Taint Tracking & Static Analysis (September 5, 2012)

Scribe: Wei Wu

September 6, 2012

1 Real-life Injection Vulnerability Applications

1.1 Cap‘n Crunch hack

- Cap‘n Crunch cereal whistle toy used to hack phone system to make calls for free
- Whistle blows at 2600 Hz, which is the exact tone that central office sends back to indicate a quarter has been dropped into pay phone =¿ can make phone calls
- Flaw in design: two kinds of data (both phone calls and tones that control the phone channel) were flowing through same line

1.2 Modems

- In order to send instructions to modem, type in terminal commands
- Type +++, and anything following that is interpreted as modem instruction, i.e. +++ATH/0
- Can get people to unknowingly send +++ instructions (i.e., send email to user, then user might reply to email, which may contain +++ in the text and transmit it over the network)

1.3 xterm–vt100

- Server can send escape codes that would display special things on terminal window
- Put the following chars into the buffer, and send it back to me as if user typed them in
- If server interprets this as anything user typed in, then this command can do anything

2 Injection Vulnerability Examples

2.1 Command-line injection

`system("mail $emailaddr")` Suppose `$emailaddr = foo@bar.com; /bin/rm -rf /`

2.2 Format string vulnerability

`printf(s);`

- `s="%d"` will print the next item on the stack. A malicious user can print all vars from the stack.

- `%n` will actually store a value at a variable, which can also be leveraged maliciously

subsectionSQL Injection `query("select * from users where username = $u")` Suppose `$u = foo; drop table user;`

3 Defending Against Injection Vulnerabilities

- Prepared statements for SQL: a query is declared and precompiled, such that the SQL library is responsible for making sure inputs are interpreted only as strings. This is an example of separating the data and control channels.
- Input validation: sanitize your inputs to make sure that it cannot be leveraged maliciously. This is hard because hackers are constantly finding new ways to disguise their malicious inputs through escaping characters, etc.
- Parse trees: create a parse tree of the query and execute that instead of raw query text

3.1 Taint tracking as an injection defense

1. Mark any user input as tainted.
2. Make policy in system call/SQL library call that allows arguments to a command to be tainted, but prohibits argument names or ";" to be tainted.

Taint-tracking allows us to track the origin of data as it flows from the application to system layer. Suppose we set up a policy like below for an SQL taint-checker (where underline denotes taintedness):

block() if q = /$\underline{\text{select}}$ — $\underline{\text{where}}$ — $\underline{\text{or}}$ — $\underline{;}$ /;

This is not an adequate policy because it over blocks—i.e.,a user can provide legitimate input that contains the word "where" but our policy disallows the query. Also, this is a blacklist, which generally will not work to catch *everything.*
A better approach would be to parse the query, keep track of what parts of the tree are tainted, and then make a policy controlling where taint is allowed to appear. However, your SQL library and the DB may parse the same query slightly differently. This is a discrepancy that an attacker may take advantage of.

4 Unique Device IDs and Privacy

When a company collects your UDID, they can:

- Create a complete profile of what you do
- Collect your personal identity and other sensitive information about yourself if you link the app with Gmail or Facebook

UDID tracking is more pernicious than cookies because cookies are supposed to be site-specific (in theory)—for example, Google cannot set a cookie that is sent to CNN when the same user visits both sites. Because UDIDs are persistent to a device, a service can track your activity across multiple applications.

4.1 Alleviating UDID-related privacy concerns

- Per-app ID: instead of global UDID, send `appid = Hash(UDID, appname)`
- Throwaway IDs: have IDs with a time-to-live, and generate a new UDID afterwards. This is like being able to clear my cookies."

Mobile OS-makers are not really incentivized to implement this because they want a thriving marketplace where a lot of app makers rely on advertising to make money

4.2 Ways to detect UDID tracking

- Network sniffer to check if any packets are sending out UDIDs
- Man in the middle surveillance on SSL-encrypted traffic
- Taint-tracking, as done in TaintDroid

4.3 Notice and consent

- Notice: Did you provide information to the user that data will be gathered?
- Consent: Did you ask the user if this is okay? Did they approve of this?