28-Sep-10

**Static Analysis for Security**

*Disadvantages:*
- # requires specifications
- # libraries: if you use libraries and don’t have access to their codes, then you can’t use Cqual
- # language specific
- # slow

*Advantages:*
- # no runtime overhead
- # catch bugs early (at compile time)
- # better coverage (extremely high)
- # reduced testing time
- # low errors - system specific

<table>
<thead>
<tr>
<th></th>
<th>Cqual</th>
<th>Metal/MECA/xg++/Coverity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>works more on data</td>
<td>works more on control flow</td>
</tr>
<tr>
<td>2</td>
<td>sound</td>
<td>unsound</td>
</tr>
<tr>
<td>3</td>
<td>high False positives</td>
<td>lower False Positives</td>
</tr>
<tr>
<td>4</td>
<td>finds one type of bug</td>
<td>finds many types of bugs.</td>
</tr>
<tr>
<td>5</td>
<td>covers the whole program</td>
<td>does local analysis (on functions, etc)</td>
</tr>
</tbody>
</table>

**CQUAL**

- $\text{tainted}$
- $\text{untainted}$

```c
$\text{tainted \ int \ x;}$
$\text{untainted \ int \ y;}$

y=x; //ERROR
x=y; //OK
```

```c
\text{int \ printf \ (\text{untainted \ char \ *, \ ...})}
\text{tprintf \ (\text{untainted \ char \ *, \ ...})}$
```

```c
$\text{tprintf} \ (s);$ //ERROR
```

```c
\text{tprintf} \ (\text{"hello"}); \text{ //OK}$
```

$Q_z$ int $z$;
$z = x$; // $Q_z = \text{tainted}$
$y = z$; // $\text{untainted} = Q_z$

which infers,
$\text{untainted} = Q_z = \text{tainted}$ // WRONG

read()
scanf()
$tainted$
Program Source

$\text{untainted} <= \text{tainted}$

$y = z$;
$Q_z <= \text{untainted}$
$Q_z$ int $<= \text{untainted}$ int

$z = x$;
$tainted <= Q_z$
$tainted$ int $<= Q_z$ int

Make a graph:

If there is a path from tainted to untainted, there is a BUG.

Therefore,
path = may be a BUG
no path = no BUG

Sanitizing your outputs helps operations in the next system as you have untainted inputs for it.