Designing the future of agent-server communication in RUDDER

Alexis Mousset - amo@rudder.io

February 4, 2020

CfgMgmtCamp Ghent 2020

Context

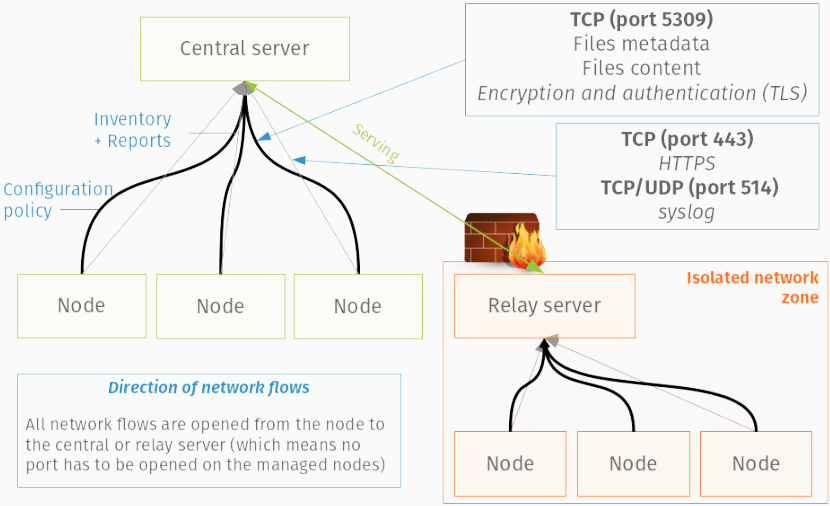
Needs and requirements

Design choices

Implementation

Perspectives

Context


Central server
TCP (port 5309)
Files metadata
Files content
Encryption and authentication (TLS)
Inventory
+ Reports
Serving
Configuration
policy
TCP (port 443)
HTTPS
TCP/UDP (port 514)
syslog
Node
Node
Node
Relay server
Isolated network
zone
Direction of network flows
All network flows are opened from the node to the central or relay server (which means no port has to be opened on the managed nodes)
Node
Node
Node

Agent-server communication

- HTTP PUT for inventories
- Custom protocol in TLS for policy copy
- Standard syslog for reporting

```
R: @@osquery_installation_and_configuration
@@result_success
@@32377fd7-02fd-43d0-aab7-28460a91347b
@@de4435a0-b5db-401a-aba1-a685d8a937e1
@@0
@@File from HTTP server
@@/etc/pki/RPM-GPG-KEY-osquery
@@2020-01-31 22:16:00+00:00
##2f4e4400-3206-4fb3-ae7f-16d1192e38ac
@#File /etc/pki/RPM-GPG-KEY-osquery is correct
```

Reporting

```
R: @@Techniques@@Type@@RuleId@@DirectiveId
@@VersionId@@Component@@Key@@ExecutionTimeStamp
##NodeId@#HumanReadableMessage
```

- Produced by the policies (a lot of work actually!)
- One log in syslog for each component
- Special cases for the first and last report
- The webapp only processes them when last report is received
- Parsed using `awk` for `rudder agent run output`

Needs and requirements

Reporting - Current limitations

- Reporting security
  - plain text
  - no authentication (easy to fake)
- Missing information in reports
  - Differences between expected and current state
  - Information about what has been repaired exactly
- Syslog itself
  - Requires a specific port (514)
  - Requires root access (to configure local syslog daemon)
  - Can interact with user's syslog configuration
  - Hard to debug (not much logs about syslog daemon by default)
  - Poor performance (for database insertion)

Constraints

- Smooth transition
  - Keep compatibility with both reporting modes for several versions
  - Allow switching at any time (same data model)
- *Keep It Simple*
  - Debuggable
  - Low operation overhead
  - Use well-known technologies
- Security
  - State-of-the-art security for reporting protocol
  - Allow future homogenization
  - Focus on security for the implementation itself

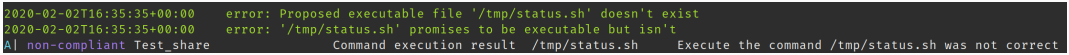
Design choices

Report → Run log

- The stream of reports is useless
- Better transmitted as a single run log
- Store information by run (in a simple file)
- Easier to manage and allows lots of improvements
  - Compression (works well!)
  - Database transaction by run
- A run log is identified by: a node id, a config id and a date+time

Improve run log

- We are missing a lot of valuable information
- ”Hidden” in agent logs (`rudder agent run -i`)
- Need to `ssh` to understand anything
- We want to capture and contextualize them


2020-02-02T16:35:35+00:00 error: Proposed executable file '/tmp/status.sh' doesn't exist
2020-02-02T16:35:35+00:00 error: '/tmp/status.sh' promises to be executable but isn't
A| non-compliant Test_share Command execution result /tmp/status.sh Execute the command /tmp/status.sh was not correct

Improve run log

The problem is that we have two (isolated) information streams:

- Reports from inside of the policies
- Agent logs (errors, executed commands, various outputs, etc)

Improve run log

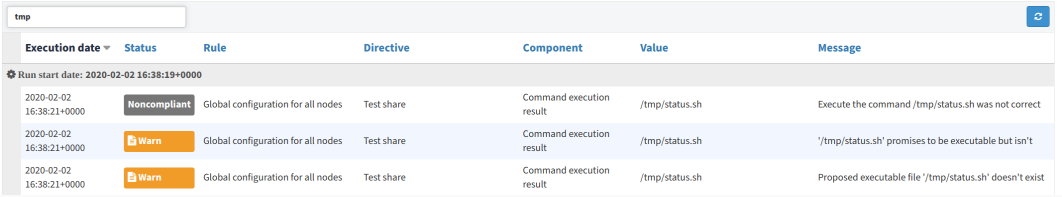
- Agents are usually not designed for error management at scale, and expect human interaction.
- Nothing built-in for automated outcome analysis (what failed and what has been done)
  - no structured errors in the policies, only access to a state (`error/ok/repaired`) from inside the policy
  - no business knowledge in the logs

Use (=parse) `stdout`

Improve run log

- Capture full agent output in info mode
- Parse it on the relay/server
- Associate simple logs with following contextualized log
- Works for log from the technique editor or modern techniques
- Not that good for legacy techniques: do everything then report
- Specific insertion/purging configuration for non-results logs (=simple logs)

```
2020-02-02T16:35:35+00:00    error: Proposed executable file '/tmp/status.sh' doesn't exist
2020-02-02T16:35:35+00:00    error: '/tmp/status.sh' promises to be executable but isn't
A| non-compliant Test_share              Command execution result  /tmp/status.sh     Execute the command /tmp/status.sh was not correct
```



Reports authentication

- We want to authenticate reporting
- We want to stay asynchronous

- End to end validation (check signature on root server)
- We need a signature (like we do for inventories)
- Prefer a standard
- We have a hierarchical node structure

S/MIME

HTTP

Use HTTP as it's:

- Already used for inventories (and Windows policy downloads)
- Well-known
- Easy operation and debugging (curl, etc.)
- Fast and powerful enough (even more with HTTP/2)
- Use simple file PUT (like inventories)

Implementation

Agent

- Use the existing `rudder agent run` wrapper
- Collect output, sign and compress
- Send to the server
- Retry in case of failure
- Allows back-filling compliance data

relayd

- A new daemon that runs on all policy servers (root + relay)
- Reminder: A root server is also a relay
- Replaces relay python API
- Layer between the webapp and the nodes
- Stateless (except for history)

Relay API

- Based on what existed since 3.2 (implemented in Python)
- Now versioned and documented
- Only listens locally
- Some endpoints behind `httpd` reverse proxy
- https://docs.rudder.io/api/relay
- Still missing a full stats/monitoring API (prometheus?)

Relay API

- /system/{status, reload, info}
- /policies/{node-id}/rules
- /remote-run/nodes/{id}
- /shared-files/{target-id}/...
- /shared-folder/{path}

relayd

- Config files: `/opt/rudder/etc/relay`
- A new `rudder-relayd` service
- Logs to `journald`
- Part of the `rudder-server-relay` package

Service hardening

```
User=rudder-relayd
ProtectSystem=strict
ReadWritePaths=/var/rudder/reports
    /var/rudder/inventories
    /var/rudder/shared-files
PrivateTmp=True
```

SELinux

A dedicated SELinux context

- Write access to work directories
- Read access to configuration and data files
- Connect to HTTP and postgresql ports
- Listen on port 3030
- Run the ‘rudder remote run‘ command with sudo

HTTP security

- Enforce TLS1.2+ everywhere (except syslog!)
- Option to check certificates in all HTTP requests
- Not directly linked
- Allows authentication of both ways
- For now, requires an existing PKI (and proper DNS setup)

Rust

- We wanted:
  - Reliability and security
  - Maintainability (<3 strong typing)
  - Low footprint (to allow ”embedded” relayd)
  - Easy packaging and deployment
- Chose Rust!

To sum up

We added a daemon between root server and nodes to:

- Forward reports and inventories (inotify-based)
- Check, parse and store reports on root server
- Provide the file sharing API
- Provide the policy and shared-files download API (for Windows)
- Only required `httpd` and agent to synchronize data files

Perspectives

Future

- Encryption option (using S/MIME)
- Use S/MIME for inventories too
- Allow policy updates over HTTPS for full HTTP communication
- Diffs for all non-compliance or errors (including files!)
- Connected mode for reactivity and continuity
- Flexible RUDDER server distribution (container, roles, cloud, etc)
- New (virtual) agents (maybe managed from relayd)
- Check server certificate by default

Feedbacks?

Thank you!