NICTA

On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices

Golam Sarwar, **Olivier Mehani**,
Roksana Boreli, Mohamed-Ali Kaafar

SECRYPT 2013, 31 July 2013

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT Trade & Investment
Victoria The Place To Be
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

- Mobile Privacy Threats
- TaintDroid
  - Taint Analysis
  - Limitations
- Attacks Against TaintDroid
  - Description
  - Evaluation
  - Mitigation
- Conclusion

Mobile Privacy Threats

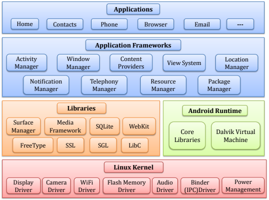
- Mobile devices
  - Always with the user
  - Always on
  - Always connected
- Trove of sensitive data
  - Private details → identity theft
  - Personal habits → profiling
- Third-party applications can access all this data
- Permissions systems easily side-stepped
  - User don't understand/care[1]
  - Developers ask too much (demo available)
  - Colluding applications[2]
- Need more effective systems

[1] A. P. Felt et al. (June 2012). "Android Permissions: User Attention, Comprehension, and Behavior". In: **SOUPS 2012**.

[2] C. Marforio et al. (Dec. 2012). "Analysis of the Communication Between Colluding Applications on Modern Smartphones". In: **ACSAC 2012**.

TaintDroid

- TaintDroid[3]
- Dynamic Taint Analysis system (see next slide)
- Taint data from sensitive sources (camera, contacts, . . .)
- Track it across **untrusted applications** (blue blocks)
- **Warns** when data reaches an untrusted sink
- Derivative protection systems (block the data)
    - AppFence[4]
    - MOSES[5]



Applications

Home | Contacts | Phone | Browser | Email | ---

Application Frameworks

Activity Manager | Window Manager | Content Providers | View System | Location Manager

Notification Manager | Telephony Manager | Resource Manager | Package Manager

Libraries

Surface Manager | Media Framework | SQLite | WebKit

FreeType | SSL | SGL | LibC

Android Runtime

Core Libraries | Dalvik Virtual Machine

Linux Kernel

Display Driver | Camera Driver | WiFi Driver | Flash Memory Driver | Audio Driver | Binder (IPC)Driver | Power Management

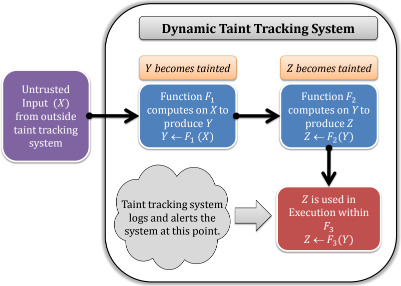
[3] W. Enck et al. (Oct. 2012). “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones”. In: **OSDI 2010**.

[4] P. Hornyack et al. (Oct. 2011). ““These Aren’t the Droids You’re Looking For:” Retrofitting Android to Protect Data from Imperious Applications”. In: **CCS 2011**.

[5] G. Russello et al. (June 2012). “MOSES: Supporting Operation Modes on Smartphones”. In: **SACMAT 2012**.

TaintDroid

Taint Analysis Primer


Dynamic Taint Tracking System
Untrusted Input (X) from outside taint tracking system
Y becomes tainted
Function F_1 computes on X to produce Y $Y \leftarrow F_1(X)$
Z becomes tainted
Function F_2 computes on Y to produce Z $Z \leftarrow F_2(Y)$
Z is used in Execution within F_3 $Z \leftarrow F_3(Y)$
Taint tracking system logs and alerts the system at this point.

- Dynamic Taint Analysis
  - Mark variables with some information
  - Propagate marks across functions
  - Track data through execution paths
- **Help to the developer**
  - Avoid using unvalidated input or derivatives
  - Built in many languages (Perl, Ruby, . . .)
- Assumptions
  - **Code is trusted**
  - Data is not

TaintDroid

Limitations of the approach

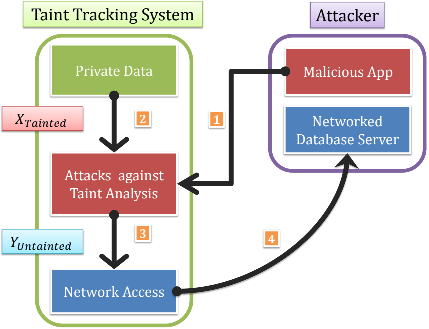
- Known limitations of Dynamic Taint Analysis[6]
  - Control dependence variable assignation
  - Subversion of benign code
  - Side channel attacks
- Assumptions no longer valid
  - Expected: Trusted code/untrusted data
  - Actual: Sensitive data/untrusted code

[6] L. Cavallaro et al. (July 2008). "On the Limits of Information Flow Techniques for Malware Analysis and Containment Detection of Intrusions and Malware, and Vulnerability Assessment". In: **DIMVA 2008**.

Attacks Against TaintDroid

Attack model

- ScrubDroid[7]
  - Android application (1)
  - Server receiving the data
- Implement attacks from vulnerable classes
  - Obtain X_{Tainted} from sensitive sink (2)
  - Untaint the variable ($Y_{\text{Untainted}}$) (3)
  - Leaks the information without warning (4)



[7] http://nicta.info/scrubdroid

Attacks Against TaintDroid

Control dependence

- Use conditional execution paths not directly using the tainted variable
- Implemented in ScrubDroid

Simple encoding Choose $Y_{\mathrm{Untainted}}$ from an array so it matches X_{Tainted}

Count-to-X Increment $Y_{\mathrm{Untainted}}$ until it is equal to X_{Tainted}

Deliberate exception Trigger X_{Tainted} exceptions for which the rescue path increments $Y_{\mathrm{Untainted}}$

Attacks Against TaintDroid

Code subversion

- Use otherwise benign code/tools to create a malevolent chain
- Implemented in ScrubDroid

System command
: Pass X_{Tainted} to system command (*e.g.*, `echo`) which outputs it verbatim, to be captured as $Y_{\text{Untainted}}$

System-file hybrid
: Use unprotected system command to write X_{Tainted} in a file, to be read as $Y_{\text{Untainted}}$

Attacks Against TaintDroid

Side channels

- Use unmonitored channel to pass information
- Implemented in ScrubDroid

Timing Set a timer to expire X_{Tainted} amount of time ahead, compute the time difference as $Y_{\text{Untainted}}$

File length Write X_{Tainted} random bytes in a file, read its lenght metadata as $Y_{\text{Untainted}}$

Bitmap cache Render X_{Tainted} on the screen, OCR $Y_{\text{Untainted}}$ out of the cache

Text scaling Change a widget's property to X_{Tainted}, retrieve it as $Y_{\text{Untainted}}$

Direct buffer Write X_{Tainted} into a memory buffer, read $Y_{\text{Untainted}}$ out

Attacks Against TaintDroid

Evaluation: Success Rates

- Process for each attack
  1. Leak untainted variable $Y_{\text{Untainted}}$
  2. Leak tainted variable X_{Tainted}
  3. Leak untainted variable $Y'_{\text{Untainted}}$
- **All false negatives**
- Direct buffer attack fix (2012-10-06..17d49f89) **leads to false positives**

Technique	$Y_{\text{Untainted}}$	X_{Tainted}	$Y'_{\text{Untainted}}$
Tainted Variable	–	✓	–
File R/W (ovrwr.)	–	✓	–
File R/W (app.)	–	✓	✓ (FP)
Simple Encoding	–	– (FN)	–
Count-to-X	–	– (FN)	–
Exception-Error	–	– (FN)	–
Shell Command	–	– (FN)	–
File-Shell Hybrid	–	– (FN)	–
Timekeeper	–	– (FN)	–
File Length	–	– (FN)	–
Clipboard Length	–	– (FN)	–
Bitmap Cache	–	– (FN)	–
Bitmap Pixel	–	– (FN)	–
Text Scaling	–	– (FN)	–
Direct Buf. (Rel.)	–	– (FN)	–
Direct Buf. (Git)	–	✓	✓ (FP)
Remote Control	–	– (FN)	–

Attacks Against TaintDroid

Evaluation: Timing

- Two types of data
  - IMEI (15 B)
  - 5 s sound recording from microphone (11 kB)
- Not practical **but doable**

Technique	IMEI		5 s audio	
	avg. [ms]	σ	**avg. [ms]**	σ
Tainted Variable	3.48	4.07	364.97	67.31
File R/W	47.62	19.56	386.01	49.85
Simple Encoding	9.55	4.55	795.72	49.12
Count-to-X	10.14	5.41	8278.64	84.20
Exception-Error	53.22	22.09	—	
Shell Command	72.22	12.69	—	
File-Shell Hybrid	78.10	25.80	—	
Timekeeper	1037.66	82.60	—	
File Length	72.37	21.78	—	
Clipboard Length	84.89	18.61	—	
Bitmap Cache	312.27	24.45	—	
Bitmap Pixel	35.95	12.35	2899.80	172.56
Text Scaling	12.92	5.91	3022.58	84.12
Direct Buffer	4.00	3.67	2988.70	87.69
Remote Control	2583.10	976.82	—	

Attacks Against TaintDroid

Mitigation

- Overmark
  - Increase false positives (*e.g.*, Direct Buffer attack)
  - Impractical in case of blocking systems
- Manual marking
  - Requires cooperative developer
- Include comparisons to propagation rules
  - Most control dependence attacks use them for checks

Conclusion

- **Taint analysis** works as a help for the developer
  - identify use of untrusted data in trusted code
- but **is limited** when used against them
  - **untrusted code can be written** to misuse and leak sensitive data
- Future work
  - Study static analysis in this context
- ScrubDroid is Open Source[8]
  - Main author: `<golam.sarwar@nicta.com.au>`
  - Longer technical report available at `http://www.nicta.com.au/pub?id=7091`

[8] `http://nicta.info/scrubdroid`

Conclusion

- **Taint analysis** works as a help for the developer
  - identify use of untrusted data in trusted code
- but **is limited** when used against them
  - **untrusted code can be written** to misuse and leak sensitive data
- Future work
  - Study static analysis in this context

- ScrubDroid is Open Source[8]
  - Main author: `<golam.sarwar@nicta.com.au>`
  - Longer technical report available at `http://www.nicta.com.au/pub?id=7091`
  - Demonstration available!

Thanks — `<olivier.mehani@nicta.com.au>`

[8] `http://nicta.info/scrubdroid`