# Trusted Solaris History

<table>
<thead>
<tr>
<th>Product</th>
<th>Year</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SunOS MLS 1.0</td>
<td>1990</td>
<td>TCSEC Conformance (1985 Orange Book)</td>
</tr>
<tr>
<td>SunOS CMW 1.0</td>
<td>1992</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 1.2</td>
<td>1995</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 2.5.1</td>
<td>1996</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 8</td>
<td>2000</td>
<td>Common Criteria Evaluated: CAPP, RBACPP, LSPP at EAL4+</td>
</tr>
</tbody>
</table>

*Mandatory Access Control, Labeled Desktop, Labeled Printing, Labeled Networking, Labeled Filesystems, Device Allocation, etc.*
## Trusted Solaris History

<table>
<thead>
<tr>
<th>Product</th>
<th>Year</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SunOS MLS 1.0</td>
<td>1990</td>
<td>TCSEC Conformance (1985 Orange Book)</td>
</tr>
<tr>
<td>SunOS CMW 1.0</td>
<td>1992</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 1.2</td>
<td>1995</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 2.5.1</td>
<td>1996</td>
<td>ITSEC Certified for E3 / F-B1</td>
</tr>
<tr>
<td>Trusted Solaris 8</td>
<td>2000</td>
<td>Common Criteria Evaluated: CAPP, RBACPP, LSPP at EAL4+</td>
</tr>
</tbody>
</table>

*Mandatory Access Control, Labeled Desktop, Labeled Printing, Labeled Networking, Labeled Filesystems, Device Allocation, etc.*
Solaris Trusted Extensions

• A redesign of the Trusted Solaris product using a layered architecture.

• An extension of the Solaris 10 security foundation providing access control policies based on the sensitivity/label of objects.

• A set of label-aware services which implement multilevel security.
Extending Solaris 10 Security Features

• Process Rights Management (Privileges)
  > Fine-grained privileges for X windows
  > Rights management applied to desktop actions
• User Rights Management (RBAC)
  > Labels and clearances
  > Additional desktop policies
• Solaris Containers (Zones)
  > Unique Sensitivity Labels
  > Trusted (label-based) Networking
Trusted Extensions in a Nutshell

• Every object has a label associated with it.
  > Files, windows, printers, devices, network packets, network interfaces, processes, etc.

• Accessing or sharing data is controlled by the relationships between the labels of different objects.
  > 'Secret' objects can not see 'Top Secret' objects.
  > 'Company Internal' can not send to 'Partner' networks.

• Administrators utilize Solaris Roles for duty separation.
  > Installation, System Admin., Security Admin., etc.
What are Label-Aware Services?

• Services that are trusted to protect multi-level information according to predefined policy.
• Trusted Extensions label-aware service include:
  > Labeled Desktops
  > Labeled Printing
  > Labeled Networking
  > Labeled Filesystems
  > Label Configuration and Translation
  > System Management Tools
  > Device Allocation
Labeled Desktop
Mandatory Access Control

This is some text in a StarOffice Writer Document that is classified Confidential: Restricted.
Putting It All Together

- Solaris 10 Security – A Secure Foundation for Success:
  > Reduced Networking Meta Cluster
  > Signed Binary Execution
  > Secure Service Management
  > User Rights Management
  > Process Rights Management
  > Resource Management
  > Kerberos, SSH, IPsec
  > Cryptographic Framework
  > Containers / Zones
  > IP Filter, TCP Wrappers
  > Auditing, BART
  > Trusted Extensions
But wait! There's more!

- Network Security Improvements
  - Kernel SSL Proxy
  - IPsec/IKE NAT Traversal
  - RIPv2 Protocol Support
  - Packet Filtering Hooks
  - Randomized TCP/UDP Ephemeral Port Selection
- Auditing Improvements
  - Audit Trail Noise Reduction
  - Audit Event Reclassification
- New Mount Options
  - noexec, nodevices
and more...

- “root” GID is now “0” (root) not “1” (other)
- ip_respond_to_timestamp now “0”.
- find(1) Support for ACLs
- “death by rm” safety
- OpenSSL libraries with a PKCS#11 engine
- Hardware RNG using Crypto Framework
- open(2) [O_NOFOLLOW], getpeerucred(3c), and many other developer enhancements...
- “Off the Record” plugin for pidgin (nee gaim)
- Sendmail support for TLS
and more...

• NFSv4
  > Support for GSS_API, ACLs, etc.
• Sendmail 8.13.8
  > Support for rate limiting and milters, TLS, etc.
• BIND 9.3.6-P1
  > DNSSEC, Views, IPv6 Support
• Java 5 Security (1.5.0_17-b04)
  > Security tokens, better support for more security standards (SASL, OCSP, TSP), various crypto and GSS security enhancements, etc.
  • ... and the list keep right on going...
Labeled Networking - Problem

• Today, *trusted networking* involves explicitly-labeled packets.

• Today, most explicitly-labeled networks must be physically secure.
  > Labels can be easily forged/injected otherwise.

• IPsec can protect packets against forgery/injection
  > But explicit labels are IP options outside ESP's protection.
Labeled Networking

Intranet

Need-to-know

Internal Use

Public

Global Zone

Solaris Kernel

Multilevel Network

SunRay Network
CIPSO problems

• Cleartext label visible on the wire in each packet
• Not protected end-to-end against modification
• Uses extra space in every packet - complicates MTU discovery
Labeled IPsec - Solution

• Why not associate a packet's label with its IPsec cryptographic key?!?
• IKE can be either unlabeled or a single explicit label, depending on existing Trusted Networking databases.
• IPsec-protected traffic can have implicit labeling, or also implicit PLUS explicit labeling (which can be different or the same as the implicit label).
• With Labeled IPsec, the network need not be physically secure.
Labeled IPsec

• Sensitivity label is an attribute of an IPsec Security Association (SA).
• Each SA is single-label, set by Key Management at creation.
• IPsec flow policy enforces label match.
• Label made available to applications is securely bound to traffic.
Labeled IPsec: SADB Extensions

- PF_KEY (RFC 2367) planned for this 10 years ago
- Labels are SA properties
- Inner sensitivity label matches cleartext traffic
- Outer sensitivity label appears on wire in clear
- Outer sensitivity label may be omitted from packet
  > No extra space on the wire vs unlabeled IPsec!
- Outer label under control of key management daemon
Labeled IPsec: more information

- Limited prototype is working; not yet available
- Watch:
  > http://www.opensolaris.org/os/project/txipsec
- Questions/comments:
  > security-discuss@opensolaris.org
Encrypted Storage with ZFS

“To create a reliable storage system from inherently unreliable components”

• Data Integrity
  > Historically considered “too expensive”
  > Turns out, no it isn't
  > Real world evidence shows silent corruption a reality
  > Alternative is unacceptable
ZFS Elevator Pitch

• Ease of Use
  > Combined filesystem and volume management
  > Underlying storage managed as Pools which simply admin
  > Two commands: zpool & zfs
    > zpool: manage storage pool
      – aka volume management)
    > zfs: manage filesystems
Back to the Elevator Pitch

• Let's add just one word:

“To create a secured reliable storage system from inherently unreliable components”
ZFS Terminology

• Pool
  > Collection of disks in RAID

• Dataset
  > Filesystem or Emulated volume (ZVOL)

• Copy on Write
  > Everything in ZFS is COW & checksummed, written in transactions. Always consistent on disk.
  > (POSIX) Sync write via Intent Log (ZIL)
High Level Requirements

• Support software only solution
  > Including single disk laptop use case
• SPARC, Intel, AMD64
  > Anything that OpenSolaris runs on and that ZFS has already been ported to
• Support keys & cryptographic operations in hardware: eg UltraSPARC T2
• Local key management:
  > HSM, TPM, smart card, passphrase
• Remote/Centralised key management
High Level Requirements

- Don't break Copy-On-Write semantics
- Integrate with existing ZFS admin model
  > CLI & GUI
- Support existing ZFS pools
- Delegation of key management to users, virtualized & Multi Level (MLS) environments
  > ability to create encrypted datasets
  > Including separation of key use vs key change
ZFS Encryption

• Set encryption policy at the ZFS data set
  > Most systems have only one or two pools but many (10s, 100s, 1000s,) datasets
  > AES-128 and AES-256 only initially but designed to be extensible

• Encrypted iSCSI & FCoE targets via ZVOLs
  > No key management on Initiator

• Encrypted datasets CAN be shared using NAS: NFSv2,v3,v4 & CIFS (SMB)
  > No key management for NAS clients
Doing the Encryption

• Data set encryption property set at create time
• Actual encryption key is randomly generated
  > wrapped by user/admin provided key
• Avoids encrypt later problem
  > Avoids old clear text due to COW
• Encryption algorithm and ZFS checksum cannot be enabled or changed later for existing dataset
  > ZFS checksum forced to SHA256 rather than default for data Fletcher2
• AES CCM MAC stored with checksum in block pointer.
Key Management

• Wrapping keys provided by user/admin
  > passphrase,
  > Raw (or hex) key
  > TPM/HSM/Smartcard (PKCS#11 accessible)

• Wrapping key inherited by child datasets

• Clones can have new encryption key
  > Opt in default is to share origin dataset key.
Key Change

• Key change supported
  > Doesn't actually re-encrypt data
    > May support this type of mode in future.
  > Changes wrapping key.

• Key Change is online
  > Datasets must be mounted – or at least key available
  > Datasets stay mounted/shared during key change
“External” Key Management

- “Base ZFS”
  - Key in file, passphrase, PKCS#11 token
- API in libzfs to provide key by value or PKCS#11 object name.
- zfs(1M) scriptable interface
- TPM support via PKCS#11
  - Future encrypted boot may access directly using TCS API
What is encrypted?

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>• All “application” data</td>
<td>• Pool metadata</td>
</tr>
<tr>
<td>• POSIX layer data</td>
<td>&gt; Disks, raid config, etc.</td>
</tr>
<tr>
<td>&gt; Permissions, owner etc</td>
<td>&gt; Dataset properties</td>
</tr>
<tr>
<td>• Directory structure</td>
<td></td>
</tr>
<tr>
<td>• All ZVOL data</td>
<td></td>
</tr>
<tr>
<td>• All the above in a snapshot</td>
<td></td>
</tr>
<tr>
<td>• All the above in a clone</td>
<td></td>
</tr>
</tbody>
</table>

**Deployment Issues**

• Dataset names
• Dataset user properties properties
SSD Storage & Crypto

• ZFS can use SSD for two distinct purposes
  > ZIL – ZFS Intent Log
    > Fast write device required
  > L2ARC – Cache between memory and disk
    > Fast read device required

• SSD is persistent so data MUST be encrypted
  > ZIL is always encrypted anyway SSD case is no different
  > L2ARC encrypt on “evict” to cache device, in memory checksum. Ephemeral key.
“The Cryptography Bit”

• Data encrypted with AES in CCM mode
  > Integrity of ciphertext
    > ZFS Checksum (SHA256) not “keyed” -> for data verification and reconstructions.
    > CCM MAC for ciphertext blocks
  > ZFS has multiple block sizes: 512 bytes -> 128k

• Key wrapping also uses AES in CCM mode
  > Wrapped key integrity
  > “Correct Key” checking for free (helps key change)
CCM Params

- Data can be 512 bytes to 128k
  - nonceSize = 12 (13 is CCM max: too small)
- AuthDataSize = 0
- Nonce built from blkptr / zio bookmark
  - Txg – 64bit non repeating transaction id for pool
    - Can't easily determine which txg a write for a given dataset will happen in, A single txg can contain writes for many datasets.
  - Blkid
  - Object
- MACsize = 16
  - MAC stored in high two uint64_t of zio checksum
  - Low two uint64_t store truncated SHA256 (bigendian)
CCM For Key wrapping

• Also use CCM for wrapping the per dataset encryption key.
• CCM Params for wrapping:
  > Noncesize 13
  > Randomly generated Nonce
    > Ensure it isn't already used on key change operations.
  > MACsize 16
• Nonce & MAC stored with wrapped key in ZFS dataset property
Future Key wrapping

• NIST AES Keywrap
  > Once it is included in PKCS#11
• May consider wrapping using RSA
• Key wrapping algorithm will be able to be changed by doing a key change operation 'zfs key -c -o keywrap=....'
• Won't require pool/dataset to be offline.
Crypto bit for Caching

- L2ARC written to “differently” from normal IO (zio_phys_write)
  > Currently a “non persistent cache” - but written to persistent media
  > Encryption using pool wide ephemeral key
  > AES_CBC not AES_CCM
    > Safe enough since we have in memory checksum
    > No space for MAC on disk
- L2ARC will switch to AES_CCM when it becomes “persistent” cache.
What about all the decrypted data?

- ZFS in memory cache (ARC) contains very large amounts of decrypted data
  > Requires full privilege to see (/dev/kmem)
    > But still a risk.
- Can control use data in the cache per dataset
  > Primarycache (memory): none, metadata, all
  > Secondarycache (SSD): none, metadata, all
- Future may have encrypted data in primarycache (ARC) as well as secondarycache (L2ARC).
Unwrapped keys in RAM?

• In the pure software case yes the unwrapped keys are in host RAM
• May not be the case with some hardware crypto keystore/accelerator.
• Keys (and expanded schedule) only in kernel memory
  > kmem_alloc(9F) on OpenSolaris, not paged when system swaps
  > OpenSolaris x86 suspend to RAM only (S3)
  > Solaris SPARC can suspend to disk
Current Deployment Restrictions

• Initially can't boot from encrypted dataset
  > /var/tmp could be a separate file system
  > /tmp is backed by swap

• No support initially for encrypted crash dump devices
  > But Swap on an encrypted ZVOL is supported
  > Encrypted crash dumps could be supported but maybe better to have support independent of ZFS.
What about when ZFS evolves

- Most storage in ZFS is via DMU layer objects
- Encryption selected per DMU object
  - Some objects need to be in the clear to allow pool traversal for resilver/scrub and initial import
  - 11 out of 40 in current codebase encrypted
  - Encryption support is a new pool version.
- Future encryption features may version on disk format – will be upwards compatible.
- New dataset types (eg for pNFS) could have different rules to filesystem & ZVOL.
ZFS Encryption Support Availability

- OpenSolaris project
  - All project code is opensource (CDDL)
  - Depends on OpenSolaris Crypto Framework
    - Porting to other OS Platforms should be relatively easy.
  - [http://opensolaris.org/os/project/zfs-crypto/](http://opensolaris.org/os/project/zfs-crypto/)
- Should port to other platforms that already have ZFS relatively easy.
  - Nothing really OpenSolaris specific
Stack Shadowing for SPARC

• Stack buffer overflows widely recognized as the leading cause of security vulnerabilities
  > Responsible for 45% of CERT advisories
  > Huge customer impact
• We can completely stop this class of attack
• Very small one-time development cost
Stack Buffer Overflow

- On a typical processor, the stack contains
  > Local variables
  > Function arguments
  > Return address
- Local variables may include an array used as a data buffer
- Writing past the end of the buffer overwrites the return address
Stack Buffer Overflow - SPARC

- SPARC passes return address and arguments in registers
- Register window contents spill to stack
- Overwriting the stack modifies registers, including return address
- SPARC still vulnerable
Partial Solutions

• Non-executable stacks
  > Default for 64-bit SPARC applications
  > Only stops one attack variant (code included in payload)

• Stack frame canaries
  > Verified upon function return
  > Adds run-time overhead
  > Can be fooled by more sophisticated attack

• Source code analysis
  > No tool that catches all vulnerabilities
  > Too much code – including code we don't control
Stack Shadowing

• Kernel maintains shadow copy of register save area
• Part of register window fault handler
  > Copy after register spill
  > Compare before register fill
• Unique advantage of SPARC architecture
  > Hardware provides traps exactly when we need them
Current Status

• Idea developed by NSA researcher
  > Presented to Sun as part of cooperative R&D agreement
• SPARC prototype developed in 2005 by Solaris Security
  > Proof of concept, but too simplistic for product
• Product-quality implementation now in progress