

Departmental PhD students workshop, 2009

Formal Analysis of Security Protocols

Ta Vinh Thong

PhD student, 1st year, (thong@crysys.hu)

Supervisor: Dr. Buttyán Levente
Department of Telecommunications

Budapest University of Technology and Economics
Laboratory of Cryptography and Systems Security (CrySys)

Outline

1. Formal Analysis of Security APIs.
2. Security Verification of Firewalls.
3. Automated Security Verification of Mobile Ad-hoc Networks.

Outline

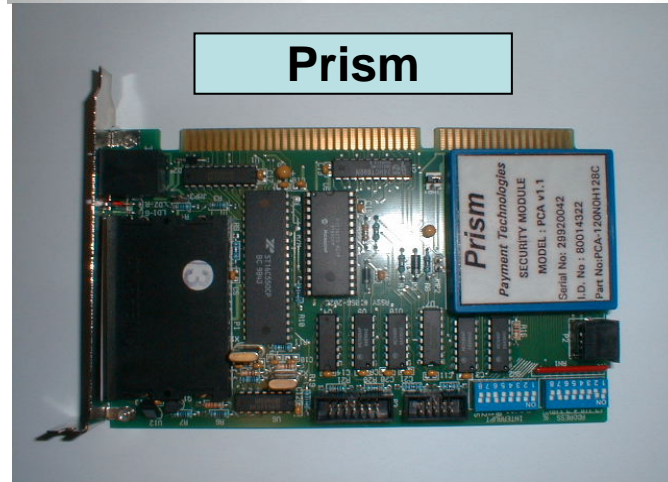
1. Formal Analysis of Security APIs.

- Overview
- Challenges and related works
- Our work
- Conclusion and future works

2. Security Verification of Firewalls.

3. Automated Security Verification of Mobile Ad-hoc Networks.

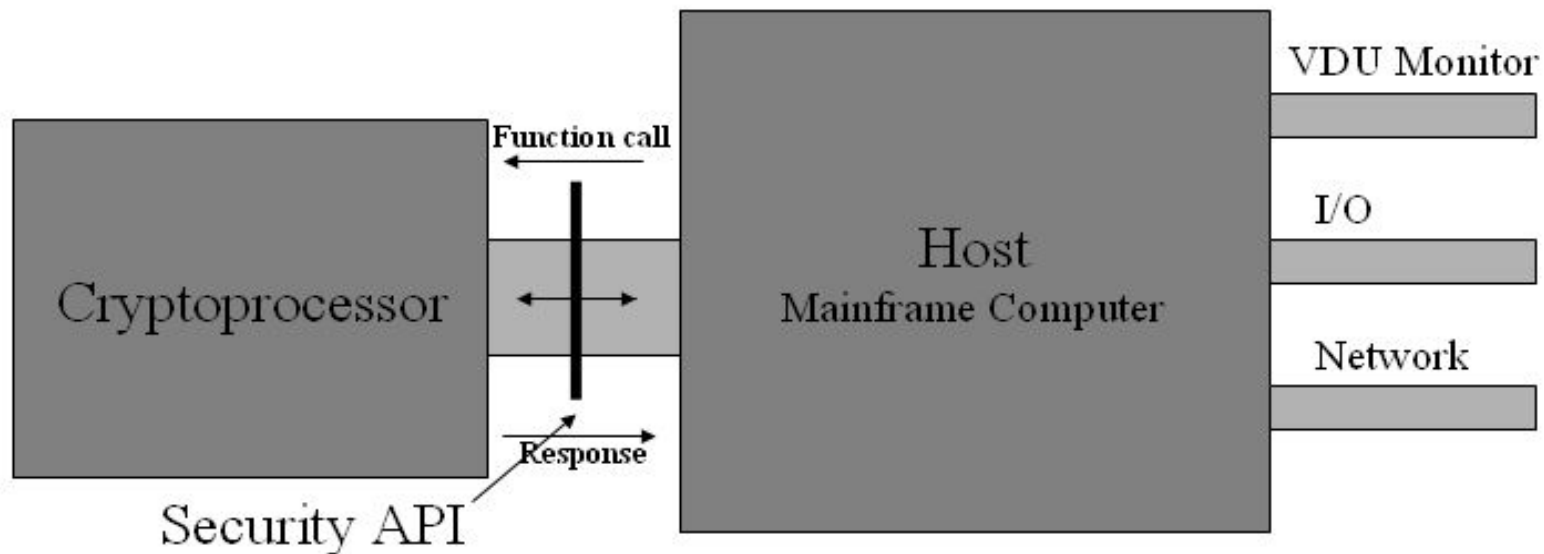
What is Hardware Security Modules(HSMs)?



- Provide high-level physical protection
- Store sensitive data
- Perform cryptographic op.
- Very expensive
- Are applied in:
 - Banking(ATM) networks
 - Electronic commerce

What is Security API (Security Application Programming Interface) ?

- To invoke the functions provided by the module.
- Use cryptography.
- Enforce a security policy.



Attacks against HSMs

- Physical attacks
 - HSMs are tamper resistant.
 - Very expensive.
- API attacks
 - Discovered ~ 2002.
 - Exploit the bad designing of APIs.
 - More cheaper.

A simplified API

- The API consists of four functions:
 1. Data-encryption
 2. Data-decryption
 3. Key-export
 4. Key-import

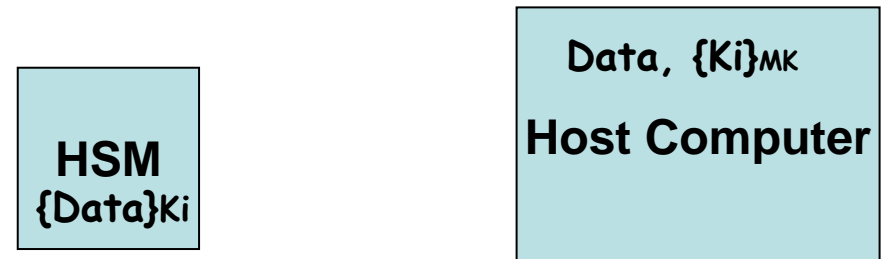
Only most important keys are stored inside due to memory restriction.

- One *MK* master key (stored inside)
- Data encryption key – K_i (stored outside in form $\{K_i\}_{MK}$)
- Key encryption key – KEK_j (stored outside in form $\{KEK_j\}_{MK}$)

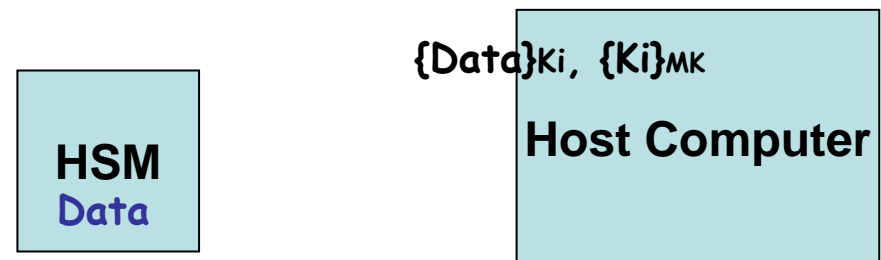
A simplified API cont'd

- data-encryption:
 - inputs: Data, $\{K_i\}_{MK}$
 - output: $\{Data\}_{K_i}$
- data-decryption:
 - inputs: $\{Data\}_{K_i}, \{K_i\}_{MK}$
 - output: Data
- key-export:
 - inputs: $\{K_i\}_{MK}, \{KEK_j\}_{MK}$
 - output: $\{K_i\}_{KEK_j}$
- key-import:
 - inputs: $\{K_i\}_{KEK_j}, \{KEK_j\}_{MK}$
 - output: $\{K_i\}_{MK}$

Invoking *data-encryption* function:



Invoking *data-decryption* function:



Attacking the simplified API

1. key-export:

- inputs: $\{K_i\}_{MK}, \{KEK_j\}_{MK}$
- output: $\{K_i\}_{KEK_j}$

2. data-decryption

- inputs: ~~$\{Data_i\}_{KEK_j}, \{KEK_j\}_{MK}$~~
- output: K_i

$K_i, KEK_j \leftarrow$ secured under the same MK master key!!!
(they have a same type)

PIN (Personal Identification Number) attacks

Extract customer's PIN by a subtle series of API call

- Key separation attack (against Visa Security Module).
- Check value attacks (Visa Sec. Module, IBM 4758)
- Decimalization attack (Financial APIs)
- Type-casting attack (IBM 4758)
- Key-conjuring attack (Financial APIs)
- Related key attack (Financial APIs)

Challenges and related works

- In real life, security APIs may contain few hundred calls.
- APIs attacks are very subtle. Most of the commercially available APIs had flaws.
- Informal verification is error-prone.

- Apply *bounded model checker* to discover API-level vulnerabilities.
 - ☺ Find old and new attacks in IBM 4758 CCA.
 - ☹ Specification, property are described in logic. Lack formal semantics.
 - ☹ Bounded model checker cannot find attacks required the verification path that has a length greater than the bounded value. (Undecidable problem)

- There are several operations (e.g., XOR) appear in APIs that are not supported in existing formal languages and tools.
 - There are some works.
 - ☹ Not based on existing formal language and tools: Lack of formal semantics, not reliable, not efficient.

- Several API attacks exploit partial information leakage (such as guessing) to discover the whole secret. (e.g., Decimalization attack, PIN Block attack, Check value attack).
 - ☹ Cannot model them directly using existing formal languages and tools.

Our Work

- We propose the application of the *spi-calculus* for security API analysis.
 - Spi-calculus: Designed for verifying security protocols. Expressive semantics.
- We demonstrated how to model security APIs with spi-calculus, developed a technique to prove the security of APIs, and proved the security of a simplified API manually.
- We demonstrated how to use the *ProVerif* tool to find attacks, and prove the security of the simplified API.
 - Proverif: Fully automatic. Based on the pi-calculus.

My Related Publications:

1. F. Kargl, P. Papadimitratos, L. Buttyan, M. Mueter, E. Schoch, B. Wiedersheim, **Ta Vinh Thong**, G. Calandriello, A. Held, A. Kung, J.-P. Hubaux, Secure Vehicular Communication Systems: Implementation , Performance, and Research Challenges, *IEEE Communications Magazine*, Vol. 46, No. 11, November 2008.
2. L. Buttyan, **Ta Vinh Thong**, Biztonsági API analízis a spi-kalkulussal , *Híradástechnika*, Vol. LXII, August 2007.
3. L. Buttyan, **Ta Vinh Thong**, Security API analysis with the spi-calculus, *Híradástechnika*, Vol. 64, January 2008.

Conclusion and Future Work

Conclusions

- spi-calculus is a promising method due to its expressive semantics, and well-suited syntatics.
- There are available automatic tools.
- However, they are primary proposed for verifying security protocols.

Possible Future Works

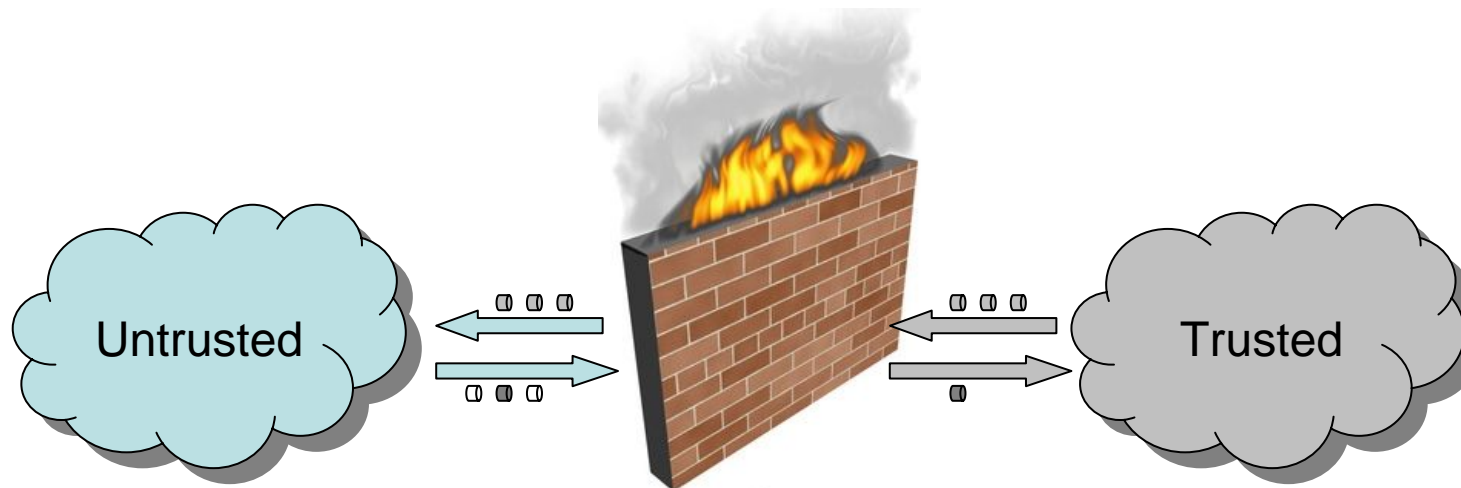
- Improve our method to verify larger real-life APIs (XOR, guessing attacks, etc.).
- ☹ Current formal languages and tools don't support them (directly)
→ need to be improved.

Outline

1. Formal Analysis of Security APIs.
- 2. Security Verification of Firewalls.**
 - **Problem**
 - **Motivation and goal**
 - **Challenges**
 - **Our work**
 - **Conclusion and future works**
3. Automated Security Verification of Mobile Ad-hoc Networks.

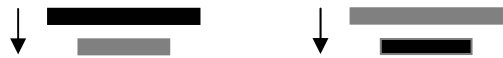
Problem

- A firewall is only as good as its configuration.
- Several firewalls are shown to be vulnerable due to bad configuration (Wool).
- Configuration errors
 - Inconsistency: Shadowing, Generalization, Correlation.
 - Inefficiency: Redundancy.



Inconsistencies and Inefficiency

Shadowing (r2 is shadowed by r1)



1.	udp	any	192.168.1.0/24	accept
2.	udp	172.16.1.0/24	192.168.1.0/24	deny
3.	udp	10.1.1.0/24	192.168.0.0/16	deny
4.	udp	172.16.1.0/24	any	accept

Shadowing is an error: Rule 1 can accept malicious traffic that Rule 2 intends to drop.

Generalization (r4 is a generalization of r2)



1.	udp	any	192.168.1.0/24	accept
2.	udp	172.16.1.0/24	192.168.1.0/24	deny
3.	udp	10.1.1.0/24	192.168.0.0/16	deny
4.	udp	172.16.1.0/24	any	accept

Generalization is a warning: It is not critical as shadowing, and can be used to reduced the size of rulesets.

Correlation (r1 and r3 are correlated)



1.	udp	any	192.168.1.0/24	accept
2.	udp	172.16.1.0/24	192.168.1.0/24	deny
3.	udp	10.1.1.0/24	192.168.0.0/16	deny
4.	udp	172.16.1.0/24	any	accept

Correlation is a warning.

accept
 deny

Motivation and goal

Motivations

- Existing inconsistency methods are proposed for only stateless firewalls
 - static analysis over "fix" ruleset.
 - FIREMAN
- There is a work [LiuSM2008] by Liu et al. where the authors formally model the connection-tracking feature of stateful firewalls.
 - but not proposed method for finding inconsistencies.
- Stateful firewalls are more broadly used than stateless firewalls.

**Goal: Inconsistency verification
of stateful firewalls.**

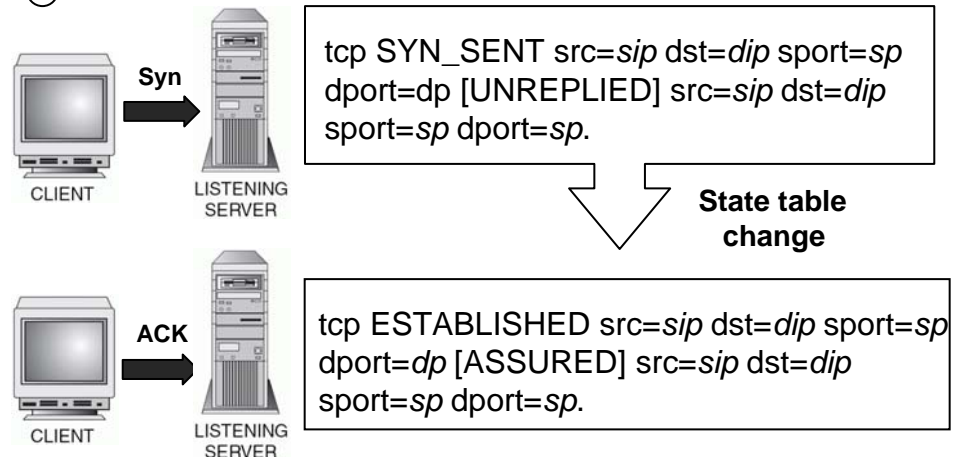
Challenges

Stateful firewall

- Handle more protocols than stateless firewalls.
- More secure than stateless firewalls.
- Connection tracking and stateful rules.

Stateful Ruleset

1. `iptables -A OUTPUT -p tcp -m state --state NEW, ESTABLISHED -j ACCEPT;`
2. `iptables -A INPUT -p tcp -m state --state ESTABLISHED -j ACCEPT;`



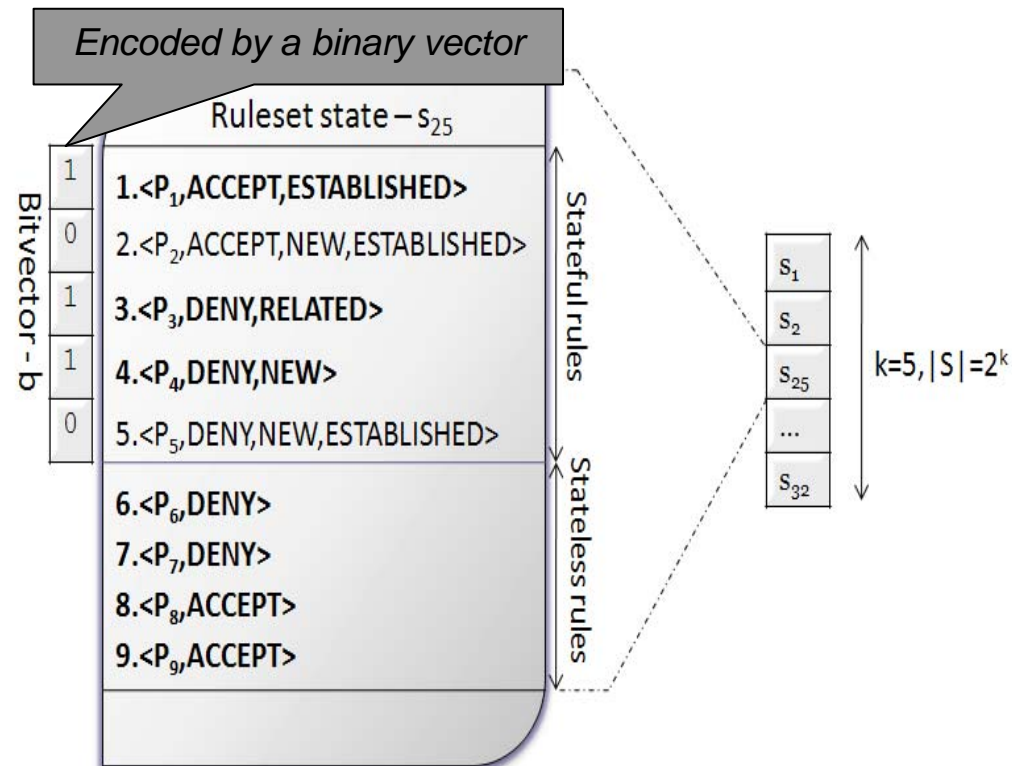
Challenges

- How to model states, define misconfigurations and security in case of stateful firewalls?
- There can be large number of states to be checked in stateful firewalls.
- Manually checking for inconsistencies is error-prone in large ruleset.

Our work

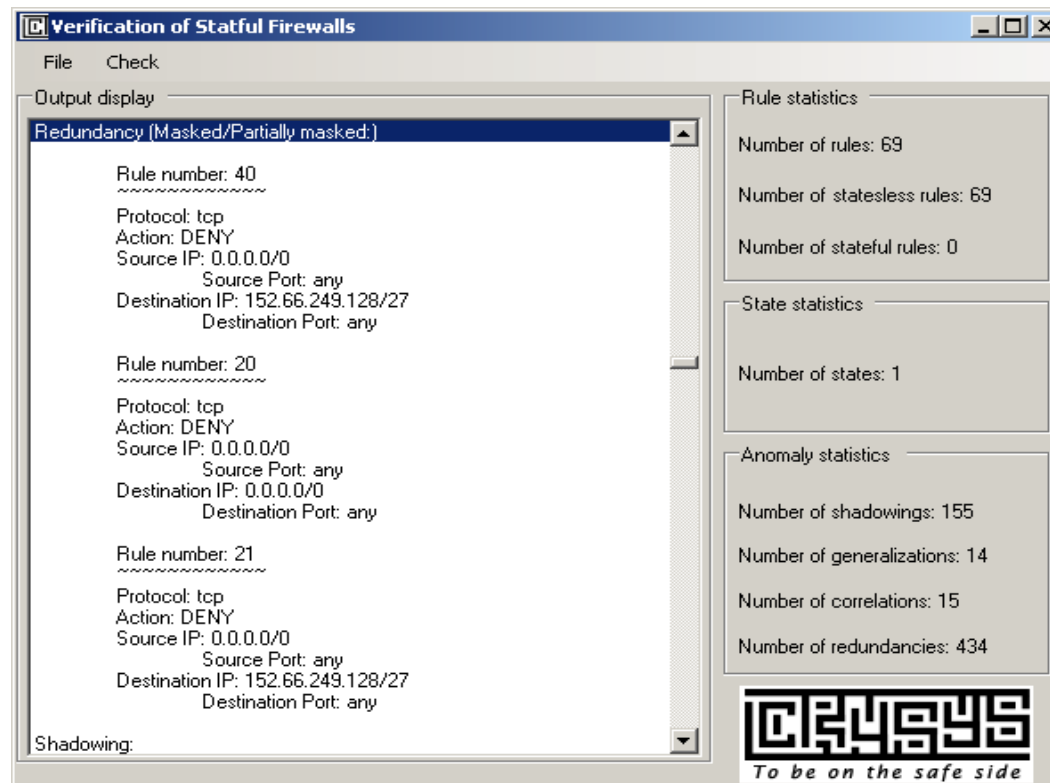
Def. The state of a firewall includes all the static firewall rules and those stateful rules that have an associated entry in the connection-tracking table.

Theorem: Let S be the state that contains all dynamic rules of the rule set. If no inconsistencies exist in state S , then all states are free from inconsistencies, and hence, the firewall configuration is correct.



Our work cont'd

- **Verification tool:**
 - *Verified the firewall of our labour.*
 - *Effective and efficient.*
- **Related Publication:** *L. Buttyán, G. Pék, Ta Vinh Thong, Consistency verification of stateful firewalls is not harder than the stateless case. Híradástechnika, vol. LXIII., no. 2009/2-3., 2009.*



Conclusion and Future Work

Conclusions

- The complexity of inconsistency verification of stateful firewall can be reduced to the stateless case.
- If there is no inconsistency in state S then all of the possible states are also inconsistency free.
- However, if inconsistencies are found in state S we have to check whether these inconsistencies can really happen. (still manually)

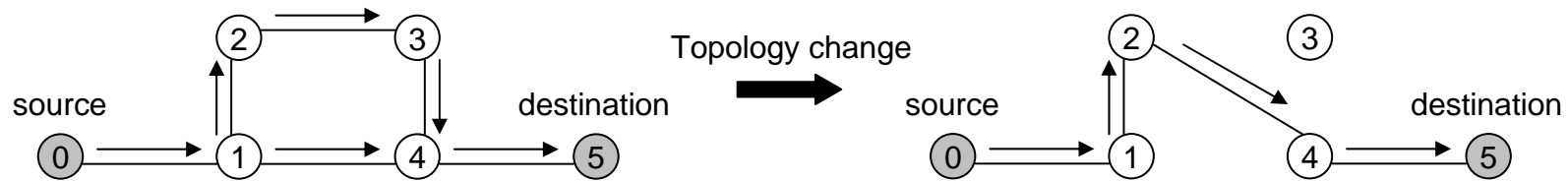
Possible Future Works

- Improve our tool to verify larger real-life statefull firewalls (complex chain, mangle, nat, etc).
- To automate the 3rd point of the conclusion part.

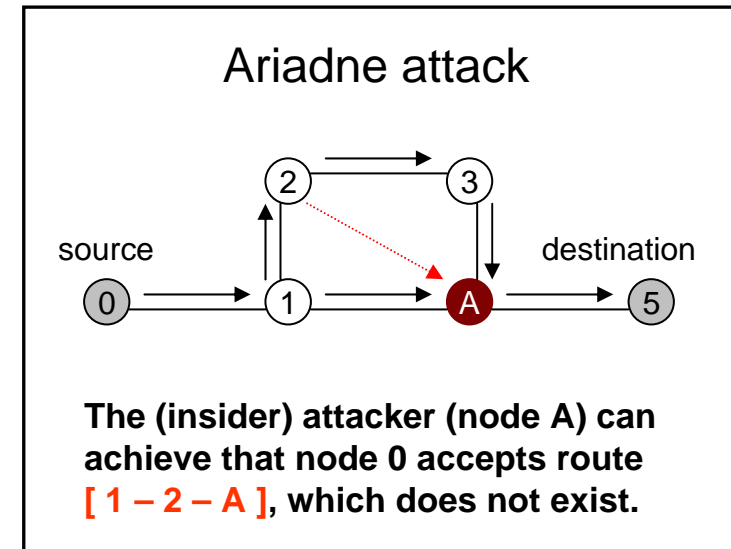
Outline

1. Formal Analysis of Security APIs.
2. Security Verification of Firewalls.
- 3. Automated Security Verification of Mobile Ad-hoc Networks.**
 - **Overview**
 - **Challenges and related works**

Overview



- Spontaneous network
 - computers which communicate over wireless medium.
- Topology
 - nodes (computers) and links (transmission ranges).
 - dynamics (topology change).
- Route **Discovery**/Maintainance
 - **route request/reply msg.**
- Several (secure) routing protocols
- Several type of attacks
 - route disruption
 - route diversion
 - rassing
 - gray hole/black hole
 - **creation of incorrect routing state attack**
 - **Modifying the routing state in some nodes the discovered route appears to be correct but in fact it does not exist.**
 - **Spoofing, forging, modifying, dropping control packets.**
 - **Degrade the quality of service.**

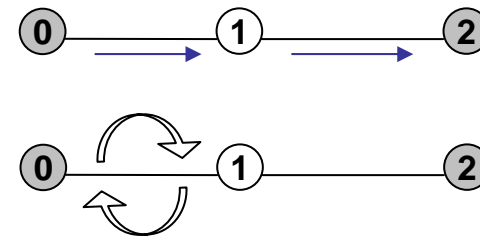


What to verify?

How to define the correct operation of an ad-hoc routing protocol?

(1) : If there at one point in time exists a path between two nodes, then the protocol must be able to find **some** path between the nodes.

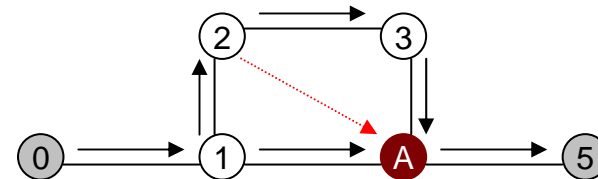
- Route from node 0 to node 2 : 0 - 1 - 2.
- This route has to be found.
- Such a routing state can be reached where (1) does not hold (e.g., loop).



Correct operation

(2): When a path has been found, it must be possible to send packets along the path from the source node to the destination node.

- In every reachable (topology, routing state) pair the routing state is consistent to the topology.
- Node A can achieve that node 0 accepts the route [1-2-A] as a valid route from 0 to 5. However, this route does not exist..



Challenges and related works

	Analysis approach	Approach characteristics			
		Discover unknown attacks	Property guarantees	Unconditional security	Fully automatic
Non-exhaustive	Visual Inspection	Yes	No	No	No
	Network Simulation	No	No	No	No
Exhaustive	Analytical proof	Yes	Yes	No	No
	Simulatability Model (Simulation paradigm)	Yes	Yes	No	No
	Formal Methods (CPAL-ES, SPIN, UPPAAL)	Yes	Yes	No	Yes

However, all of these methods make several constraints on the behaviour of attackers and the operation of protocols and topology.

- ☹ Large number of possible topology
 - ☹ Attacker can do anything (overhear, modify, drop, etc.)
- } **Huge number of states that cannot be stored and handled.**
- ➡ Good symbolic method required to prevent state explosion.

Thank you very much

