Advancing PL Based Formal Methods Research and Education

Harley Eades III
Computer Science
School of Computer and Cyber Sciences
Augusta University

Who is this guy?

- Ph.D. - Theoretical Computer Science, University of Iowa, 2014
- Thesis: The Semantic Analysis of Advanced Programming Languages
- Now: Research Faculty at Augusta University

Research Interests

- Computational Logic
- Foundations of Programming Languages
- Software Verification
- Interactive/Automated Theorem Proving
- Pure and Applied Mathematics

Overall Research Goals

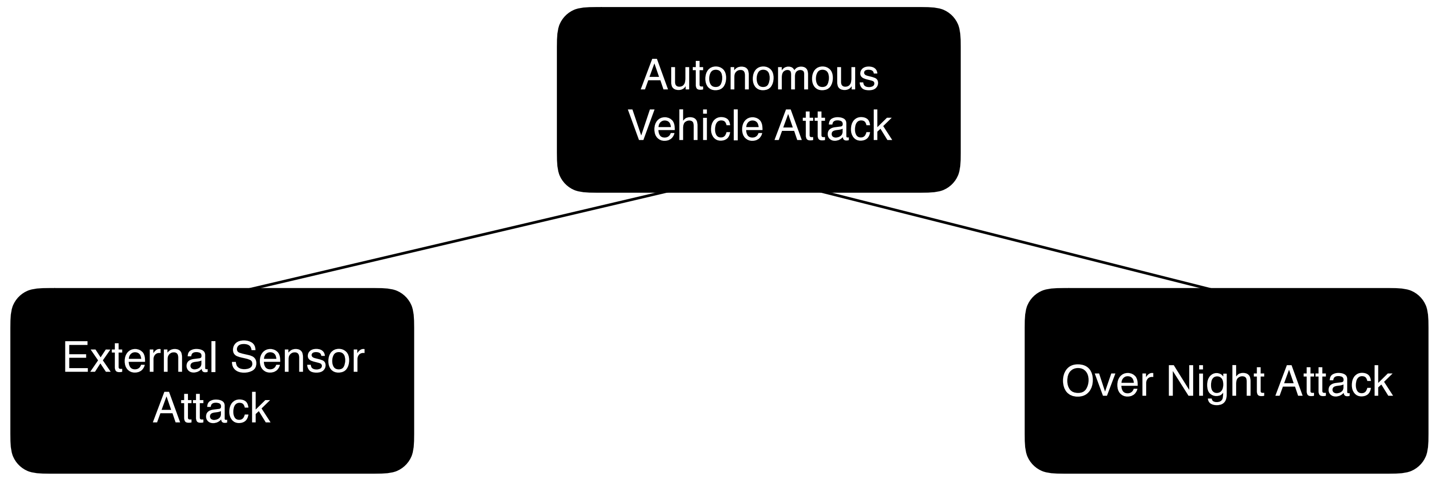
Advance the theory of programming languages and interactive theorem proving so that it is more applicable to real-world problems.

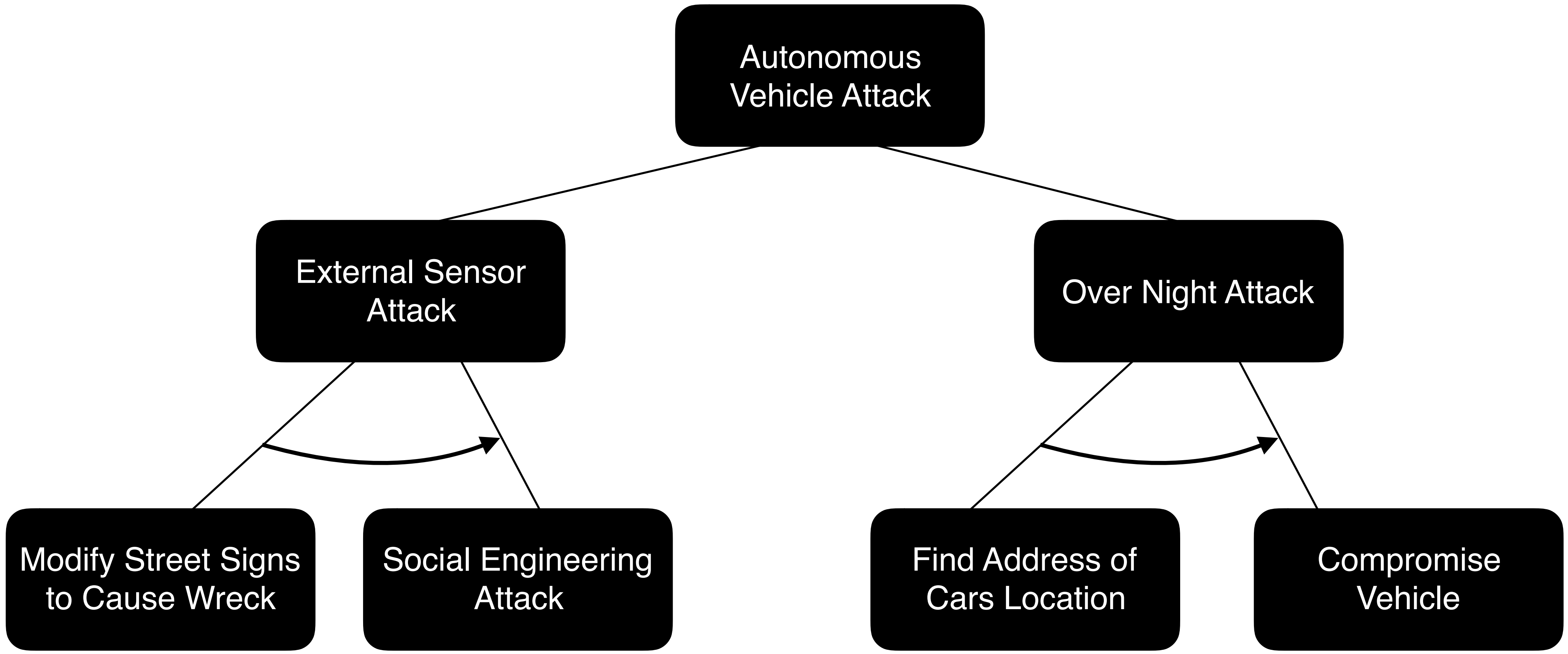
Overall Research Goals

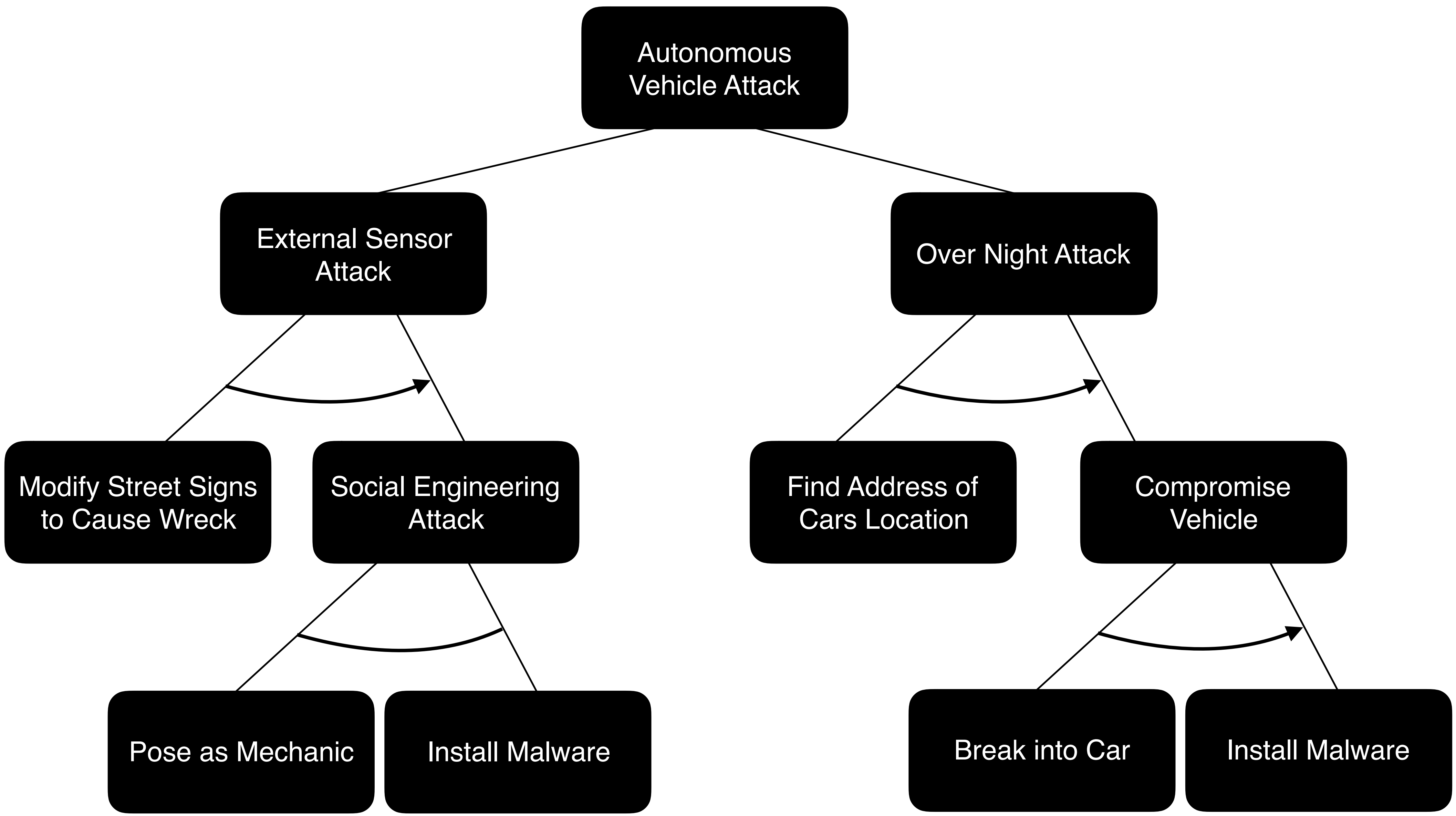
Applying the theory of programming languages and interactive theorem proving to new areas of computer science.

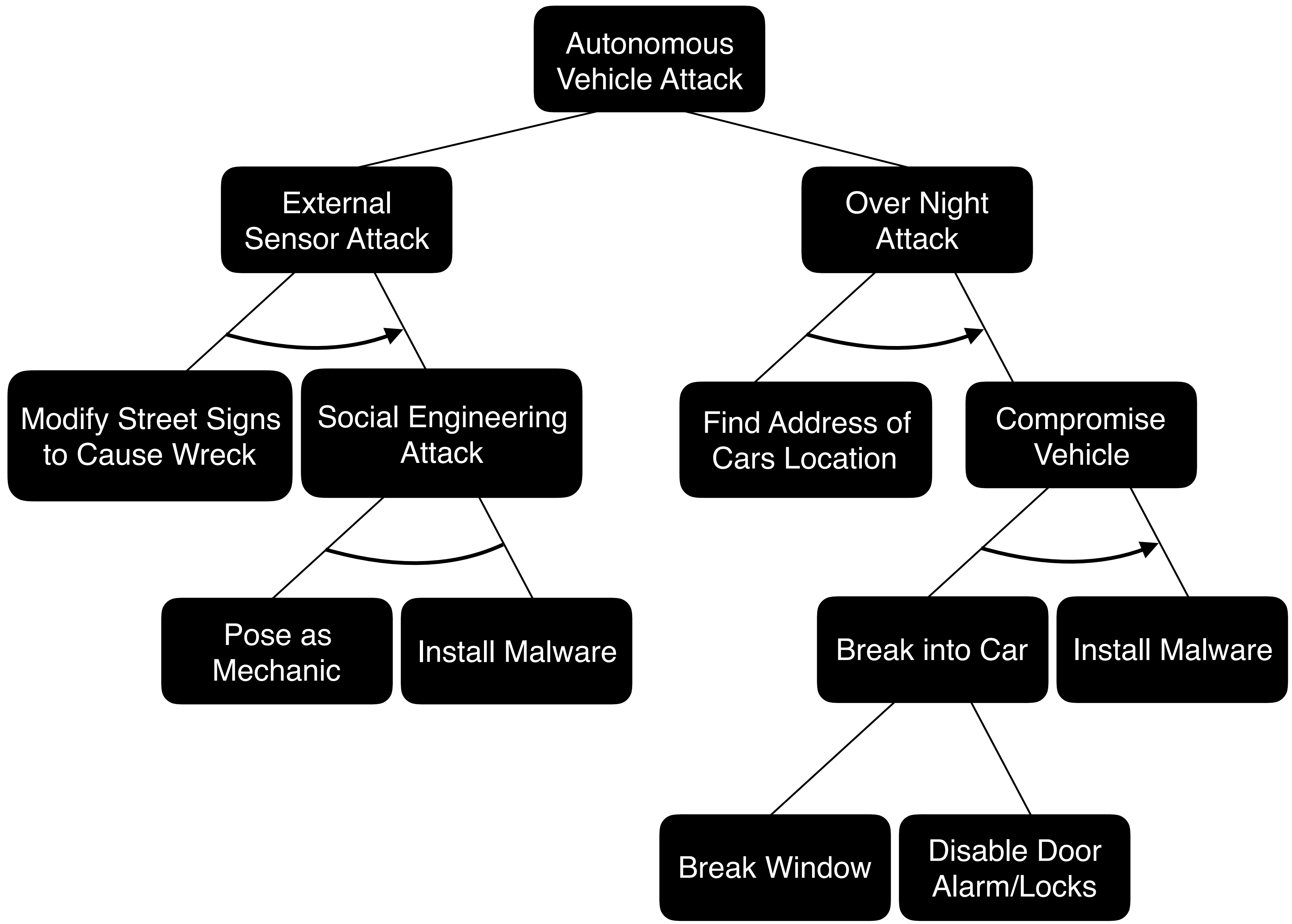
Threat Analysis using Attack Trees

Autonomous Vehicle Attack

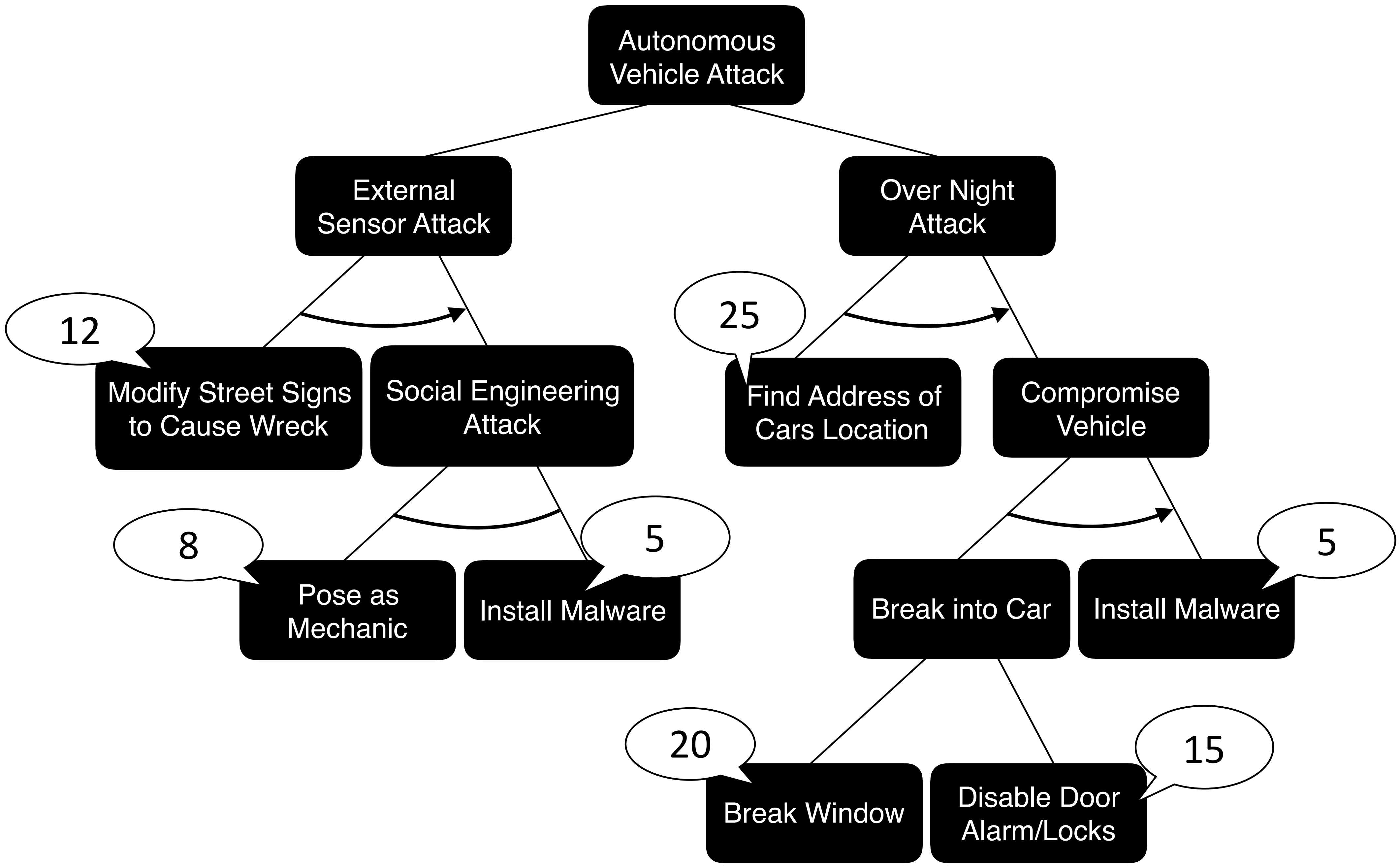

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack

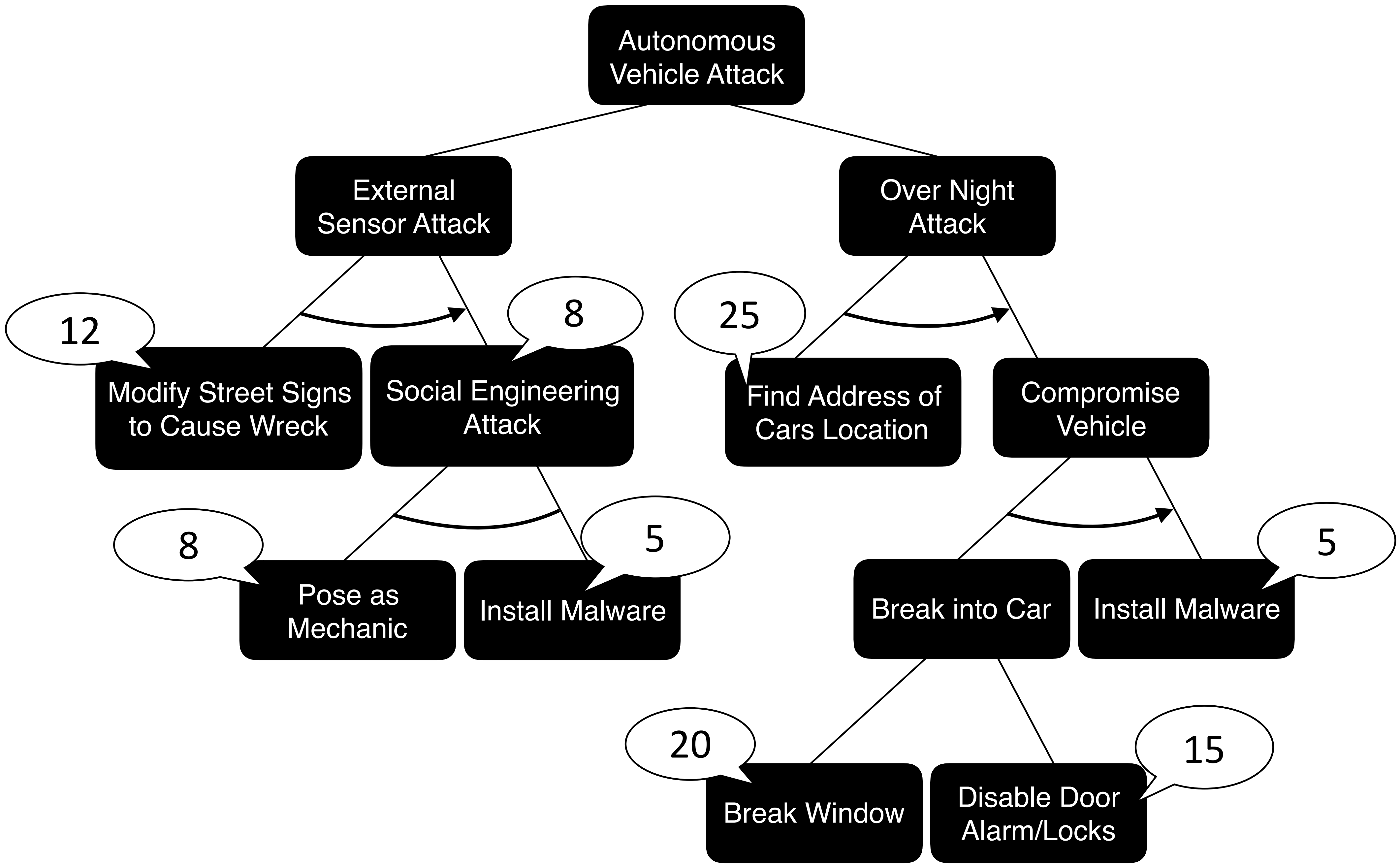

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle

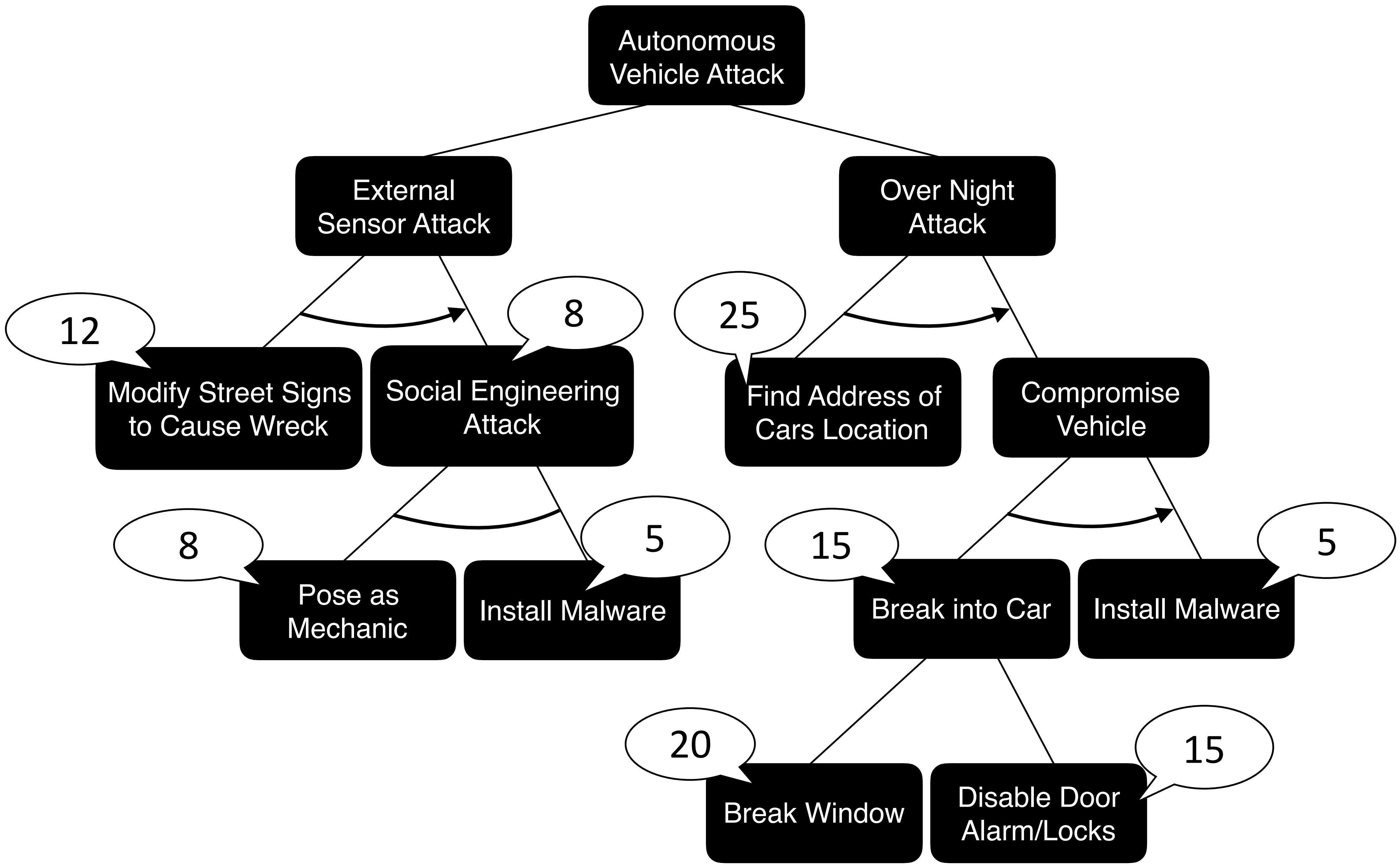

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware



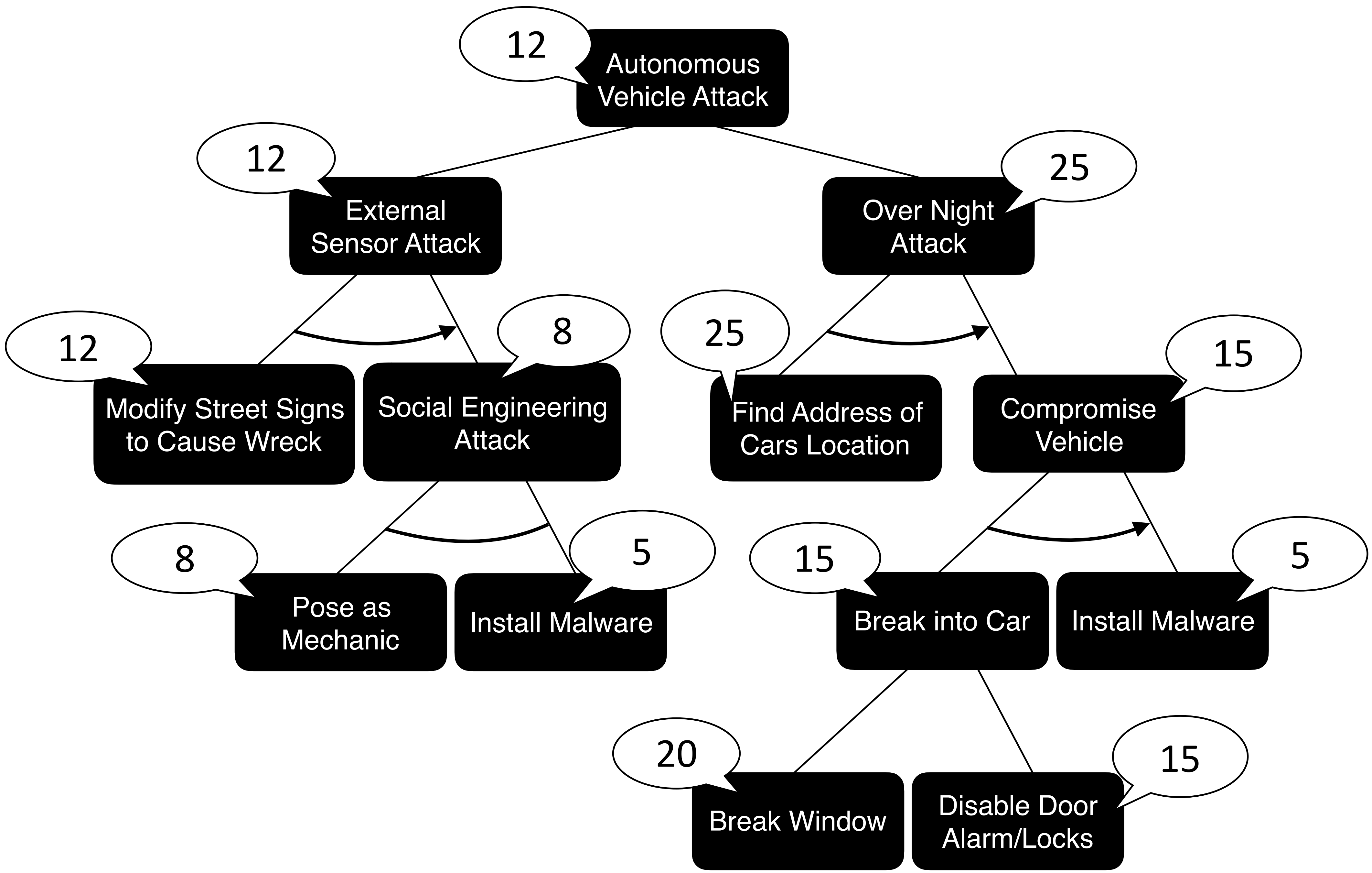
Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks

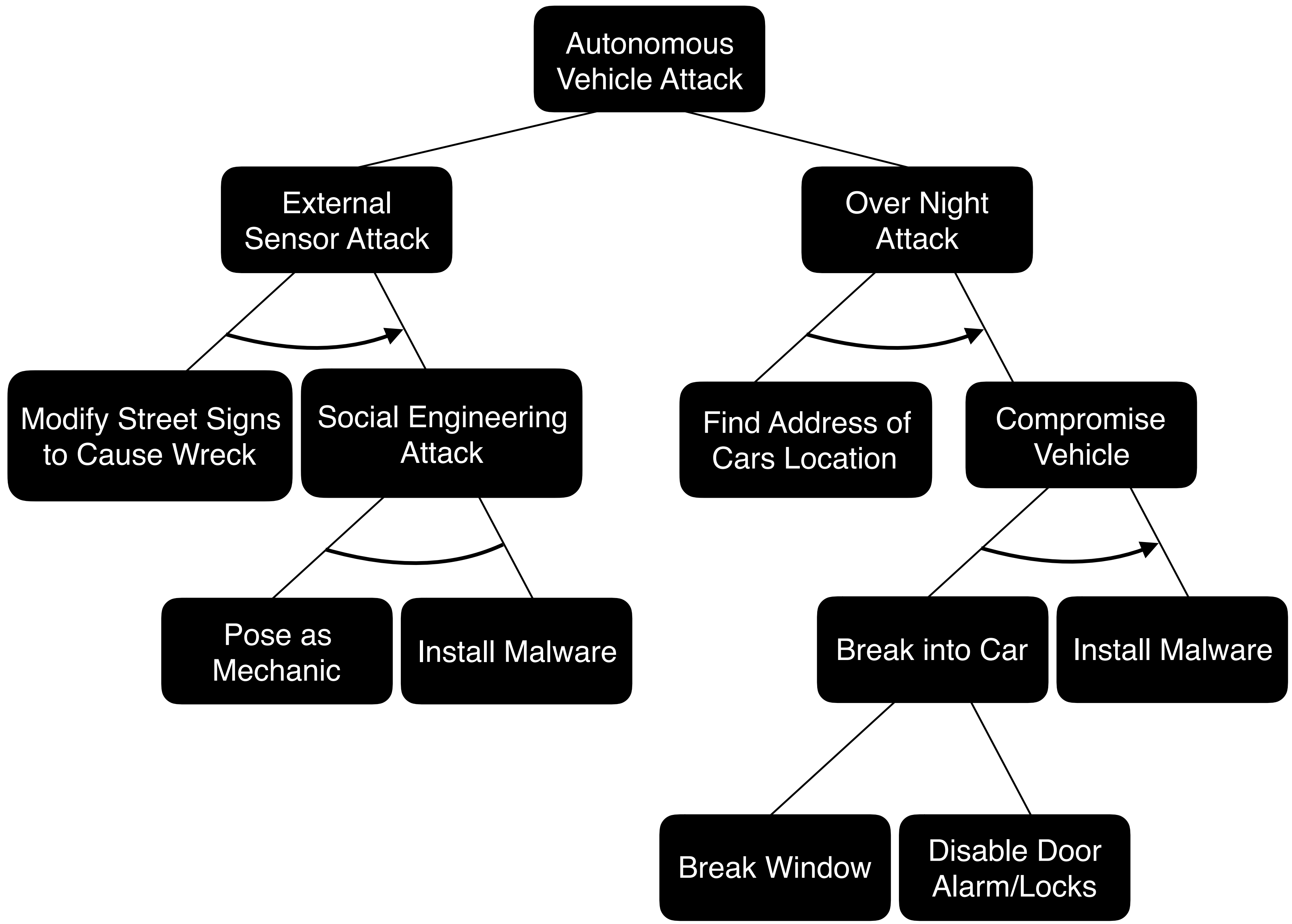

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks

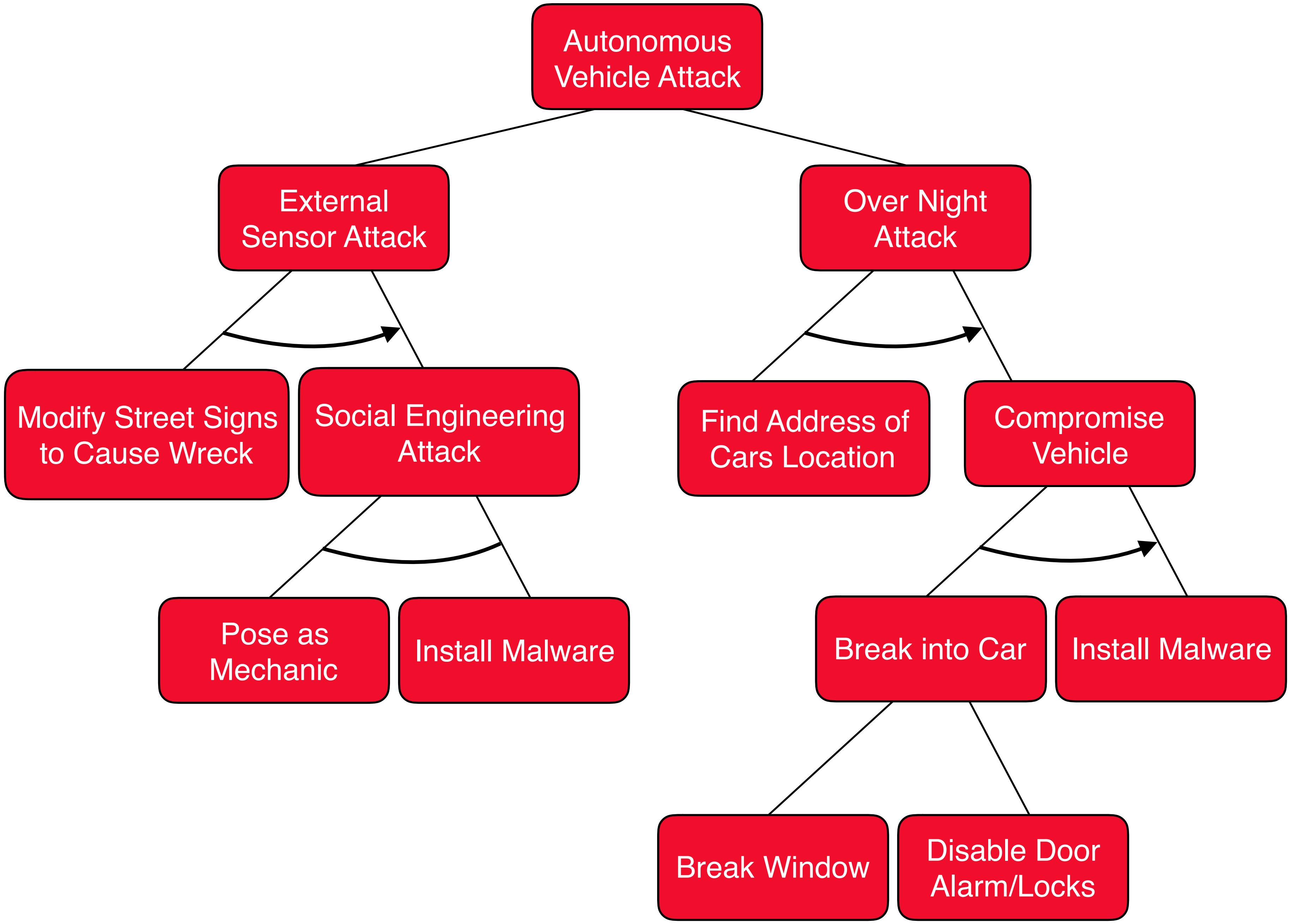

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks



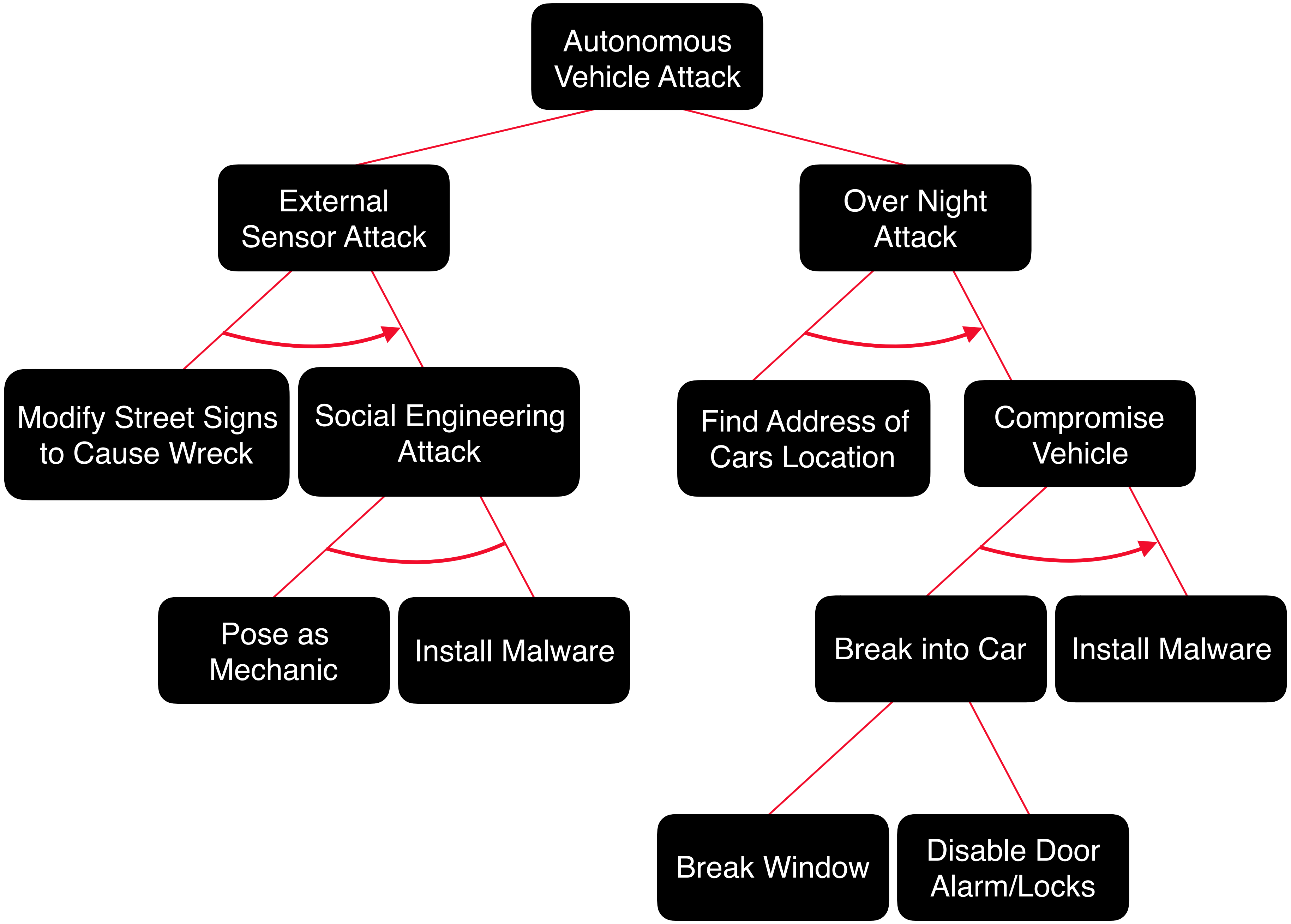
Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
15
Break into Car
5
Install Malware
20
Break Window
15
Disable Door Alarm/Locks

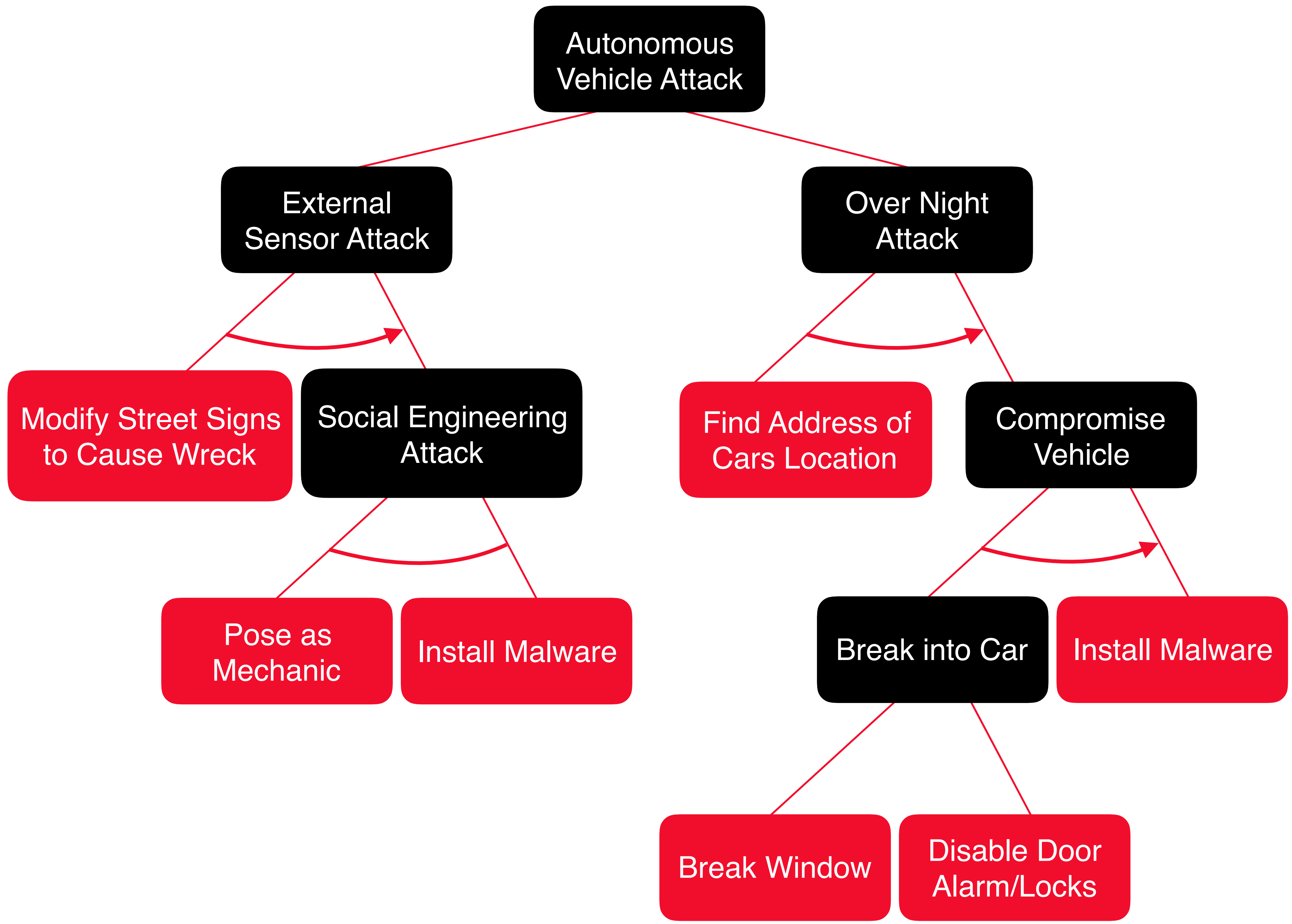

12
Autonomous Vehicle Attack
12
External Sensor Attack
25
Over Night Attack
12
Modify Street Signs to Cause Wreck
8
Social Engineering Attack
25
Find Address of Cars Location
15
Compromise Vehicle
8
Pose as Mechanic
5
Install Malware
15
Break into Car
5
Install Malware
20
Break Window
15
Disable Door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

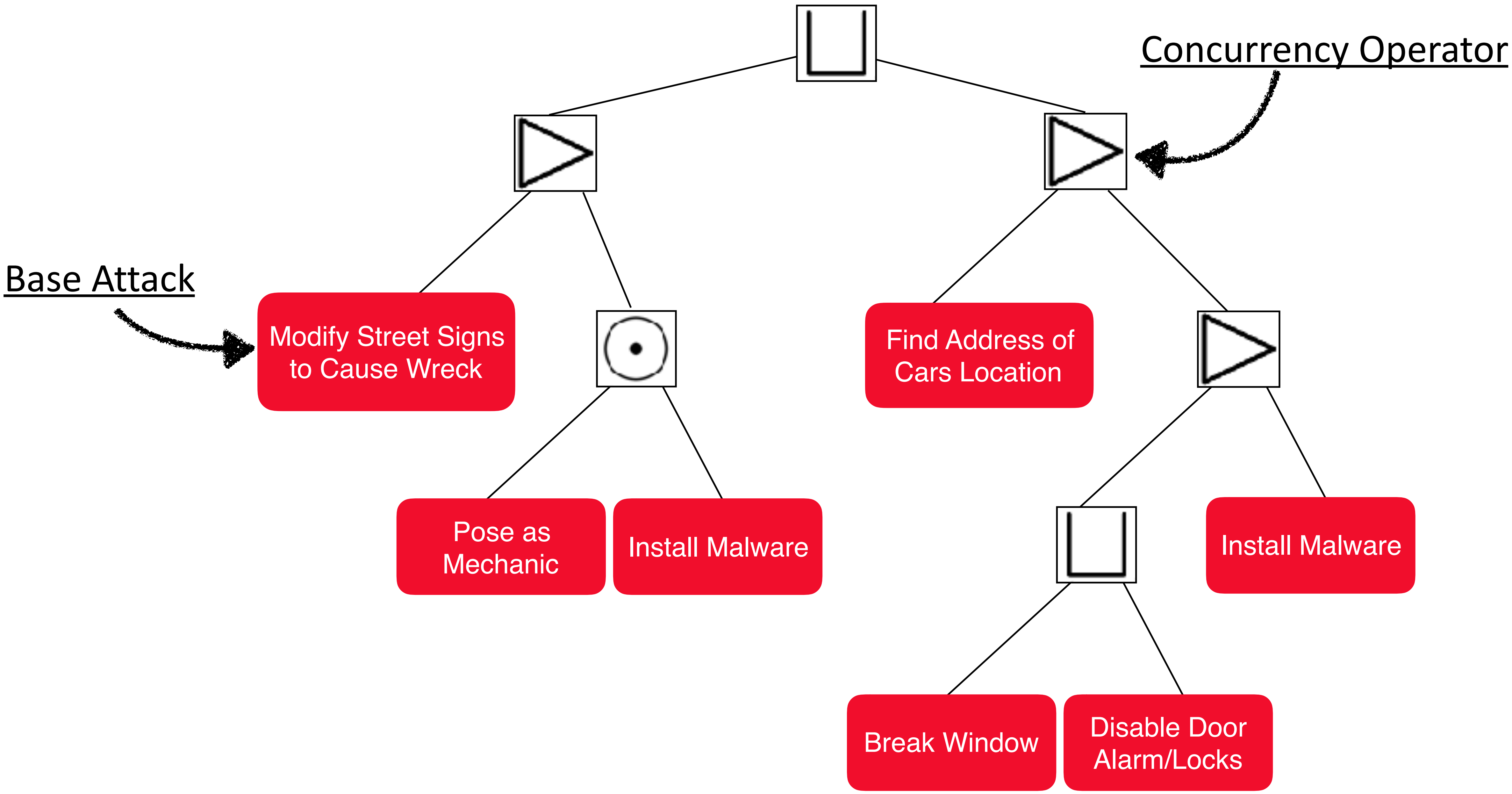

Autonomous Vehicle Attack
External Sensor Attack
Over Night Attack
Modify Street Signs to Cause Wreck
Social Engineering Attack
Find Address of Cars Location
Compromise Vehicle
Pose as Mechanic
Install Malware
Break into Car
Install Malware
Break Window
Disable Door Alarm/Locks



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
Pose as Mechanic
Install Malware
Find Address of Cars Location
Break Window
Disable Door Alarm/Locks
Install Malware

$A =$ "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

$A =$ "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"

$$
\begin{aligned}
&(A \rhd (B \odot C)) \sqcup (D \rhd ((E \sqcup F) \rhd C)) \\
\equiv\ &((A \rhd B) \odot (A \rhd C)) \sqcup ((D \rhd (E \rhd C)) \sqcup (D \rhd (F \rhd C)))
\end{aligned}
$$

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 and SMT
- Open Source and Available on Github: https://github.com/MonoidalAttackTrees/Lina

Lina: An EDSL for Threat Analysis

```
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))
```

Lina in the Future

- Attack Trees as Comonads?
- Developing a benchmarking library using random generation of attack trees via QuickCheck.

Takeaways

- Attack Trees are used to assess threat of security critical systems
- Attack Trees are **process trees**.
- 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.

Resource-Sensitive Dependent Types

Joint Work with:

Dominic Orchard and Vilem Liepelt, University of Kent

Resource-Sensitive Logics

- Resource-Sensitive Logics = Substructural Logics
  - Linear, Affine, Contractive, Non-commutative Logic
- Limit how hypothesis (variables) are used to control resources
  - Control structural rules for exchange, weakening and contraction

The Structural Rules

$$\frac{\Gamma_1, x : A, y : B, \Gamma_2 \vdash t : C}{\Gamma_1, y : B, x : A, \Gamma_2 \vdash t : C}\ \mathrm{EX}$$

The Structural Rules

$$\frac{\Gamma_1, \Gamma_2 \vdash t : B}{\Gamma_1, x : A, \Gamma_2 \vdash t : B}\ \text{WEAK}$$

The Structural Rules

$$\frac{\Gamma_1, x : A, y : A, \Gamma_2 \vdash t : B}{\Gamma_1, x : A, \Gamma_2 \vdash [x/y]t : B} \textsc{contract}$$

Resource-Sensitive Logics

- Lambek Calculus = STLC - Ex - Weak - Contract
- Linear Logic = STLC - Weak - Contract
- Affine Logic = STLC - Contract
- Contractive Logic = STLC - Weak

Resource-Sensitive Logics

- Linear Logic = Lambek Calculus + Ex
- Affine Logic = Linear Logic + Weak
- Contractive Logic = Linear Logic + Contract
- STLC = Linear Logic + Weak + Contract

What Types of Resources?

Examples:

- Memory consumption,
- State-based systems,
- Time complexity, etc.

Dependent Types

$$\forall (l_1\, l_2\, l_3 : \mathsf{List}\, A) \to ((l_1 +\!\!+\, l_2) +\!\!+\, l_3) \equiv (l_1 +\!\!+ (l_2 +\!\!+\, l_3))$$

Dependent Types

- Write programs and prove them correct in the same language.
  - Specifications for programs are sets of dependent types.
  - Writing programs with these dependent types is equivalent to proving each property in the specification.
  - Type checking these programs machine checks your proofs.

Dependent Types

Not resource sensitive; has all of the structural rules!

Resource-Sensitive Dependent Types

Generalize Linear Logic to a Dependent-Type System

Easier said than done!

$$\mathsf{id} : (A : \mathsf{Type}) \to (x : A) \to A$$

$$\mathsf{id}\, A\, x = x$$

Resource-Sensitive Dependent Types

Naive linear dependent type theory is unusable.

Resource-Sensitive Dependent Types

We need an mechanism to relax the system when we want.

Resource-Sensitive Dependent Types

Our Solution:

Naive Linear Dependent Type Theory

+

Graded Modalities

Resource-Sensitive Dependent Types

Graded Modalities: programmer precisely controls the usage of variables.

$$\mathsf{id} : (A : \mathsf{Type}) \to (x : A) \to A$$

$$\mathsf{id}\, A\, x = x$$

$$\mathsf{id} : (|A| : \mathsf{Type}\,|2 : 0|) \to (x : A\,|0 : 1|) \to A$$

$$\mathsf{id}\; A\,|x| = x$$

Type Level Usage

$$\mathsf{id} : (|A| : \mathsf{Type}\,|2 : 0|) \to (x : A\,|0 : 1|) \to A$$

Program Level Usage

$$\mathsf{id}\ A\,|x| = x$$

Education

Overall Education Goals

Incorporating formal-methods reasoning principles and techniques into the primary - university CS education.

Overall Education Goals

Exploiting formal-methods research to develop new education tools to make learning and teaching easier for students and educators respectively.

The Pull CS Back Initiative

The Pull CS Back Initiative

The goal is to assistant CS primary school through secondary school educators with little CS background incorporate CS topics into their curriculum.

The Pull CS Back Initiative

Masters Degree:

- Broadly introduce educators to CS topics and its pedagogy.
- Fast: One year
- Collaboration between CS department and college of education.

The Pull CS Back Initiative

Pullback Seminar:

- An inclusive environment anyone can participate in to learn about CS education topics.
- Open to the public
- Free!
- A way for non-university educators to keep learning about CS.

Education Tools

Disco Lang

- A language designed to bring functional programming and formal methods into discrete mathematics.
- Syntax must be based on prior mathematical knowledge.
- Good errors messages are extremely important.
- Joint work with Brent Yorgey, Hendrix College.

```
implication : B -> B -> B
implication x y =
  {? false   if x and not y,
     true    otherwise
  ?}
```

Haskell QuickGrader

- An auto grader for Haskell assignments.
- Grading is done using the QuickCheck library.
- Incorporated into a Gitlab server.
- Students just push on solution branch to trigger grading, and report is generated and pushed back.