# CSE 509 Lecture 20

Prof. R. Sekar, Scribe:Arun Rathakrishnan

November 6, 2013

## 1 Blackbox Taint Tracking Technique

Fine grained dynamic taint tracking requires source code instrumentation and policies setup at each function call. There is significant runtime overhead as a result, the approach is intrusive and requires a special complier to maintain, the tag bits and shadow stack. The alternative is to intercept the HTTP requests as input to the webserver and consider requests from webserver like sending a HTTP response or issuing a query to a database server as output.

### 1.1 Comparison of Input and Output

In this approach, the HTTP request is intercepted by plugins for Apache server, which tracks a session ID to match a request message with another output message. The data in the request is collected and represented as name-value pairs, which is used by the rest of the system. Any output message from the webserver is matched by the session ID and the system checks if the output and input messages have similarity. This represents the taintedness of the message, which indicates the amount of control, the user exerts on the program behaviour. We can do this by checking if, the input has a common substring or subsequence with the output. The latter approach is more robust.

### 1.2 SQL/Command Injection

```
$cmd = "SELECT price FROM products WHERE name = '" . $name . "'";
```

An attacker usually injects SQL code, by introducing a new SQL statement with a semicolon.

```
$cmd = "SELECT price FROM products WHERE name = '" . $name . "'";
```

The name is usually read from the HTTP request, which is typically passed from a user input from a form. The user can input the below instead of a name and can set prices in products table to 0.

```
name = 'xyz'; UPDATE products SET price = 0 WHERE true
```

This can be prevented by scanning the user input and looking for malicious characters like ';'. But the identification is not that simple.

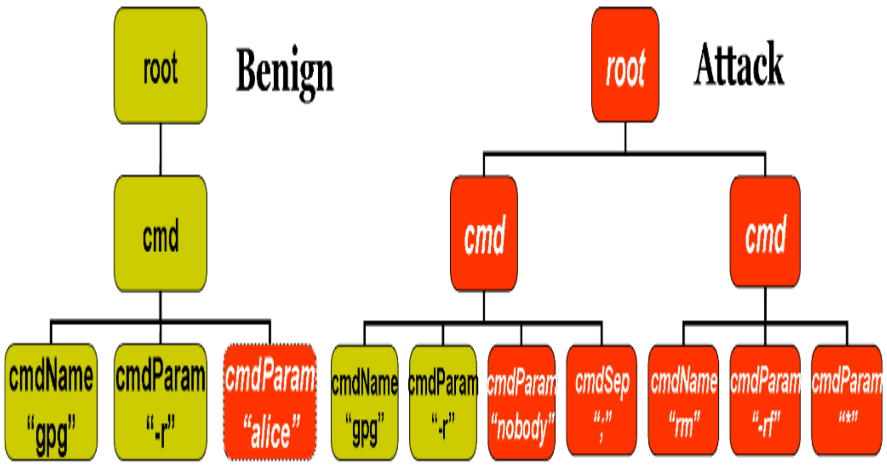
### 1.3 Taint Tracking And Policies

The system needs to determine,

- Determine how much control is exerted by the user, through taint tracking.
- If the contorl exerted is reasonable, by setting up policies.

## 2 Syntax And Taint Aware Policy

The presence of ';' as a statement delimiter generally means that, the tainted output message possibly contains an SQL injection. But if ';' occurs within quotes or in any form other than as a statement delimiter, the message is valid. Thus we can say that simple string matching can not enforce a strong policy.



Syntatic Structure captures Command Injection [1]

### 2.1 Syntatic Structure

An attack typically causes a change in syntatic structure of the query being executed as shown above. In general, if the input spans more than one token, and if the tainted tokens correspond to SQL or shell commands we can say that an injection attack has been attempted. Thus knowledge of syntax and taintedness helps detect attacks in general. An application and language independent approach to this problem is undertaken in [1].

### 2.2 Buffer Overflow

In addition to injection attacks, taint tracking can detect memory errors like buffer overflow. This can be done by checking if any code pointer (function pointer or return address) is changed before calling or making a jump to another instruction. If the code pointer is tainted, it means that user input has been written into the location, thus detecting buffer overflows and stack smashing accurately.

## 3 Static Analysis: Model Checking

A model captures the behaviour of a system. A program can be represented as a model. Model checking statically find errors in a program given a set of property specifications. Property specifications can describe expected behaviour or deviations from expected behaviour. From a security perspectve we are interested in two types of properties rules of correctness or violations of correctness.

### 3.1 Example: Root Privilege in FTP Server

A FTP server process for an authenticated user needs root privileges for certain actions like binding to a new port. The privlege must be acquired before the bind operation and must be relinquished immediately after it.

```
old_id = geteuid();
setuid(0); \\Acquire root privilege
bind();
setuid(old_id);
```

After acquiring root privilege the process should not perform any other system call. For example, it should not call the system function.

```
old_id = geteuid();
setuid(0); \\Acquire root privilege
system(input);
setuid(old_id);
```

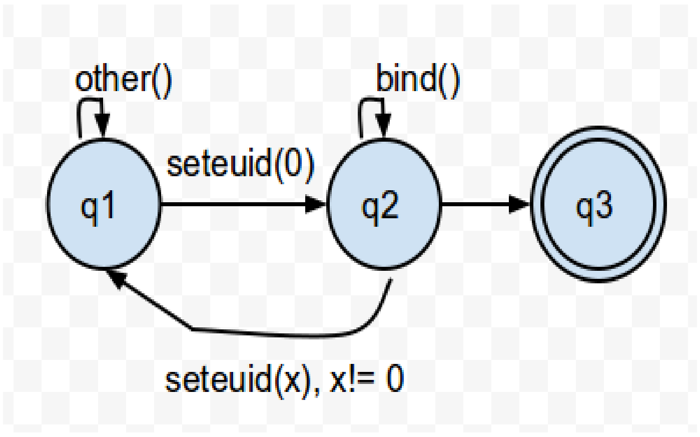
### 3.2 Models and Properties

Models can be represented Finite State Machine, Context Free Grammar, etc. Here we consider only the FSM. Even though the model is finite, a small memory of 100 bytes can represent $2^{800}$ states. There is an explosion of statespace and its important to keep the statespace small.

Properties can be classified into two,

- Saftey: bad things never happen.
- Liveness: good things eventually happen.

Liveness can not be modelled FSM. They can be represented by models like Buchi automata.

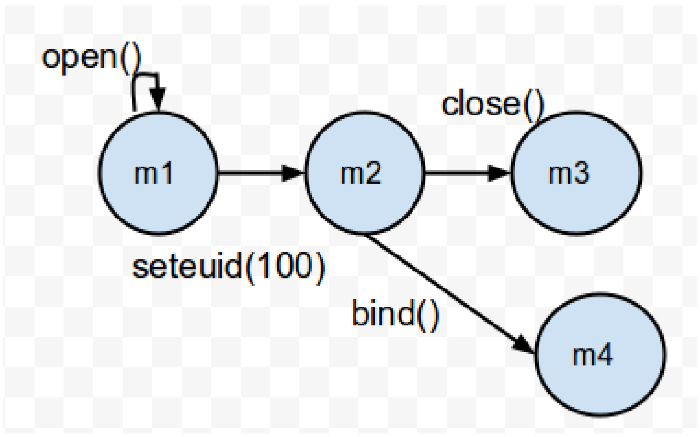


FSM For violation of root previlege

The above FSM captures a bad property discussed in the above example, namely misuse of root privilege by FTP Server process.

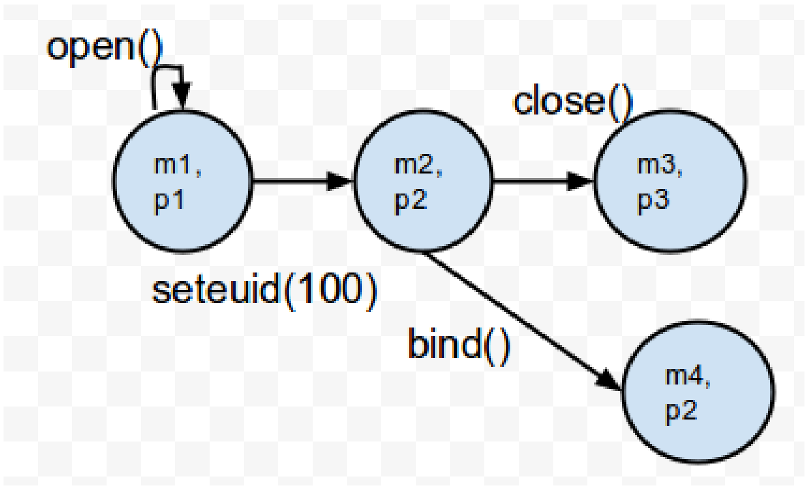
## 3.3 Analysis

Given a model $M$, and a property $P$, model checking involves finding if $M \cap P$ has a path that leads to a bad state in $P$. Then the model is said to have an error.



Model for FTP server process

In model checking, the model $M$ for a program can be checked against a set of properties to determine if the program does not violate some conditions encoded by the properties.



Model checking FTP Server

## 4 Reference

R. Sekar, "An Efficient Black-box Technique for Defeating Web Application Attacks".