

Dawn Song dawnsong@cs.berkeley.edu

The Problem

 How to ensure the execution of a given program will not leak private information?

• Why should we care?

- Users download/execute third-party code often

» Spyware

- » Trojan
- » Can't trust reputably vendor: e.g., Sony rootkits
- In security-critical systems (e.g., military setting)
 » How to ensure no malicious actions embedded in thirdparty code?

- Misconfiguration can cause privacy leakage

Two Steps Causing Privacy Leakage

- 1. Reading/accessing sensitive inputs
- 2. Leaking info about sensitive inputs through attacker-observable outputs

Assuming definition of sensitive data is given.

Why not just Sandboxing?

- Why not just disallow read/access to private data?
 Overly strict for some applications

 Toolbar, anti-virus, etc.
 - . .
- Why not just disallow network access if a program reads/accesses private data?
 - Anti-virus software needs network for update
 - $-\operatorname{Vs.}$ GoogleDesktop sends home the index
- Thus, needs to determine whether accessed private data will be leaked through outputs

Relationship to Information Flow

- Information flow: from output x, can you infer information about input s?
- Noninterference: Program p satisfies the noninterference property if changing confidential inputs of e does not affect the outputs observable to attackers.
- Attacker observable outputs
 - Network data
 - Timing, cache and other covert channels (out of scope)

How to Identify Information Flow?

- · Static analysis
- Dynamic analysis

Static Analysis (I): Behavior-based Spyware Detection

- CFG-based reachability analysis
- Does the component which handles browser events make dangerous Windows API calls?
- Rationale
 - Event-handling code gets information about user - Dangerous Windows API calls may leak information to outside world
 - » File write, network send, etc.

Challenges

- · Identifying event-handling code - Need to identify event-specific instruction
 - Can you do better?
- Analyzing binary for reachability analysis Need to disassemble
 - » Issues?
 » Can't handle packed code
 - Build CFG
 - » Issues?
 - May be incomplete due to indirect jumps, etc. - Better binary analysis can help
- Compile the blacklist for API calls
 - Manual effort
 - Automatic learning
 - » Issues? » Can you do better?

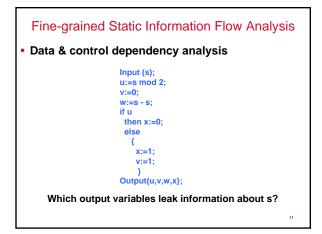
Limitations (I)

Coverage: False Negative

- Different ways for attackers to gain user information? » Read shared memory
- Different ways for attackers to send out user
- information?
 - » Not through Windows API calls » Native API?
- » Going through legitimate code?

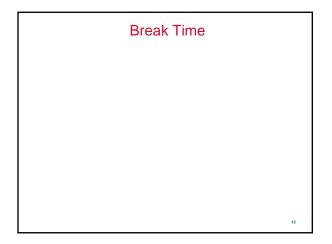
Limitation (II)

- Precision: false positive
 - CFG-based reachability analysis: conservative
 - No data-dependency analysis
 - Sent-out information may have nothing to do with sensitive input

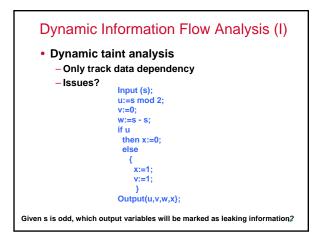


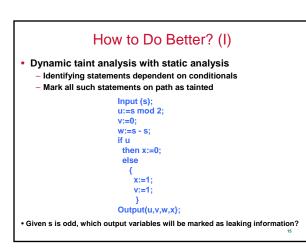
Challenges

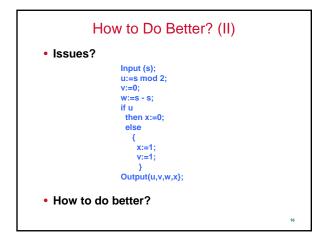
- Static analysis difficult to be precise
 » Conservative
- Malware code obfuscation













Other Limitations of Dynamic Taint Analysis for Information Flow Tracking?

High runtime overhead
 Static code instrumentation/rewriting
 Runtime binary instrumentation

TightLip

17

18

Doppleganger processes

- Doppelganger & original run in parallel
- As long as outputs are same, output does not depend
- on sensitive input
- Dynamic estimate of non-interference
- How to compare with the accuracy of dynamic taint analysis?

Challenges

- Divergence: False positives
 - Doppleganger needs to be exact shadow
 - » In order delivery
 - » Signal handling, etc.
 - Control flow divergence » How to scrub data?
- Zero side effect
- False negatives?

Open Mic

- Brainstorming: better approach?
- Other comments?

Limitations of Noninterference

- Overly strict
 - Password check
 - Meta-data update in GoogleDesktop

Solutions

- Declassification
- Quantitative information flow

Summary

- Detection of privacy breach
 Relationship with information flow
 Static & dynamic techniques
- Next class:
 - Stealthy malware
 - Info on project proposal

22