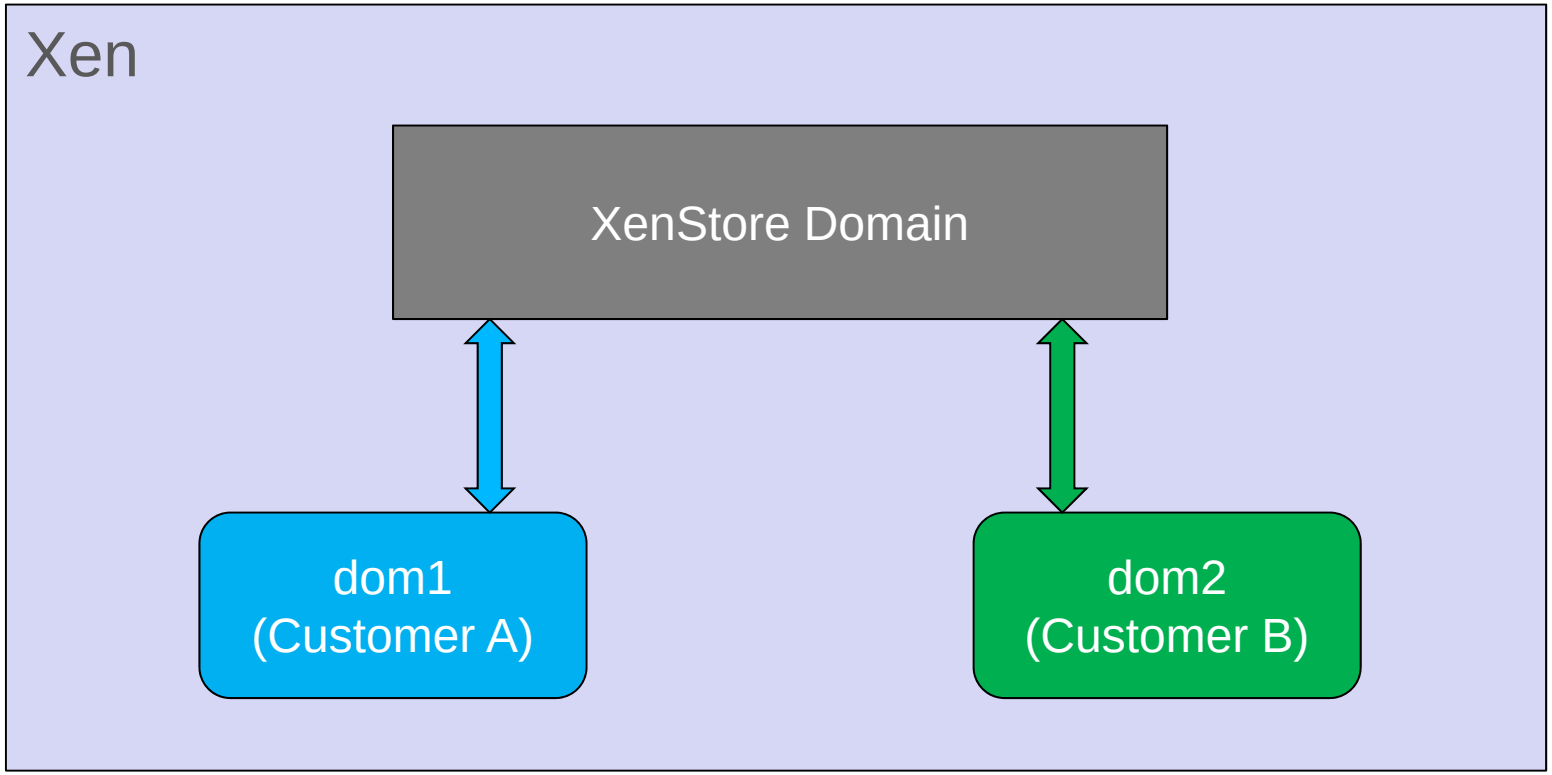
XenStore Mandatory Access Control

James Bielman (jamesjb@galois.com)

Xen Developer Summit | August 18th, 2014

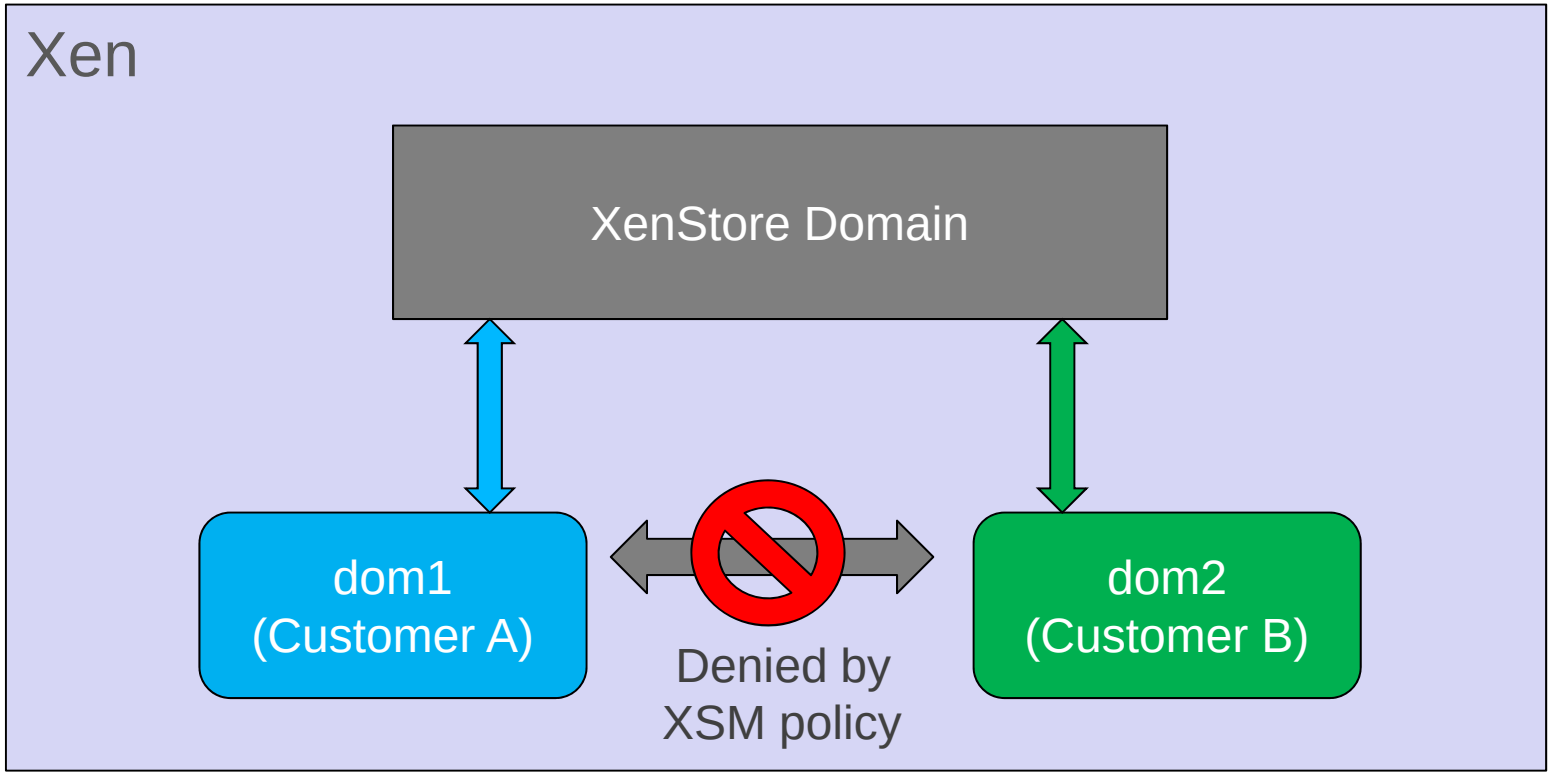
|galois|

Example scenario:
Xen host with two customer VMs



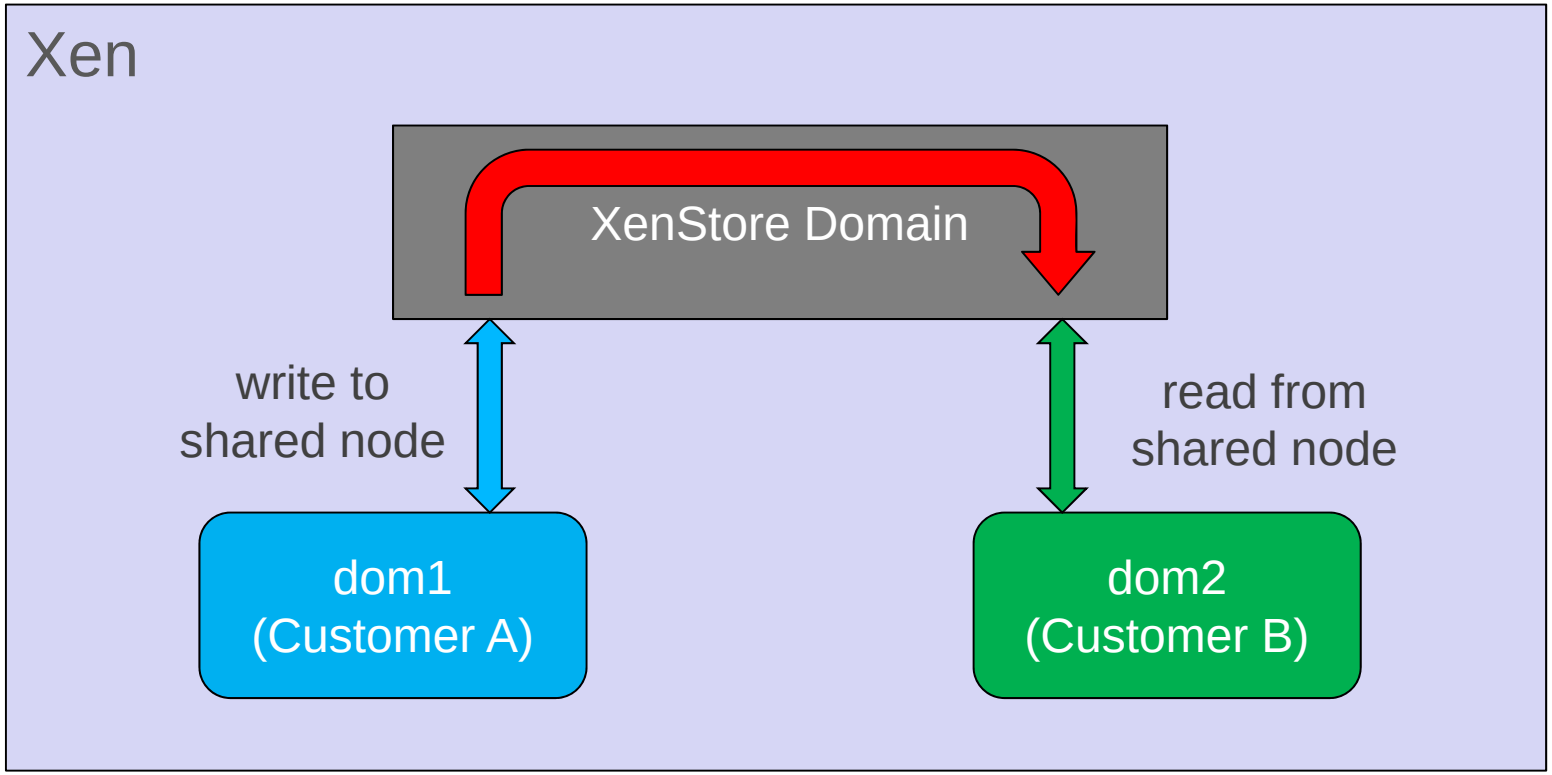
What's the problem?

XSM can prevent direct communication between dom1 and dom2



What's the problem?

But XSM cannot prevent communication via XenStore
if both domains can access the same nodes



What's the problem?

- XenStore allows information to flow between domains that is not under the control of XSM policy.
- In this talk, I will show:
  - Why XenStore's discretionary access control is not enough to prevent this.
  - Our implementation of MAC for Mirage's XenStore.
  - Examples of XenStore security policy.

Roadmap

- Why is XenStore's DAC not enough?
- Why add MAC to XenStore?
- How is XenStore+MAC implemented?
- What does XenStore security policy look like?

Why is XenStore's DAC not enough?

- XenStore's access control is *discretionary*:
  - Nodes are owned by a domain
  - Domains control the permissions of the nodes they own

Why is XenStore's DAC not enough?

- XenStore's DAC allows domains to give away access to nodes.

```
dom1# xenstore-write /local/domain/1/data/x secret
dom1# xenstore-chmod /local/domain/1/data/x n1 r2

dom2# xenstore-read /local/domain/1/data/x
secret
```

Roadmap

- Why is XenStore's DAC not enough?
- Why add MAC to XenStore?
- How is XenStore+MAC implemented?
- What does XenStore security policy look like?

Why add MAC to XenStore?

- With mandatory access control in XenStore, the security policy states which domains may access which XenStore nodes.
- Domains cannot give away access to XenStore nodes they own, unless explicitly allowed by the policy.

How XenStore+MAC works

- All objects are assigned a security label.
  - Xen domains
  - XenStore nodes
- Example labeling:
  - `/local/domain/1/data` `xs_dom1_data_t`
  - `/local/domain/1/data/x` `xs_dom1_data_t`
  - `/local/domain/2/data` `xs_dom2_data_t`

How XenStore+MAC works

- A security policy lists which operations are allowed based on:
  - the subject performing the request
  - the object being accessed
  - the type of access being requested

How XenStore+MAC works

- Example policy:
  - `allow dom1_t xs_dom1_data_t`
    `: xenstore { read write create delete };`
  - `allow dom2_t xs_dom2_data_t`
    `: xenstore { read write create delete };`

How does XenStore+MAC address this threat?

- Each domain has full access to its own “data” nodes, but there is no cross-domain access allowed by the policy.
- Domains cannot grant additional access beyond the policy, so the communication channel between dom1 and dom2 is closed.

Roadmap

- Why is XenStore's DAC not enough?
- Why add MAC to XenStore?
- How is XenStore+MAC implemented?
- What does XenStore security policy look like?

XenStore+MAC: Platform

- Work is based on Mirage's XenStore
- Why Mirage?
  - Mirage is under active development and contains a solid XenStore implementation.
  - A unikernel is a good fit for XenStore.

XenStore+MAC: Nested Security Server

- XenStore security policy should be managed by XenStore, not the hypervisor.
- But XenStore policy must be able to reference Xen domains from the hypervisor's XSM policy.

XenStore+MAC: Nested Security Server

- XenStore is responsible for loading its policy and performing security checks against it.
- A “context database” describes how to map security labels from the parent XSM policy to XenStore policy.

XenStore+MAC: Low-Level Design

- Modular design:
  - added security hooks to core XenStore module
  - added security checks against installed hooks
  - same approach taken by LSM in Linux and XSM in Xen
- Default “dummy” security hooks revert to existing DAC-only behavior.

XenStore+MAC: xenstore-flask module

- Uses `libsepol` to load and perform access checks against an SELinux binary policy.
- The SELinux policy compiler (`checkpolicy`) is used without modification to compile policy.
- The binary policy is augmented with:
  - a path database used for labeling
  - a context database to import security IDs from XSM

Roadmap

- Why is XenStore's DAC not enough?
- Why add MAC to XenStore?
- How is XenStore+MAC implemented?
- What does XenStore security policy look like?

Policy examples: Node Labeling

Node Path	Security Label
/	xs_root_t
/local	xs_root_t
/local/domain	xs_local_domain_t
/local/domain/1	xs_dom1_ctl_t
/local/domain/1/data	xs_dom1_data_t
/local/domain/2	xs_dom2_ctl_t
/local/domain/2/data	xs_dom2_data_t

Policy example: Node labeling

- By default, new nodes inherit their security label from parent node.
- To override the default behavior:
  - Assign the path a label in the path database.
  - Add a `type_transition` statement to the policy assigning the node a label based on the parent and path labels.

Policy example: Node labeling

Path database: Specifies what kind of type transition is used to label a new node.

```
# type   path                    label
ctx      /local/domain           xs_local_domain_path_t
ctx      /local/domain/*/data    xs_domain_data_path_t
dom      /local/domain/*
```

Policy example: Node labeling

Labeling policy: Specifies new node label based on path type and the parent node label or domain ID from the path.

```
type_transition xs_root_t xs_local_domain_path_t
  : xenstore xs_local_domain_t;
type_transition xs_dom1_ctl_t xs_domain_data_path_t
  : xenstore xs_dom1_data_t;
type_transition dom1_t xs_local_domain_t
  : xenstore xs_dom1_ctl_t;
```

Policy example: Node labeling

Path database: Specifies what kind of type transition is used to label a new node.

```
# type   path                  label
ctx      /local/domain         xs_local_domain_path_t
ctx      /local/domain/*/data  xs_domain_data_path_t
dom      /local/domain/*
```

Policy example: Node labeling

Labeling policy: Specifies new node label based on path type and the parent node label or domain ID from the path.

```
type_transition xs_root_t xs_local_domain_path_t
  : xenstore xs_local_domain_t;
type_transition xs_dom1_ctl_t xs_domain_data_path_t
  : xenstore xs_dom1_data_t;
type_transition dom1_t xs_local_domain_t
  : xenstore xs_dom1_ctl_t;
```

Policy example: Node labeling

Path database: Specifies what kind of type transition is used to label a new node.

```
# type   path                    label
ctx      /local/domain           xs_local_domain_path_t
ctx      /local/domain/*/data    xs_domain_data_path_t
dom      /local/domain/*
```

Policy example: Node labeling

Labeling policy: Specifies new node label based on path type and the parent node label or domain ID from the path.

```
type_transition xs_root_t xs_local_domain_path_t
  : xenstore xs_local_domain_t;
type_transition xs_dom1_ctl_t xs_domain_data_path_t
  : xenstore xs_dom1_data_t;
type_transition dom1_t xs_local_domain_t
  : xenstore xs_dom1_ctl_t;
```

Policy example: Binding

- The security policy can define the shape of the XenStore tree by how the “bind” permission is allowed.

Policy example: Binding

Newly created nodes must be allowed to bind to their parent.

```
#     parent              child
allow xs_root_t           xs_local_domain_t : xenstore bind;
allow xs_local_domain_t   xs_dom1_ctl_t     : xenstore bind;
allow xs_dom1_ctl_t       xs_dom1_data_t    : xenstore bind;
```

Policy example: Macros

- M4 macros are used to capture common patterns in policy such as connecting XenStore nodes for device frontends and backends.

Policy example: Macros

Connecting device frontends in domU domains to a driver domain with M4 macros:

```
# Privileged dom0, driver domain:
xs_domain(dom0)
xs_device_backend(dom0, vbd)

# domU created by, and using devices from, dom0:
xs_domain(domU)
xs_control_domain(domU, dom0)
xs_device(domU, dom0, vbd)
```

Policy example: Macros

These macros expand into policy setting up permissions for the front and back driver domains:

```
# Allow the frontend domain to read backend nodes.
allow domU_t xs_vbd_backend_for_domU_t : xenstore { read };
# Allow the frontend domain to write frontend nodes.
allow domU_t xs_domU_vbd_frontend_t : xenstore { write create };
# Allow the backend domain to read frontend nodes.
allow dom0_t xs_domU_vbd_frontend_t : xenstore { read };
```

Summary

- MAC is cool; Xen already supports it via XSM.
- XenStore should be secured with MAC as well.
- Get our MAC implementation for Mirage XenStore:
  - `opam repo add https://github.com/GaloisInc/opam-repo.git`
  - `opam install mirage`
  - `git clone https://github.com/GaloisInc/ocaml-xenstore-xen.git`
- Our goal is to merge these changes upstream to Mirage's XenStore.