CSE 409/509 Computer System Security

- **Static Analysis for security**

<table>
<thead>
<tr>
<th>cqual</th>
<th>vs metal/MECA/xg++/Coverity</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>control</td>
</tr>
<tr>
<td>sound</td>
<td>unsound</td>
</tr>
<tr>
<td>high false positives</td>
<td>lower false positives</td>
</tr>
<tr>
<td>one type bug</td>
<td>many kinds of bugs</td>
</tr>
<tr>
<td>global</td>
<td>local</td>
</tr>
</tbody>
</table>

* soundness = won’t miss bugs
  = no false negatives

- **Static Analysis**
  + Catch bugs early
  + No runtime overhead
  + (extremely) High coverage
  + Reduced testing time
  + Low error
  - Requires specifications
  - Requires libraries
  - Language specific
  - System specific

- **cqual**
  the simplest rule
  types must match

ex)
$ tainted int x;
$ untainted int y;
y = x; //ERROR!!
x = y; //OK

ex)
int printf ($untainted char *s ...);
int tprintf ($tainted char *t ...);
$tainted char *s;
...
printf (s); //NOT ALLOWED
tprintf ("Hello"); //ALLOWED
• Imagine……
  $tainted\ int\ x = 5;
  $untainted\ int\ y;
  $Qz\ int\ z;
  y = z;
  z = x;

\[
\frac{\text{\$Qz} \leq \text{\$untainted}}{	ext{\$Qz int} \leq \text{\$untainted int}} \quad \frac{\text{\$tainted} \leq \text{\$Qz}}{	ext{\$tainted int} \leq \text{\$Qz int}}
\]

$Qz \leq \text{\$untainted}
$Qz \leq \text{\$tainted}

ex) char * strcpy (char *dst, char *src)
{
    while (*src)
        *dst++ = *src++;
}
$tainted\ char\ *s;
      char\ *t;
strcpy\ (t, s);
printf\ (t);

Path = maybe bugs
No Path = no bugs
* Solution – Sanitize

`$untainted char * Sanitize ($tainted char *t)
{
    ...
    // Clean Up t
    ...
    return ($untainted char *) t;
}

→ applied to: SQL Injection, Cross-site Scripting (sanitize outputs)