

# UMD Lablet Summary

**Jonathan Katz**  
Dept. of Computer Science



## Lablet overview

- 20 faculty researchers involved
  - 14 from UMD, 6 external collaborators
- UMD faculty drawn from five different departments on campus
  - CS, ECE, Information Studies, Criminology, Reliability Engineering
  - Collaboration fostered by the Maryland Cybersecurity Center (MC2)

## Lablet participants

- Adam Aviv, USNA
- John Baras
- Marshini Chetty
- Michael Clarkson, Cornell
- Michel Cukier
- Tudor Dumitras
- Jeff Foster
- Jen Golbeck
- Michael Hicks
- David Van Horn
- Joseph JaJa
- David Maimon
- Babis Papamanthou
- Aditya Prakash, VA Tech
- Elaine Shi
- Katie Shilton
- VS Subrahmanian
- Mohit Tiwari, UT Austin
- Sam Tobin-Hochstadt, IU
- Poorvi Vora, GWU


## Lablet organization

- Strengths:
  - Scalability and composability
  - Security metrics (empirical security)
  - Human behavior
- Lablet efforts organized around 9 tasks

## Label tasks

- Scalability/composability
  - Verification of hyperproperties
  - Trustworthy and composable software systems with contracts
- Security metrics
  - Empirical n
  - Human beh
- Human beha
  - Does the p behavior?
  - User-centered design for security
  - Understanding developers' reasoning about privacy/security
  - Reasoning about protocols with human participants
- Policy-governed secure collaboration
  - Trust, recommendation systems, and collaboration

Some tasks comprise  
multiple projects



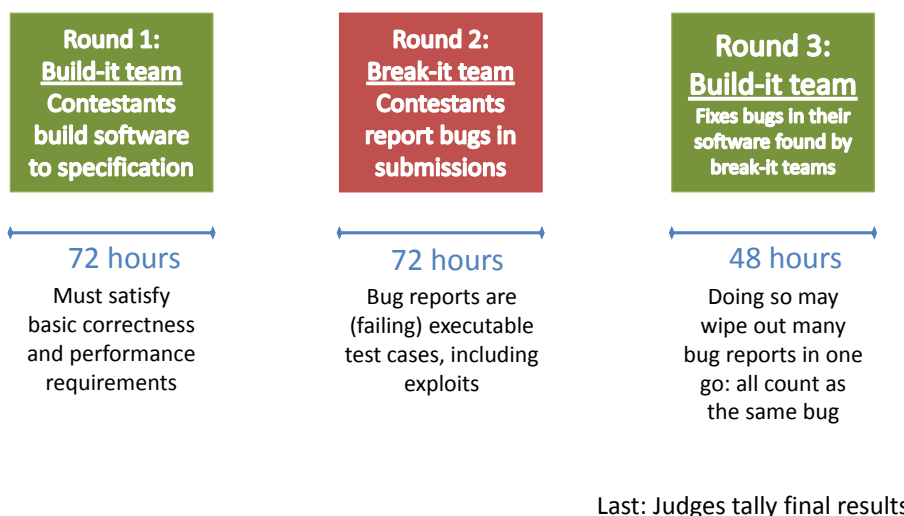
**BUILD  
BREAK  
FIX IT**

A security-minded programming contest

## MOTIVATING A NEW SECURITY CONTEST

- Today's contests reward those who can **break** systems by finding vulnerabilities
  - DEFCON CTF, Collegiate Cyber defense challenge (CCDC), Pwn to Own, ...
- But we also want the opposite: reward those who can **build more secure systems**
  - Not the same skillset set as breaking things
  - Of direct relevance to companies, and society

## BUILD IT, BREAK IT, FIX IT: OVERVIEW



## GOALS

- Empirically assess what actually works by correlating features of submission with team performance
  - Programming language, framework, library, ...
  - Developer experience, S/W process, ...
  - Using static analysis, fuzz testing, etc. ...
- Encourage defense, not just offense
  - Tie together security with reliability: Bugs are bad, whether they are exploitable or not
  - Elevate real concerns: performance and feature-fullness
- Provide direct feedback to contestants
  - The contest penalizes a lack of security: “feel” the mistake!

## Lablet tasks

- Scalability/composability
  - Verification of hyperproperties
  - Trustworthy and composable software systems with contracts
- Security metrics
  - Empirical models for vulnerability exploits
  - Human behavior and cyber vulnerabilities
- Human behavior
  - Does the presence of honest users affects intruders’ behavior?
  - User-centered design for security
  - Understanding developers’ reasoning about privacy/security
  - Reasoning about protocols with human participants
- Policy-governed secure collaboration
  - Trust, recommendation systems, and collaboration

## Verification of hyperproperties

Task leads: Michael Hicks, Michael Clarkson

Hard problem(s): Composability

## Properties (Lamport)

- **Trace:** sequence of execution states
- **Property:** set of traces  
trace  $t$  satisfies the property  $P$  iff  $t \in P$   
satisfaction depends on the trace only!
- A system/program satisfies a property iff all its traces satisfy the property

## Verification of properties

- Manual verification for classes of properties based on logical proof systems [Gabay et al. '80]
- Automated verification for classes of properties based on model checking [Clarke et al. '86]
- Partly automated verification [Alpern-Schneider '87]
- Can formalize/verify any property

## But...

- Many natural security policies cannot be cast as properties
  - E.g., information flow
    - Depends on *pairs* of traces

## Hyperproperties

- Satisfaction depends on *sets* of traces  
[McLean '96]
- A hyperproperty is a set of *sets* of properties  
[Clarkson-Schneider '08, '10]
  - A system/program satisfies a hyperproperty  $H$  iff the set of its traces is in  $H$
- All(?) security policies can be expressed as hyperproperties

## Verification of hyperproperties?

- Safety and liveness methodology?
  - [Clarkson-Schneider '08, '10]
- Model-checking approaches?
  - [Clarkson et al. '14]
- Logical proof systems? ...this task
  - **Idea:** extend linear-time temporal logic to reason about sets of traces
  - **Idea:** investigate compositional proofs of security in a logic for hyperproperties



## **Trustworthy and composable software systems with contracts**

**Task lead: David Van Horn**  
**Hard problem(s): Composability**

### **Program verification**

- Very successful for detecting/preventing many types of software vulnerabilities
- Two limitations of current state-of-the-art:
  - Assume analysis of a complete program, rather than allowing for component-wise analysis
  - Assume program written in a single programming language
- Would like better *composability!*

## Contracts

- Semantic invariants guaranteed by software components in the source code, specified as pre-/post-conditions

```

module math
export derivative
with contract
  faulty component "blamed" for contract
  violation
    f : (0,1] -> (0,1]
    δ : positive
    ->
    f' : (0,1] -> real
    ...

```

## Drawback

- Dynamic enforcement of contracts impose significant/unpredictable run-time overhead
  - By a factor of  $10 - 10^4$
- Dynamic enforcement leaves open the question of how to *recover* from a detected contract violation

## This task

- *Static* contract checking
  - Verify what you can at compile-time, but can still fall back on run-time guarantees
- Program *components* can be analyzed/verified independently, in a *composable* manner
- Extensions to multi-language programs
  - Evaluate using the Racket standard library

## Main ideas

- Apply symbolic-execution techniques to contract checking at compile-time
- Combine this with algebraic reasoning to achieve completeness
- In conjunction, these allow for analysis of more complex programs, more quickly, than prior work on static contract checking

## **Empirical models for vulnerability exploits**

**Task lead: Tudor Dumitras**  
**Hard problem(s): Security metrics**

### **Background/motivation**

- Security of deployed systems not adequately captured in current models/metrics
  - E.g., estimating # vulnerabilities in software does not account for the fact that many of these are never exploited
  - E.g., “patched” vulnerabilities may still be present in the wild due to failure to apply patches

## This task

- Derive empirical models of vulnerabilities and attack surfaces; correlate with real-world attack data
  - What vulnerabilities are exploited in real world?
- Understand deployment-specific factors that influence security of real systems
  - How to best characterize *attack surface*
- *Using real-world field data from WINE*

## Measuring security of deployed systems

- Count of vulnerabilities exploited
- *Exploitation ratio*: ratio of exploited vulnerabilities to disclosed vulnerabilities
- *Survival probability*: time to exploit
- *Exercised attack surface*: number of distinct exploits on a host/month

## Exploitation ratio

- Identify exploits from Symantec signature definitions (Allodi, 2013)
  - [http://www.symantec.com/security\\_response/threatexplorer/azlisting.jsp](http://www.symantec.com/security_response/threatexplorer/azlisting.jsp)

Product	Exploited Vulnerabilities	Exploitation Ratio
Office 2000	26	0.32
Office 2003	41	0.36
Office 2007	17	0.31
Office 2010	4	0.29
Adobe Reader 6	5	0.21
Adobe Reader 7	11	0.17
Adobe Reader 8	29	0.16
Adobe Reader 9	29	0.11
Adobe Reader 10	12	0.09
Adobe Reader 11	4	0.07

**Fewer than 40%** of known vulnerabilities are exploited

**Decrease with newer versions**

## Implications

- Scarcity of exploits matches cybercrime data
  - 2013: \$100,000 per zero-day exploit
- Reasons?
  - System-security technologies that render exploits less likely to work
  - Commoditization of malware industry
- Take-aways?
  - Prioritization of patch deployment
  - Risk assessment

## **Human behavior and cyber vulnerabilities**

**Task lead: VS Subrahmanian**

**Hard problem(s): Security metrics, human behavior**

### **Background/motivation**

- When vulnerabilities are exploited, patches are often released soon after
  - But past work indicates that patches are not fully deployed even 4 years after disclosure
  - Why?

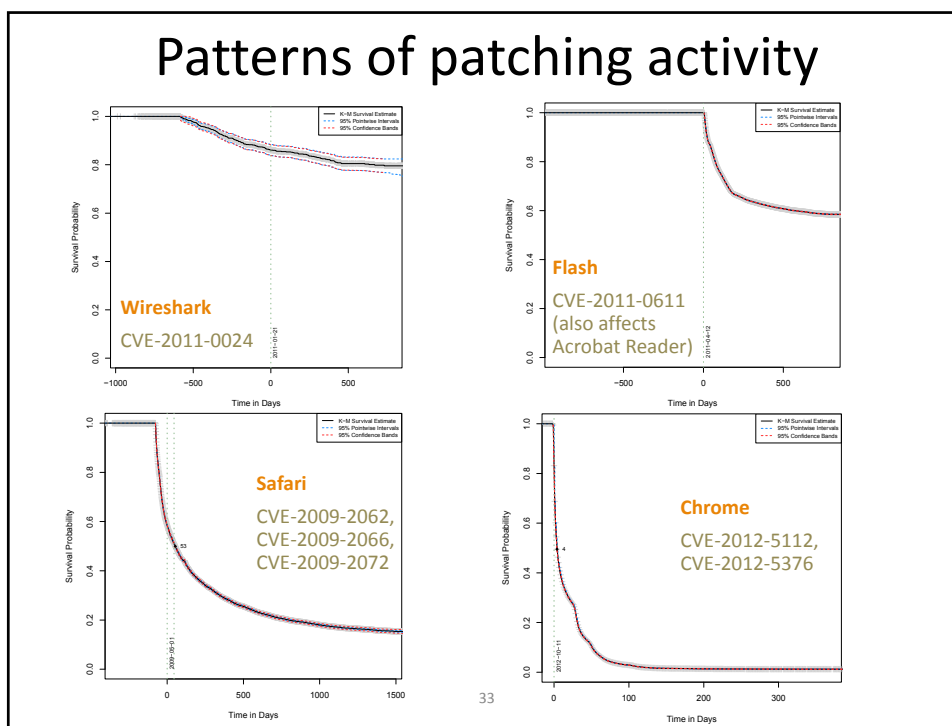
## This task

- Characterize the rate of vulnerability patching
- Determine the factors that influence the rate of patch deployment
  - Technological
    - Using WINE dataset
  - Sociological
    - Based on targeted user studies  
<http://netchi.umd.edu/software-updating-study.html>

## Vulnerability patching

- Goal: characterize the rate of vulnerability patching
  - Start of patching
  - Time to patch 50%, 90%, 95% of vulnerable hosts





## Implications

- Patch deployment exhibits a long tail
- Considerable variation
  - Automated updates faster
  - Hosts may remain vulnerable even after user believes patch was deployed

## Sociological factors (host-level data)

- Users classified into one of several categories
  - Gamers, professionals, s/w developers, other
  - Classified based on software installed
- Several factors investigated for correlation with patch rate
  - Number of (unsigned, low-frequency) binaries downloaded
  - Travel history
  - Time of login

## Sociological factors (user-level data)

- Investigate human barriers to deploying software updates
  - User surveys + in-depth interviews with network administrators
  - Develop improved interfaces/incentives for patch deployment
- Determine if user-level data matches host-level data

## Does the presence of honest users affect intruders' behavior?

Task leads: Michel Cukier, David Maimon

Hard problem(s): Human behavior

## Social sciences and cybersecurity

- **Idea:** Investigate application of criminological theories to cybersecurity
  - Routine activity theory
  - Rational choice theory
  - Deterrence theory
- Using *honeypot data* collected at UMD

## This task

- What is the effect of legitimate users on system trespassers' online activities?

## Experimental setup

- Honeypots accessed through vulnerable SSH
- Randomly assign honeypot configuration to each attacker
  - Control: no legitimate users present
  - Condition 1: One non-admin user present
  - Condition 2: One admin user present
  - Condition 3: 10 non-admin users present at any time
  - Condition 4: 10 admin users present at any time
  - (Honest users cycle every 8 hours; are idle)
- Study attacker network activity/keystrokes/etc.

## Questions to be addressed

- Does the presence of honest users affect the duration of an attacker's login?
- Does their presence affect number/type of network activities?
- Replication over time?
- Reproduction (using different methods)?

## (Potential) implications

- Simulate presence of honest users to deter/influence attacker behavior
- Generate new IDS rules based on these insights
  - E.g., look for execution of "who" command

## **User-centered design for security**

**Task leads: Jen Golbeck and Adam Aviv**

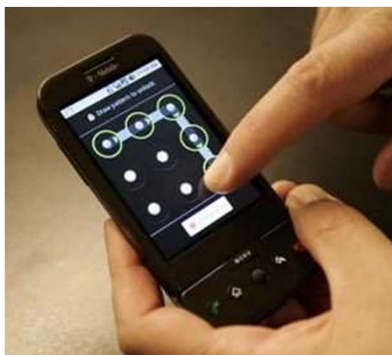
**Hard problem(s): Human behavior, security metrics**

## **User-centric design**

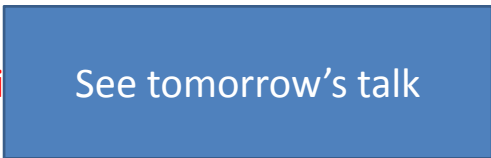
- Goal: development of new, usable-security measurement techniques and metrics to inform the design and development of new cybersecurity applications
  - Empirical measurements; usability metrics
- Specifically:
  - Visual perceptions of security/usability
  - Impact of security policies on user behavior

## Visual perceptions of security/usability

What visual properties of passwords do people *perceive* as secure?



## Research questions

- What **visual features most affect** users' perceptions of security/usability?
- How well **with real**  **respond**
- Can we use insight gained about user perceptions to **design better security systems**?
  - E.g., “nudge” users to better choices

## **Understanding developers' reasoning about privacy and security**

**Task lead: Katie Shilton**

**Hard problem(s): Human behavior,  
policy-governed collaboration**

### **Background/motivation**

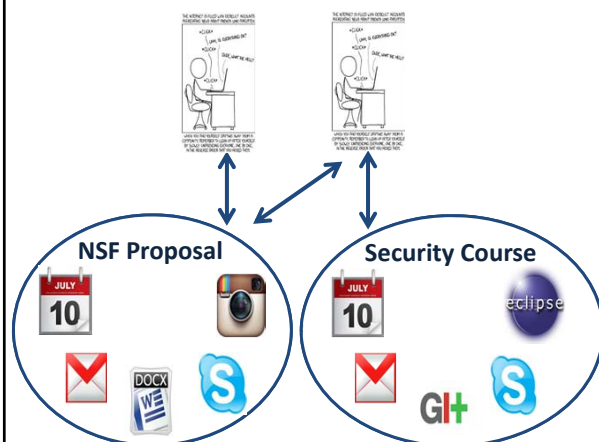
- Mobile app developers often request more permissions than necessary
  - Why?
- Users are unclear what permissions need to be granted to enable some functionality
  - Cannot often get the full functionality they desire
- Both can lead to security/privacy concerns



## This task

- Design and evaluate a new system (“Bubbles”) that can
  - Simplify user-directed information-flow control, especially to other users
  - Simplify developers’ design/implementation
  - Simplify system-level information-flow tracking and control

## Main idea



- Data clusters around real-world contexts
- Privacy policy as **access control** on **contexts**
- Apps run in Bubbles; cannot affect privacy

## Challenges

- Lots of bubbles  
→ provide UI for navigation
- Apps no longer own data  
→ Provide an API for developers
- System implementation  
→ Virtualize dangerous cross-bubble declassification



## Future plans

- Healthcare application
- Developer study planned
- Possible user studies as well

## Reasoning about protocols with human participants and physical objects

Task lead: Jonathan Katz

Hard problem(s): Human behavior,  
resilient architectures

### The challenge

- Traditional protocol analysis limited to analyzing *computers* exchanging *electronic messages*
- This task: extend this to explicitly model *human users* (+ computers) exchanging *physical objects* (+ electronic messages)
  - With the *Remotegrity* voting protocol as an initial test case

## Why Remotegrity?

- Good example of a protocol explicitly designed with human users in mind, and with physical objects inherent
- Practical impact
  - Used in Takoma Park municipal elections (2011)

## Why voting protocols?

- Several jurisdictions world-wide considering some form of end-to-end verifiable voting
  - Such protocols must take human users into account
  - Such protocols must include a physical component for post-election auditing

## Scantegrity II

- Scantegrity II is an *end-to-end verifiable* voting protocol
  - Vote privacy if majority of trustees are honest
  - Vote integrity -- voters and independent auditors can verify that all votes are counted
  - Dispute resolution: in case of dispute, a third party can determine who is cheating

## Remotegrity

- Adds support for *remote/absentee voting* on top of Scantegrity
  - Can also be used with other paper-ballot systems
- In addition to added functionality required, an additional concern is possible *malicious software* on voters' (home) computers
  - Cannot treat voters' computers as trustworthy

## Traditional security models



Voter  
(poly time)



Election  
trustees

## Our model



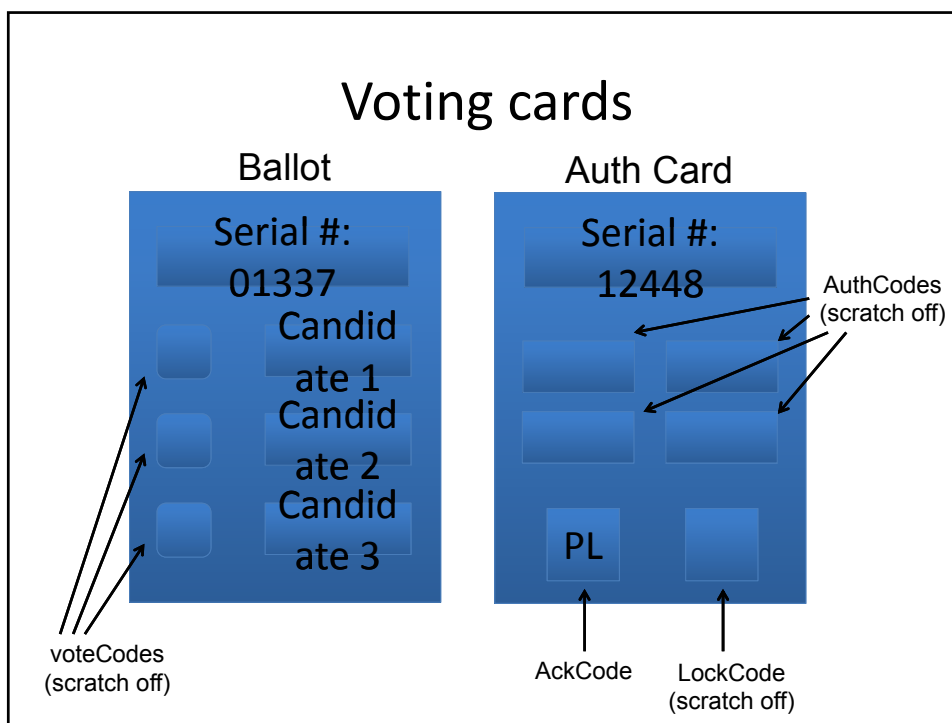
Voter  
(human!)



Local  
terminal  
(may be malicious!)



Election  
trustees



## Scratch-off cards

- Physical object
- Assumptions:
  - Value cannot be read until relevant portion of card is scratched off
  - Once an area is scratched off, it cannot be undone

## More broadly (future work)

- General results on what is possible using protocols with humans and physical objects
  - Can we design protocols with limited (or no) trusted computers, relying on “human computation” only?
  - What types of functionality can we achieve using physical objects (and weaker, or no, cryptographic assumptions)?

## **Trust, recommendation systems, and collaboration**

**Task lead: John Baras**

**Hard problem(s): Policy-governed collaboration,  
human behavior**

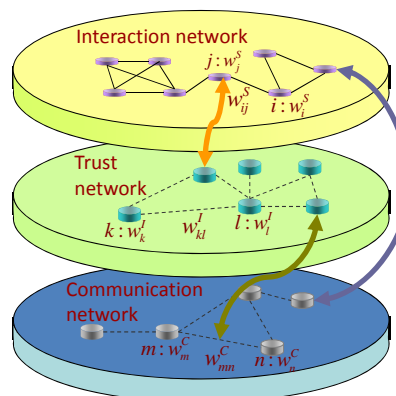


## Overview

- Develop a theory of *trust*, including its impact on collaboration in dynamic, networked, multi-agent systems
  - Understand effect of *malicious* attacks on trust inference (i.e., attempts to influence trust improperly)
- Take human behavior into account

## General model

- Multiple interacting graphs
  - *Nodes*: agents, groups, organizations
  - Directed graphs
  - *Links*: ties, relationships
  - *Weights on links* : value (strength, significance) of tie
  - *Weights on nodes* : importance of node (agent)
- Dynamic, time-varying graphs (relations, weights, policies)



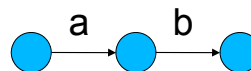
## Indirect trust

- How to establish a trust relation between users  $i, j$ , that have not had prior direct interaction?
- Trust computation: path problem on a graph
  - Look at (directed) paths from  $i$  to  $j$ ; combine information along each path

## Trust semiring properties

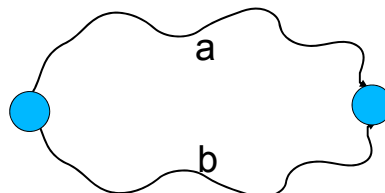
- Semiring  $(\mathbb{R}^+, \oplus, \otimes)$

–  $\otimes$  = sequential composition



– Axiom:  $a \otimes b \leq a, b$

–  $\oplus$  = parallel composition



– Axiom:  $a \oplus b \geq a, b$

- Can study abstract properties of such systems

## Modeling

- How well does any particular set of operations model human perceptions of trust?

## Lablet participants

- Adam Aviv, USNA
- John Baras
- Marshini Chetty
- Michael Clarkson, Cornell
- Michel Cukier
- Tudor Dumitras
- Jeff Foster
- Jen Golbeck
- Michael Hicks
- David Van Horn
- Joseph JaJa
- David Maimon
- Babis Papamanthou
- Aditya Prakash, VA Tech
- Elaine Shi
- Katie Shilton
- VS Subrahmanian
- Mohit Tiwari, UT Austin
- Sam Tobin-Hochstadt, IU
- Poorvi Vora, GWU

**Questions?**