Automated Verification of Security Chains in Software-Defined Networks with Synaptic

Nicolas Schnepf, Rémi Badonnel,
Abdelkader Lahmadi, and Stephan Merz

University of Lorraine, LORIA – Inria Nancy, France

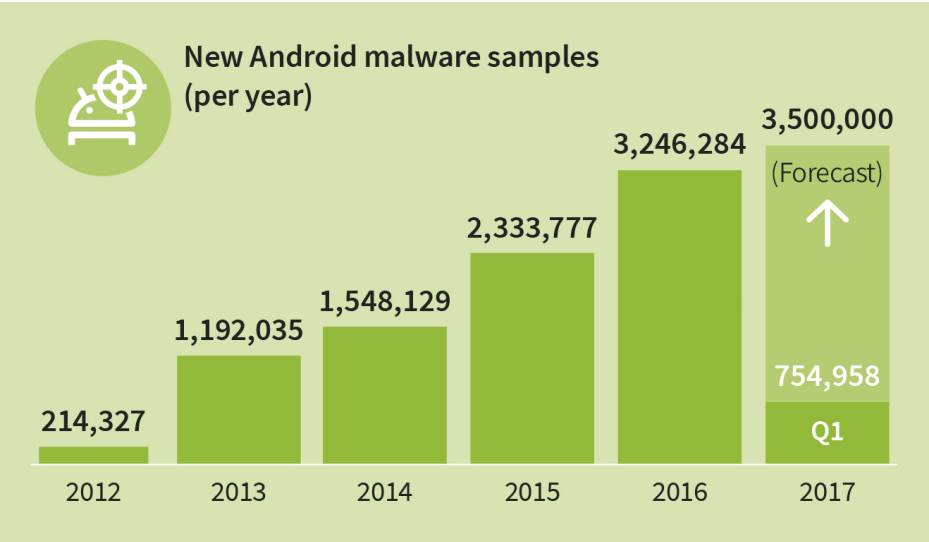
Cloud Computing Days 2017, Nancy

Outline

1. Problem Statement
2. Related Work
3. Automated Verification of Security Chains
4. Prototype and Performance Evaluation
5. Conclusions

Motivations

- Growing number of malicious applications on Android devices
  - 3.25 million malicious applications in 2016 (Gdata Security Labs)



- Limitations of market preventive security methods
- On-device security constrained in terms of resources
- Partial outsourcing through cloud-based security chains
- Can automated verification methods help ensure their quality?

Outline

1. Problem Statement
2. Related Work
3. Automated Verification of Security Chains
4. Prototype and Performance Evaluation
5. Conclusions

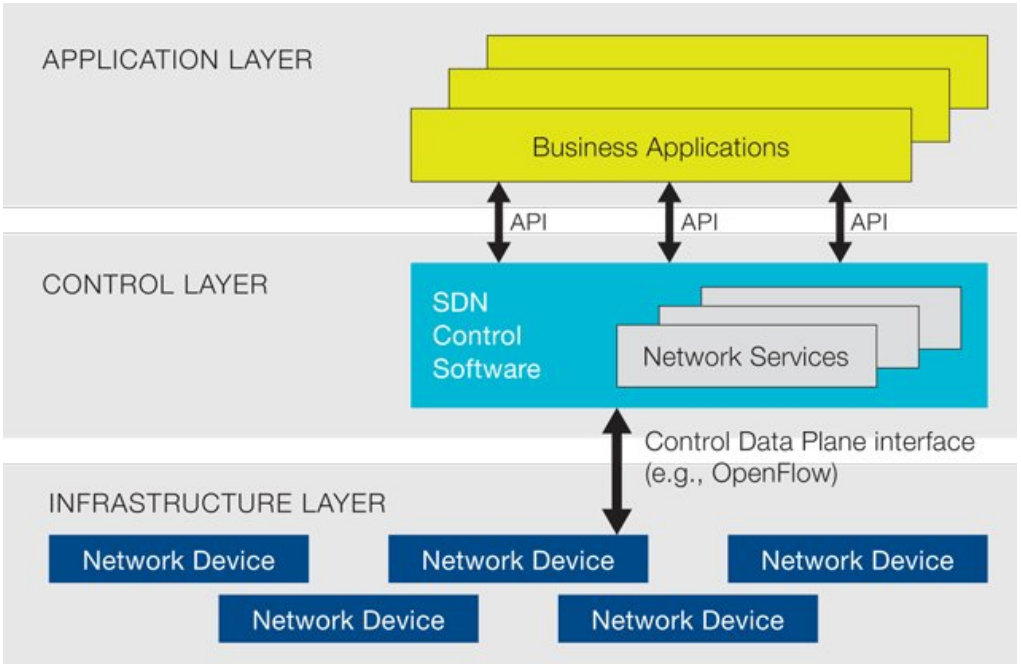
Security middleboxes in cloud infrastructures

- Virtualization and outsourcing of network security functions in cloud infrastructures
- Target: higher flexibility and better manageability

- Uniform middlebox modelling proposed by Joseph and Stoica
- Firewall middlebox specification with *FlowGuard*

- Automating large-scale deployments remains difficult
- Lack of uniform interfaces for these devices
- Formal verification could help validating their correctness

Network programmability as a management enabler



SDN architecture (source: `www.opennetworking.org`)

- Easier automation of middlebox deployment and chaining
- Low level interface needs abstraction and formal methods

Techniques for formal verification

SMT solving

- Representation of systems and properties as logical formulas
- SMTlib: a standard input language for SMT solvers
- Verification based on checking satisfiability/validity

Model checking

- Representation of systems based on finite state automata
- Properties expressed in temporal logics: LTL (Linear Temporal Logic) or CTL (Computation Tree Logic)

Security chains in software-defined networks

Chaining security functions for protecting end users

- Automated building in cloud infrastructures using SDN
- Dynamic and optimized placement of security functions based on available resources

Efforts for formally verifying such security compositions

- Validation of distributed firewall policies: *Al-Shaer et al.*
  - Example: do not simultaneously authorize and block port 5000
- Validation of traffic properties: Vericon *(Ball et al.)*
  - Example: check that a policy blocks incoming connections

The Pyretic language for describing network policies

Pyretic

- Python-embedded domain-specific programming language for writing SDN control programs
- Network data-plane behavior: policy functions
- compiles policies to OpenFlow-based switches

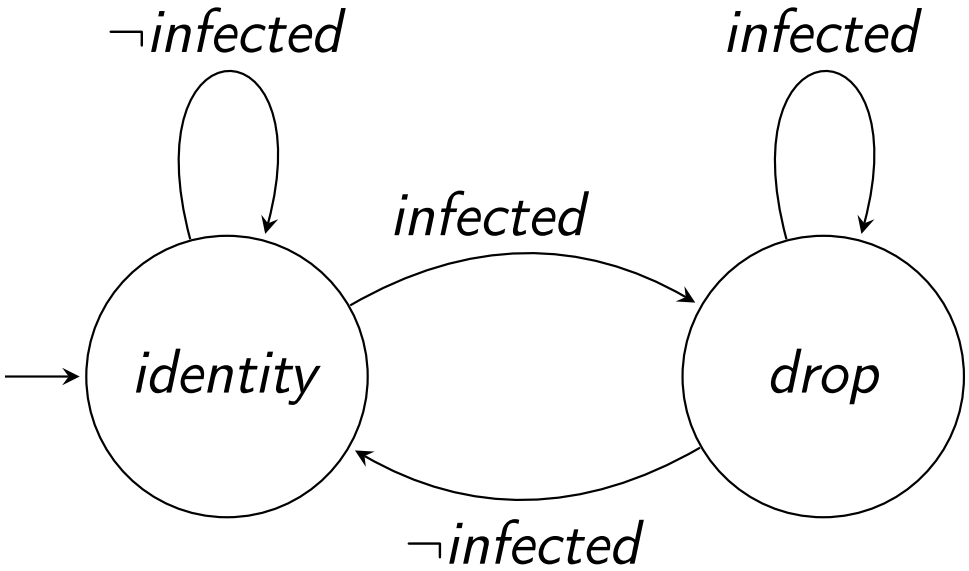
Pyretic rules and compositions

- 4 atomic rules: identity, drop, match and modify
- Sequential composition operator: »
- Parallel composition operator: +

Examples of simple policy chains

- Parallel composition: match(port=5000) + match(port=4000)
- Sequential composition: match(port=5000) » drop

Verifying the control plane of Pyretic chains with Kinetic



- **Kinetic:** represent control dynamics as a Finite State Machine
- Validation by model checking with NuSMV
- Example property: $\mathbf{A}\,\mathbf{G}\,(\mathit{infected} \rightarrow \mathbf{A}\,\mathbf{X}\,\mathit{drop})$
- Objective: extend formal verification to the data plane

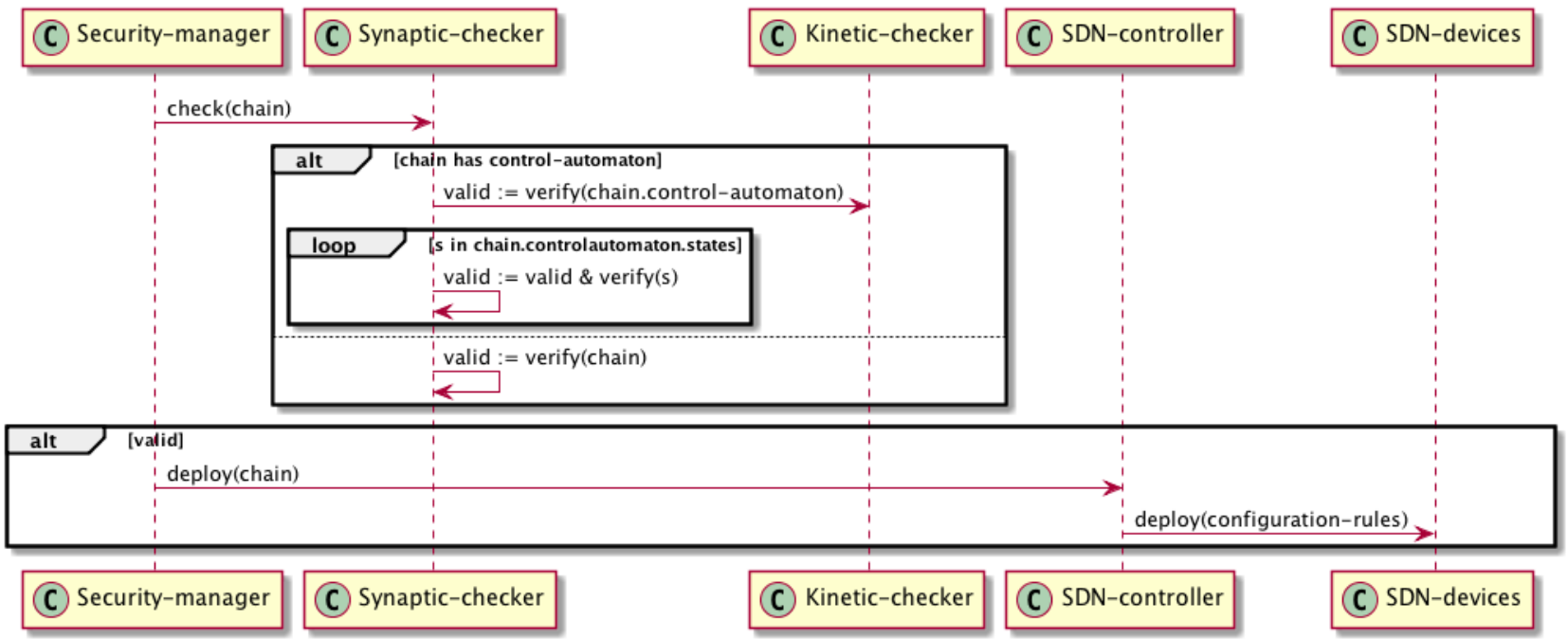
Outline

1. Problem Statement
2. Related Work
3. Automated Verification of Security Chains
4. Prototype and Performance Evaluation
5. Conclusions

Proposed verification strategy

Verification of both control and data planes of security chains

- Extension of the Kinetic language to the data plane
- Verification using SMT solving and model checking
  - Chains translated to both types of tools
  - Addition of properties to constrain the model



Verifying security chains using SMT solving

Modelling and translation

- Chains and constraints expressed with SMTlib
- Representation based on logical formulas
- Prototype supporting the CVC4 and veriT SMT solvers

Advantages of SMT solving for our strategy

- Pyretic expressions correspond to logical formulas
- Easy to add constraints expressed as equalities
- Example: port==5000 $\rightarrow$ flow accepted

Compilation of a chain for SMTlib

- Atomic rules represented using Boolean values and equalities
- These rules represent the basic building blocks

- Composition operators encoded using Boolean connectives
- Used to build up complex security chains

- Traffic invariants expressed as logical conditions
- Success: the chain validates the constraints

Compilation examples

Compilation of a simple chain

- match(port=5000) + match(port=4000)
  $\rightarrow$ (OR (= port 5000) (= port 4000))
- Production of a packet condition

Compilation of a condition

- ip=192.168.0.15 $\rightarrow$ (= ip 1)
- replacement of network elements by integer values

Verification of security chains using model checking

Modelling and translation

- Represent the network as a finite state automaton
- Express the invariants in temporal logic (CTL)
- Success: the automaton satisfies the invariants

Advantages of model checking for our strategy

- Already used for the control plane (Kinetic)
- Possibility of extending this model to the data plane by using nuXmv instead of NuSMV

Compilation of a chain for nuXmv

- Express the network as a finite state automaton
- Variables represent packet fields

- Strictly sequential policies correspond to automaton transitions
- Join points give rise to states of the automaton

- Traffic invariants expressed as CTL properties
- Authorized traffic accepted by the automaton
- Forbidden packets are blocked

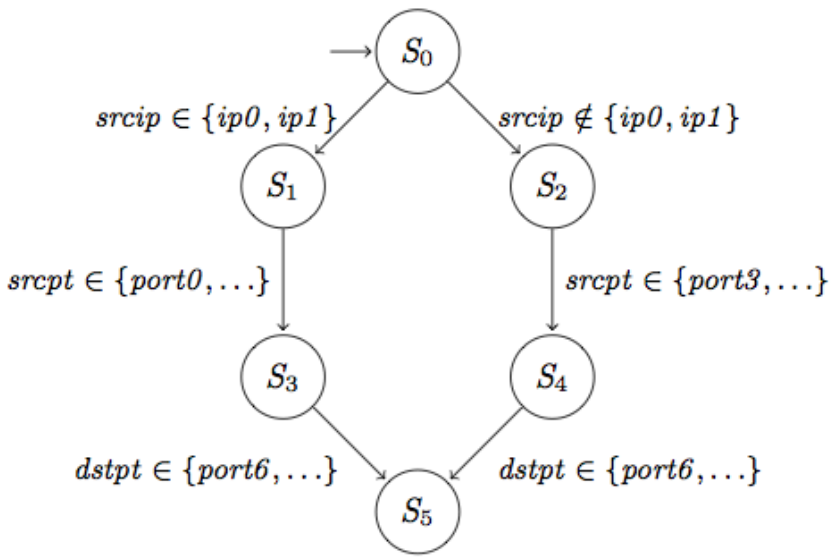
Example of verification by model checking

```
F1 = match(srcip=IP("198.122.37.15")) +
     match(srcip=IP("253.182.3.14"))

F2 = match(srcport=1000) + match(srcport=2000) +
     match(srcport=3000)

F3 = match(srcport=4000) + match(srcport=5000) +
     match(srcport=6000)

F4 = match(dstport=7000) + match(dstport=8000) +
     match(dstport=9000)

chain = ((F1 >> F2)+(~F1 >> F3)) >> F4
```

Pyretic firewall chain

```
AG(((srcip=ip0 | srcip=ip1) &
    (srcpt=port0 | srcpt=port1 | srcpt=port2) &
    (dstpt=port6 | dstpt=port7 | dstpt=port8))
   -> EF state=S5)

AG(((!(srcip=ip0 | srcip=ip1)) &
    (srcpt=port3 | srcpt=port4 | srcpt=port5) &
    (dstpt=port6 | dstpt=port7 | dstpt=port8))
   -> EF state=S5)
```

Chain specifications in CTL



Data plane FSM

Outline

1. Problem Statement
2. Related Work
3. Automated Verification of Security Chains
4. Prototype and Performance Evaluation
5. Conclusions

Our prototype: the Synaptic checker

- Checker implemented in Python
- Extension of Pyretic and Kinetic languages
- Implementation of a module dedicated to traffic properties

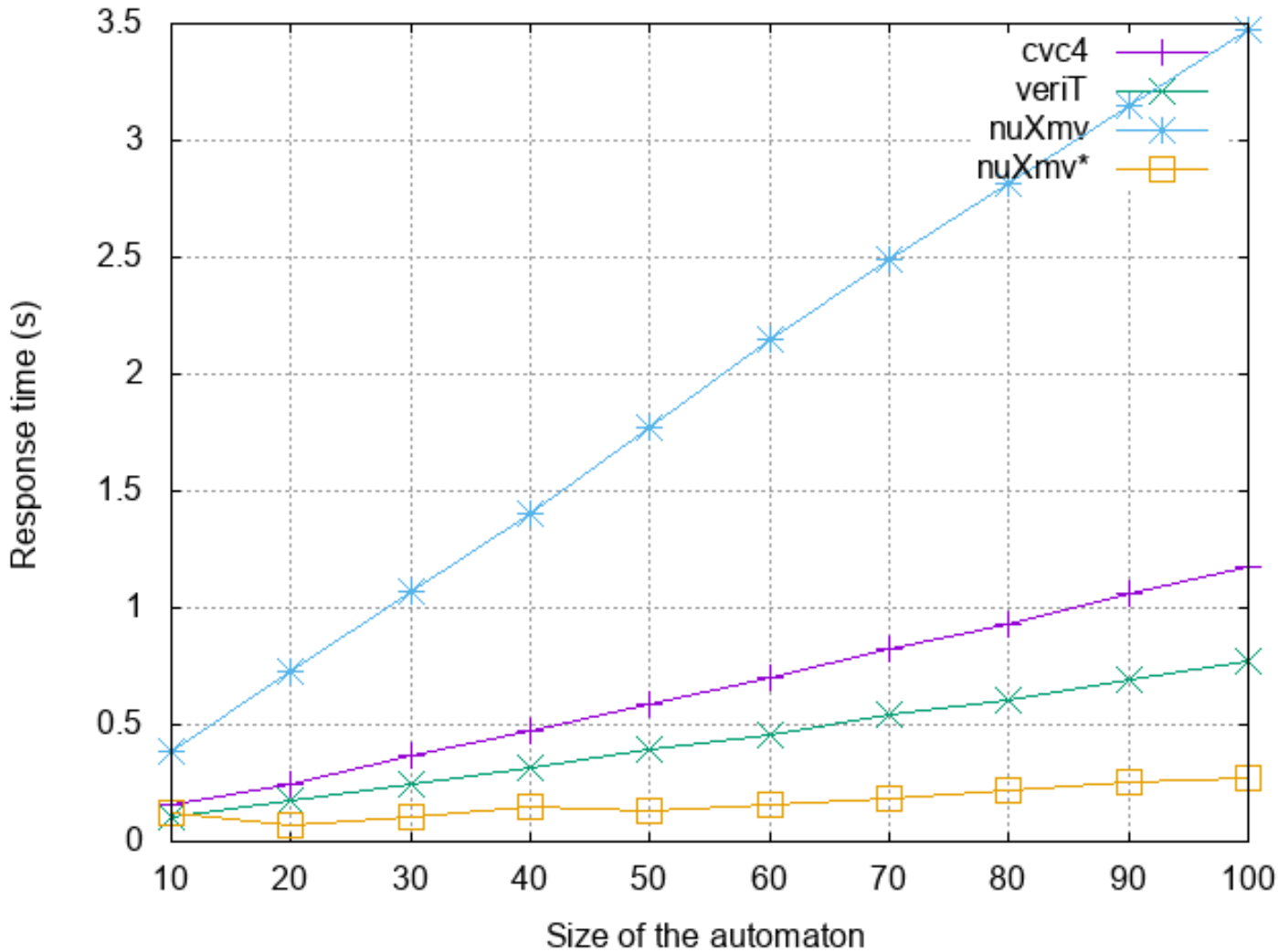
- Integration of formal verification methods
- Use nuXmv or SMT solvers as backend verifiers

Experimental setup and evaluation criteria

Evaluation based on a set of synthetic security chains

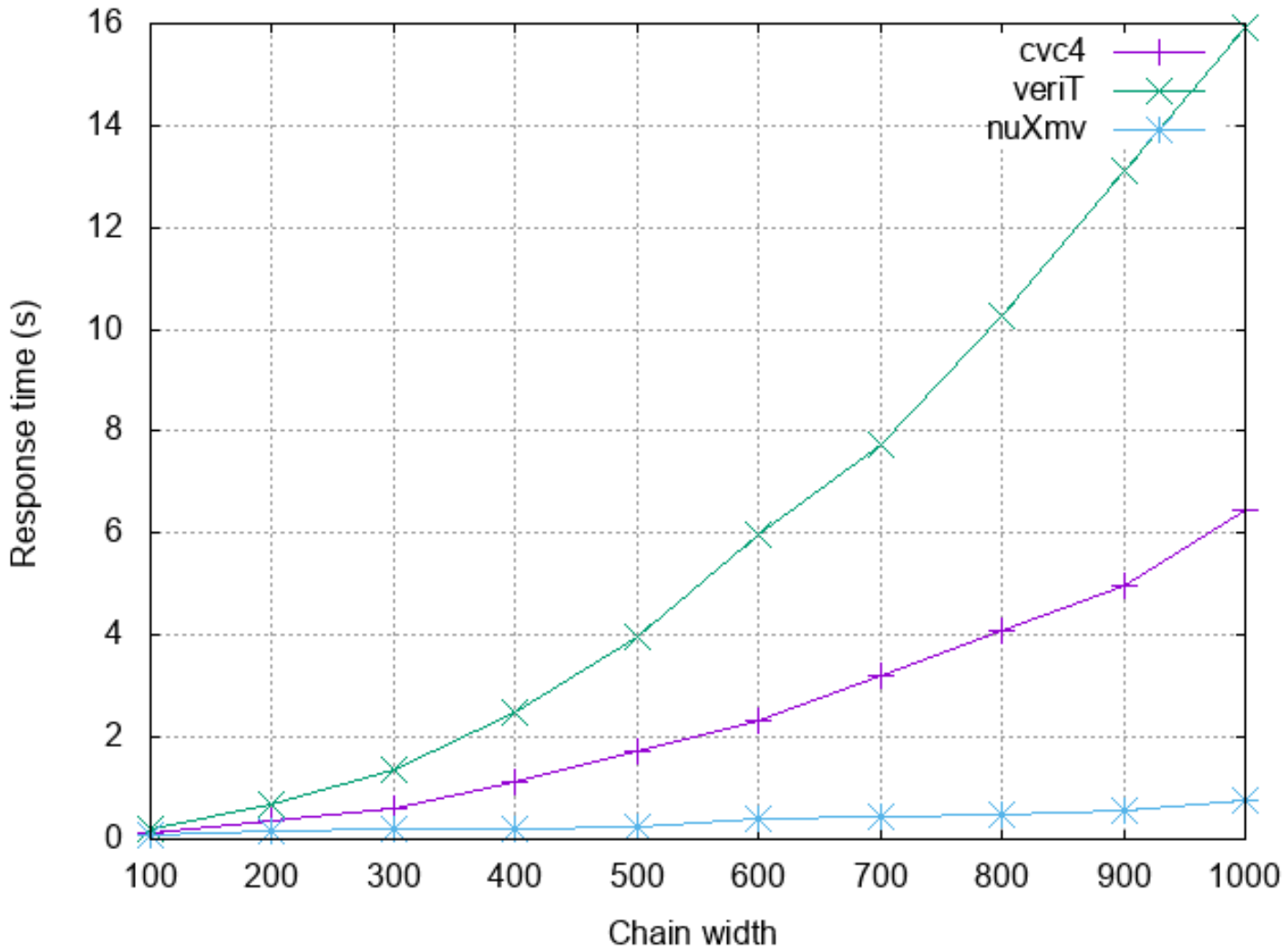
- Impact of several chain-related parameters
  - Size of the control automaton
  - Length and width of a security chain
  - Number of properties to be verified
- Comparison of verification performances
  - Response time
  - Memory space consumption

Impact of the size of the control plane automaton



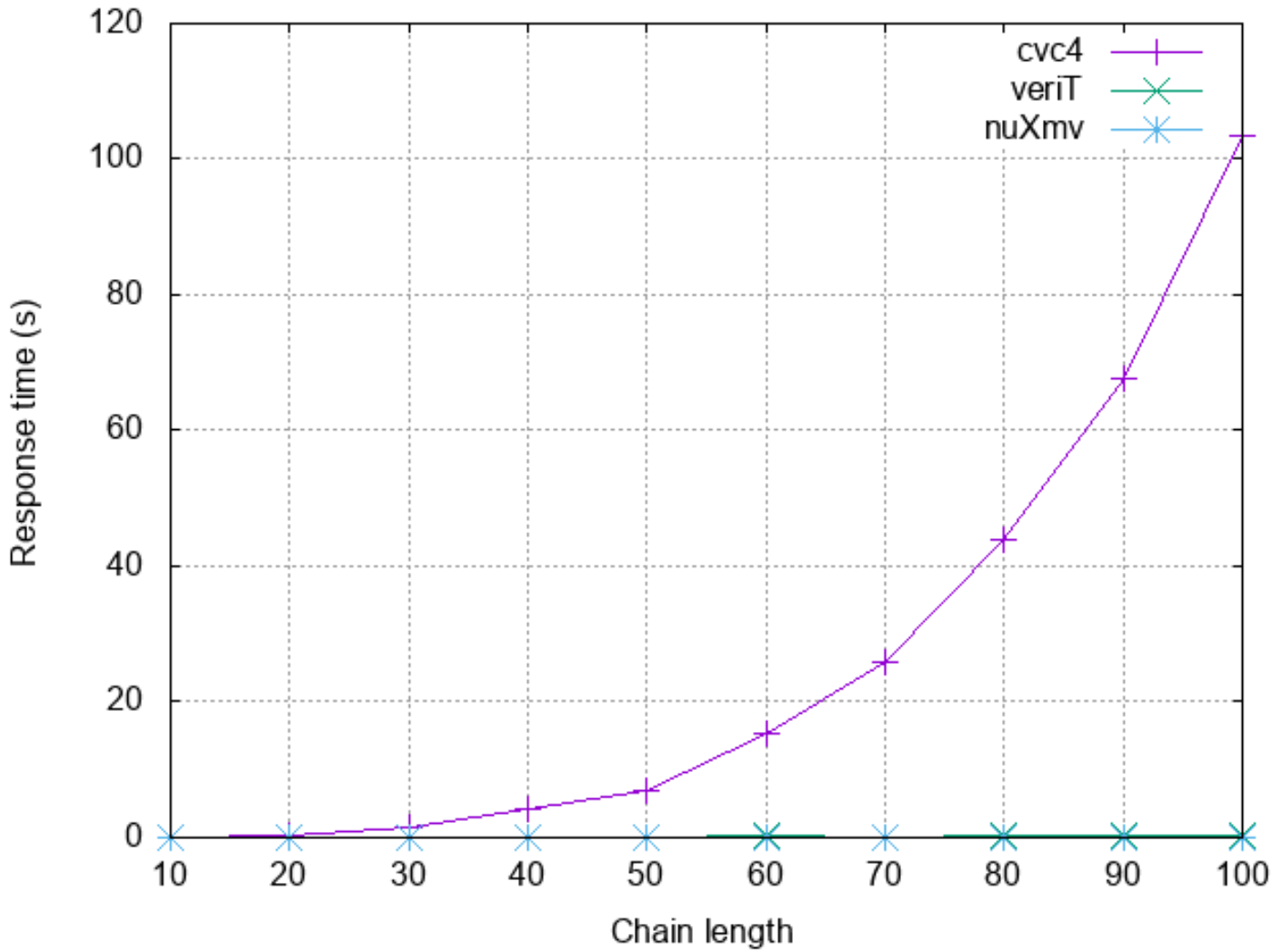
Response time vs. size of the control automaton

Impact of the width of a chain



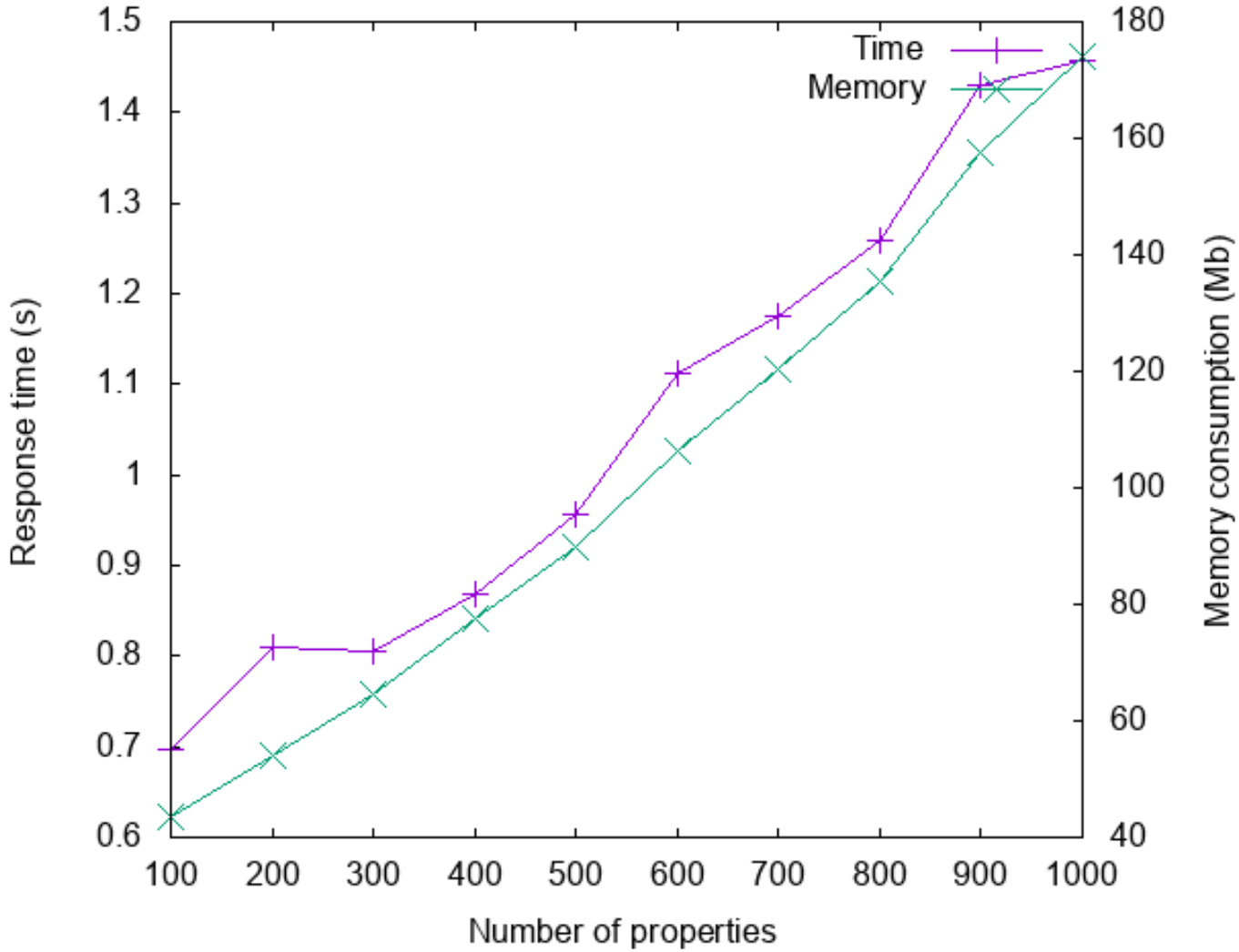
Response time vs. width of a chain

Impact of the length of a chain



Response time vs. length of a chain

Impact of the number of properties to verify



Resource consumption vs. number of properties (nuXmv)

Outline

1. Problem Statement
2. Related Work
3. Automated Verification of Security Chains
4. Prototype and Performance Evaluation
5. Conclusions

Conclusions

Automated techniques for verifying SDN security chains

- Checking both the control and the data planes
- Translation of security chains for automatic verification
- Backends based on SMT solving or model checking

Prototyping and performance evaluation

- Development of the Synaptic checker
- nuXmv shows best overall performance in our experiments

Perspectives

- Extensions for more complex and advanced rules
- Further optimization of the encodings into SMTlib and nuXmv
- Integration into an automated security framework