Continuing Work and Developments

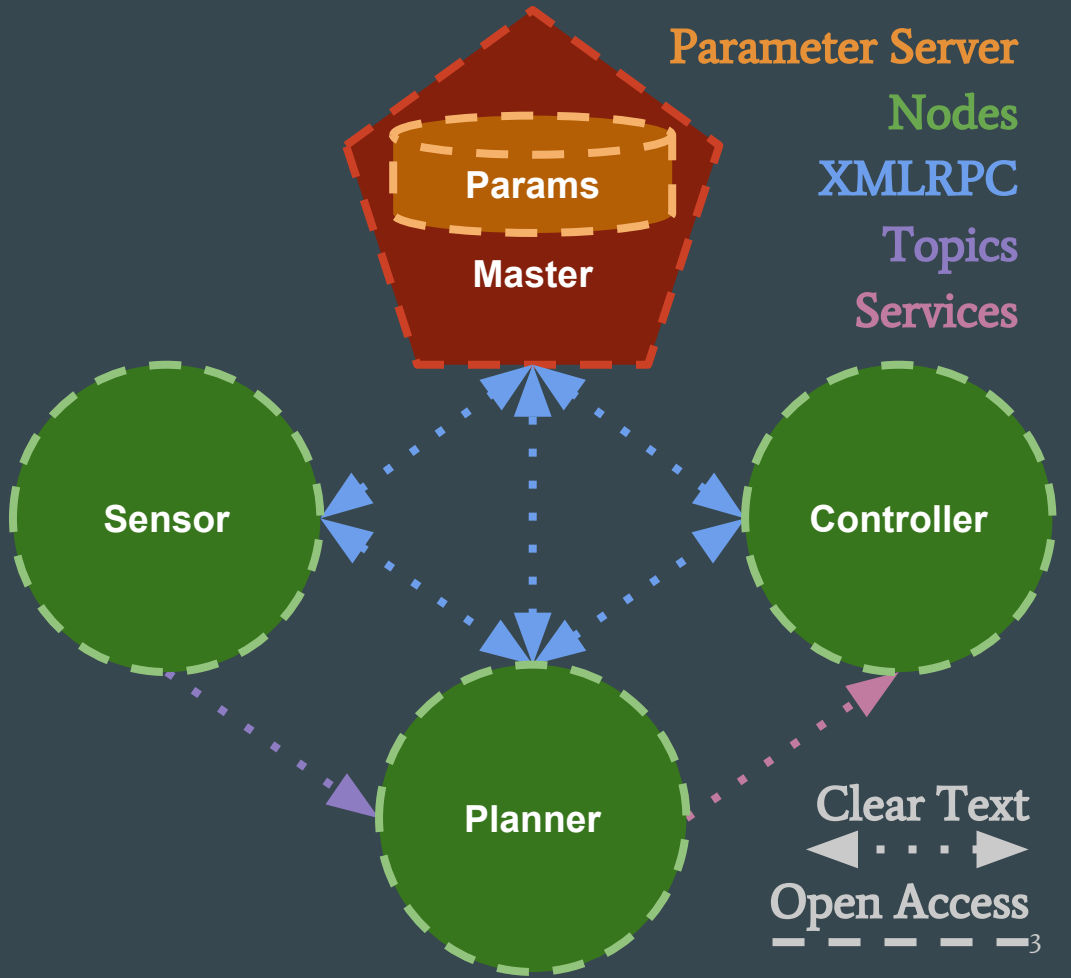
By Ruffin White and Gianluca Caiazza

Design Outline

- Harden ROS1 API
  - Validate API calls on client & server
  - Crosscheck using certificate policy extensions
  - Filter or redact response in accordance with policy
- Standardize Policy Profile Syntax
  - AppArmor-like Policy Profile Syntax for users
  - Intelligent permission models and alias
  - Simplify ROS1/ROS2 access definitions
- Integrate Policy Profile Autogeneration
  - Formal SROS logging formating and verbosity
  - Deliberate policy adjustments using SROS logs
  - User interaction through sros-genprof CLI

ROS1 Graph

Distributed Computation Graph
Communication is peer-to-peer
Master “Resolver” for pub/sub



ROS1 API

A collection of sub APIs
Each specific to a given role
Roles: Master vs Nodes

Parameter API:
setParam
getParam
hasParam
deleteParam

getParamNames
searchParam

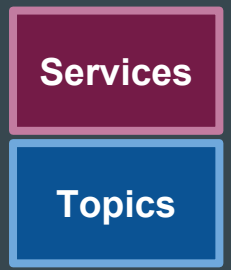
subscribeParam
unsubscribeParam

Slave API:
getBusStats
getBusInfo
getMasterUri
shutdown
getPid
getSubscriptions
getPublications
paramUpdate
publisherUpdate
requestTopic

Master API:
registerService
unregisterService
registerSubscriber
unregisterSubscriber
registerPublisher
unregisterPublisher

lookupNode
getPublishedTopics
getTopicTypes
getSystemState
getUri
lookupService

Namespace Specific:



What end of the API does a given role reside on?

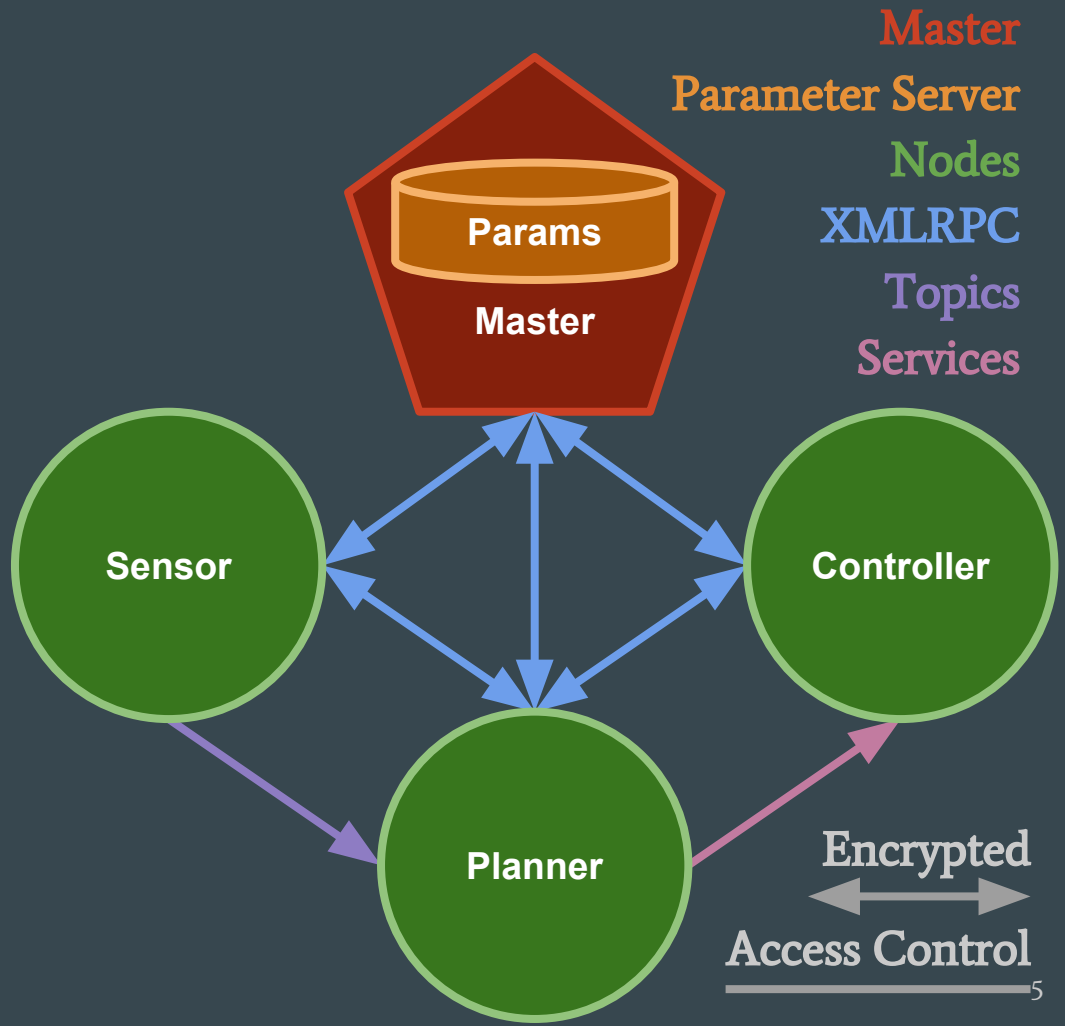
	Parameter API	Slave API	Master API
Master	Server	Client	Server
Nodes	Client	Server	Client

SROS1 Graph

All Traffic is Encrypted:
via. TLS and PKI

API has Access Control:
via. policy extensions in X.509

- **Client** check that **Server** is authorised to respond to the API call
- **Server** check that **Client** is authorised to request the API call

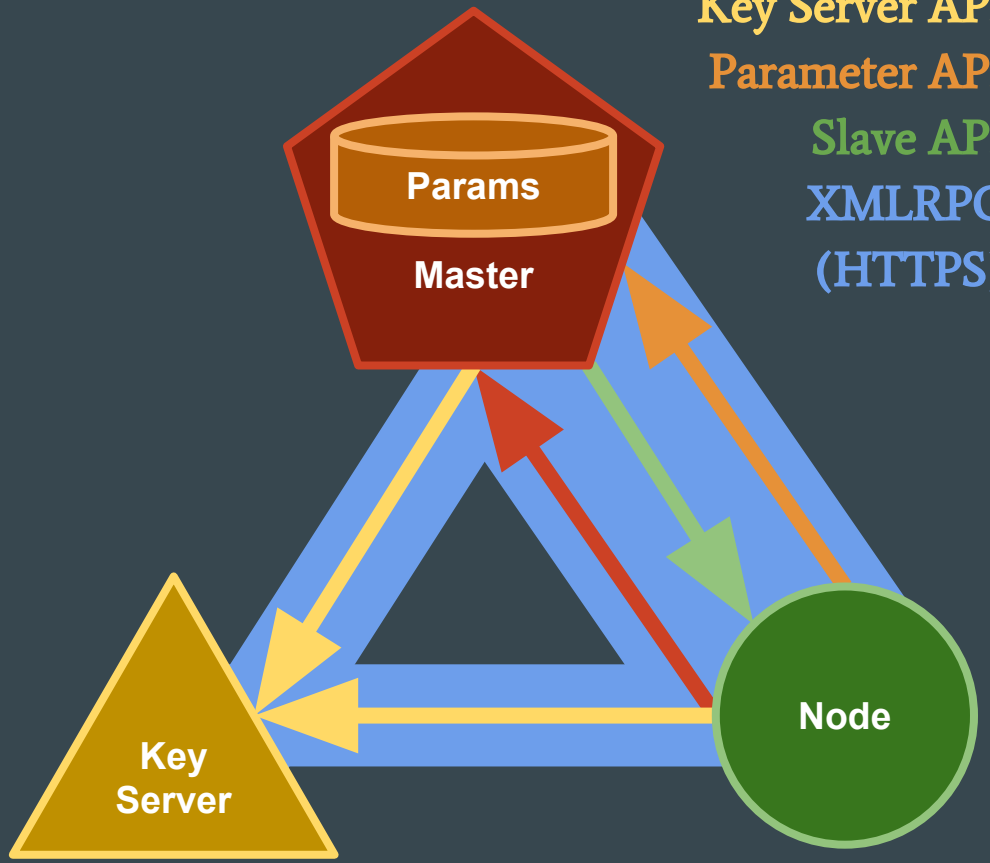


SROS1 API

Roles are enforced
Requests v.s. Responses
Strict asymmetric API access

✔ Client
✘ Server

✘ Client
✔ Server



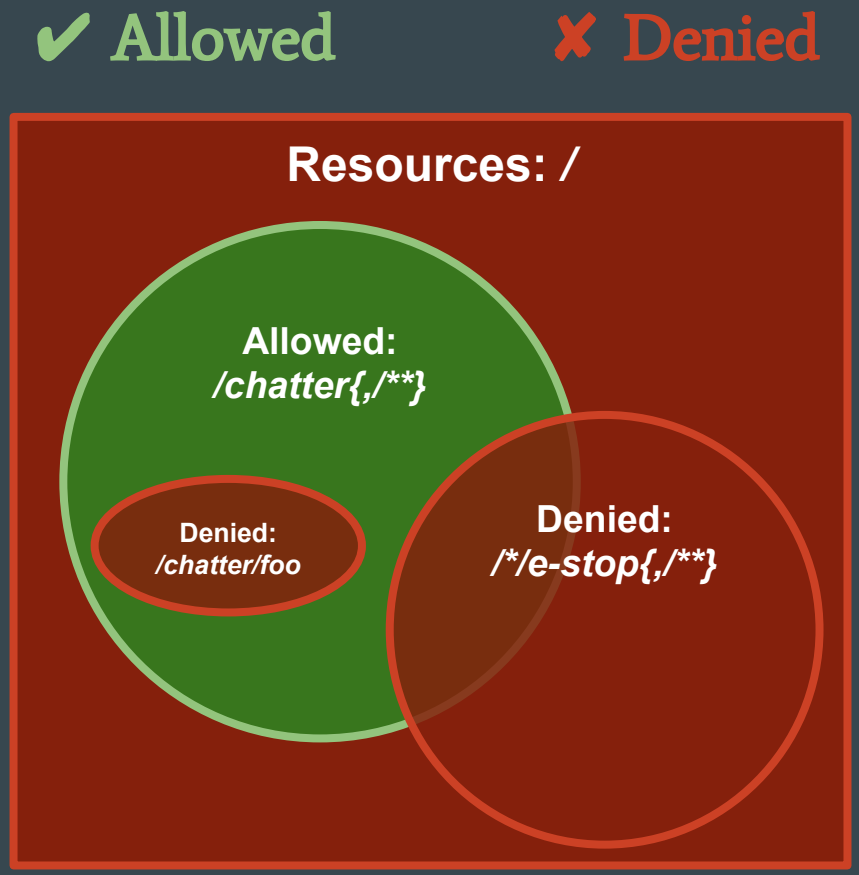
SROS1 Access Control

Using Mandatory Access Control (MAC),
Global security policy is: *deny by default*

Explicit permission to resource is required,
where adequate scope must be satisfied

Conflicts in allowed and denied scopes are
resolved by denying the intersection overlap

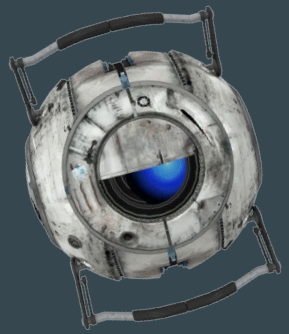
Path globbing is used to formulate a scope,
Like wildcards or regular expression, regex



SROS1 PKI

Using X.509 Certificates

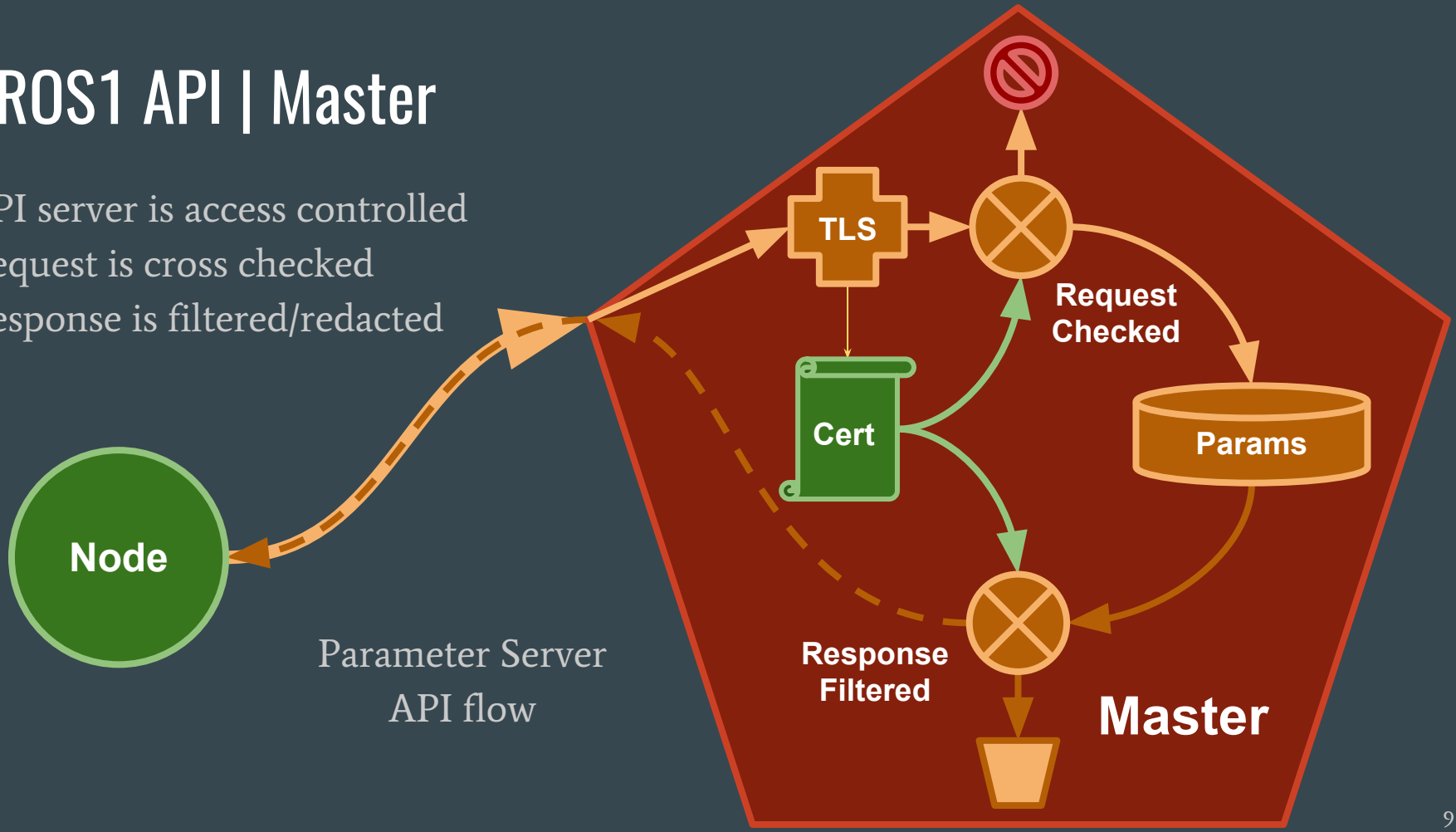
- Issued to each Node
- Signed by a trusted CA
- Embedded w/ Policy Exertions
- Access Control defined over
  - API Roles
    - Server
    - Client
  - Allow/Deny Resources
    - Parameters
    - Services
    - Topics



Subject name: */wheatley*
Subject Alternative Name: */wheatley{,/*}*
Issuer Name: Aperture Science CA
Validity period: Not Before->Not After
Subject Public Key: ...
...
X.509 V3 Extensions
Certificate Policies: *critical*
Policy: ~~Master~~ Slave API *Server* OID
Policy: Publishable *Topics* OID
CPS: */chatter{,/**}*
Policy: Denied Publishable *Topics* OID
CPS: */chatter/foo*
CPS: */*/e-stop{,/**}*
Policy: Executable *Services* OID
CPS: */wheatley/get_loggers*
CPS: */wheatley/set_logger_level*
Policy: Readable *Parameters* OID
CPS: */use_sim_time*
...

SROS1 API | Master

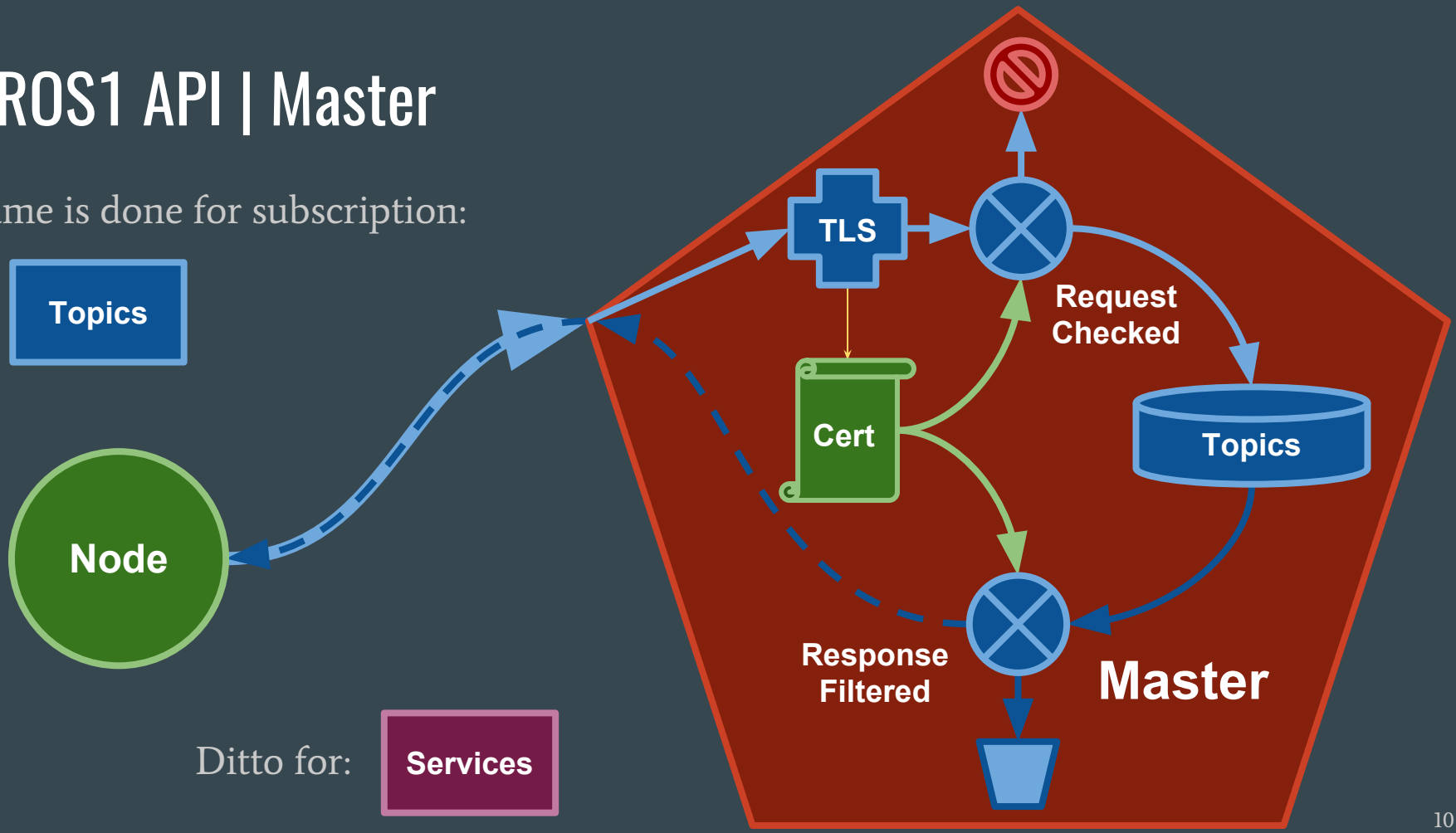
API server is access controlled
Request is cross checked
Response is filtered/redacted



Parameter Server
API flow

SROS1 API | Master

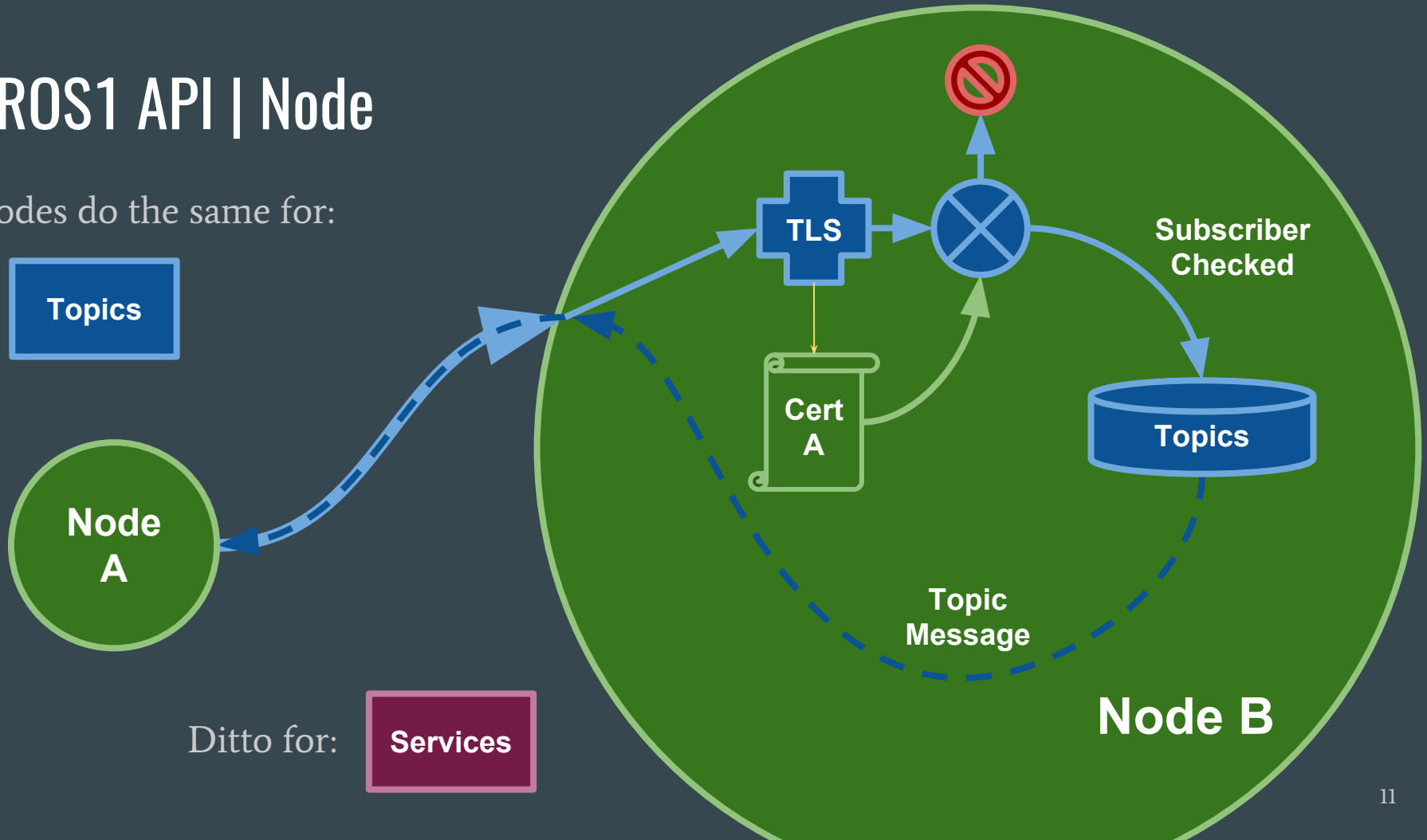
Same is done for subscription:



Ditto for: **Services**

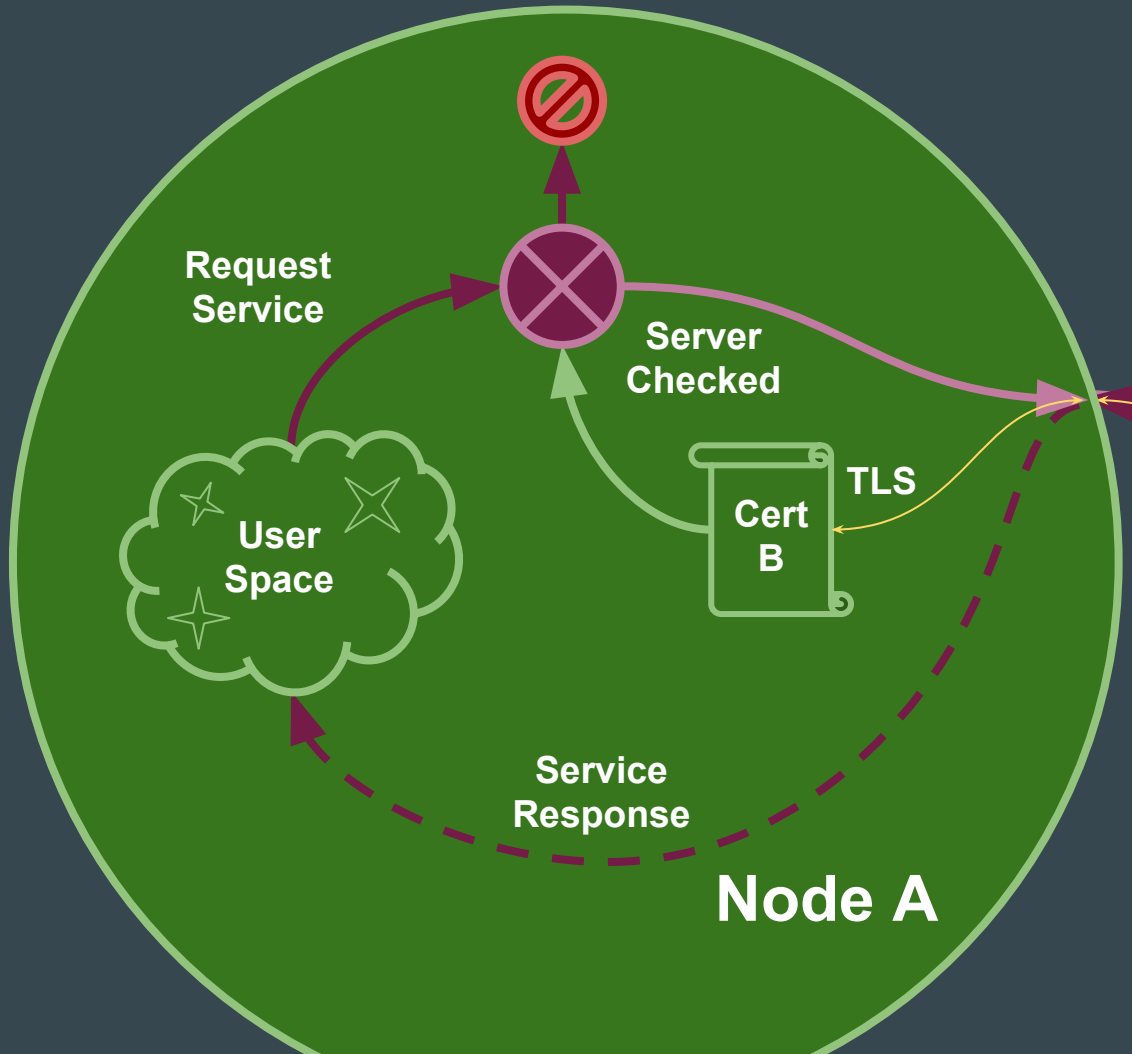
SROS1 API | Node

Nodes do the same for:



Ditto for: **Services**

SROS1 API | Node



Checks are done client side too, before any API is requested:

Services | Topics

Node B

Ditto for: Master × Any API Role

SROS1 API

Validation Terminology:

Legend Key	Description
Permission Required	API called must respect Permission type granted
Caller ID Matched	Caller ID must must respect Subject Alternative Name scope
Resource Checked	Received arguments must respect Allowed & Denied, resource scope
Response Sanitized	Returning responses must respect Allowed & Denied resource scope

Permission Required	Description
master	Is a master or rosmaster node
slave	Is a slave or regular node
read	Read a scope of parameters
write	Write to a scope of parameters
publish	Publish to a scope of topics
subscribe	Subscribe to a scope of topics
call	Call or request a service
execute	Execute or advertise a service

SROS1 | Parameter API

API access is contingent upon the call's intrinsics and if permissible by scope

Can only mutate scope that is writable, and see what is readable

*[TODO]: Unsure about union of two scopes? Doesn't work well with second point above, but unsure of all uses.

Parameter API	Permission Required	Caller ID Matched	Resource Checked	Response Sanitized
setParam	write	✔	✔	
deleteParam	write	✔	✔	
getParam	read	✔	✔	
hasParam	read	✔	✔	
getParamNames	read ∪ write*	✔	✔	✔
searchParam	read ∪ write*	✔	✔	✔
subscribeParam	read	✔	✔	
unsubscribeParam	read	✔	✔	

SROS1 | Slave API

Validation is simple

As only the Master may call the the Slave API

*slave-to-slave subscription

Slave API	Permission Required	Caller ID Matched	Resource Checked	Response Sanitized
getBusStats	master			
getBusInfo	master			
getMasterUri	master			
shutdown	master			
getPid	master			
getSubscriptions	master			
getPublications	master			
paramUpdate	master			
publisherUpdate	master			
*requestTopic	master slave	✔	✔	✔

SROS1 | Master API

API access is contingent upon the call's intrinsics and if permissible by scope

Can only mutate scope that is writable, and see what is readable

*Extra care in scoping

**Extra care in sanitizing

Master API	Permission Required	Caller ID Matched	Resource Checked	Response Sanitized
registerService	execute	✔	✔	
unregisterService	execute	✔	✔	
registerSubscriber	subscribe	✔	✔	
unregisterSubscriber	subscribe	✔	✔	
registerPublisher	publish	✔	✔	
unregisterPublisher	publish	✔	✔	
lookupNode	*pub ∪ sub	✔	✔	✔
getPublishedTopics	subscribe	✔	✔	✔
getTopicTypes	subscribe	✔	✔	✔
getSystemState	slave	✔	✔	**✔
getUri	slave	✔	✔	
lookupService	call	✔	✔	✔

SROS1 | Topics & Services

Transport connections checks opposite peer of the connection is permitted to proceed.

*[TODO]: Transports are not quite the same format as the rest of the API validation.

Topic Transport	Permission Required	Caller ID Matched	Resource Checked	Response Sanitized
connect_topic	publish	✔	✔	
accept_topic	subscribe	✔	✔	

Service Transport	Permission Required	Caller ID Matched	Resource Checked	Response Sanitized
connect_service	execute	✔	✔	
accept_service	call	✔	✔	

SROS1 API

In summary:

Server and Clients check peer's roles when requesting and responding to API calls.

API Calls are scrutinized via permissions & scopes, with responses sanitized as needed.

Topic and Service transport is scrutinized on Server and Clients side as well, with scope permissions considered in the connection.

SROS Policy Profile Syntax

Similar to that of Apparmor

- Supports MAC
  - Permissions are explicit
- Path Globbing
  - To define scopes
- Importing
  - #include rules for reuse
- Parsable format
  - Help autogenerate profiles
- Human readable
  - Auditing & debugging clarity

An Example **Apparmor** Policy Profile:

```
#include <tunables/global>
#include <tunables/ros>

/opt/ros/kinetic/bin/rosmaster {
  #include <ros/base>
  #include <ros/node>
  #include <ros/python>

  @{ROS_INSTALL_BIN}/rosmaster rix,
}

/opt/ros/kinetic/share/rospy_tutorials/001_talker_listener/listener.py {
  #include <ros/base>
  #include <ros/node>
  #include <ros/python>

  @{ROS_INSTALL_SHARE}/rospy_tutorials/001_talker_listener/listener.py r,
}

/opt/ros/kinetic/share/rospy_tutorials/001_talker_listener/talker.py {
  #include <ros/base>
  #include <ros/node>
  #include <ros/python>

  @{ROS_INSTALL_SHARE}/rospy_tutorials/001_talker_listener/talker.py r,
}
```

SROS Policy Profile Syntax

Profiles are applied to node *Namespaces*

Namespace matched nodes incur those *Profiles*

Profiles are composed of resource access *Rules*

Rules specify resource type, scope, role, and permissions the policy allows or denies

```
/namespace
{
  #include role
  resource /scope masks
}
```

Profile

```
/talker
{
  #include node
  topic /chatter p,
}
```

SROS Policy Profile Syntax

Resource types make a rule explicit to a specific resource

Scope defines the globbing namespace for the permission

Permissions are specified via masks, masks are also resource explicit

Deny is used to revoke permissions, superseding any applicable allow

An Example **SROS** Policy Profile:

Resource	Mask	Permission
Parameters	r	Read
	w	Write
Topics	s	Subscribe
	p	Publish
Services	c	Call
	x	Execute

```
/wheatley
{
  #include <ros/slave>
  param /use_sim_time r,
  topic /chatter{,/**} p,
  deny topic /chatter/foo p,
  deny topic /*/e-stop{,/**} p,
  service /wheatley/get_loggers x,
  service /wheatley/set_logger_level x,
}
```

API Role
master
slave

SROS Logging

Similar to that of Apparmor

- Security Events
  - Access attempts logged
- Logging Levels
  - Changing verbosity
- Parsable format
  - Help autogenerate profiles
- Human readable
  - Auditing & debugging clarity

An Example **Apparmor** Log:
(roslaunch failing to signal interrupt nodes)

```
Jan 25 12:31:27 dox kernel: [108436.948583] audit:
type=1400 audit(1485376287.948:83):
...
```

Context

```
apparmor="DENIED"
...
operation="signal" profile="ros/talker_listener_py"
pid=32701 comm="roslaunch" requested_mask="receive"
denied_mask="receive" signal=int peer="ros/roslaunch"

operation="signal" profile="ros/talker_listener_py"
pid=32702 comm="roslaunch" requested_mask="receive"
denied_mask="receive" signal=int peer="ros/roslaunch"

operation="signal" profile="ros/rosout"
pid=32627 comm="roslaunch" requested_mask="receive"
denied_mask="receive" signal=int peer="ros/roslaunch"

operation="signal" profile="ros/rosmaster"
pid=32627 comm="roslaunch" requested_mask="receive"
denied_mask="receive" signal=int peer="ros/roslaunch"
```

Info

SROS Logging

- Same logging format as ROS
  - Node
    - Logging node of origin
  - Verbosity
    - Access control severity
  - Datetime
    - `yyyy-MM-dd HH:mm:ss,fff`
  - String
    - Log message info

An Example **ROS** Log:

```
[rosmaster.main]  [INFO] 2017-01-25 18:47:43,225: initialization complete, waiting for shutdown
[rosmaster.main]  [INFO] 2017-01-25 18:47:43,225: Starting ROS Master Node
[xmlrpc]          [INFO] 2017-01-25 18:47:43,226: XML-RPC server binding to 0.0.0.0:11311
[xmlrpc]          [INFO] 2017-01-25 18:47:43,226: Started XML-RPC server [http://GLaDOS:11311/]
[xmlrpc]          [INFO] 2017-01-25 18:47:43,227: xml rpc node: starting XML-RPC server
[rosmaster.master][INFO] 2017-01-25 18:47:43,227: Master initialized: port[11311], uri[http://GLaDOS:11311/]
[rosmaster.master][INFO] 2017-01-25 18:47:43,285: +PARAM [/run_id] by /roslaunch
```

SROS Logging

For profiling, debugging policies and autogeneration

Compatible format for working with existing tools
*[TODO]: message syntax

Verbosity Level	Message Purpose
INFO	Mode Status
DEBUG	"AUDIT"
WARN	"COMPLAIN"
ERR	"DENIED"

An Example **SROS** Log:
(wheatley failing to register as publisher)

```
[rosmaster.master][INFO]   2017-12-31 12:34:56,789: sros="STATUS" operation="runtime_mode" mode="audit"

[rosmaster.master][DEBUG] 2017-12-31 12:34:56,795: sros="AUDIT" operation="registerPublisher" node="/wheatley" resource="topic" path="/chatter"                                                .

[rosmaster.master][DEBUG] 2017-12-31 12:34:56,850: sros="AUDIT" operation="registerService" node="/wheatley"  resource="service" path="/wheatley/get_loggers"

[rosmaster.master][DEBUG] 2017-12-31 12:34:56,880: sros="AUDIT" operation="registerService" node="/wheatley" resource="service" path="/wheatley/set_logger"

[rosmaster.master][WARN] 2017-12-31 12:38:57,789: sros="COMPLAIN" operation="getParam" node="/wheatley" resource="parameter" path="/use_sim_time"

[rosmaster.master][ERR]   2017-12-31 12:34:57,839: sros="DENIED" operation="registerPublisher" node="/wheatley" resource="topic" path="/chatter/foo"
```

Profile Autogeneration

Similar to that of Apparmor

- Log Auditing
  - Runtime generates events
- Demonstration Learning
  - Events are extracted from logs
- Command Line Interface
  - Help profile events & policies
- Debugging readable
  - CLI suggests policy modifications

Example **Apparmor** CLI:
(debugging roslaunch with aa-logprof)

```
$ sudo aa-logprof
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:

Profile:     ros/rosmaster
Access mode: receive
Signal:      int
Peer:        ros/roslaunch

 [1 - signal receive set=int peer=ros/roslaunch,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
Adding signal receive set=int peer=ros/roslaunch, to profile.

...

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - ros/rosout]
  2 - ros/talker_listener_py
  3 - ros/rosmaster
(S)ave Changes / Save Selec(t)ed Profile
[(V)iew Changes] View Changes b/w / (C)lean profiles / Abo(r)t
Writing updated profile for ros/rosmaster.
Writing updated profile for ros/rosout.
Writing updated profile for ros/talker_listener_py.
```

Profile Autogeneration

Workflow:

1. An empty profile is loaded
2. Profile is set to complain mode
3. ROS app is put through its paces
4. SROS violations are logged
5. Users then runs logprof
6. Tools suggests policy amendments
7. Users audits using a CLI dialogue
8. New policy saved, old config cleaned
9. Repeat steps 3-8 until satisfied
10. Finaly profile is set to enforce mode

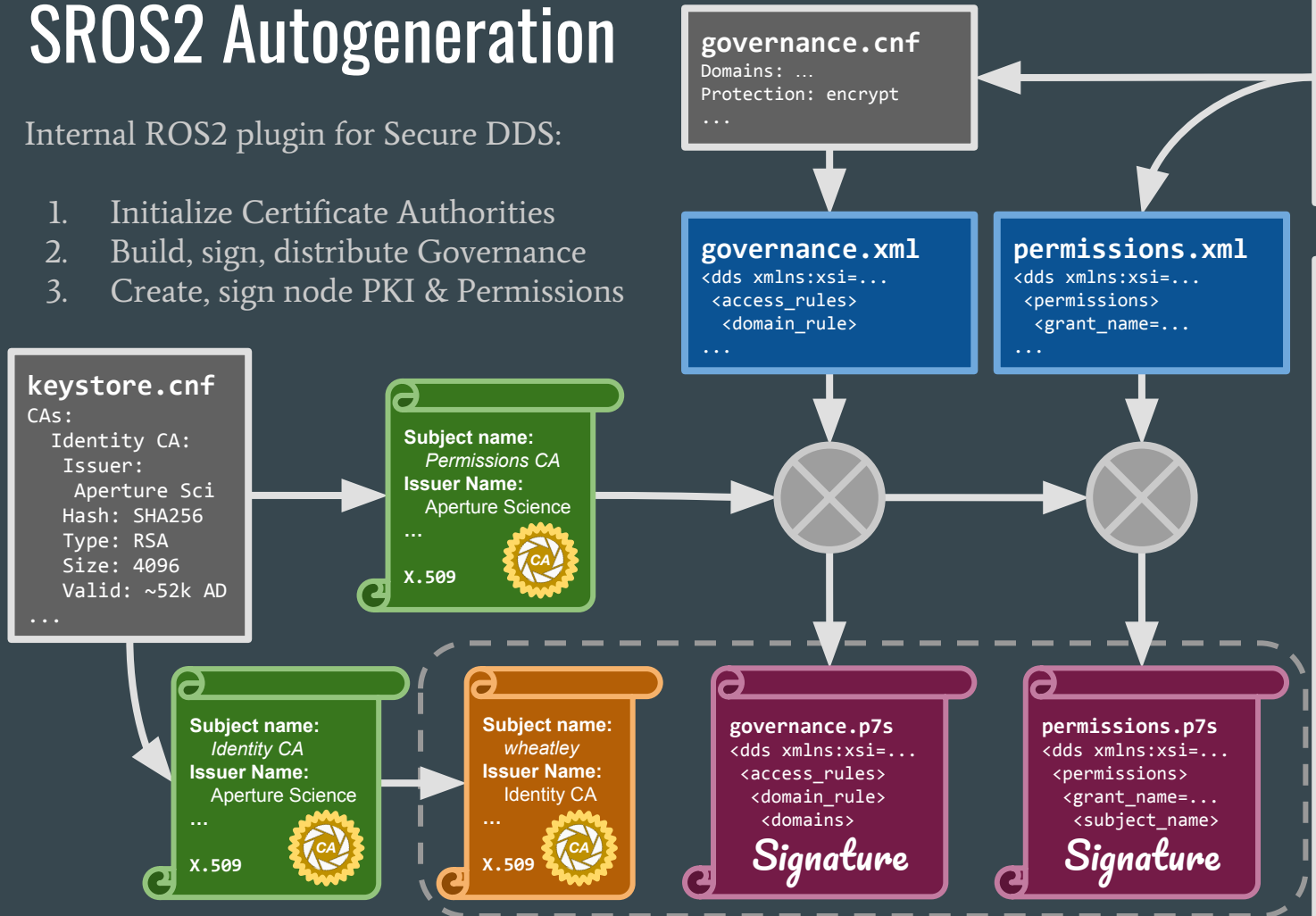
Proposed **SROS** CLI:
(debugging a ROS node with logprof)

```
$ sros-logprof
Reading log entries from /home/user/.ros/log/
Updating SROS profiles in /home/user/.ros/sros.d.
Complain-mode changes:

Profile:     ros/wheatley
Access mode: publish
Topic:       /chatter/foo


 [1 - topic /chatter/foo p,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
Adding topic /chatter/foo p, to profile.

...

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - ros/wheatley]
  2 - ros/listener
  3 - ros/rosmaster
(S)ave Changes / Save Selec(t)ed Profile
[(V)iew Changes] View Changes b/w / (C)lean profiles / Abo(r)t
Writing updated profile for ros/listener.
Writing updated profile for ros/rosmaster.
Writing updated profile for ros/wheatley.
```

SROS2 Autogeneration

Internal ROS2 plugin for Secure DDS:

1. Initialize Certificate Authorities
2. Build, sign, distribute Governance
3. Create, sign node PKI & Permissions



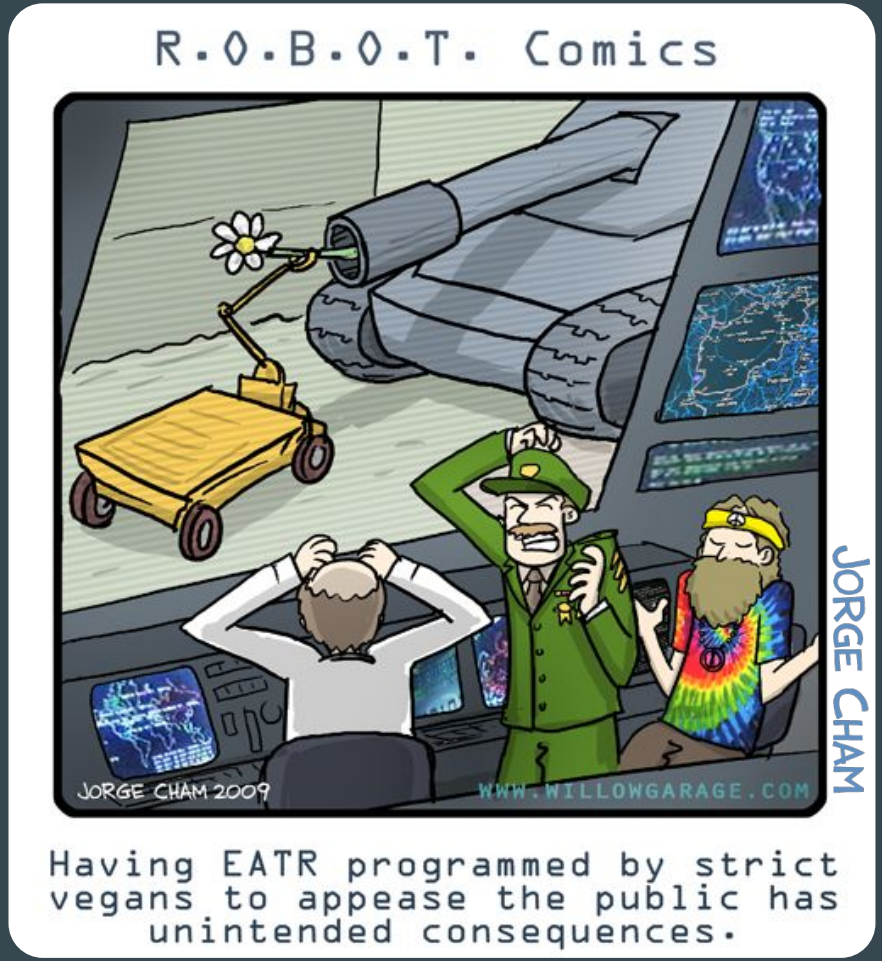

```
$ ros2 secure keystore auto
      /namespace/here/wheatley

$ tree keystore_root/
keystore_root/
├── governance.cnf
├── keystore.cnf
├── permissions.cnf
├── participants
│   └── namespace
│       └── here
│           └── wheatley
│               ├── cert.pem
│               ├── key.pem
│               ├── permissions.p7s
│               └── permissions.xml
├── private
│   ├── identity.key.pem
│   ├── governance.xml
│   └── permissions.key.pem
├── public
│   ├── identity.cert.pem
│   └── permissions.cert.pem
└── shared
    └── governance.p7s
```

Conclusion

Presented design affirms SROS's objective to secure transport and application layers

Remain agnostic to transport or release to benefit all platforms from shared tooling

Promote high level interfaces and plugins to simplify use, thus encouraging adoption



Support

CONTEXTUAL ROBOTICS INSTITUTE

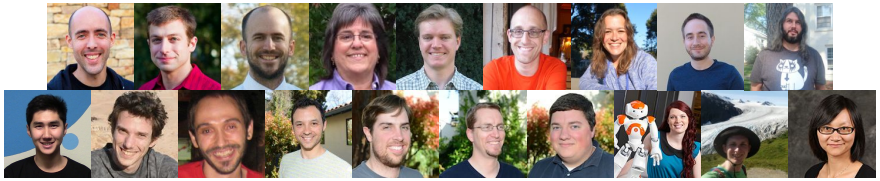
UC San Diego

SEE DO THINK

Cognitive Robotics

"...to advance contextual robotics through relevant grand challenge research, to educate and train students who are prepared to catalyze future developments in robotics; and to provide the talent and innovation to establish San Diego as a leading robotics hub."

Open Source Robotics Foundation



"...to support the development, distribution, and adoption of open source software for use in robotics research, education, and product development."

VENETIARVM VNIVERSITAS · IN DOMO FOSCARI · PAX TIBI MARCE EVANGELISTA MEUS

Università Ca'Foscari Venezia

Advances in Autonomous, Distributed and Pervasive systems

Resources

SROS1 Documentation:

- wiki.ros.org/SROS

SROS2 Tickets:

- Access Control Policy Format
  - github.com/ros2/design/issues/140
- Keystore Proposal
  - github.com/ros2/sros2/issues/21
- Security Event Logging
  - github.com/ros2/design/issues/150

SROS Publications:

White, R., Caiazza, G., Christensen, H., Cortesi, A., (2017) *SROS1: Securing ROS over the wire, in the graph, and through the kernel*. Manuscript submitted for publication.

More about:

Ruffin: about.me/ruffin

Gianluca: about.me/caiazza