# Applying formal methods to complex problems in human-systems interaction

Adam M. Houser

PhD Candidate,
Department of Industrial and Systems Engineering

University at Buffalo The State University of New York

# Talk Outline

- Overview of education and background, Buffalo and prior
- Technical experience 0: NASA NextGen airspace management project
- Technical experience 1: Mental models in cybersecurity
- Contact information, Q&A

# Speaker background

- PhD candidate, human factors engineering (expected Sept 2018)
  - MS, Industrial Engineering, 2015
  - MAE, Secondary Science Education (Physics, Chemistry), 2012
  - BA, Applied Philosophy (Epistemology, Analytic Philosophy), 2010
- RA, Formal Human Systems Lab
- Junior Cognitive Systems Engineer, Resilient Cognitive Solutions

# Motivation: why formal methods?

- Complex, safety-critical systems: systems, operators, and the world (dynamic)

- Human error as the “cause” or “major contributing factor” of system failure
  - AF447, CA3407, Therac-25, Three Mile Island, USS John S McCain, ...
  - 70% - 80% of civil and military aviation accidents (FAA, 2001)
  - >250,000 deaths *per annum* due to medical error (The BMJ, 2016)

- Often result from complex, unanticipated human-systems interaction

- FM: discovery of unanticipated interactions through exhaustive statespace search

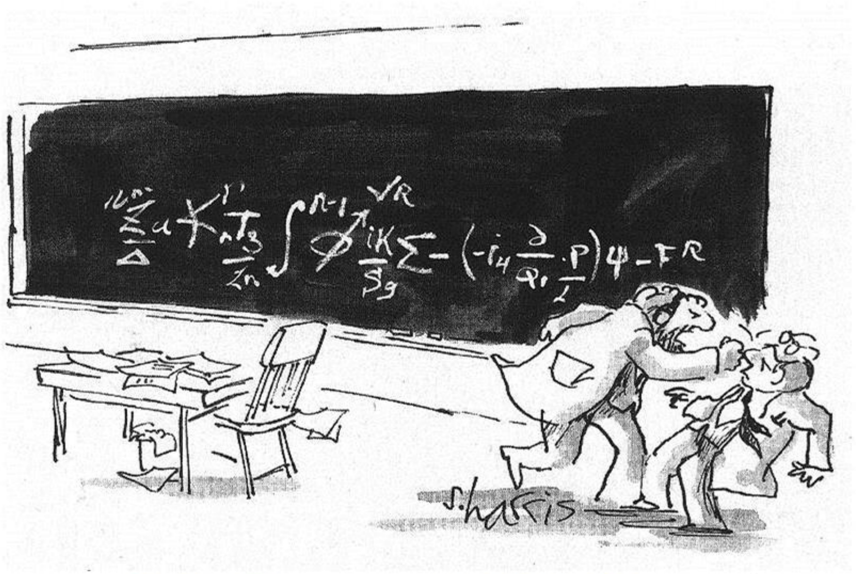
# Formal methods and model checking

- Well-defined mathematical languages and techniques for modeling, specifying, and verifying systems

- Models: mathematical description of target system behavior

- Specifications: logical assertion of desirable system behaviors as properties

- Verification: mathematical *proof* about whether the model satisfies the specifications

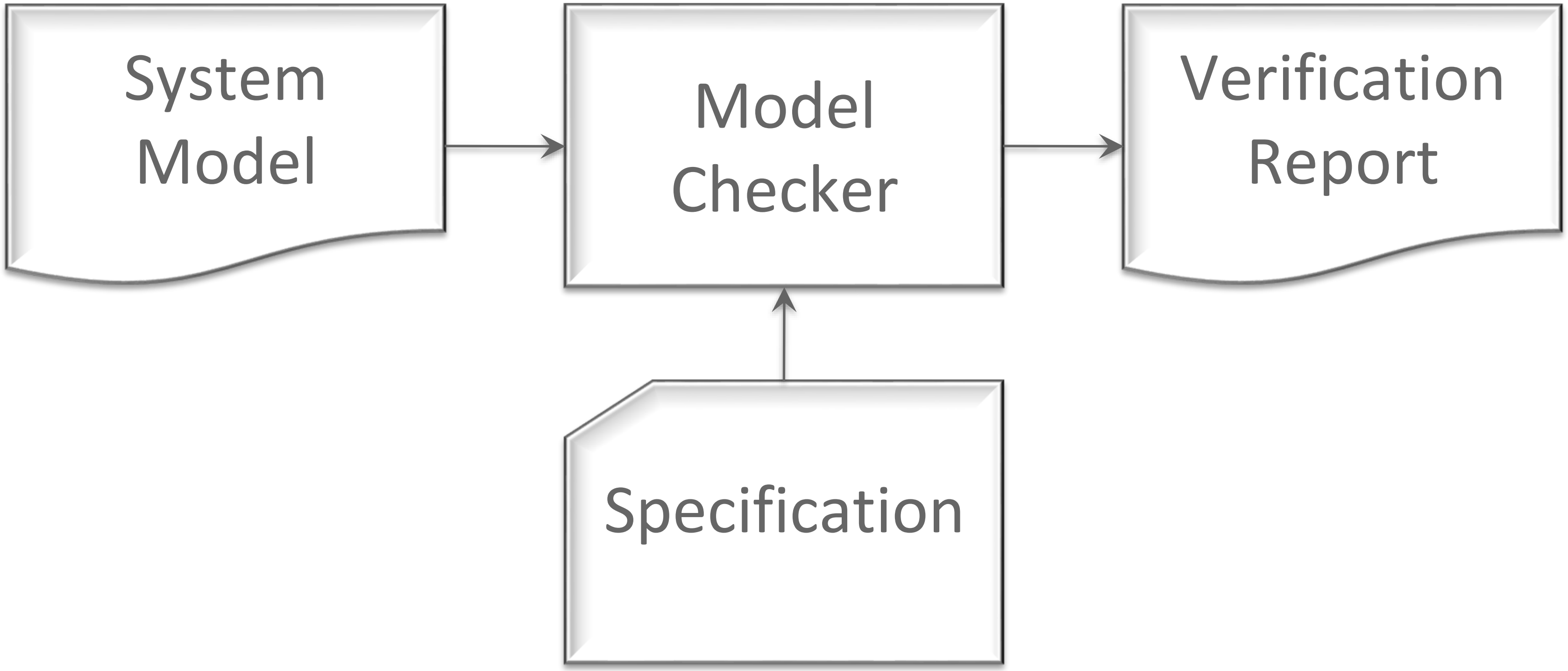


YOU WANT PROOF? I'LL GIVE YOU PROOF!

University at Buffalo The State University of New York

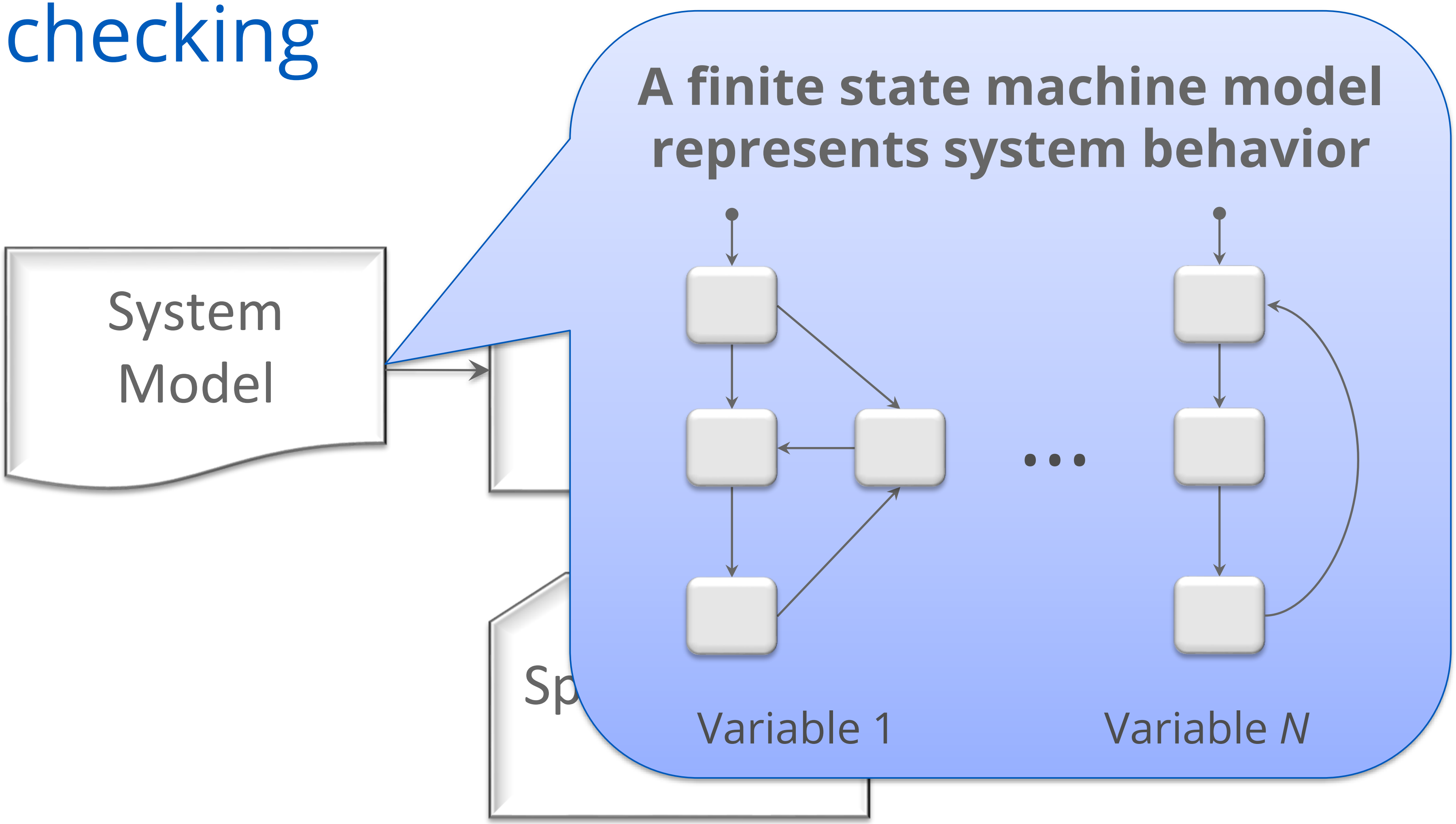
# Model checking

An automatic means of performing formal verification



University at Buffalo The State University of New York

# Model checking



University at Buffalo The State University of New York

# Model checking

System Model

**A temporal logic specification property asserts desirable qualities about the system**

For example: “The system should never reach unsafe state X”

$$G \neg (X)$$

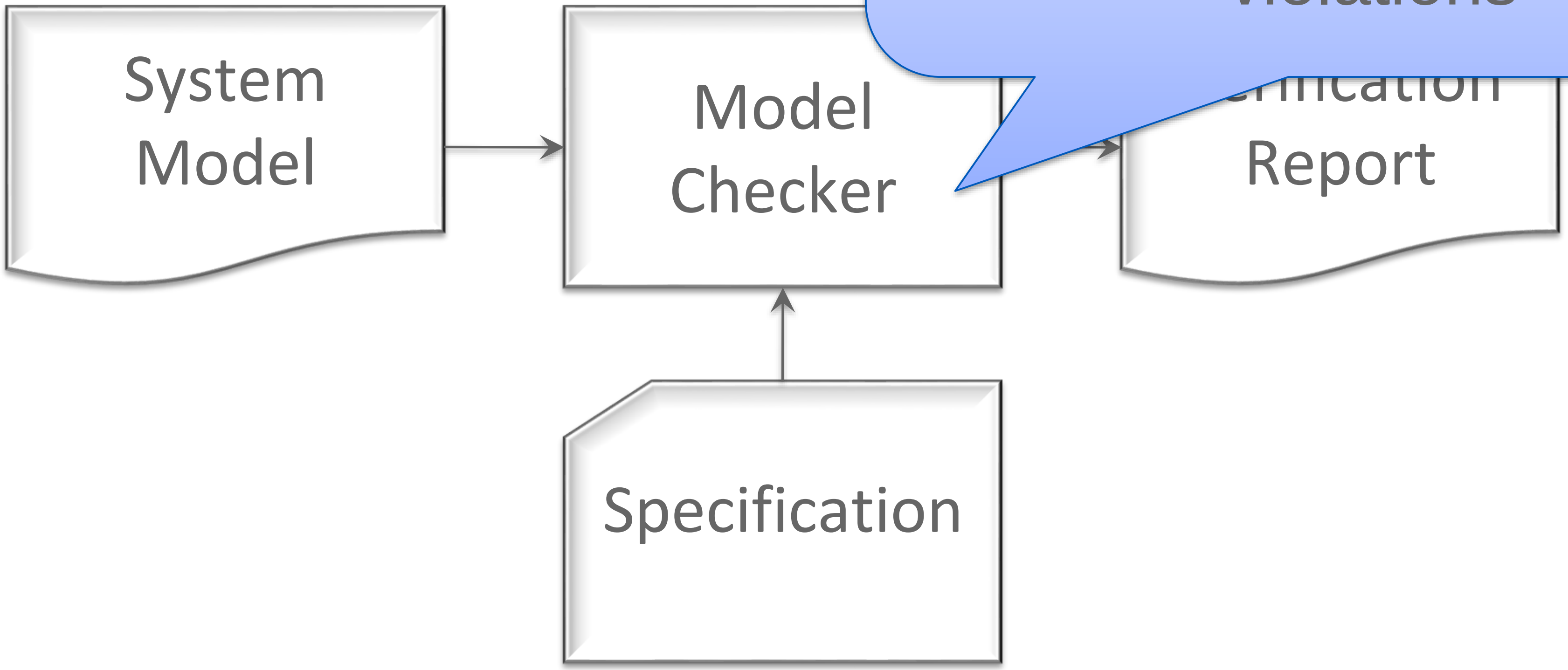
Or, “The system should always eventually reach state Y”

$$F (Y)$$

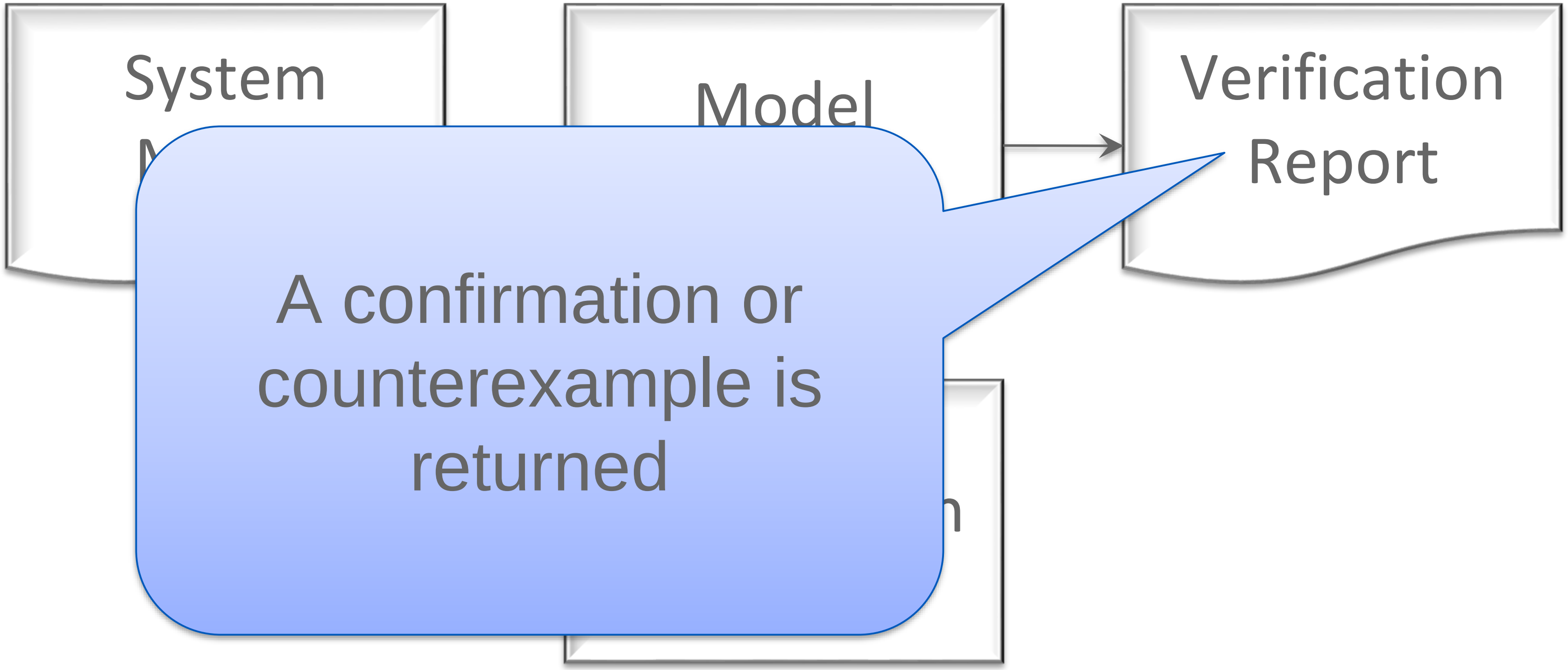
Specification

# Model checking

A model checker "searches" through the model's statespace looking for violations

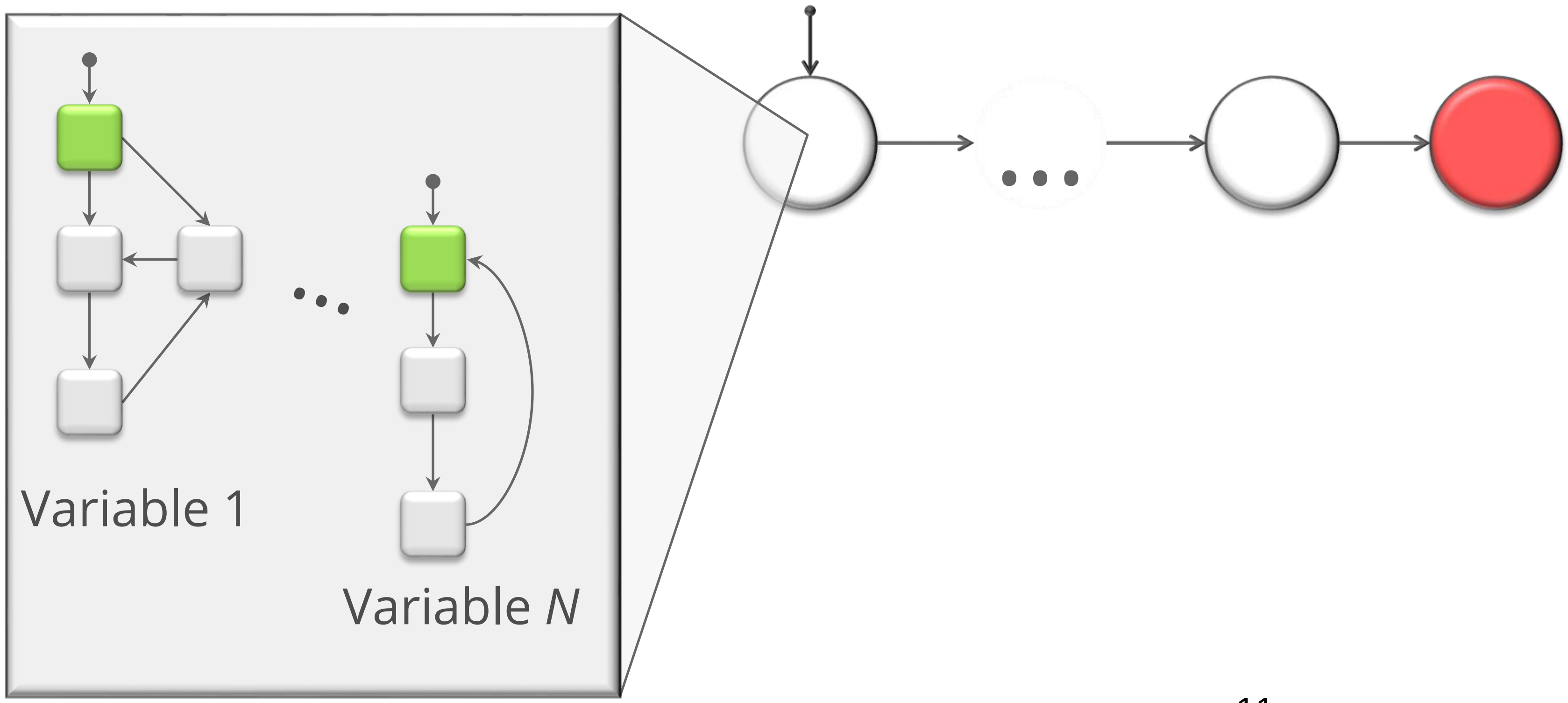


# Model checking



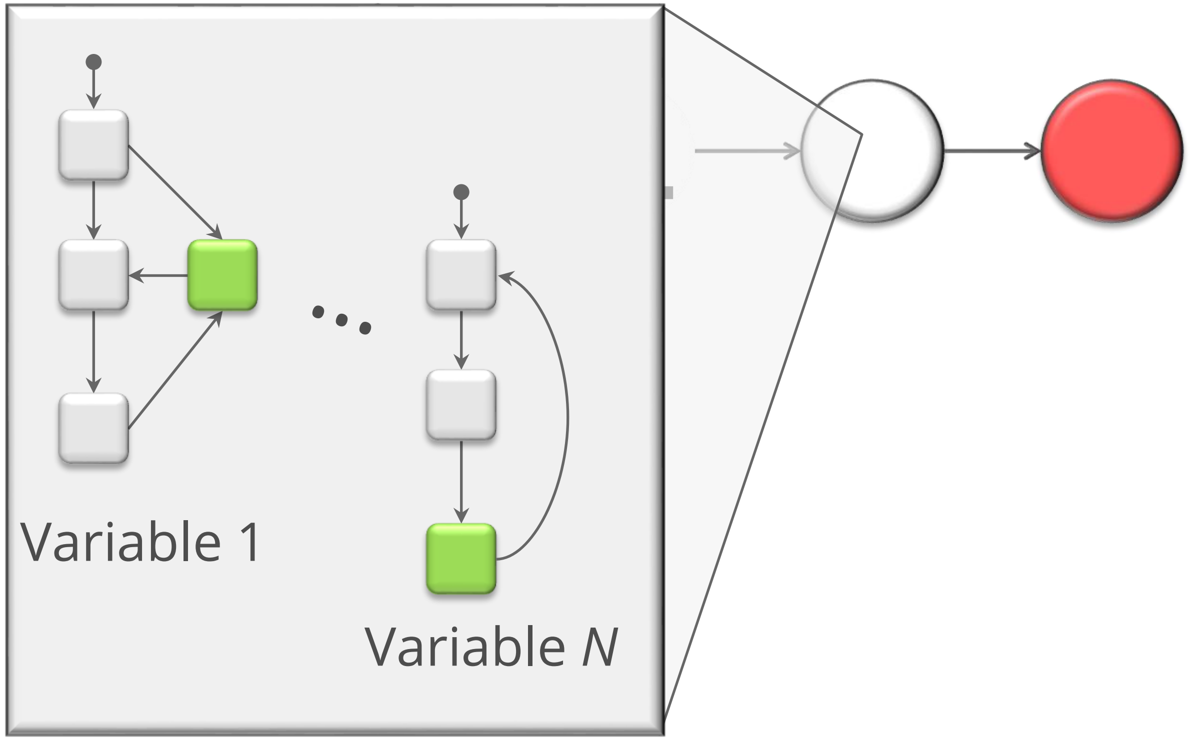
# Counterexample

A sequence of states that lead up to a violation



# Counterexample

A sequence of states that lead up to a violation






12

# Limitations of these techniques

- Statespace explosion and scalability
- Limited expressive power
- Models are only robust to the properties that have been captured

University at Buffalo The State University of New York

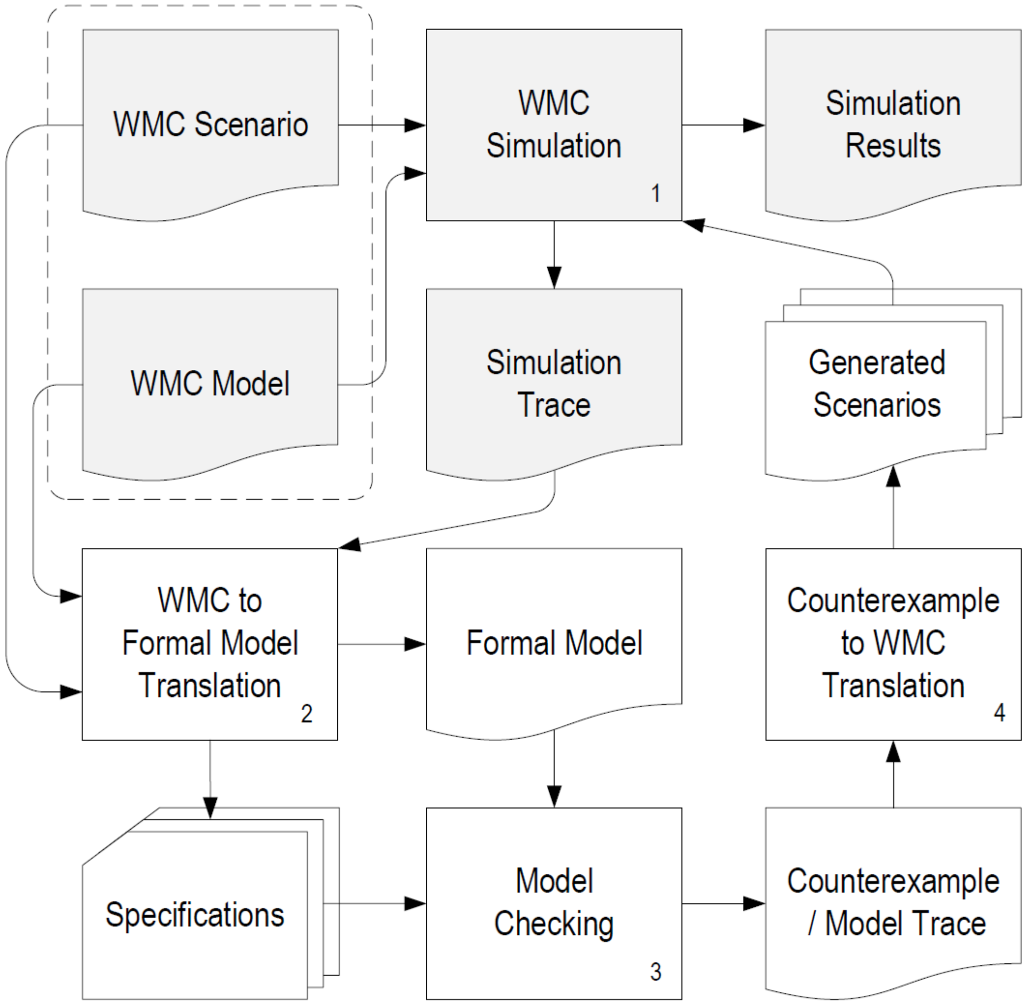
# NASA NextGen airspace management

Synergistically using formal methods and simulation to search for excessive pilot workload scenarios

# NASA NextGen: Simulation and formal methods

- NextGen AMS: introducing more autonomy into airspace mgmt
  - Function allocation changes between ATC, pilots, and automation
  - Also changes autonomy, authority, and responsibility
  - Distributed, complex, safety-critical system
- Problem 1: how can we synergistically use formal methods and simulation to discover these events?
- Problem 2: are there combinations of actions/events allocated to human agents that could result in unsafe operating conditions?
- Problem 3: what can we recommend to mitigate these conditions?

# NASA NextGen: Simulation and formal methods architecture



# NASA NextGen: Discovering unsafe conditions

$$
\begin{array}{l}
FindExcessiveDelay \models \\
\mathbf{G}\left(\begin{array}{l}(actions[j].state \neq notAssigned) \\ \Rightarrow \left(\left(\begin{array}{l} globalTime \\ -actions[j].update \end{array}\right) < timeMax\right)\end{array}\right).
\end{array}
$$

$$
\begin{array}{l}
FindNoOverload \models \\
\mathbf{G}\neg\left(\begin{array}{l}\left(\begin{array}{l} status = doing \vee status \neq doing \\ \wedge \left(\begin{array}{l}\left(\begin{array}{l} cardinality(delayed) \\ +cardinality(interrupted) \end{array}\right) \\ \leq agent[i].inactiveCapacity \end{array}\right)\end{array}\right) \\ \mathbf{U}\,(globalTime \geq Never)\end{array}\right).
\end{array}
$$

# NASA NextGen: Results and recommendations

University at Buffalo The State University of New York

# Dissertation: Formal methods, mental models, and cybersecurity

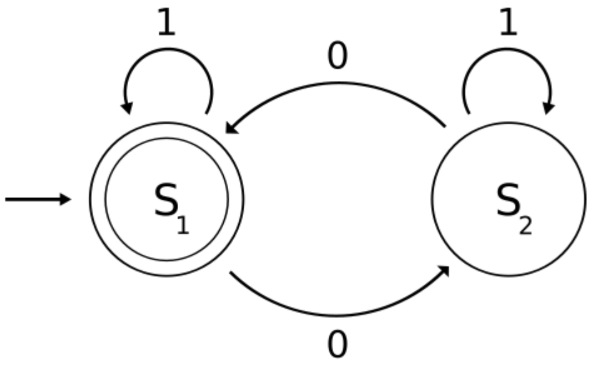
Discovering unanticipated human-systems interaction to recommend attacker mitigations

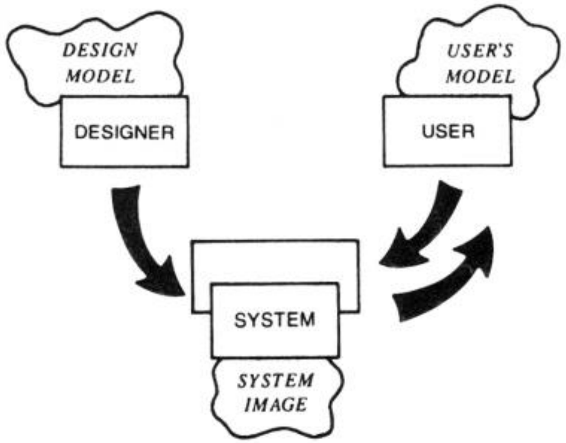
# Mental models in human factors engineering

- Internalized representations of system functionality
- Different representational strategies:
  - "Pictures in the mind" (de Kleer & Brown, 1981)
  - Descriptive system abstractions (Rasmussen, 1971; Rouse & Hunt, 1986)
  - "Structured knowledge" (Dutton & Starbuck, 1971)
- Strategies are not mutually exclusive (Sanderson, 1990)

# Mental models in human factors engineering

- For this work, Norman (1983) outlines key aspects:
  - “Runnability” of mental models
  - Agreement between the user’s model and the system image (Norman, 1986)

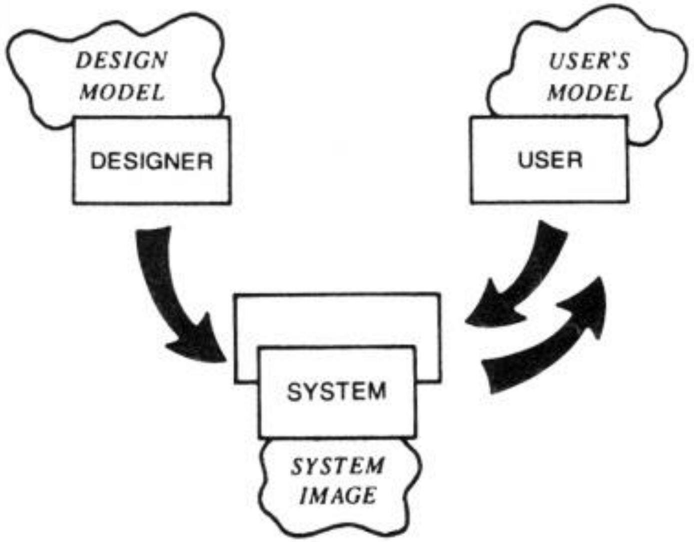


# Examples of analysis with formal methods

- Particular success with finding user-system mismatches for safety
  - Aircraft autopilot (Degani & Heymann, 2002)
  - Aircraft autoland (Oishi, et al., 2002)
  - Vehicle cruise control (Degani, 2004)

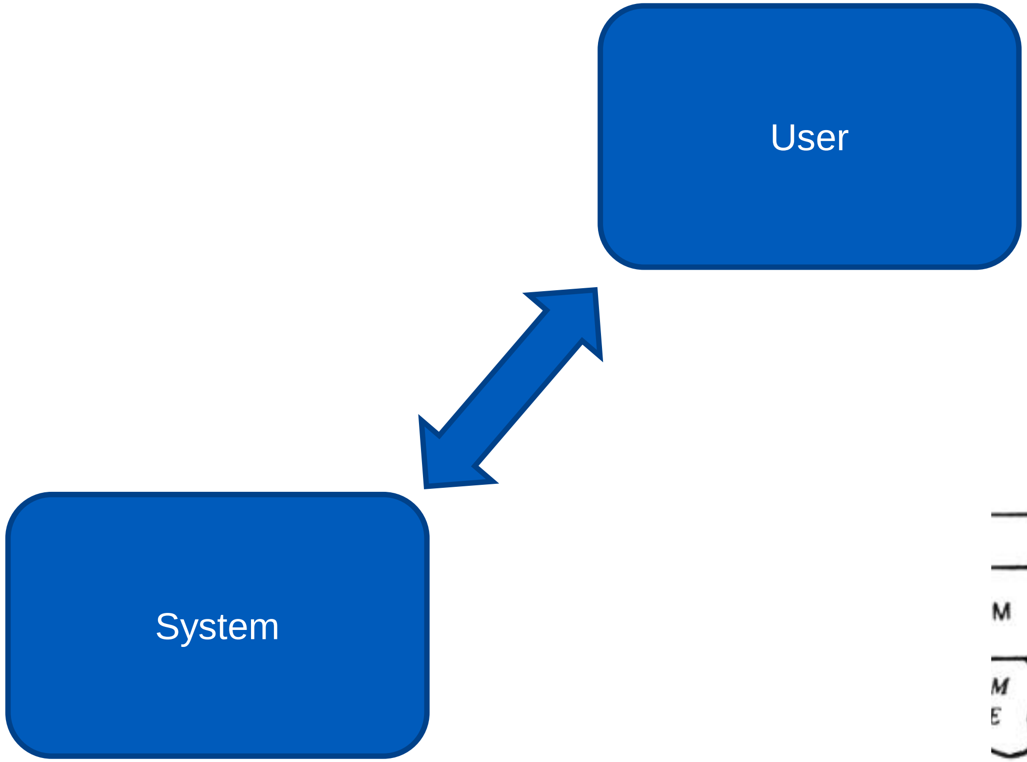
- Discovery of <u>unanticipated</u> user-system mismatches through exhaustive statespace search

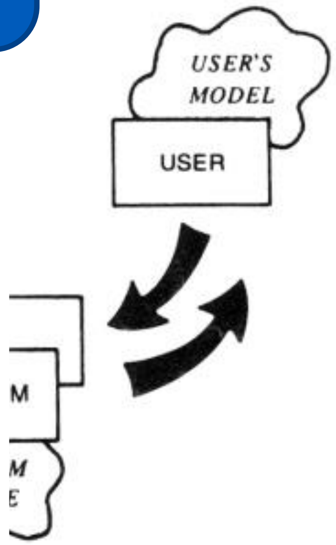


# My research objective

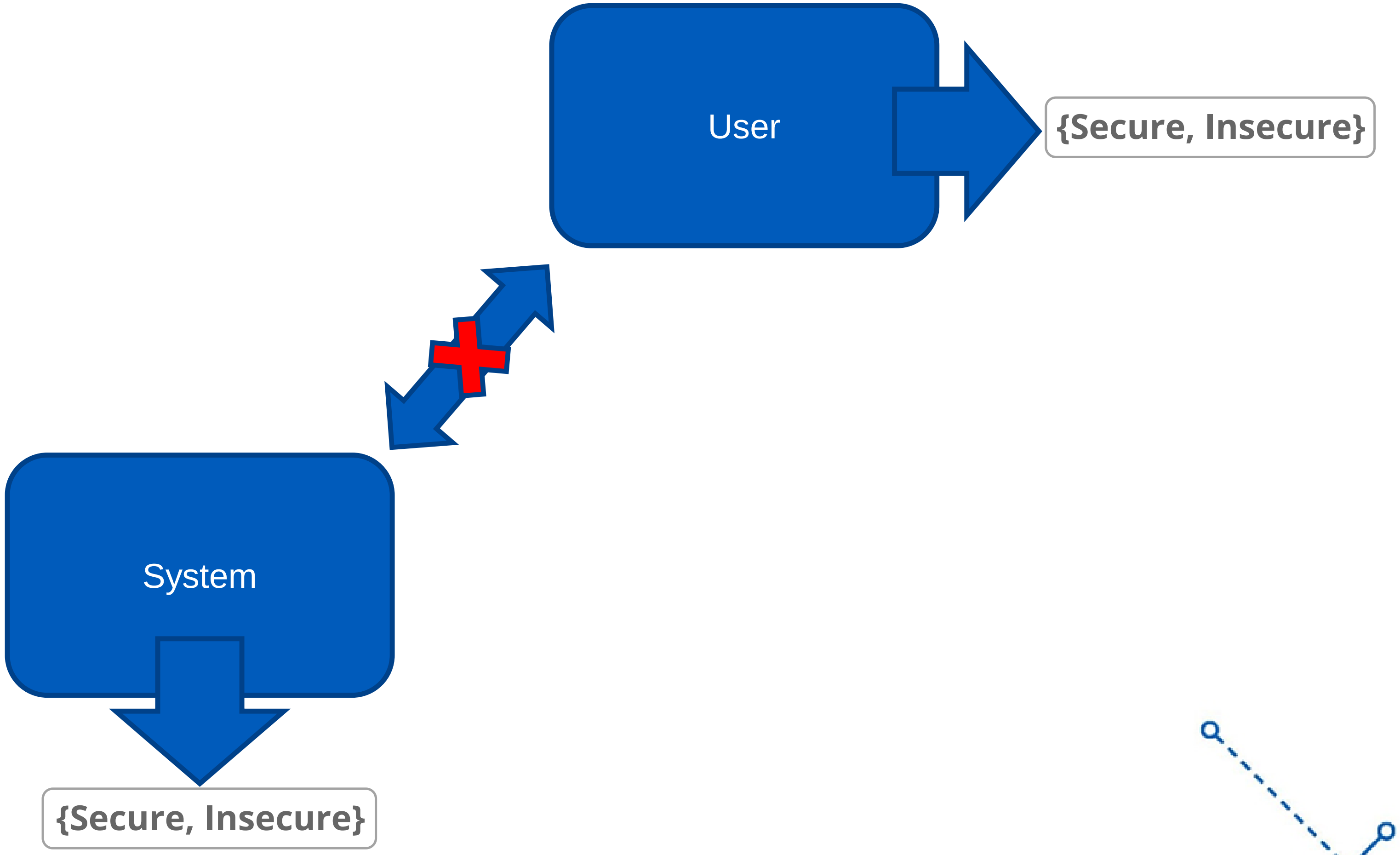
By synergistically integrating work from human factors, cybersecurity, and formal methods, we can **discover unanticipated interactions** between user mental models and program features or behaviors that are exploitable by attackers.

By identifying and describing these interactions, we can **recommend interface changes or software patches** to mitigate their harmful effects.

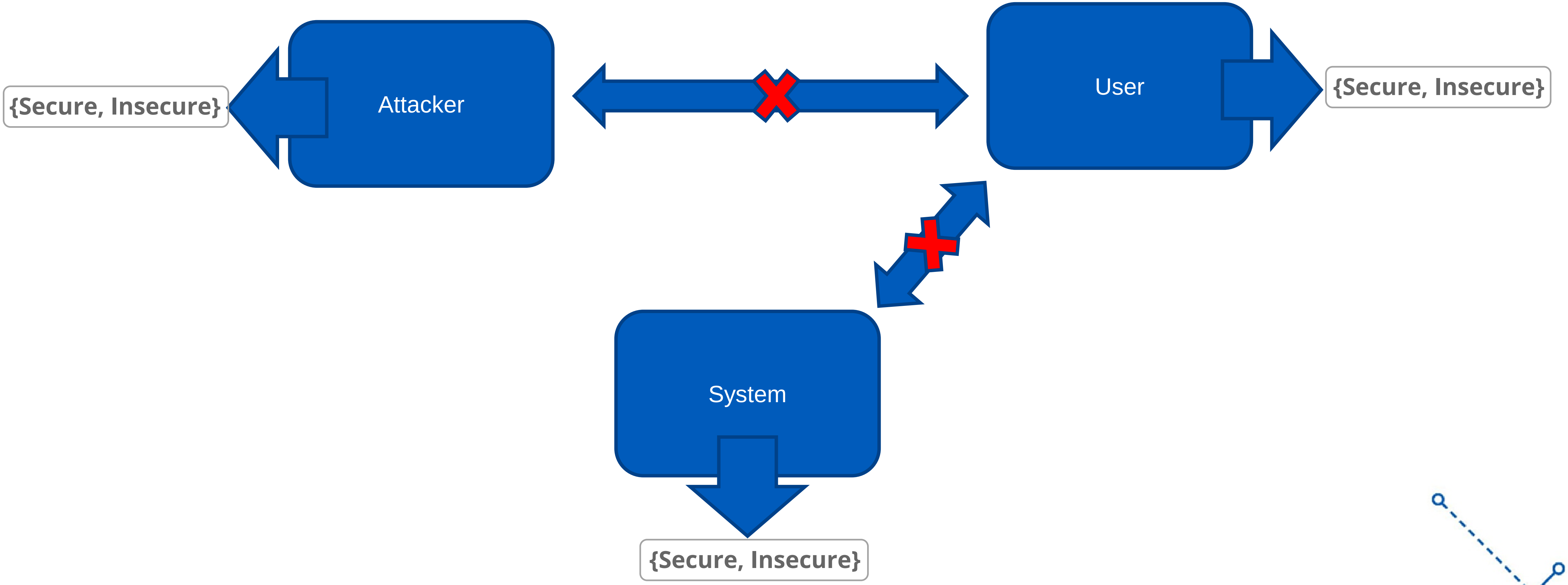

User
System


USER'S
MODEL
USER

# Phase I model architecture



# Phase II model architecture



# Component 1: User models

## How do we capture “user behavior” in a formal model?

| | | *Virus Models* | | | | *Hacker Models* | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Viruses are Bad | Buggy Software | Mischief | Support Crime | Graffiti | Burglar | Big Fish | Contractor |
| *1.* | Use anti-virus software | ?? | xx | ?? | !! | | !! | xx | xx |
| *2.* | Keep anti-virus updated | xx | xx | ?? | !! | | | | xx |
| *3.* | Regularly scan computer with anti-virus | xx | xx | ?? | !! | | | | xx |
| *4.* | Use security software (firewall, etc.) | xx | | ?? | | ?? | ?? | ?? | xx |
| *5.* | Don’t click on attachments | !! | !! | !! | !! | !! | !! | | |
| *6.* | Be careful downloading from websites | ?? | !! | ?? | !! | ?? | ?? | xx | xx |
| *7.* | Be careful which websites you visit | | xx | !! | ?? | !! | !! | ?? | !! |
| *8.* | Disable scripting in web and email | | | | | | | | xx |
| *9.* | Use good passwords | | | | | ?? | | ?? | xx |
| *10.* | Make regular backups | | ?? | !! | xx | !! | xx | xx | xx |
| *11.* | Keep patches up to date | | ?? | xx | !! | !! | !! | xx | xx |
| *12.* | Turn off computer when not in use | | xx | xx | !! | ?? | !! | xx | xx |

| | | |
|---|---|---|
| !! | Important | It is very important to follow this advice |
| ?? | Maybe | Following this advice might help, but it isn’t all that important to do |
| xx | Not Necessary | It is not necessary to follow this advice |
| | Not Applicable | This model does not have anything to say about this advice, or there is insufficient data from the interviews to determine an opinion |

**Table 3: Summary of Expert Security Advice. Each folk model responds to this advice differently.**

Everyone understood the need for care in choosing what to download. Downloads were strongly associated with viruses in most respondents’ minds. However, only users with well-developed models of viruses (the *Mischief* and *Support Crime* models) believed that viruses can be “caught” simply by browsing web pages. People who believed that viruses were buggy software didn’t see browsing as dangerous because they weren’t actively clicking on anything to run it.

Wash, 2010. “Folk models of home computer security,” p. 10.

University at Buffalo The State University of New York

# Component 2: Attacker models

How do we capture attacker strategies (TTPs) in a formal model?

ATT&CK™ Adversarial Tactics, Techniques & Common Knowledge

Main page
Help
Contribute
References

Tactics
- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Execution
- Collection
- Exfiltration
- Command and Control

Techniques
- All Techniques
- Technique Matrix

Groups
- All Groups

Software
- All Software

Tools
- Printable version
- Permanent link

Follow @MITREattack

Page | Discussion

## Windows Technique Matrix

| Persistence | Privilege Escalation | Defense Evasion | Credential Access | Discovery | Lateral Movement |
|---|---|---|---|---|---|
| Accessibility Features | Access Token Manipulation | Access Token Manipulation | Account Manipulation | Account Discovery | Application Deployment Software |
| AppInit DLLs | Accessibility Features | Binary Padding | Brute Force | Application Window Discovery | Exploitation of Vulnerability |
| Application Shimming | AppInit DLLs | Bypass User Account Control | Create Account | File and Directory Discovery | Logon Scripts |
| Authentication Package | Application Shimming | Code Signing | Credential Dumping | Network Service Scanning | Pass the Hash |
| Bootkit | Bypass User Account Control | Component Firmware | Credentials in Files | Network Share Discovery | Pass the Ticket |
| Change Default File Association | DLL Injection | Component Object Model Hijacking | Exploitation of Vulnerability | Peripheral Device Discovery | Remote Desktop Protoc |
| Component Firmware | DLL Search Order Hijacking | DLL Injection | Input Capture | Permission Groups Discovery | Remote File Copy |
| Component Object Model Hijacking | Exploitation of Vulnerability | DLL Search Order Hijacking | Network Sniffing | Process Discovery | Remote Services |
| DLL Search Order Hijacking | File System Permissions Weakness | DLL Side-Loading | Private Keys | Query Registry | Replication Through Removable Media |
| External Remote Services | Local Port Monitor | Deobfuscate/Decode Files or Information | Two-Factor Authentication Interception | Remote System Discovery | Shared Webroot |
| File System Permissions | New Service | Disabling Security Tools | | Security Software | Taint Shared Content |

# Results from Phase I analysis

- Smaller-scale version: searching for potentially dangerous and unexpected human-systems interactions
- Use case: risks posed by receiving malicious URLs on a mobile device
- User model leverages “big fish” folk model, clicks with little regard to device safety

# Results from Phase I analysis

- “Big Fish” victims resilient to neither phishing attacks nor drive-by downloads, passive compromise, etc
- Open to many different avenues of attack
- Little user regard for inconveniences posed by mobile IU (ex: hovering over links, URL appearance in omnibar)

# Capturing user behavior and mental models

```
DEFINITION
    siteContent =
        IF sensitiveInfo = TRUE THEN personal
        ELSIF sensitiveInfo = FALSE THEN basic
        ELSE personal
        ENDIF;

    user =
        IF omnibarLength = compact AND hoverPossible = FALSE and sensitiveInfo = TRUE THEN vulnerable
        ELSIF omnibarLength = compact AND hoverPossible = FALSE and sensitiveInfo = FALSE THEN vulnerable
        ELSIF omnibarLength = compact AND hoverPossible = TRUE and sensitiveInfo = TRUE THEN vulnerable
        ELSIF omnibarLength = compact AND hoverPossible = TRUE and sensitiveInfo = FALSE THEN safe
        ELSIF omnibarLength = full AND hoverPossible = FALSE and sensitiveInfo = TRUE THEN vulnerable
        ELSIF omnibarLength = full AND hoverPossible = FALSE and sensitiveInfo = FALSE THEN vulnerable
        ELSIF omnibarLength = full AND hoverPossible = TRUE and sensitiveInfo = TRUE THEN vulnerable
        ELSIF omnibarLength = full AND hoverPossible = TRUE and sensitiveInfo = FALSE THEN safe
        ELSE safe
        ENDIF;
```

# Capturing user behavior and mental models

| | | *Virus Models* | | | | *Hacker Models* | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Viruses are Bad | Buggy Software | Mischief | Support Crime | Graffiti | Burglar | Big Fish | Contractor |
| *1.* | Use anti-virus software | ?? | xx | ?? | !! | | !! | xx | xx |
| *2.* | Keep anti-virus updated | xx | xx | ?? | !! | | | | xx |
| *3.* | Regularly scan computer with anti-virus | xx | xx | ?? | !! | | | | xx |
| *4.* | Use security software (firewall, etc.) | xx | | ?? | | ?? | ?? | ?? | xx |
| *5.* | Don't click on attachments | !! | !! | !! | !! | !! | !! | | |
| *6.* | Be careful downloading from websites | ?? | !! | ?? | !! | ?? | ?? | xx | xx |
| *7.* | Be careful which websites you visit | | xx | !! | ?? | !! | !! | ?? | !! |
| *8.* | Disable scripting in web and email | | | | | | | | xx |
| *9.* | Use good passwords | | | | | ?? | | ?? | xx |
| *10.* | Make regular backups | | ?? | !! | xx | !! | xx | xx | xx |
| *11.* | Keep patches up to date | | ?? | xx | !! | !! | !! | xx | xx |
| *12.* | Turn off computer when not in use | | xx | xx | !! | ?? | !! | xx | xx |

| | | |
|---|---|---|
| !! | Important | It is very important to follow this advice |
| ?? | Maybe | Following this advice might help, but it isn't all that important to do |
| xx | Not Necessary | It is not necessary to follow this advice |
| | Not Applicable | This model does not have anything to say about this advice, or there is insufficient data from the interviews to determine an opinion |

**Table 3: Summary of Expert Security Advice. Each folk model responds to this advice differently.**

Everyone understood the need for care in choosing what to download. Downloads were strongly associated with viruses in most respondents' minds. However, only users with well-developed models of viruses (the *Mischief* and *Support Crime* models) believed that viruses can be "caught" simply by browsing web pages. People who believed that viruses were buggy software didn't see browsing as dangerous because they weren't actively clicking on anything to run it.

# Remaining work

- Complete Phase I analysis (additional properties, if any)
- Refine into Phase II architecture (particular focus on attacker tradecraft)
- Write everything up

University at Buffalo The State University of New York

# Questions?

Adam M. Houser

appliedcaffeine.org

adamhous@buffalo.edu

@neutrinos4all

# Reserve Slides

Type definitions allow model concepts to be defined with domain specific values.

```
microwave3 : CONTEXT =
BEGIN

%hardware model variables are below:
    doorState                : TYPE = {Open, Closed};
    heatingElementState      : TYPE = {On, Off};
    timerState               : TYPE = {Expired, Running, Paused};
    buttonPressed            : TYPE = {nothingPressed, Start, Cancel};
    cookTimeInput            : TYPE = {Entered, notEntered};
    foodStatus               : TYPE = {Uncooked, notCooking, Cooking, doneCooking};

%mental model variables are below:
    mTimerState              : TYPE = {mExpired, mRunning, mPaused};
    mfoodStatus              : TYPE = {mCooking, mNotCooking, mDoneCooking};

    Door : MODULE =
        BEGIN
            INPUT   Actuate : BOOLEAN
            OUTPUT  Door    : doorState

            INITIALIZATION
                Door = Closed;

            TRANSITION
                Door' =
                    IF      Actuate
                            AND Door = Closed
                            OR NOT Actuate
                            AND Door = Open
                    THEN    Open
                    ELSE
                            Closed
                    ENDIF;
        END;

    Element : MODULE =
        BEGIN
            INPUT   Door        : doorState
            INPUT   Button      : buttonPressed
            INPUT   Timer       : timerState
```

Modules represent components of the system that work together to achieve required system behavior.

Transition statements describe the behavior of those components in formal representations.

Fig 1. Snippet of a formal model.

Symbolic Analysis Laboratory

The model composition statement describes how each of the modules will be composed for checking:
synchronously || 
or
asynchronously []

Specifications use LTL to check safety or behavioral properties, as well as liveness (freedom from deadlock), reachability (attainability of all nodes in the graph), and other properties.

```
Microwave : MODULE = Door || Element || Timer || HumanMentalModel;

Facemelt        : THEOREM Microwave |- G((Door = Closed AND Heating = On AND Timer = Running) => G(NOT(Door = Open AND Heating = On AND Timer = Running)));
Autostart       : THEOREM Microwave |- G(NOT(Heating = On AND Door = Closed) AND HumanAction = notEntered);
Basic           : THEOREM Microwave |- G(NOT(Heating = Off AND Door = Closed AND Timer = Running));
Runaway         : THEOREM Microwave |- G(NOT(Heating = On AND Timer = Paused AND Button = Start AND Door = Open));
```

Fig 2. Example specifications.

During model checking, SAL will programmatically translate the model into a finite state machine...

```
number of system variables: 18, number of auxiliary variables: 6
converting flat module to BDD representation (initial states, and transition relation)...
  creating BDD variables...
  computing static variable ordering (minimizing support)...
    collecting state variables dependencies...
  static order time: 0.0 secs
  number of BDD variables: 48
  creating definition section BDDs...
  creating valid state predicate BDDs...
  creating BDD: set of initial states...
  creating BDD: transition relation...
  rearranging clusters...
  reordering BDD variables...
  transition relation - size: 308 (nodes), number of clusters: 1
  compressing BDD clusters...
  rearranging clusters...
  transition relation - size: 308 (nodes), number of clusters: 1
    cluster compression time: 0.0 secs
  statistics:
    number of nodes in initial states: 18
    number of nodes in transition relation: 308
    transition relation detailed information:
      monolithic cluster size: 308 nodes
  flat-module -> BDD conversion time: 0.04 secs
proving invariant or producing counterexample using BDDs...
  using forward search
  iteration: 1
  frontier lower bound: 18 nodes, upper bound: 18 nodes
  using frontier with 18 nodes
  total bdd node count: 434
  iteration: 2
  frontier lower bound: 18 nodes, upper bound: 16 nodes
  using frontier with 16 nodes
  total bdd node count: 462
  number of visited states: 24.0
  verification time: 0.0 secs
proved.
total execution time: 0.04 secs
```

... search the state transition diagram for a path through the diagram or condition satisfying the specification ...

... and return a proof if the specification holds.

Fig 3. Snippet of a proof.

If the specification does not hold, SAL will begin building a counterexample…

…that captures the state of all variables at each step…

… the path through the transition system that led to the violation of that specification …

… and how long it took to execute the entire process.

```
  total bdd node count: 402
  number of visited states: 36.0
  INVALID, building counterexample...
  verification time: 0.0 secs
Counterexample:
========================
Path
========================
Step 0:
--- Input Variables (assignments) ---
Actuate = true
--- System Variables (assignments) ---
ba-pc!1 = 1
Button = Cancel
Door = Closed
Food = Uncooked
Heating = Off
HumanAction = notEntered
Timer = Expired
mDoor = Open
mFood = Uncooked
mHumanAction = notEntered
mTimer = mExpired
------------------------
Transition Information:
(module instance at [Context: microwave3, line(148), column(27)]
  ((module at [Context: microwave3, line(127), column(23)]
   (module instance at [Context: microwave3, line(127), column(42)]
     else transition at [Context: microwave3, line(89), column(11)]))
   (module instance at [Context: microwave3, line(127), column(51)]
     else transition at [Context: microwave3, line(122), column(13)])))
------------------------
Step 1:
--- Input Variables (assignments) ---
Actuate = false
--- System Variables (assignments) ---
ba-pc!1 = 0
Button = Cancel
Door = Open
Food = Uncooked
Heating = Off
HumanAction = notEntered
Timer = Expired
mDoor = Open
mFood = Uncooked
mHumanAction = notEntered
mTimer = mExpired
total execution time: 0.04 secs
```

Fig 4. Snippet of a counterexample.

# Code snippets: system-level behavior

```
DEFINITION
    omnibarLength =
        IF browserType = mobile THEN compact
        ELSIF browserType = tablet THEN full
        ELSIF browserType = desktop THEN full
        ELSE compact
        ENDIF;

    hoverPossible =
        IF urlLength = long AND omnibarLength = full THEN TRUE
        ELSIF urlLength = long AND omnibarLength = compact THEN FALSE
        ELSIF urlLength = short AND omnibarLength = full THEN TRUE
        ELSIF urlLength = short AND omnibarLength = compact THEN FALSE
        ELSE FALSE
        ENDIF;

    software =
        IF softwarePatched = TRUE AND adBlocker = enabled AND siteContent = basic THEN secure
        ELSIF softwarePatched = TRUE AND adBlocker = enabled AND siteContent = personal THEN secure
        ELSIF softwarePatched = TRUE AND adBlocker = disabled AND siteContent = basic THEN insecure
        ELSIF softwarePatched = TRUE AND adBlocker = disabled AND siteContent = personal THEN insecure
        ELSIF softwarePatched = FALSE AND adBlocker = enabled AND siteContent = basic THEN insecure
        ELSIF softwarePatched = FALSE AND adBlocker = enabled AND siteContent = personal THEN insecure
        ELSIF softwarePatched = FALSE AND adBlocker = disabled AND siteContent = basic THEN insecure
        ELSIF softwarePatched = FALSE AND adBlocker = disabled AND siteContent = personal THEN insecure
        ELSE insecure
        ENDIF;
```

# Limitations of these techniques

- Statespace explosion and scalability
  - Abstraction, λ-Calculus, constraint application, lookup tables, ...
- Limited expressive power
  - Potential use of outboard tools (ex: simulation)
- Models are only robust to the properties that have been captured
  - Combefis, Giannakopoulou, Pecheur, & Feary, 2011
  - Bolton, Jimenez, van Paassen, and Trujillo, 2014

# Folk models in cybersecurity

- Similarities and differences between folk and mental models
  - Description of user expectations about system behavior
  - Folk models rely more heavily on metaphor (Camp, 2009)
  - Mental models more heavily emphasize runnability

- Some work moving towards mental models (Blythe & Camp, 2012)

# Folk models in cybersecurity

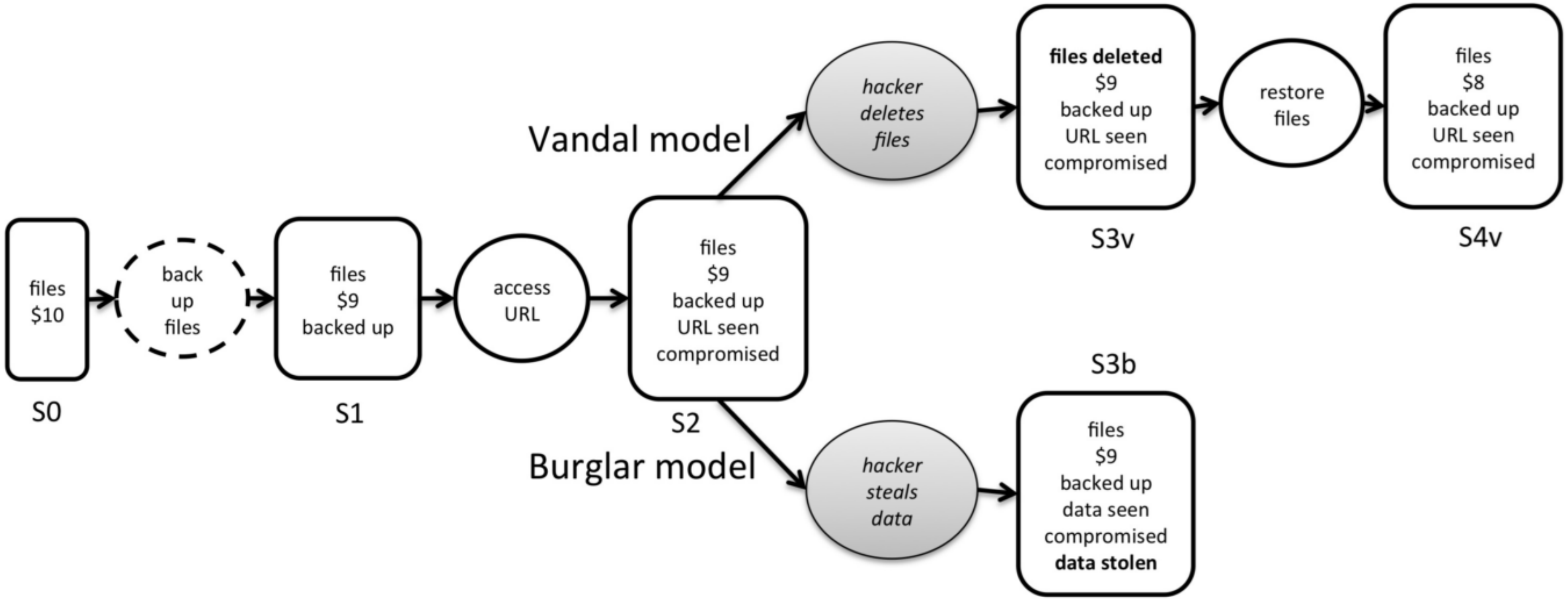
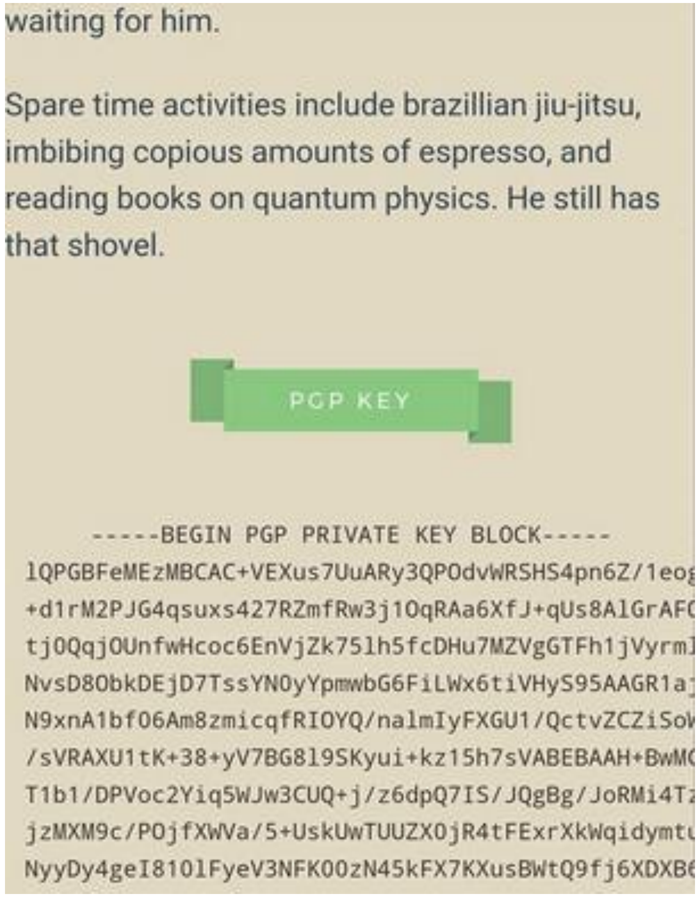



Figure 1. Simulation of a decision to “back up files” run against Wash (2010)’s vandal and burglar hacker models (Blythe & Camp, 2012, p. 89).

# Mental model elicitation

- There exist a number of methods for model extraction
  - Card-sorting tasks (Asgharpour, et al., 2007)
  - Structured and semi-structured interviews (Wash, 2010)
  - Task observations (Dutton & Starbuck, 1971)
  - Cognitive walkthroughs (Ford & Sterman, 1997)
  - Training artifact analysis (Rushby, 2001)

# End-user key management is still hard


waiting for him.

Spare time activities include brazillian jiu-jitsu, imbibing copious amounts of espresso, and reading books on quantum physics. He still has that shovel.

PGP KEY


```
-----BEGIN PGP PRIVATE KEY BLOCK-----
lQPGBFeMEzMBCAC+VEXus7UuARy3QPOdvWRSHS4pn6Z/1eo
+d1rM2PJG4qsuxs427RZmfRw3j1OqRAa6XfJ+qUs8AlGrAF
tj0QqjOUnfwHcoc6EnVjZk75lh5fcDHu7MZVgGTFh1jVyrm
NvsD8ObkDEjD7TssYN0yYpmwbG6FiLWx6tiVHyS95AAGR1a
N9xnA1bf06Am8zmicqfRIOYQ/nalmIyFXGU1/QctvZCZiSo
/sVRAXU1tK+38+yV7BG8l9SKyui+kz15h7sVABEBAAH+BwM
T1b1/DPVoc2Yiq5WJw3CUQ+j/z6dpQ7IS/JQgBg/JoRMi4T
jzMXM9c/POjfXWVa/5+UskUwTUUZX0jR4tFExrXkWqidymt
NyyDy4geI810lFyeV3NFK00zN45kFX7KXusBWtQ9fj6XDXB
```


https://twitter.com/thesl3ep/status/876066176589336576