



OpenSolaris Secure Deployment: Role-Based Access Control and the Cryptographic Framework

Session Code: S304261

Christoph Schuba

Senior Research Staff, Solaris Security

Agenda

- Role-Based Access Control (RBAC)
- Cryptographic Framework
- Featured OpenSolaris Security Projects
 - FMAC
 - Crypto ZFS
 - Validated Execution

Agenda

- **Role-Based Access Control (RBAC)**
- Cryptographic Framework
- Featured OpenSolaris Security Projects
 - FMAC
 - Crypto ZFS
 - Validated Execution

The problem

- Traditional Unix superuser model:
root vs. non-root
 - too coarse grained
 - root too powerful

- Principle of Least Privilege:

“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”

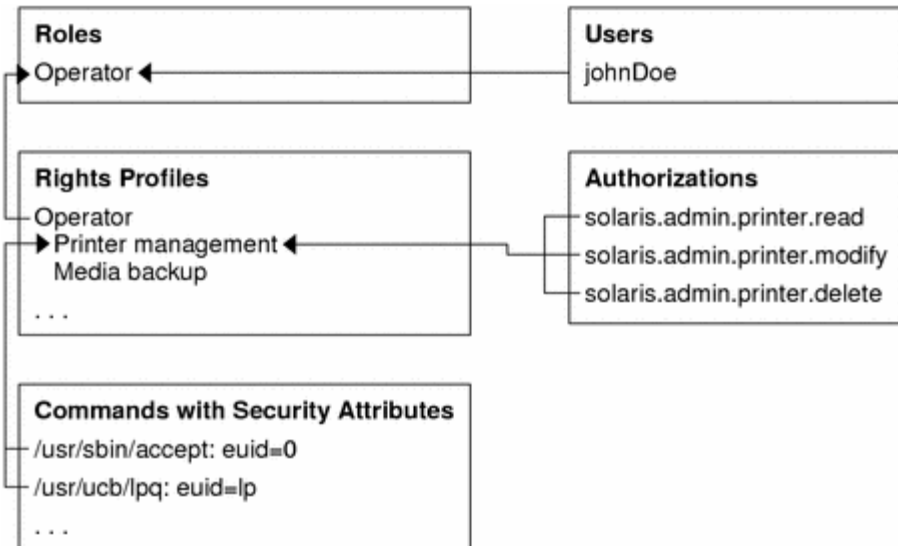
- Protection of data and functionality
 - from faults (fault tolerance)
 - malicious behavior (computer security)

Solaris Role-based Access Control (RBAC)

➤ Goal:
Assign privileged functions
to specific user accounts

➤ Elements:

- Role designations
- Rights profiles
- Authorizations



Authorizations

- Unique string that represents a user's right to perform some operation or class of operations
 - `solaris.admin.printer.read`
 - `solaris.admin.printer.modify`
 - `solaris.admin.printer.delete`
- Defined in database:
`/etc/security/auth_attr`
- Check authorization using
`chkauthattr(3SECDB)`

Authorization Characterizations

```
[global 0]: $ cat /etc/security/auth_attr
solaris.:::All Solaris Authorizations::help=AllSolAuthsHeader.html
[..]
solaris.grant.:::Grant All Solaris Authorizations::help=PriAdmin.html
[..]
solaris.system.:::Machine Administration::help=SysHeader.html
solaris.system.date.:::Set Date & Time::help=SysDate.html
solaris.system.shutdown.:::Shutdown the System::help=SysShutdown.html
[..]
```

- used by admin tools
 - usermod, rolemod, SMC (Solaris Management Console) to check validity of an auth

Mapping of authorizations to users

> Database `policy_conf(4)`

```
[global 0]: # more /etc/security/policy.conf
[...]  
AUTHS_GRANTED=solaris.device.cdrw  
[...]
```

> Database `user_attr(4)`

```
[global 0]: # more /etc/user_attr  
[...]  
root:::type=role;auths=solaris.*,solaris.grant;profiles=All;\br/>    lock_after_retries=no;min_label=admin_low;clearance=admin_high  
[...]
```

Rights Profiles

- A mechanism used to bundle together
 - the commands and
 - the authorizationsneeded to perform a specific function

Mapping of profiles to users

> Database `policy_conf(4)`

```
[global 0]: # more /etc/security/policy.conf
[...]  
  
PROFS_GRANTED=Basic Solaris User  
[...]
```

> Database `user_attr(4)`

```
[global 0]: # more /etc/user_attr
[...]  
  
root:::type=role;auths=solaris.*,solaris.grant;profiles=All;\
  lock_after_retries=no;min_label=admin_low;clearance=admin_high  
  
jdoe:::profiles=Primary Administrator;roles=root  
[...]
```

> Database `prof_attr(4)`

```
[global 0]: # more /etc/security/prof_attr
[...]  
  
Primary Administrator:::Can perform all administrative tasks:\
  auths=solaris.*,solaris.grant;help=RtPriAdmin.html  
[...]
```

Users and Roles

- Roles are based on rights profiles
(`roleadd(1M)`)
- Roles are assigned to users who are trusted to perform the tasks of the role. E.g.,
`usermod -r`
- Users log in with their user name
- Users assume roles that can run restricted commands
(via `su`)
or
- Users use `pfexec`

Admin. Primary Administrator Profile

```
[global 1]: $ whoami
jdoe

[global 1]: $ profiles
Primary Administrator
Console User
Basic Solaris User
All

[global 1]: $ auths
solaris.*

[global 1]: $ grep jdoe /etc/user_attr
jdoe:::type=normal;profiles=Primary Administrator;roles=root

[global 1]: $ pfexec usermod -P "" jdoe
UX: usermod: jdoe is currently logged in, some changes may
not take effect until next login.

[global 1]: $ grep jdoe /etc/user_attr
jdoe:::type=normal;roles=root

[global 1]: $ auths
solaris.device.cdrw,solaris.profmgr.read,solaris.jobs.user,\
solaris.mail.mailq,solaris.device.mount.removable,\
solaris.admin.usermgr.read,solaris.admin.logsvc.read,\
solaris.admin.fsmgr.read,solaris.admin.serialmgr.read,\
solaris.admin.diskmgr.read,solaris.admin.procmgr.user,\
solaris.compsys.read,solaris.admin.printer.read,\
solaris.admin.prodreg.read,solaris.admin.dcmgr.read,\
solaris.snmp.read,solaris.project.read,solaris.admin.patchmgr.read,\
solaris.network.hosts.read,solaris.admin.volmgr.read
```

Admin. Primary Administrator Profile (cont.)

```
[global 1]: $ pexec usermod -P "Primary Administrator" jdoe
X: usermod: ERROR: Permission denied.

[global 1]: $ auths|grep grant

[global 1]: $ su root
Password:

[global 1]: # whoami
root

[global 1]: # usermod -P "Primary Administrator" jdoe
UX: usermod: jdoe is currently logged in, some changes may not take \
effect until next login.

[global 1]: # exit
exit

[global 1]: $ grep jdoe /etc/user_attr
jdoe:::type=normal;profiles=Primary Administrator;roles=root

[global 1]: $ auths
solaris.*
```

Programmatic RBAC checks

➤ Synopsis: `chkauthattr(3SECDB)`

```
int chkauthattr(const char *authname,  
                const char *username);
```

➤ The `chkauthattr()` function checks the

- `policy.conf(4)`
- `user_attr(4)`
- `prof_attr(4)`

databases for a match to the given authorization

Legacy root check

```
ruid = getuid();  
  
if (ruid != 0) {  
    crabort(NOTROOT);  
}
```

Authorization check

```
ruid = getuid();  
if ((pwp = getpwuid(ruid)) == NULL)  
    crabort(INVALIDUSER);  
if (!chkauthattr("solaris.jobs.admin", pwp->pw_name)) {  
    crabort(NOTROOT);  
}
```

Privileges and RBAC

- Complementary and compatible
- Rights profiles, privileges, and authorizations can be assigned directly to users
- Privileges and authorizations can be assigned directly to roles

Agenda

- Role-Based Access Control (RBAC)
- **Cryptographic Framework**
- Featured OpenSolaris Security Projects
 - FMAC
 - Crypto ZFS
 - Validated Execution

Cryptographic Framework - Motivation

- Crypto used in lots of places:
 - Userland: SSL, SSH, GSS, Kerberos, IKE, PAM, ...
 - Kernel: IPsec (AH, ESP), WiFi drivers, lofi, KSSL, ...
- Problems with duplicate implementations/libs
 - OpenSSL, Mozilla NSS,...
- Problems with versioning and software defects
- Administrative challenge to utilize hardware acceleration
- Need to take advantage of standard APIs
- Need for pluggable interface

Solaris Cryptographic Framework

- Introduced in Solaris 10
- New features added on ongoing basis
 - E.g., niagara2 crypto provider, ECC algorithms/modes
- User-level and kernel-level interfaces
- Administrative commands control the framework
- Consumer-producer architecture
- Benefits:
 - no duplicate implementations
 - optimization for hardware
 - transparent use of sw providers & hw accelerators

Typical Default Providers

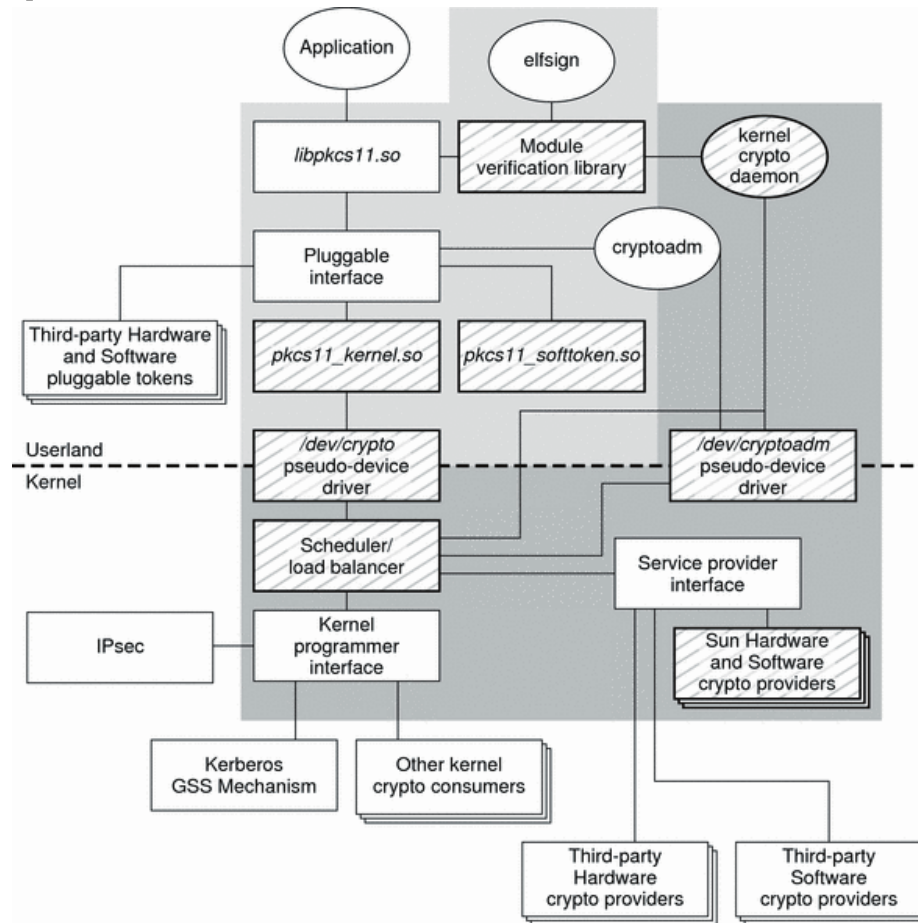
```
[global 0]: $ cryptoadm list
User-level providers:
Provider: /usr/lib/security/$ISA/pkcs11_kernel.so
Provider: /usr/lib/security/$ISA/pkcs11_softtoken.so
```

Kernel software providers:

```
des
aes
arcfour
blowfish
ecc
sha1
sha2
md4
md5
rsa
swrand
```

Kernel hardware providers:

Solaris Crypto Framework Architecture



- Private components
- User portion of cryptographic framework
- Kernel portion of cryptographic framework

Terminology

- PKCS#11 Standard API from RSA Labs
- Consumers: application, library, or kernel using cryptographic functions
 - E.g., userland: digest, mac, encrypt, decrypt, IKE, pktool, ...
 - E.g., kernel: AH/ESP, WiFi drivers, ZFS/lofi crypto, ...
- Producers: library, kernel, or hardware offering implementation of cryptographic functions
- Plug-in: aka producer
- Mechanism: algorithm, mode, and security goal
 - E.g., CKM_AES_CBC, CKM_AES_CTR, CKM_SHA256_RSA_PKCS, CKM_SHA256_HMAC
→ determines features and required arguments

Terminology (cont.)

- Token: implementation of a mechanism
 - can be stateful
 - collection of PKCS#11 mechanisms
 - maybe some storage for keys and other data
 - hardware or software, often hybrid
- Slot: place a token is found
- Metaslot: virtual/default slot, superset of all slots
 - selects best implementation
 - (Open)Solaris-specific feature
- Session: active connection between consumer and token
- Objects: store information (e.g., keys)
 - session objects: destroyed on application exit
 - token objects: persistent across sessions

uCF - User-Level Crypto Framework

- libpkcs11.so is your friend!
- API version 2.20
- Convenience functions. E.g.,
 - SUNW_C_GetMechSession
 - SUNW_C_KeyToObject
- Engine for OpenSSL library
- Digest library
 - libmd.so

Software Token Provider

```
[global 0]: $ man pkcs11_softtoken  
[..]
```

```
CKM_AES_KEY_GEN  
CKM_AES_ECB  
CKM_AES_CBC  
CKM_AES_CBC_PAD  
[..]
```

```
CKM_MD5_RSA_PKCS  
CKM_MD5  
CKM_MD5_HMAC  
CKM_MD5_HMAC_GENERAL  
CKM_MD5_KEY_DERIVATION  
[..]
```

Administration and Commands

- Crypto Framework managed via SMF
- Crypto Framework administration
 - cryptoadm(1M)
- Provider verification
 - elfsign(1)
- Setup of CF threadpool and Runtime verification
 - kcfd(1M)
- User-level commands
 - digest(1), encrypt(1), decrypt(1), mac(1)

Common CF Command Usage

```
[global 1] $ digest -l
sha1
md5
sha256
sha384
sha512

[global 1] $ digest -a sha1 /etc/passwd
4a02fee38d5daf1e56fe1098dde4b3b032999a59
```

```
[global 1] $ encrypt -l
Algorithm      Keysize:  Min    Max (bits)
-----
aes            128     256
arcfour        8       2048
des            64      64
3des          192     192
```

➤ Key generation/encryption using a file-based keystore

```
[global 1] $ pktool genkey keystore=file keytype=aes keylen=192 outkey=key
[global 1] $ encrypt -k key -a aes -i /etc/passwd -o passwd.enc
```

Common CF Command Usage (cont.)

➤ Key generation/encryption using a token-based key

```
[global 1] $ pktool tokens
Token Label                               Manuf ID           Serial No          PIN State
Sun Software PKCS#11 softtoken           Sun Microsystems   default

[global 1] $ pktool setpin token="Sun Software PKCS#11 softtoken"
Enter token passphrase: changeme
Create new passphrase: abc
Re-enter new passphrase: abc
Passphrase changed.

[global 1] $ pktool genkey token="Sun Software PKCS#11 softtoken" \
  keytype=3des keylen=192 label=key2
Enter PIN for Sun Software PKCS#11 softtoken: abc

[global 1] $ pktool list token="Sun Software PKCS#11 softtoken" objtype=key
Enter PIN for Sun Software PKCS#11 softtoken: abc
Found 1 symmetric keys.
Key #1 - Triple-DES:  key2 (192 bits)

[global 1] $ encrypt -a 3des -T "Sun Software PKCS#11 softtoken" -K key2 \
  -i /etc/passwd -o passwd2.enc
Enter PIN for Sun Software PKCS#11 softtoken: abc
```

kCF - Kernel-Level Crypto Framework

- Similar programming logic to PKCS#11
- Kernel-Level software providers
 - loadable kernel modules
 - kernel-level software providers are synchronous
- Kernel-Level hardware providers
 - device nodes and drivers
 - kernel-level hardware providers are usually asynchronous
 - framework handles scheduling, callbacks, state, etc.

Example Kernel Software Provider

```
[global 0]: $ modinfo | grep -i swrand
```

```
80 f91fe000 1914 - 1 swrand (Kernel Random number Provider)
```

```
[global 0]: $ cryptoadm list -mv
```

```
[..]
```

| mechanism name | min | max | H | E | D | D | S | S | V | V | K | P | U | D | E |
|------------------------|-------|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ----- | ----- | ----- | W | n | e | i | i | g | r | r | e | a | n | r | C |
| | | | - | - | - | - | - | - | - | - | - | - | - | - | - |
| CKM_AES_CBC | 16 | 32 | . | X | X | . | . | . | . | . | . | . | X | X | . |
| CKM_AES_CBC_PAD | 16 | 32 | . | X | X | . | . | . | . | . | . | . | X | X | . |
| CKM_AES_ECB | 16 | 32 | . | X | X | . | . | . | . | . | . | . | X | X | . |
| CKM_AES_KEY_GEN | 16 | 32 | . | . | . | . | . | . | . | . | X | . | . | . | . |
| [..] | | | | | | | | | | | | | | | |
| CKM_MD5 | 0 | 0 | . | . | . | X | . | . | . | . | . | . | . | . | . |
| CKM_MD5_HMAC | 1 | 64 | . | . | . | . | X | . | X | . | . | . | . | . | . |
| CKM_MD5_HMAC_GENERAL | 1 | 64 | . | . | . | . | X | . | X | . | . | . | . | . | . |
| CKM_SSL3_MD5_MAC | 1 | 512 | . | . | . | . | X | . | X | . | . | . | . | . | . |
| CKM_MD5_RSA_PKCS | 256 | 4096 | . | . | . | . | X | . | X | . | . | . | . | . | . |
| CKM_MD5_KEY_DERIVATION | 1 | 16 | . | . | . | . | . | . | . | . | . | . | . | . | X |
| [..] | | | | | | | | | | | | | | | |

Agenda

- Solaris Privileges and
Role-Based Access Control (RBAC)
- Cryptographic Framework
- **Featured OpenSolaris Security Projects**
 - **FMAC**
 - **Crypto ZFS**
 - **Validated Execution**

Flexible Mandatory Access Control (FMAC)

- Joint project. Initiated by NSA and Sun
- Incorporate Flask architecture and Type Enforcement (TE) into the OpenSolaris™ Operating System
- Investigate how to best marry Solaris Trusted Extensions and Type Enforcement
- Complement existing Solaris™ security mechanisms
- Preserve existing Solaris APIs
- Provide Flask-compatible APIs
- Specify a single policy for a system

FMAC Status

➤ Project foundation

- discussion list, mercurial gate, contribution charter

➤ Development foundation

- design discussions
- Flask/TE v15 integration into project repo (based on NV101)
- Process and file context support
- Default context support
- Library and utility support
- RBAC mashup

➤ Get involved:

- <http://opensolaris.org/os/project/fmac/>

ZFS Encryption

- Set encryption policy at the ZFS data set
 - Most systems have only one or two pools but many datasets
 - AES-128 and AES-256 only initially but designed to be extensible (through minor code changes).
- Encrypted iSCSI/FCoE target via ZVOLs
- Initially can't boot from encrypted dataset
 - /var/tmp could be a separate file system
 - /tmp is backed by swap
 - Swap on an encrypted ZVOL is supported
 - No support initially for encrypted crash dumps

ZFS Encryption: Key Management

- Key scope:
 - Wrapping key is inherited, unless
 - explicitly set to something different
 - at create time, or
 - at key change time.
- Keys as passphrase, in file, or HSM/Smartcard

ZFS Encryption: Key Management (cont.)

- Data set encryption property set at create time
 - Actual encryption key is randomly generated and wrapped by user/admin provided key (pool / dataset)
 - Avoids encrypt later problem
 - Avoids old clear text due to COW
 - Policy Cannot be enabled or changed later for existing dataset
- Key change supported
- Get involved:
 - <http://opensolaris.org/os/project/zfs-crypto/>

Validated Execution

- Automatic integrity verification of all code before execution
 - Kernel
 - Libraries
 - Utilities, including shell scripts, Java, Perl, etc.
- Secure boot
- Uses Trusted Platform Module (TPM)
 - Inexpensive security module on system motherboard
 - Provides keystore and “sealed storage”

Validated Execution (cont.)

- Signed manifests describing executables
 - Shipped with Solaris releases and patches
 - Can be generated by customers and ISV's
- Administrator control over
 - Which signatures to accept and maximum privileges available to unverified executables
- Verify executables
 - `exec()`, `dlopen()`, `mmap()`, `modload`
- Boot: use TPM storage to validate initial modules
- Get involved:
 - <http://opensolaris.org/os/project/valex/>



thank you

OpenSolaris Secure Deployment:

Role-Based Access Control and
the Cryptographic Framework

Session Code: S304261

Christoph Schuba

Christoph.Schuba@Sun.COM

<http://blogs.sun.com/schuba>