Data-driven Software Security: Models and Methods

Úlfar Erlingsson

Google, Inc.

Presented by William Johnson and Kellan Christ

Outline 📜

- Author's Background
- Introduction
  a. Security Models & Setting Policies
  b. Low-level Software Policies
- Data-Driven Software Security Model
  a. Anomaly and Intrusion Detection
  b. Open Questions & Formal Modeling
- Methods for Data-driven Software Security
  a. Efficient Monitoring
  b. Privacy Preservation
  c. Match User Expectations and Software Permissions
- Conclusion

Author's Background 📖

- **Úlfar Erlingsson**, Chief Architect for cloud security firm `Lacework`, with focus on framework for end-to-end visibility across cloud, and detection of threats, vulnerabilities, misconfigues, etc.
- Previously head of Security Research Group at `Google`

Introduction 🔌

- Security a concern in 1950s and became prevalent by late 1960s and 1970s
- Key figures:
  - `Jerome Saltzer` `(MIT)`
    - Multics Operating System (time-sharing)
    - End-to-end principle with David Reed and David Clark
  - `Michael Schroeder` `(MIT)`
    - Needham–Schroeder key transport protocol (symmetric encryption)
  - `Butler Lampson` `(UC Berkeley)`
    - 1992 ACM Turing Award winner for contribution to personal computing

Introduction

- Saltzer, Shroeder and notably **Lampson** contributed to formalizing computer security and defining important security models: **access-control lists, capabilities, etc.**
- **Principle of Least Privilege:** permit only the required low-level executions, unless programmer requires explicit, special permission
- Software security is a **form of correctness**
- One-to-one correspondence between security and programming

```
In Security                     In Programming

Security policy        =        Functional specification
Security mechanism     =        Software implementation
Security assurance     =        Program correctness
Security model         =        Programming methodology
```

Introduction: Security Models & Setting Policies

- Secure computing has stagnant progress
- Challenges compared to traditional security:
  - Universal networking as doors for attack surfaces
  - Effective defense requires absence of vulnerabilities
- Task may be easy to approach in one Turing-complete language, but difficult in other » same issue with security models
- Functional specification is hard to get right
  - Difficult to specify intended functionality
  - Specification likely to be wrong

Introduction: Low-level Software Policies

- Good security model has simple policies to thwart low-level software vulnerabilities
- Placement of Stack Guard / Canary
  - **Model:** nonce / random value placed **before** the base pointer and instruction pointer
  - **Implementation:** `-fstack-protector` (Enforced in compilers by default)
  - **Policy:** one should not be able to mess with return value stored on the stack

Introduction: Low-level Software Policies

- Data-driven Software Security Application
  - Software is more complex
  - Data-driven approach is effective (as seen in fighting spam and abuse)
  - Past information to figure out what should happen in future
    - Policies can be derived from **historical evidence** of captured executions » used to minimize attack surfaces (i.e. policy is **empirical**)
    - Enforce: Focus primarily on events that should not occur

Introduction: Low-level Software Policies

- Lampson's gold standard
  - Authorization
  - Authentication
  - Audit
- Figure 4 - Access Control
- Figure 5 - Information Flow

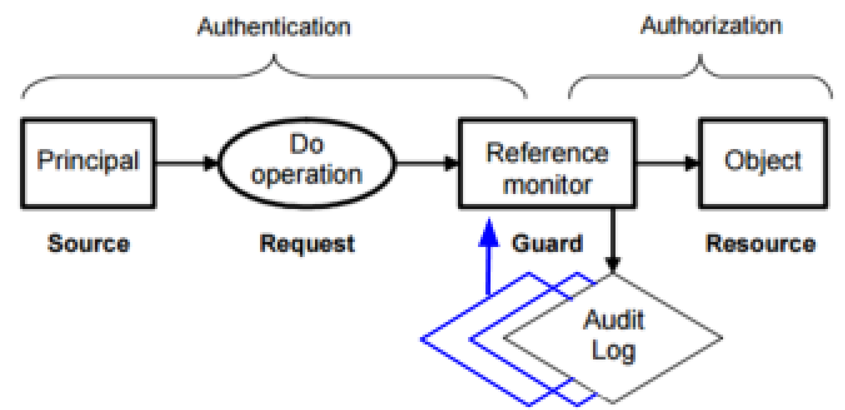


Fig. 4. Data-driven software security tied to access control, as per [5].

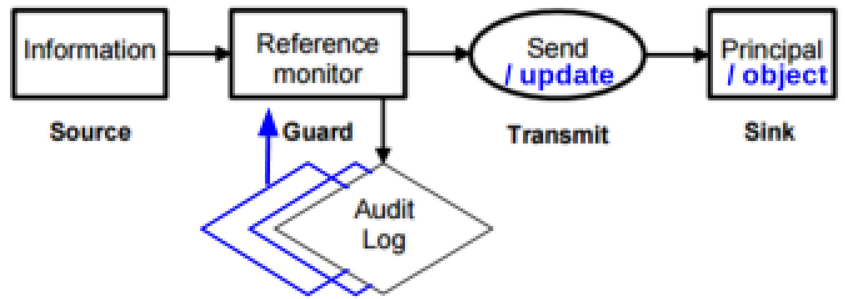


Fig. 5. Data-driven software security tied to information flow, as per [5].

Data-Driven Software Security Model: Anomaly and Intrusion Detection 🔥

- Previous approaches use traces from benign runs ("normal" behavior)
  - Subject to false positives
- Data-driven model suggests empirical program abstractions
  - Empirical suggests encompassing **all real-world execution traces** to reduce false positives (not just from training runs)
  - Not too fine-grained
    - i.e. Cannot include user-specific behavior
  - Technique integrated throughout software development cycle

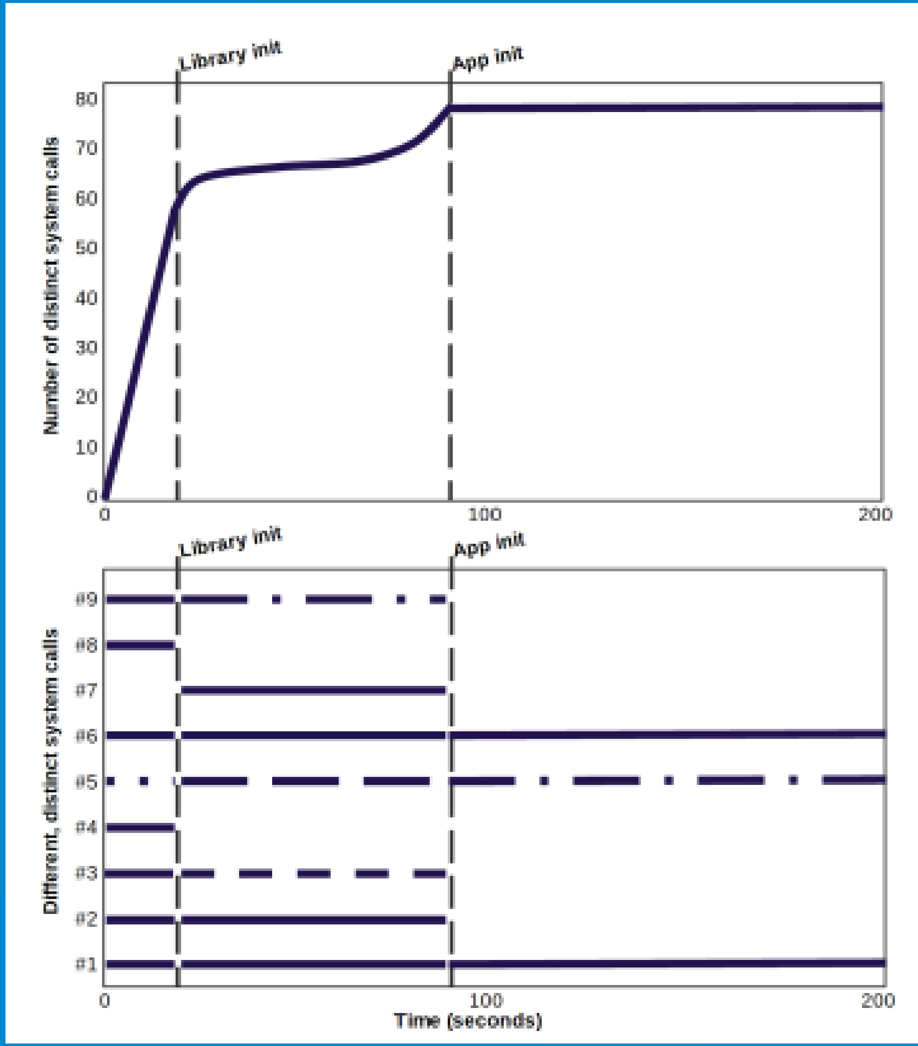
Data-Driven Software Security Model: Open Questions and Formal Modeling 📈

- Can model be attacked?
  - Sibyl attack (51% consensus to force malicious behavior to be classed as benign)
  - Overcome by eliminating Sibyl or managing numbers
- How can formal challenges associated with empirical program abstraction be accounted for?
  - Programs function as language recognizers
    - Recognizing too large a set of inputs
  - Would require restricting set of recognized inputs

Methods for Data-driven Software Security ⭐

- Google's test-driven development combined with process instance execution-trace summaries
- Goals:
  - Utilize system-call-trace-based security policies
    - Collect, summarize, enforce with standard technologies
      - `ptrace`, `seccomp_bpf`
  - Collect data without violating user privacy
    - RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response)
  - Reinforce data usage with more efficient tracing
    - Reordering executable-binary code (message-marshaling code)
    - Handle abrupt, unexpected changes in software behavior

Methods for Data-driven Software Security: Efficient Monitoring 🔍



Methods for Data-driven Software Security: Privacy Preservation 🕵️

- RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response)
  - **In frequency of system calls:** Utilizes binary form of randomized responses (adds noise to data)

Methods for Data-driven Software Security: Match User Expectations and Software Permissions 🔓

- Estimate user expectations of software behavior
  - Creating "peer groups" of similar software
- Employ machine learning
  - word2vec skip-gram model in identification of descriptions and user interactions
- Remediation options
  - Fail-stop enforcement to halt execution for non-critical software
- Deployment bootstrapping
  - Enforcement once policies have converged and stabilized
  - Integrate with software development lifecycle

Conclusion 🏁

- Draw historical evidence to identify what constitutes "normal" software execution
- Data-driven model is distinct from traditional anomaly and instrusion approach
  - Empirical (consider all execution traces)
- Employing data-driven model would reduce attack surfaces in similar manner to a well-designed firewall

Related Works

- A. Gorla, I. Tavecchia, et al. "Checking app behavior against app descriptions"
- U. Erlingsson, V. Pihur, et al. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response"