taint tracking

taint tracking idea

a lot of attacks — user data ends up in bad place

one way to diagnose/mitigate — track flow of user data

one possibility: *dynamic taint tracking*

some values ‘tainted’

using them to compute other values ‘taints’ them, too

taint tracking implementations

for the programmer:

- supported as optional langauge feature — Perl, Ruby
- doesn't seem to have gotten wide adoption?

for the malware analyst/user

- as part of a custom x86 VM (whole system, on machine code)
- as part of a custom Android system
- ...

taint tracking in Perl (1)

```
#! perl -T
# -T: enable taint tracking
use warnings; use strict;
$ENV{PATH} = '/usr/bin:/bin';

print "Enter name: ";
my $name = readline(STDIN);
my $dir = $name . "-dir";

system("mkdir $dir");
```

“Insecure dependency in system while running with -T switch at perltaint.pl line 10, <STDIN> line 1.”

taint tracking in Perl (2)

```
#! perl -T
# -T: enable taint tracking
use warnings; use strict;
$ENV{PATH} = '/usr/bin:/bin';

print "Enter name: ";
my $name = readline(STDIN);
# keep $name only if its all alphanumeric
# this marks $name as untainted
($name) = $name =~ /^([a-zA-Z0-9]+)$/;
my $dir = $name . "-dir";

system("mkdir $name");
```

taint tracking for malware analysis

mark contents of file as tainted, then ID how used

- find out if/how data of file gets to output
- track tainted accesses to record 'path' of file data use

figure out where network packet data goes

- mark input as 'tainted'
- identify what functions process packet data
- see where packet data ends up on disk

can 'tag' each byte of input differently

- identify which bytes of input jump depends
- identify which bytes of input malicious command came from

whole-system probably too high overhead to do in realtime

taint tracking assembly

taint-tracking often proposed at *assembly* level

examples:

Panda.RE (2013–??)

along with virtual machine record+replay

Panaroma (Yin and Song, UC Berkeley, CCS '07)

high-level overview

lookup table for each register and byte of memory:

- where did this value come from?

```
add %r9, (%r8):
memory-taint-table[register-values[R8]] =
                  register-taint-table[R9]
```

also similar for virtual disk, network, ...

custom VM: all applications and the OS run with taint tracking

- tracks data moving between programs “for free”

Panaroma special cases

`xor %eax, %eax`: special case: remove taint from %eax

Windows keyboard input did something like:

```
keycode = GetFromKeyboard();
switch (keycode) {
case KEYCODE_A: return 'a';
case KEYCODE_B: return 'b';
...
}
```

defeating ASM-based checking

if a malware author wanted to defeat this taint checking, what ideas seem promising for confusing the analysis?

A. timing arithmetic operations to see if the machine is unusually slow

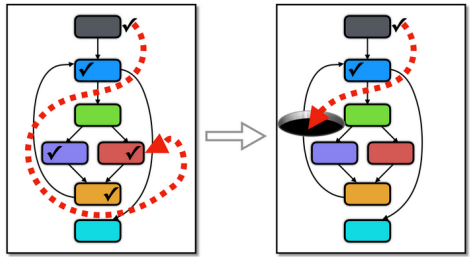
B. computing the hash of the malware's machine code and comparing it to a known value

C. changing `x = y` to `switch (x) { case 1: y = 1; break; case 2: ...}`

D. changing `x = y` to `x = z + y; x = x - z;`

Tigress's transformation

Anti Taint Analysis



The goal of this transformation is to disrupt analysis tools that make use of dynamic taint analysis.

Diversity

We use two basic ways to copy a variable using control-, rather than data-flow:

1. counting up to the value of the variable, and
2. copying it bit by bit, tested in an if-statement.

example: TaintDroid

TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones

William Enck
The Pennsylvania State University

Peter Gilbert
Duke University

Byung-Gon Chun
Intel Labs

Landon P. Cox
Duke University

Jaeyeon Jung
Intel Labs

Patrick McDaniel
The Pennsylvania State University

Anmol N. Sheth
Intel Labs

TaintDroid instrumentation

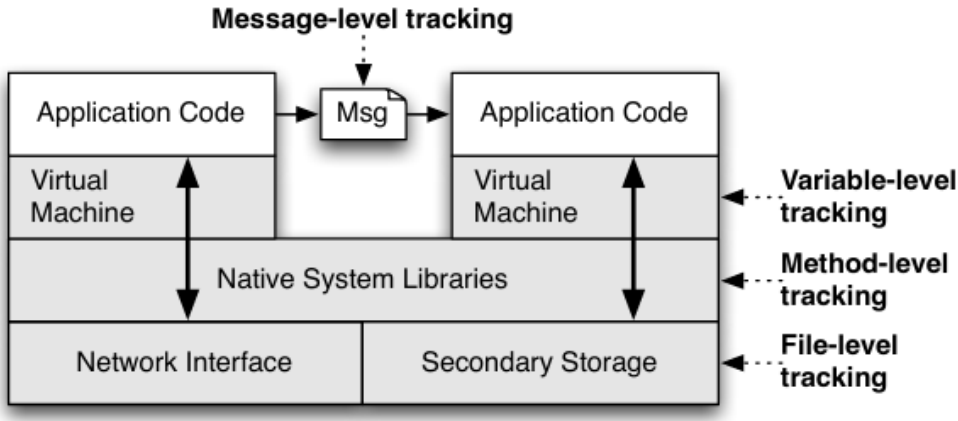


Figure 1: Multi-level approach for performance efficient taint tracking within a common smartphone architecture.

TaintDroid results

Table 3: Potential privacy violations by 20 of the studied applications. Note that three applications had multiple violations, one of which had a violation in all three categories.

Observed Behavior (# of apps)	**Details**
Phone Information to Content Servers (2)	2 apps sent out the phone number, IMSI, and ICC-ID along with the geo-coordinates to the app's content server.
Device ID to Content Servers (7)*	2 Social, 1 Shopping, 1 Reference and three other apps transmitted the IMEI number to the app's content server.
Location to Advertisement Servers (15)	5 apps sent geo-coordinates to ad.qwapi.com, 5 apps to admob.com, 2 apps to ads.mobclix.com (1 sent location both to admob.com and ads.mobclix.com) and 4 apps sent location† to data.flurry.com.

* TaintDroid flagged nine applications in this category, but only seven transmitted the raw IMEI without mentioning such practice in the EULA.

†To the best of our knowledge, the binary messages contained tainted location data (see the discussion below).