Laint: Static Taint Tracking using Linear Types

Machiry Aravind Kumar

UCSB

1 Taint Tracking

Any variable that can be modified/provided by an outside user poses a potential security risk, if used unverified in an program. Example: strcpy of an user provided data without checking for length. Taint Tracking tracks these variables inside a program and warns if used in places where untainted or safe data is expected.
This can be done statically or dynamically (by instrumenting the program).
Any taint tracking system needs following information:

- Taint Introduction.

  How taint is introduced into the program. Ex: function that reads input from user.

- Taint Policy

  Functions or Points in program where corresponding variables should be checked for taint and appropriate action must be taken.

- Taint Propagation

  Rules on how taint propagates in the program.

2 Goal

Viewing static taint tracking as a type checker for linear type system and implementing the same.

3 Types

3.1 Linear Types or (Semi-Linear Types)

- Taint (or taint)

  This type signifies something which is unverified and could be potentially malicious.

3.2 Non Linear Types

- Un Taint (or untaint)

  This type signifies something which is verified and safe.

3.3 Typing Rules

There are special functions called **Sanitizers**, which have atleast one parameter of type taint and return untaint value.

- Taint Introduction: Explicit declaration or return from function can introduce or change the variable type to taint.
- Taint Policy: All expression used in if must be of untainted type
  This is to avoid any side channel attacks.
- Taint Policy: Non Linear types can be converted into linear types but not the other way round.
- Taint Propagation: Any expression involving taint type will get taint type, expect for Sanitizers
- Taint Policy : Sanitizers consume taint arguments, a program cannot be able to use a tainted variable used as an argument to a sanitizer function.