Verification and Synthesis of Security Chains

Stephan Merz
joint work with N. Schnepf, R. Badonnel, A. Lahmadi

Inria & LORIA, Nancy, France

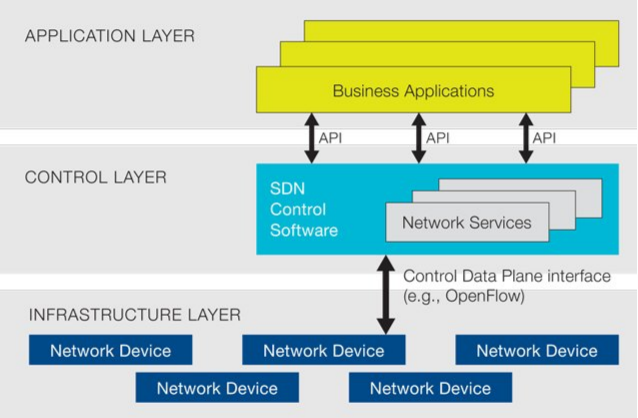
IFIP Working Group 2.2

Vienna, September 2019

Challenges

- Mobile equipments as attack platforms
  - $>$ 3M malicious applications on Google Play (G-Data, 2018)
  - ubiquity of phones and tablets attracts attackers
- Safeguarding the network
  - prevent attacks mounted from mobile terminals
  - network infrastructure enables protective measures
- Programmable networks (SDN)
  - allow for flexible network reconfiguration
  - virtual routers deployed in a cloud infrastructure
  - complex configuration rules are error-prone

SDN Architecture



- Two layers of processing rules
  - control plane: rules for forwarding packets to routers
  - data plane: process packets, mostly based on header information

Contents

SDN Programming and Verification

- Pyretic: a DSL for programming SDN controllers [Foster et al. 2013]
  - higher-level programming abstractions, compiled to OpenFlow
  - atomic rules: identity, drop, match, modify
    (plus some operators defined in libraries)
  - sequential and parallel composition: $\gg$, $+$

match(dstip=127.93.256.*) $\gg$
((match(port=4000) + match(port=5000)) $\gg$ drop)

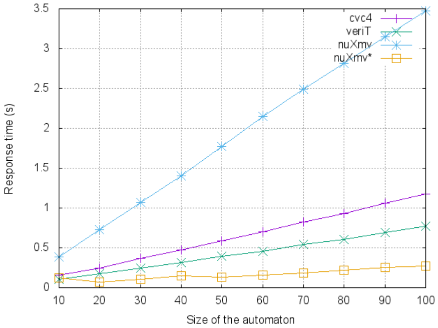
- Existing work for verifying SDN rules
  - data plane: Vericon [Ball et al. 2014], FlowChecker [Shaer et al. 2010], ...
  - control plane: Kinetic [Kim et al. 2015]

Formal Verification of Control and Data Planes

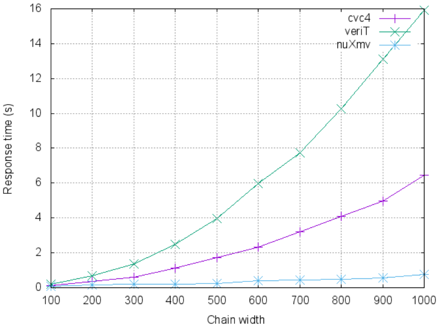
1. Encoding of Pyretic programs in SMTlib
    - represent addresses and ports by formal constants
    - match, modify: equations on header fields
    - $\gg$, $+$ represented as conjunction and disjunction
    - drop: negate expression describing rejected packets
    - properties express constraints about accepted / rejected traffic

2. Encoding as nuXmv models
    - represent control flow as finite state machine
    - constraints on headers processed in data plane
    - express properties as LTL or CTL formulas

Performance evaluation



Varying size of control plane



Varying width of data plane

nuXmv is both expressive and fast

Contents

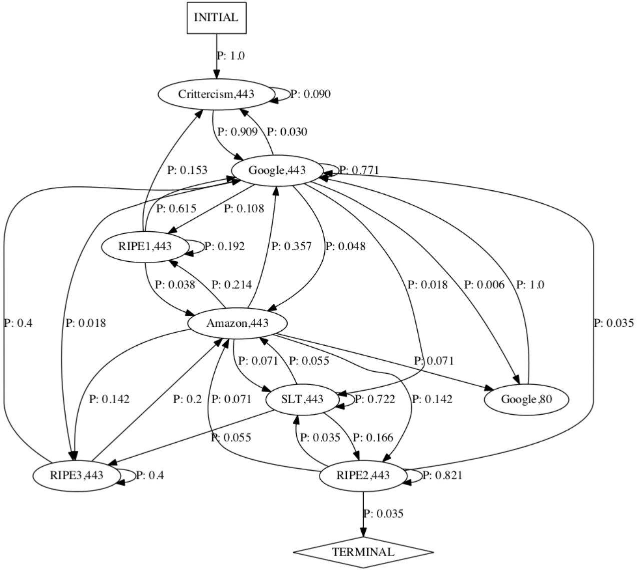
Objectives

- Generate security chains for mobile applications
  - observe the network traffic that an application generates
  - represent the network behavior as a Markov chain
  - synthesize an SDN program enforcing network policies
- Network traffic represented as flows
  - information about packets for same destination
  - ignore packet contents (often encrypted anyway)
  - useful for detecting attacks (DoS, port scanning, botnets etc.)
  - collect on device: associate flow with application
  - existing data sets [CTU 2013, Flowoid]

From Network Flows to Markov Chains

- States correspond to network destinations
  - record which servers an application contacts
  - aggregate IP addresses according to their orgname
- Transitions reflect successions of destinations
  - record in which order destinations are visited
  - transition probabilities according to frequency of visits
- Adaptation of techniques for process learning
  - favorable comparison with existing tools (Synoptic, Invarimint)

Example: Automaton for Pokemon Go



Classify Application Behavior

- Detect potential malicious behavior
  - basis: network behavior represented by Markov chain
  - appeal to BGP ranking service: trustworthiness of destinations
  - operator-defined thresholds for identifying attacks
  - take into account application permissions (spyware)
- Encode classification rules as Horn clauses
  - declarative representation for ease of modification
  - basis for reasoning about properties of synthesized chains
- Example of classification rule

$$\begin{aligned} dos(a) \ \leftarrow \ & \wedge f \in t_{app} \wedge a = f.dstaddr \wedge (l_f, p, l_f) \in T_{app} \\ & \wedge p \geq attack\_limit \wedge count(a, l_f) \geq ip\_limit \\ & \wedge avg\_interval(l_f) \leq min\_interval \wedge avg\_size(l_f) \leq min\_size \end{aligned}$$

Infer High-Level Representation of Security Chains (1)

- Determine which elementary rules should be deployed
  - forward, block or limit the number of packets
  - ensure that packets match protocol type (tcp, udp, http, ...)
  - invoke filtering or deep packet inspection services

$$\begin{aligned} deploy_{block}(a, pt) &\leftarrow botnet(a, pt) \\ deploy_{limit}(a) &\leftarrow dos(a) \\ deploy_{forward}(a) &\leftarrow \neg worm(a, pt) \wedge \neg botnet(a, pt) \end{aligned}$$

- Define the effect of elementary rules on network traffic

$$\begin{aligned} forward(a, t) &= restrict(t, \lambda pk : pk.dstaddr = a) \\ block(a, pt, t) &= restrict(t, \lambda pk : pk.dstaddr \neq a \wedge pk.dstport \neq pt) \\ limit(a, t) &= cut(forward(a, t), ip\_limit) \end{aligned}$$

Infer High-Level Representation of Security Chains (2)

- Group inferred rules into security functions

$$
\begin{aligned}
stateless\_firewall(t) &= \\
&\qquad \bigoplus \{\, forward(a,t) : deploy_{forward}(a),\ a \in \textsc{Addr} \,\} \\
&\oplus \quad \bigoplus \{\, block(a,pt,t) : deploy_{block}(a,pt),\ a \in \textsc{Addr},\ pt \in \textsc{Port} \,\} \\
ids(t) &= \bigoplus \{\, limit(a,t) : deploy_{limit}(a),\ a \in \textsc{Addr} \,\} \\
stateful\_firewall(t) &= \ldots
\end{aligned}
$$

- Build chains from security functions

$$dos\_chain = stateless\_firewall \gg ids \gg stateful\_firewall$$

- Properties of chains ensured by construction
  - absence of loops and black holes
  - shadowing freedom, coherence of single chains
  - chains for different applications need not be coherent

Evaluation of Generated Chains

- Method of evaluation
  - 7000 network flows corresponding to 10 applications
  - use 70% of each flow for generating the chains
  - inject port scanning attack into remaining 30%

application	# dests.	# rules	avg. acc.
disneyland	5	44	0.992
dropbox	17	311	0.997
faceswitch	30	425	0.812
lequipe	208	1640	0.518
meteo	90	716	0.837
ninegag	124	930	0.509
pokemongo	24	485	0.743
ratp	3	28	0.940
skype	442	6529	0.998
viber	176	4163	0.683

$\Rightarrow$ Improve detection for applications whose destinations vary

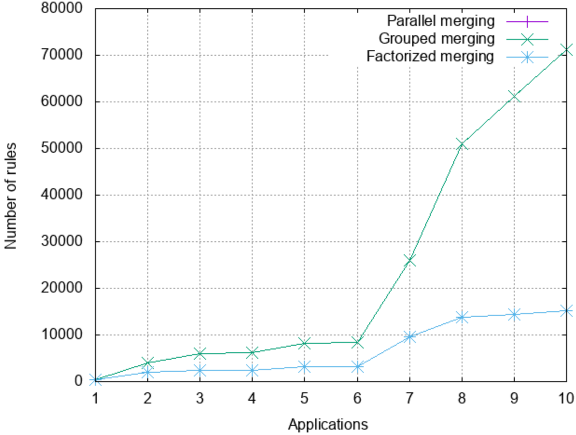
Contents

Combine Chains for Different Applications

- Must handle packets generated from different applications
  - naive approach: parallel composition or joint learning
    $\rightsquigarrow$ large chains, learning effort, risk of incoherence
  - in practice, many chains have common elements
- Algorithm for merging security chains
  - merge functions of same type (firewall, IDS, ...)
  - combine the rules for these functions
  - identify conflicting rules and choose between them
- Properties of combined chains
  - absence of loops and black holes, shadowing freedom
  - coherence of overall chains, but risk of loss of precision

Experimental Evaluation

- Number of rules when composing chains

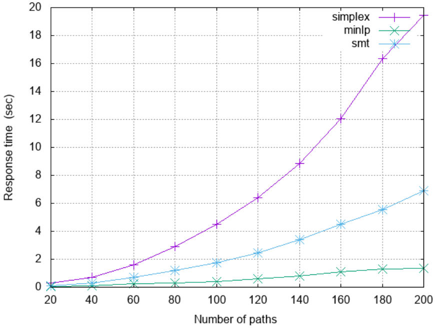


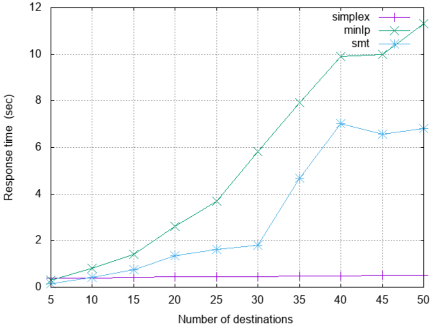
- Accuracy of attack detection unchanged
  - no conflicting rules in our experiments

Placement of Security Chains

- Assign rules to switches, forward packets according to chain
  - preserve the order of rules within a chain
  - respect capacities of switches and of interconnection network
  - optimize for network utilization, service congestion, availability
- Encode the problem using (non-)linear integer programming
  - aggregate destinations based on channel capacity
  - aggregate switches into network paths
  - constraints represent resource requirements of the chain
  - objective functions express (normalized) optimization criteria
  - use Simplex, MINLP, and optimizing SMT solvers

Performance Evaluation





- Preliminary evaluation over crafted examples
  - Simplex is robust to the number of destination aggregates ...
  - ... but highly sensitive to number of network paths

Contents

Summing Up

- Use of formal techniques in the context of SDN
  - verification techniques (SMT, model checking)
  - automaton learning for characterizing application behavior
  - declarative programming for chain synthesis
  - merging and optimization for the deployment of chains
- Experiences and perspectives
  - promising experiments in simulated environments
  - improve accuracy of chains in the case of varying destinations
  - enable on-the-fly adaptations of chains
  - better take into account application permissions and privacy risks