Analysis of a Secure Service Proxy Toolkit

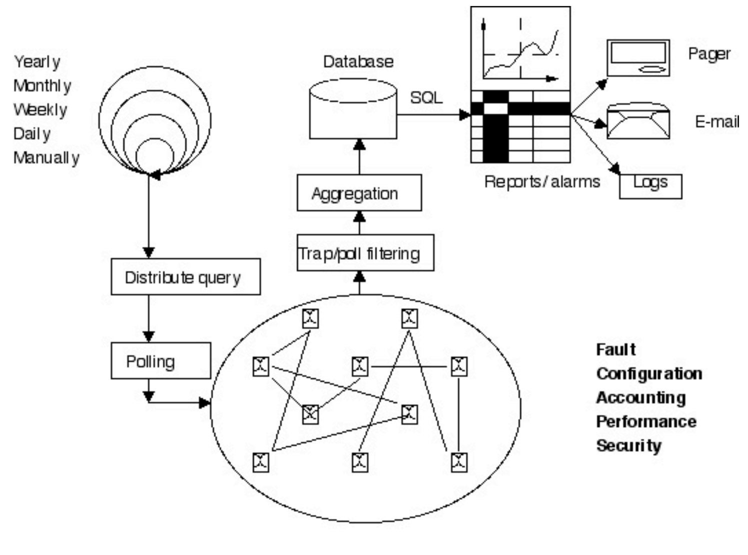
Carolyn Talcott
SRI International

In collaboration with
John Mitchell, Ninghui Li, Derrick Tong
Stanford

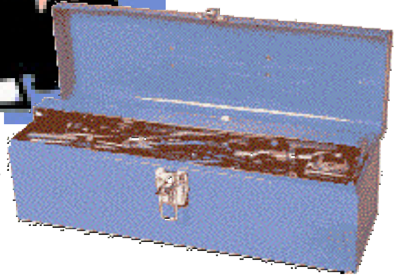
FTN Case Studies

- Proxy-based Distributed Systems
  - secure distribution of component based systems
- DDOS models
  - formal attack models, formal v&v?
- Secure Spread (Maude + uCAPSL)
  - is it secure?
  - is the group semantics preserved?
- TIARA project
  - intrusion tolerance for ad hoc networks
- Distributed/replicated databases
  - formal verification of core algorithms
  - reuse to verify DB specific optimizations

Maude Methodology

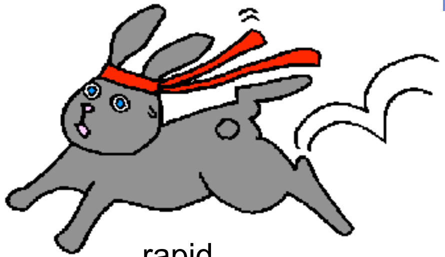


Resource monitoring



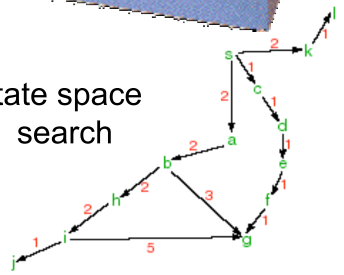
S |=□

model checking



rapid prototyping

state space search



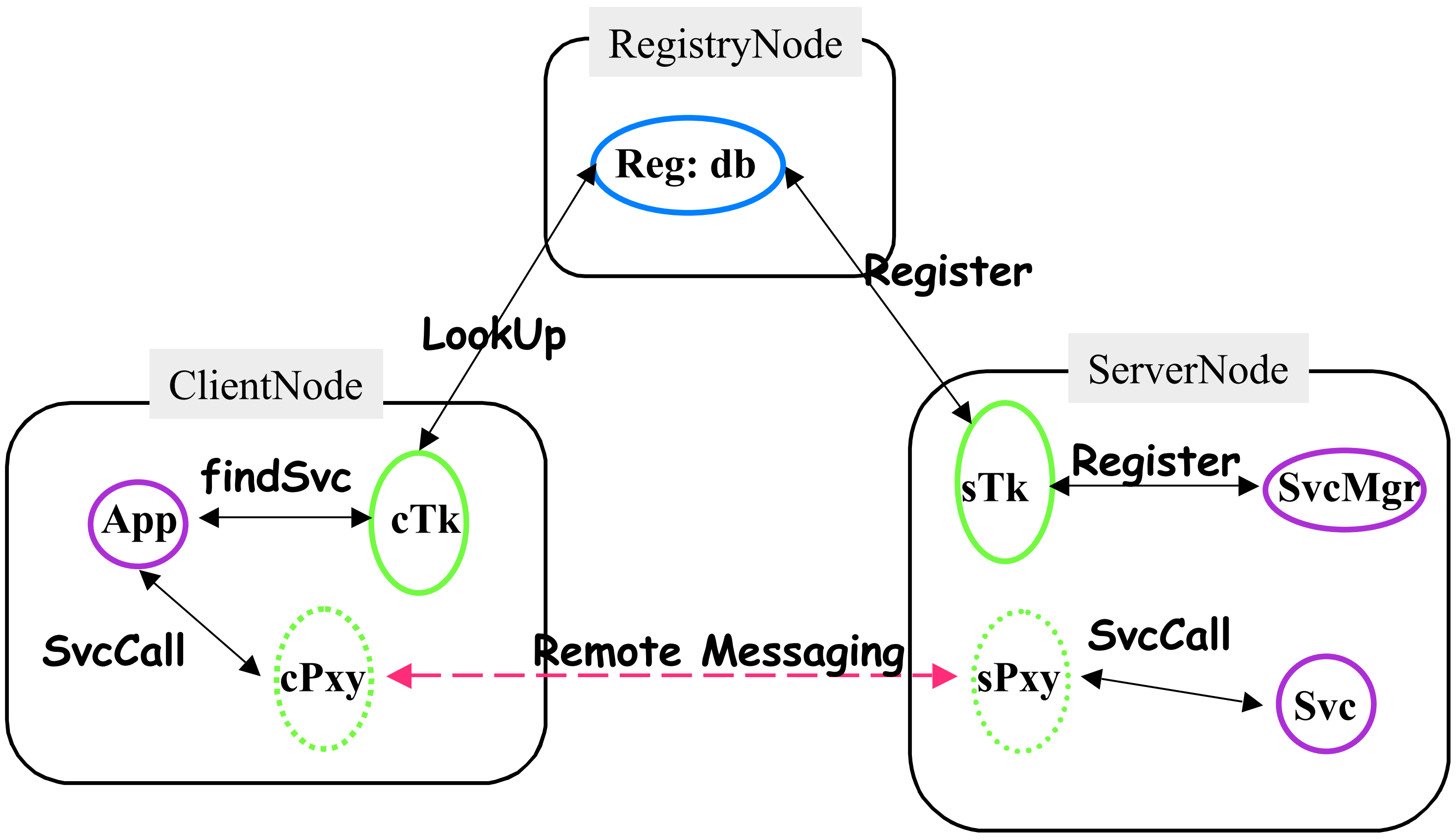
Remote Services

Requirements

- Publication and discovery
- Remote messaging
- Qos
  - Transparency
  - Security
    - Getting the right / expected service
    - Confidentiality

Approach: Service Proxy Toolkit (SPTK)

Service Proxy Toolkit Architecture

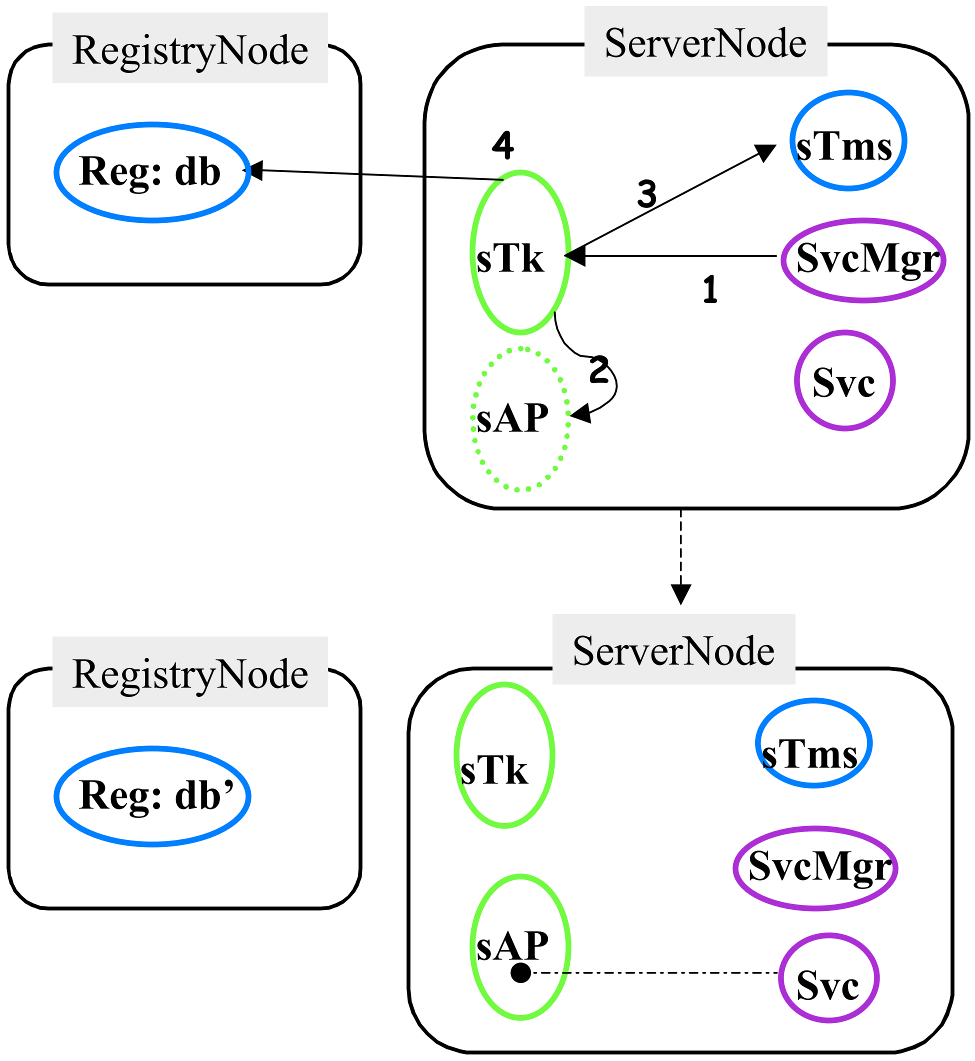


The Secure Server Proxy Toolkit in Pictures

Security Goals

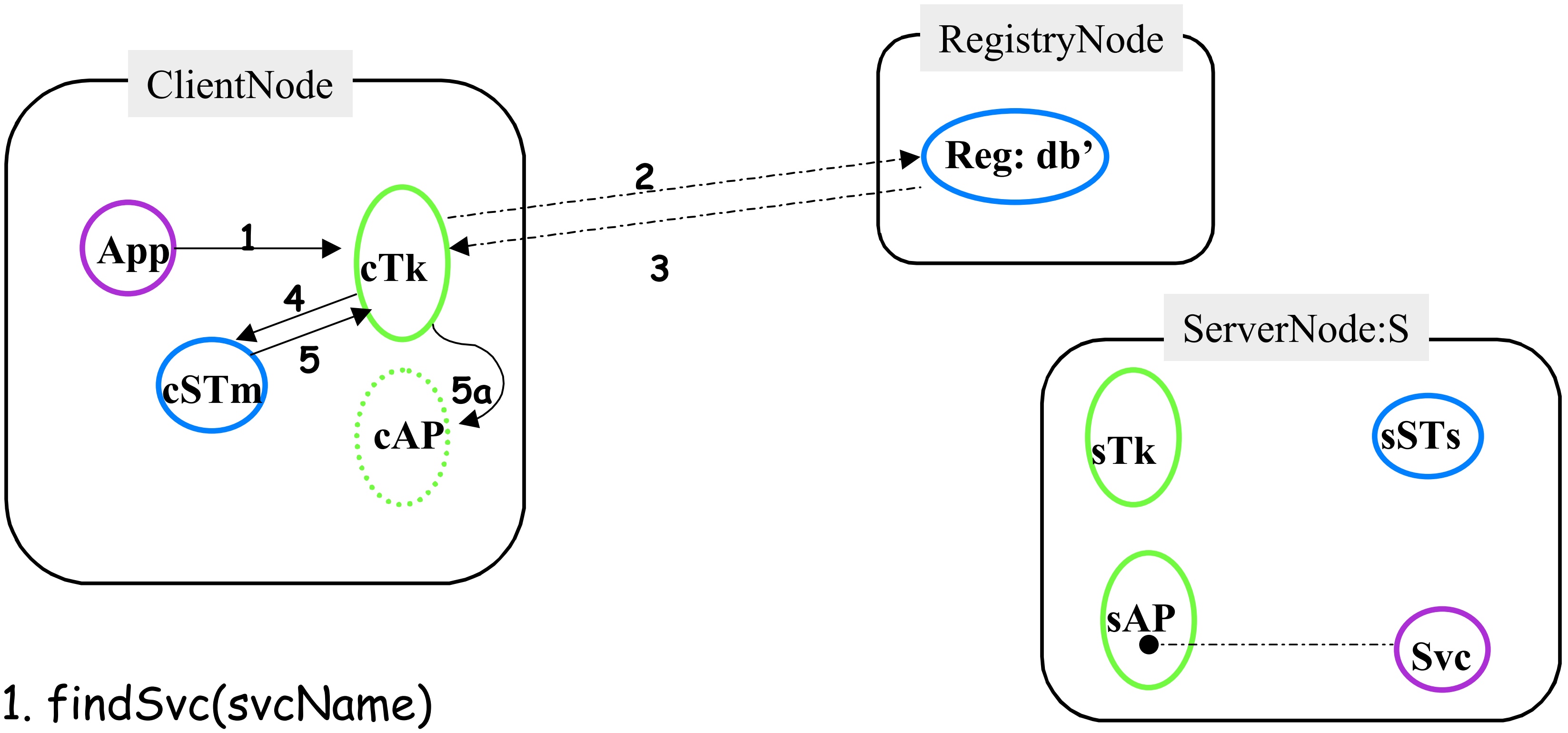
- Goal 0: Client VM protected from evil proxy
- Goal 1: Secure client-server communication
- Goal 2: Client can authenticate service proxy
- Goal 3: Server can also authenticate client

Registering a signed secure proxy



1. RegReq(svcName,Svc)
2. create(sAP)
3. sign(cAPd)
4. register(svcName,signedcAPd)

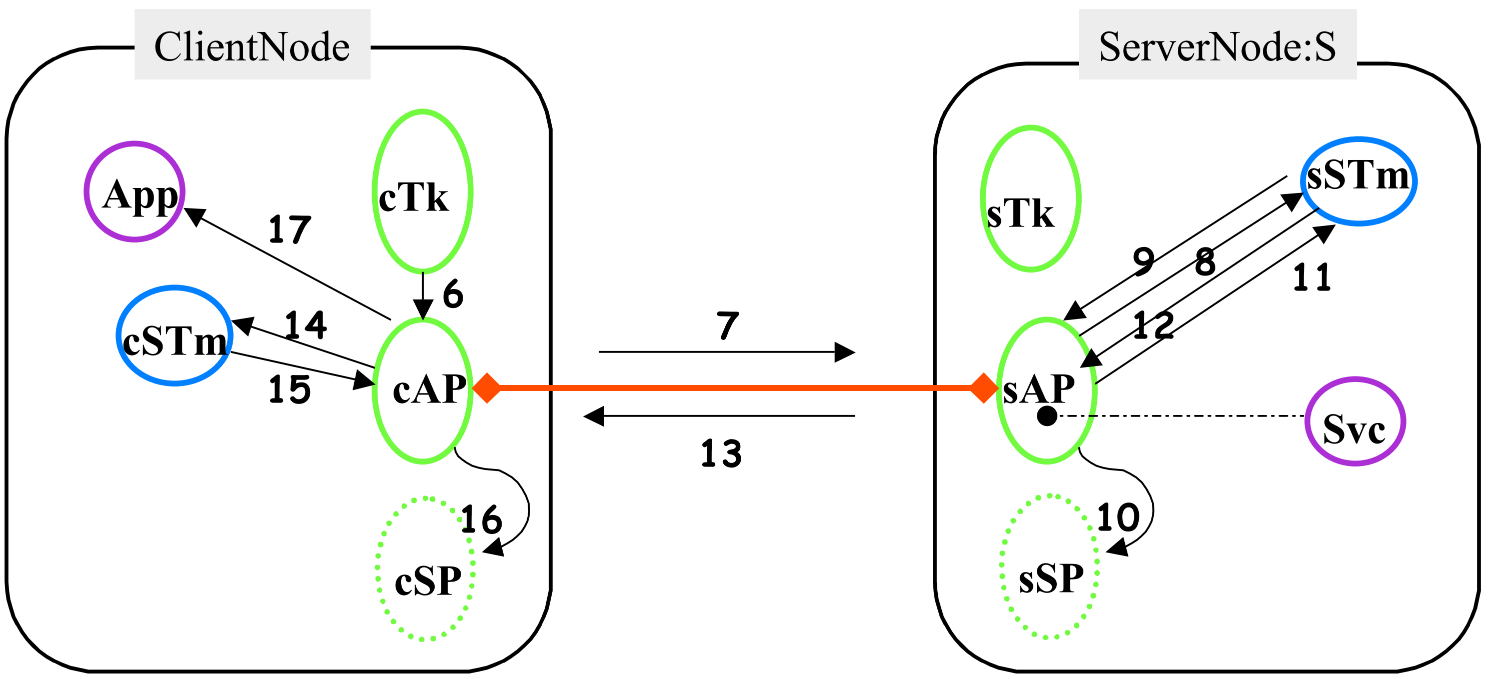
Getting an authentication proxy



1. findSvc(svcName)
2. lookup(svcName)
3. reply(signedcAPd)
4. verify(signedcAPd,tsK) must check description!
5. ok(apxd)

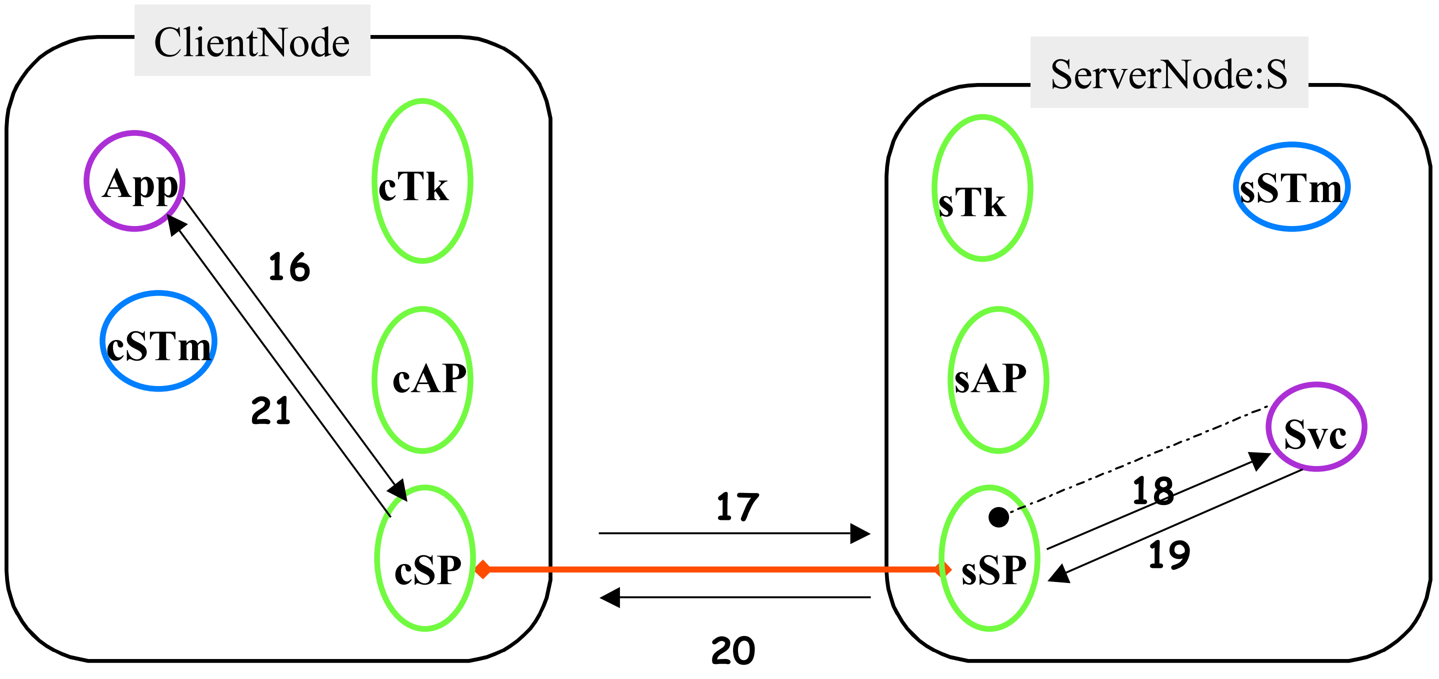
5a. install(apxd)

Setting up a secure session



6. authenticate(ccred)
6a. setup secure cnx ---
7. authenticate(ccred)
8. checkClient(ccred,svcName)
9. clientOk(perms)
10. install(sspd)
11. encryptReq(csspd,ccred)
12. encrypted(csspd,ccred)
13. encrypted(csspd,ccred)
14. decrypt(encrypted(csspd,ccred))
15. ok(csspd)
16. install(csspd)
17. findSvcReply(cSP)

Accessing the service



16. serviceCall(args)
17. serviceCall(args)
18a. check(args,per)
18. serviceCall(args,cId)

19. serviceReply(result)
20. serviceReply(result)
21. serviceReply(result)

The Secure Server Proxy Toolkit in Maude

Overview

- Model
  - SPTK Architecture
  - Attacker
  - Toolkit

- Compose and analyze

- Use
  - Maude object notation (with Russian dolls)
  - ACI (multi-set) rewriting
  - Rule conditions

Security Goals

- Goal 0: Client VM protected from evil proxy
- Goal 1: Secure client-server communication
- Goal 2: Client can authenticate service proxy
- Goal 3: Server can also authenticate client

Proxy Toolkit Models

- Level 0 : naive proxy (just does rmi)
  - Relies on JVM to achieve goal 0
- Level 1: Level 0 + secure communication
  - *Achieves goal 1*
- Level 2[t,f]: Level 1 + signed proxy
  - [with,without] checking proxy service name
  - *Achieves goal 2*
- Level 3: Level 2t + mutual authentication
  - *Achieves goal 3*

SPTK Architecture in Maude

- Infrastructure
  - JVM -- java execution env
  - ETHER-CLASS -- communication media (synch)
- Main components
  - SVC -- generic service
  - APP -- generic app/client
  - LOOKUP-IFACE -- registry interface
- Toolkit
  - SPTK-CLASS -- role independent structure
  - SSPTK-IFACE -- server-side interface
  - CSPTK-IFACE -- client-side interface
  - SP-CLASS -- proxy interface and structure

Server Node

```
< JS : JVM | jsatts:AttributeSet,
      {  socf:Configuration
         <  JS . sptk : SSPTK | tkatts:AttributeSet >
         <  JS . svc :  SVC |   svcatts:AttributeSet >
        msg( JS . sptk, JS . mgr, registerReq("Quote", JS . svc))
      } >
```

Attacker Models

- Attacker in the ether
  - Can read and modify communications
- Attacker lookup service
  - Chooses what to serve
- Both can
  - Register (dis) services
  - Impersonate valid client

ETHER Attack Configuration

- TEST
- ATTACKER
- LOOKUP
- initial configuration template

  econf<_> = (eacf ljcf ajcf<_> sjcf<_> cjcf<_>)
  - eacf = ether attack
  - ljcf = lookup node
  - ajcf<_> = impersonator
  - sjcf<_> = server node
  - cjcf<_> = client node

LOOKUP Attack Configuration

- TEST
- ATTACKER-LOOKUP
- initial configuration template

  lconf<_> = eecf aljcf<_> sjcf<_> cjcf<_>)
  - eecf = ether node
  - aljcf<_> = lookup attack + impersonator
  - sjcf<_> = server node
  - cjcf<_> = client node

SPTK Modules

SPTK<j> -- behavior rules for level j

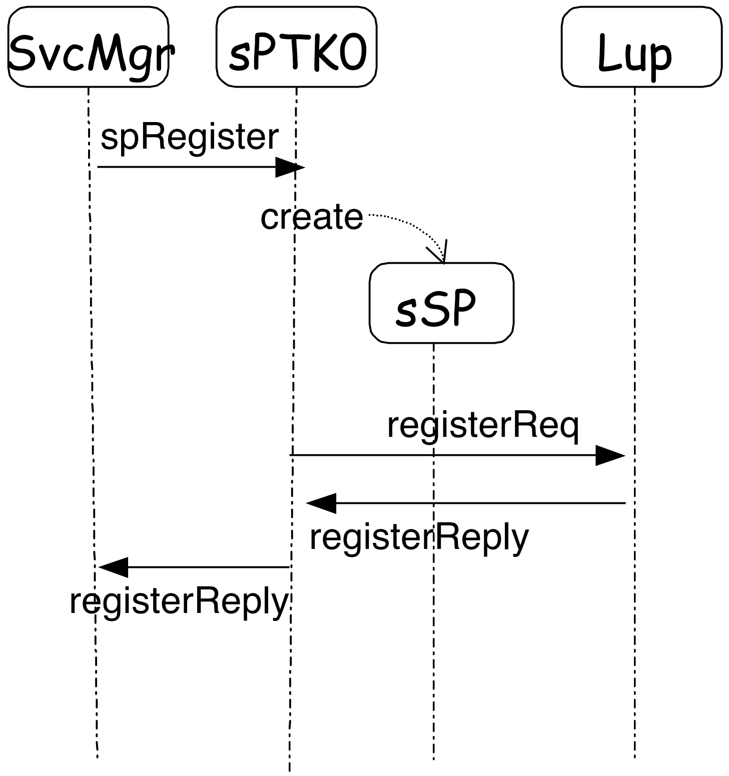
- Toolkit -- CSPTK<j> + SSPTK<j>
- Proxies -- CSP<j> + SSP<j>
- Level 3 adds pair of authentication proxies

Configurations to analyze:

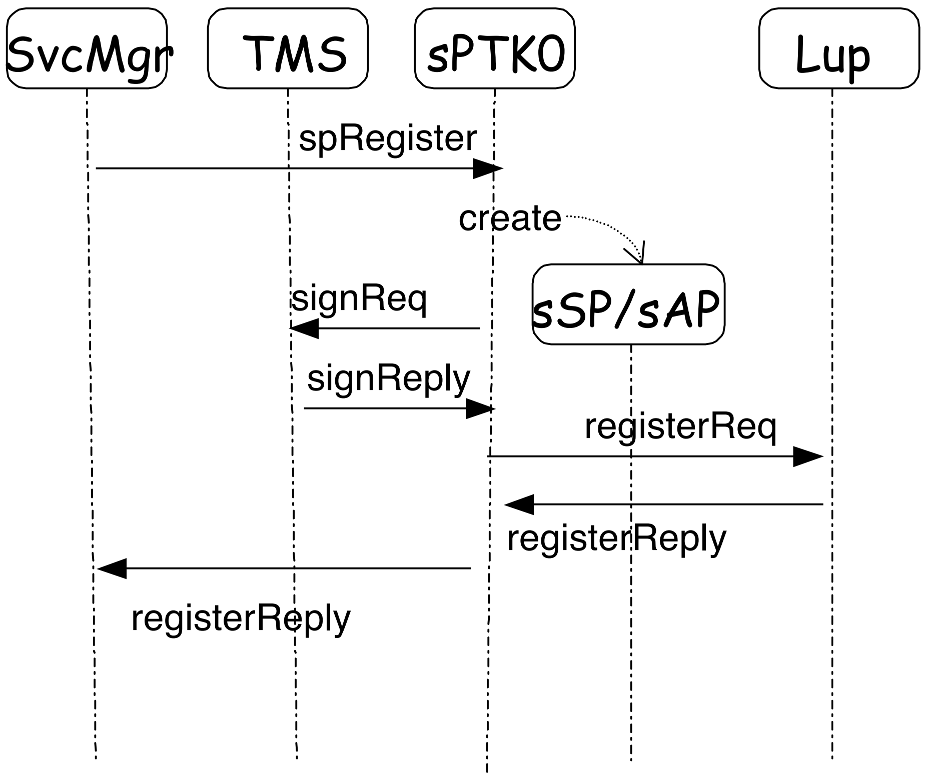
- econf<j> in TEST-ETHER + SPTK<j>
- lconf<j> in TEST-LOOKUP + SPTK<j>

Registering a service

Levels 0,1

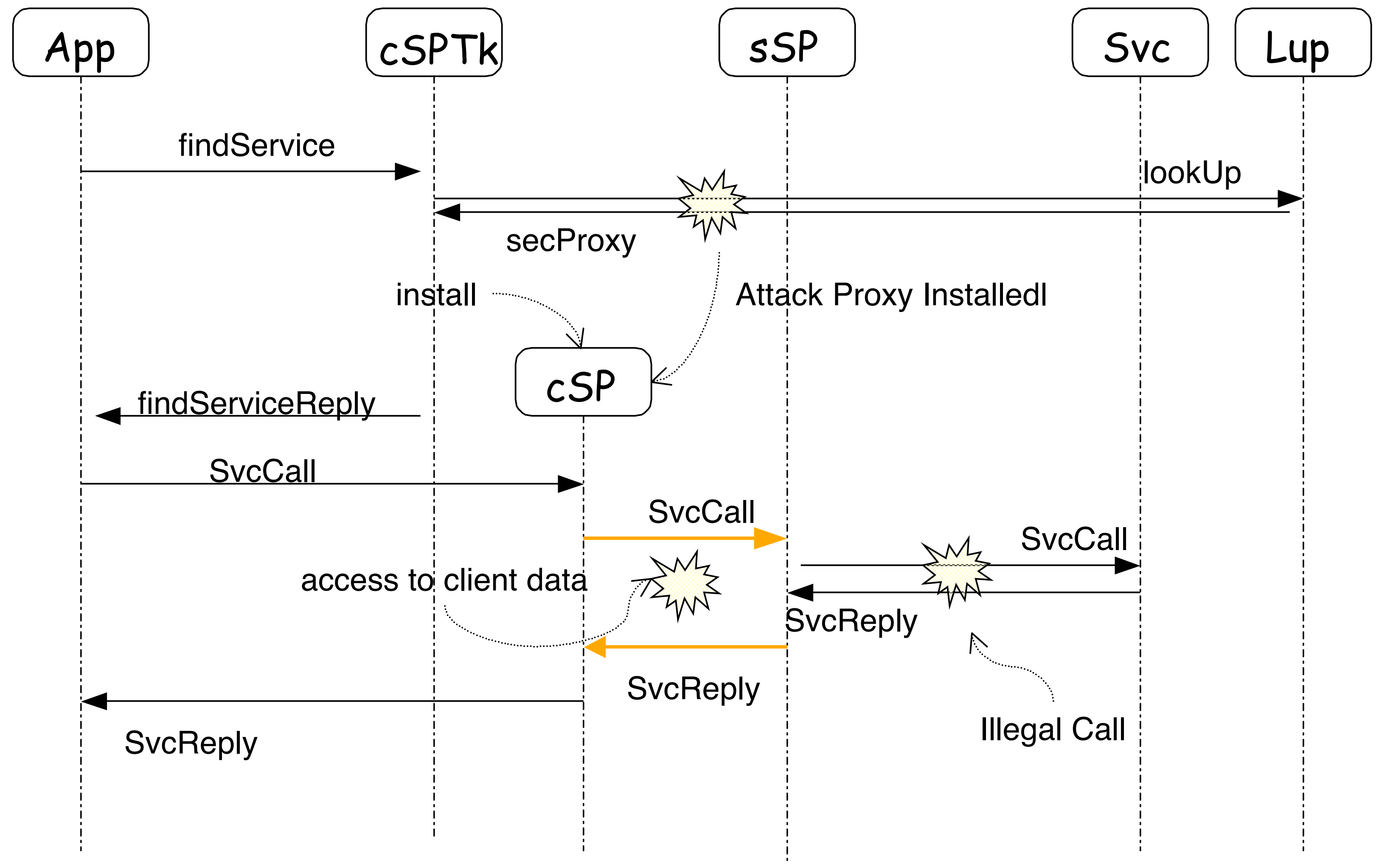


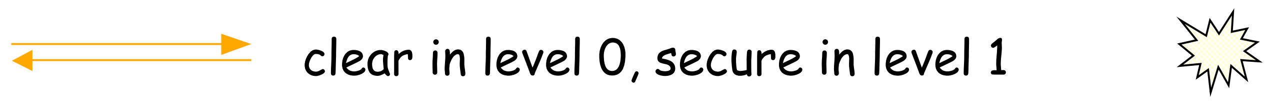
Levels 2,3



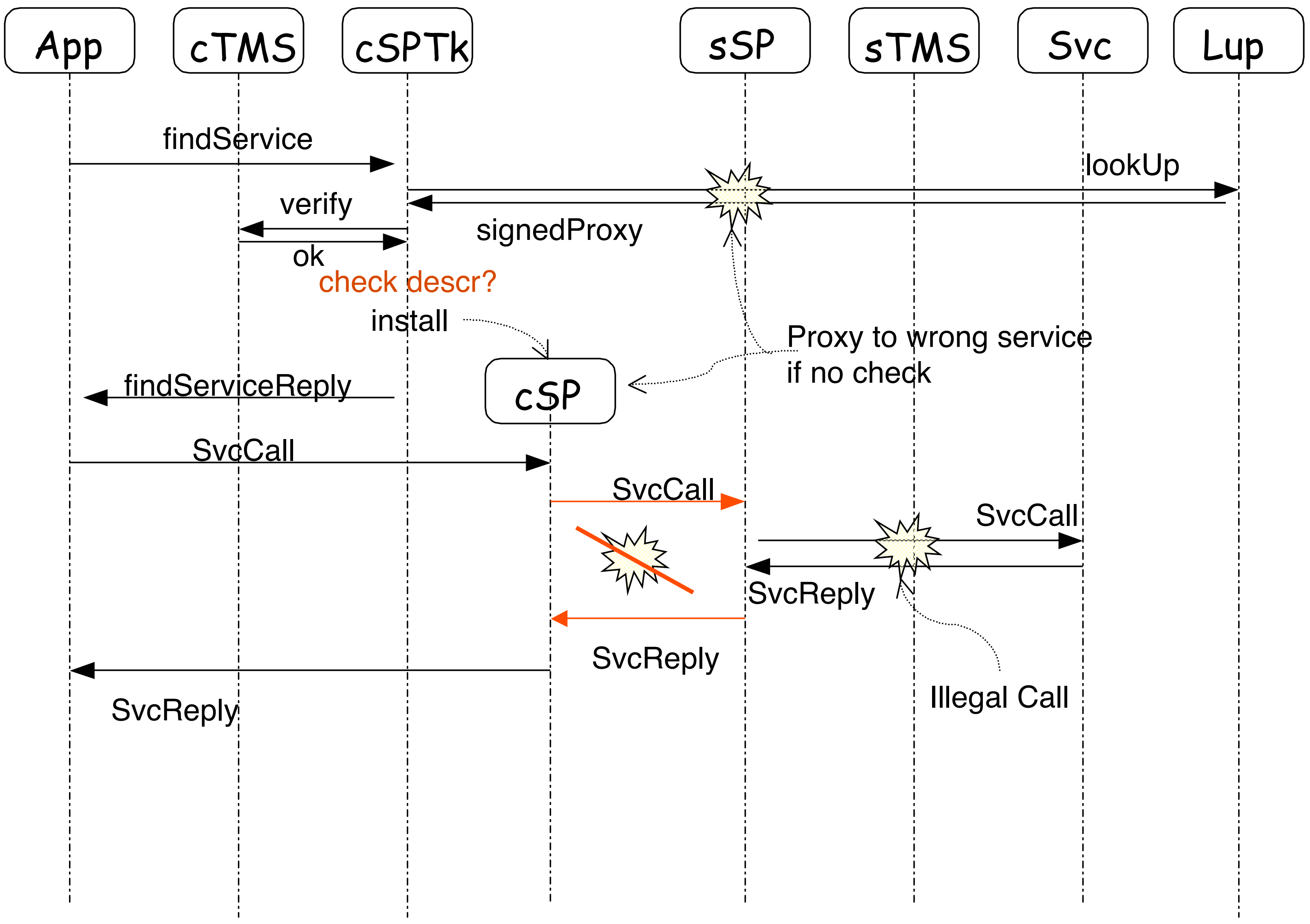
Levels differ in choice of proxy behavior

Scenario -- finding and using a service via Level 0,1 PTK

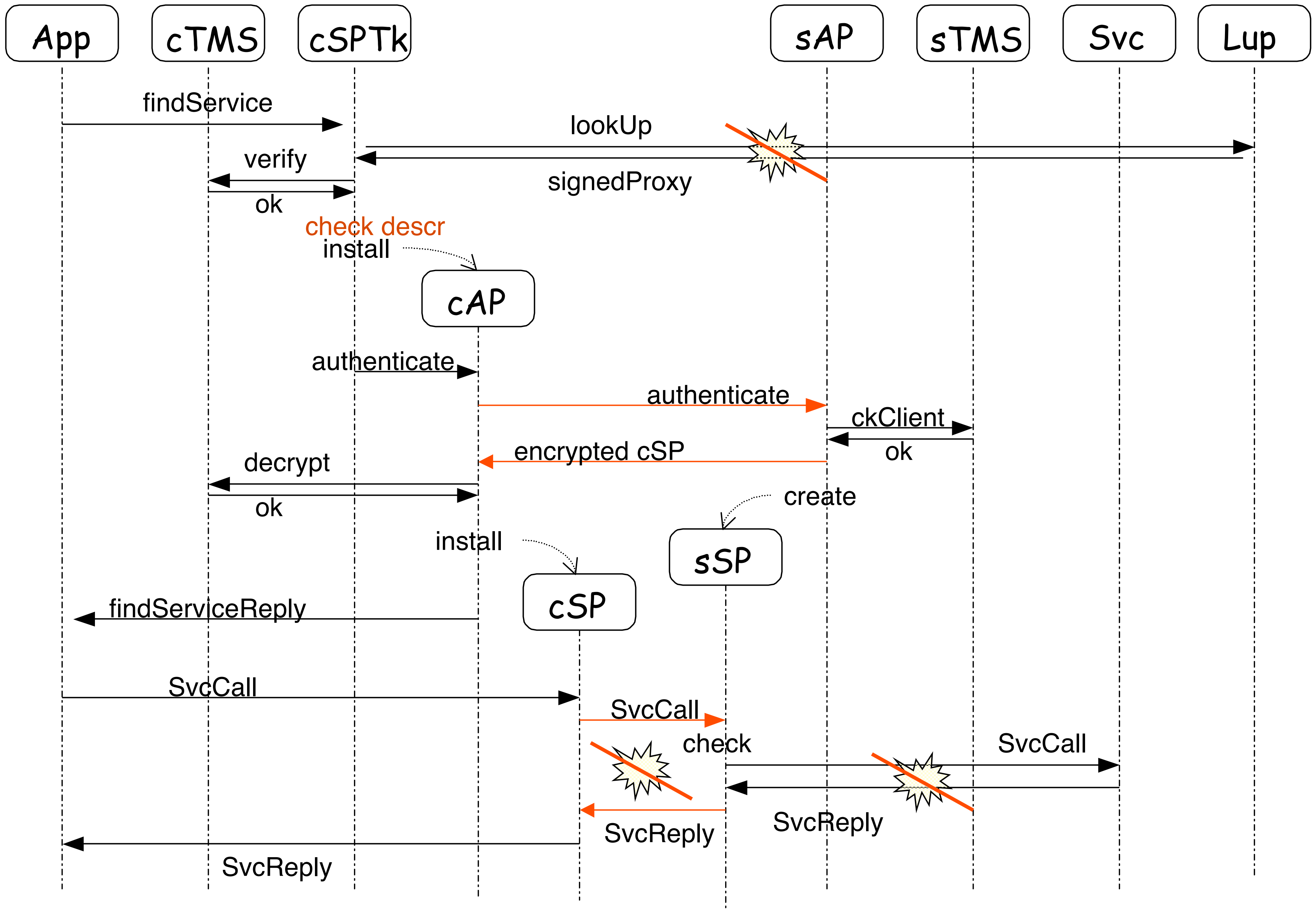




Scenario -- finding and using a service via Level 2 PTK



Scenario -- finding and using a service via Level 3 PTK



Analysis summary

- Define protoypical SPTK configuration
- Compose with attacker configuration
- Simulate -- execute with some scheduler
- Check security properties
  - Search state space

Properties for Ether Attack

1. Attacker in the ether + impersonator

Properties checked:

- 1.1 attacker see/modify client data
  - sent as service arguments or received as reply
- 1.2 client gets answer from wrong service
- 1.3 unauthorized service call succeeds
- 1.4 client impersonator succeeds

Ether attack search patterns

```
*** attacker gets private data sent in reply
search [1] (eacf ljcf sjcf cjcf) =>+
   ( cf:Configuration
    < eee : Attacker | atts:AttributeSet,
      clientDs(reply(d:Data, mycall,"sam") ds:DataSet)> ) .

*** Attacker impersonates (gets reply to request from "sam")
search [1] (eecf ljcf sjcf cjcf ajcf ) =>+
   ( cfx:Configuration < JA : JVM | jatts:AttributeSet,
     { cf:Configuration
       < JA . app : APP | catts:AttributeSet,  waitfor(sid:Oid) >
      msg(JA . app, sid:Oid, svcReply(reply(d:Data, dc:Data, sam"))) } > ) .
```

Summary of analyses

compromised ether

Property:	1.1	1.2	1.3	1.4
Level 0	+	+	+	+
Level 1	--	+	+	+
Level 2f	--	+	+	+
Level 2t	--	--	+	+
Level 3	--	--	--	--

Lookup Attack Properties

2. Attacker controls Lookup node

Properties checked:

- 2.1 client app can get proxy to requested service (sanity check)
- 2.2 client accepts proxy to attacker service
- 2.3 client accepts wrong server proxy
- 2.4 service integrity violated

Lookup Attack Search Patterns

```
*** client accepts proxy to wrong service at trusted server
search [1] icf-aa =>+
  ( cf:Configuration
   < JC : JVM | jcatts:AttributeSet,  { cocf:Configuration
      < O:Oid : cl:Cid | svc(S:Oid), oatts:AttributeSet >
     msg(JC . app, JC . csptk, findServiceOk(sn:String, O:Oid)) } >
   < JS : JVM | jsatts:AttributeSet,  {  socf:Configuration
     < S:Oid : spc:Cid |  sname(ssn:String), satts:AttributeSet > } >
   )
  such that (ssn:String =/= sn:String) .
```

Summary of analyses

compromised registry

Property:	2.1	2.2	2.3	2.4
Level 0	+	+	+	+
Level 1	+	+	+	+
Level 2f	+	--	+	+
Level 2t	+	--	--	+
Level 3	+	--	--	--

Conclusions

Value added:

- Documentation of SSPTK architecture
  - Modular, tunable security levels
- Formalization of some security goals
- Security hole closed

What more?

- Formalize simplifications
- Can we reduce arbitrary configurations to finite?