Static transformation inserts new security code into existing source code, Taint identifying systems must maintain information about taint or the bounds of the pointer. This identifies points at which untrusted input is used in the program in a dangerous manner.

Requirements for either system

1. Metadata
2. Maintenance
3. Usage

Taint Tracking -
Runtime taint example:

```c
void foo(void){
    int x, y;
    read(stdin, &x, sizeof(x);
    tagmap[&x>>2]=0xFF
    y=x;
    tagmap[&y>>2]=tagmap[&x>>2]
    bar(y)
}
void bar(int z){
    dangerous_op(z)
}
```
- The taint information is passed to the tagmap during assignments to maintain a level of protection.
- When bar is called the information about the taint must be sent as well.

Taint tracking using global: (colors represent above functions and global).

```
Global
char tagmap[2^30]; // This is a byte array

- The entries will be marked 0 if the memory is not tainted
- The entries will be marked 1 if the memory is tainted.

char tagstack[2^11];
int count = 0
tagstack[count++]=tagmap[\&y>>2];
tagmap[\&z>>2]=tagstack[count--];
if(tagmap[\&z>>2]) abort();
dangerous_op(z)
```

- Alternative to abort() in the function you can validate the information.
  - If (tagmap[\&z>>2]) check_policy(\&z) ← This is the validation check that could be inserted instead of the abort function.

Evaluate the costs:

1. Execution time.
2. Memory overhead.
3. Effectiveness.
4. Programmer effort.

- Tagmap is allocated dynamically so it only takes a small amount of memory.

Tagmap has a 25% overhead.
Optimizations:

1. Dynamic allocation of tagmap.
2. remove address of operations.
   a. Make a new variable char taint_y;
   b. Store taint info, then use tagstack from local variable. tagstack[count++] = taint_y;

Protection of tagmap:

- Tagmap is located in memory;
  o tagmap [&p>>2]=0;
  o *p=0;

  We need to make sure if p points to the tagmap the second line fails.

- Unmap tagmap portion that corresponds to the tagmap itself since it will never be tainted, the first line will cause a segfault.

Problems

- Untransformed libraries don’t follow taint info.
  o False negative.
- Tainted turned to untained but not noted.
  o False positives.

Policy authoring

- Uses markup language sets up rules so that there is only one literal node.