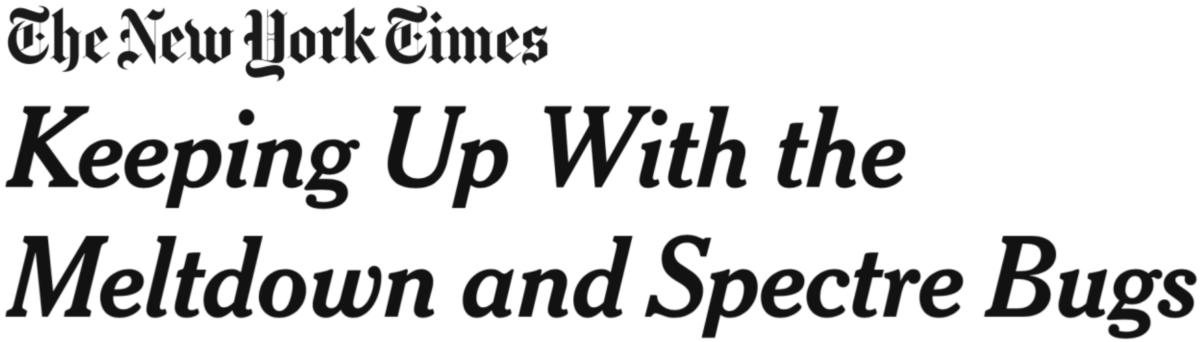
Reliable Security Guarantees

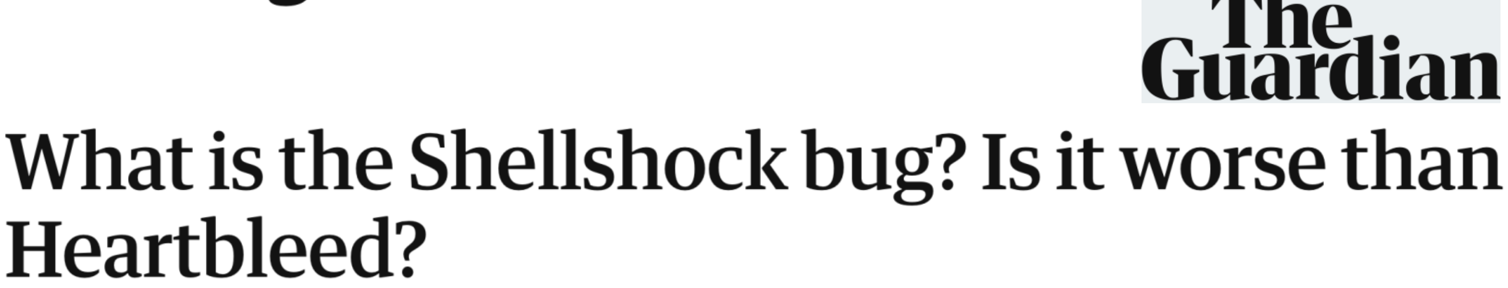
Swen Jacobs | CISPA SeCon | April 3rd, 2019

The Security Arms Race



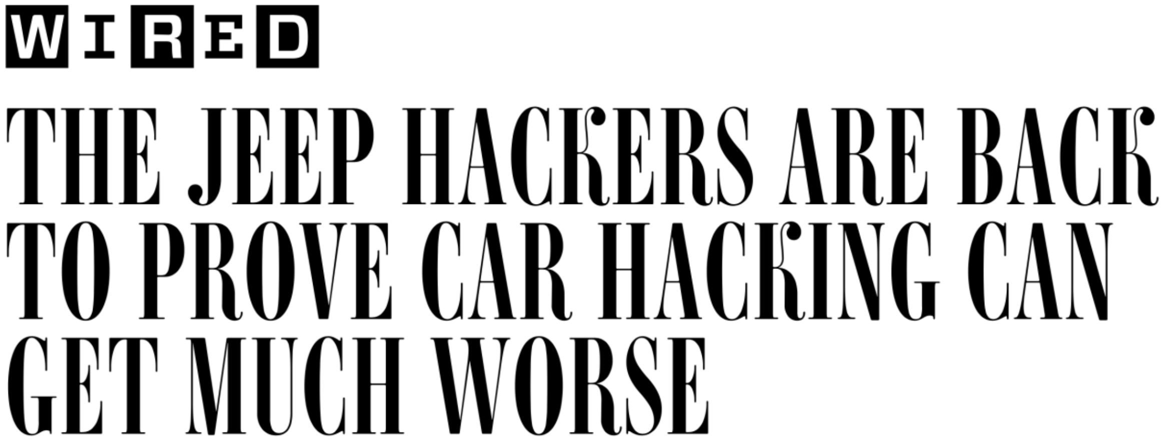
The New York Times

Keeping Up With the Meltdown and Spectre Bugs



The Guardian

What is the Shellshock bug? Is it worse than Heartbleed?



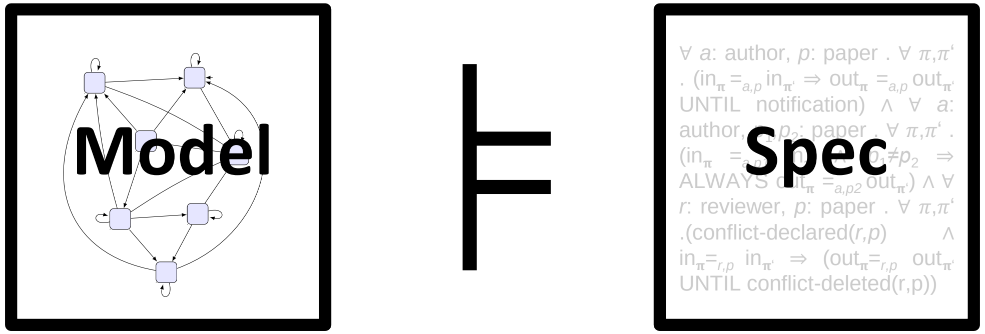
WIRED

THE JEEP HACKERS ARE BACK TO PROVE CAR HACKING CAN GET MUCH WORSE

Formal Methods: Basic Idea

First principles that allow us to obtain reliable security guarantees:

Formal models, formal specifications, mathematical proofs



$\forall$ *a*: author, *p*: paper . $\forall \pi,\pi'$. (in$_\pi =_{a,p}$ in$_{\pi'}$ $\Rightarrow$ out$_\pi =_{a,p}$ out$_{\pi'}$ UNTIL notification) $\wedge$ $\forall$ *a*: author, p_1,p_2: paper . $\forall \pi,\pi'$. (in$_\pi =_{a,p1}$ in$_{\pi'}$ $\wedge$ $p_1 \neq p_2$ $\Rightarrow$ ALWAYS out$_\pi =_{a,p2}$ out$_{\pi'}$) $\wedge$ $\forall$ *r*: reviewer, *p*: paper . $\forall \pi,\pi'$.(conflict-declared(*r*,*p*) $\wedge$ in$_\pi =_{r,p}$ in$_{\pi'}$ $\Rightarrow$ (out$_\pi =_{r,p}$ out$_{\pi'}$ UNTIL conflict-deleted(r,p))

Break out of the arms race:

Rule out **entire classes of attacks**

Formal Methods: Manual or Automatic?

Yesterday (and later today): proving security of a protocol by hand

- more flexible, possibly more precise
- human effort is not easily reusable
- makes sense for a protocol standard, but usually not for a concrete implementation

Now: automated methods for proving correctness

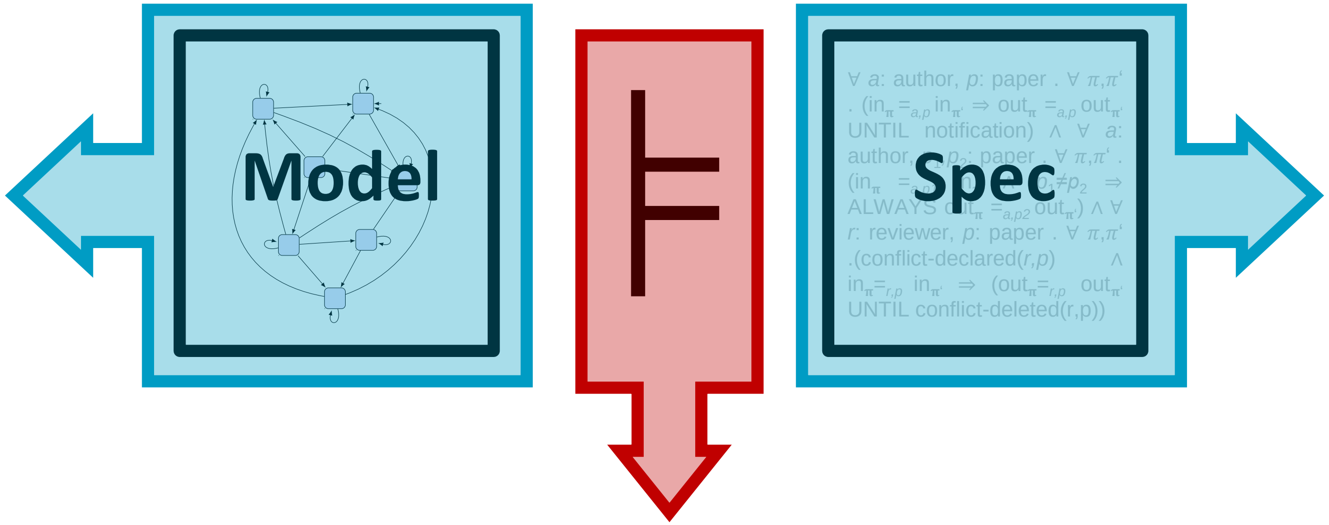
- difficult to obtain full automation, often sacrifices precision
- human work is done up front, effort for verifying a given implementation is negligible
- needed if we want to verify large parts of our implementations

Formal Methods: Design Choices

how much detail do we need?

which features of the system can we safely abstract from?

does our model allow efficient automated reasoning?


Model
⊨
Spec
∀ a: author, p: paper . ∀ π,π' . ($in_\pi =_{a,p} in_{\pi'}$ ⇒ $out_\pi =_{a,p} out_{\pi'}$ UNTIL notification) ∧ ∀ a: author, p_1,p_2: paper . ∀ π,π' . ($in_\pi =_{a,p1} in_{\pi'}$ ∧ $p_1 \neq p_2$ ⇒ ALWAYS $out_\pi =_{a,p2} out_{\pi'}$) ∧ ∀ r: reviewer, p: paper . ∀ π,π' .(conflict-declared(r,p) ∧ $in_\pi =_{r,p} in_{\pi'}$ ⇒ ($out_\pi =_{r,p} out_{\pi'}$ UNTIL conflict-deleted(r,p))

what kind of properties do we want/need to express?

how to express them s.t. their verification is possible/easy?

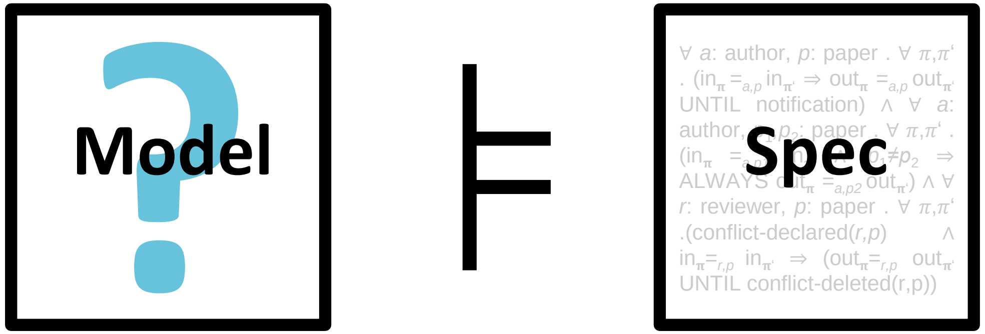
for given choice of systems and specifications:

- how can we efficiently reason about them?
- can it be completely automated?
- if so, can we guarantee termination?

Formal Methods: Advanced Techniques

More challenging approach: automatic synthesis of systems that are correct by construction

Find a model that satisfies a given specification:


Model
?
⊨
Spec
$\forall$ *a*: author, *p*: paper . $\forall \pi,\pi'$. (in$_\pi =_{a,p}$ in$_{\pi'}$ $\Rightarrow$ out$_\pi =_{a,p}$ out$_{\pi'}$ UNTIL notification) $\wedge$ $\forall$ *a*: author, p_1,p_2: paper . $\forall \pi,\pi'$. (in$_\pi =_{a,p1}$ in$_{\pi'}$ $\wedge$ $p_1 \neq p_2$ $\Rightarrow$ ALWAYS out$_\pi =_{a,p2}$ out$_{\pi'}$) $\wedge$ $\forall$ *r*: reviewer, *p*: paper . $\forall \pi,\pi'$.(conflict-declared(*r,p*) $\wedge$ in$_\pi =_{r,p}$ in$_{\pi'}$ $\Rightarrow$ (out$_\pi =_{r,p}$ out$_{\pi'}$ UNTIL conflict-deleted(r,p))

For this lecture, just assume that synthesis techniques are generalizations of verification techniques

Overview

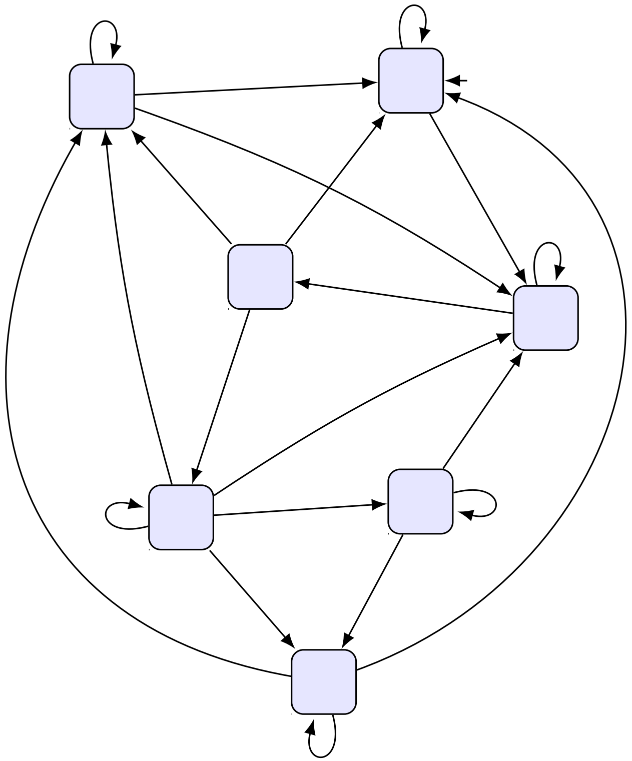
- Formal Verification Basics
- Security Protocol Verification
- Parameterized Systems
- Reliability and Fault Tolerance
- Information Flow and Hyperproperties

Formal Verification Basics

Formal System Models

Main ingredients:

- state space (may be infinite)
- transition relation (may be non-deterministic)
- possibly: observations, i.e., state labels



Symbolic representations:

- represent large (or infinite) state spaces more efficiently
- Example:
  - valuations of a set of Boolean variables $x_1, \dots, x_n$ yield a state space of size 2^n
  - transition relation and properties can be defined over sets of states, defined by Boolean formulas $x_1 \wedge (x_2 \vee x_3 \vee x_4)$

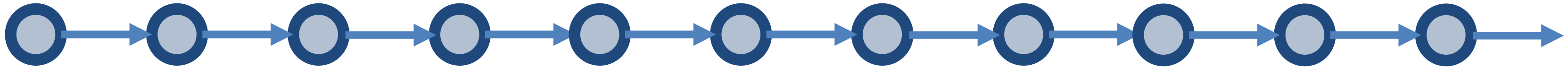
Formal Specifications

Safety properties: “something bad should never happen”

⇨ need only define states (of the model) that are errors/insecure/undesirable

Liveness properties: “if the system runs sufficiently long, something good will happen”

⇨ define what should happen under which conditions

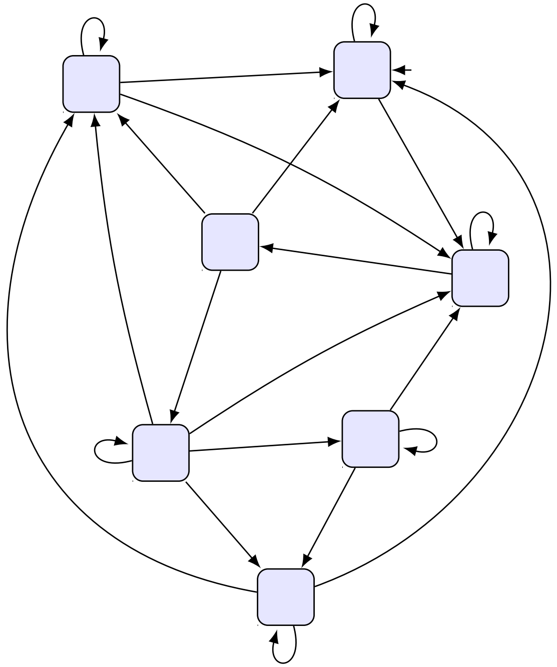


More complex properties possible

Expressed in dedicated specification languages

Verification Algorithms

Simple case: explicit state traversal



Symbolic safety model checking:

- start with symbolic representation of error states
- repeatedly "apply" (symbolic representation of) transition relation
- after each application, check if fixpoint or error states are reached
- efficient implementation e.g. with BDDs

⇨ allows verification of systems with "10^{20} states and beyond" [Burch et al. 1992]

Verification Algorithms (II)

Problem: state space and system runs may be infinite/unbounded

⇨ verification needs to handle infinity in a finite way

Handling infinite runs:

- automata theory (e.g. Büchi automata)

Handling infinite state space:

- abstraction (e.g. equivalence relations, symmetry reduction)
- induction (hard to automate)

Questions?

Security Protocol Verification

Security Protocols: Examples

SSL/TLS (e-commerce, web search, etc.)

WEP, WPA in secure WLAN

authentication at ATM

Pictures: designed by Freepik

What is a Security Protocol?

Security protocols are

distributed programs ⇒ hard to analyze

that use cryptography ⇒ adds complexity

to achieve security properties ⇒ difficult to test/verify

in the presence of a malicious adversary. ⇒ makes everything much more difficult

Formal Models for Security Protocols

Dolev-Yao model [Dolev/Yao 1983]:

- perfect crypto assumption
- attacker controls network and has unbounded computational power
- easy to reason about, automation possible

Computational model:

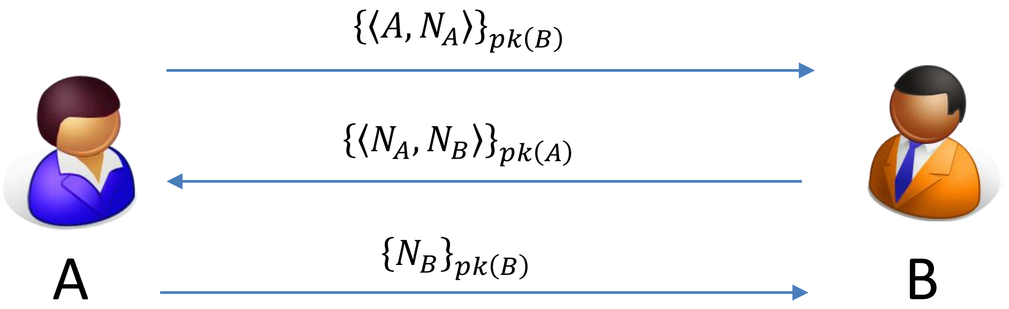
- crypto can be broken
- computational power of attacker is bounded
- hard to reason about, automation difficult

Computational soundness [Abadi/Rogaway 2002]:
under which conditions does a proof in the Dolev-Yao model
imply correctness in the computational model?

Example: Needham-Schroeder Protocol [Needham/Schroeder 1978]

Goal: use *public key encryption* to

1. exchange a confidential message
2. authenticate participants


$\{\langle A, N_A \rangle\}_{pk(B)}$
$\{\langle N_A, N_B \rangle\}_{pk(A)}$
$\{N_B\}_{pk(B)}$
A
B

Semi-formal description:

1. $A \rightarrow B$: $\{\langle A, N_A \rangle\}_{pk(B)}$
2. $B \rightarrow A$: $\{\langle N_A, N_B \rangle\}_{pk(A)}$
3. $A \rightarrow B$: $\{N_B\}_{pk(B)}$

Why did we say “semi-formal”?

Semi-formal description:

1. $A \rightarrow B$: $\{\langle A, N_A\rangle\}_{pk(B)}$
2. $B \rightarrow A$: $\{\langle N_A, N_B\rangle\}_{pk(A)}$
3. $A \rightarrow B$: $\{N_B\}_{pk(B)}$

Are there restrictions on the messages?

yes, these are “nonces” (numbers used once), i.e., need to be fresh

This is one possible execution – the intended one. Can other things happen if we arrange these message exchanges differently? Which restrictions are enforced on such message exchanges?

A more formal description

$$
\begin{array}{lcr}
A \xrightarrow{\{\langle A,N_A\rangle\}_{pk(B)}} & \qquad & \xrightarrow{\{\langle x,y\rangle\}_{pk(B)}} B \\
A \xleftarrow{\{\langle N_A,z\rangle\}_{pk(A)}} & \qquad & \xleftarrow{\{\langle y,N_B\rangle\}_{pk(x)}} B \\
A \xrightarrow{\{z\}_{pk(B)}} & \qquad & \xrightarrow{\{N_B\}_{pk(B)}} B
\end{array}
$$

Now, x, y, z are variables – they are not known before, and the participants will react to any value they receive.

Such formalizations are useful when thinking about *what could go wrong?*

…17 years later [Lowe 1995]

Lowe’s man-in-the-middle attack:

$$A \xrightarrow{\{\langle A, N_A \rangle\}_{pk(C)}} C \xrightarrow{\{\langle A, N_A \rangle\}_{pk(B)}} B$$

$$A \xleftarrow{\{\langle N_A, N_B \rangle\}_{pk(A)}} C \xleftarrow{\{\langle N_A, N_B \rangle\}_{pk(A)}} B$$

$$A \xrightarrow{\{N_B\}_{pk(C)}} C \xrightarrow{\{N_B\}_{pk(B)}} B$$

Violates authentication: B thinks it communicates with A, but the messages are coming from C

Violates secrecy: C finds out N_B

Easily fixed:
include identity also in response message (from B)

Hard part:
detect such vulnerabilities

Formalizing Protocols: Dolev-Yao Model

Cryptographic primitives and other operations are modeled as function symbols. Messages are terms composed from functions and basic values (nonces, identities) according to fixed set of rules.

$$\{\langle A, N_A\rangle\}_{pk(B)}$$

Attacker model defines which messages attacker can generate, and which information attacker can obtain from messages sent over the network.

$$\frac{x \quad y}{\{\langle x, y\rangle\}_{pk(B)}} \qquad \frac{\{x\}_{pk(B)} \quad sk(B)}{x}$$

State of system = set of messages attacker knows or can generate

Transition = step of protocol or derivation of attacker

Formal Specifications in the Dolev-Yao Model

Secrecy violation = attacker can derive value that should be secret, i.e.,

can we reach a state (set of messages) that includes the secret?

Authentication violation = participant successfully finishes protocol & draws incorrect conclusions

The Intruder Deduction Problem

Definition: For a given set of inference rules I, the intruder deduction problem is to decide whether t *can be derived from* S for arbitrary (finite) set of terms S and term t.

In general, the intruder deduction problem is undecidable [Abadi/Cortier 2006], i.e., no general algorithm that solves all instances of the problem exists.

It is however decidable for certain classes of inference systems. In particular, it is decidable in PTIME for the Dolev-Yao inference system.

⇨ verification can be efficiently automated

Verification Algorithms for Security Protocols

SATMC (used in AVISPA tool) [Armando/Compagna 2008]:

- encodes problem into SAT for efficient solving
- only for fixed bounds on # sessions and # derivation steps

ProVerif [Blanchet 2001]:

- symbolic model of protocol as Horn clauses
- models unbounded number of sessions but over-approximates behavior

Tamarin [Meier et al. 2013]:

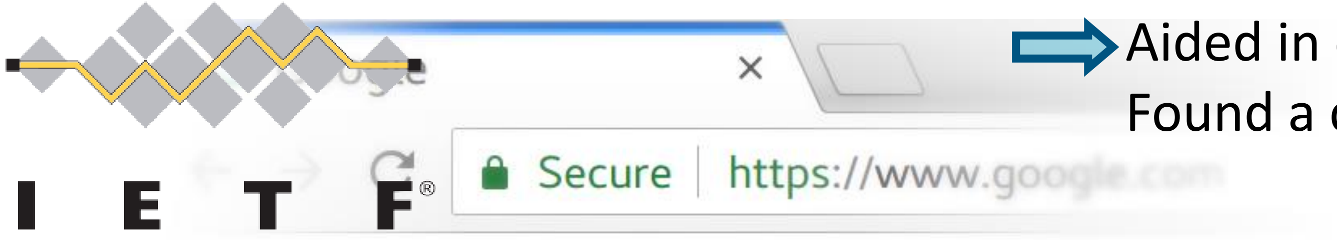
- very flexible modeling and reasoning
- complex problems can be solved in interactive mode

Formal Verification of Security Protocols: Success Stories

ISO/IEC 9798 and **11770**

⇨ Found several flaws [Cremers/Horvat 2016]
Standards were updated

IETF **TLS 1.3**



⇨ Aided in developing new standard
Found a critical attack [Cremers et al. 2017]

5G-AKA

⇨ Found under-specified assumptions
that enabled an attack [Cremers/Dehnel-Wild 2019]

QUESTIONS?

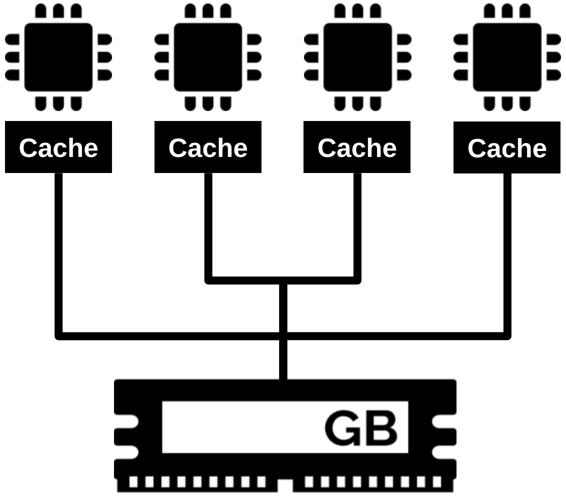
Restrictions of Verification Algorithms

To make verification terminate, usually only a bounded number of sessions are verified

However, without a formal argument why every attack is possible with a given bound, this means the verification algorithms are incomplete

VERIFICATION OF PARAMETERIZED SYSTEMS

Parameterized Verification Problems



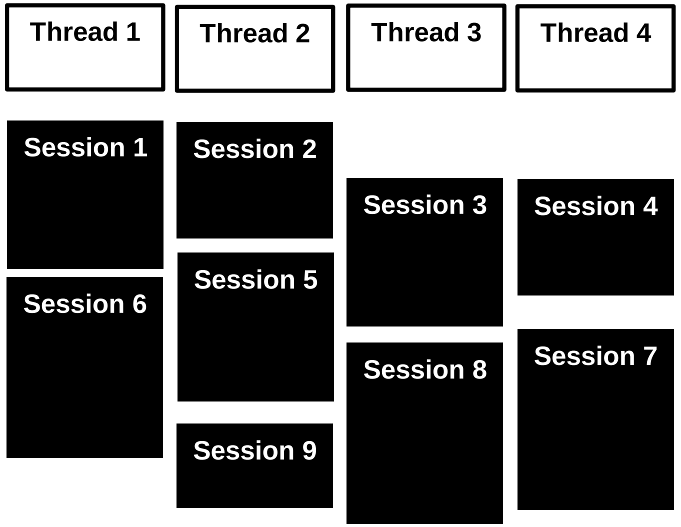
Cache

Cache

Cache

Cache

GB



Thread 1 | Thread 2 | Thread 3 | Thread 4

Session 1

Session 2

Session 3

Session 4

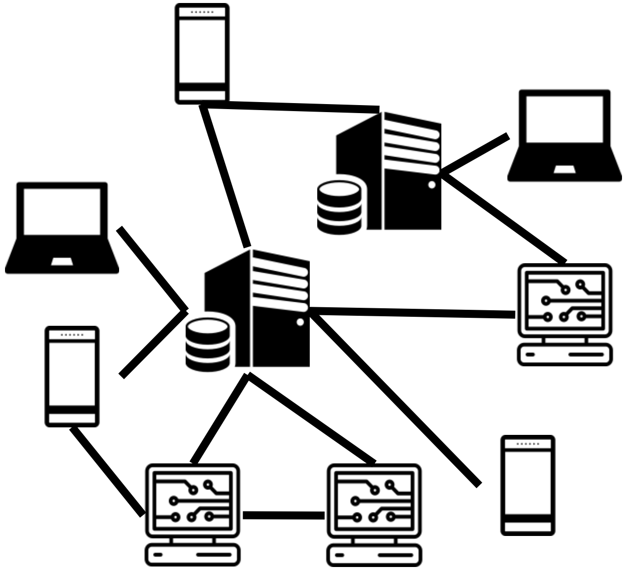
Session 5

Session 6

Session 7

Session 8

Session 9

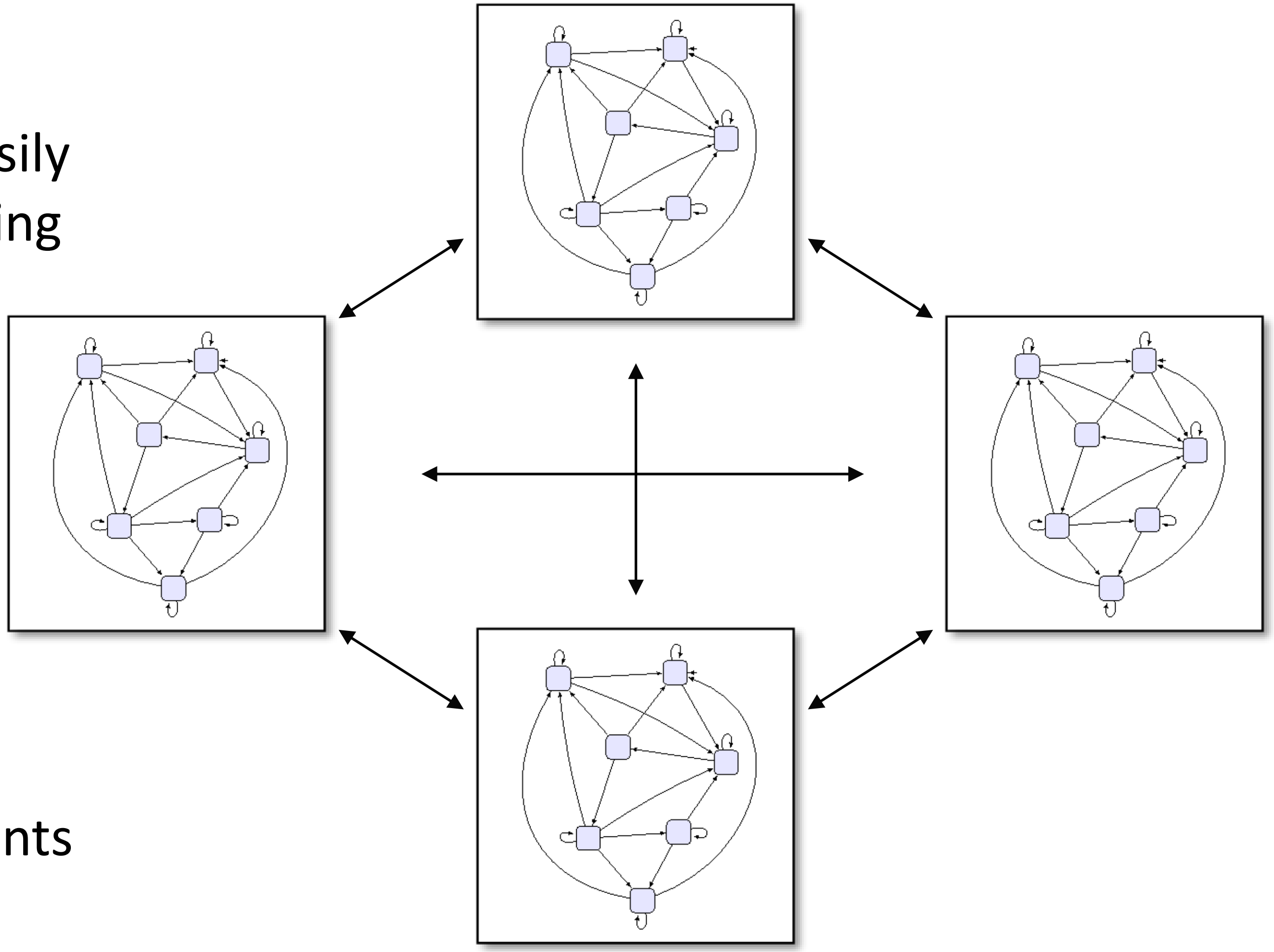


How to obtain reliable correctness guarantees that hold **for any value of the parameter**?

Pictures: designed by Freepik

Verification of Concurrent Finite-State Systems

Consider fixed number of components, each finite state, with some means of communication between them:



In theory, problem is decidable, following easily from decidability of finite-state model checking

In practice, problem is more difficult than might be expected: state space explodes since we need to consider all combinations of local states

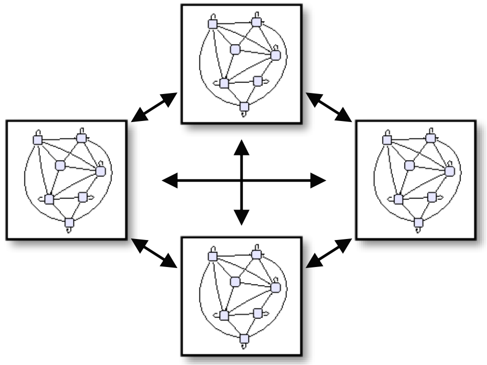
(in Example, $7^4 = 2401$ states)

⇨ state space exponential in # components

Parameterized Verification by Abstraction

Assume: all components (sessions, processes, cores, …) of the system are
uniform in their internals and in their connections

Then: if multiple components are in same state and one makes a transition,
it does not matter which of them does it



⇨ abstract system state by vector that counts how many processes are in each (local) state

Comp. 1	Comp. 2	Comp. 3	Comp. 4	Comp. 5	Comp. 6
a	b	b	c	a	b

⇨

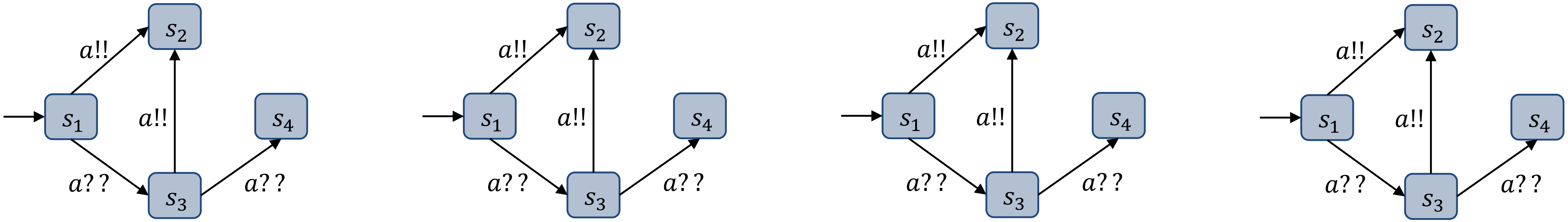
2	3	1
a	b	c

Allows safety model checking
regardless of number of components
under certain assumptions on transition relation

Drawback:
System must be symmetric
and components uniform,
cannot model e.g.
components with unique IDs

Parameterized Verification by Abstraction

Example: Broadcast Protocols [Esparza et al. 1999]



$$\begin{pmatrix}4\\0\\0\\0\end{pmatrix} \xrightarrow{a} \begin{pmatrix}0\\1\\3\\0\end{pmatrix} \xrightarrow{a} \begin{pmatrix}0\\2\\0\\2\end{pmatrix}$$

Computation steps are independent of number of processes; this enables parameterized verification

Broadcast sender in s_1: $\begin{pmatrix}4\\0\\0\\0\end{pmatrix} - \begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \begin{pmatrix}3\\0\\0\\0\end{pmatrix}$

Broadcast receivers move from s_1 to s_3 or from s_3 to s_4:

$$\begin{pmatrix}3\\0\\0\\0\end{pmatrix}\begin{pmatrix}0&0&0&0\\0&1&0&0\\1&0&0&0\\0&0&1&0\end{pmatrix} = \begin{pmatrix}0\\0\\3\\0\end{pmatrix}$$

Broadcast sender moves to s_2: $\begin{pmatrix}0\\0\\3\\0\end{pmatrix} + \begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0\\1\\3\\0\end{pmatrix}$

Parameterized Verification: Intuition & Facts

Widespread intuition about correctness:

> "if an error is possible, then it already appears with a small number of components"

This is true in many cases, and also an error/attack that needs a large number of components may be unlikely to happen in practice (think security protocols).

However, it is not true in general. For many classes of systems, we can find errors that need a number of components that cannot be bounded by any constant.

Parameterized Verification (& Synthesis) by Cutoffs

Notion of cutoffs formalizes this intuition:

Let M be the model of one component, M^n a system composed of n copies of M, and φ the specification of a property.

A natural number c is a cutoff if

$$\forall n \geq c: (M^n \vDash \varphi \iff M^c \vDash \varphi)$$

That is, if the cutoff system is correct, then all larger systems are also correct.

⇨ if we have cutoffs, parameterized verification is easy

under some additional conditions, cutoffs also allow synthesis of systems that are correct by construction regardless of # components

How to Obtain a Cutoff

Cutoff proofs by hand [Emerson/Namjoshi 1995]:

- prove a simulation property, depending on the class of specifications under consideration
- usually requires restrictions on system (components, communication primitives, topology) and on specification
- cutoff may depend on properties of components (e.g., local state space) and specification

Automatic cutoff detection [Kaiser et al. 2010]:

- based on sufficient condition for cutoffs that can be checked statically
- approximative: existing cutoff may not be detected
- termination not guaranteed

Cutoffs for Security Protocol Verification

In symbolic model (Dolev-Yao or similar), a cutoff for the number of agents has been shown [Heather/Schneider 2000,Comon-Lundh/Cortier 2003]:

in most cases, it is sufficient to consider two agents (one honest, one dishonest)

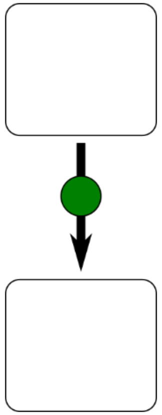
Note: in general, number of sessions is still unbounded.

Bounding number of sessions is also possible in some cases. The cutoff for sessions is then exponential in the number of roles in the protocol (which is usually 2 or 3).

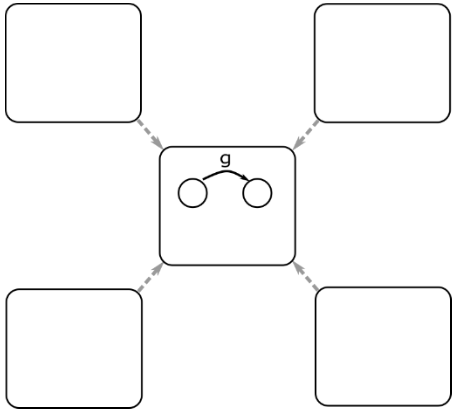
Cutoffs for Verification of Distributed Systems

Usually for classes of systems with different communication primitives:

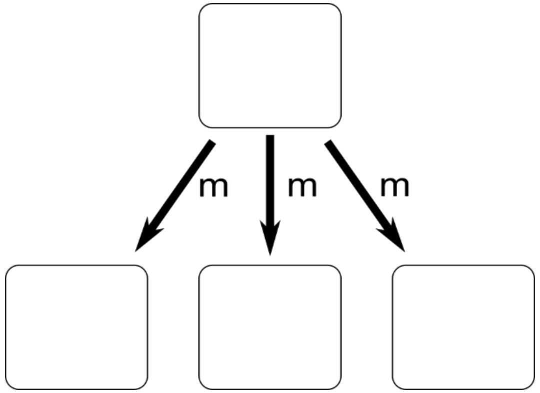
token-passing



guarded transitions



broadcasts



usually constant cutoffs

cutoffs linear in local state space

cutoffs exponential in local state space

smaller cutoffs are possible with additional restrictions (on properties or systems)

Formal Verification (& Synthesis) of Parameterized Systems: Success Stories

Verification of cache coherence protocols [Emerson/Kahlon 2003]

constant cutoffs for a custom-made model (with broadcast and guarded transitions)
that can express all standard textbook cache coherence protocols (only safety properties)

Verification of reference monitors/hypervisors [Franklin et al. 2010]

cutoffs exist for a model where rows of page table are components, and communication is by broadcast (only safety properties)

Synthesis of AMBA bus controller [Bloem et al. 2014]

cutoffs for token-passing systems can be used to synthesize correct implementation of bus controller; long-standing benchmark problem for formal synthesis (safety and liveness properties)

QUESTIONS?

Reliability and Fault Tolerance

Motivation: Faults and Attackers

To be realistic, formal models need to allow behavior that is not intended and may only happen sporadically, or if provoked by an attacker.

In secure messaging, this may be:

- loss of messages (due to network error or interception by attacker)
- loss of state (hardware fault or attacker can corrupt memory)
- loss of phone

In general:

- hardware parts may sporadically fail (and **will** eventually fail in big system)
- attackers may control parts of the system

Different Types of Faults/Attacks

Faults/attacks can be classified along different dimensions

- temporary or permanent?
- crash or arbitrary behavior ("Byzantine fault")?
- local or global?

Interesting cases:

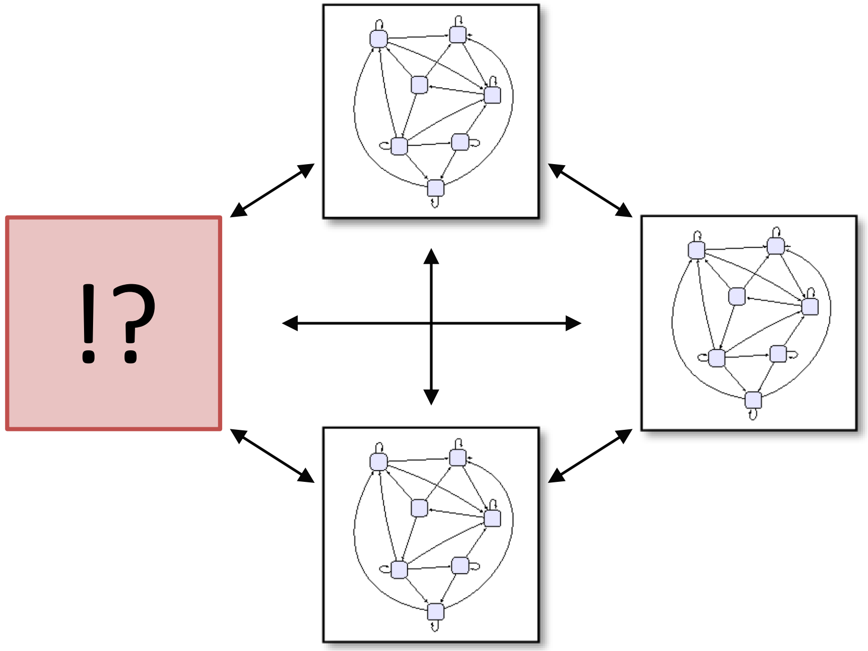
- temporary, arbitrary behavior, global ("self-stabilization") ⇒ memory corruption that can bring system into an arbitrary state
- permanent, arbitrary behavior, local ("Byzantine fault-tolerance") ⇒ attacker controls (permanently) one of the sensors in a car
- permanent, crash, local

Formal Verification in the Presence of Faults

Faults and attackers can either be modeled explicitly for the given context,
(see symbolic model of security protocols)

or based on general principles like self-stabilization and Byzantine fault-tolerance.

General-purpose verification algorithms can be used to express the latter in model and/or specification,
e.g., replace one component by one that behaves arbitrarily



To be useful in practice they need to

- be specialized to be efficient
- give parameterized correctness guarantees

Formal Verification of Fault-tolerant Systems: Success Stories

Synthesis of self-stabilizing algorithms [Mirzaie et al. 2018]

self-stabilization in ring structure, communication by guarded updates;
synthesized algorithms for standard problems like maximal matching or three coloring

Verification of threshold-based algorithms [John et al. 2013, Konnov et al. 2017]

Byzantine fault resistance for components with pairwise communication and guarded updates based on thresholds; verified Byzantine consensus and atomic commit algorithms

Synthesis of synchronous counters [Bloem et al. 2016]

supports self-stabilization and Byzantine faults;
synthesized solution for synchronous counting, providing solutions with smaller stabilization time and fewer states than any known solution

QUESTIONS?

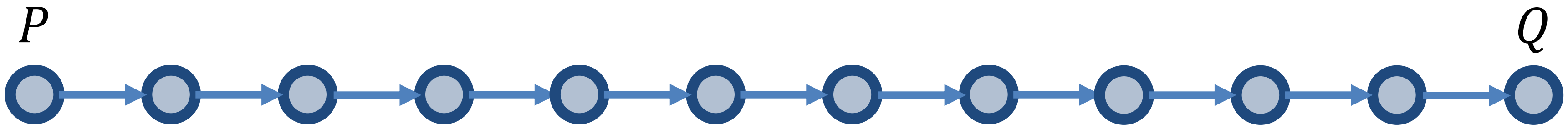
Information Flow and Hyperproperties

Until Now: Verification of Trace Properties

All properties considered until now can be verified on a single execution trace:

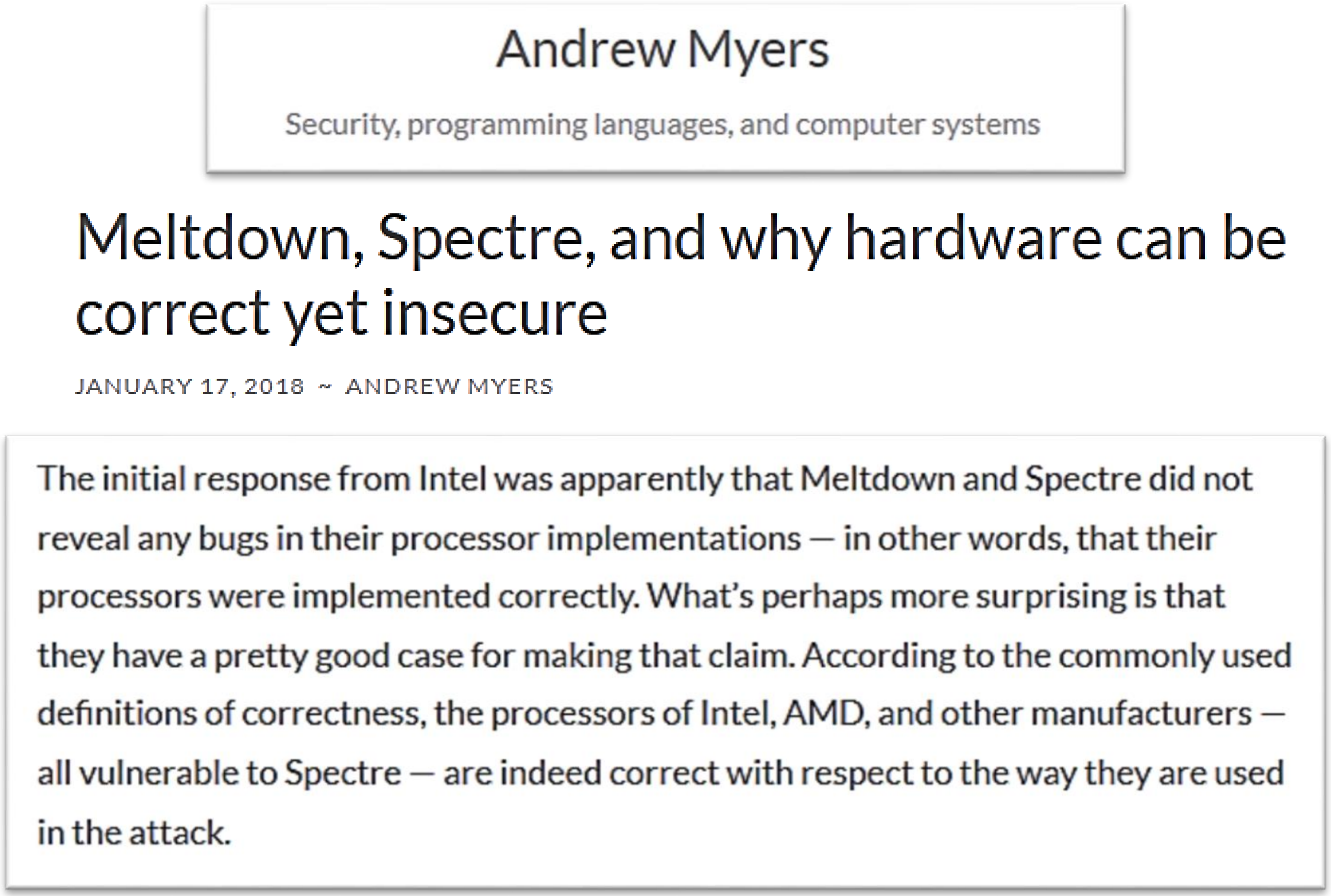
- safety properties (bad things will never happen)
- liveness properties (good things will eventually happen)

"if inputs satisfy property P, then outputs must satisfy property Q"



This type of verification is supported by efficient and mature tools, and is now used in industrial hardware and software design. (Intel, ARM, Microsoft, Google, Amazon, Facebook)

Security Verification Needs to Go Beyond Trace Properties



Andrew Myers

Security, programming languages, and computer systems

Meltdown, Spectre, and why hardware can be correct yet insecure

JANUARY 17, 2018 ~ ANDREW MYERS

The initial response from Intel was apparently that Meltdown and Spectre did not reveal any bugs in their processor implementations — in other words, that their processors were implemented correctly. What's perhaps more surprising is that they have a pretty good case for making that claim. According to the commonly used definitions of correctness, the processors of Intel, AMD, and other manufacturers — all vulnerable to Spectre — are indeed correct with respect to the way they are used in the attack.

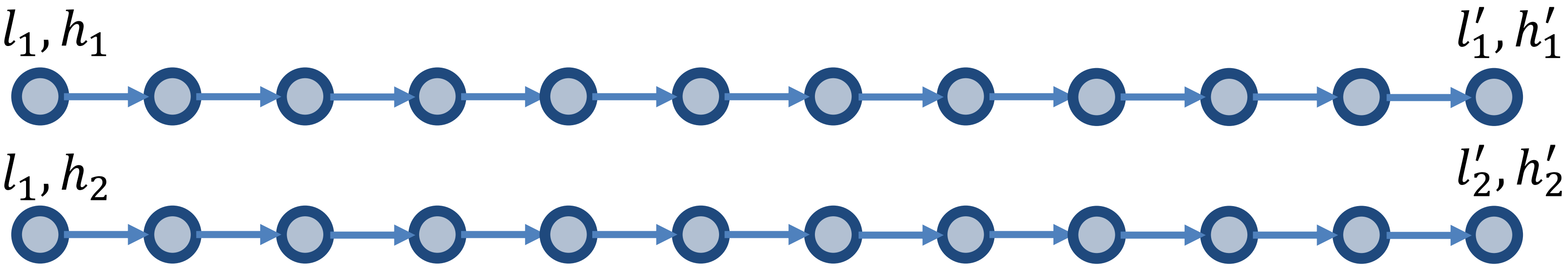
Standard notion of correctness:
"if inputs satisfy property P,
then outputs must satisfy property Q"

Meltdown/Spectre:
compare multiple executions
of a program function and
derive information from differences
between different executions

Non-interference as a Hyperproperty

Non-interference:

“if low-security inputs of two executions are equal,
then (regardless of high-security inputs) the low-security outputs must be equal”



$\boldsymbol{l'_1 = l'_2}$?

Hyperproperties are properties that compare two or more executions of a system.

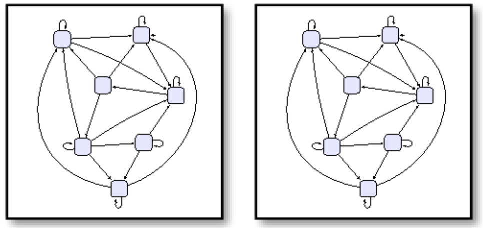
Verification of Information-flow Properties

Information-flow Model Checking, basically:

Model remains the same, but specification talks about multiple executions

⇨ need to devise new algorithms for verification
that can keep track of multiple executions in parallel

Alternative approach: self-composition [Barthe et al. 2004]



Consider a modified model that runs two copies of the system in parallel

Then non-interference can be expressed as
a trace property of this self-composed system

⇨ in principle, existing verification algorithms can be used

Information-flow Properties: Relaxing Non-interference

In practice, strict non-interference is often too strong a requirement

Leakage of some information may be acceptable, or we may assume that an attacker cannot observe the difference between certain possible outputs.

E.g., if output is a list, then only require the set of elements to be equal, and ignore their order.

Verification by self-composition allows to define an indistinguishability criterion I, and instead of $l_1' = l_2'$ we can require $I(l_1', l_2')$ with minor modification of the verification algorithm.

Information-flow Properties: Relaxing Non-interference (II)

In practice, strict non-interference is often too strong a requirement

Similarly, downgrading of information can be supported [Terauchi/Aiken 2005],
such that the low outputs can depend on certain properties of the high inputs.

E.g., if high inputs are lists, then low outputs must only be equal if these lists are of same length.

This is essential to prove information-flow properties of login procedures,
since they will always leak at least the information whether the password was correct or not
(but should not leak additional information).

Efficient Information-flow Model Checking

Observation:
model-checking the self-composed system is costly, and full duplication is often not necessary

Lazy self-composition [Yang et al. 2018]:

- work on a single copy of the system at first
- try to detect if information leak is possible based on symbolic taint analysis (an over-approximation of actual information leaks)
- if taint analysis reports a possible information leak, use self-composition locally, i.e., only for the relevant part of the state space

Quantification of Information Leaks [Backes et al. 2009]

Instead of just checking if there exist information leaks, we can try to find out how much information can be leaked.

This can be represented by an equivalence relation over the high inputs, i.e., a partition of the high inputs s.t. the attacker can distinguish whether two high inputs are from different sets.

It can be computed as follows:

1. start with partition R that consists of a single set
2. use self-composition with downgrading based on R to check for an information leak
3. if there is one, use it to refine R and go back to 2

The resulting R can then be used to compute common measures of information leaks, such as Shannon entropy or guessing entropy.

Adding Side Channels

The self-composition approach can be further generalized to check for side-channel attacks, e.g., timing side channels [Almeida et al. 2016].

To model side channels, add a leakage model that maps each state s of the program to an observation $L(s)$ of the attacker.

A leakage model for timing side channels in software will usually reveal which branch is taken in an if-statement or at the beginning of a while-loop.

Depending on its precision, it may also reveal

- memory addresses used in load or store operations (to model cache timing attacks)
- size of instruction operands (e.g., timing of division is sensitive to size of operands)

Then, two traces are indistinguishable if the have the same sequence of observations.

Formal Verification of Information Flow: Success Stories

Checking crypto libraries for side-channel attacks [Almeida et al. 2016]

Tool ct-verif verified libraries NaCl, parts of OpenSSL, and two elliptic curve arithmetic implementations.

Improved efficiency by lazy self-composition [Yang et al. 2018]

Not used on large-scale benchmarks, but shows orders of magnitude better performance than eager self-composition on many examples

Questions?

Overview

- Formal Verification Basics
- Security Protocol Verification
- Parameterized Systems
- Reliability and Fault Tolerance
- Information Flow and Hyperproperties

References (in order of appearance)

[Burch et al. 1992] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, L. J. Hwang: Symbolic Model Checking: 10^{20} States and Beyond. Inf. Comput. 98(2)

[Dolev/Yao 1983] Danny Dolev, Andrew Chi-Chih Yao: On the security of public key protocols. IEEE Trans. Information Theory 29(2)

[Abadi/Rogaway 2002] Martín Abadi, Phillip Rogaway: Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). J. Cryptology 15(2)

[Needham/Schroeder 1978] Roger M. Needham, Michael D. Schroeder: Using Encryption for Authentication in Large Networks of Computers. Commun. ACM 21(12)

[Lowe 1995] Gavin Lowe: An Attack on the Needham-Schroeder Public-Key Authentication Protocol. Inf. Process. Lett. 56(3)

[Abadi/Cortier 2006] Martín Abadi, Véronique Cortier: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. 367(1-2)

[Armando/Compagna 2008] Alessandro Armando, Luca Compagna: SAT-based model-checking for security protocols analysis. Int. J. Inf. Sec. 7(1)

[Blanchet 2001] Bruno Blanchet: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. CSFW 2001

[Meier et al. 2013] Simon Meier, Benedikt Schmidt, Cas Cremers, David A. Basin: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. CAV 2013

[Cremers/Horvat 2016] Cas Cremers, Marko Horvat: Improving the ISO/IEC 11770 standard for key management techniques. Int. J. Inf. Sec. 15(6)

[Cremers et al. 2017] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, Thyla van der Merwe: A Comprehensive Symbolic Analysis of TLS 1.3. ACM Conference on Computer and Communications Security 2017

[Cremers/Dehnel-Wild 2019] Cas Cremers, Martin Dehnel-Wild: Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion. NDSS 2019

[Esparza et al. 1999] Javier Esparza, Alain Finkel, Richard Mayr: On the Verification of Broadcast Protocols. LICS 1999

[Emerson/Namjoshi 1995] E. Allen Emerson, Kedar S. Namjoshi: Reasoning about Rings. POPL 1995

[Kaiser et al. 2010] Alexander Kaiser, Daniel Kroening, Thomas Wahl: Dynamic Cutoff Detection in Parameterized Concurrent Programs. CAV 2010

References (in order of appearance)

[Heather/Schneider 2000] James Heather, Steve Schneider: Towards Automatic Verification of Authentication Protocols on an Unbounded Network. CSFW 2000

[Comon-Lundh/Cortier 2003] Hubert Comon-Lundh, Véronique Cortier: Security Properties: Two Agents Are Sufficient. ESOP 2003

[Emerson/Kahlon 2003] E. Allen Emerson, Vineet Kahlon: Exact and Efficient Verification of Parameterized Cache Coherence Protocols. CHARME 2003

[Franklin et al. 2010] Jason Franklin, Sagar Chaki, Anupam Datta, Arvind Seshadri: Scalable Parametric Verification of Secure Systems: How to Verify Reference Monitors without Worrying about Data Structure Size. IEEE Symposium on Security and Privacy 2010

[Bloem et al. 2014] Roderick Bloem, Swen Jacobs, Ayrat Khalimov: Parameterized Synthesis Case Study: AMBA AHB. SYNT 2014

[Mirzaie et al. 2018] Nahal Mirzaie, Fathiyeh Faghih, Swen Jacobs, Borzoo Bonakdarpour: Parameterized Synthesis of Self-Stabilizing Protocols in Symmetric Rings. OPODIS 2018

[John et al. 2013] Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, Josef Widder: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. FMCAD 2013

[Konnov et al. 2017] Igor Konnov, Marijana Lazic, Helmut Veith, Josef Widder: Para^2: parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. Formal Methods in System Design 51(2)

[Bloem et al. 2016] Roderick Bloem, Nicolas Braud-Santoni, Swen Jacobs: Synthesis of Self-Stabilising and Byzantine-Resilient Distributed Systems. CAV 2016

[Barthe et al. 2004] Gilles Barthe, Pedro R. D'Argenio, Tamara Rezk: Secure Information Flow by Self-Composition. CSFW 2004

[Terauchi/Aiken 2005] Tachio Terauchi, Alexander Aiken: Secure Information Flow as a Safety Problem. SAS 2005

[Yang et al. 2018] Weikun Yang, Yakir Vizel, Pramod Subramanyan, Aarti Gupta, Sharad Malik: Lazy Self-composition for Security Verification. CAV 2018

[Backes et al. 2009] Michael Backes, Boris Köpf, Andrey Rybalchenko: Automatic Discovery and Quantification of Information Leaks. IEEE Symposium on Security and Privacy 2009

[Almeida et al. 2016] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Michael Emmi: Verifying Constant-Time Implementations. USENIX Security Symposium 2016