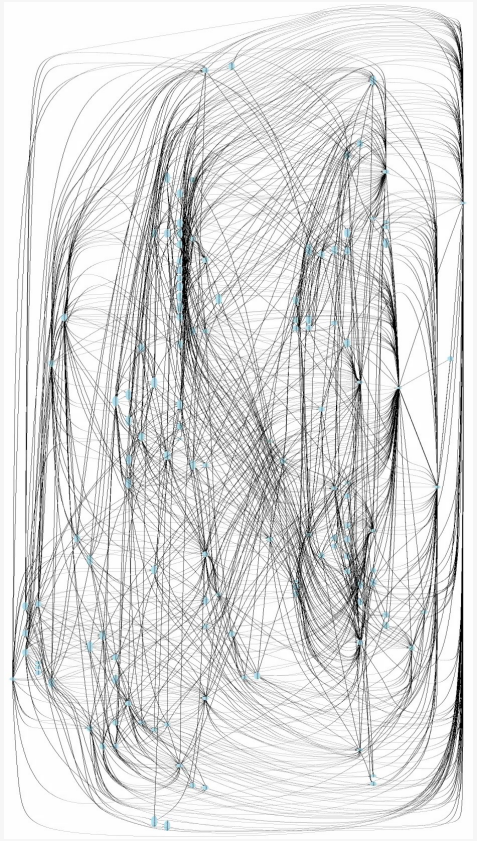
Open Universiteit

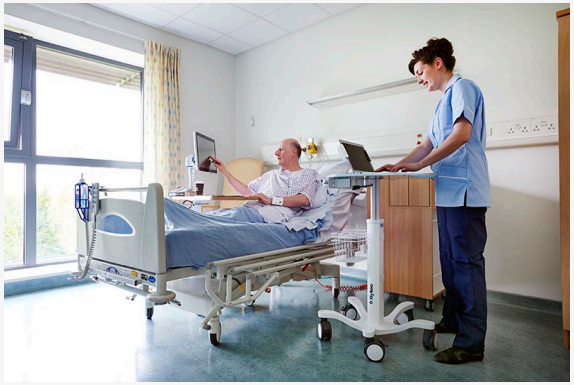
Exploit Logic

May 25th, 2021

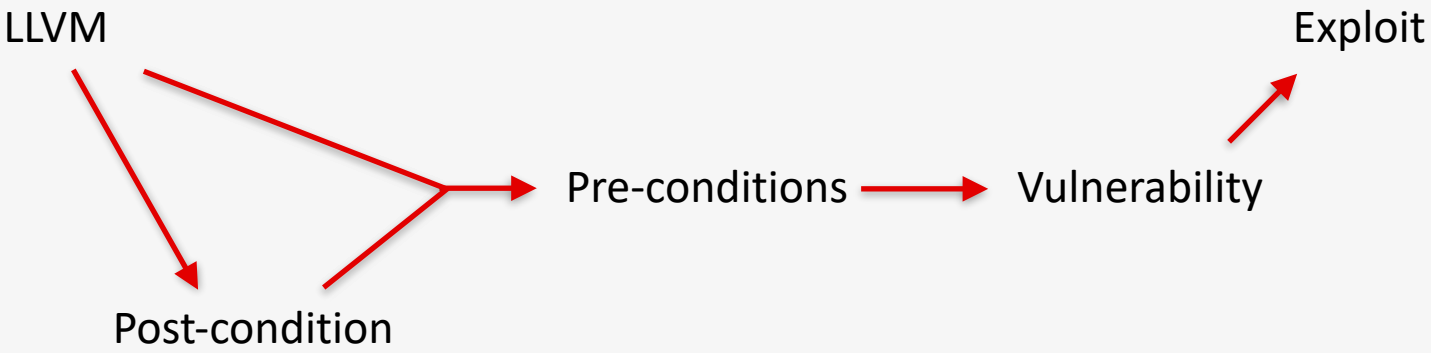
Dr. Nico Naus

Why?





What/how?


LLVM
Exploit
Pre-conditions
Vulnerability
Post-condition

Program Logics

Hoare Logic

$$\{P\}\ p\ \{Q\} \equiv \forall \sigma \cdot P(\sigma) \implies \forall \sigma' \cdot \sigma \longrightarrow \sigma' \implies Q(\sigma')$$

```
void foo(int64_t i, int64_t v) {
  int64_t x[10];
  if (random()) x[i] = v;
}
```

Exploit occurs when we write outside the array

Program Logics

Reverse Hoare Logic

$$[P]\ p\ [Q] \equiv \forall \sigma' \cdot\ Q(\sigma')\ \Longrightarrow\ \exists \sigma \cdot \sigma \longrightarrow \sigma' \wedge\ P(\sigma)$$

[w = 100] v := w [v > 42]

Exploit Logic

$$\langle P \rangle\ p\ \langle Q \rangle \equiv \forall \sigma \cdot P(\sigma) \implies \exists \sigma' \cdot \sigma \longrightarrow \sigma' \wedge\ Q(\sigma')$$

```
void foo(int64_t i, int64_t v) {
  int64_t x[10];
  if (random()) x[i] = v;
}
```

⟨i = 10 ∧ v = a⟩ foo(i, v) ⟨x[i] = a⟩

$$\langle w = 100 \rangle\ v := w\ \langle v > 42 \rangle$$

JUMP language

Block

$\underline{b}$	=	$\underline{s}$; $\underline{b}$	Sequence
	|	Jump $\underline{e}$ a_1 a_2	Conditional jump
	|	Exit	Exit

Statement

$\underline{s}$	=	Assign v $\underline{e}$	Variable assignment
	|	Obtain v Where $\underline{e}$	Nondeterministic assign
	|	Store $\underline{e}$ v	Store v in region e

Expression

$\underline{e}$	=	w	Value
	|	v	Variable
	|	!$\underline{e}$	Dereference
	|	$\underline{e_1} \oplus \underline{e_2}$	Binary operation
	|	$\neg \underline{e}$	Negation

$\oplus$	$\in$	$\{+, -, \times, \%, <, \leq, =, >, \geq, \wedge, \vee, \ldots\}$	Binary operators

Precondition generation

Program:

$$\tau_P(p, Q) = \tau_B(\text{blocks}(a_0), Q)$$

Block:

$$\tau_B(\underline{\text{s}};\underline{\text{b}}, Q) = \bigcup\{\tau_S(\underline{\text{s}}, P) \mid P \in \tau_B(\underline{\text{b}}, Q)\}$$

$$\tau_B(\texttt{Jump}\ \underline{\text{e}}\ a_1\ a_2, Q) = \{P_1 \wedge \underline{\text{e}} \mid P_1 \in \tau_b(\text{blocks}(a_1), Q)\} \cup \{P_2 \wedge \neg\underline{\text{e}} \mid P_2 \in \tau_b(\text{blocks}(a_2), Q)\}$$

$$\tau_B(\texttt{Exit}, Q) = \{Q\}$$

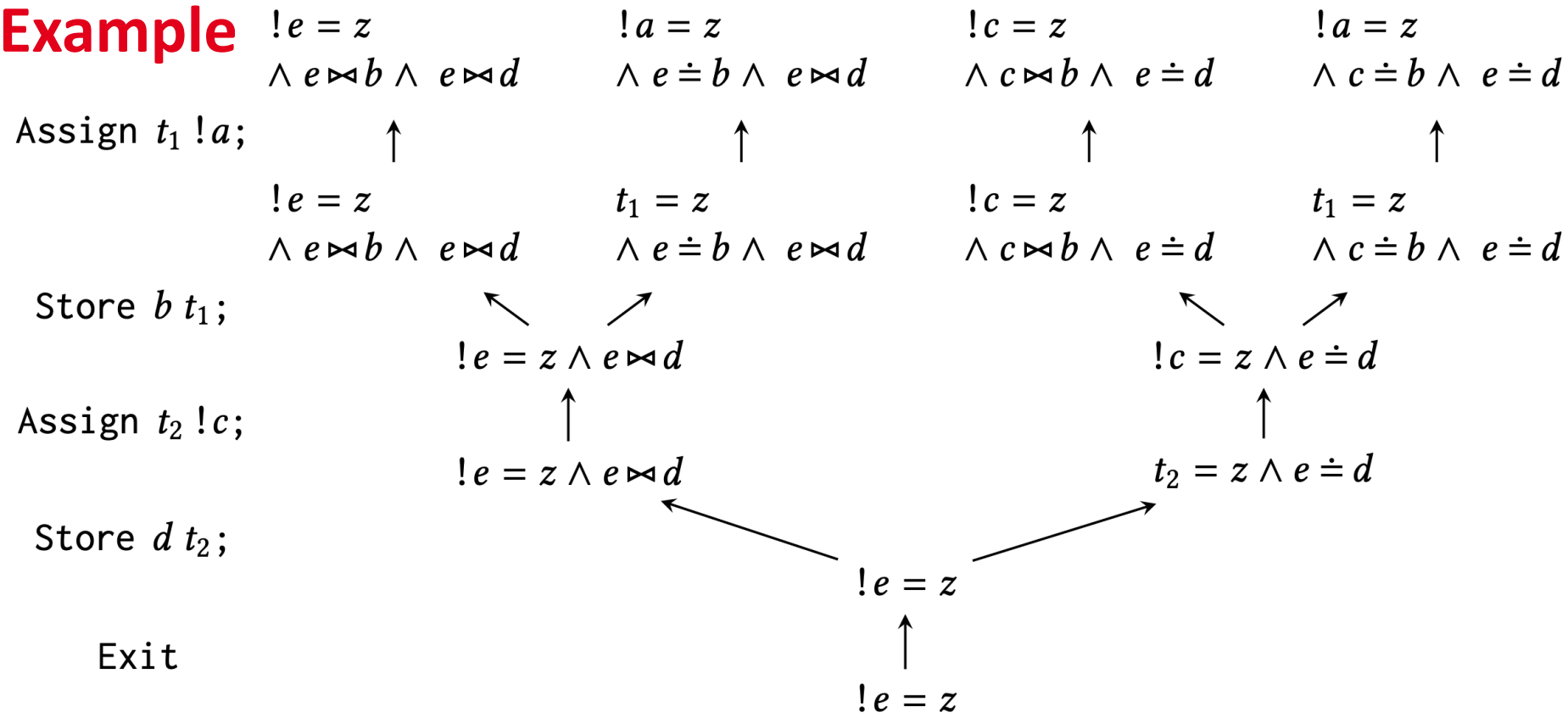
Statement:

$$\tau_S(\texttt{Assign}\ v\ \underline{\text{e}}, Q) = \{Q[v := \underline{\text{e}}]\}$$

$$\tau_S(\texttt{Obtain}\ v\ \texttt{Where}\ \underline{\text{e}}, Q) = \{\exists i \in \underline{\text{e}} \cdot Q[v := i]\}$$

$$\tau_S(\texttt{Store}\ \underline{\text{e}}\ v, Q) = \{Q' \wedge P \mid (Q', P) \in \tau_{\texttt{store}}(\underline{\text{e}}, v, Q)\}$$

Example

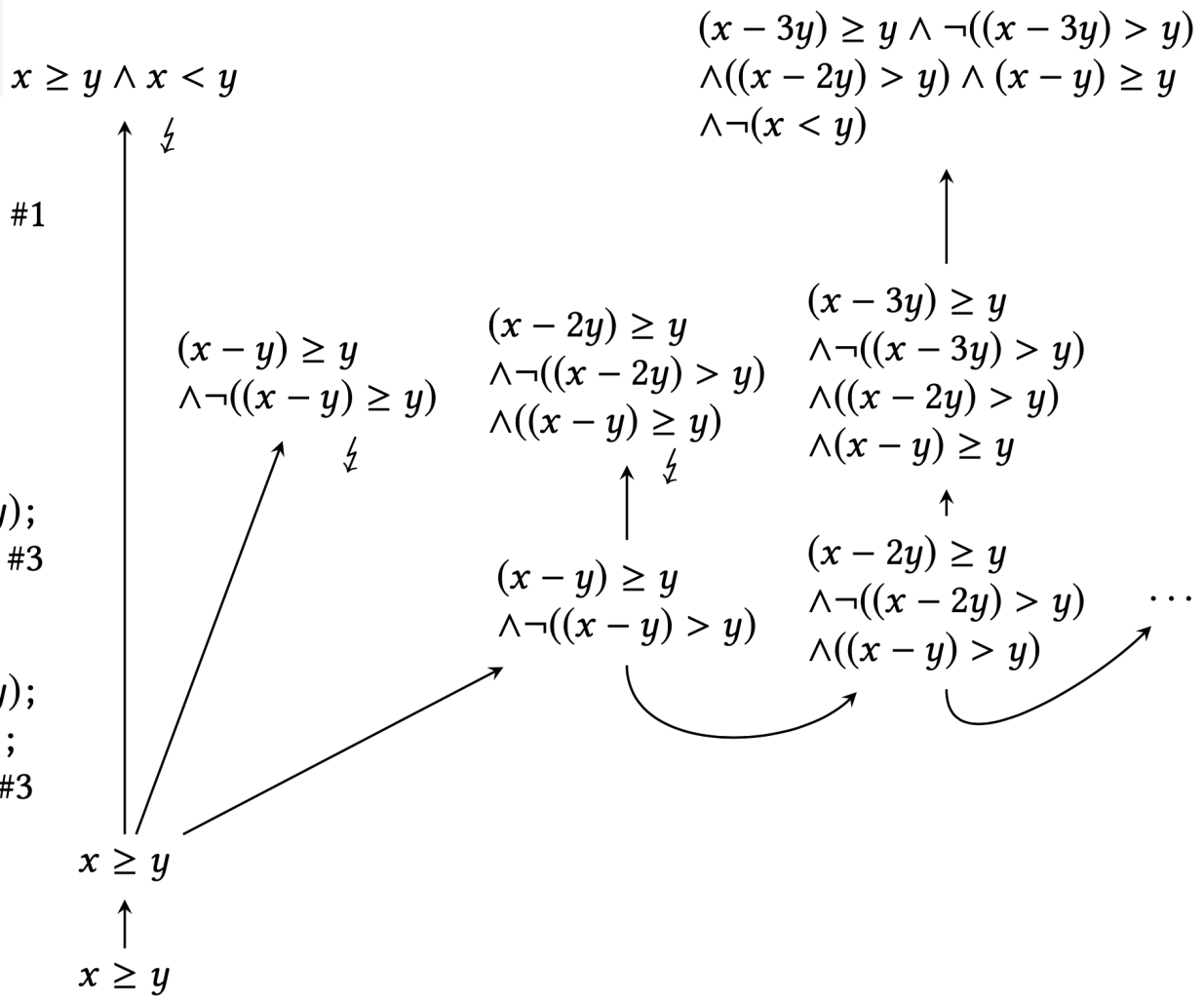

!e = z ∧ e ⋈ b ∧ e ⋈ d
!a = z ∧ e ≐ b ∧ e ⋈ d
!c = z ∧ c ⋈ b ∧ e ≐ d
!a = z ∧ c ≐ b ∧ e ≐ d
Assign t1 !a;
!e = z ∧ e ⋈ b ∧ e ⋈ d
t1 = z ∧ e ≐ b ∧ e ⋈ d
!c = z ∧ c ⋈ b ∧ e ≐ d
t1 = z ∧ c ≐ b ∧ e ≐ d
Store b t1;
!e = z ∧ e ⋈ d
!c = z ∧ e ≐ d
Assign t2 !c;
!e = z ∧ e ⋈ d
t2 = z ∧ e ≐ d
Store d t2;
!e = z
Exit
!e = z

#0
Jump $(x < y)$ #3 #1

#1
Assign i 1;
Assign x $(x - y)$;
Jump $(x \geq y)$ #2 #3

#2
Assign x $(x - y)$;
Assign i $(i + 1)$;
Jump$(x > y)$ #2 #3

#3
Exit



Future work

- Finding a postcondition automatically
- Searching though exploit space efficiently
- Expanding the JUMP language

Summary

- Exploit Logic as a formal basis for automatic exploit generation
- Precondition generation rules build the exploit space
- Finding only one satisfiable precondition is enough