Adam Retter

eXist @ XML Prague
14th Feb 2014

Correctness, Identity and Precision

Security Concerns in eXist

http://www.adamretter.org.uk adam.retter@googlemail.com

Sponsors!

- **The initial Development was sponsored by:**

  - **Many thanks to Joe Wicentowski**

- **I agreed to implement one thing… but this quickly snowballed into many more changes to eXist!**
  - **It's Open Source so we all win ;-)**

Correctness?

Security Correctness in eXist

For file operations,

let's compare what eXist does

with what Unix does...

Creating Files

- What owner and mode does Unix set on a new file/directory?

- What owner and mode does eXist set on a new resource/collection

Creating Files

- What owner and mode does Unix set on a new file/directory?
  - Owner is the current user and their primary group
  - Mode is default mode with the UMask subtracted
- What owner and mode does eXist set on a new resource/collection

Creating Files

- What owner and mode does Unix set on a new file/directory?
  - Owner is the current user and their primary group
  - Mode is *default* mode with the UMask subtracted
- What owner and mode does eXist set on a new resource/collection
  - Owner is the current user and their primary group
  - Mode is *default* mode with the UMask subtracted

```
eXist is CORRECT :-)
```

Copying Files

- What owner and mode does eXist set on the destination, when:
  - Copying a Collection to an empty destination?

Copying Files

- **What owner and mode does eXist set on the destination, when:**
  - **Copying a Collection to an empty destination?**
    - **Previously** – Permissions were copied from the Source.

eXist was WRONG :-(

Problem! - Cannot ever create a copy of container (Collection), which belongs to the copier.

e.g. You could not then make the copy private and update!
- Only DBA can chown uid, and only owner (and group member) can chown gid. (which you may not be either!)

Copying Files

- **What owner and mode does eXist set on the destination, when:**
  - **Copying a Collection/Resource to an empty destination?**
    - **Now** – Same as storing a new Collection/Resource
      - **Owner is the current user and their primary group**
      - **Mode is *default* mode with the UMask subtracted**

Result: If you copy something, it belongs to to you!

Copying Files

- **What owner and mode does eXist set on the destination, when:**
  - **Copying a Collection to an existing destination?**

Copying Files

- **What owner and mode does eXist set on the destination, when:**
  - **Copying a Collection/Resource to an existing destination?**
    - **Previously** – Permissions were copied from the Source.

      `eXist was WRONG :-(`

      Problem! - Existing permission on dest are being overwritten. Perhaps you are changing the permissions on someone else's file!

Copying Files

- **What owner and mode does eXist set on the destination, when:**
  - **Copying a Collection/Resource to an <u>existing</u> destination?**
    - **<u>Now</u> –** Now permissions are preserved on the destination

Result! - When you copy something over something else, you are really only copying the content. You now cannot subvert others files!

Copying Files

- **Other improvements:**
  - **Copy Permission Checks were too strict:**
    - No Longer need READ on destination Collection.
    - No Longer need READ on destination Document
      - Only the SYSTEM needs READ access to the metadata (transparent).

Result! - You can now copy things where before you would have been unnecessarily prohibited.

Copying Files

- **Intentional differences when compared to Unix:**
  - **eXist will check all permissions before copying a Collection**
    - Reduces the likelihood of a copy failing to copy ***all*** files
    - Not full-proof:
      - Dirty Writes to metadata (i.e. no Locking)
      - To be Atomic, would need COW or Locking scheme.

Moving Files

- Still need to investigate!
  - Suspect that Move to empty destination is most likely correct
  - Move to existing destination needs to be checked

Identity?

Running a Process inside eXist

- When you do anything in eXist, there is a known user id:
  - This is the unprivileged '*guest*' user...
    unless you have otherwise authenticated
    - This is true for all processes:
      - REST Server
      - RESTXQ
      - XQuery Trigger
      - Scheduled XQuery Trigger
      - Scheduled XQuery Job
      - etc...

Authenticating for processing

- **With RESTServer and RESTXQ you can either (or both):**
  - Authenticate, using challenge-response `HTTP 401 Unauthorized`
  - Or the **EVIL** option: ***xmldb:login*** and/or ***system:as-user***
    - Have to hardcode username/password or lookup!
      - Either way something *somewhere* is stored unencrypted
- **With Scheduled XQuery Job or XQuery Trigger:**
  - Execution is unattended so you cannot authenticate up-front
  - You still have the **EVIL** option available!

Changing Identities

- **We *now* store two user identities during processing:**
  - **Real User**
    - The user whom initiated the processing (e.g. XQuery execution)
  - **Effective User**
    - The user identity that the process should use when interacting with the database
      - Either way something *somewhere* is stored unencrypted
- **In keeping with eXist's current approach and the Unix permissions model, by default the Effective User will be the same as the Real User.**

Masquerade

- So now we have multiple identities, how can we exploit them?
  - Strangely, about this time two years...
    - eXist's permission model was completely re-written
    - Addition of setUid and setGid bits in the mode
      - However, these were not exposed to users and were not acted upon

Security Identity in eXist

We have now implemented

setUid and *setGid*

- **Affects Processes (e.g. XQuery)**
- **Collection and Resource mode advances**

setUid & setGid

- **setUid**
  - Set User Id
  - Changes the Effective User of Process execution
- **setGid**
  - Set Group Id
  - Changes the Effective User of Process execution
  - Changes the owner group on created sub-Resources/Collections

setUid & setGid

- **What do setUid & setGid bits look like in eXist?**
  - **As a mode string:**
    - The setUid and setGid char is 'S' or 's' (with exec)
      - `r-xr-S--` owner: *read* & *exec*, group: *read* & *setGid*
      - `r-xr-s--` owner: *read* & *exec*, group: *read*, exec & *setGid*
  - **As a mode octal number:**
    - setUid = 4000
    - setGid = 2000
      - `04450` owner: read & setUid, group: read & exec
      - `04550` owner: read, exec & setUid, group: read & exec

Case 1 – setUid Processes

When the *setUid* bit is set on an XQuery, and that XQuery is executed (by any user that has execute rights on it)...

...the *Effective User* will be the *owner user* of the XQuery Process.

Case 2 – setGid Processes

When the *setGid* bit is set on an XQuery, and that XQuery is executed (by any user that has execute rights on it)...

...the *Effective User* will be the Real User augmented with the *owner group* of the XQuery Process.

Case 3 – setUid & setGid Processes

When the setUid & *setGid* bits are set on an XQuery, and that XQuery is executed (by any user that has execute rights on it)...

...the *Effective User* will be the *owner user* of the XQuery augmented with the *owner group* of the XQuery.

setUid & setGid Processing

- **Examples of what you can we do with setUid & setGid on Processes:**
  - Execute XQuery with elevated/different privileges (than the Real User)
  - Very useful for XQuery Triggers / Scheduled Tasks
    - No Longer need to use *xmldb:login* / *system:as-user*
  - Very useful for XQuery executed by REST Server / RESTXQ:
    - Can be executed as *guest* but perform as a different Effective User
- **Warning:** It is a *very* sharp tool!
  - Best to use specific *service* user and group accounts for *setUid*/*setGid* execution, then only grant access to these service accounts to minimum Collections and Documents

Case 4 – setGid on Collections

- **setGid bit also has an impact when set on a Collection:**
  - **Resources created in the Collection will:**
    - *Inherit*the parent Collection's group owner as their own
  - **Sub-Collections created in the Collection will:**
    - *Inherit*the parent Collection's group owner as their own
    - Have their setGid set (so that setGid is inherited downwards also).

setGid on Collections

- **Examples of what you can we do with setGid on Collections:**
  - Makes it easy to share documents/collections between a group of users
  - Combined with ACLs it becomes it becomes even more powerful
    - setGid on a Collection allows you to store where you could not before
    - ACL's then allow many groups access to the documents
  - Can be used from setUid/setGid Processes
    - XQuery could store a document into a Collection as it is setUid/setGid
    - A different group of users may access/modify this later as they were setGid on the Collection.

XQuery Function Changes

- **Modified to act on Effective User:**
  - *xmldb:get-current-user*
  - *xmldb:is-authenticated*
  - *xmldb:login and system:as-user*
  - *session:set-current-user*
  - *sm:has-access*
- **New Functions:**
  - *sm:id* replaces *xmldb:get-current-user*
    - Retrieves both Real and Effective User identities

Future Work

- **setUid & setGid are static**
  - Sometimes you do not know authentication in advance, you need to be more dynamic
    - Currently you have **EVIL** *xmldb:login / system:as-user*
    - ***Not good enough!***
      - Replace *xmldb:login / system:as-user* with ***sm:sudo***
      - sm:sudo requires sudoeres config file in database security config
      - sm:sudo accepts a higher-order function as an argument

```
sm:sudo("aretter", function() {
    sm:id()
})
```

Precision.

Security Precision in eXist

Time

- **eXist stores:**
  - Created date on Collections
  - Created date and Last Modified date on Resources
  - Not always consistently implemented :-(

This is really not sufficient. A major part of any security policy should involve being able to identify who changed something and when something changed.

- We know the Who
- We do not know the When!

Security Precision in eXist

Time

- **Unix stores:**
  - **atime** The access time
  - **mtime** This is the modification time of **content**
  - **ctime** This is the modification time of **metadata** and/or **content**
  - Created date (on most modern filesystems)
- eXist *should* also adopt this model!
  - atime will be a challenge, due to XQuery across collections/documents

Questions?