Administrivia

- **Encryption utility now due Friday (Feb 25)**
- **Midterm Tuesday March 1**
  - Open book, open note, bring papers
- **New section times:**
  - Wednesday 7:10pm-8:10pm, 719 Bway #709
  - Thursday 6:00pm-7:00pm, 719 Bway #709
- **My office hours will be *before* midterm next week**
  - Monday 4:30-6pm
- **By request, Antonio's office hours now after class**

Terra

- **Goal: closed-systems type security on general-purpose computers**
  - *Root secure* – evan platform owner
  - *Attestation* – prove to remote hosts what software you are running
  - *Trusted path* – prove to user what software s/he is interacting with
- **Idea: Give up on fixing Windows/Linux**
  - Just run closed system in parallel with other OSes
  - Note: Rosenblum one of the founders of VMWare

How do devices work

- **Management VM creates other VMs, devices**
- **Virtualize the host OS's devices**
  - E.g., emulate internal network, emulate NAT
  - Make a file/partition look like a disk to a VM
  - TVMM has raw, MACed, and encrypted+MACed disks
- **Note that VMs themselves can look like devices**
  - E.g., Closed crypto VM could look like USB device

Hardware support required

- **Hardware attestation (prove which TVMM you booted)**
- **Sealed storage**
  - Can only access if current TVMM hash same as when stored
  - Use to store TVMM private key
- **Secure counter – could detect replay attacks**
- **Wish list:**
  - Virtualizable CPU, Secure I/O, Device isolation, real-time support

Attestation

- **This is the same idea as Taos, different implementation**
- **What needs to be attested:**
  - BIOS
  - bootloader, TVMM
  - VM OS or boot image
- **Look at quake example**

Dangers

- **Interoperability & Consumer Protection (p. 5)**
  - What's the issue here?
  - Is it solvable?
- **What are privacy issues? Group signatures?**
- **Why should TVMM be more secure that OS?**
  - Probably not hard when competition is windows
  - What about drivers (esp. video drivers)?
- **Is attested software more secure?**
  - No, it's just attested

[XOM discussion]

hypothetical DRM-enabled music player

```
int drm_ok
  = debit_account_for_cost_of_one_listen (user, song);
if (!drm_ok) {
  drm_warn ("Sorry, you must deposit more money\n");
  exit (1);
}
play_music (song);
```

DRM-enabled music player 2

```
/* read buffer from disk before decrypting */
int
get_buffer (char *buf, size_t size)
{
  char *plaintext_buf;
  int n;
  plaintext_buf = malloc (size);
  n = read (drm_file_fd, plaintext_buf, size);
  if (n > 0)
    copy_buffer_from_null (buf, plaintext_buf, n);
  free (plaintext_buf);
  return n;
}
```

SIGCHLD handler from mpg123

```
static
void catch_child (void)
{
  while (waitpid(-1, NULL, WNOHANG) > 0);
}
```

hypothetical RSA decrypt inner-loop

```
/* compute m^d (mod n) */
bigint
decrypt_inner (bigint m, bigint d, bigint n)
{
  bigint result = 1;
  while (d != 0) {
    if (d & 1)
      result *= m;
    m = m * m;
    d >>= 1;
  }
  return result;
}
```

[stretch break]

AEGIS: An alternative to XOM

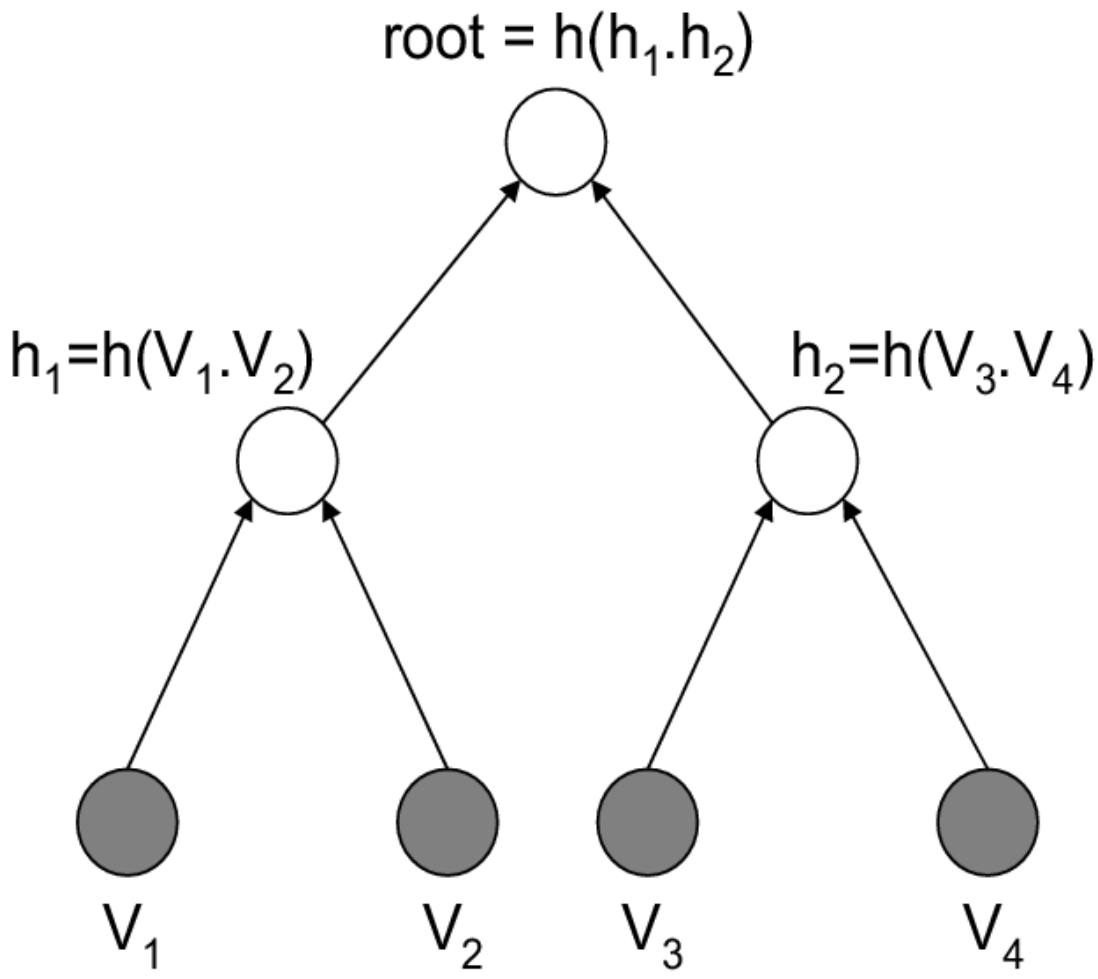
- ***Tamper-evident* environments (TE)**
  - Prove that something is the result of a computation
  - Possibly useful for Grid-type applications
- ***Private tamper-resistant* environments (PTR)**
  - E.g., for DRM, more XOM-like applications
- **Major advantages over XOM**
  - Provides real attestation (stronger than XOM's self-attestation)
  - Protects memory against replay & tampering attacks
  - Uses dynamic keys to encrypt mutable data

Tamper-evident processing

- **New operations: enter_aegis, exit_aegis, sign_msg**
- **enter_aegis starts tamper-evident processing**
  - Operation computes hash of the current program $H(\mathrm{Prog})$
  - Note Prog doesn't have to be the program, can be a "stub"
  - Stub contains hashes of other parts, verifies them on-demand
- $\mathrm{sign\_msg}(M) \rightarrow \{H(\mathrm{Prog}), M\}_{K_P^{-1}}$
  - If you trust hardware and know it's key K_P, then proves that M is output of Prog
- **Note memory protection based on virtual address**
  - E.g., DMA into low memory, copy to protected memory
  - Easier to reason about than XOM

Ensuring data integrity

- **Recursive use of cryptographic hash… Merkle tree**



Memory verification details

- **Can trust the contents of the L2 cache (on chip)**
- **For initialization, don't want to hash giant tree**
  - Include bit as to whether memory has been seen yet
  - "Prunes" sections of the full tree
  - Hash is recorded when you first read or write protected cache line
- **Optimizations:**
  - Use speculative execution as optimization
  - Use second TLB for hash tree, to avoid slowing main TLB

Private Tamper-Resistant Processing

- **Subdivide protected mem for private region**
- **set_aegis_mode sets PTR mode from within TE**
  - Instruction takes $\{H(\mathrm{Prog}), K_{\mathrm{static}}\}_{K_P}$ as argument
  - K_{static} used to decrypt program
  - ...but only if $H(\mathrm{Prog})$ matches program
  - Note encryption must be non-malleable (implied by adaptive chosen ciphertext security)

Memory encryption details

- **Mutable data encrypted with dynamic key**
  - Provides forward secrecy
  - Avoids possibly divulging repeated plaintext blocks
- **Each cache encrypted with AES in CBC-mode**
- **IV is constructed as follows**
  - Chose random 32-bit RV (random vector)
  - IV = virtual-address | RV | 0^{64}
- **Hash tree is over plaintext, not ciphertext**
  - Should probably have been other way around
  - Can you find an attack?

[Quiz Review]

Questions to ask about protocols

- **Is everything explicitly stated in messages?**
- **Does one party act as a decryption/signing oracle?**
  - E.g., kerberos kinit decrypts data and sends it onto network
- **Can one party to impersonate the other?**
- **Does the protocol provide freshness?**
  - Might some message be recyclable in a replay attack?
- **Does the protocol provide forward secrecy?**
  - Long-lived keys should be for authentication, not secrecy

Questions to ask about passwords

- **Who can mount off-line guessing attacks?**
  - Should the system be using something like SRP?
- **Is the password "salted"?**
  - Is there a cost parameter? (Usually not, but it's a good idea.)
- **What are consequences of server compromise?**
  - For users who use same password on several systems?
  - For users with good (hard to guess) passwords?
- **Is any timing information leaked?**
  - e.g., `strcmp (passwd, system_passwd)`

Web security

- **SSL authenticates server to client**
  - Based on globally-trusted CA
  - SSL usually not used for user-authentication
- **User-authentication usually w. passwords/cookies**
  - User supplies password, gets cookie from server
  - Future requests from client contain the cookie
- **Beware home-brew cookie authentication schemes**
  - "Interrogative adversary" surprisingly powerful attacker
  - Must not be possible for attacker to make up valid cookies
  - Don't rely on clients to invalidate cookies

Kerberos

- **Basic idea: Emulate CA without public key crypto**
  - Users and servers each share a secret with trusted KDC
  - KDC leverages shared secrets to establish session keys
- **Basic protocol:**
  - Ticket: $T = \{s, c, \text{addr}, \text{expire}, K_{s,c}\}_{K_s}$
    (K_S is key s shares with KDC)
  - Authenticator: $A = T, \{c, \text{addr}, \text{time}\}_{K_{s,c}}$
- **Login protocol:**
  - $c \rightarrow t$: c, t (t is name of TG service)
  - $t \rightarrow c$: $\{K_{c,t}, T_{c,t} = \{s, t, \text{addr}, \text{expire}, K_{c,t}\}_{K_t}\}_{K_c}$
  - $c \rightarrow t$: $s, T_{c,t}, \{c, \text{addr}, \text{time}\}_{K_{c,t}}$
  - $t \rightarrow c$: $\{T_{s,c}, K_{s,c}\}_{K_{c,t}}$

SDSI

- **The SDSI/SPKI public key infrastructure**
  - No global names, as in other infrastructures
  - All terms must begin with a Key: K_{dm} verisign
- **Name certificates bind terms to names in local namespace**
  - Can reference other people's namespaces, e.g.: K_{dm} verisign shop.com
- **Authorization certificates specify actions allowed to which principals**
- **Clarke result: Terms efficiently evaluatable**

Mandatory access control (MAC)

- **Goal: Prevent information flow**
  - E.g., Just because A can read a classified file, does not mean he should be able to send the contents to B
- **Classify data by sensitivity (secret, top secret, ...)**
  - Security policy prevents "read up" and "write down"
- **Classify data by compartments**
  - E.g., cryptography, nuclear, special intelligence, ...
- **Integrity uses same mechanism upside down**

Jif

- **Controls information flow at programming language level**
  - Assumes trusted execution environment
  - Lets users run untrusted code on sensitive data
- **Makes declassification part of the model**
  - Labels remember which users impose which restrictions
  - E.g., $\{o_1 : p_1;\ o_2 : p_2\}$
- **Nice language features**
  - Automatic label inference, label polymorphism, etc.

Capabilities

- **For each process, store list of allowed objects**
  - In contrast to access control lists, stored at object
- **Examples of capability-like things:**
  - File descriptors (granting access to a file)
  - Java pointers (grant access to object)
- **Programs explicitly invoke particular capabilities**
  - Tells system which rights they want to invoke
  - Avoids the confused deputy problem

Taos

- **Logic to reason about when a principal *says* X**
- **Some principals say things directly**
  - Channels (symmetric keys) and public signature keys
- **Otherwise, reason with $\Rightarrow$ (*speaks for*) relation**
  - If ($A \Rightarrow B$) and (A says S) then (B says S)
- **Restrict principals with roles:** $A \Rightarrow (A$ **as** $R)$
- **"B quoting A" written** $(B|A)$
  - Can allow people to quote: A says $((B|A) \Rightarrow (B$ for $A))$
  - B for A means you've inferred such a delegation