Xen Security Modules (XSM)

George Coker
National Information Assurance Research Lab
National Security Agency (NSA)
gscoker@alpha.ncsc.mil
What is XSM?

• A generalized security framework for Xen
  – Allows custom security functionality in modules
  – Creates general security interfaces for Xen
  – Removes security model specific code from Xen
Framework Interface Goals

• Capable of supporting known security models
  – hard to be "complete"
• Minimize impact on Xen
• Config enable/disable for Xen
Rational for XSM

- New usage models for Xen have different security goals
- Shouldn't "hardwire" security model
  - Xen should be capable of supporting many through configuration
- New security functionality without changes to Xen mechanisms
New Usage Models

• Decomposed dom0
  – Removal of all-powerful dom0?
    • Least privilege for each domain
      – e.g. separation of platform/hardware config and domain building privileges
    • Security module could be created to define these new privileges?
New Usage Models (cont.)

• Resource partitions
  – How are resources partitioned and controlled?
    • Ability to allow certain domains to control resource allocations
      – e.g. multiple domain builders
    • Security module could be defined to control allocations?
New Usage Models (cont.)

- Protection of the platform from third party software
  - How are resources partitioned and controlled?
    - e.g. device driver isolation
    - e.g. sandboxing
  - Security module could be defined to mediate accesses across domains
New Usage Models (cont.)

• Protection for core platform security services
  – How to safely create platform wide services?
    • e.g. media encryption
    • e.g. IP-filtering/routing
    • e.g. measurement & attestation
  – Security module could be defined to isolate, mediate access to, and guarantee invocation of services
XSM Security Benefits

• Encapsulation of security functionality
  – A well-defined security architecture is required for a well-defined TCB
    • common criteria

• Extensible security functionality
  – e.g. trusted IVC
    • security modules can enable security identification of communications channels or peers without changing the implementation of existing channel mechanisms (grants and events)
XSM Implementation

- Derived from Linux Security Modules (LSM)
  - Linux 2.6.13.4
- Security function infrastructure
  - Derived from ACM
  - New security functionality
- Security module
  - Implements security hooks
  - Specific to a security model
XSM Today

- 57 hooks to date
  - 60% complete (estimate)
  - Target privileged hypercalls (initially)
  - Comprehensive hook placement
    - attempts to anticipate new modules
XSM Specifics

• Early initialization
  – Prior to idle domain creation

• Early allocation/late deallocation of domain security structures
  – domain_create
  – domain_destroy
XSM Specifics (cont.)

- No excepted event channels
  - evtchn_init
  - evtchn_bind_interdomain
- Will not support stacking
  - Security module behavior cannot be predicted under stacking and may violate the goals of the security module while meeting few or none of the goals of the user
  - Stacking should be a property of the security module.
XSM Modules

• Registered and linked in at boot
• Modules may register a security hypercall
• Modules may register a policy magic number to identify and load a policy from boot
Existing XSM Modules

- Dummy (XSM default)
- ACM/sHype (IBM)
- Flask (NSA)
XSM Hooks

• What are the hooks doing?
  – Interpose on code path
  – Allocation/setting of security structures
  – Platform security initialization
Hook Placement Philosophy

• Positioned at key locations
  – Identified by analysis
    • Availability of security relevant objects
    • Safety of hook location in code path
    • General benefit to security
    • Comprehensive placement
  – Localized in code path
    • Balance between hook placement and maintenance
Hook Placement Philosophy (cont.)

- Minimize impact to Xen code paths
  - Leverage existing exit/error paths wherever possible
- Rely on calling function to hold references to objects
  - Safer for Xen if security modules do not hold references to Xen objects
Current Hook Locations

- dom0_ops.c
- domain.c
- grant_table.c
- event_channel.c
- setup.c
- mm.c
First Class Security Objects

• Generic security pointers on Xen first class objects
  - e.g. struct domain & struct evtchn
  - Allocation/access via hook functions
    • Only if required by module

• Modules may hold security labels on other platform resources that Xen does not manage
  - e.g. physical interrupts
Performance

- Are more checks more expensive?
  - Small constant XSM overhead per hook
  - Premise "basic" call/return is a minimal overhead
  - Extra overhead for hooks is module specific

- Presently no "observed" degradation
  - Further investigation required
ACM Module

• How will it affect sHype/ACM?
  – sHype/ACM will plug into XSM hooks
  – Changes are transparent to sHype management / use
  – sHype/ACM will support a single policy (CHWALL/STE)
ACM Module Implementation

• Little modification required to turn into XSM module
  – modifications in Xen code only!
  – no change to user-space tool chain

• TODO: Nativization cleanups
  – Better use of hook references
  – Remove refactored functionality
Flask Module

- Xen nativization of Flask security code
  - Linux 2.6.13.4
- Capabilities
  - RBAC/TE
  - MLS/MCS
- Security server optimized for small memory footprint
  - Memory footprint comes from number of types, not number of permissions or hooks used
Flask Policy

- Uses existing SELinux policy language
  - Common policy generation and analysis toolchain
- Xen policy is less complex than Linux
  - Fewer security controls
How does Flask use XSM?

- **Event Channels**
  - Fine-grain allocation of physical interrupts (example)
- **Grant Tables**
  - Fine-grain sharing between domains (example)
- **Dom0 Operations**
  - Fine-grain allocation of io resources (example)
- **MMU**
  - Fine-grain control of foreign mappings (example)
Event Channels (example)

static int flask_evtchn_pirq(struct domain *d, struct evtchn *chn, int pirq)
{
    u32 newsid;
    u32 psid;
    int rc;
    struct domain_security_struct *dsec;
    struct evtchn_security_struct *esec;

dsec = d->ssid;
esec = chn->ssid;

rc = security_pirq_sid(pirq, &psid);
if (rc)
    return rc;

rc = security_transition_sid(dsec->sid, psid, SECCLASS_EVENT,
                           &newsid);
if (rc) {
    printk("%s: security_transition_sid failed, rc=%d (pirq=%d)\n",
           __FUNCTION__, -rc, pirq);
    return rc;
}

rc = avc_has_perm(dsec->sid, newsid, SECCLASS_EVENT,
                  EVENT__CREATE, NULL);
if (rc)
    return rc;
rc = avc_has_perm(newsid, psid, SECCLASS_EVENT,
                  EVENT__BIND, NULL);
if (rc)
    return rc;
esec->sid = newsid;
return rc;
}
Grant Tables (example)

static int flask_grant_mapref(struct domain *d1, struct domain *d2, uint32_t flags)
{
    u32 perms = GRANT__MAP_READ;

    if (flags & GTF_writing)
        perms |= GRANT__MAP_WRITE;

    return domain_has_perm(d1, d2, SECCLASS_GRANT, perms);
}
Dom0 Operations (example)

static int flask_iomem_permission(struct domain *d,
    unsigned long mfn, uint8_t access)
{
    u32 perm;
    u32 rsid;
    int rc = -EPERM;

    struct domain_security_struct *ssec, *tsec;

    rc = domain_has_perm(current->domain, d, SECCLASS_RESOURCE,
        resource_to_perm(access));

    if (rc)
        return rc;

    if (access)
        perm = RESOURCE__ADD_IOMEM;
    else
        perm = RESOURCE__REMOVE_IOMEM;

    ssec = current->domain->ssid;
    tsec = d->ssid;

    rc = security_iomem_sid(mfn, &rsid);
    if (rc)
        return rc;

    rc = avc_has_perm(ssec->sid, rsid, SECCLASS_RESOURCE, perm, NULL);
        if (rc)
            return rc;

    return avc_has_perm(tsec->sid, rsid,
        SECCLASS_RESOURCE, RESOURCE__USE, NULL);
}

static inline u32 resource_to_perm(uint8_t access)
{
    if (access)
        return RESOURCE__ADD;
    else
        return RESOURCE__REMOVE;
}

static inline u32 resource_to_perm(uint8_t access)
{
    if (access)
        return RESOURCE__ADD;
    else
        return RESOURCE__REMOVE;
}
static int flask_mmu_normal_update(struct domain *d, intpte_t fpte)
{
    u32 map_perms = MMU__MAP_READ;
    unsigned long fmfn;
    struct page_info *fpage;
    struct domain *fd;
    u32 fsid;
    struct domain_security_struct *dsec, *fsec;
    dsec = d->ssid;

    if ( get_pte_flags(fpte) & _PAGE_RW )
        map_perms |= MMU__MAP_WRITE;

    fmfn = ((unsigned long)(((fpte) &
        (PADDR_MASK&PAGE_MASK)) >> PAGE_SHIFT));
    if (mfn_valid(fmfn)) {
        /*fmfn is valid if this is a page that Xen is tracking!*/
        fpage = mfn_to_page(fmfn);
        fd = page_get_owner(fpage);
    } else {
        /*possibly an untracked IO page?*/
        map_perms |= MMU__MAP_ANONYMOUS;
        fd = d;
    }

    switch ( fd->domain_id )
    {
    case DOMID_IO:
        fsid = SECINITSID_DOMIO;
        break;
    case DOMID_XEN:
        fsid = SECINITSID_DOMXEN;
        break;
    default:
        fsec = fd->ssid;
        fsid = fsec->sid;
    }

    return avc_has_perm(dsec->sid, fsid, SECCLASS_MMU,
            map_perms, NULL);
}
Next Steps

- Submit remaining hooks & flask module updates
- Discuss XSM with community and argue for acceptance
Next Steps (cont.)

- Experimentation with new architectures for security
  - Xen for security vs. security of Xen
- Security in user-space
  - Xen control plane
  - decomposed dom0
Questions?

George Coker
National Information Assurance Research Lab
National Security Agency (NSA)
gscoker@alpha.ncsc.mil