Proxies Considered Harmful

Applying afterthought to the design of Grid permission delegation.

Foreword

- This presentation was originally written as a condensation of some historical thoughts around late June 2014.
- This version includes later development of the ideas, and some specific sections to address specific concerns.
- In the interests of maintaining the original design of the document, most additions are constrained to the end of the presentation.
- All additions are flagged with the corner note as so:

Note: added in Rev 2 of this document.

The Problem

- Jobs sent to remote services need to have the right to act on behalf of the job submitter.
  - Access VO/user specific software.
  - Read/Write VO/user specific data.

Naïve Solution

- Send a copy of our identity (Credential) along with our job, so it can act as us.
- This has security implications if the credential is compromised.
    - So sign a short-lived credential with our credential, blessing it.
    - Call this a “Proxy credential”

Implementing VOMS

- A job also needs to be able to choose which VO (and Role/Group) it acts as.

- Since we are already signing short-lived credentials, add VO information to the proxy

- But this should also be short-lived, so it doesn’t cause too many problems with security.

The Problem

- As Proxies have unlimited capabilities to act as the signer, we limit their lifespan.
- What happens if our job doesn’t get run/complete before the proxy lifetime runs out?
- Inconvenient for user to keep on making new proxies for jobs as they expire.
- (And same for VOMS extensions)

Naïve Solution (2)

- Maybe it would be okay to have longer lived proxies if they were "securely" stored by a trusted service.
    - "MyProxy" server.
- Let the MyProxy server sign (2nd level proxies) with its proxy, on receipt of a shared secret (password).
- Now we can automate our proxy lifetime extension!

Naïve Solution (3)

- Big VOs like Pilot Frameworks.
  - These need to act as other users…
  - But we authenticate other users with delegated proxies.
- Invent a new framework for user account switching
  - glExec!

Why you've just made Bruce Schneier sad.

- We started with a secure infrastructure, with public key cryptography, and one copy of each users' credential.

- We now have a system with many entities that can act with the users' capabilities, and a weakly-secured (password-only!) factory for making more of them!

- We've then patched it up repeatedly to try to recover some semblance of our original security…

Requirements for a Better Solution.

- Delegate only capabilities that a job needs.
  - Not omnicapable tokens.
- Bind the capability delegation to the job's tasks.
- Avoid limits that are not part of the task.
  - No lifetime for tokens (artificial impression of security)

A Less Naïve Solution

- Start with a job “payload”.
- SIGN the payload with user credential (private key).
- Distribute the payload to job management system along with the user certificate (public key).
- Distribution is over a channel authenticated with the user credential (prevents replay attacks).

Why is this safe?

- The User Certificate (Public Key) is always safe to distribute everywhere.

- The Signed Payload is proof that the User authorised the job.

- Together, the pair is a bound copy that only allows the Payload to be run - nothing else can be authorised by the pair.

Adding VOMS

- The only purpose of VOMS extensions is to bind a DN to a specific (authorised) membership.
- Sign copies of the USER CERTIFICATE (Public key) with the VOMS key + extension for specific role.
    - One new Public VOMS Certificate for each (VO, Role, Group) the User wants.
    - (Manage these in a keyring)

Adding VOMS (2)

- Now, rather than distributing the bare User Certificate with the Payload, we can substitute the Public VOMS Certificate with the appropriate group binding.

- These cannot sign anything either.

- And they are useless without a signed Payload.

- (VOMS bindings therefore don’t need to expire!)

Job Instantiation

- Signed Payload + Certificate arrive at Job Execution Endpoint (CE, Pilot, whatever).
    - 3 verifications:
        - Verify Certificate against CA Certs.
        - Verify VOMS signature against VOMS Cert.
        - Verify Payload signature against Certificate.

Job Instantiation (2)

- Create a new container, with throw-away user. Map VO-specific filesystems, User-specific filesystems within container.
- Unpack payload into container.
- Execute payload.
- [Specifically, we avoid long lists of user/group mappings as they are hard to maintain, and introduce unavoidable eventual consistency issues.]

Note: modified in Rev 2 of this document.

What about Storage?

- Storage often requires more levels of delegation than job execution
  - We don’t know/can’t specify the actual name of our destination file before we perform metadata operations. (Although, of course, we know its catalog name.)
- There are two problems here.

Note: modified in Rev 2 of this document.

The simple problem: Output Sandbox.

- If job simply writes output locally for staging back.
    - Secure output by signing/encrypting with user certificate (public key).
    - Allow user to retrieve (authenticate with credential, only user can decrypt sandbox with their private key).

The hard problem: LFC/SE/ etc

- Well written code will always know which files it will need before execution, and which files it expects to produce when complete.

- We can therefore map these to a series of Transactions : Source -> Destination

- Sign Transactions and distribute as part of payload.

  - We can bind the Transactions to the payload signature for better capability limitation and make them “one time”.

Note: modified in Rev 2 of this document.

Transaction Binding

- As a minimum, the user agent can sign each Transaction including a hash of the Payload in the resulting Signed Transaction.
- Storage agents can require proof that the Payload is present before allowing Transactions.
- (This is easier if the Payload is a script which executes preinstalled binaries, for example.)

Note: added in Rev 2 of this document.

Transaction Binding

- If we are prepared to trust the CE/Batch system:
  - The CE can also add additional signed bindings to the Transaction, binding the Transaction to a particular originating IP (a container, vm or worker node, for example).
  - The storage agent in this case will have to trust the CE (but we assume this is handled via the usual X509 trust hierarchy).

Note: added in Rev 2 of this document.

Read/Write asymmetry

- If we implement Grid Storage as immutable object placement, then Write requests are automatically idempotent (as each Write to the same name after the first fails).

- Read requests are not automatically idempotent, but are also not potentially polluting of the storage.

- Deletion requests should not be delegated (or allowed to be).

Note: added in Rev 2 of this document.

Negotiating Capability Delegation with Storage.

- Stage-In Transactions can potentially be resolved to local SE on submission (removes need for delegation).

- Stage-Out Transactions potentially need to support redirection by a catalogue service, and then by an SE.

- How do we let the SE know that its storage name is the same as the LFN in the Transaction?

Capability Delegation (2)

- Assumption: SEs trust a limited number of “File Catalogues”
- FC receives Transaction
  - (Verifies signature)
  - Append (SE,SURL) pair, and sign set with FC key.
- Agent sends augmented Transaction to SE.

An Alternative?

- We could also avoid the need for Storage transaction delegation by avoiding the need for FCs.
- Algorithmic SE,SURL generation (cf RADOS, Rucio, etc etc).
- May require consistent knowledge of World SE status between SE and Payload, if we want to locate SEs algorithmically as well.
- (Verification by performing same mapping at payload and SE)
- This has big problems with scaling the consistency traffic.

An optimisation

- Remove indirection levels in SEs in favour of bare object store interfaces (object names are hashes of the FC path).

- Now it is the *Storage* that performs the algorithmic authentication process to confirm that the object hash matches the FC name.

- (The FC in this case does hierarchical redirection to an SE that definitely has the file, but does not have to know its name there, just sign the request.)

Note: added in Rev 2 of this document.

Revoking Rights

- User Banning
  - Works as normal - we still verify against the public certificate and DN.
- User Credential Revocation
  - Works as normal - we still verify the public certificate against the CA + CRLs.

Revoking Rights (2)

- Revoking VO Membership and Roles.
  - Change: VOMS server distributes CRLs as CAs do.
  - Servers check against CRLs to validate VO signatures.

Additional notes

- “Grid Proxies” do currently provide a capability limitation mechanism (they can be limited in their scope to sign other proxies, for example).

- VOMS “roles” and “groups” etc can be used to emulate other capability limitations (in supporting middleware), by restricting particular capabilities (“get files”) to particular roles or groups.

Note: added in Rev 2 of this document.

Additional notes

- The problem is that:
  - Grid Proxies only *allow* the restriction of capability, they do not enforce it.
  - As such, they are vulnerable to the “lazy user” security hole (“Wouldn’t it be easier if we could all just look at anything?”)
  - Actual user experience on the grid, and numerous talks during the NGS era (from sysadmins as well as users) underline the above problem.

Note: added in Rev 2 of this document.

Additional notes

- VOMS Capabilities:
  - While there is some scope for limiting classes of activity to VOMS sub-hierarchies, there is no scope for “transaction specific” limitation.
  - VOMS just doesn’t scale to that, as it is not designed to.
  - (And most entities using VOMS quickly try to reduce the complexity of their group/role hierarchy anyway, thanks to the “lazy user” and “operational complexity” issues.)

Note: added in Rev 2 of this document.

Security Holes.

- This mechanism is not resilient against a root-level entity controlling the destination site, or the execution host (VM/VM host/container host).
- However, the cost of a job hijacking in this model is less than with proxy-based systems.
- The hijacker only gains the ability to execute the payload in question, or perform the data access actions associated with *specific files only* (and potentially only from a particular IP!)

Note: added in Rev 2 of this document.

Security Mitigation

- WORM/immutable files after placement removes much of the vulnerability for Write Transactions being hijacked (reducing it to a race condition, which is easily detectable if triggered).

- “One-use” Transactions similarly reduce the vulnerability for all storage Transaction hijacking (particularly as a job itself might acquire read tokens immediately on execution).

- (This also makes the Payload itself less vulnerable, if hijacked after some Transactions have been spent.)

Note: added in Rev 2 of this document.