TOPIC: STATIC CHECKING FOR SECURITY

ADVANTAGES of Static Checking

- Bugs can be caught and removed before deployment of Software.
- No runtime overhead
- Better coverage when compared to testing
- Reduces testing time
- Not Prone to human error

DOWNSIDE of Static Checking

- Requires accurate specifications
- Requires access to libraries: will not work in case of propriety libraries
- Tends to be language specific and system dependent (Same bugs may not occur on different systems hence adaption is needed for specifications)
- Can be computationally slow

CQUAL vs METAL

- Deals with data of program
- It is Sound technique (0 false negatives)
- Higher rate of false positive
- It can detect only one type of bug (i.e. type mismatch)
- CQUAL is able to analyze program completely (Global analysis)
- Library dependent. CQUAL cannot operate if library is not available (proprietary)

- Deals with order of statements and control flow
- Unsound technique for bug detection
- Lower false positives
- It can detect many types of bugs by analyzing control flow of program
- Local analysis (one function at a time)
- Does not depend on libraries: It analyses kernel modules (which are self contained)
CQual employs a technique where the programmer specifies constraints on data flow using ‘types’. The following examples use ‘tainted’ (Variables used by untrusted adversary) and ‘untainted’ (All other values) qualifiers.

E.g.

```c
int printf(untainted char *,.....)
tainted char *s;
printf(s); // INVALID
```

This indicates the following sub typing relations:

untainted int < tainted int

In case of function calls:

Eg.

```c
int printf(untainted char *,.....)
tainted char *s;
printf(s); // INVALID
```

```c
tprintf(tainted char*,...)
tprintf("hello"); // VALID
```
Type Inference

Tagging any program greater than 1K LOC can be a menial task. Therefore the inference model is used. In this model, the user introduces a small number of annotations at key places in the program, and CQUAL infers the types of the other expressions in the program.

Eg.

tainted int x;
untainted int y;
$Q_z$ int z; // $Q_z$ is the inferred type of z
z=x; // here z becomes tainted as its value is derived from a tainted source \($tainted=Q_z\$\)
y=z; // y becomes tainted as well \($Q_z = $untainted\$\)

Hence,
untainted int <= $Q_z$ int <= tainted int

Eg. 2

char * strcpy(char *dst, char *src)
{
    while (*src)
        *dst++=*src++;
    return tmp;
}

$tainted$ char *s;
char *t,*y;
y=strcpy(t,s);
printf(y);
Directed Graph indicating subtype relations

If there is a path from tainted to untainted \( \Rightarrow \) program may have bug
i.e. \( \text{tainted} \leq Q_z \leq \text{untainted} \) // error !!!

If no path \( \Rightarrow \) no bug

The following shows a structure of a purging function to resolve SQL injection attack

```c
$\text{untainted char *} \text{sanitize} (\text{tainted char *} s) {
    // clean up t
    return (\text{untainted char *} )t;
}
```