic execution
IDA Pro
A. G. Illera
F. Oca
Nebraska
CIVL
2015
for concurrency
S. F. Siegel (D)
Manchun Zheng (D), ...
DARPA CGC
2016
Cyber Grand Challenge
Trail of Bits
binary
Manticore
2017
symbolic execution, taint analysis

Angr



应用接口

二进制文件加载、分析等

符号执行引擎前端

中间语言，基于Vagrant的VEX

Angr

- Project
  - 工作空间
  - loader
    - proj.loader
  - factory
    - Blocks
      - Angr以基本块为单位进行分析
    - States
      - 模拟执行的程序状态SimState
      - 一个状态包括了程序内存、寄存器、文件系统、数据等
        - state.regs、state.mem
      - 与数值类型不同，state中的数据使用位向量表示*bitvectors*
        - state.mem[proj.entry].int.resolved
    - Simulation Managers
    - Analyses

```
>>> import angr
>>> proj = angr.Project('/bin/true')
```

HackCon 2016 angry-reverser

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char v4; // [rsp+0h] [rbp-20h]
4
5    __isoc99_scanf("%s", &v4, envp);
6    if ( ptrace(0, 0LL, 1LL, 0LL) < 0 )
7    {
8      puts("NOPE!");
9      exit(1);
10   }
11   if ( (unsigned __int8)GoHomeOrGoCrazy(&v4) )
12     printf("YAYY : %s\n", &v4);
13   else
14     puts("NOPE");
15   return 0;
16 }
```

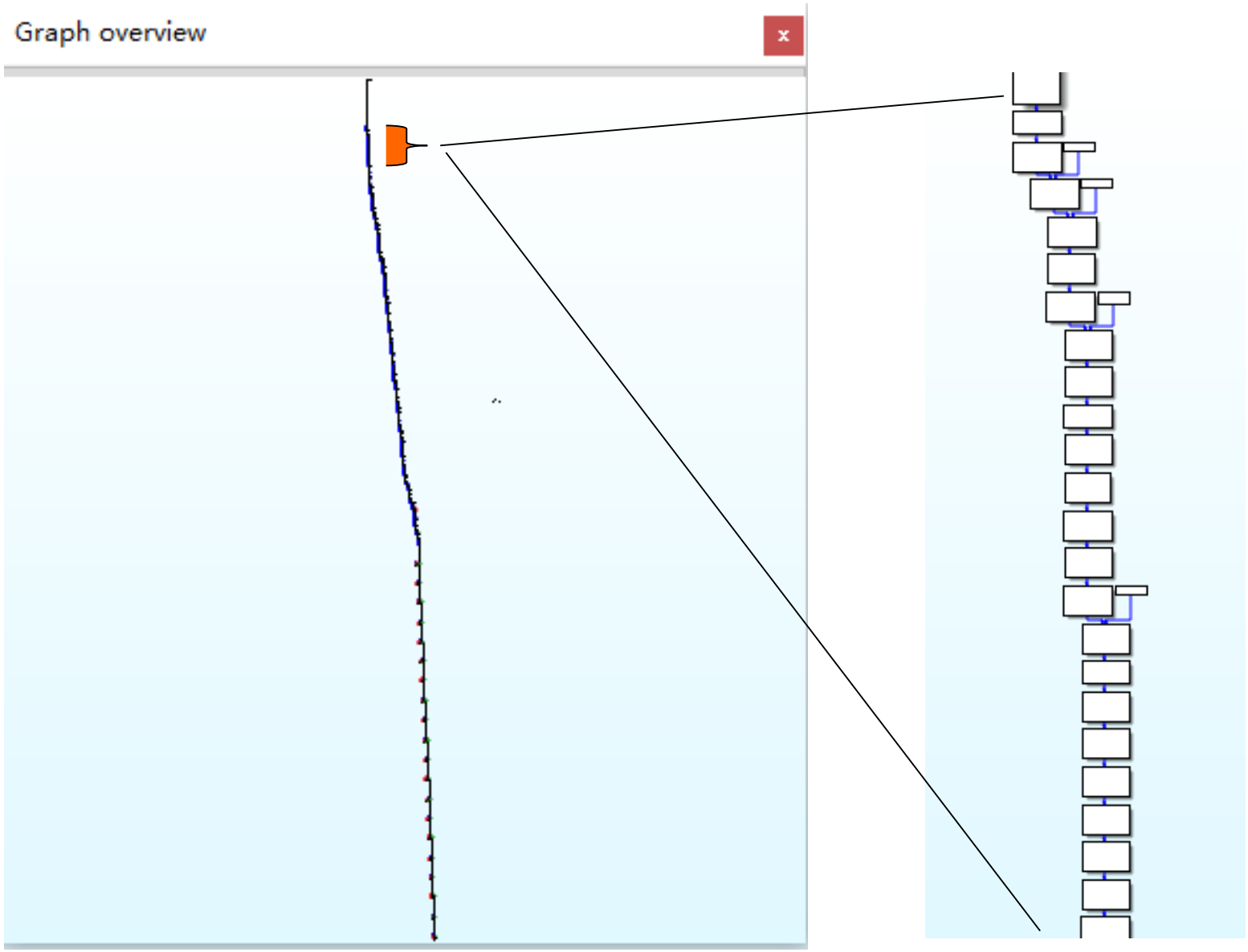
HackCon 2016 angry-reverser

```
.text:0000000000400646 var_C           = dword ptr -0Ch
.text:0000000000400646 var_8           = dword ptr -8
.text:0000000000400646 var_4           = dword ptr -4
.text:0000000000400646
.text:0000000000400646 ; __unwind {
.text:0000000000400646                 push    rbp
.text:0000000000400647                 mov     rbp, rsp
.text:000000000040064A                 sub     rsp, 650h
.text:0000000000400651                 mov     [rbp+var_648], rdi
.text:0000000000400658                 jmp     short loc_40065C
.text:0000000000400658 ; ---------------------------------------------------------------------------
.text:000000000040065A                 dw 98B9h
.text:000000000040065C ; ---------------------------------------------------------------------------
.text:000000000040065C
.text:000000000040065C loc_40065C:                             ; CODE XREF: GoHomeOrGoCrazy+12↑j
.text:0000000000405A27                 jmp     short loc_405A2B
.text:0000000000405A29 ; ---------------------------------------------------------------------------
.text:0000000000405A29                 fiadd   dword ptr [rbx]
.text:0000000000405A2B
.text:0000000000405A2B loc_405A2B:                             ; CODE XREF: GoHomeOrGoCrazy+53E1↑j
.text:0000000000405A2B                 mov     ecx, 0
.text:0000000000405A30                 mov     edx, 1
.text:0000000000405A35                 mov     esi, 0
.text:0000000000405A3A                 mov     edi, 0          ; request
.text:0000000000405A3F                 mov     eax, 0
.text:0000000000405A44                 call    _ptrace
.text:0000000000405A49                 test    rax, rax
.text:0000000000405A4C                 jns     short loc_405A62
.text:0000000000405A4E                 mov     edi, offset s   ; "NOPE!"
.text:0000000000405A53                 call    _puts
.text:0000000000405A58                 mov     edi, 1          ; status
.text:0000000000405A5D                 call    _exit
.text:0000000000405A62 ; ---------------------------------------------------------------------------
.text:0000000000405A62
.text:0000000000405A62 loc_405A62:                             ; CODE XREF: GoHomeOrGoCrazy+5406↑j
.text:0000000000405A62                 jmp     short loc_405A66
```

控制结构混淆

ptrace检测

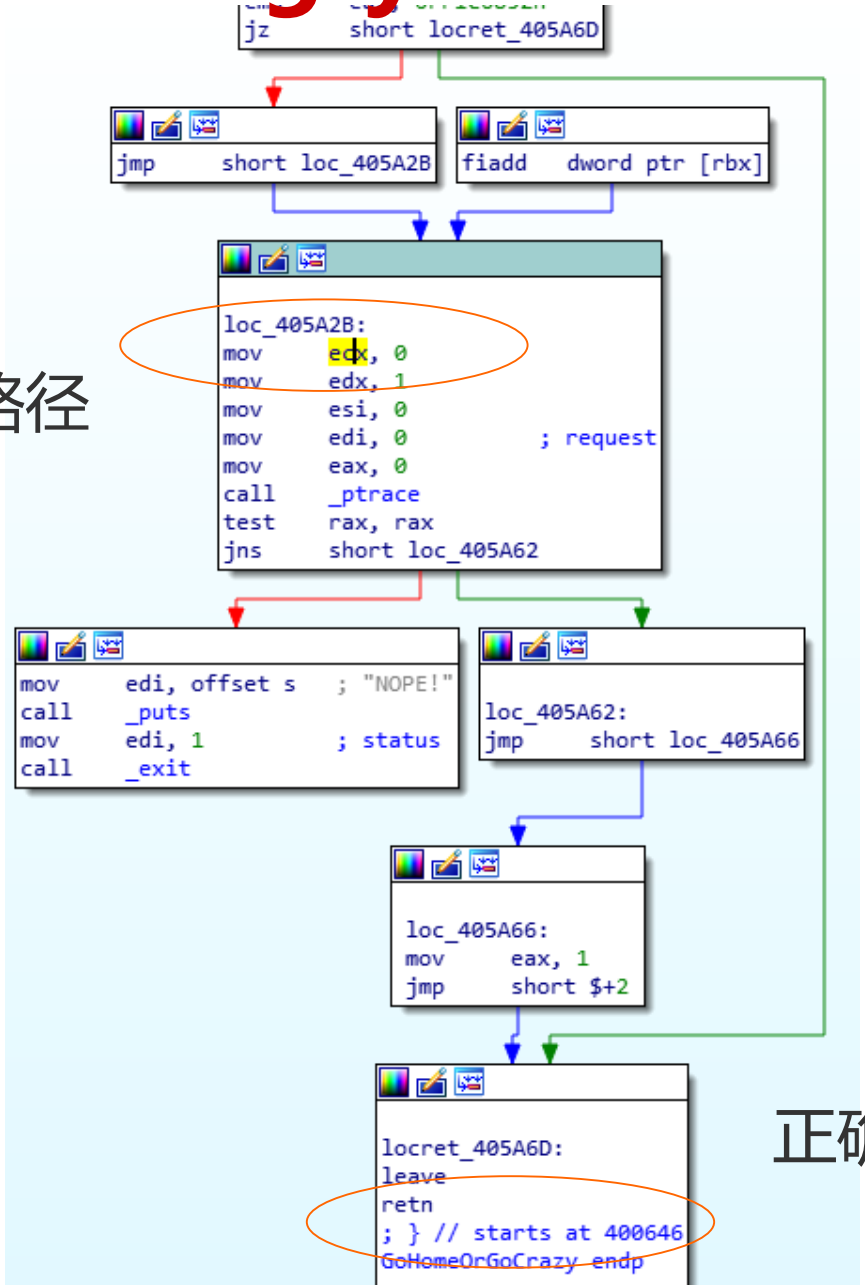
HackCon 2016 angry-reverser



HackCon 2016 angry-reverser

- 找出正确路径

规避路径



正确路径

HackCon 2016 angry-reverser

```python
import angr
import claripy

def main():
    flag    = claripy.BVS('flag', 20*8, explicit_name=True)
    buf     = 0x606000# 存储flag字符串
    crazy   = 0x400646# 符号执行的起始位置
    find    = 0x405a6e# 符号执行的目标位置

    avoids = [0x402c3c, 0x402eaf, 0x40311c, 0x40338b, 0x4035f8, 0x403868, 0x403ad5, 0x403d47,
              0x403fb9, 0x404227, 0x404496, 0x40470a, 0x404978, 0x404bec, 0x404e59, 0x4050c7,
              0x405338, 0x4055a9, 0x4057f4, 0x405a2b]

    proj = angr.Project('./yolomolo')
    state = proj.factory.blank_state(addr=crazy, add_options={angr.options.LAZY_SOLVES})
    state.memory.store(buf, flag, endness='Iend_BE')
    state.regs.rdi = buf

    for i in range(19):
        state.solver.add(flag.get_byte(i) >= 0x30)
        state.solver.add(flag.get_byte(i) <= 0x7f)

    simgr = proj.factory.simulation_manager(state)
    simgr.explore(find=find, avoid=avoids)
    found = simgr.found[0]
    return found.solver.eval(flag, cast_to=bytes)

if __name__ in '__main__':
    import logging
    logging.getLogger('angr.sim_manager').setLevel(logging.DEBUG)
    print(main())
```

HackCon 2016 angry-reverser

```
(angr) john@john-vm:~/workspace/tools/angr-doc/examples/hackcon2016_angry-reverser$ ls
solve.py  yolomolo
(angr) john@john-vm:~/workspace/tools/angr-doc/examples/hackcon2016_angry-reverser$ python solve.py
INFO    | 2019-12-27 08:51:39,083 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,098 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,108 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,121 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,130 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,142 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,160 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,170 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2019-12-27 08:51:39,183 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>

INFO    | 2019-12-27 08:52:16,338 | angr.sim_manager | Stepping active of <SimulationManager with 2 active, 17 avoid>
INFO    | 2019-12-27 08:52:16,403 | angr.sim_manager | Stepping active of <SimulationManager with 2 active, 18 avoid>
INFO    | 2019-12-27 08:52:16,514 | angr.sim_manager | Stepping active of <SimulationManager with 2 active, 18 avoid>
INFO    | 2019-12-27 08:52:16,579 | angr.sim_manager | Stepping active of <SimulationManager with 2 active, 19 avoid>
INFO    | 2019-12-27 08:52:16,593 | angr.sim_manager | Stepping active of <SimulationManager with 2 active, 19 avoid>
b'HACKCON{VVhYS04ngrY}'
(angr) john@john-vm:~/workspace/tools/angr-doc/examples/hackcon2016_angry-reverser$
```

执行时间 <1 分钟

CTF竞赛特点

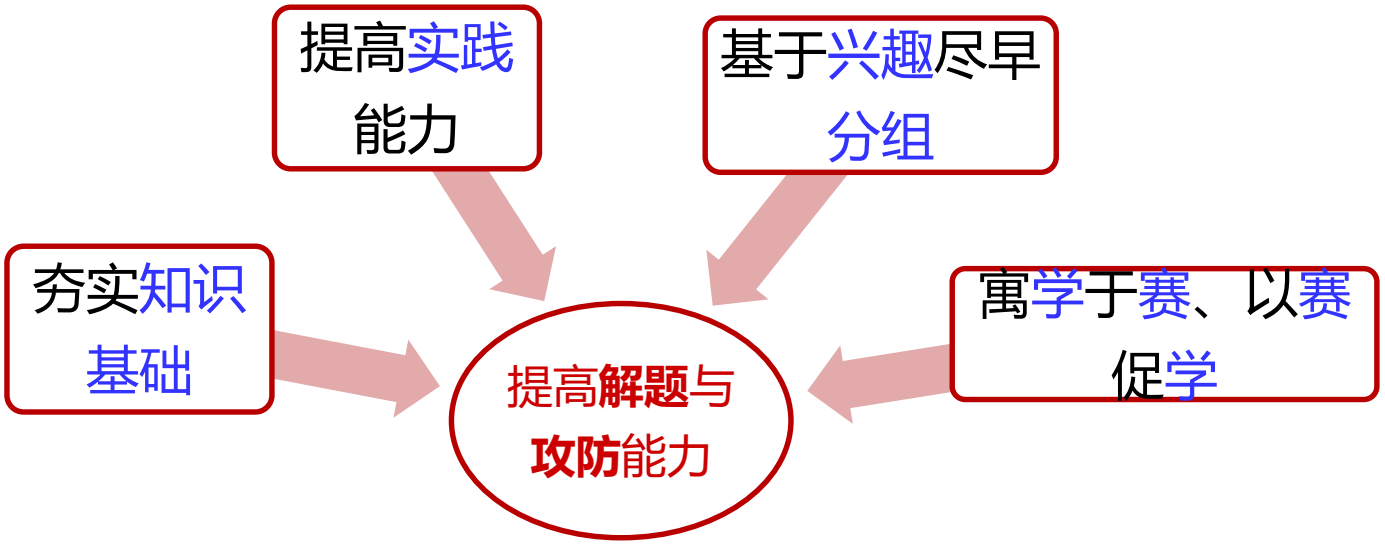
- 知识范围广，覆盖面宽
  - 密码学、系统安全、网络安全
  - 物理层、网络层、应用层...
  - 网络系统、软件系统、物联网系统、数据库与大数据、云与虚拟化、人工智能、区块链...
- 比赛形式多
  - 夺旗
  - 攻防
  - 过关
  - 渗透
  - 登顶
  - ...

CTF竞赛特点

- 知识的依赖度高，知识精深，学习难度大，学习曲线陡峭
  - 以二进制RE、PWN题目为例
    - 需要掌握程序设计语言、数据结构与算法、处理器体系结构、操作系统、软件工程方法等基础知识；
    - 需要掌握系统漏洞的研究需要汇编语言、编译与操作系统底层知识；
    - 复杂漏洞的成因还需要基于处理器的运行状态分析系统内存镜像等。
- 安全技术变化快，攻防对抗场景越来越丰富
  - 信息技术的变化非常快，新的系统使用了新的技术，新技术的产生又催生了新系统的开发，新的应用场景的出现又催生了新的安全需求，需要新的安全技术手段满足这些需求。
- 逆向思维与创新实践能力是安全专业学习的核心素质要求
  - 逆向思维，发掘漏洞
  - 快速学习，快速应用

CTF竞赛特点

- CTF竞赛核心能力
  - 依据网络空间安全专业知识体系的特殊性



网络攻防大赛发展趋势

- 人工智能攻防大赛/AI CTF
  - 2016年，DARPA CGC大赛，开启了自动/智能网络攻防的时代
    - DARPA期待有一天，美国可实现无需人力或工具便能自动抵御网络威胁。
    - Cyber Grand Challenge, 2014年启动，2016年在DEF CON大会上举行总决赛。
    - 是世界上第一场人工智能攻防大赛。
  - 2017年，XCTF总决赛，引入AI工具辅助CTF，属国内首次
    - 参赛战队需开发AI工具，自动发现与挖掘比赛题目中的漏洞，人类战队进行漏洞利用。
  - 2017年，武汉网信办主办“首届国际机器人网络安全大赛”
    - 人与机器网络攻防竞赛Robo Hacking Game, roboCTF/RHG
  - 2018年，国家互联网应急中心CNCERT/CC举办“中国网络安全技术对抗赛”
    - 人工智能安全夺旗赛，AI系统的漏洞挖掘和利用
  - 2019年，百度举办BCTF
    - 在DEF CON CHINA中举办，AI系统自动攻防CTF
  - 2019年，国家网信办主办“强网杯”全国网络安全大赛
    - 单独设置人工智能攻防赛AI CTF

网络攻防大赛发展趋势

- Real World CTF
  - 2018年，国际35C3 CTF中引入zajebiste题型，波兰语，即Real World题型
    - 共5题，包括
      - VirtualBox虚拟机逃逸
      - Web CMS漏洞
      - Keybase漏洞
      - logrotate漏洞
      - Ruby解析器漏洞
  - 2019年，“强网杯”总决赛引入Real World题型
    - 3个部署运行的Web CMS系统
    - D-Link固件0day
    - QEMU虚拟机CVE
    - Ubuntu内核提权等

网络攻防大赛发展趋势

- 软件破解大赛
  - 国际PWN2OWN大赛，美国ZDI主办的世界最著名、奖金最丰厚的破解大赛
    - 2019年破解目标包括虚拟化类（VirtualBox/VMware Workstation/VMware ESXi/Hyper-V）、浏览器（Chrome/Edge/Safari/Firefox）、企业应用（Adobe Reader/Office 365/Outlook）、服务端（Windows RDP）和特斯拉电动汽车。
  - 国内KEEN Team举办的极棒大赛GeekPwn，关注智能设备的安全
    - 2019年pwn目标包括“CAAD 语音对抗样本挑战赛”、“CAAD CTF 图像对抗样本挑战赛”、“云安全挑战赛”等，还包括“隐私安全”、“华为智能设备安全挑战专场赛”。
  - “天府杯”TFC 2018 国际网络安全大赛，国内多家厂商共同举办
    - 破解目标包括：
      - Edge、Firefox、Chrome、Safari
      - Office、Adobe PDF Reader
      - iPhone X + iOS 12
      - Microsoft Remote Desktop
      - Oracle Virtual Box、VMWare Workstation 15
      - Xiaomi Mi8、OPPO R17、VIVO x23

加强交流，合作共赢，建立命运共同体！

!!! 注 意 事 项

- 攻防对抗专业人才特殊性
  - 容易触及法律，高风险
- 网络安全法
  - 第五条 国家采取措施，监测、防御、处置来源于中华人民共和国境内外的网络安全风险和威胁，保护关键信息基础设施免受攻击、侵入、干扰和破坏，依法惩治网络违法犯罪活动，维护网络空间安全和秩序。
- 禁止滥用安全技术
- 将安全技术用于保护自己

全国人民代表大会
The National People's Congress of the People's Republic of China
中国人大网 www.npc.gov.cn [中文版] [English]

首页 | 宪法 | 国家机构 | 人大机构 | 票战书委员长 | 代表大会会议 | 常委会会议 | 委员长会议 | 权威发布 | 立法工作 | 监督工作
代表工作 | 对外交往 | 选举任免 | 理论研究 | 机关工作 | 地方人大 | 图片 | 视频 | 直播 | 访谈 | 专题 | 手机版 | 英文版

新闻

当前位置：首页 >

中华人民共和国网络安全法

（2016年11月7日第十二届全国人民代表大会常务委员会第二十四次会议通过）

来源： 中国人大网 2016年11月7日 17:31:34 浏览字号： 大 中 小

目 录

谢谢!