redhat.

SECURING YOUR JAVA EE APPLICATIONS

What they don‘t tell you.

Markus Eisele
markus@jboss.org

@myfear
blog.eisele.net

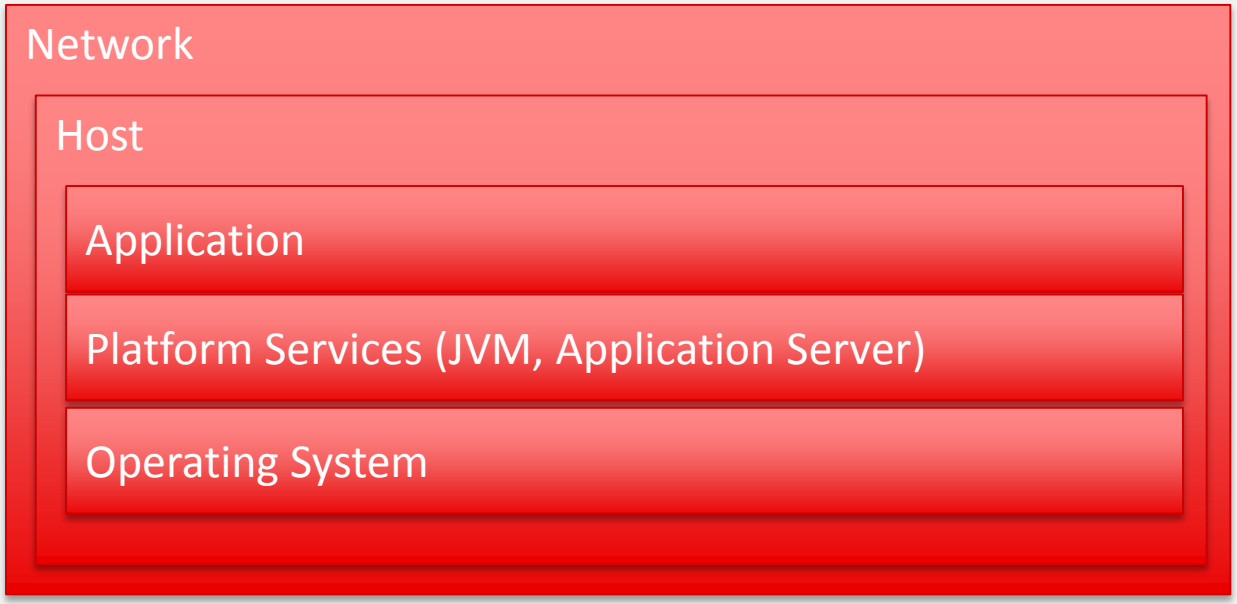

YOU KNOW JAVA EE
redhat.

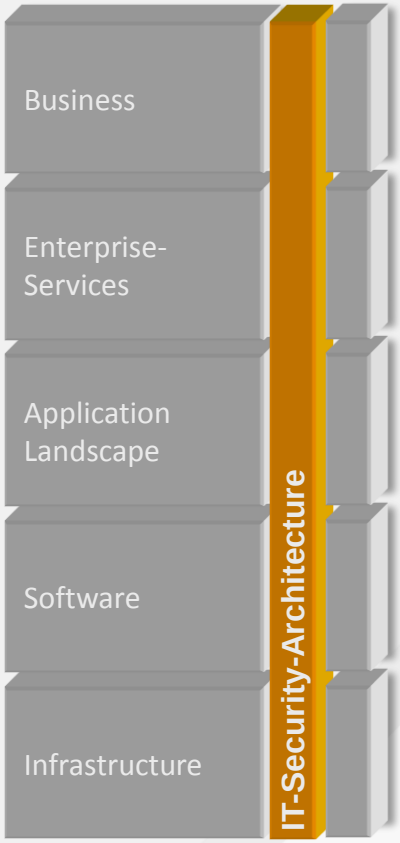

BUT WHAT DO YOU KNOW ABOUT
SECURITY?
redhat.


PASSWORD

HOLISTIC SECURITY

To build a secure Java EE application, a holistic approach to application security is required and security must be applied at all layers and services.





redhat.

SECURE NETWORKS

Secure applications rely on secure networks

Things to look out for:

- Router
- Firewall
- Switch
- Protocols
- Ports
- Firmware
- Standard passwords

SECURE HOSTS / OPERATING SYSTEMS

Repeat for all of them: including database, message broker, httpd-server and all the others.

Things to look out for:

- Shared Volumes
- Running Services
- Accounts (user and service)
- Auditing and Logging
- Files and directories
- Patches and updates

PLATFORM SERVICES

All pre-installed runtimes, like the JVM/JDK, Application server, etc.

Things to look out for:

- Policy Files
- File Access Rights
- Default Passwords
- Admin Console

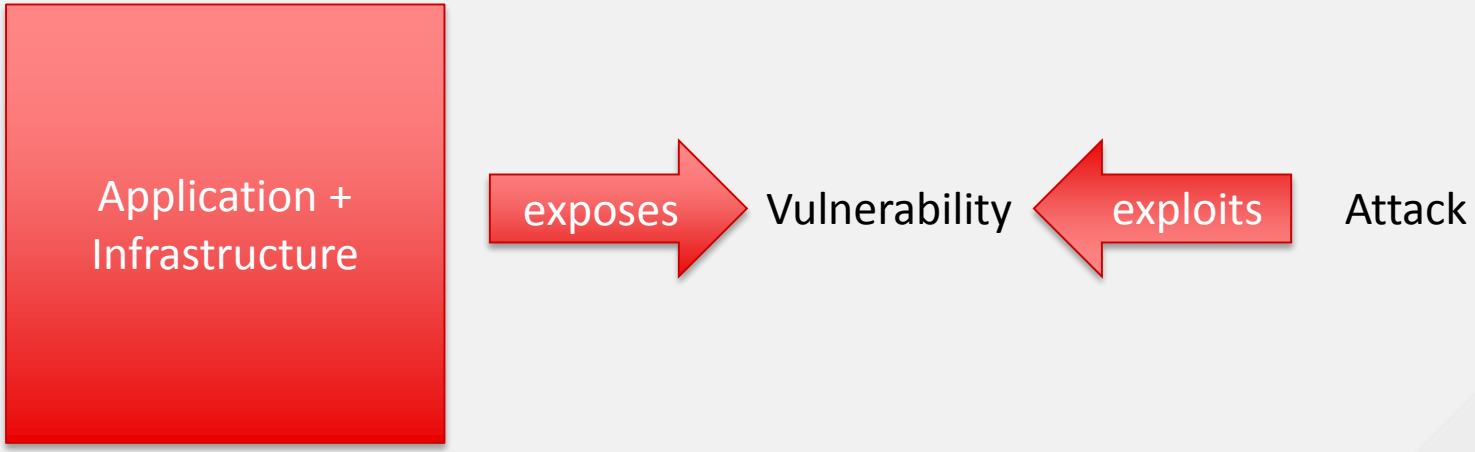
WHAT IS APPLICATION SECURITY?

Security relies on the following components

THREATS, VULNERABILITIES, AND ATTACKS

A thread is a potential event that may affect your system. An attack exploits a vulnerability in your system.

How do you decide if an application is secure?



NO SECURITY WITHOUT THREATS

It is not possible to design and build a secure Web application until you know your threats.

Thread modelling in the design phase:

STRATEGY – APPROACH - GOAL

APPLICATION VULNERABILITY CATEGORIES

Some basic categories mapped down from the security components

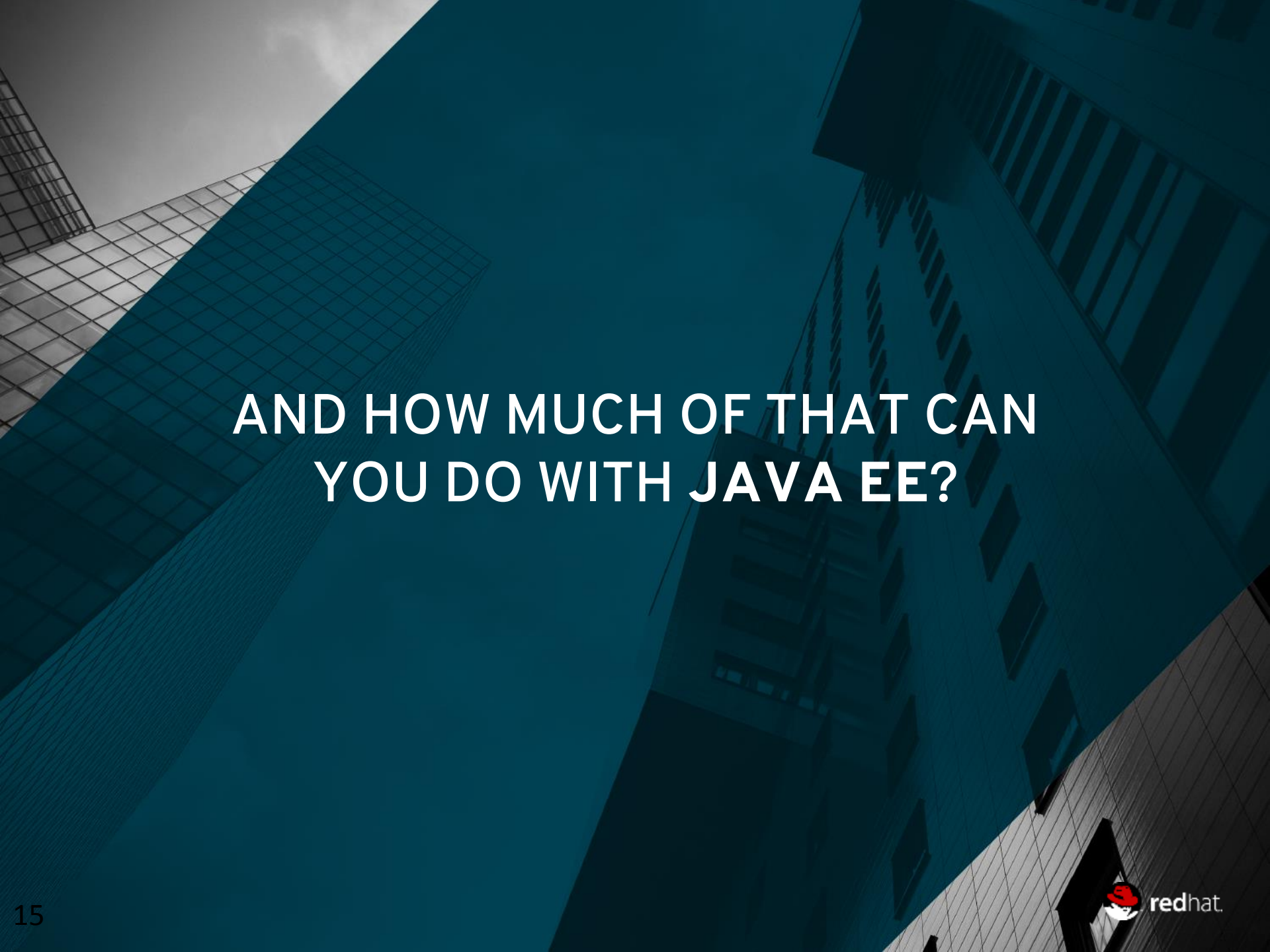
- Input Validation
- Authentication
- Authorization
- Configuration Management
- Sensitive Data
- Session Management
- Data Encryption
- Parameter Manipulation
- Exception Handling
- Auditing and Logging
- Timestamping
- Role Based Access Control
- Execution Sandboxing
- Process Separation

CORE SECURITY PRINCIPLES

Best Practices for designing secure applications.

THINGS TO KEEP IN MIND FOR YOUR SYSTEM DESIGN

- Compartmentalize
- Use least privilege
- Apply defense in depth
- Do not trust user input
- Check at the gate
- Fail securely
- Secure the weakest link
- Create secure defaults
- Reduce your attack surface


AND HOW MUCH OF THAT CAN
YOU DO WITH JAVA EE?

AUTHENTICATION

Basic, Form, Digest, Client, Mutual

The means by which communicating entities, such as client and server, prove to each other that they are acting on behalf of specific identities that are authorized for access. This ensures that users are who they say they are.

- Configuration via deployment descriptor
- Application roles mapping into server groups
- Most servers provide standard login modules (DB, LDAP, Client-Cert, Filesystem)
- Additional resources (Keystores, LDAP, DB)

- Custom Authentication Logic via:
  - JASPIC
  - JAAS
  - Server Specific Features (Realms)

AUTHENTICATION

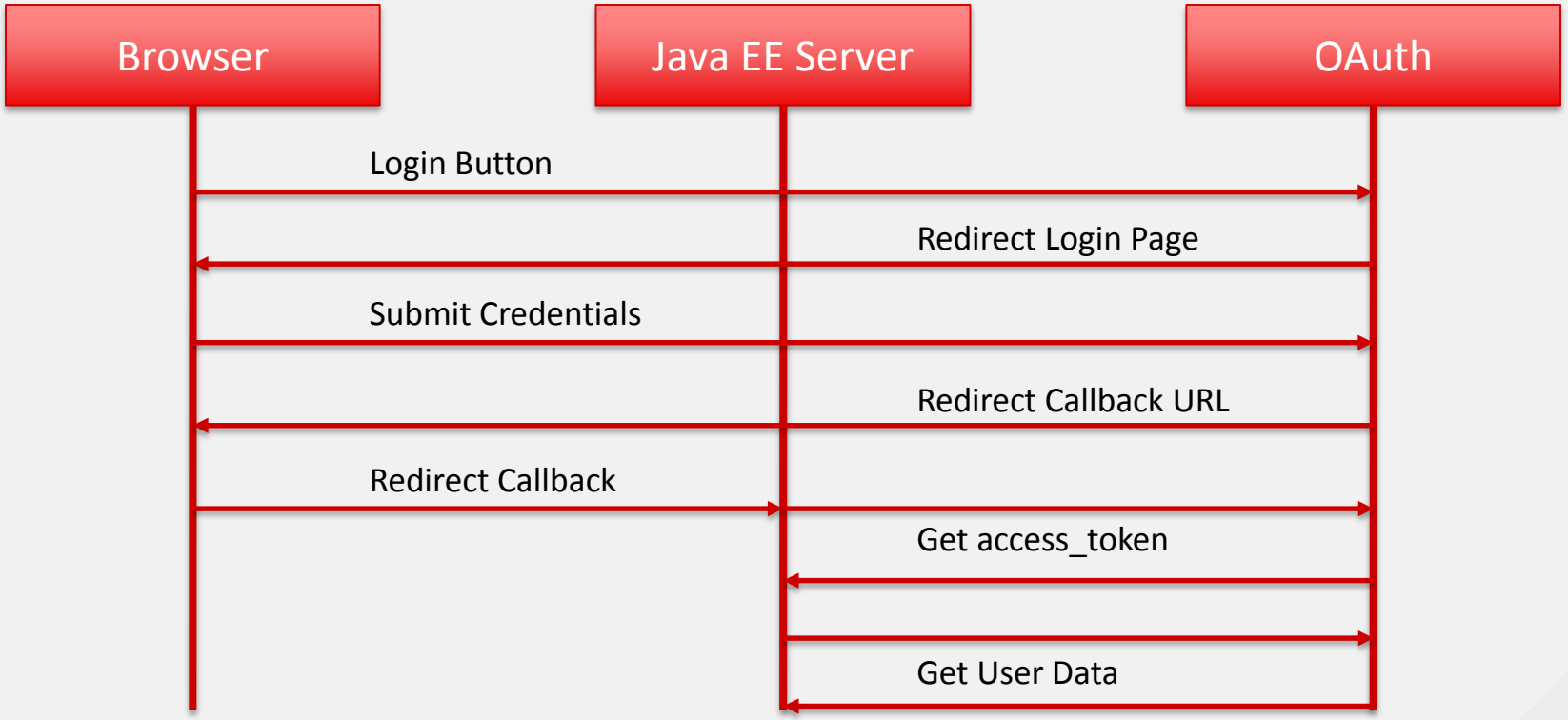
The missing pieces

Well known and broadly used and still missing in Java EE

- Two Factor Auth Support (Custom Token, Hardware Token, Soft-Token)
- Social Login (Facebook, Twitter, GitHub)
- User Registration / Self Service
- Easy Customizing for custom login pages
- OAuth support
- Policies (Password / Revocation)

AUTHENTICATION

Option: Custom implementation (e.g. OAuth w/ programmatic security)



https://oneminutedistraction.wordpress.com/2014/04/29/using-oauth-for-your-javaee-login/

AUTHENTICATION

Custom Implementation

ADVANTAGES

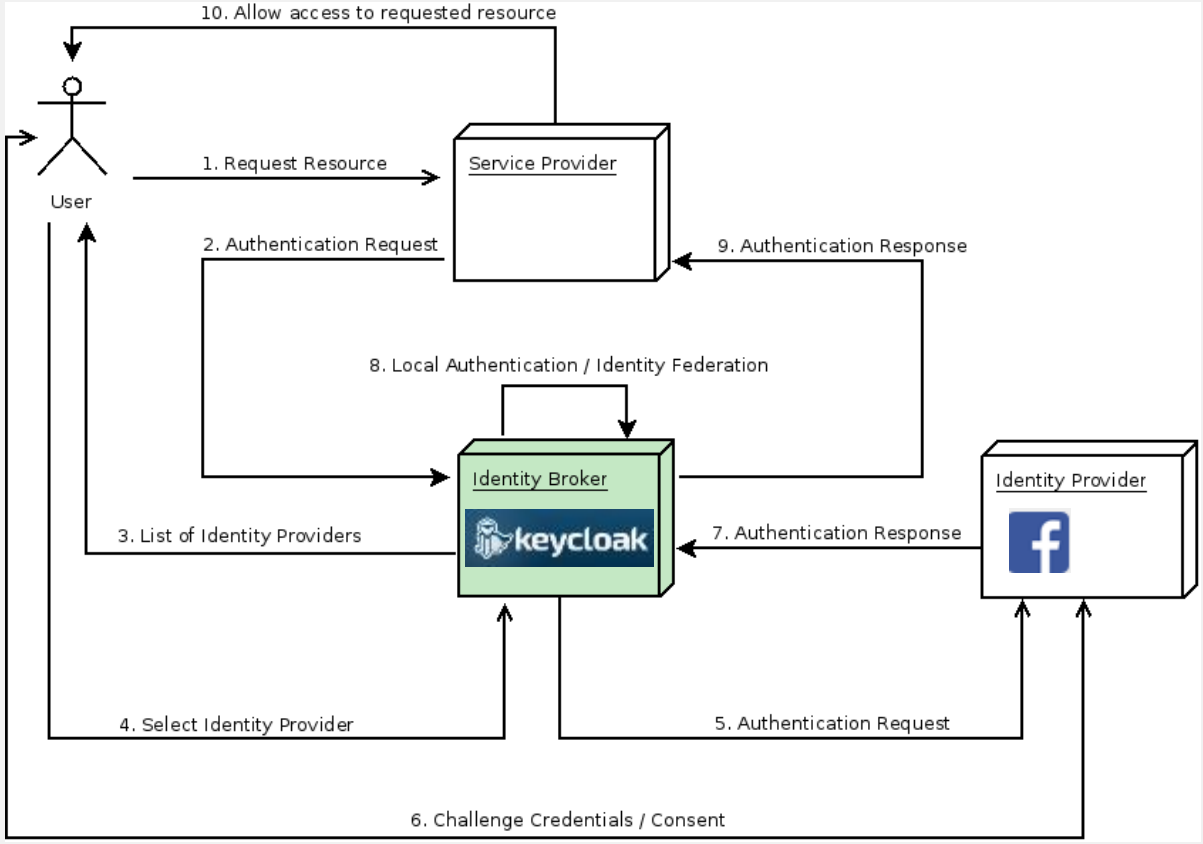
- Minimal – Meet Needs
- Fast Implementation

DISADVANTAGES

- Tied to one provider
- Hard to extend
- Hard to test
- Potential implementation mistakes expose vulnerabilities
- JAAS/JASPIC hard to understand
- Easy to be accidentally tied to a specific server
- Not “microservices-ready”

AUTHENTICATION

Option: Identity Broker – e.g. Keycloak



http://keycloak.jboss.org/docs

AUTHENTICATION

Identity Broker

ADVANTAGES

- Flexible
- Configurable and extendable
- Runtime Changes
- Many Identity Providers
- Additional Features (Government/Policies)
- Ready for microservices (downstream propagation)

DISADVANTAGES

- Separate System - Complexity
- Takes User/Roles management out of the Java EE Server

AUTHORIZATION

Or Access Control

The means by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints. This ensures that users have permission to perform operations or access data.

- Configuration via deployment descriptor or Annotations (@RolesAllowed)

- Custom Authorization Logic via:
  - Programmatic Security +
  - CDI / Aspects +
  - Caching
  - JACC

AUTHORIZATION

The missing pieces

Well known and broadly used and still missing in Java EE

- Fine Grained Access Control (technical or business)
- Data Protection Model through different specifications (e.g. data access per user/tenant)
- Coherent AccessException Model

AUTHORIZATION

Option: Custom Implementation

Use a mixture of standard authorization mechanisms (Principal) and add a custom user object which get‘s enriched with Permissions.

- Authentication via form-based or other method
- Creating an application User per Principal
- Resolving custom rights per user (Datastore, In-Memory DB)
- Implementing Around Invoke Aspects (@AllowedTo(Permission.WRITE)

http://kenai.com/projects/javaee-patterns/sources/hg/show/javaee-security

AUTHORIZATION

Custom Implementation

ADVANTAGES

- Native to the Java EE programmer
- Simple, Customizable at development time
- Assignable per application

DISADVANTAGES

- Potential implementation mistakes expose vulnerabilities
- Common Code tied into all modules
- Complex permission matrices need to be extensively cached
- Cache invalidation
- No runtime changes to permissions

AUTHORIZATION

Option: JACC (Java Authorization Contract for Containers)

It “defines a contract between a Java EE application server and an authorization policy provider" and which "defines java.security.Permission classes that satisfy the Java EE authorization model”

Implement:

- A factory that provides an object that collects permissions
- A state machine that controls the life-cyle of this permission collector
- Linking permissions of multiple modules and utilities
- Collecting and managing permissions
- Processing permissions after collecting
- An "authorization module" using permissions for authorization decisions

http://arjan-tijms.omnifaces.org/2015/03/java-ee-authorization-jacc-revisited.html

AUTHORIZATION

JACC (Java Authorization Contract for Containers)

ADVANTAGES

- Integrated with the Java EE container
- Part of the standard
- As secure as it can get

DISADVANTAGES

- C.O.M.P.L.E.X.I.T.Y
  - Amount of classes
  - APIs
  - Verbose
  - Not truly portable
  - Application Server Wide

AUDITING

Logging which security relevant changes happen.

An **audit** trail (also called **audit log**) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event.

- There‘s nothing here in Java EE

Options:

- Custom Implementations
- Logger Implementations
- Data centered approaches (JPA / Database)
- Server specific implementations

AUDITING

Option: Custom Implementation

A transactional application often requires to audit the persistent entities. Sometimes you have to be able to track down changes and build up a history of changes. Or you're requirement to have a long term rollback or need to archive in accordance with legal audit requirements.

- Aspects in the widest sense (EJB, CDI)
- @AroundInvoke captures relevant events on the service layer
- Events should be persisted asynchronously
- Ideally includes user information, date time and event type

http://blog.eisele.net/2011/01/five-ways-to-know-how-your-data-looked.html

AUDITING

Custom Implementation

ADVANTAGES

- Suitable for business activity auditing
- Simple to implement with a small set of activities.
- Maybe used to also track performance of service calls

DISADVANTAGES

- No rollback or data centered auditing
- Doesn't include security relevant audits (aspects don't work in login-modules or realms
- May be "forgotten"
- Performance critical
- No UI to generate an audit trail

AUDITING

Option: Logger Implementation (e.g. Log4j)

Some logging frameworks were specifically designed to support audit features. One among them is Log4j. Others with more respect to legal audit requirements also do exist.

- Use Log4j‘s audit logging features
  - Populate ThreadContext Map with basic data like: user-id, remote-IP address, application-name, application-version, etc.
  - Create a StructuredDataMessage for every event
  - EventLogger.logEvent(msg)
- Select an appropriate provider (Syslog, JMS, JDBC, Flume, etc.)
- Combine with aspects and automatically constructed DataMessages for broad coverage on services.

AUDITING

Logger Implementation

ADVANTAGES

- Native for Java EE developers
- Easy to use as “logging”
- Will work in security relevant classes

DISADVANTAGES

- Guaranteed delivery ?
- Performance overhead with growing events

AUDITING

Option: JPA Centered Approaches (Hibernate ORM Envers)

The Envers module aims to enable easy auditing/versioning of persistent classes. All that you have to do is annotate your persistent class or some of its properties, that you want to audit, with @Audited.

Just add @Audited to your entities

- auditing of all mappings defined by the JPA specification
- auditing of some Hibernate mappings, which extend JPA, like custom types and collections/maps of "simple" types (Strings, Integers, etc.) (see here for one exception)
- logging data for each revision using a "revision entity"
- querying historical data

http://hibernate.org/orm/envers/

AUDITING

Option: JPA Centered Approaches (Hibernate ORM Envers)

ADVANTAGES

- Historical Data
- Revisions
- Fully integrated into JPA

DISADVANTAGES

- Only Data Centered Auditing

CONFIDENTIALITY

or data privacy

Confidentiality is the process of maintaining data privacy wherein we secure the communications channel, to make sure the data is not accessed in its original form by a third party by eavesdropping.

Java EE doesn‘t bring a hell lot to the table here.

Options:

- SSL Termination on the Application Server
- SSL Termination on a box before (httpd, router, appliance)

CONFIDENTIALITY

Terminating SSL on the App-Server

<transport-guarantee>CONFIDENTIAL | INTEGRAL</transport-guarantee>

A value of INTEGRAL means that the application requires the data sent between the client and server to be sent in such a way that it can't be changed in transit.

A value of CONFIDENTIAL means that the application requires the data to be transmitted in a fashion that prevents other entities from observing the contents of the transmission.

CONFIDENTIALITY

Terminating SSL on the App-Server

ADVANTAGES

- Access to the SSL connection information (ssl-session-id)

DISADVANTAGES

- Performance
- Configuration complexity
- Testing locally

CONFIDENTIALITY

Terminating SSL on a box

Httpd server (Apache)

- Mostly in place anyway for load-balancing (therefore needs to inspect the data anway)
- OpenSSL as a way to terminate SSL/TLS connections

Firewall / Router (including or not LB features)

- See above.
- Mostly hardware accelerated SSL termination.
- High performance.

CONFIDENTIALITY

Terminating SSL on a box

ADVANTAGES

- If SSL session information are required, they have to be passed downstream
- Considered “best practice” for failover, clustering and load-balancing

DISADVANTAGES

- Additional infrastructure

INTEGRITY

Mostly Data Integrity

Data integrity is the process of verifying if data is transmitted without corruption or modification, thus making sure that the data received at the receiver end is in fact the same message sent by the sender.

In Java EE mostly two big areas:

Implemented for data access/exchange with JPA and JTA.

INTEGRITY

Mostly Data Integrity

Data integrity is the process of verifying if data is transmitted without corruption or modification, thus making sure that the data received at the receiver end is in fact the same message sent by the sender.

In Java EE mostly data integrity

- Implemented for data access/exchange with JPA and JTA.

Options:

- Enforce integrity on the database
- Store and compare hashes of entities

AVAILABILITY

Clustering, Scaling et al

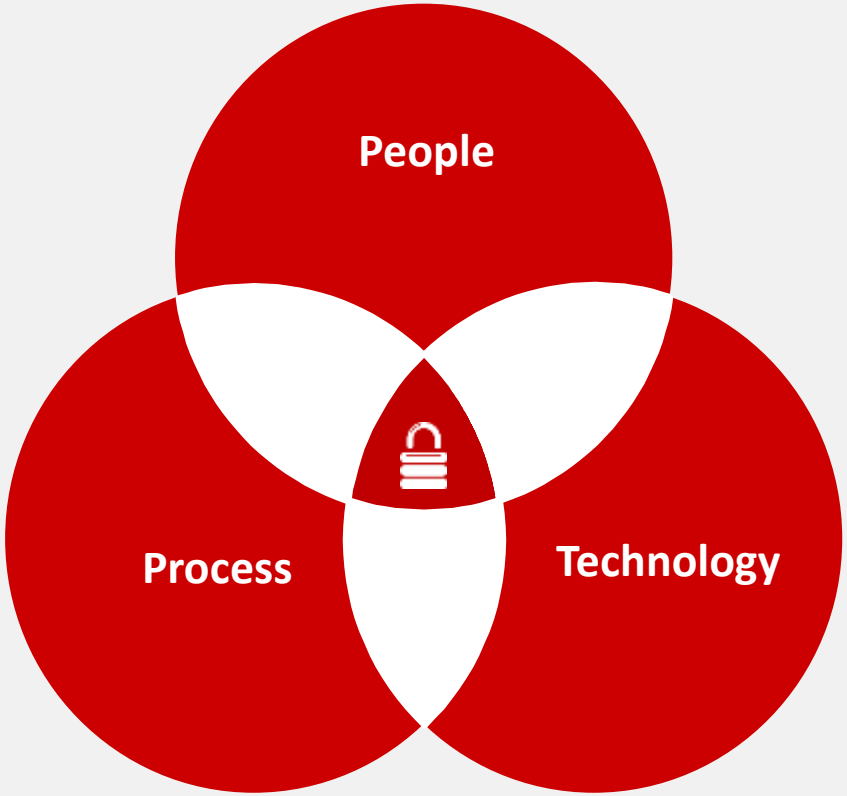
Availability is commonly referred to as the most underspecified non-functional requirement for applications. Fact is: Applications only earn money, while they are running.

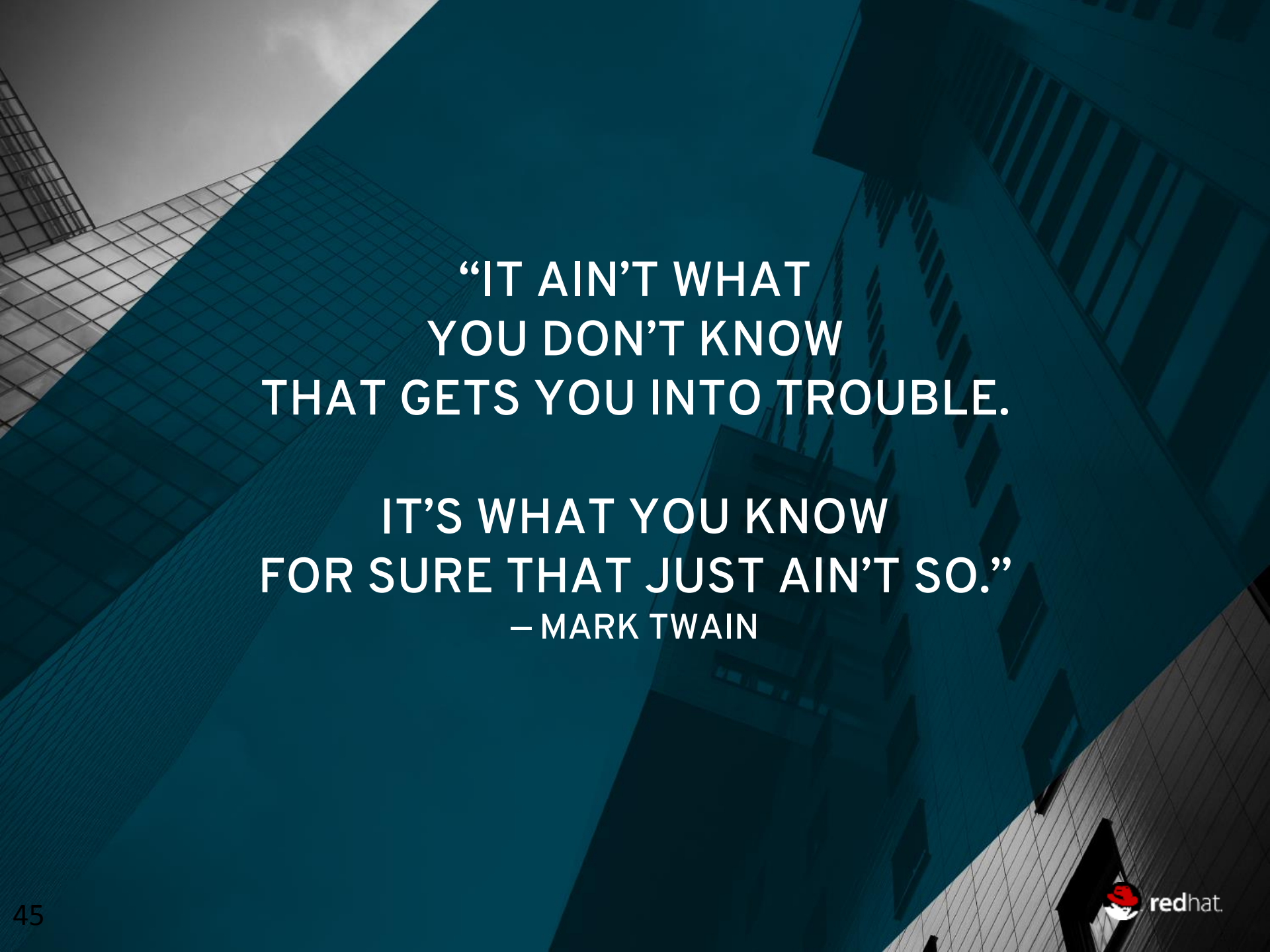
There‘s nothing in Java EE.

- Clustering, Scaling and other features aren‘t specified.
- Completely vendor specific and server dependent.

JUST SECURE CODE ISN‘T ENOUGH

It needs a lot more.




“IT AIN’T WHAT
YOU DON’T KNOW
THAT GETS YOU INTO TROUBLE.
IT’S WHAT YOU KNOW
FOR SURE THAT JUST AIN’T SO.”
– MARK TWAIN

Q&A

FURTHER READING

- http://de.slideshare.net/mraible/java-web-application-security-with-java-ee-spring-security-and-apache-shiro-uberconf-2015
- http://de.slideshare.net/MasoudKalali/top-security-risks-for-java
- https://abhirockzz.wordpress.com/2014/12/04/whats-up-with-java-ee-8/
- https://blogs.oracle.com/theaquarium/entry/jsr_375_java_ee_security
- http://www.infoq.com/news/2014/11/javaee8-security-jsr
- https://docs.oracle.com/javaee/7/tutorial/security-intro001.htm
- http://www.oracle.com/technetwork/java/seccodeguide-139067.html
- http://www.informit.com/store/java-coding-guidelines-75-recommendations-for-reliable-9780133439519
- https://www.owasp.org/index.php/Category:OWASP_Guide_Project
- https://www.owasp.org/index.php/OWASP_Appsec_Tutorial_Series