



# Tango

Software development and control systems integration services

www.s2innovation.com



# **Client-Server** Authentication for



# Who we are?

## 2017

S2Innovation was founded in December 2017 by 2 employees of synchrotron SOLARIS:

- Piotr Goryl, Head of IT and Controls,
- Wojciech Soroka, Procurement officer.

## Slide 2



## SOLARIS NATIONAL SYNCHROTRON RADIATION CENTRE



# Main clients:



## and European Spallation Source (ESS)







# **Client-Server Authentication for Tango**

Agenda:

- 1. Client-Server Authentication for Tango Control System Use Case.
- 2. Tango Access Control (TAC).
- 3. How is it solved in other facilities?
- 4. Client-Server Authentication proposition.
- 5. Q&A.







The **goal** of the presentation is to trigger discussion about the improvement of security in Tango Controls.

Client-Server Authentication for Tango Control System

Example Use Case: Users on one beamline can, by mistake, access devices on another beamline and cause some damage.





# **Client-Server Authentication in Tango** Using Tango Access Control (TAC)

Tango Access Control (TAC) is a built-in authorization mechanism within the Tango Controls framework.

It allows device servers to accept or reject client requests based on detailed identity information transmitted from the client with every request.

TAC is implemented through:

- IDL-based method extensions, which accept identity information,
- 2. a server-side access control mechanism,
- 3. and structured identity data known as *ClientIdent*.

## Slide

6

```
CFP_6: IBL6 data structure for C++ clients
          JAVA 5: IDL6 date structure for Java clients
                    switch (LockerLange)
     // pre-IDL6 data structure for C++ client
                   TIDEST CRO.GLAT:
     // pre-1016 data structure for lava clients
              Jave Cloticent java clot;
     // IDL6 data structure for D++ clients
                   1010-21 6 CC0 CUPT 6
          16 data structure for Java clients
                     ident_6 java_cint_s;
212
213
221 enum Attr@uality
22
             ATTR_VALID
              ATTR INVALUE
              STTP II ARE
             ATTR_CHANGING
             ATTR MARNENS
                                                       5
     ervus AttrWriteType
231
             RPAT:
233
             READ_WITH_WRITE,
             WHITE.
             READ WRITE
         BT_UNKNDNN
237 1:
    prum AttrBataFormat
```





# Tango Access Control (TAC) -**Client Responsibilities**

Clients are responsible for **generating** and **sending** <u>identity information</u> with every request. This information is structured and sent via the *ClientIdent* object.

*ClientIdent* Structure

Defined in *tango.idl* as a CORBA union to support multiple client

1/ 10	C trace context version 6
CRSR	83C_TC_V0:
	WSCFraceDostextVB data;
h:	
11	
// Ab	out Cintident & trace context propagation (obervability service)
11	
// Th	e simplest way to add support for obervability is to extend the existing
// =E	intident' (i.e., client identification) struct. The main advantage of
// m	is approach is to limit the modifications of the Tango IOL to the Cintident
11.85	ruct itself while maintaining the rest of the COMSA interface unchanged.
11 10	reover, since the trace context is propagated by the client (i.e., the
// ca	conty, it makes parks to inject it this the contident struct.
11	
11	
11	
11	Data types for client identification
11	
11	
1.1.	
typed	ef unsigned lang long
typed	of unsigned long Copulations
struc	t JavaCintIdest
(	(1000)
	string MeinClass;
	Javaeein ooid;
87 C	
11.	
11 1	Teleford extension for TRL A
11.	anistrand extension for the e
stear	Trout of I dear
1	· SSAMATECHA ···
	// the data structure introduced in IOL 4 for C++ clients identification
	robelet(dest em.elst;
	// the WSC trace context headers (obervability service)
	TraceContext trace_context;
32	Contraction of the second s
100	
11+++	
// Ja	veCintiBent extension git for IDL 6
11	
struc	t JavaEintIdent_6
1	
	// the data structure introduced in IOL 4 for Java clients identification
	JavaEintIdent java_cint;

Slide

7



# Tango Access Control (TAC) -**Server Responsibilities**

On the server side, each device server must:

- Receive and deserialize the *ClientIdent* data.
- 2. Use the AccessControlList (ACL) system to decide if a request is allowed.
- 3. Allow or deny based on client attributes:
  - IP address,
  - MainClass,
  - UUID/session ID,
  - command or attribute being accessed.



Slide

8

Chefits are responsible for generating and sending identity information with every request. This information is structured and sent via the clientident object.



# Tango Access Control (TAC) -Server Responsibilities

If **unauthorized**, the server responds with a CORBA exception:

exception DevFailed {
 DevErrorList errors;
};



chemis are responsible for generating and sending identity information with every request. This information is structured and sent via the clientident object.



# Tango Access Control (TAC) -Method Flow

Sequence of events:

- 1. Client Application creates a *ClientIdent* with MainClass and a unique session UUID.
- 2. DeviceProxy sends a command using a method like command\_inout\_4() including the identity.
- 3. The server receives the request and invokes *is\_allowed()* internally.
- 4. If allowed, the command executes.
  - If denied, a *DevFailed* exception is returned.;

```
546 /++
147 # Bet connerd history buffer
     * Return command result history for polled command
     Operan commend ascil string s.g. "On"
     goaran o The history depth
     Greturn connend history.
             DevCedHistory_4 command_insut_history_4(in string command,
                                                       in long n) raises (DevFailed);
     · Execute a conmand on a device synchronously with
      * one input parameter and one output parameter
      paran command assii string e.g. "On"
     Roaran argin command input parameter e.g. float
      param source The data source. Used to specify if the command result must be
     read from the polling cache buffer or from the device itself
     @param cl_ident The client identificator
     Breturn connand result.
     ww.
             any command_input_4(in string command.
                                 in any argin.
                                 in DevSource source.
                                                    t_ident) raises(DevFailed);
                                     in Cintident
     Sumd a variable list of attributes from a device
                                                                                                              D
     doaran mane list of attribute manes to read
     Boaran source The data source. Used to specify if the command result must be
    read from the polling rache buffer or from the device itself
    Greturn list of attribute values read
177
     441
             AttributeValueList_4 read_attributes_4(in DevVarGtringArray names.
                                                    in BevSource source, in Clittident sl_ident) raises(SevFailed);
101
162
    · write a variable list of attributes to a device
    Operan values list of attribute values to write
185
164 Greture nothing
165
     ++/
186
             void write_attributes_4(in AttributeYaluoList_4 values,in C<mark>ITT</mark>dent cl_ident) raises(DevFailed,MultiDevFailed)
187
188
100
     * set the configuration for a variable list of attributes from the device
    Operan new, conf list of attribute configuration to be set
191 Greture nathing
192
     ++1
             void set_attribute_config_4(in AttributeConfigList_3 new_conf.in CC+Pident cl_ident) raises(DevFailed);
```





# **Client-Server Authentication for Tango** Example Use Case

Users on one beamline can, by mistake, access devices on another beamline and cause some damage.

ESRF – is using TAC Max IV – every beamline has its own sub-network

How it is solved in you facility?

- 1. Soleil?
- 2. ALBA?
- 3. Elettra?
- 4. Solaris?
- 5. SKAO?







# **Client-Server Authentication for Tango** Proposition of the solution

All Tango messages exchanged over the network include a digital signature which authenticates the client performing given operation and contents of the message.

The **key pair** used for signing messages can be generated per host, user or application.

This can be configured e.g. in */etc/tangorc*, environment variables or on command line.



# Client-Server Authentication for Tango Proposition of the solution – Client Side

The private key resides securely on the client.

- It **is used to sign** the <u>message/command</u> before sending.
- This key should never be shared.

ICIUIEU AIIU SEIIL VIA LITE ClientIdent ODJECL.





# Client-Server Authentication for Tango Proposition of the solution – Device Server Side

The **public key** corresponding to the client's private key is available on or retrievable by the **Device Server**.

• It is used to **verify** the digital signature in the message.

JCIUIEU AIIU SEIIL VIA LITE ClientIdent UDJECI.







# **Client-Server Authentication for Tango Benefits and Strengths**

Cryptographically Strong Authentication

- 1. Current TAC is trust-based, relying on environment variables like <u>username</u> and IP, which can be **spoofed** in untrusted environments.
- The proposal uses digital signatures, which:
  - Cannot be fake without the **private key**.  $\bullet$
  - Are **verifiable by the server** using the sender's public key.  $\bullet$
  - Ensure non-repudiation, i.e. the client cannot deny sending the message.  $\bullet$







# **Client-Server Authentication for Tango** Challenges and Considerations

- 1. Where and how are private keys stored and protected?
  - Must avoid leaking keys (e.g. from /etc/tangorc).  $\bullet$
  - Might require keyring or encrypted storage with optional passphrases.  $\bullet$
- 2. Public key distribution:
  - How does the server obtain and trust the client's public key?  $\bullet$
  - Consider storing verified public keys in the Tango database (alongside  $\bullet$ the policy).
  - Optional support for X.509 certificates or key signing may improve trust  $\bullet$ management.



Chefits are responsible for generating and sending identity information with every request. This information is structured and sent via the clientident object.



# Client-Server Authentication for Tango Comparison with Existing TAC

## Feature

Authentication method

**Enforcement location** 

Trust model

Granularity

Message integrity

Mutual authentication

**Existing TAC** 

OS user + IP

Client-side only

Trust OS/host reporting

System/user/device

Not protected

Not supported

## Slide 17

## **Proposed Solution**

Public/private key pair (digital signature)

Server-side (on device server)

Cryptographic verification Same + more options (command, attribute level)

Ensured via digital signatures

Supported



# **Client-Server Authentication for Tango** Proposition of the solution

**Please** provide your feedback on GitLab:









Lukasz Zytniak – COO of S2Innovation Email: lukasz.zytniak@s2innovation.com Mobile: (+48) 789 339 875







# **Client-Server Authentication for Tango** Proposition of the solution

Every Tango message exchanged over the network is digitally signed by the sender using an asymmetric key pair.

**Device servers** verify the client's signature using its **public key**, and clients can optionally verify the server's responses similarly.

- Client identity:
  - Based on a public key instead of OS username/IP.
- Policy enforcement:
  - Defined at the server side using access control rules stored in the Tango database (or equivalent).
- Configurable granularity:
  - Rules may apply to system, server, device, command, or attribute level.

Slide XX



# **Client-Server Authentication for Tango** Proposition of the solution – Optional (for mutual authentication)

If the Device Server signs its responses:

- The Device Server has its own private key, and
- The Client must have access to the server's public key to verify.

ICIUIEU AITU SEIIL VIA LITE ClientIdent UDJECI.

