APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 26-28 SEPTEMBER 2012

Security Issues with loginAdministrative

Angela Schreiber

Overview

- Introduction
- loginAdministrative & Friends
- Risks and Threats
- Identifying Security Issues
- Conclusions

Introduction

loginAdministrative and friends

- are easy to use
- are used frequently
- are powerful
- pose a potential source of security issues

Introduction

Review Findings

- 50% found unnecessary or problematic
- potential for minor flaws to become vulnerabilities

- everybody's code was affected

Overview

- Introduction
- loginAdministrative & Friends
- Risks and Threats
- Identifying Security Issues
- Conclusions

loginAdministrative & Friends

Sling API

- SlingRepository.loginAdministrative
- ResourceResolverFactory.
  getAdministrativeResourceResolver

JCR API

- Session.impersonate

Custom API and Services

- Exposing resources, adaptables or data associated with admin session

Overview

- Introduction
- loginAdministrative & Friends
- Risks and Threats
- Identifying Security Issues
- Conclusions

Risks and Threats

- Circumventing access control evaluation
- Threat boundary violation
- Privilege escalation

Risks and Threats

Effects on Read

- Information disclosure
- Privacy violations
- SQL/XPath injection (i.e. executing JCR query)

Risks and Threats

Effects on Write

- Content manipulation
- Changing access control or user content
- Creating or modifying executable content
- Access to the web console
- File system access

Overview

- Introduction
- loginAdministrative & Friends
- Risks and Threats
- Identifying Security Issues
  - Request Handling
  - Services and API
  - Content Structure
- Conclusions

Identifying Security Issues

Request Handling

- Admin session used in servlets and scripts
- Usage of services that use admin session
- Mixing different sessions
- Missing path normalization

Identifying Security Issues

Example: GET Request

```
public class MyServlet extends SlingAllMethodsServlet {

   protected void doGet(...) {
      Resource resource = request.getResource();
      Asset asset = getAsset(resource);

      AssetWriter w = new AssetWriter(response);
      w.dumpAssetInfo(asset);

      // FIXME 1: expose info read with admin
      Session admin = slingRepo.loginAdministrative(null);
      w.dumpUserInfo(asset.getUserID(), admin);

      if (needsUpdate(asset)) {
         // FIXME 2: write in GET; admin used to prevent exception
         updateAsset(asset, request, admin);
      }
   }
}
```

Identifying Security Issues

Example: POST request

```
public class MyServlet extends SlingAllMethodsServlet {

    protected void doPost(...) {
       String assetName = request.getParameterValue("assetName");
       String path = folderPath + "/" + assetName;

       // FIXME 1: admin session used to serve POST
       Session admin = slingRepo.loginAdministrative(null);

       // FIXME 2: missing path normalization
       Node asset = JcrUtil.createPath(path, "sling:Folder",
                    "nt:unstructured", admin, true);
       for (String name : getParameterNames(request)) {
           String value = request.getParameterValues(name);
           asset.setProperty(name, value);
       }
       ...
```

Identifying Security Issues

Services and API

- API definition forces the use of admin session
- Implementation makes use of admin session
  - Reading or writing data with admin
  - Objects bound to admin session
  - Mixing different sessions
- Caching Issues
- Admin session fields

Identifying Security Issues

Example: API forces use of admin

```
public class AssetService implements AssetManager {

    public AssetService() {...}

    public void activate() {
       adminSession = slingRepo.loginAdministative(null);
    }

    public Asset getAsset(String id) {
       // FIXME: service or API definition forces use of admin
       return getAsset(id, adminSession);
    }
}
```

Identifying Security Issues

Example: Implementation uses admin

```
public interface Asset {
    public String getId();
    public Extra getExtraInfo();
}
```

```
public class AssetService implements AssetManager {

    public Asset getAsset(String id, ResourceResolver resolver) {
       Resource asset = getAssetResource(id, resolver);

       // FIXME 1: extra info is bound to admin.
       Extra eInfo = getExtraInfo(id, adminResourceResolver);

       // FIXME 2: asset based on different sessions
       return new MyAsset(asset, eInfo);
    }
}
```

Identifying Security Issues

Example: Caching issue

```
public class AssetService implements AssetManager {

    public Asset getAsset(String id, ResourceResolver resolver) {
       // FIXME: cache entry created with different resolver
       if (cache.containsKey(id)) {
           return cache.get(id);
       } else {
           Asset asset = buildAsset(id, resolver);
           cache.put(id, asset);
           return asset;
       }
    }
}
```

Identifying Security Issues

Example: admin resolver/session field

```
public class AssetService implements AssetManager {

    private ResourceResolver adminResourceResolver;

    public void activate() {
       // FIXME 1: get and close resourceResolver on demand
       adminResourceResolver =
                   factory.getAdministrativeResourceResolver(null);
    }

    public Session getSession() {
       // FIXME 2: exposing admin session
       // FIXME 3: method naming doesn't reveal the 'admin' nature
       return adminResourceResolver.adaptTo(Session.class);
    }
}
```

Identifying Security Issues

Content Structure

- Data model requires use of admin session
- Data container such as e.g.
  - storing backup information
  - collecting status information
  - tracking
- Structure created by services without proper access control setup

Identifying Security Issues

Example: Content structure

```
public class AssetService implements AssetManager {

    public void backup(Asset asset) {
       // FIXME 1: use of admin (asset-session not allowed)
       Session admin = getAdminSession();

       // FIXME 2: backup-store is not ac-protected
       String path = getBackupPath(asset);
       Node store = JcrUtil.createPath("/var/assetBackup", path,
                    "sling:Folder", ntName, admin, true);

       // FIXME 3: original access control is lost
       writeBackup(asset, store);
    }
}
```

Overview

- Introduction
- loginAdministrative & Friends
- Risks and Threats
- Identifying Security Issues
- Conclusions
  - Request Handling
  - Designing Services and API
  - Content Modeling

Conclusions

Request Handling

- Always use the request session
- Avoid mixing different sessions
- Understand the nature of services and API used
- Review access paths to the scripts
- Normalize paths generated from input parameters

Conclusions

Designing Services and API

- Design API and Services without admin session
- Consider security requirements at an early stage
- Explicit method and field names
- Implementation
  - Avoid caches accessed by different sessions
  - Review access paths to your services
  - Consider using a configurable user with specific permissions

Conclusions

Content Modeling

- Let security requirements drive your content structure:
  - Managing access control should be easy
  - Enforcing access control comes for free
- Avoid copying data to containers (e.g. /var/*)
- Verify proper permission setup

Thank you !