Software Security

Taint analysis

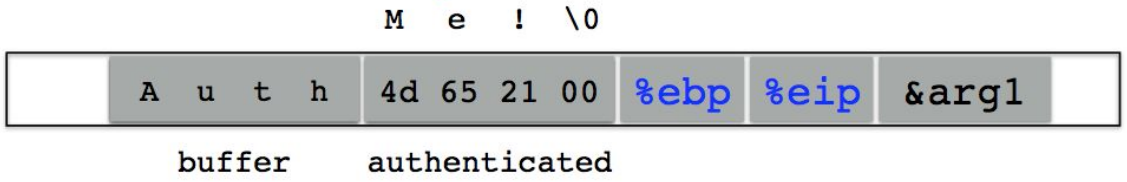
Alessandra Gorla

institute imdea software

Buffer overflow vulnerability

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) { ...
}

int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Code still runs; user now 'authenticated'



Injections

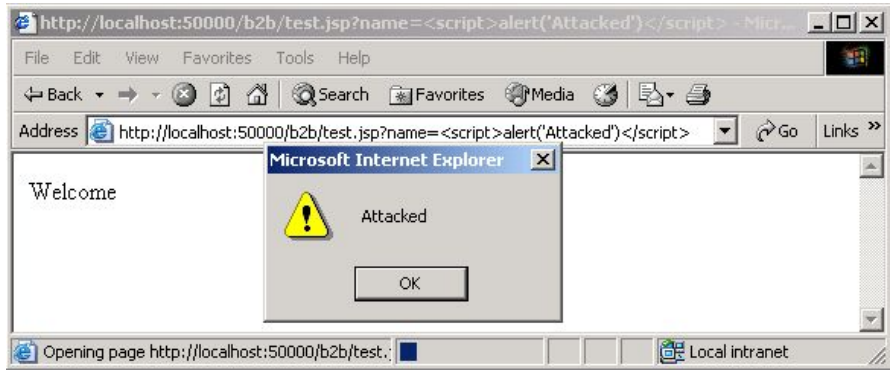
- Injection attacks trick an application into including unintended commands in the data sent to an interpreter.
- Interpreters
  - Interpret strings as commands.
  - Ex: SQL, shell (cmd.exe, bash), LDAP, XPath
- Key Idea
  - Input data from the application is executed as code by the interpreter.

SQL injection

```
$link = mysql_connect($DB_HOST, $DB_USERNAME, $DB_PASSWORD) or die ("Couldn't connect: " .
mysql_error());

mysql_select_db($DB_DATABASE);

$query = "select count(*) from users where username = '$username' and password = '$password‘ ";

$result = mysql_query($query);
```

A simple example

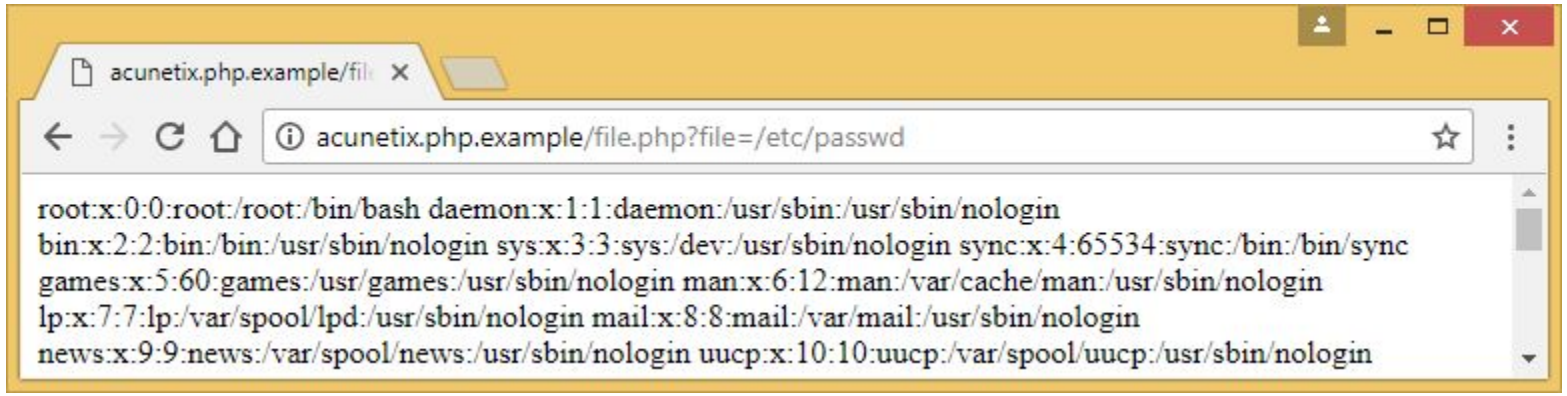
http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>




```
<HTML>
<Body>
Welcome <script>alert("Attacked")</script>
</Body>
</HTML>
```

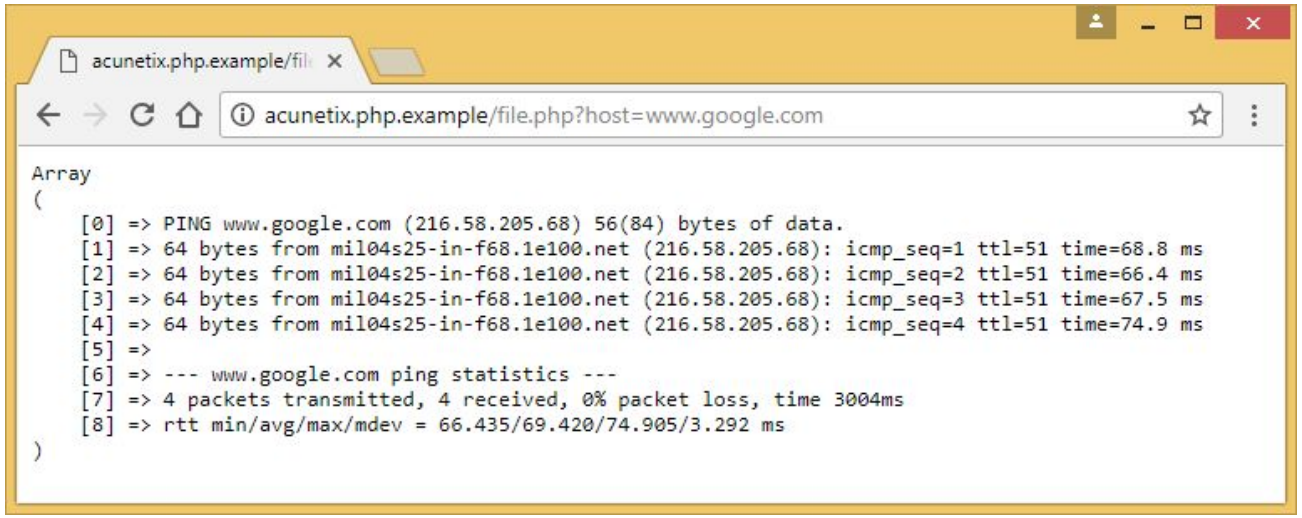
Directory traversal

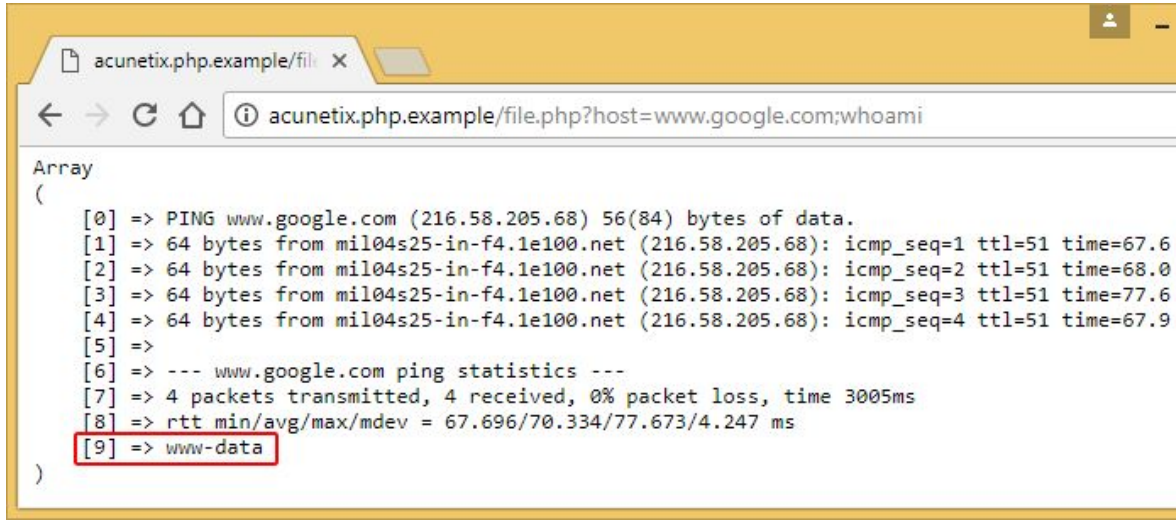
```
$file = $_GET['file'];

include($file);
```



Command injection

```
exec("ping -c 4 " . $_GET['host'], $output);
echo "&ltpre>";
print_r($output);
 echo "&lt/pre>";
```





Common issue?

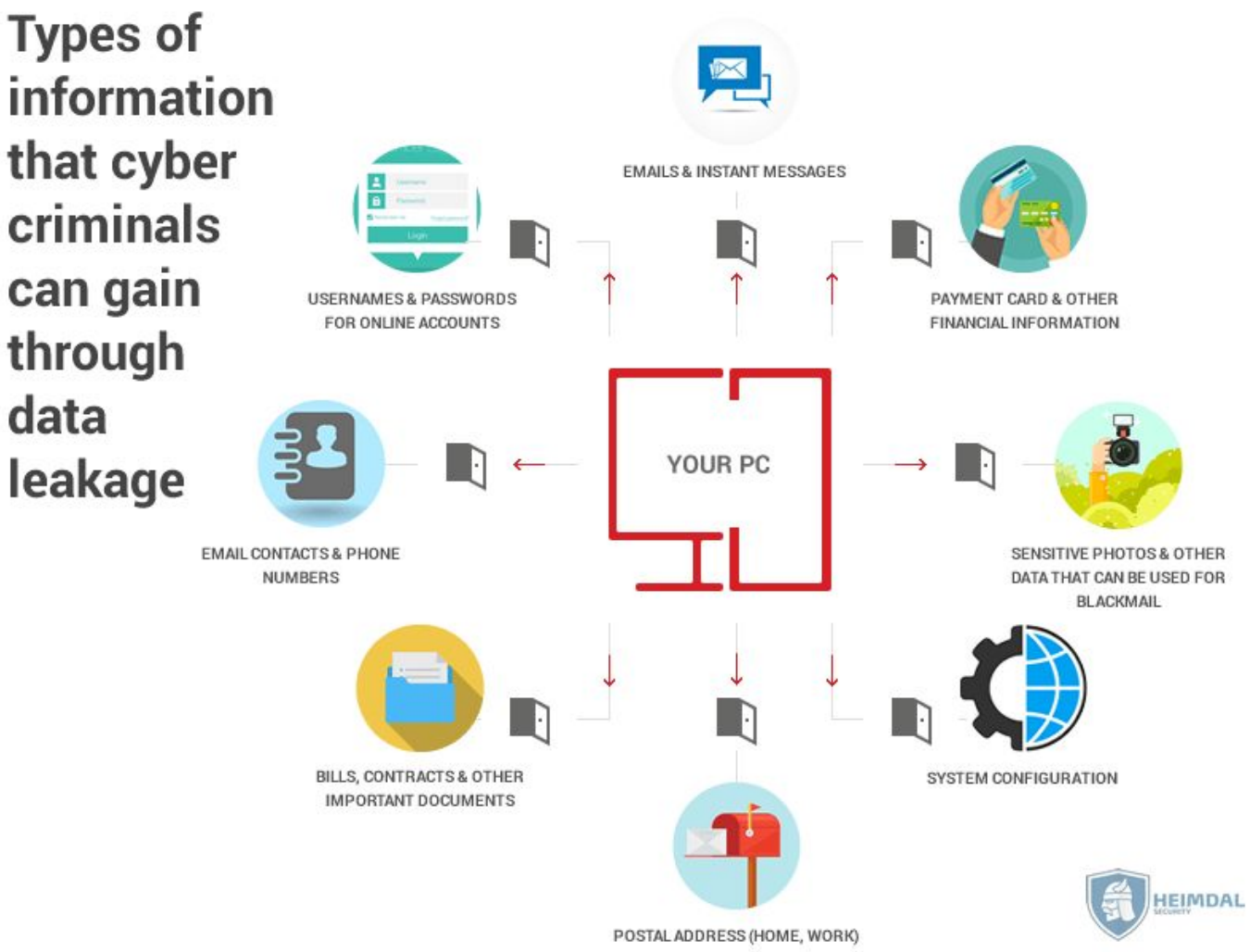
Input from user is used directly *without any sanitisation*

Can we automatically find these issues?

YES! With **taint analysis**

- Information flow analysis.
- Used in the security domain.
  - Tracking how user input flows to critical statements (e.g. sql execution statements)
  - Tracking how private information flows through the program and if it is leaked to public observers.

Information leak



Example

```
1. input = get_input();
2. tmp = “select…”  + input;
3. query(tmp);
4. send_internet(tmp);
```

Simple rules

Untainted	+	Untainted	=	Untainted
Untainted	+	Tainted	=	Tainted
Tainted	+	Tainted	=	Tainted

Example

```
1. input = get_input();
2. tmp = "select…"  + input;
3. query(tmp);
4. send_internet(tmp);
```

possible SQL injection
possible sensitive information leaked

Terminology

- Sources
  - Private data of interest / user inputs
- Sinks
  - Locations of interest / critical statements
  - Check taints of incoming information
  - Determines if there is a leak in the program/ if there is a possible injection

Example

```
1. input = Source();
2. tmp = "select…"  + input;
3. Sink(tmp);
4. Sink(tmp);
```

possible SQL injection
possible sensitive information leak

Example

```
1. input = Source();
2. tmp = "select…"  + input;
3. tmp = encode(tmp)
4. Sink(tmp);
```

Dynamic Taint Analysis

Dynamic Taint Analysis

Track what are the taints that are influencing the values of the program

```
1. x = get_input();
2. y = 1;
3. z = x;
4. w = y + z;
5. exec("SELECT … "+w);
```

Example

```
1. x  = Source(0);
2. y  = 1;
3. z  = x;
4. w  = y + z;
5. Sink(w);
```

x ⟶ 0→T

Is there a warning?

```
x = Source(0);
y = x;
if(y == 0) {
    z = 2
}
else {
    z = 1
}
Sink(z);
```

Implicit flows

- Tainted data affects the value of another variable indirectly.
- Needed for sound analysis.

Implicit flows

```
x = Source(0);
y = x;
if(y == 0) {
    z = 2
}
else {
    z = 1
}
Sink(z);
```

Explicit information flow

Implicit information flow

Implicit flows

```
x = Source(0);
y = x;
if(y == 0) {
    z = 2
}
else {
    z = 1
}
Sink(z);
```

Limits of Dynamic Analysis

Results are input dependent.

Implicit flows needed for sound analysis, but difficult to track.

Static Taint Analysis

Static Taint Analysis

- Track, at each instruction, what are the taints that are influencing the variables of the program.

Example

```
1. x  = Source();
2. y  = 1;
3. z  = x;
4. w  = y + z;
5. Sink(w);
```

x→T

x→T

x→T, z→T

x→T, z→T, w→T

x→T, z→T, w→T

Dynamic - static

```
x = Source();
y = x;
if(y == 0) {
    z = 2
}
else {
    Sink(y);
}
```

Limitations of static analysis

- Do not know what values might cause the leak.
- Overtainting

e.g.

```
1. x = Source(args[0]);
2. Object o = foo();
3. v = o.equals(x);
```

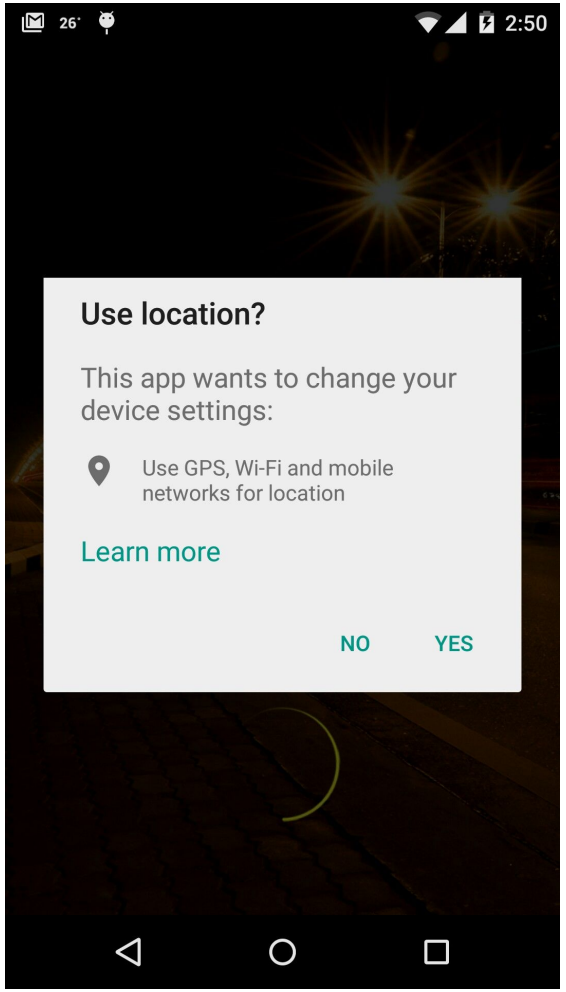
All implementations of equals should be analyzed!

Combining static and dynamic analyses

- Dynamic analysis may miss some information
- Static analysis may report false alarms
- Best solution: Combine them when possible!

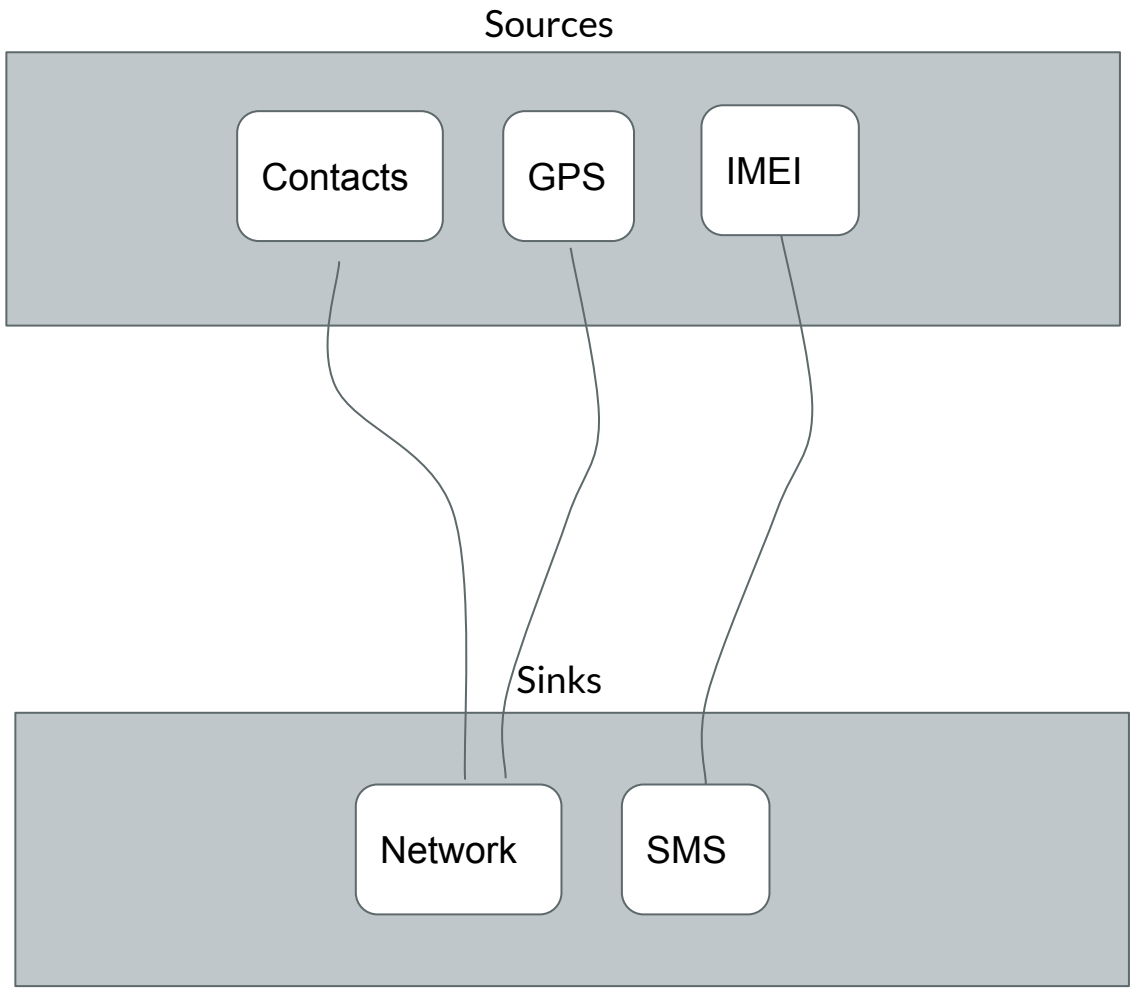
Taint Analysis for Privacy

Taint analysis for mobile apps



- Should I grant the permission?
- What does the app do with this information?
- Does it send my location somewhere?

Taint analysis for mobile apps



Sources: where data comes from

Sinks: where data flows to

How to do that in Android?

```
deviceInfo = AndroidAPI.getDeviceInformation();
gpsInfo = AndroidAPI.getGPSInformation();
```

…

```
Network.send(deviceInfo.IMEI, gpsInfo.Location);
```