Reuse without misuse – Application isolation done right

Mirko Jahn*, Boris Terzic, Markus Gumbel*+

* Speaker

+ hochschule mannheim

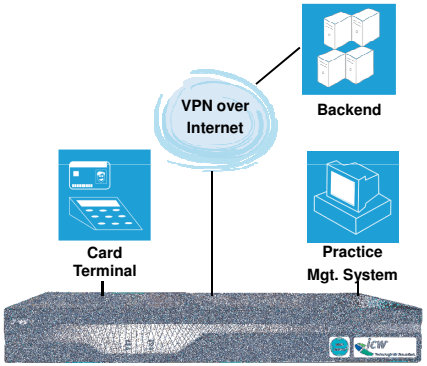
Department for Computer Science,
Institute for Medical Informatics

Agenda

- Motivation
- Java Security shortcomings
  and what OSGi can add
- What is still missing:
  Concept for bundle-like domain security
- Remaining Challenges / Lessons Learned (so far)

Why more security? An example

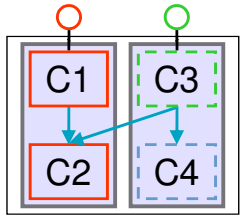
- Germany's internet-based e-health-infrastructure project
- Infrastructure has to be very secure
- Business logic (e.g. prescription workflow) requires digital signatures



"ICW healthcare connector based on Cisco AXP"

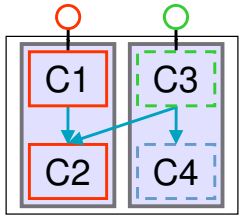
Secure and certified applications

- Two types of applications
  - "Must-have" (and highly secure)
  - on demand
- Reuse (3rd party) components
- Contradiction
  - Certification requires a minimal and inflexible but at least an isolated system
  - Added value applications require installation of new and potentially malicious software



Application domain separation

- How to isolate an application domain?
  - OS level (e.g. virtual OS)
  - Process level (different native applications)
  - ☑ Component level (within one JVM)
- Resource-friendly
- Less complex and easier to maintain



Wanted! A suitable Java-component platform

- OSGi
  - … becoming the standard for Java-based component technology
  - Simple but powerful model with clear versioning concept
  - Hot deployment and remote updates
- This approach holds true for other OSGi-based domains as well
  - e.g. automotive sector

Fort Knox, Kenntucky (USA) – Screenshot taken from google maps.

Requirements for application level security

- Application isolation within ONE JVM
- Dynamic adjustable upon ones needs
- Not managed by one and only authority

Plain Java™ 2 Security

- Enforced by the JVM
  - Loadtime: Code verifier
  - Runtime: ClassLoader and SecurityManager
- customizable security through extension of `java.security.Permission`
- ProtectionDomain objects as main instance to handle roles and permissions
- Policy files as configuration entity

Shortcomings of plain Java™ 2 Security

- Not focused on “component” isolation, but application isolation (like applets)
- No notion of service level security
- No actual notion of lifecycle or dynamism

From Clay Bennett: http://www.claybennett.com/pages2/security.html (slightly adapted ;-))

What OSGi has to add?

- Finer grained isolation with new permissions
  - `PackagePermission` (import, export)
  - `BundlePermission` (provide, require, host, fragment)
  - `ServicePermission` (register, get)
  - `AdminPermission` (execute, lifecycle, resolve,...)
  - ➔ Better encapsulation of components
- Dynamic management of policies/ permissions
  - Permission Admin `(org.osgi.service.permissionadmin.*)`
  - Conditional Permission Admin `(org.osgi.service.condpermadmin.*)`

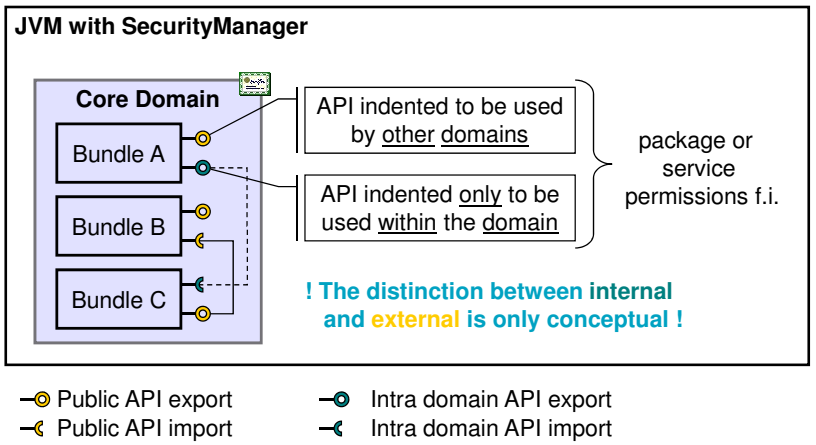
How to isolate your code? Identify…

- … application domains and sub-domains (pretty much analog to composites in SCA)
- … intra and inter domain boundaries (boundaries between composites)
  - The minimal set of Permissions each bundle needs (explicitly via Java API calls)
  - Import and export statements in the manifest files
  - Services
  - Lifecycle dependencies
  - System (sockets, file access, properties,…)

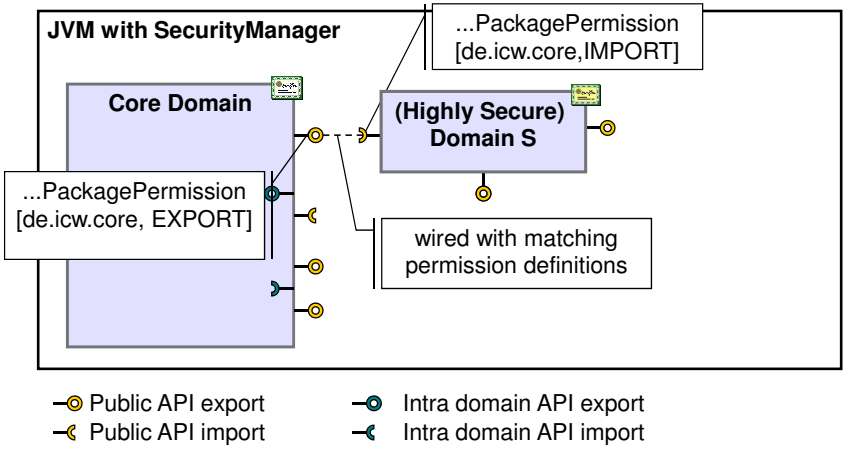
How to isolate your code? Assign…

- … different certificates for each domain/ sub-domain
- … permissions on these certificates based on the findings before (including intra and inter domain dependencies)
- … scoping limit for each bundle (only the actual permission needed by the bundle) – use the `permission.perm` file

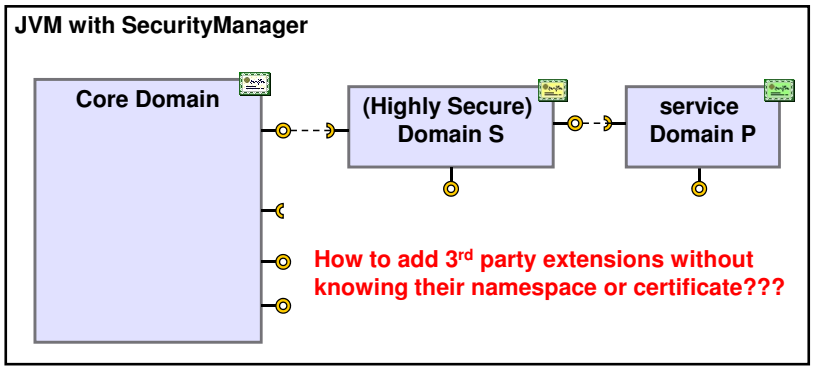
Define your domain application(s)



How to wire your domains?

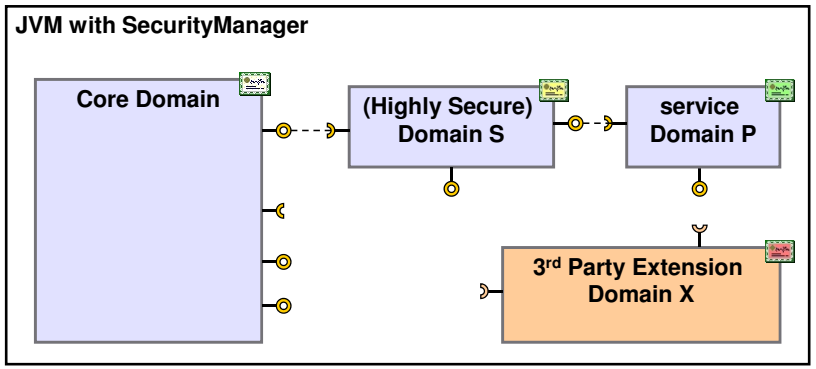


How to wire your domains?



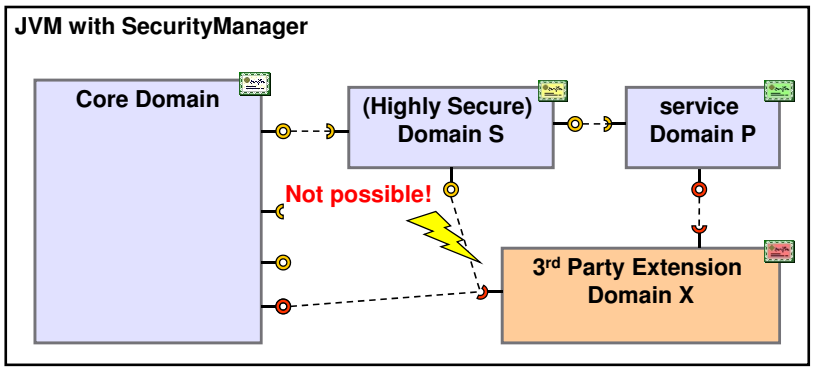
- Public API export
- Public API import

How to integrate 3rd party extensions?



- Public API export
- Public API import

How to integrate 3rd party extensions?



➔ **Now, only the exposed API is security relevant!**

Run & Refactor

- Run usually reveals a huge amount of missing permissions
- Refactor
  1. Investigate missing permissions
     ➔ either add permissions or use doPrivileged()
  2. Examine permissions assigned to the certificates for undesired overlaps or dependencies.
     ➔ move or refactor/ redesign
  3. API analysis on tainted parameters based on permissions assigned to the certificates

Remaining Challenges/ Lessons learned

- How to dynamically assign permissions on domains without prior knowledge of their features and without potentially compromising installed and certified domains/ applications?
- What about start-up behavior? First come, first serve? How to define, which bundle is the one to set permissions? (Secure Infrastructure – where to start, where to stop)
- How bullet proof is your OSGi container? Who can tell?
- How to smoothly integrate security in the development process while improving code quality and removing vulnerabilities
- Tooling support to audit development and the live system
- Standardized meta data format to express permission requirements (mandatory and optional ones)
- Certificate Access for a installed bundle
- Using OSGi Security drives design (in a good way)

Q & A

References

- [1] Secure Code Guidelines (Sun) http://java.sun.com/security/seccodeguide.html
- [2] Security Documentation (Sun) http://java.sun.com/javase/6/docs/technotes/guides/security/
- [3] Coding for least privilege http://en.wikipedia.org/wiki/Least_privilege
- [4] InterComponentWare AG (ICW) http://www.icw-global.com
- [5] Cisco AXP Router and the ICW Box http://www.cisco.com/en/US/prod/collateral/routers/ps9701/data_sheet_c02_459078.html
- [6] Spring Application Platform http://www.springsource.com/web/guest/products/suite/applicationplatform

Trademarks & Copyrights