

Security for Middleware Extensions: Event Meta-Data for Enforcing Security Policy

CBCU

Clinical & Biomedical
Computing Unit - Research
NHS
<http://www.cbcu.nhs.uk/>

ecric
Fighting cancer with information

Eastern Cancer Registration
and Information Centre
<http://www.ecric.nhs.uk/>

Brian Shand and Jem Rashbass

Brian.Shand@cbcu.nhs.uk,
jem@cbcu.nhs.uk

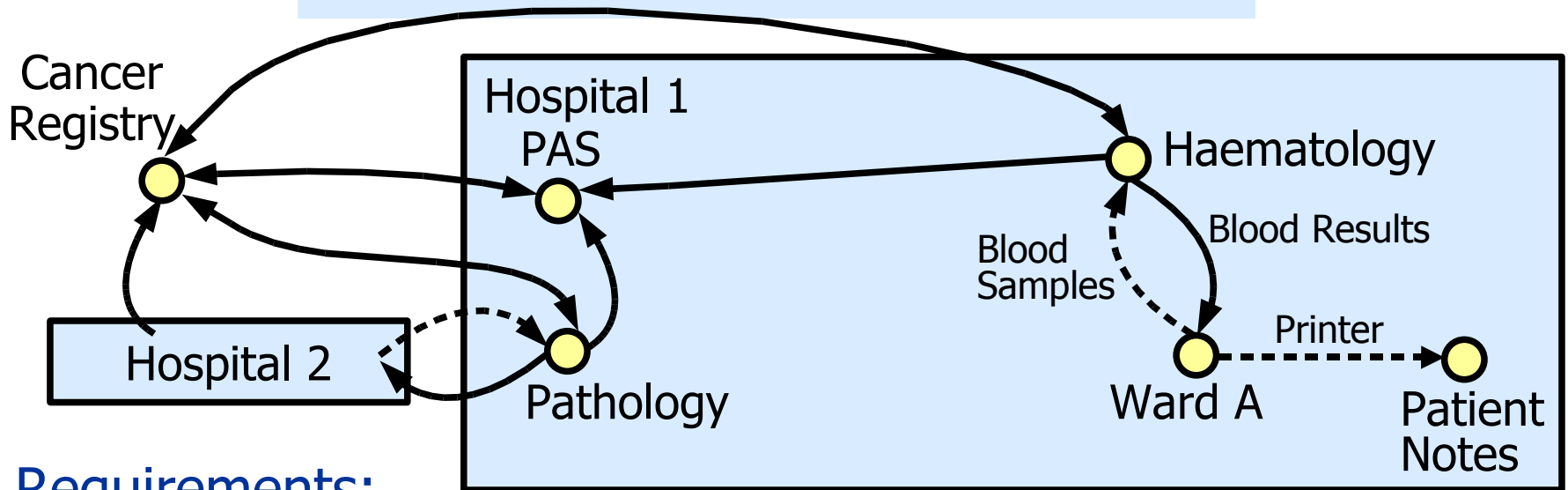
MidSec Workshop, 2 December 2008



Background

- Event-based systems are good for building large distributed applications
 - Simple and robust mechanism – message passing
 - Natural failure modes
- Advantages to extending middleware, instead of building wrappers
 - Simplify application code and improve reliability
 - Potential efficiency gains
- Security risks of multiple middleware extensions
 - Extension interaction, complexity of configuration

Motivating Example: Healthcare



Requirements:

(1) Confidentiality

All clinical data must be encrypted in transit

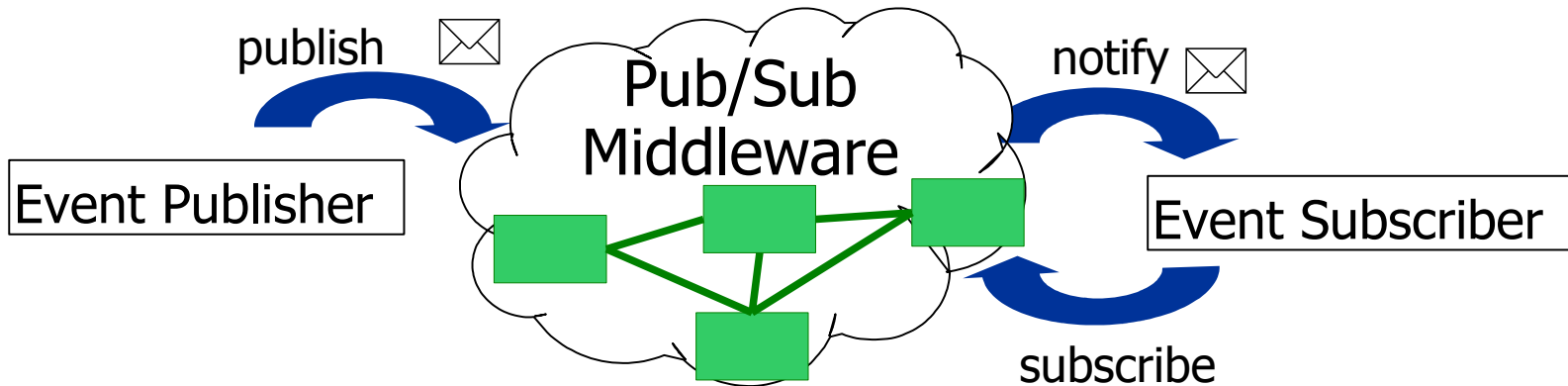
(2) Audit

All communication must be independently audited, including sender, recipient and a secure hash of the data

(3) Access control

Only authorised users may access a patient's data

Extensions in Event-based Middleware



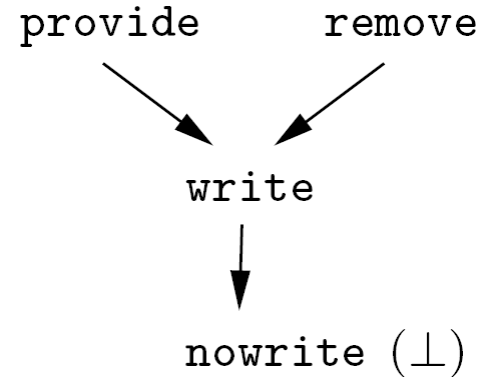
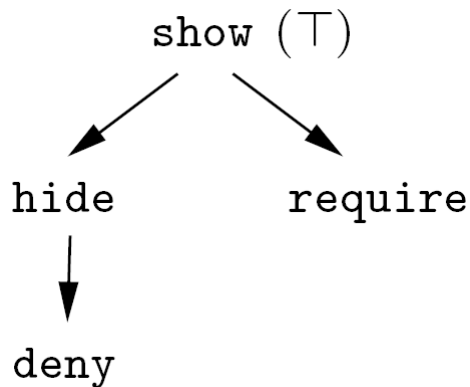
- Event-based middleware allows one-to-one (point-to-point) and/or one-to-many (publish/subscribe) interaction
- Middleware *brokers* process events in transit
- Middleware extensions are located at brokers
 - Extensions intercept and modify events (messages)
 - Extensions & policy specified by appl'n or domain administrators
- Examples of extensions:
 - Composite event detection, delivery notification, message audit₄

Messages and Meta-data

- Message format:
 - (1) Message body (unstructured),
 - (2) Filterable section (for topic- or content-based filtering)
 - (3) Extension meta-data blocks labelled `pubsub.ext.label`
- Examples of meta-data uses:
 - Extension `add_hash` stores a secure hash in `pubsub.ext.hash` meta-data
 - Extension `encryption` deletes the message body, and places encrypted data in `pubsub.ext.encrypted` meta-data
 - For a valid hash, `add_hash` must precede `encryption`.
- Meta-data blocks and extensions are named independently

Read/Write Permissions for Extensions

- *Extension policy* defines how extensions can interact with meta-data.
 - Meta-data read permissions, meta-data write permissions, permission to modify event data
- Partial orders of read, write permissions according to information available, and information that may be written:



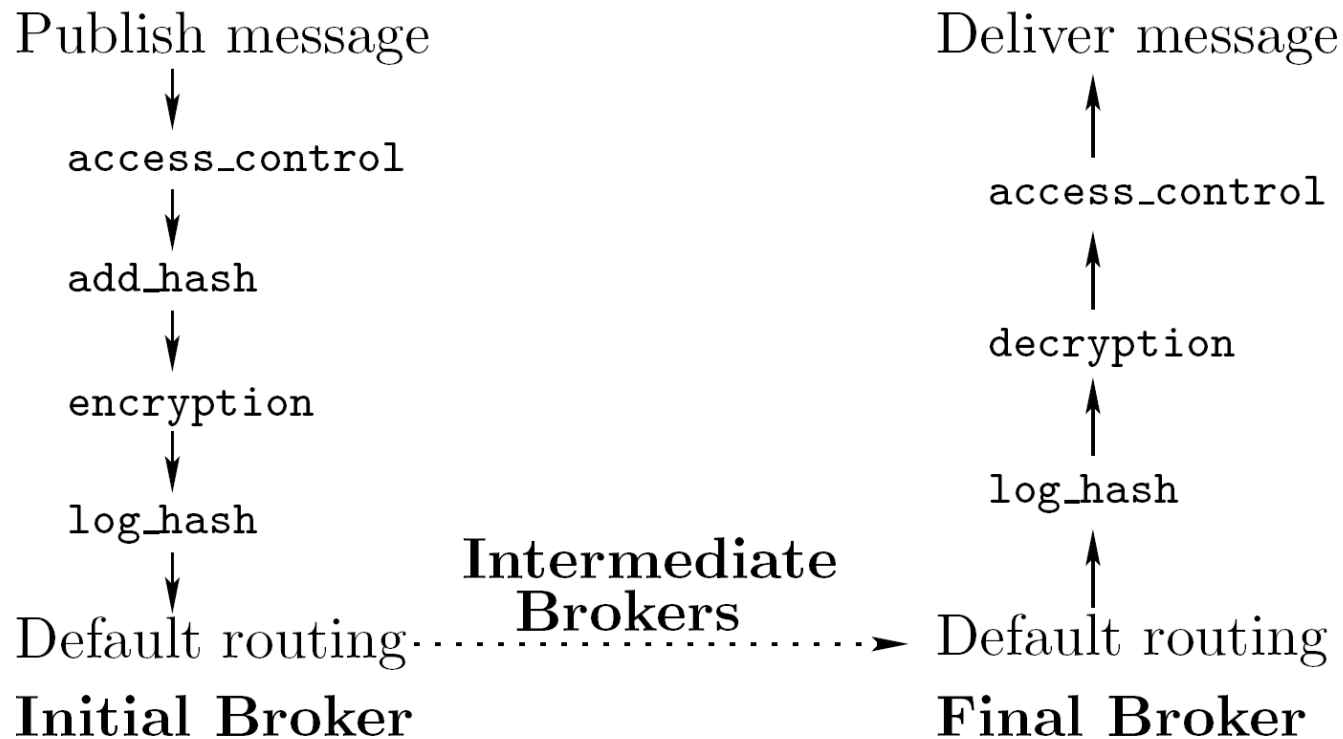
Extension Policy Specification

- Overall middleware policy is determined by combining each extension's policy and any system-wide policy
 - Default meta-data permission: (hide, nowrite)
 - System override policy can resolve conflicts
 - Remaining conflicts are automatically resolved with a minimum privilege assignment, and trigger a warning
 - Read permission $p_r = \min \{p_{r_1}, p_{r_2}, \dots, p_{r_m}\}$
 - Conflicts not resolved trigger an error
- Sample extension policy

```
1: <Extension add_hash>
2: provide pubsub.ext.hash
3: deny pubsub.ext.hash
4: deny pubsub.ext.encrypted
5: </Extension>
```

Application Scenario

- `add_hash` and `log_hash` together securely log message hashes, with no risk of disclosing patient data.
- encryption and decryption use `pubsub.ext.encrypted` to allow policy-based enforcement of privacy protection.
- Extensions can be ordered linearly or as a directed graph.



Extension Policy Checking

- Policy checks help prevent unintended extension interactions that compromise security or reliability
- Class 1 consistency checks: Static checks of each extension's read/write permissions
 - System-wide policy `"show pubsub.ext.hash"` would conflict with `add_hash`'s policy `"deny pubsub.ext.hash"`
- Class 2 consistency checks: Static checks of extension ordering
 - `log_hash` has `"require pubsub.ext.hash"`, and must be preceded by an extension with `"provide pubsub.ext.hash"` or `"write pubsub.ext.hash"`
- Class 3 consistency checks: Dynamic checks of extension ordering
 - Applied where class 2 checks are inconclusive, e.g. `require` vs `write`.

Conclusions

- Event-based middleware is increasingly complex and modular
 - Security policy must be enforced within the middleware and its extensions
- Tagging message meta-data in the middleware provides static and dynamic security guarantees
- Extending meta-data to applications extends the benefits, linking application and middleware security.
- Further work (Smart Flow EPSRC project)
 - Different extension policies for independent data streams
 - Extension sandboxing and federated extension support
 - Strategies for parallelisation of event processing