Software Security in the Real World

Úlfar Erlingsson
Security Research at Google

There's an Essay (Arxiv 1605.08797)

Data-driven Software Security: Models and Methods

Úlfar Erlingsson
Google, Inc.

***Abstract*—For computer software, our security models, policies, mechanisms, and means of assurance were primarily conceived and developed before the end of the 1970's. However, since that time, software has changed radically: it is thousands of times larger, comprises countless libraries, layers, and services, and is used for more purposes, in far more complex ways. It is worthwhile to revisit our core computer security concepts. For example, it is unclear whether the Principle of Least Privilege**

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

Fig. 1. The correspondence between aspects of computer security and computer software, the last one an addition to those identified by Lampson [5].

Computer Security is an Old, Tough Nut to Crack

Identified as a crucial problem since the 1950's, at least.

Key concepts developed by the 1970's and 1980's.

- E.g., Software protection / access control (Lampson), and key security principles (Saltzer and Schroeder)

Haven't made much progress since. ☹

Computer Security in the Real World

Series of talks and papers between 2000 to 2005
- Butler Lampson, looking back over 30 years

➔ Computer security is even harder than real-world security

Software security is a form of correctness
- But, dealing with malicious adversaries, not (random) faults
- Any flaw can be reliably exploited, infinitely often

Foundations of the Gold Standard of Security

Same key aspects in software construction & computer security

In programming		In security
Specification	=	Security policy
Implementation	=	Enforcement mechanism
Correctness	=	Assurance
Methodology*	=	Security model

* e.g., functional vs. declarative vs. imperative programming

Data-driven Software Security

Writing policy (aka specs) is the hard bit

- Sometimes easy — e.g., in programmer-intent model used in control-flow and data-flow integrity work

Propose ***data-driven software security*** model

- Use historical evidence to guide enforcement
- Based on ***empirical programs*** that capture *all* security-relevant events ever seen, in all executions
- Easy to write policies: they just define what events are

Why a Data-driven Approach to Software Security ?

Today's computer software is not that of the 1960's.

- 1000x larger, more complex, with "emergent properties"
- A "found artifact" for both developers and users
- Data-driven approaches are often successful (spam, AI)

Easy to use historical evidence for **attack-surface reduction**

Online, networked computing makes monitoring feasible

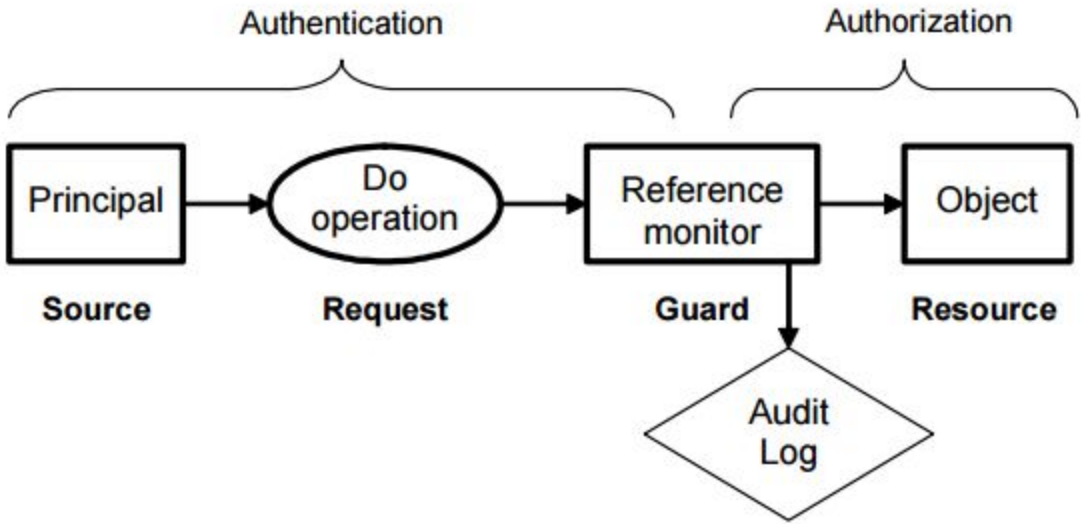
- Can possibly collect all the data, from all software uses

What could Data-driven Software Security Mean?

Could define it using all data relevant to a software execution

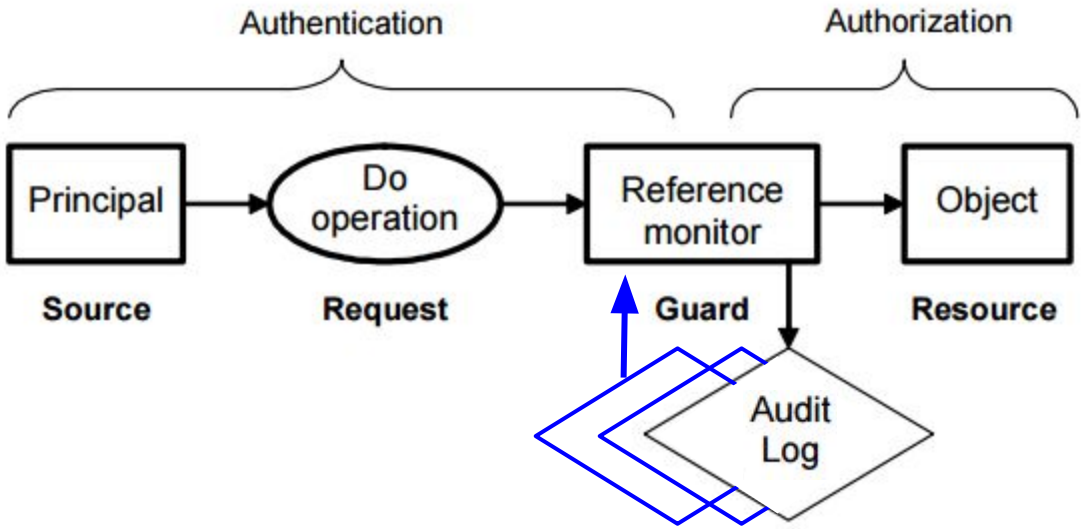
- **Can't concern only a single execution trace**
  —as in state-, context-, or history-based access control
- **Can account for context of the software's use**
  —time of year and day, properties of the user, etc.
- **Can even account for & monitor implicit aspects**
  —such as user's intentions and expectations
- **Can use data from other (e.g., similar) software**

Data-driven Security vs. Access Control Models



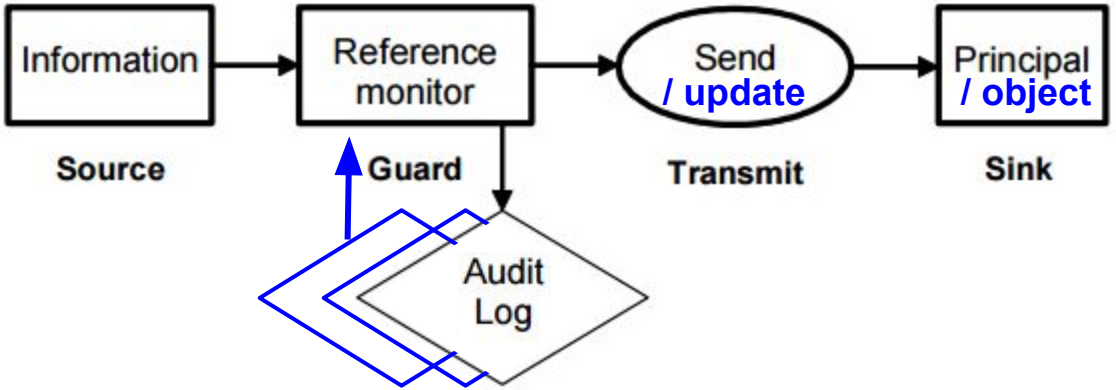
Data-driven Security vs. Access Control Models

Comprehensive audit logs are a key to policy & enforcement



Data-driven Security vs. Information-flow Models

Comprehensive audit logs are a key to policy & enforcement —same as with access control



Example Benefits

Microsoft Windows Solitaire game

- Has never used the networking libraries it includes

Heartbleed vulnerability in OpenSSL

- No TLS message ever had a huge heartbeat payload

Linux **`keyctl`** kernel vulnerability CVE-2016-0728

- The **`keyctl`** system call isn't used by ~any~ software

Applying Data-driven Software Security

Define empirical program abstraction

- What's the program? (E.g., Linux binary, or C source code.)
- What's a security-relevant event? (E.g., an RPC message.)
- How are execution traces collected? (E.g., as summaries.)

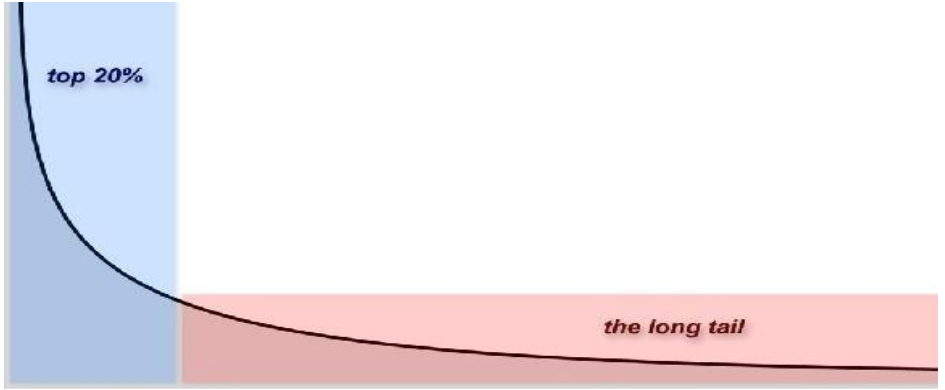
Set how policy interprets historical evidence. And enforce!

For example: Focus on what *never* happens

- What system calls does it never make?
- What code & services does it never use?

Empirical Programs must Capture *All* Executions

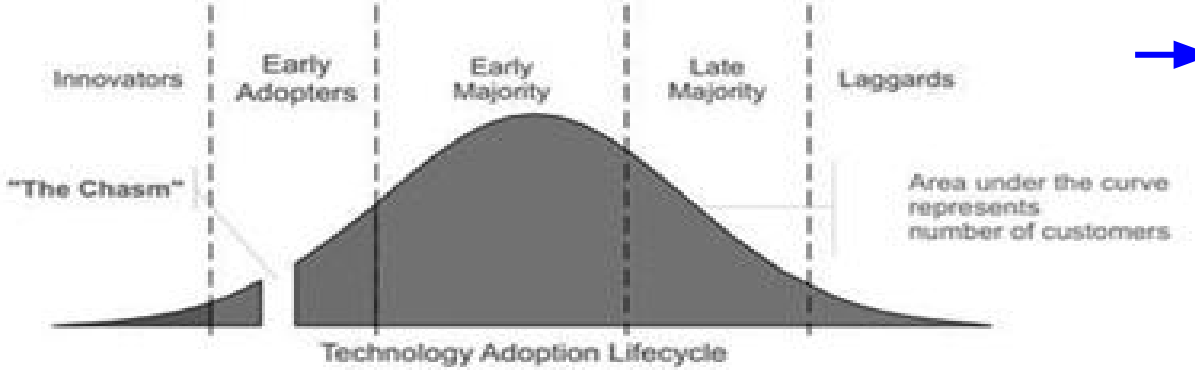
- Not enough training data to capture the long tail of behaviors
- Users and contexts may vary widely

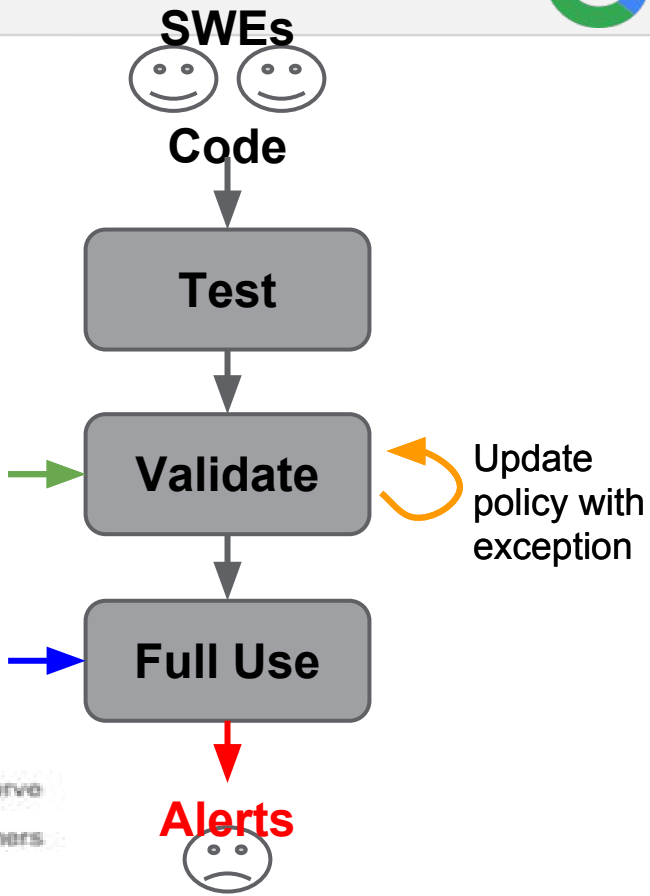


- If some behavior has never been seen—ever, in all executions—isn't that a security violation or bug ?

Bootstrapping Enforcement

- Utilize tests and old versions
- Build on Dev & Beta executions
- Integrate with update process
- Avoid Y2K surprises!





Three Challenges

1. How to monitor and collect data efficiently enough
   - Possible for system calls (alt-syscall, seccomp_bpf, SIGSYS)
2. How to collect data without violating users' privacy
3. How to best make use of the data

Efficient Large-Scale Monitoring of Software Behavior

How to do efficient, ubiquitous monitoring of software?

Focus on simple case: What does software *not* do.

- What system calls does it never make?
- What code & services does it never use?

Work with many collaborators at Google:

- Yinqian Zhang, Michael Vrable, Noah Johnson, Ludovico Cavedon.

Learning Concrete Software Behavior with Privacy

What is software actually doing, on users' computers?

WARNING: Just knowing syscalls can violate privacy!!
e.g., users calling decodeXdiv354 may be pirates.

- Must count across all software instances, with privacy!

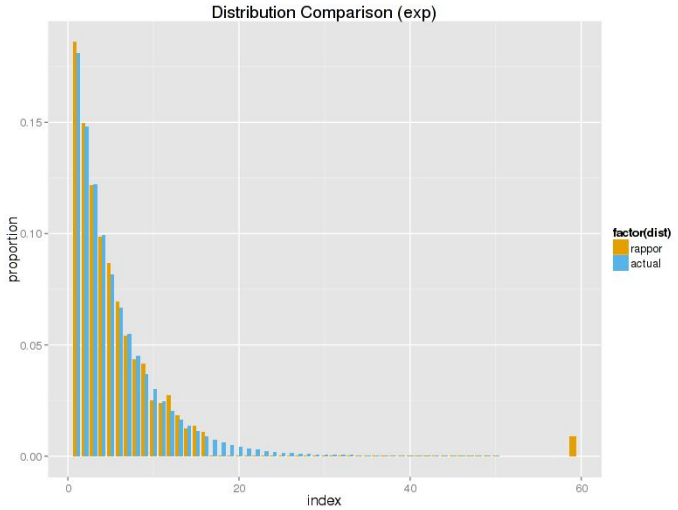
Work with many collaborators at Google:

- Ananth Raghunathan, Ilya Mironov, Andy Chu, Giulia Fanti, Vasyl Pihur, Aleksandra Korolova, and more.

Deployed Open Source Effort

https://github.com/google/rappor

- demo you can run with a couple of shell commands
- client library
- analysis tool and simulation
- documentation



Used by Chrome
- billions of metrics collected daily
- Google can only see the forest, but not any trees.

Discovering User Expectations of Software Behavior

What do its users think that software should be doing?

User expectations should strongly influence policy !

Can try to learn expectations from data

- E.g., by using modern machine learning: ***Deep Learning***

Work with many collaborators at Google:

- ***Martin Pelikan***, Sooel Son, Karen Lees, Suman Jana, Iulia Ion, etc.

Data-driven Least Privilege

Principle of least privilege: “Only what you need”
- At any level, e.g., system calls to OS or service calls

But, modern software has libraries & code to do everything
- Can do all traces {a,b}* — like a LAMP stack
- Hardening (type-safety, CFI, etc.) doesn’t fix this

Data-driven: Experience shows what software needs
- Reduces possibilities down the “science of security” lattice

A Data-driven Software Security Model

Worth considering:
A practical, simple way to increase software security

Existence proof:

- Used in development of ChromeOS
- There, defended against CVE-2016-0728

Fun connections between empirical programs and PL theory