Empirical Evaluation of API Usability and Security

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Sam Weber*, Brad Myers^, Forrest Shull*, Robert Seacord*, Michael Coblenz^, Joshua Sunshine^, David Keaton*
* = SEI, ^ = CMU

Outline

- Project Vision
- Background
- Approach and Progress
- Summary

Project Aims and Vision

Our project will develop and empirically test *concrete* and *actionable* API design principles that lead to more secure code

Long-term vision: To empirically evaluate secure development practices

- Many decades of work on secure development practices, most of it based upon experience and reflections by smart people
  - Including many thousands of pages of secure coding guideline books
- Little, if any, information about validity and relative merit of different practices
  - "Common Sense" is often wrong
  - Necessary to improve practices, make cost/benefit decisions, decide between competing guidelines

Principle: Programmers and designers are people, too.

- Need to design systems that people can securely code to

Why APIs?

APIs have large impact upon system security

- C String library still major cause of problems
- See Wang et al, "Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization"

APIs are long-lasting

APIs are generally designed by a small number of more-experienced people

Note: We are making little distinction between APIs, SDKs, and language features

- Examples:
  - concurrency is built-in to Java, but thought of as a library in C (although compiler-writers disagree with this)
  - C++'s const and Java's final have different semantics – we will examine the control of mutations, irrespective of language choices

Project Decisions and Goals

Will concentrate on usability and security of ***non-security-related*** APIs

- Threat model: programmers are well-meaning, not malicious, but code that they write subject to malicious attacks
- Interested in security impact when programmers are thinking of functionality, **not** security

Goals:

- Actionable and specific guidance to API designers about
  - The impact of API design decisions on security
  - The interaction between usability and security
- Guidance to language designers about language features that affect ability of API designers to express important properties
- Accepted methodology for research in this area

Concrete Motivating Issue -- Mutability

Mutability: whether or not an object's state can be modified after creation

Security community values immutability of objects:

- Oracle: "Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code"
- Java security guidelines stress additional work required for mutable objects
- Closely related to TOCTOU attacks
  - Object's state should not be able to be changed between check and use

Empirical Usability Research on Mutability

Stylos and Clarke: "Usability Implications of Requiring Parameters in Objects' Constructors"

Two fundamental patterns for object constructors:

1. Required-constructor:

```
foo = new FooClass(arg1, arg2);
```

2. Create-set-call:

```
foo = new FooClass();
foo.setArg1(arg1);
foo.setArg2(arg2);
```

Create-set-call implies objects must be mutable

Details

Experiments done using thirty professional programmers in lab

- Both creating own APIs, and using supplied APIs in different styles
- Methodology:
  - Gave programmers description of task, then asked them what code they would expect to write
  - Asked to design own API
  - Then gave them (sequentially) different tasks. In some tasks, only one style API was given to each programmer, in others they had to create objects of both kinds.
  - Interviewed participants afterwards.
- Consistently and strongly preferred create-set-call style, and more effective
  - Required-constructor interfered with common learning techniques, less flexible

Usability vs security

Apparent trade-off between usability and security wrt mutability

- Is this real? Can we measure?

Guesses:

- Usability and security sometimes aligned, sometimes not
  - Concurrency probably both factors aligned

Methodology and Project Plans

Initial focus on mutability

- Isolating mutability from concurrency

Stage 1: Construction of measurable hypotheses

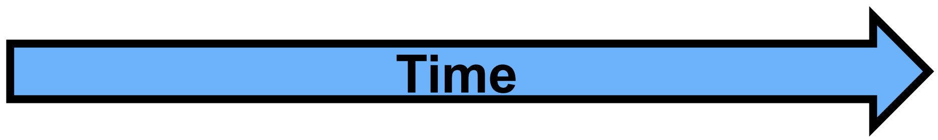
- Literature/corpus search for immutability usage and impact
- Structured interviews with key designers
- Interviews/surveys of professional programmers

Stage 2: Lab studies

- Students and/or professional programmers
- Controlled experiments
  - Start with controlled elicitation of how programmer thinks code *should* be written
  - Give tasks with alternate APIs
  - Use think-aloud protocol
  - Within-subject design: all subjects use all APIs

Example Lab Study

	Task 1	Task 2	
Group A	Style A	Style B	(interview)
Group B	Style B	Style A	(interview)



Candidate usability measures:

- **Effort**
- **Correctness**
- **Subjective rating**

Candidate security measures:

- **Avoidance of known (seeded) vulnerabilities**

Summary

Doing empirical investigations of usability and security impacts of API designs

- Starting with object mutability

Intended results:

- Concrete and actionable guidance to API and language designers
- Methodology for user-studies of security development methodologies

Threats to Validity

1. Study duration
   - Studies necessarily limited in time
   - Real-life performance involves multiple factors
     - Initial code creation
     - Code modification ease
     - Testability
     - Communication between programmers
   - We argue that increasing study duration is not as critical as isolating strands of overall development quality and productivity
2. Programmer variability
   - Controlled by within-subjects methodology
   - Will track programmer expertise
3. Security/Usability dimensionality
   - Multiple dimensions to each
   - Qualitative feedback will help us realize more subtle aspects