Locking Service I System Requirements and Scope and Approach

version: 2020-08-20T12:00:00CST

Stash Link to this document

the lock mess monster working group:

- Matt Kelly
- Andrew Nieuwsma
- Sean Wallace

Overview

We have a current locking implementation in HSM that allows services to 'lock' an xname as a way to tell other services *"I'm doing a protected, sensitive, or state changing operation on this xname, don't attempt to do the same thing right now"*

We also recognize the necessity of protecting things like PDUs and NCNs from being arbitrarily controlled given their HIGH level of impact to the system.

However, we do not have the capability for two services to coordinate with each other such that one service could delegate their control of a device temporarily so the other service could perform a necessary action.

Why now?

As the Shasta product continues to mature more and more services are created. These services are composable and have many complex interaction patterns. As there is not a single service (a monolith) controlling all aspects of system management and service interaction is it important that we advance the maturity of our systems management approach to ensure that our services do not unduly impact each other or decrease the operational capability of the system.

What is locking? What are reservations?

The 'locking' service introduces two closely related concepts: locking and reservations.

Locking and reservations is an abstraction layer in the systems management infrastructure. Our systems have many services that have to work in concert with each other. Some of our services have direct hardware access (via Redfish predominately, or sometimes IPMI). Most hardware related requests are benign in that they are reads only, they do not influence the state of the hardware (aside from putting load on the device). However some hardware requests do influence the underlying hardware; specifically power state change, firmware updates, configuration changes, etc.

Hardware requests that change the state of the system are not inherently risky; e.g. a power off or firmware update. However when done without any coordination could result in degraded performance of the hardware (including bricking or actually breaking hardware), as well as causing substantial outages to the overall system (e.g. turning off the NCNs) or impacting customer jobs.

Locking is an admin operation that prevents hardware from being reserved by low level services (CAPMC, FAS, etc). Locking is a signal that identifies "this hardware should not have its state changed".

Reservations is service level operation that reserves hardware for a duration of time to carry out some process. A reservation is a signal that identifies "this hardware is in use by a process, don't interfere".

Though these terms are very similar their distinction is important. A lock is a prevention mechanism; e.g. "don't do anything disruptive"; a reservation is a coordination mechanism; e.g. "This is in use, don't do anything disruptive". Locks are used by admins to tell the system "this hardware is off limits"; whereas reservations are used by services to tell other services "don't step on me".

Guiding principles

- Simplicity - we value a simple solution; as they are easier to maintain
- Ease of understanding - we value a solution that is clear and understandable so we can build a strong shared mental model.
- Clear intents - the purpose of this lock is to help put guardrails around our services. It is to make our services better, to increase operational stability of the system.

Disclaimers

Safety Critical Systems

Our l2/l3 hardware systems **are not safety critical systems**. These are NOT hardware locks, these are software locks. Their purpose is to facilitate the use of services, NOT to ensure physical safety. Any technician must follow appropriate safety controls to prevent danger, death, or destruction.

AUTHZ/AUTHN

This locking service is not a substitute or replacement for authorization (AUTHZ) or authentication (AUTHN); we expect those to be already set up and configured correctly. The locking service depends up AUTHN & AUTHZ. The lock service is not an attempt to ensure 'unauthorized' actors cannot do malicious actions, instead it is a safeguard for the hardware, to ensure conflicting (authorized) actions cannot cause outages to the system by causing our hardware management services (hardware gatekeepers) from stepping on each-other.

Use Cases:

Some of the predominate use cases we examined for locking are:

1. Protecting NCNs. We do need to be able to perform firmware updates and power control of NCNs but it's important that hardware management services cannot change the state of an NCN if another process is using the NCN.
2. Protect services from stepping on each-other.
3. Enable services to be able to coordinate with each-other via delegation; i.e. IFS calls BOS, BOS calls CAPMC, both IFS and CAPMC need lock control, so these services need a way to delegate lock control.

Charter

We are trying to protect with a locking service:

1. running compute jobs
2. the NCNs because they are the brain
3. the individual services from each other

What are we preventing with a locking service:

1. Power state change, on, off, power capping
2. firmware version change
3. configuration change (certs, ntp, syslog, creds, ssh/console keys, power capping)

What are we NOT preventing or interfering with, with a locking service:

1. registering for redfish alerting
2. querying power state
3. querying firmware version
4. other GETs
5. HSM inventory (external)
6. Discovery process (external)

What we CANNOT prevent with a locking service:

1. anyone just directly doing Redfish, IPMI, etc.

Service Interactions

Here are the services that we know about today that interact with each other and could step on each others locks.

- IFS -> BOS -> CAPMC
- FAS -> CAPMC (possible scenario)
- REDS/MEDS -> SCSD

Proposed Solution

Our proposal is to create a service with an API (RESTful microservice) outside of state manager that will provide an interface to manage locking and can be extended to apply a set of predetermined policies to the system. We have decided it is best to centralize an interface rather than having each service cope with knowing what other services are doing. This decoupled approach is necessary because of the wide swath of services in systems management and because of their complex interactions. To code explicit service knowledge into every service would be unmaintainable and likely lead to confusion, and system corruption.

Philosophy of Use

The primary philosophy of use of the locking service is to protect services from stepping on each other, and enable protecting the NCNs from being inadvertently controlled. The locking service does not have hooks back into other services to enforce that they cease and desist an operation if an xname lock has expired. This is up to the service to do correctly and the organization to ensure the code adheres to standards.

CAPMC, FAS, etc do not take initiative, they do things as initiated by an actor, usually human. Certain operations are like bullets from a gun, once fired impossible to call back. Other operations are like a freight train, you can ask it to stop, but it will take many seconds to a few minutes to come to a halt. These constraints must be taken into consideration by the system operator and by the integrating services.

A delegate should do its best to ensure that no action is started AFTER an expiration time has elapsed. An admin should do their best to not 'pull the rug out' from under a service, as the service will likely let momentum carry it for a least a little bit. There is no real 'programming' around this, but this must just be an expectation and realization on the capabilities of the system.

Alternatives

We have done some investigation to commercial or open source offerings that would fit our use cases and have not identified any available solution. In doing research on what other offerings hardware systems integrators in the HPC and data center space do, I believe we have a unique, or at least not-common concern. Generally speaking the control of hardware in these spaces is done through an interface (like Redfish). Access to these endpoints is controlled through AUTHN/AUTHZ (which we are not trying to supplement), however it does not appear that these systems have a higher level of coordination to protect the system (at least not one published), which is probably partially due to our NCN architecture.

The implementation inside state manager is insufficient because it does not allow for delegation of access, or infinite duration locks. Furthermore as there is a need for an agent to enforce policy, that is not appropriate to put into HSM. We believe that a separate implementation is the correct approach because it doesn't add more complexity to state manager (and break its paradigm), while also allowing the developers the greenfield space to iterate and innovate.

How does locking fit into systems management?

System Metaphor - Civil Engineering

It is important to understand the role that locking fills in the Shasta systems management infrastructure. There are several related concepts mechanism, policy, procedure, and coordination.

Allow me to draw a parallel to help illustrate how these concepts interrelate and to depict how locking fits into the system as a whole.

Imagine a large civil engineering project, specifically the ancient cisterns below modern day Istanbul, Turkey. One cistern in particular, the Basilica Cistern was constructed in the 3rd century and holds 80,000 cubic meters of water. The ceiling of the cistern is held up by 336 marble columns, each 9 meters high, spaced 5 meters apart.

Each column in the cistern is responsible for an incredible amount of load and serves a critical mission of supporting the four meter thick ceiling, which was the foundation for a basilica, hence the name, while allowing for thousands of gallons of water to be stored. How many columns do you think could be lost before the ceiling collapsed, destroying the cistern and the basilica? Probably not many.

Further consider the process of repairing a column of such importance. Could you afford the risk of removing multiple columns at the same time? How would you support the stress to the system? It doesn't take much of a civil engineer to understand that you must support the weight of the ceiling even if you remove the pillars!

There is a process for repairing and replacing pillars. Namely, shore up the area around the pillar, transfer the load to a temporary scaffolding, then replace the pillar. However you cannot do this to all pillars at the same

time for a few reasons: one - because there is not enough room, two - because there are not enough workers, three - because it would put too much risk on the system. So the process must be coordinated.

The following elements exist:

- mechanism - supporting the area around a pillar and removing the pillar.
- policy - workers are not free to just replace pillars at their discretion, they must be instructed to do so.
- process - the ceiling around a pillar must be supported before the pillar is removed.
- coordination - not too many pillars can be replaced at the same time because of risks and constraints.

All elements are needed, failure to have any element risks the integrity of the entire system. Coordination ensures all workmen understand the process. Coordination also ensures that policy is being followed and that workmen understand the mechanisms needed to accomplish the task.

How this metaphore relates to Shasta systems management

In this example:

- the workers and the tools are the services (the mechanisms) that have direct hardware access: CAPMC (power control), FAS (firmware), SCSD (configuration). If the tools and workmen act with no coordination, or don't have policies preventing their usage, they can interfere with each-other and can cause undesirable outcomes to the system.
- the pillars are the NCNs (and more generally the hardware at large). The NCNs are what support the entire system and allow it to function; take away too many NCNs too fast the system crashes.
- the locking API is part of the policy. The policy informs workers (CAPMC, FAS, etc) if it is okay to do work on the pillars (NCNs). Applied policy, like -> the NCNs are locked, is what prevents the services from powering them off.
- there must be a process for quiescing traffic on an NCN before any mechanism is applied that would power off an NCN or change its state. The process works with the policy to ensure that no 'rogue worker' does something before the system is ready.
- ultimately there must be an even higher level of coordination on the system. It is very valuable to have a process to 'SAFELY' power off or upgrade an NCN, but that process is not sufficient to ensure a higher level coordination and guarantee that there is not TOO much risk to the system at any one time.

We are proposing the Locking API as policy tool that will act like a gatekeeper, to ensure that no workmen (CAPMC, FAS) do actions to the NCNs without some level of process. CAPMC and FAS (and other tools like it -> reds, meds, scsd) are the direct access to the hardware. These services are the gates to hardware behind it. If we are able to deny any hardware access except through our approved gates we can build effect policy, process and coordination tools to ensure no harm happens to the system.

Requirements

Definitions

The following sections will use terminology like: authorize, deputy/reservation key. These terms have been co-opted but do NOT represent a cryptographic tool or standard. A deputy key is the key that can be used to check if a lock is still valid. It is the key that can be shared with a delegate who will act on the originators behalf. A reservation key is the key that the originator gets and can be used to renew/release the key. It is private and should not be shared. Authorized in this context means that a service wants another service to act on its behalf and it delegating its deputy key to that second service, so the second service can access the lock.

When we refer to admins we are referring to a persona. However we are also referring to the fact that not every endpoint/resource will be available to services. Certain API paths will only be available to admins, not services. We will leverage AUTHZ to ensure that certain functionality relating to the locking service cannot be done by non admin actors.

Behavioral / Functional

1. The system shall provide an interface that will:
    1. allow the state of xnames to be viewed. This information will include:
        1. lock state
        2. reservation state (includes expiration time)
        3. 'who' took the lock or reservation
    2. allow xnames to be locked.
        1. locking may only be done via the admin interface. Locking done by the admin interface will not produce a reservation, but a reservation mechanism shall be provided.
        2. locks have no expiration time, but may be unlocked.
    3. allow xnames to be reserved.
        1. The act of reservation shall cause a deputy/reservation key pair to be generated.
            1. The deputy key may be passed at the callers discretion to a delegate process. Possession and utilization of deputy key allows the holder to use the reserved hardware for the intent of the original caller. The keys should not be cached for future use; only to complete the request of the original caller. The delegate may pass the key on to other delegates as necessary.
            2. The reservation key may be used to release and renew the reservation. The reservation key

should not be shared with other processes.
3. A new reservation/deputy pair cannot be generated as long as the current reservation is in effect.

2. any reservation held by a service may only be held for a limited duration *d* not to exceed 15 minutes and no less than 1 minute, and in increments of 1 minute. The reservation may be:
    1. Released - which removed the reservation and allows another caller to request a reservation.
    2. Renewed - which allows the current reservation holder to renew their lease and extend the expiration time out *d* minutes.
        1. A reservation may be renewed subsequent times, there is no limit to the number of time a reservation may be renewed (extended).
        2. No reservation may be extended more than one period out (e.g. 5 back to back calls to renew only extend the reservation a total length of 1 period until the current period elapses).
        3. The entity that generated the deputy/reservation key MUST renew the lock, as they have the reservation key, no one else may renew or release the reservation.
    3. Expired - the caller does not renew the reservation, the reservation expires and is removed. The previous reservation holder may not renew the reservation, but they may attempt to get a new reservation.
3. any reservation held by an admin will be held for an infinite duration (e.g. admin reservations do not automatically expire). The reservation may be:
    1. Released - which removed the reservation, but not the lock, and allows an admin to request a new reservation.
4. an admin must hold the lock before they can generate a reservation

4. allow an admin or authorized service to break a lock and or break reservations. The purpose of this mechanism is to allow an admin to take control of an xname and prevent a service from locking or reserving the xname. Additionally the purpose is to allow CAPMC to submit an Emergency Power Off (EPO).
    1. a broken lock cannot be unlocked or locked or reserved. It must be repaired. Once the lock is repaired it can be set into locked or unlocked state. Once the lock is repaired reservations can be taken against the xname.
    2. No service shall be able to repair a lock; only an admin.
    3. a broken lock remains broken until it is repaired. No policy agent may compensate for this.

5. allow an admin to repair a broken lock
    1. the lock may be put back into locked or unlocked status. Reservations may now be taken against the xname.

2. log every request and event. Specifically:
    1. locking
    2. unlocking
    3. breaking
    4. repairing
    5. releasing
    6. renewing
    7. delegating

Policy Agent

Decision Point

We need to flush out how much of an active policy agent we will have for initial release. The tradeoffs are:

- Nothing automatic -> admins must identify and control
- fully automatic -> we have to develop an API to codify rules and then come up with the algorithm for applying them. This will take some time!
- partially automatic -> we bake in a rule like 'lock all NCNs' the first time you see them. Downside is there is NO visibility to this, and no easy way to change it. Probably more harm than its worth.

Requirements

1. allow for creation of locking policy. A locking policy is a general rule that will be applied to a set of xnames. The intent of a locking policy is to create sane defaults for certain classes of xnames in order to protect the system. Examples include: locking the NCNs so that they can not be power controlled without first being unlocked (or having a delegate created). The locking policy shall:
    1. Allow an admin to create, read, update, delete policies.
    2. Allow an admin to specify xname lock state of: `locked`. The default state for any xname is `unlocked`.
    3. Allow an admin to specify a filter for xname selection to include:
        1. role
        2. xname
        3. wildcarded xname

    4. component type (or device type?)

2. The system shall have an agent that will enforce locking policies

    1. Locking policies can be added that locks certain xnames according to filters that identify the xname as part of some set (role, deviceType, name, etc).
    2. **RESOLVE:** should the lock status be uninitialized? or default to unlocked?
    3. The system shall continually scan for new xnames in HSM and apply policies.
    4. Policies shall be immediately enforced upon creation or or upon new conditions that match the criteria.
    5. **RESOLVE** can there be conflicting policies? If the only position allowed is UNLOCKED then no; but if a policy can be LOCKED or UNLOCKED we might get into trouble.
    6. **RESOLVE** will the policy layer try to reset anything after the lock is repaired? How would we protect the system from a bad cycle of break, repair, re-lock. Really the question is can an ADMIN unlock without a token?
    7. **RESOLVE** should the policy layer specify any other trigger? How do we control WHEN the policy becomes enforceable? obviously the rule has to exist, but is there a downside to LOCKING a PDU or NCN right out of the gate?

Usability considerations

Principle:

- We don't want to make it harder than it has to be, admins will not appreciate having to make *n* calls to do something.
- Related; we don't want to add arbitrary service overhead; so if we can 'combine' *n* calls that would be more performant.

Generally applied concepts:

- all endpoints (reasonably) support filters:
- we want them to be able to filter on:
    1. xname[]
    2. partition[]
    3. group[]
    4. *RESOLVE* do we want the API to support xname wildcarding? roles?

- we have to consider ownership if we are able to do a request.
    - nothing can unlock a lock it didn't take.
    - admins can break everything
    - everything can unlock their own lock

- we have to consider ownership for subordinate key generation/revocation
  - cannot generate token for a lock you do not have
  - cannot generate a token if a token already exists

- allow support for variable success criteria
  1. `all or nothing` -> the entire transaction will pass or fail as a whole.
  2. `best attempt` -> lock what you can.

- when performing a bulk operation; allow for an array of tokens or xnames
- performing an operation is the same for 1 as n. The user can specify an array of xnames to work on and we will give them an array of tokens

Constraints

As part of implementing the locking service, the following constraints exist:

1. This service Shall replace the HSM (state manager) implementation and API of `locks`.
2. **RESOLVE** unclear how the deprecation will work
3. All services who call the HSM API must now call the Locking service
4. When an admin creates a lock (which will always be of infinite duration); they can generate a key (deputy/reservation pair) for the lock. The key will also be of infinite duration. The admin will use the deputy key to pass to a service and authorize the service to use the lock on its behalf. The admin can use the reservation key to release the key (which allows a new key to be generated, and will be considered good practice).
5. When a service create a lock (which is always of finite duration); they will get a key (deputy/reservation pair). The key has the same duration as the lock. The originating service MUST renew the key to extend the lifetime of the lock, which can be done via the reservation key. They can also release the lock with the reservation key. The deputy key can be passed to a service that is authorize to use the lock on the behalf of the originator.
6. CAPMC, FAS and any other gatekeeper (defined as those who directly use redfish/ipmi to change state on a device) services SHALL integrate with the locking API. They MUST be able to accept a deputy key (or list of them). The gatekeeper service MUST use the deputy key to query the locking service to find our if the lock is valid immediately before performing any operation against redfish or ipmi (that is considered a state change; e.g. dont need to check to do a READ, but SHALL check to do a WRITE).