IBM Research

Trusted Computing for Linux

David Safford, Mimi Zohar, Alan Boulanger

August 2005

Project Team

- **Research (1PY ACT-I, 1PY C4EB)**
  - Dave Safford, Mimi Zohar, Alan Boulanger
- **Tivoli, other SWG**
  - Jim Whitmore
- **C4EB Team**
  - John Walicki
  - Dave Koeller
  - Eric Barkie

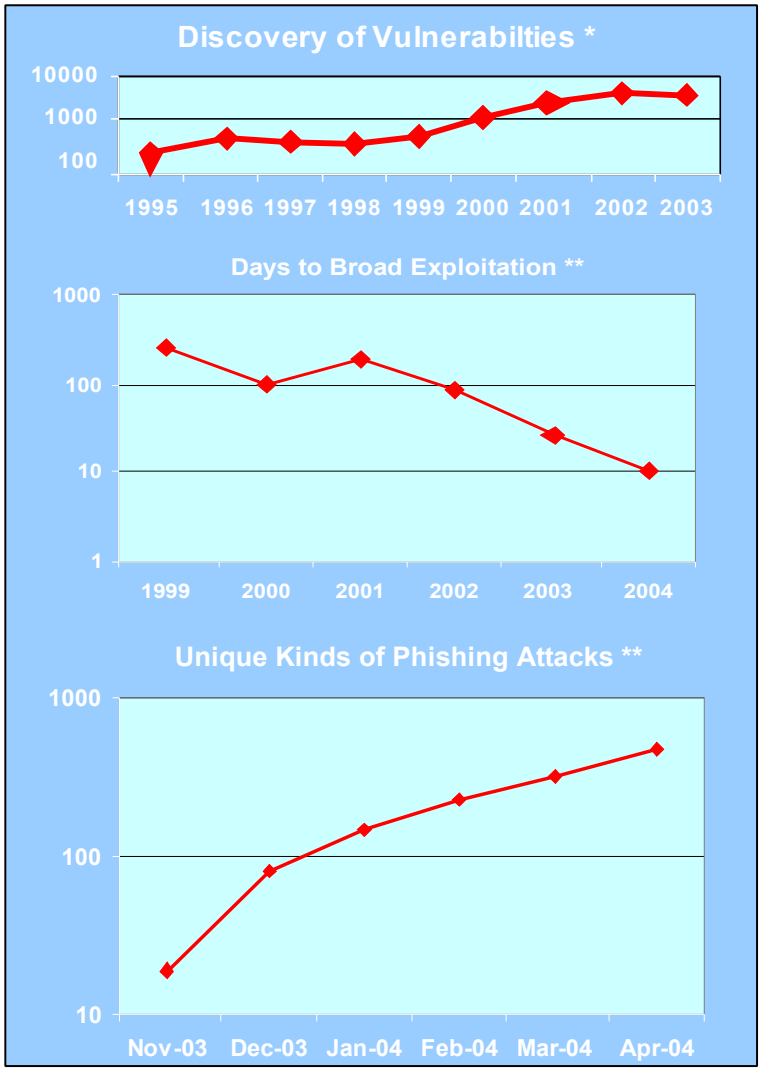
Outline

- **Threat Trends**
- **Trusted Computing**
- **Trusted Computing for Linux**

  Trusted boot
  File Integrity Verification
  File Integrity Protection
  File Integrity Attestation
  Trust Domains

The Problem: Client Risk is Dramatically Rising

- **The number of attacks in the wild, and their lifetimes and impact are growing fast**
  - 450% increase in Windows viruses over last year
  - 1500% growth in BotNets Jan to Jun 2004
  - The myDoom.O virus overloaded networks around the world in August 2004
  - Blaster worm attack cause First Energy's Davis Besse Nuclear Reactor to loose digital control for over four hours in January 2003
  - Viruses are already deploying attacks against AV software
  - 80% of clients have spyware infestations
  - 30% of clients already have back doors (FSTC)
- **Increase in vulnerability rate is slowing, but the time between the publication of a security vulnerability and the broad exploitation of it is markedly decreasing**
- **Financial losses rapidly increasing:**
  - Phishing attacks: $924M direct losses in 2004
  - Identity theft is the fastest growing crime in US

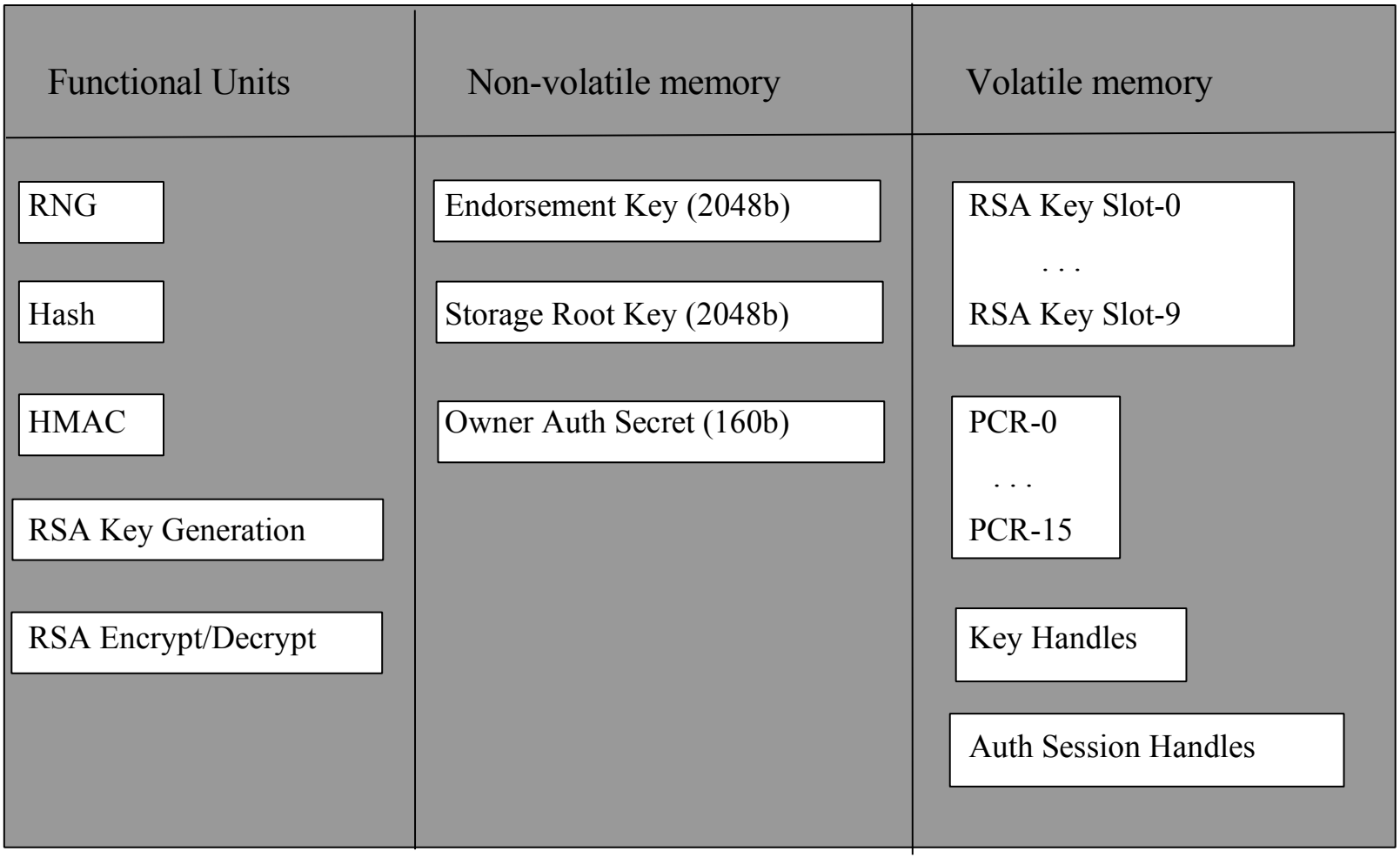


* cert.org Nov 2004
**July2004 Information Security

How can the Trusted Platform Module (TPM) Help?

- **RSA crypto**
  key generation, signature, encrypt, decrypt
- **Secure storage**
  private keys
  master keys (eg loopback)
- **Integrity measurement**
  Platform Configuration Registers (PCR)
  compromise detection
  Tie key use to uncompromised environment
- **Attestation**
  host based integrity/membership reporting
  (RSA 2004 Demo)

Programming view of the TPM

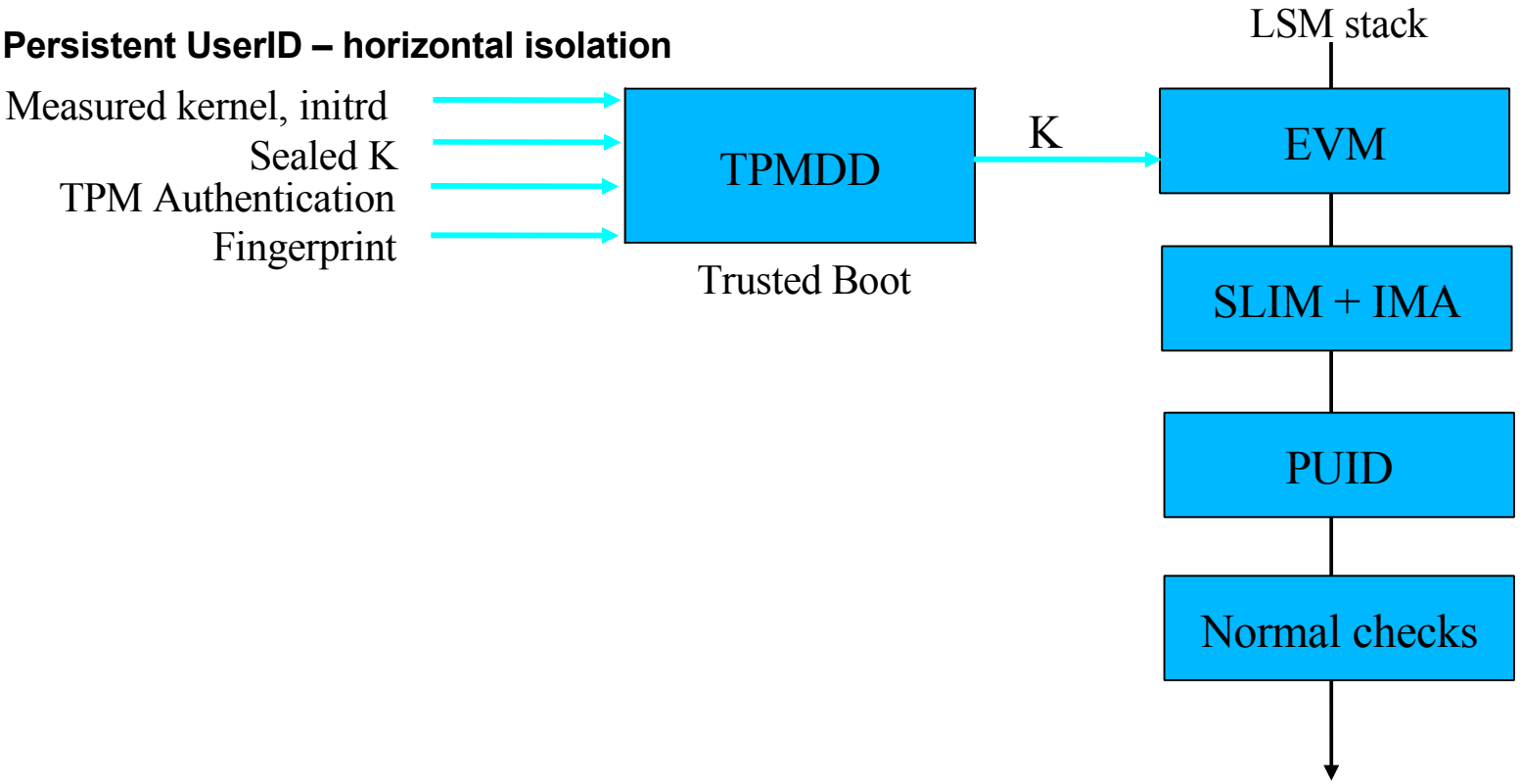


Trusted Computing for Linux: Components

- **Trusted Boot**
  - GRUB measurement
  - TPMDD – TPM device driver
  - Initrd master kernel key unsealing and verification
- **Integrity Measurement**
  - EVM - Extended Verification Module
- **Integrity Protection**
  - SLIM – Simple Linux Integrity Module
- **Integrity Attestation**
  - IMA - Integrity Measurement Architecture
- **Security Domains**
  - PUID – Persistant User ID
  - Unionfs – stacked, copy-on-write, sandbox (Stony Brook)
  - Per-process filesystem namespaces

Trusted Computing Modules:

- **Fingerprint Sensor – hardware verifies fingerprint before boot**
- **TPM: measures integrity of kernel and initrd, and unseals kernel key**
- **EVM: Extended Verification Module – authenticates data and metadata**
- **SLIM: Simple Linux Integrity Module – Mandatory Access Control Sandbox**
- **IMA: Integrity Measurement Architecture – attestation with TPM**
- **PUID: Persistent UserID – horizontal isolation**



Trusted Boot

- BIOS and GRUB measure boot through kernel and initrd
- Initrd asks TPM to unseal kernel master key
- If measurements match, TPM releases master key.
- Master key K is used to generate derived keys for
  encrypted home directory
  authenticated file attribute checking (EVM)
- If measurements don't match, boot is halted

Extended Verification Module: EVM

- **Use extended file attributes to store authenticated file metadata**

  file data hash
  mandatory access control labels
  Metadata integrity HMAC
- **Use tpm based symmetric kernel key to HMAC these attributes**
- **Verify file once at open/execute, and cache verification**
- **“heavy lifting” done at install time, runtime is just file hash and HMAC**
- **Extensible, policy based definition of attributes and actions**

SLIM: Mandatory Access Control Alternatives

- **Bell and LaPadula:**

  Secrecy based Mandatory Access Control

- **Biba:**

  Integrity based Mandatory Access Control
  Low/High-watermark option

- **Lomac: Linux module version of Biba Low-watermark**

  Only 2 integrity levels trusted & untrusted make it easy to administer
  All process start out trusted, only when a process attempts an untrusted operation is it demoted (i.e. reading from the network)
  Problem: guard processes to promote untrusted objects to trusted.

- **Caernarvon: modified Biba integrity model**

  Incoporated support for guard processes, verified trusted programs, which are allowed to remain high integrity, while reading low integrity objects
  Separation of reading and writing/execution permissions
  Does not support low/high-watermark

SLIM: Mandatory Access Control Alternatives Cont.

- **Security Enhanced Linux (SELinux): LSM framework based**

  Permits the creation of a wide range of security models
  Default security model: Type Enforcement (TE)
  More powerful than Biba or Caernarvon
  Problems:
  - Difficult to configure: default configuration file ~33,000 line ruleset
  - Default configuration does not ensure integrity containment
  - Cannot create low/high watermark policies

SLIM Sandbox:

- **Simple Linux Integrity Module (SLIM)**

  Use of LSM framework hooks
  EVM context information to enable sandbox decision
  Includes Lomac's low-water mark integrity model for ease of administration
  With Caernarvon's separation of read and write/execute permissions
  With Caernarvon's signed guard processes – verified trusted programs

- **Basic Integrity Operation:**

  Low Integrity processes can read and execute up, but not write up
  High Integrity processes can write down, but are demoted on read/execute down
  Trusted “guard” processes, verified by EVM, can read down without demotion
    rpm
    sshd

SLIM Access Classes

All Files are labeled with an Integrity and Secrecy MAC label

```
  Integrity Access Classes (IAC)
     SYSTEM
     USER
     UNTRUSTED
     EXEMPT

  Secrecy Access Classes (SAC)
     SENSITIVE
     USER
     PUBLIC
     EXEMPT
```

All Processes have upper and lower Integrity and Secrecy labels:

```
  Integrity Write/Execute Access Class (IWXAC)
  Integrity Read Access Class (IRAC)
  Secrecy Write Access Class (SWAC)
  Secrecy Read/Execute Access Class (SRXAC)
```

(Upper and Lower are the same, except for guard processes.)

EVM and SLIM Extended Attributes

EVM Extended Attributes:

```
security.evm.hash  - hash of file data (from signed rpm)
security.evm.hmac  - hmac-sha1 of security.* attributes
security.evm.packager – signer of package
security.evm.version  - version of package
```

SLIM Extended Attributes

```
security.slim.level – six class values (values are space delimited)

   IAC    - File's Integrity Access Class
   SAC    - File's Secrecy Access Class

   IRAC   - guard process Integrity Read Access Class
   IWXAC  - guard process Integrity Write/Execute Class
   SWAC   - guard process Secrecy Write Access Class
   SRXAC  - guard process Secrecy Read/Execute Class
```

EVM and SLIM Extended Attributes

Attributes of "/bin/ping":

```
[root@localhost safford]# getfattr -d -m "^security" /bin/ping
security.evm.hash="\\1\265Ad6a4d94fb694cffd2847acf40dbc6485"
security.evm.hmac=0s8ZT9309FkQBag7HkqqieSeuya3s=
security.evm.packager="\\1\265ARed Hat,  Inc.
      <http://bugzilla.redhat.com/bugzilla>"
security.evm.version="\\1\265A20020927(rel16)"
security.selinux="system_u:object_r:bin_t\000"
security.slim.level="SYSTEM"
```

Running "/bin/ping" first time (from dmesg logging):

```
evm_calc_hmac: ping - security.evm.hash included
evm_verify_xattr: verification of security.evm.hmac succeeded
evm_inode_permission: 'ping' HMAC verification succeeeded
evm_inode_permission: 'ping' HMAC verify xattr
evm_inode_permission: security.evm.hash is d6a4d94fb694cffd2847acf40dbc6485
evm_inode_permission: security.evm.hash succeeded
```

Running "/bin/ping" subsequent times:

```
evm_analyze_cacheinfo success
```

Remote Attestation

SSL and IPSec provide secure communication

But what about the integrity of the remote system itself?

How can you trust a remote system hasn't been compromise

IMA: Integrity Measurement Architecture

- Extend TPM-based attestation into the system runtime

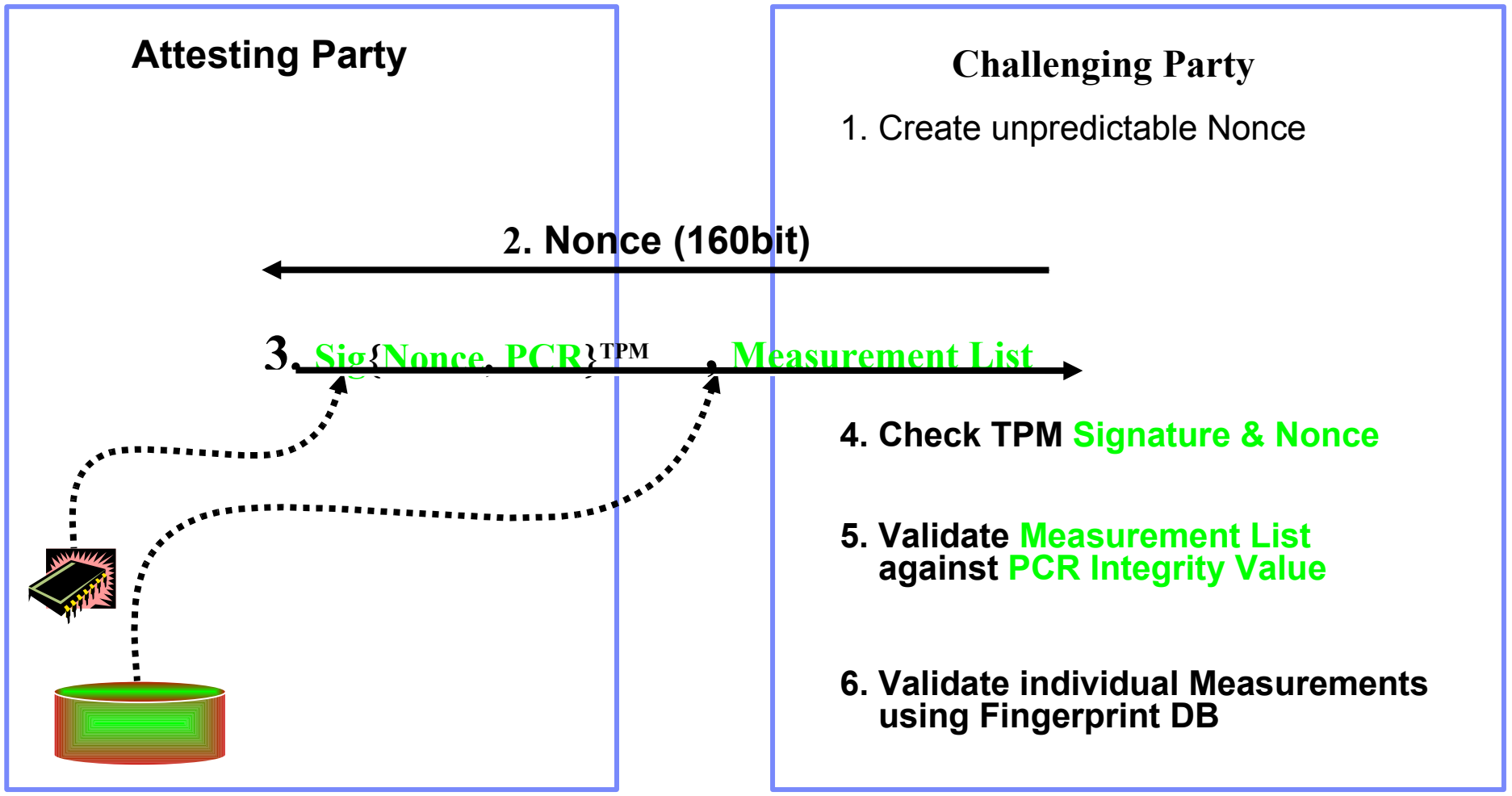
  EVM provides the measurements
  SLIM identifies
  - all executables (programs, scripts)
  - all SYSTEM level integrity objects (config files)

- Attest the Software Stack

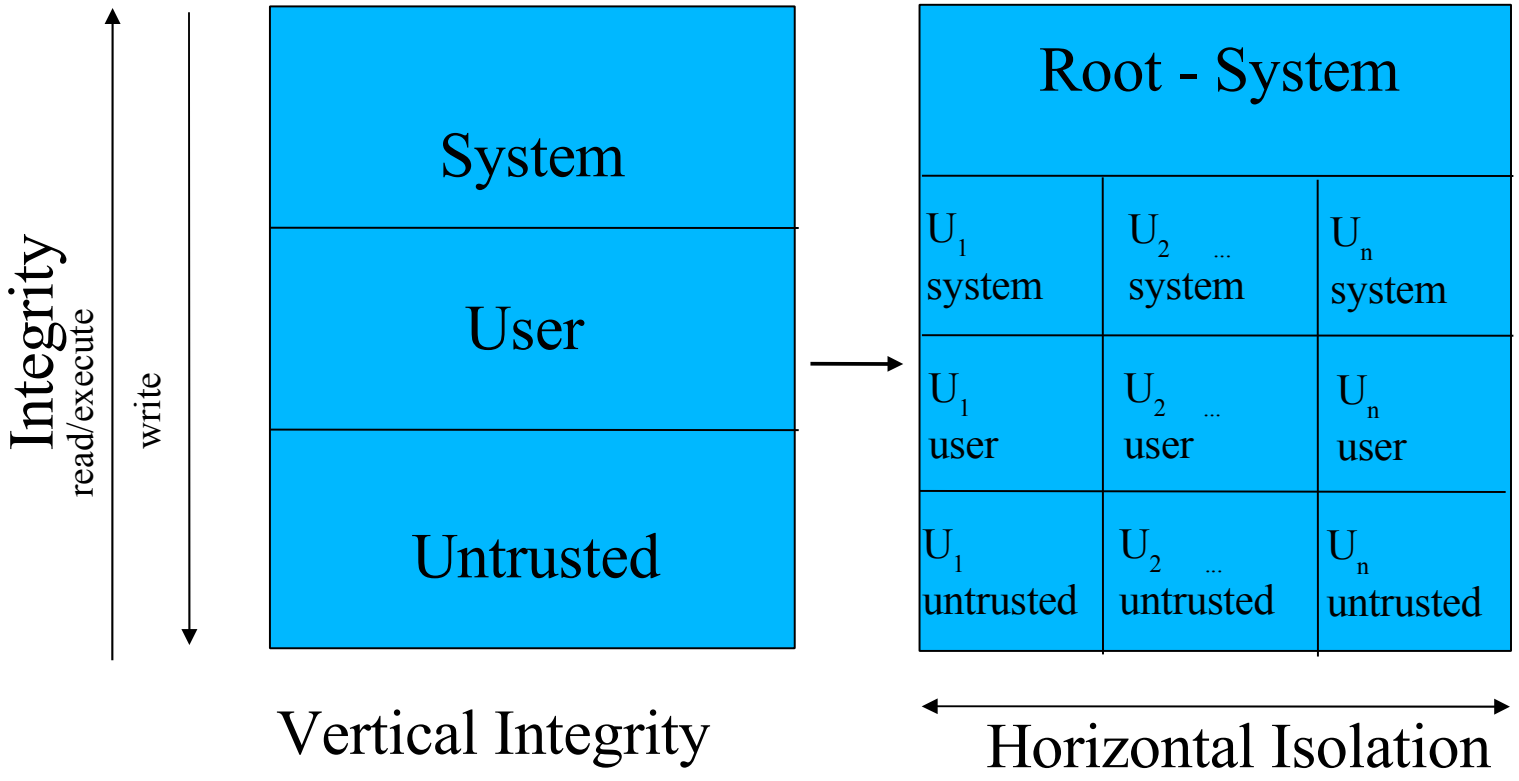
  Adjust Protected hw Platform Configuration Register (PCR) to maintain measurement list Integrity Value
  Adds measurement to ordered measurement list

```
# cat /sys/kernel/security/ima/ascii_runtime_measurements
10 1ac2616d0a992514d02c69ba1e2cfde6e9096fce boot_aggregate
10 668a6a23db3b47ffd229f3642ab8b86d00000000 /sbin/init
10 e78baf9c4cf6accebc7ce36891a9238a00000000 /lib/ld-2.3.4.so
10 3b21b2c293b97b231edc89a34f02593300000000
   /etc/ld.so.cache
```

Attestation Protocol



Mandatory Access Control Models



Trust Domains

- **Per-process namespace (linux 2.6)**
- **Unionfs: stack private copy-on-write ontop of /**

  /domains/avis/root = rw
  /=ro
- **Persistant or Ephemeral domains**
- **Extremely fast creation/destruction (15ms)**
- **Unlike chroot based systems (Solaris, vserver), no copying of base filesystem is needed.**
- **Provides MAC with read/execute up, but no write up.**

Persistent User ID – PUID

- Loadable Linux Security Module (LSM)
- Horizontal isolation via MAC
- Protects files, tasks, and devices
- Isolation of “ps” and “ls” between users
- Supports limited (“tainted”) root use

Comparison to Solaris Containers, Linux Vserver

- **Containers/Vserver:**
  - Use chroot for file visibility control
    - Do not control file access, just visibility
    - Require duplication of ~1GB of system files (~8 minutes)
- **TC4L complements these models with:**
  - True mandatory access control for
    - Strong horizontal isolation
    - Vertical Integrity protection
  - Stacked file system for copy-on-write
    - much simpler/faster instantiation/management
    - Integrity measurement, attestation