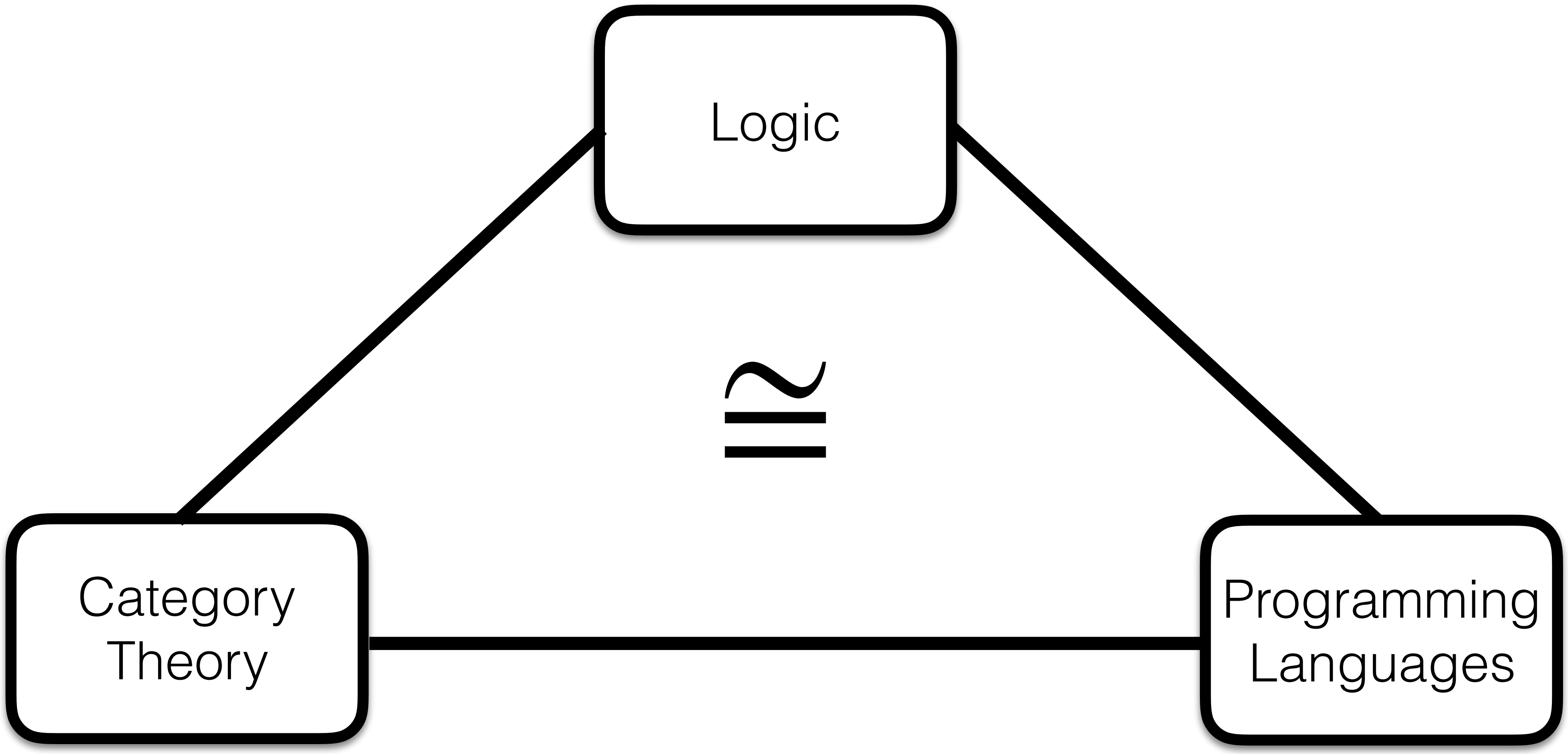
On Linear Logic, Functional Programming, and Attack Trees

Harley Eades III
Augusta University

Jiaming Jiang
North Carolina State University

Aubrey Bryant
Augusta University

How I Approach Problems

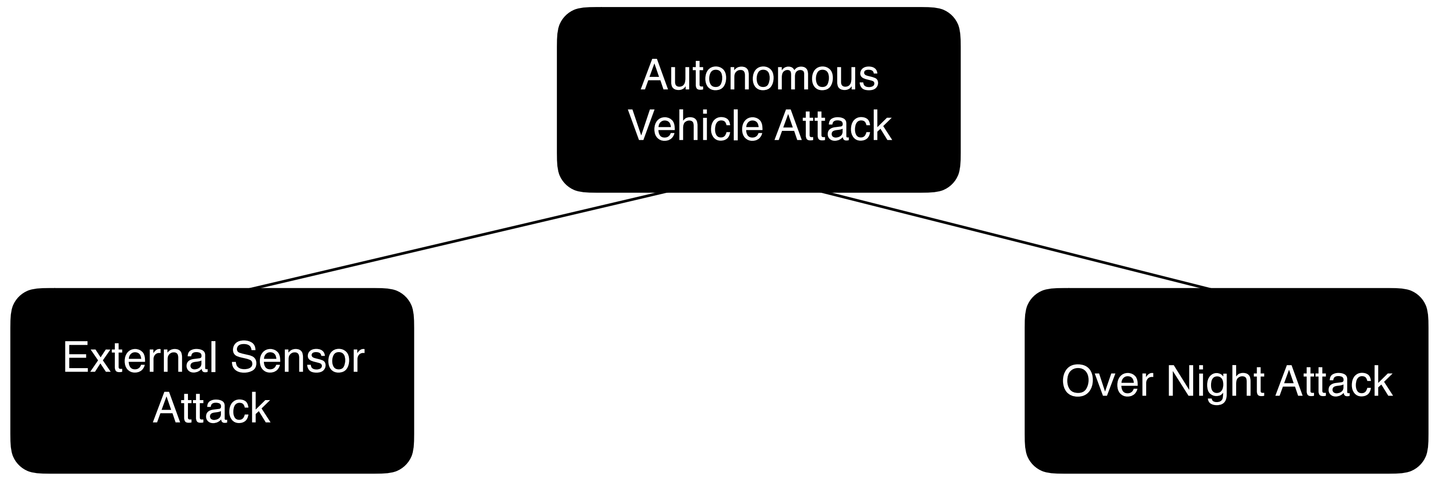


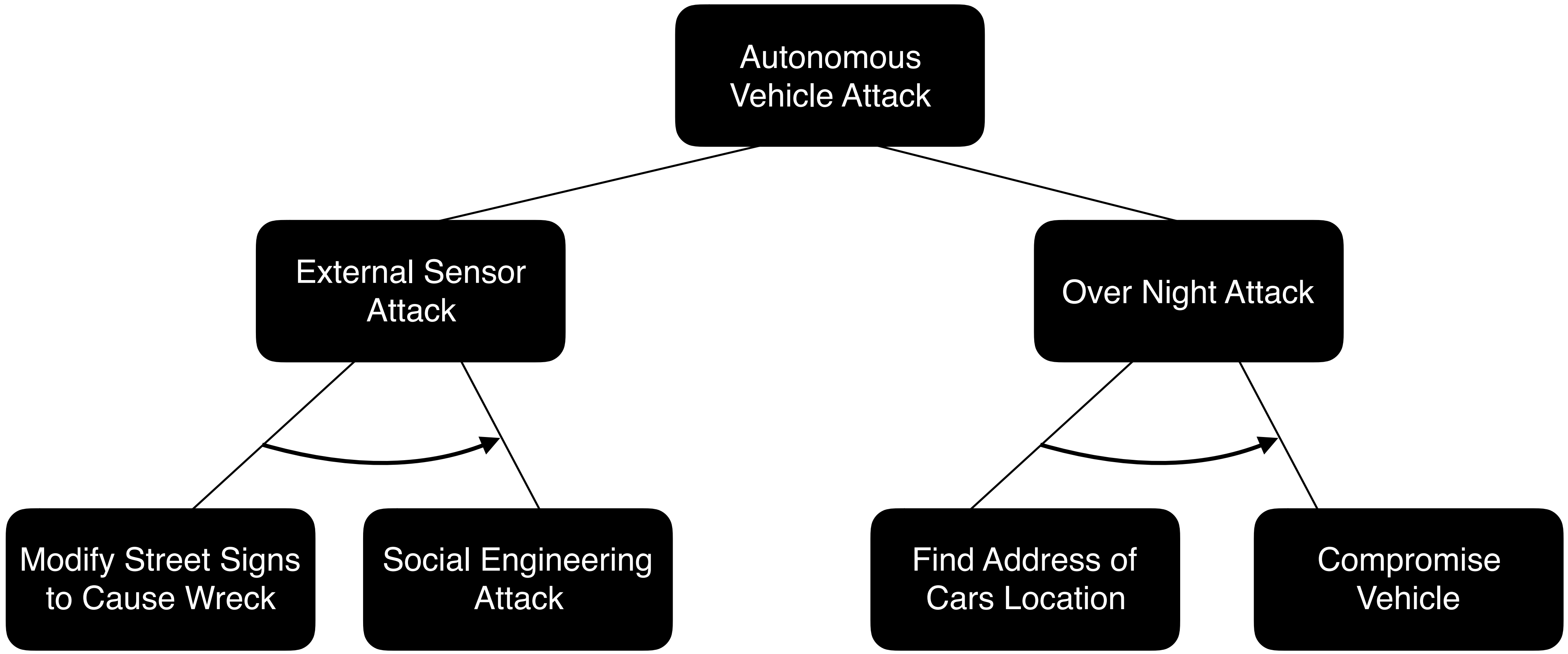
The Three Perspectives of Computation

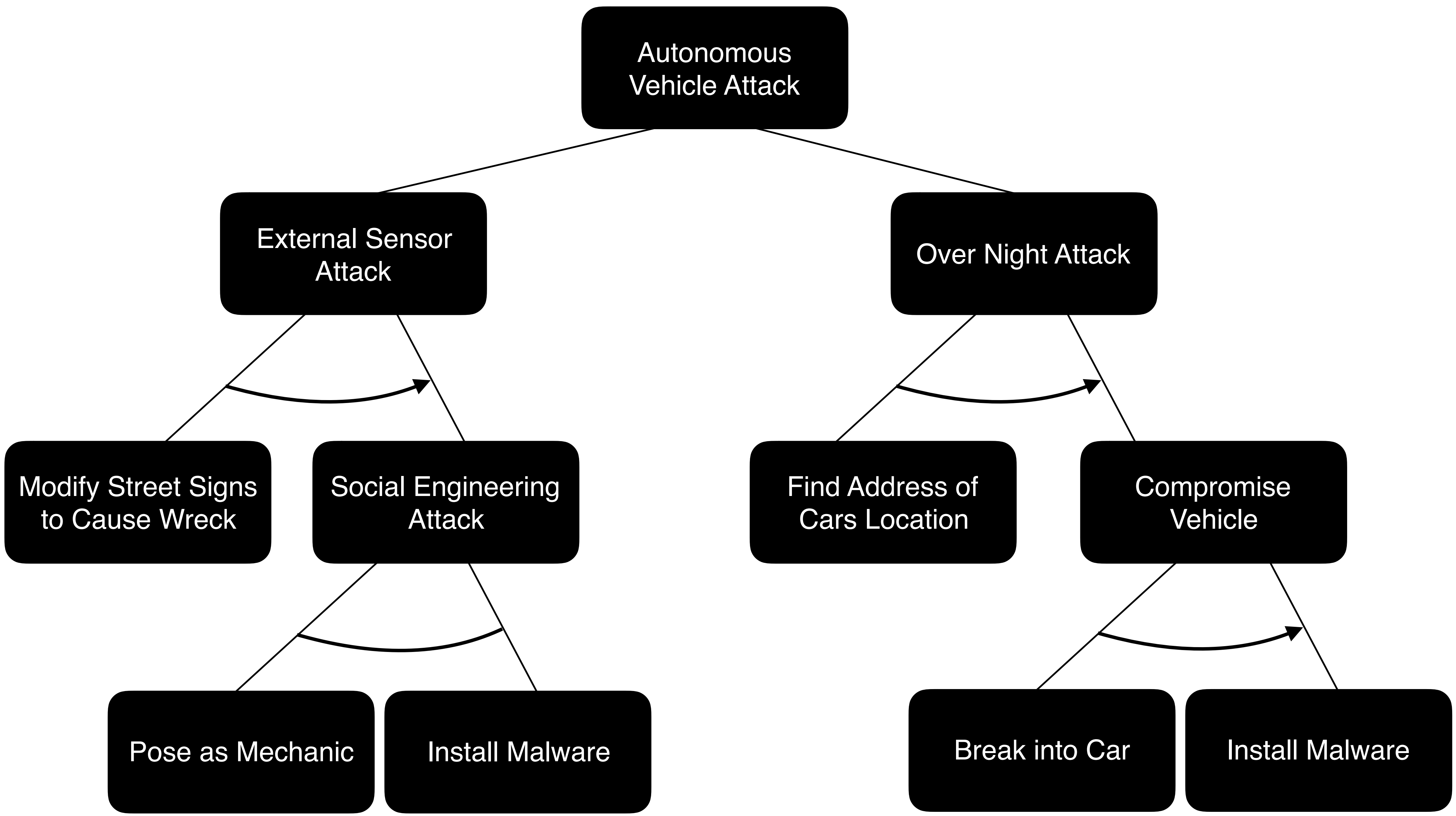
What is an attack tree?

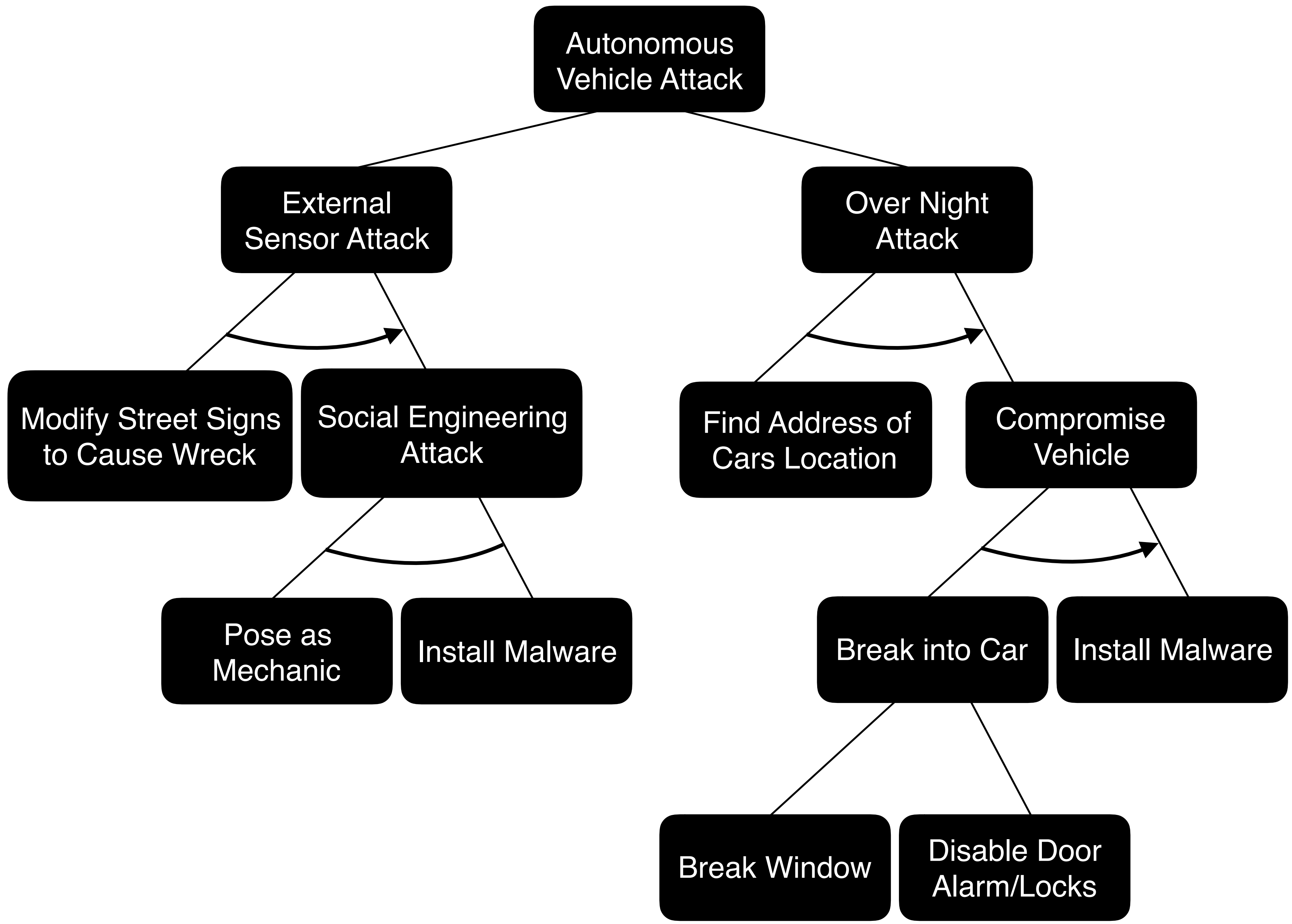
When are we allowed to modify an attack tree?

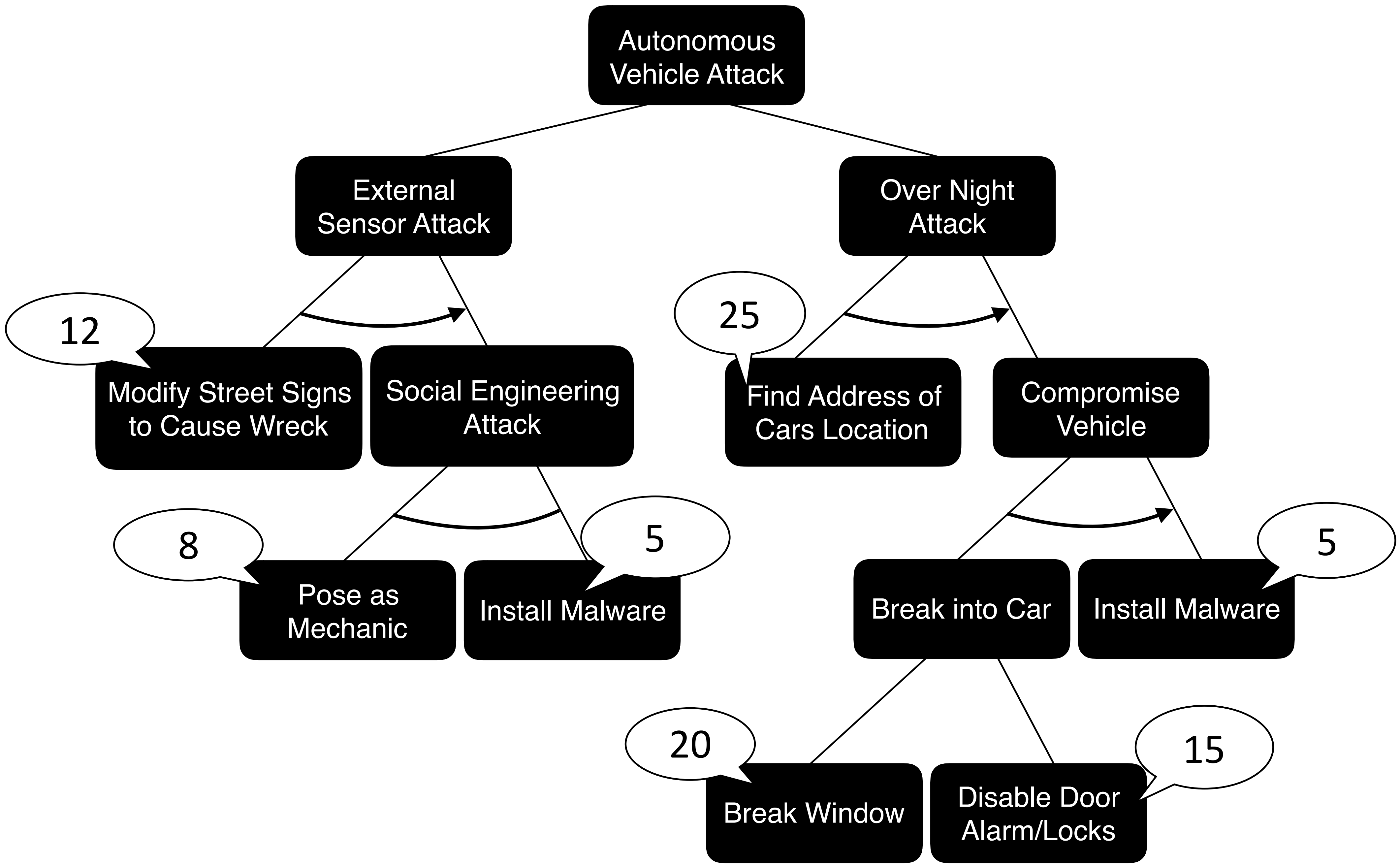
Autonomous
Vehicle Attack

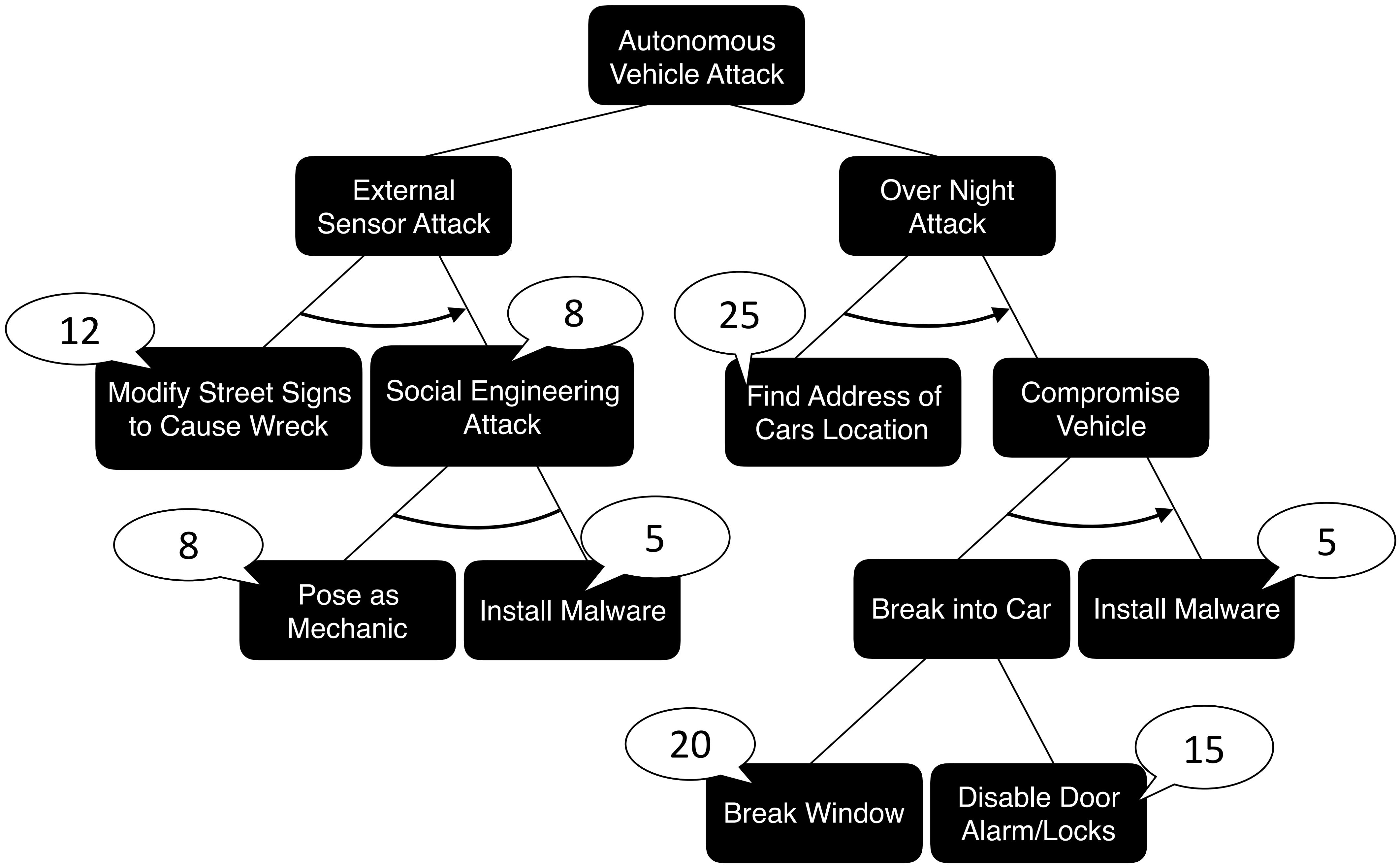

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack

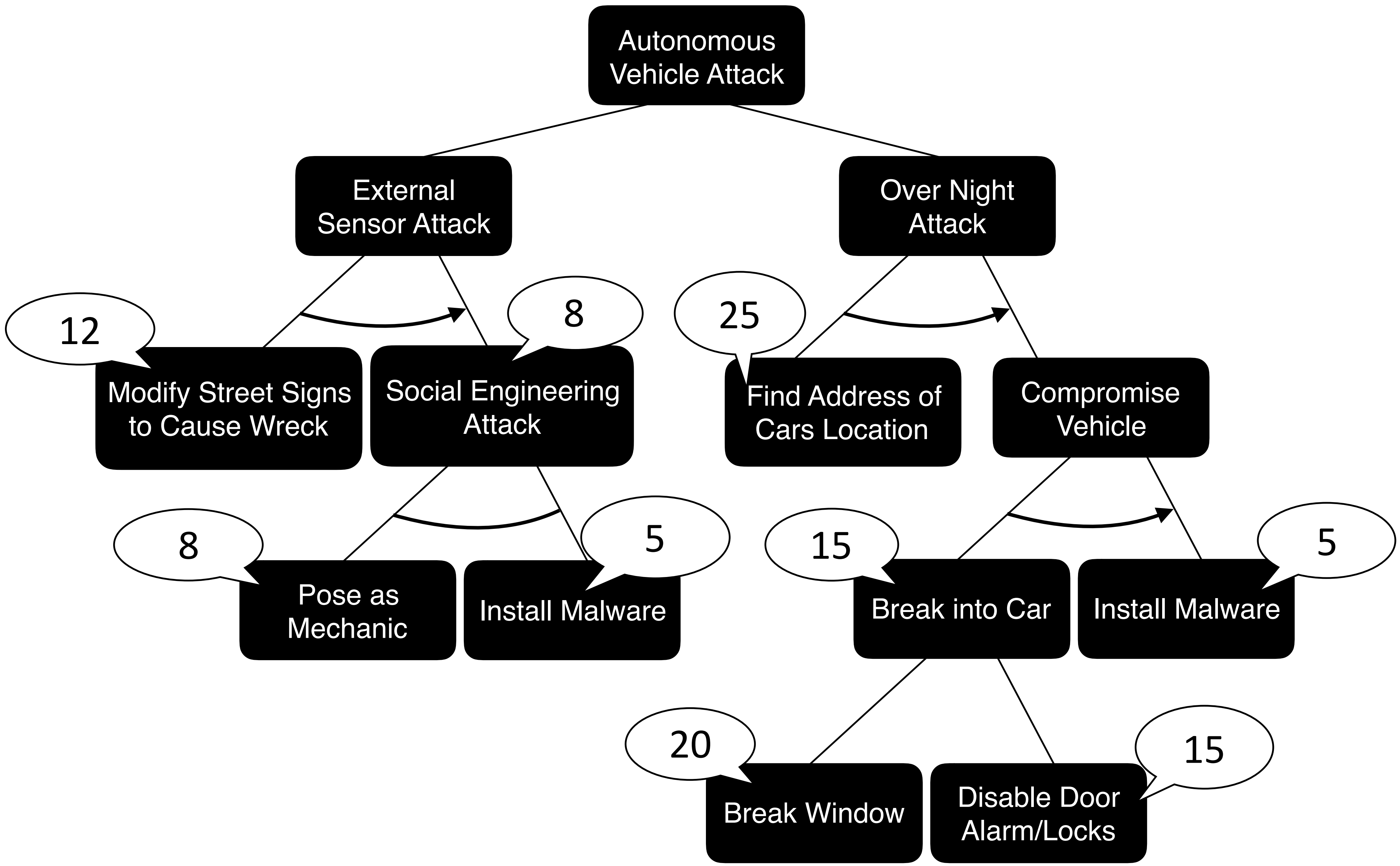

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle

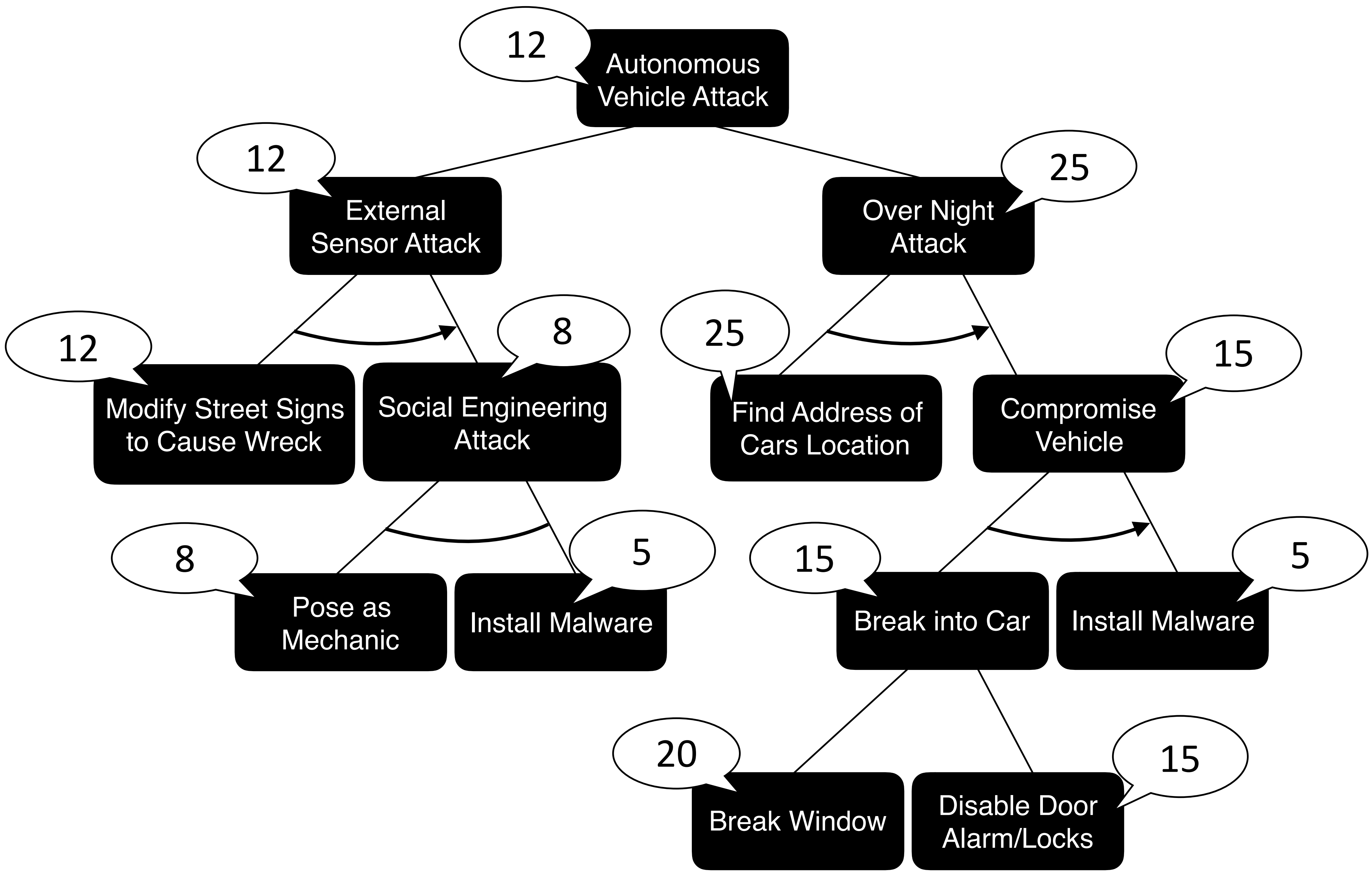

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware

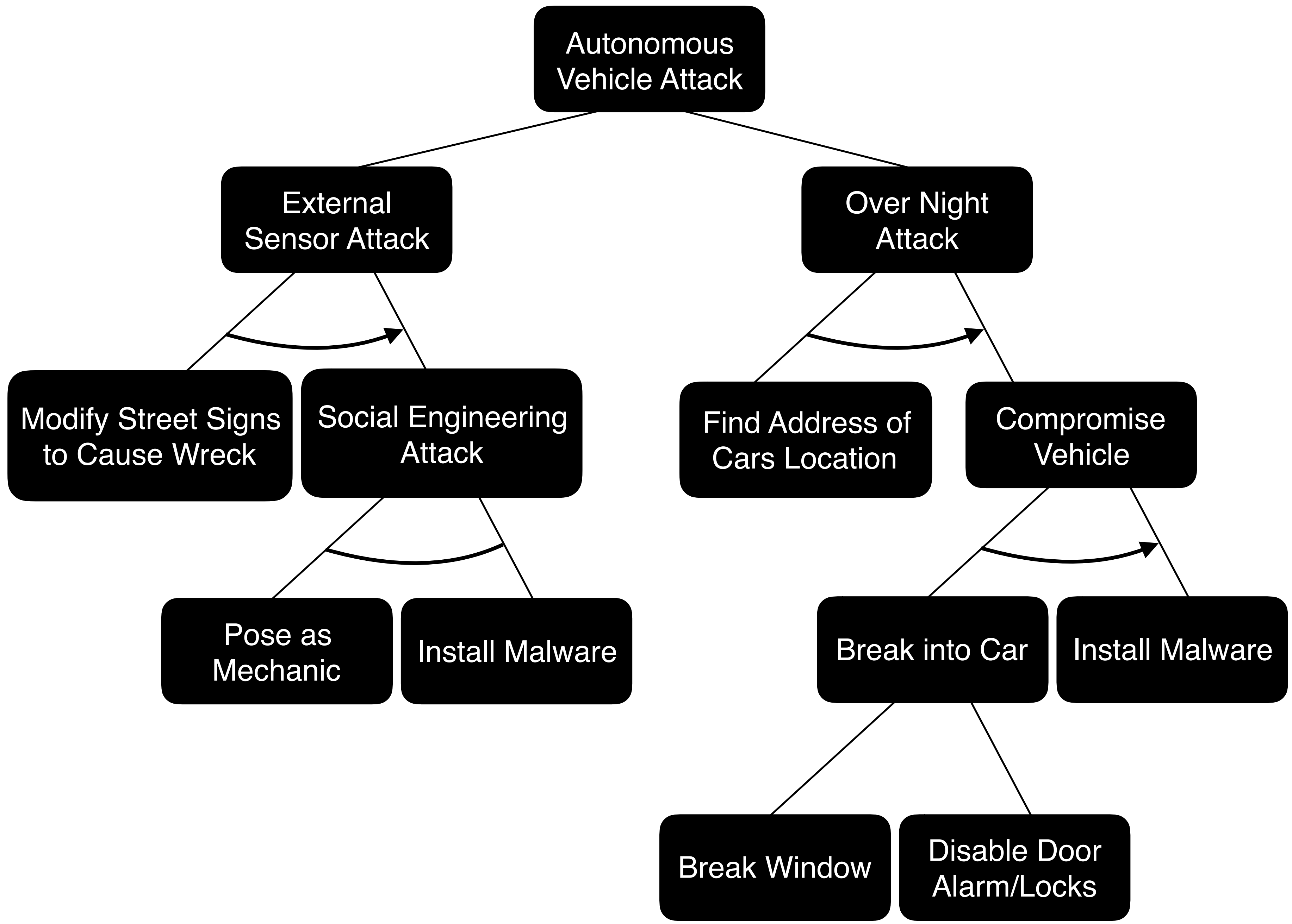

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks

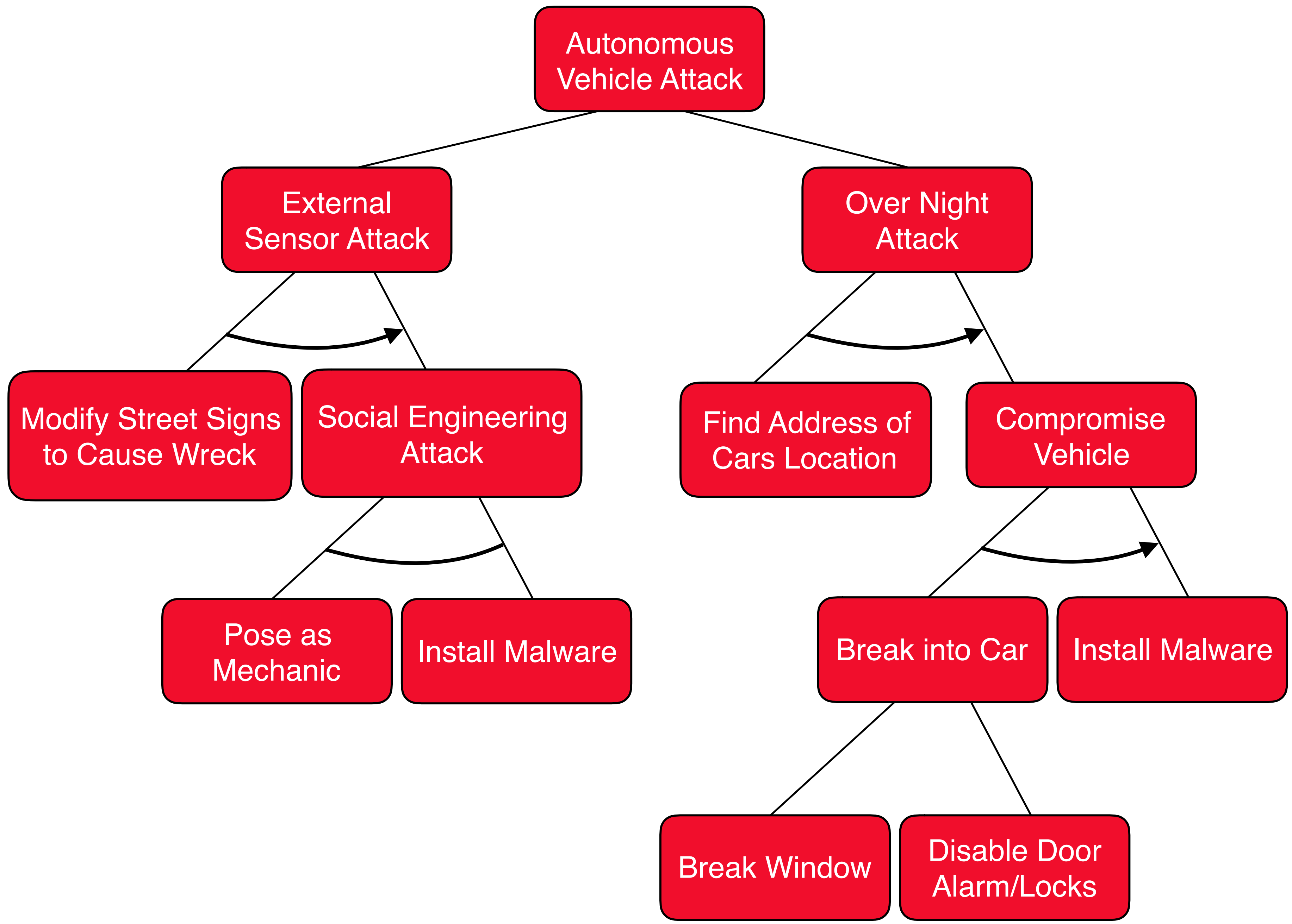

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks

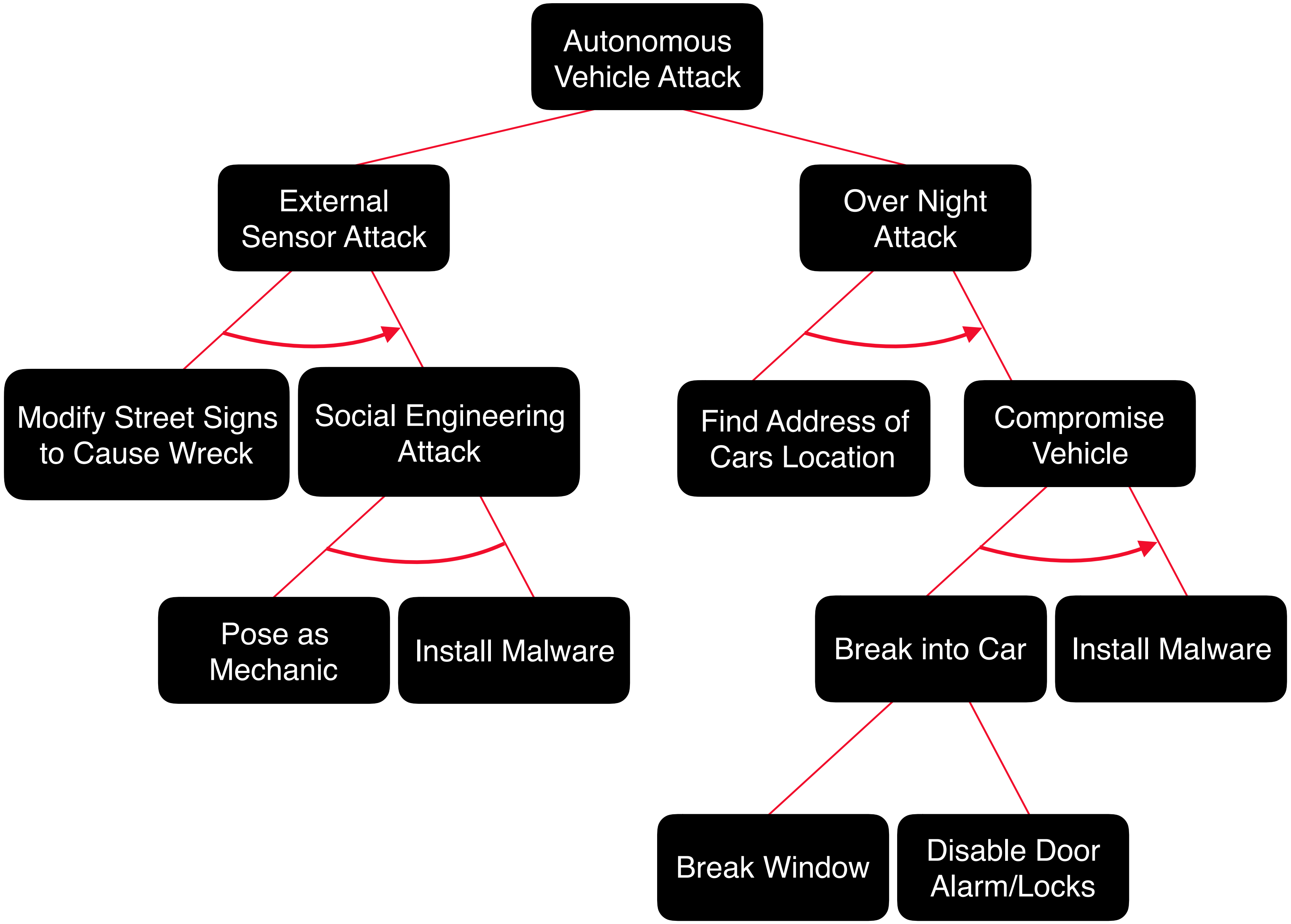

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks


Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
15
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks

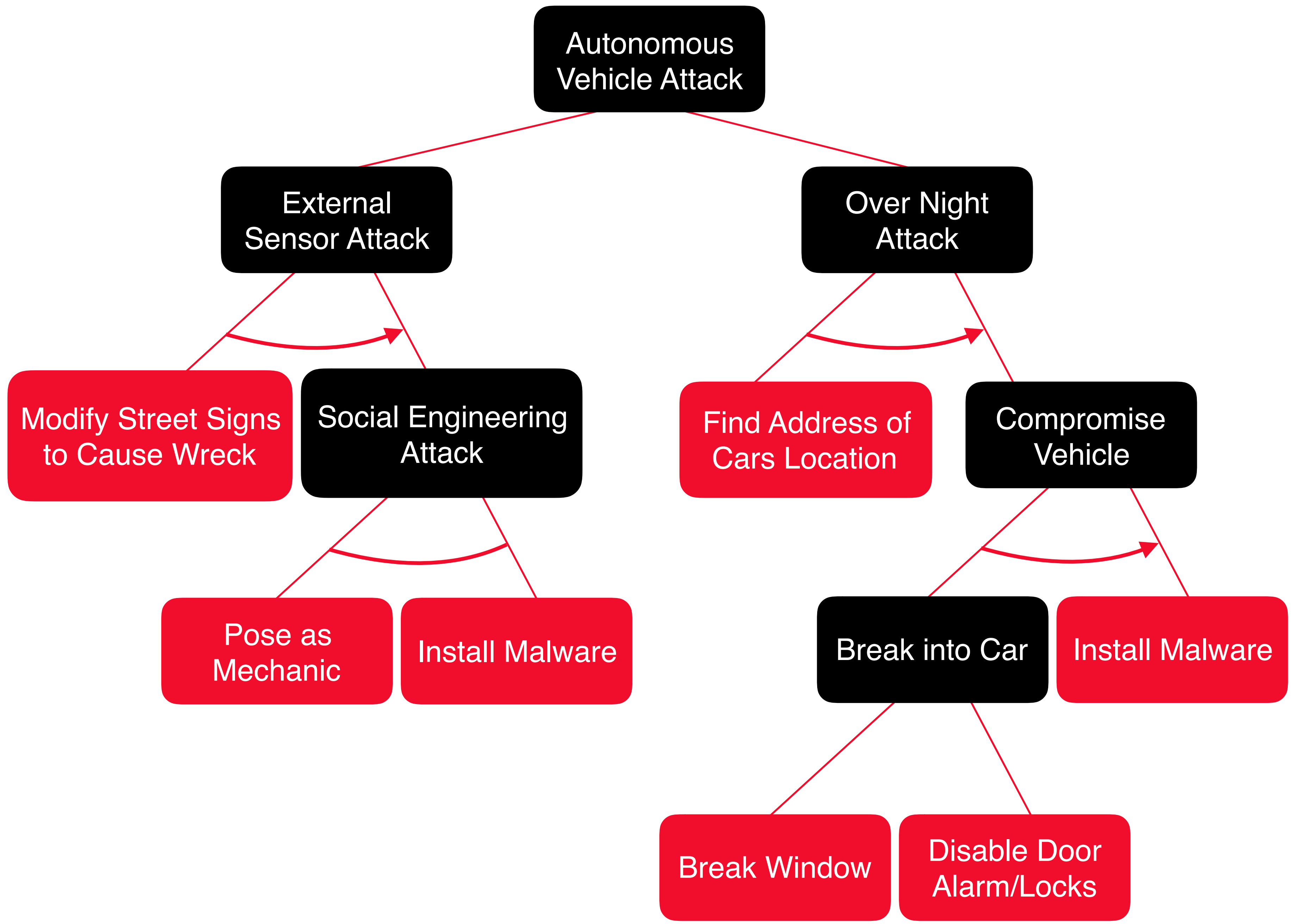

12
Autonomous Vehicle Attack
12
External Sensor Attack
25
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
15
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
15
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks

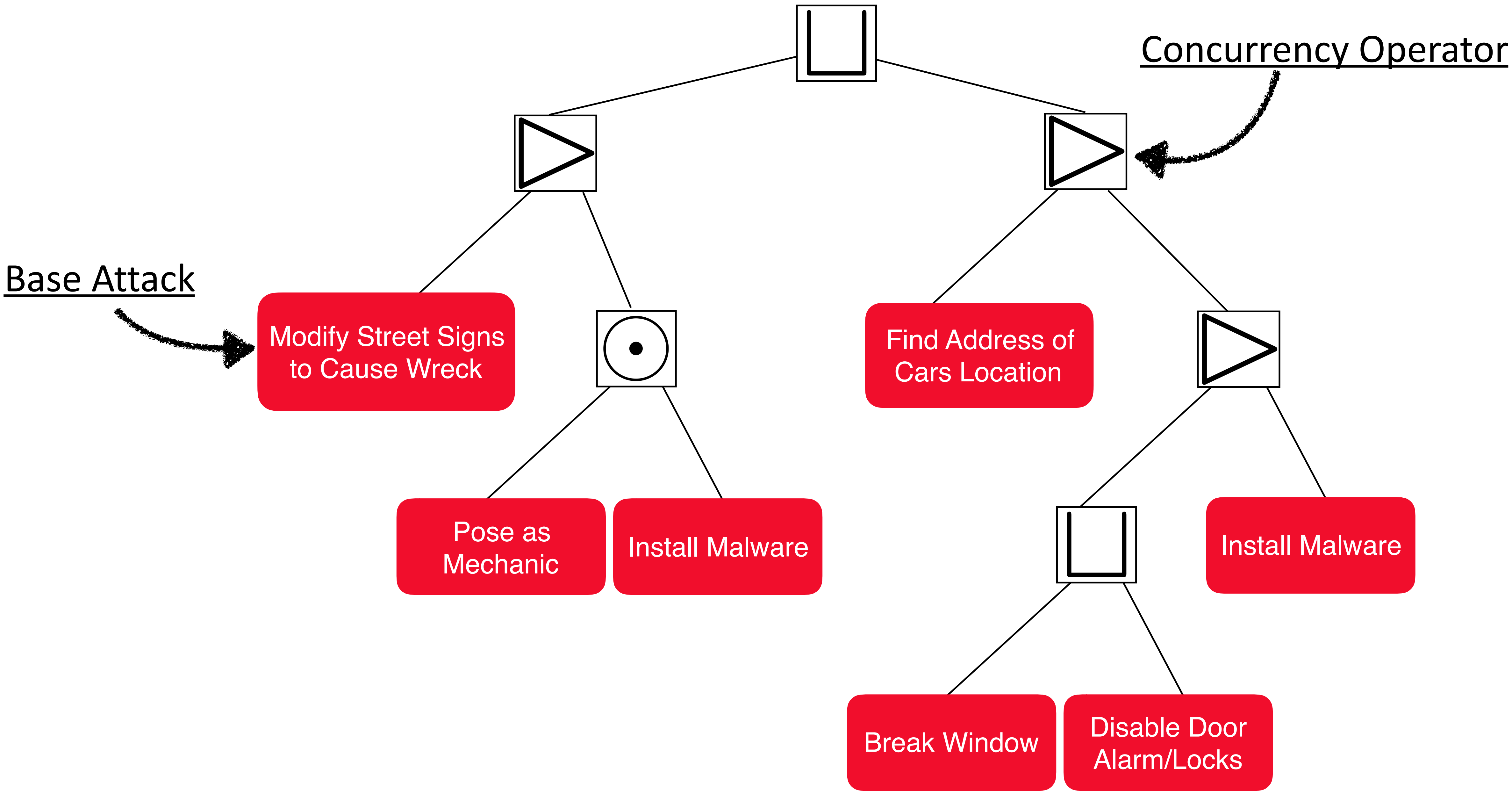

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks


Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks



Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks


Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks


Concurrency Operator
Base Attack
Modify Street Signs to Cause Wreck
Find Address of Cars Location
Pose as Mechanic
Install Malware
Install Malware
Break Window
Disable Door Alarm/Locks

$A =$ “Modify Street Signs to Cause Wreck“
$B =$ “Pose as Mechanic“
$C =$ “Install Malware“
$D =$ “Find Address of Cars Location“
$E =$ “Break Window“
$F =$ “Disable Door Alarm/Locks“

$$(A \rhd (B \odot C)) \sqcup (D \rhd ((E \sqcup F) \rhd C))$$

Attack Trees in Resource-Sensitive Logics

Resource-Sensitive Logics:

- Model Resource Critical Systems as Formulas
- Prove Properties about the Modeled Systems by Proving Properties about Formulas
- Understands Concurrency
- Formally Controls Duplication of Resources

Attack Trees in Resource-Sensitive Logics

Reasoning about Attack Trees:

- Model **Attack Trees** as Formulas in Resource-Sensitive Logics

- Prove Properties about **Attack Trees** by Proving Properties about Formulas

- Respects the Concurrency Perspective of Attack Trees

Quaternary Semantics of Attack Trees

- Four-Valued Truth Table Semantics
- Intuitionistic
- Proofs are simple
- Resource Sensitive

Quaternary Semantics of Attack Trees

Supports Specializations [Horne et al.:2016]:

Prove implications between attack trees that take into consideration both the **logical structure** of the tree and the **attribute domain**.

Quaternary Semantics of Attack Trees

Two Types of Semantics [Horne et al.:2016]:

- Ideal Semantics
- Filter Semantics

Ideal Quaternary Semantics of Attack Trees

Truth tables over propositional variables:

$$A, B \in \{0, \frac{1}{4}, \frac{1}{2}, 1\}$$

Ideal Quaternary Semantics of Attack Trees

Choice:

$A \sqcup_I B = \mathsf{max}(A, B)$

Sequence:

$0 \rhd_I B = 0$

$A \rhd_I 0 = 0$

$A \rhd_I B = \dfrac{1}{2}$, when $A \in \{\dfrac{1}{2}, 1\}$

Parallel:

$0 \odot_I B = 0$

$A \odot_I 0 = 0$

$A \odot_I B = 1$

Ideal Quaternary Semantics of Attack Trees

Logical Sequent (implication) is a Partial Ordering:

$A \leq_4 B$

Equivalence of Attack Trees:

$A \equiv B$ iff $(A \leq_4 B)$ and $(B \leq_4 A)$

Ideal Quaternary Semantics of Attack Trees

Basic Properties for Choice:

$$A \leq_4 (A \sqcup_I B)$$

$$B \leq_4 (A \sqcup_I B)$$

$$(A \sqcup_I B) \equiv (B \sqcup_I A)$$

$$((A \sqcup_I B) \sqcup_I C) \equiv (A \sqcup_I (B \sqcup_I C))$$

If $A \leq_4 C$ and $B \leq_4 D$, then $(A \sqcup_I B) \leq_4 (C \sqcup_I D)$

Ideal Quaternary Semantics of Attack Trees

Basic Properties for Parallel:

$$(A \odot_I A) \not\equiv A$$

$$(A \odot_I B) \equiv (B \odot_I A)$$

$$((A \odot_I B) \odot_I C) \equiv (A \odot_I (B \odot_I C))$$

If $A \leq_4 C$ and $B \leq_4 D$, then $(A \odot_I B) \leq_4 (C \odot_I D)$

$$(A \odot_I (B \sqcup_I C)) \equiv ((A \odot_I B) \sqcup_I (A \odot_I C))$$

Ideal Quaternary Semantics of Attack Trees

Basic Properties for Sequence:

$$(A \rhd_I A) \not\equiv A$$

$$(A \rhd_I B) \not\equiv (B \rhd_I A)$$

$$(A \rhd_I (B \rhd_I C)) \equiv ((A \rhd_I B) \rhd_I C)$$

If $A \leq_4 C$ and $B \leq_4 D$, then $(A \rhd_I B) \leq_4 (C \rhd_I D)$

$$(A \rhd_I (B \sqcup_I C)) \equiv ((A \rhd_I B) \sqcup_I (A \rhd_I C))$$

Ideal Quaternary Semantics of Attack Trees

Ideal Properties [Horne et al.:2016]:

$$((A \odot_I B) \rhd_I (C \odot_I D)) \leq_4 ((A \rhd_I C) \odot_I (B \rhd_I D))$$

$$((A \odot_I B) \rhd_I C) \leq_4 (A \odot_I (B \rhd_I C))$$

$$(A \rhd_I (B \odot_I C) \leq_4 (B \odot_I (A \rhd_I C))$$

$$(A \rhd_I B) \leq_4 (A \odot_I B)$$

Note: Not equivalences!

Filterish Quaternary Semantics of Attack Trees

Choice:

$A \sqcup_F B = \max(A, B)$

Sequence:

$0 \rhd_F B = 0$

$A \rhd_F 0 = 0$

$A \rhd_F B = 1$, when $A \in \{\frac{1}{2}, 1\}$

$\frac{1}{4} \rhd_F B = \frac{1}{4}$

Parallel:

$0 \odot_F B = 0$

$A \odot_F 0 = 0$

$A \odot_F B = \frac{1}{2}$

Filterish Quaternary Semantics of Attack Trees

Same basic properties for each form of composition.

Filterish Quaternary Semantics of Attack Trees

Filter properties that **hold** [Horne et al.:2016]:

$$((A \rhd_F C) \odot_F (B \rhd_F D)) \leq_4 ((A \odot_F B) \rhd_F (C \odot_F D))$$

$$(A \odot_F (B \rhd_F C)) \leq_4 ((A \odot_F B) \rhd_F C)$$

Filter properties that **fail** [Horne et al.:2016]:

$$(A \rhd_F (B \odot_F C)) \leq_r (B \odot_F (A \rhd_F C))$$

$$(A \rhd_F B) \leq_4 (A \odot_F B)$$

Filterish Quaternary Semantics of Attack Trees

Question:

Can we define a quaternary semantics that is complete for all of the filter properties?

PL Theory for Threat Analysis

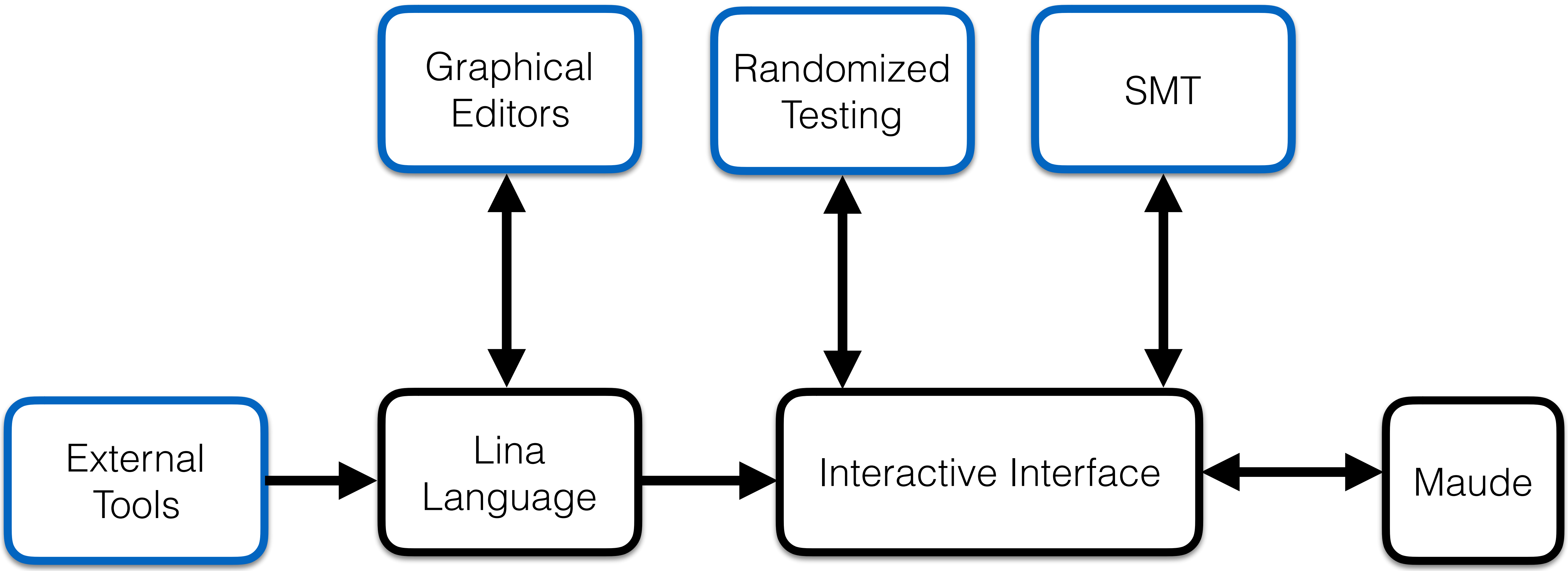
Question:

Can we use the theory of programming languages to build new (semi-)automated tools for conducting threat analysis in a semantically valid way?

Lina: An EDSL for Threat Analysis

- Embedded Domain Specific Functional Programming Languages
  - Host Language: Haskell
- Compositional Attack Tree Specification Language
- Automated Reasoning about Attack Trees using Maude (Quaternary Semantics) and SMT
- Open Source and Available on Github: https://github.com/MonoidalAttackTrees/Lina

Lina: Now and Later



Note: Blue nodes correspond to future additions.

How can Lina be Used?

- By humans: manually code and conduct threat analysis.
  - Interactive querying interface.
- By machines: as a target for other threat analysis tools; for example, after autogenerating attack trees.

Lina: An EDSL for Threat Analysis

```haskell
import Lina.AttackTree

vehicle_attack :: APAttackTree Double String
vehicle_attack = start_PAT $
  or_node "Autonomous Vehicle Attack"
    (seq_node "External Sensor Attack"
       (base_wa 0.2 "Modify Street Signs to Cause Wreck")
       (and_node "Social Engineering Attack"
          (base_wa 0.6 "Pose as Mechanic")
          (base_wa 0.1 "Install Malware")))
    (seq_node "Over Night Attack"
       (base_wa 0.05 "Find Address where Car is Stored")
       (seq_node "Compromise Vehicle"
          (or_node "Break In"
             (base_wa 0.8 "Break Window")
             (base_wa 0.5 "Disable Door Alarm/Locks"))
          (base_wa 0.1 "Install Malware")))
```

Lina: An EDSL for Threat Analysis

```
se_attack :: APAttackTree Double String
se_attack = start_PAT $
  and_node "social engineering attack"
     (base_wa 0.6 "pose as mechanic")
     (base_wa 0.1 "install malware")
```

```
bi_attack :: APAttackTree Double String
bi_attack = start_PAT $
  or_node "break in"
     (base_wa 0.8 "break window")
     (base_wa 0.5 "disable door alarm/locks")
```

```
cv_attack :: APAttackTree Double String
cv_attack = start_PAT $
  seq_node "compromise vehicle"
    (insert bi_attack)
    (base_wa 0.1 "install malware")
```

```
es_attack :: APAttackTree Double String
es_attack = start_PAT $
  seq_node "external sensor attack"
       (base_wa 0.2 "modify street signs to cause
                     wreck")
       (insert se_attack)
```

```
on_attack :: APAttackTree Double String
on_attack = start_PAT $
  seq_node "overnight attack"
     (base_wa 0.05 "Find address where car
                    is stored")
     (insert cv_attack)
```

```
vehicle_attack'' :: APAttackTree Double String
vehicle_attack'' = start_PAT $
  or_node "Autonomous Vehicle Attack"
    (insert es_attack)
    (insert on_attack)
```

Lina: An EDSL for Threat Analysis

```
-- Internal Attack Tree
data IAT where
    Base :: ID -> IAT
    OR   :: ID -> IAT -> IAT -> IAT
    AND  :: ID -> IAT -> IAT -> IAT
    SEQ  :: ID -> IAT -> IAT -> IAT
```

Lina: An EDSL for Threat Analysis

```haskell
-- Attributed Process Attack Tree
data APAttackTree attribute label = APAttackTree {
  process_tree :: IAT,
  labels :: B.Bimap label ID,
  attributes :: M.Map ID attribute
}
```

Lina: An EDSL for Threat Analysis

```
-- Full Attack Tree
data AttackTree attribute label = AttackTree {
      ap_tree :: APAttackTree attribute label,
      configuration :: Conf attribute
}
```

Lina: An EDSL for Threat Analysis

```
data Conf attribute = (Ord attribute) => Conf {
  orOp  :: attribute -> attribute -> attribute,
  andOp :: attribute -> attribute -> attribute,
  seqOp :: attribute -> attribute -> attribute
}
```

Lina: An EDSL for Threat Analysis

```
-- Full Attack Tree
data AttackTree attribute label = AttackTree {
      ap_tree :: APAttackTree attribute label,
      configuration :: Conf attribute
}
```

Lina: An EDSL for Threat Analysis

- Query Attack Trees for:
  - Most Likely Attack
  - Least Likely Attack
  - Set of all Attacks
- Prove Properties of Attack Trees using Logical Theory:
  - Equivalence of Attack Trees
  - Specializations

Lina: An EDSL for Threat Analysis

```
> :load source/Lina/Examples/VehicleAttack.hs
...
Ok, modules loaded
> get_attacks $ vehicle_AT
...
```

Lina: An EDSL for Threat Analysis

```
SEQ("external sensor attack",0.6)
        ("modify street signs to cause wreck",0.2)
        (AND("social engineering attack",0.6)
                ("pose as mechanic",0.6)
                ("install malware",0.1))
```

```
SEQ("over night attack",0.8)
        ("Find address where car is stored",0.05)
        (SEQ("compromise vehicle",0.8)
                ("break window",0.8)
                ("install malware",0.1))
```

```
SEQ("over night attack",0.5)
        ("Find address where car is stored",0.05)
        (SEQ("compromise vehicle",0.5)
                ("disable door alarm/locks",0.5)
                ("install malware",0.1))
```

Future Work

Two new new formal model of causal attack trees:

- Petri Net Model
    - Categorically shown that this is a model of linear logic
- Causal Attack Tree Expressions
    - Attack trees as “regular” expressions in Pomset automata based in the concurrent Kleene algebra

Future Work

- Attack Trees as Comonads?
- Developing a benchmarking library using random generation of attack trees via QuickCheck.
    - Randomized Property based testing of threat analysis algorithms
    - Generate large trees during testing

Takeaways

- Attack Trees can be modeled as **formulas in resource-sensitive logics.**
- Analysis of Attack Trees can be **automated** using their logical semantics.
- **Lina** is a functional programming language that supports this new perspective.