CNS Lecture 13

Network defenses
IPsec
Virtual Private Networks (VPNs)
Wireless security
Kerberos
Trusted systems
Secure OS

You are here ...

 Attacks & Defenses
- Risk assessment
- Viruses
- Link security
- Authentication
- Network security
  - Firewalls,vpn,IPsec,IDS
- Forensics

Cryptography
- Random numbers
- Hash functions
- MD5, SHA256
- Classical + steps
- Number theory
- Symmetric key
- DES, Rijndael, AES

Applied crypto
- SSH
- PGP
- X.509
- SSL
- Kerberos
- IPsec
- Crypto APIs
- Public key
- Secure coding

RSA, DSA, D-H-ECC

In the news

Microsoft workstation buffer overflow
Microsoft XML core services remote code execution
Microsoft agent buffer overflow
WinZip remote code execution

Network security

VULNERABILITIES
- Denial of service
- SYN, TCP, UDP, ICMP
- IMS
- Traversal attacks
- Spoofing
- Session capture/modified
- TCP/UDP payloads
- Denial of service

COUNTERMEASURES
- Double authentication
- Content encryption
- Key management
- Key exchange
- Protocol
- Secure key exchange
- Authentication
- Secure remote access
- Secure file transfer
- Secure network services

Where to encrypt?

- Link layer
  - Encrypted modem, NIC (wireless)
  - Transparent, best
  - Non-transparent (policy)
  - Custom host-based keying
  - Over public net

- System layer
  - Encrypted file systems (EFS, CIFS)
  - Application layer
  - End-to-end public net
  - Custom applications (PGP, ssh, ssl)
  - API for application development

Internet protocol (IP)

Is IP secure?

- Integrity -- checksum?
- Can you trust the source address?
- Privacy?
- IP security option (RFC 760)
  - Military security model, subject/object labels
  - Label each IP datagram (secret, top secret, unclassified)
  - IP stack and routers enforce access controls
  - Only effective in very controlled environment
**IP header**

```
+-------------+-------------+-------------+-------------+-------------+
|     0       |     1       |     2       |     3       |
+-------------+-------------+-------------+-------------+
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 |
| Version     | Header Type | Total Length |
|            |            |              |
| Identification | Flags | Fragment Offset |
|              |        |                |
| Time to Live | Protocol | Header Checksum |
|              |         |                |
| Source Address |      |                |
|              | Destination Address | |
|              | OPTIONAL | Options | Padding |
|              |          |            |          |
| IPSec Header Format |
| Protocol: 1 ICMP | 6 TCP | 17 UDP | 55 ESP | 51 AH |
```

**IPsec**

- RFCs for IPsec (v4 and v6)
- Specifies implementation
- Authenticates packets
- Prevents spoofed source addresses
- Encrypted packets (transport or tunnel)
- Prevents anonymity
- Does not specify policy
- Now includes key management
- Could use on a host
- Could use on a router (tunnel)

**IPsec services**

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets
- Confidentiality (encryption)

**IPsec – new IP security headers**

- Prevents spoofed source addresses

**IPsec services**

- Encryption without authentication is useless

**TCP keyed-MD5 option**

- RFC 2385 (BGP, LDP, MSOP)
- Authenticates routing protocols
- Include a keyed-MD5 checksum in TCP option field
- Each TCP segment carries keyed checksum
- Implemented in kernel
- Limited deployment: Cisco/Juniper routers, FreeBSD/OpenBSD
- setsockopt() to enable and set key TCP_MD5SIG
- Probably better to use HMAC as part of "application packet" or use secure transport (SSL, IPsec)
IPsec protocol

- Establish a security association in each direction
- Negotiate parameters/algorithms
  - exchange a secret (session key)
  - Authenticate
  - Not unique per SA
- Keyed hashes (MD5/SHA/RIPEMD-160) in HMAC
- Public keys (RSA/DSA) or pre-shared secret
- Block encryption
  - AES/DES/3DES/Blowfish/CAST/IDEA/RC5
- Diffie-Hellman (mod p or ECC)
- Tunnel and transport mode
- Requires modifications to OSI

Security Association (SA)

- Sender/receiver security info
- SA for each direction
- Maintained by kernel
- Identified by SPI (handle) and destination
- Specifications
  - Encryption key, IV, algorithm (DES, 3DES, CAST, Blowfish, AES)
  - Authentication algorithm (MD5, SHA)
  - Key lifetimes
  - SA lifetime
  - Security labels

SA

```c
/* Security association data for IP Security */
struct sa_data {
    u_int  len;
    u_int  type; /* Length of the data (for radio) */
    u_int  type; /* Type of association */
    u_int  state; /* State of the association */
    u_int  state; /* Sensitivity level (unused) */
    u_int rkey; /* Key length */
    u_int  rkey; /* Key vector length */
    u_int  rkey; /* Algorithm match index */
    u_int  rkey; /* Type of lifetime */
    u_int  rkey; /* Initialization vector */
    u_int  rkey; /* Key value */
    u_int  rkey; /* Lifetime value */
    u_int  rkey; /* Lifetime value 1 */
    u_int  rkey; /* Security association */
    u_int  rkey; /* Source host address */
    u_int  rkey; /* Destination host address */
    u_int  rkey; /* Originator of association */
    u_int  rkey; /* Transform private data: data */
    void *ig_data;
};
```

AH header

```
IvPv4 Header | Hop-by-Hop/ Routing | AH Header | Options | Upper Protocols

| IPv4 Header | AH Header | Upper Protocols [e.g. TCP, UDP] |
```

Transport and tunnel modes

- Tunnel mode has additional header that can specify different target (e.g. firewall or VPN)
IPsec encryption

- IP proto 50  RFC 2406  ESP

ESP header

- IPsec processing
  - use SPI from packet to look up SA
  - authenticate with info from SA
  - decrypt with info from SA
  - transport mode protects payload (host based)
  - tunnel mode protects entire packet
  - thwart some traffic analysis
  - used by VPNs (router/firewall)

IPsec implementations

- many OS vendors have implementations
- also shrink-wrapped VPN solutions
- export is a problem
- NAT may be a problem
- freeware OS patches
- policy: all (transparent to applications) or application request
- block connect() if SA established

NRL's API (standard?)

after socket():

setsockopt(fd, SOL_SOCKET, SO_SECURITY_AUTHENTICATION, auth, len = sizeof(int));

setsockopt(fd, SOL_SOCKET, SO_SECURITY_ENCRYPTION_TRANSPORT, esptrans, len = sizeof(int));

setsockopt(fd, SOL_SOCKET, SO_SECURITY_ENCRYPTION_NETWORK, espnet, len = sizeof(int));
IPsec key management

Internet Key Exchange (IKE)
- manual keying or automatic key establishment
- proposals (SKIP, ISAKMP, Photuris)
- SKIP
  - Light-weight
  - In-band
  - Diffie-Hellman (signed public keys)
- KE (ISAKMP/Oakley) RFC2409
  - out-of-band, daemon (UDP port 500)
  - negotiate (Oakley)
  - Diffie-Hellman, public keys

IKE

Oakley RFC2412
- key exchange protocol
  - speed vs more secure
  - ID vs anonymity
  - raw vs re-key
- Diffie-Hellman with authentication (5 groups: 3 mod exp, 2 ECC)
- authenticate with signatures, or public key encryption, or pre-shared secret
- cookie to thwart replay (pseudo-random number)
- Simple encoding of BIG integers (32-bit length, and n 32-bit length, MSWF)
- compromise of a single key will permit access to only
- data protected by a single key
  - key used to protect transmission of data MUST NOT be used to derive any additional keys

Aggressive Oakley key exchange

IKE – ISAKMP RFC 2408
- protocols and packet formats to establish, modify, and delete security associations
  - application on Alice wants to communicate with application on Bob, kernel policy says an SA is needed, Alice’s ISAKMP daemon is notified to get an SA with Bob
  - Negotiate algorithms, key sizes, type of association
  - phase I – two ISAKMP peers (daemons) establish a security association (SA)
  - phase II – negotiate and establish SA for requesting application (IPsec)
  - packet formats are chain of payloads
  - IKE is Oakley plus ISAKMP

ISAKMP formats

ISAKMP payloads

CNS Lecture 13 - 25

CNS Lecture 13 - 28

CNS Lecture 13 - 27

CNS Lecture 13 - 28

CNS Lecture 13 - 28

CNS Lecture 13 - 28
### ISAKMP exchanges

| Exchange | Exchange
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E = E</td>
<td>E = E</td>
</tr>
<tr>
<td>ISAKMP</td>
<td>ISAKMP</td>
</tr>
</tbody>
</table>

### The IPsec dance on the wire (tcpdump)

- **VPN client requesting secure tunnel (wireshark)**
- **IKE on UDP port 500**
- **-Encrypted File: IP proto 50**

```
06:52:00.852251 205.138.57.223.1147 > 192.31.96.188.500: udp 300
06:52:02.791434 192.31.96.188.500 > 205.138.57.223: ip-proto-50 76 (DF)
06:53:40.713605 192.31.96.188 > 205.138.57.223: ip-proto-50 76 (DF)
```

### Virtual Private Networks

- **Tunneling traffic over the Internet**
  - Usually more secure (PTT)
- **Most now based on SSL and may be interoperable**
- **Alternative: PPTP or remote dial-in**
- **Construct encrypted tunnels over the Internet**
- **Route-tunneled and standalone clients (routers)**
- **Use for internal privacy too**
- **Clients for Linux and Win***
- **No change to applications**
- **Network address translation (NAT)** makes client appear like its on local net

### VPNs

- **Components**
  - Cryptographic keys and software
  - VPN client software
  - VPN server software
  - Key exchange
  - Key transport
- **Criteria**
  - Platform, interoperability
  - Proprietary or open (IPsec)
  - Ease of use
  - Strength of security
  - Performance (server bottleneck?)
  - Mobile user support
  - Ease of management (key mgt.)
  - Network address translation (NAT)
- **VPN client software**
  - Connecting boxes/software
  - Cost (free clients?)
  - Proprietary or open (IPsec)

### VPN example

- **Remote user**
  - **Client connects to Internet**
  - **Click on VPN icon**
  - **VPN client software**
    - Connection is made to enterprise VPN server, e.g. port 443
    - Appear as a new IP address on enterprise network
      - Can access internal file shares
      - Access internal only web services etc.
      - No change to applications
  - **Tunneled traffic goes thru the tunnel**

### VPN example (tcpdump)

```
Before

06:52:02.791434 192.31.96.188.500 > 205.138.57.223: ip-proto-50 76 (DF)
```

```
After

06:52:00.852251 205.138.57.223.1147 > 192.31.96.188.500: udp 300
06:52:02.791434 192.31.96.188.500 > 205.138.57.223: ip-proto-50 76 (DF)
```
VPN establishment on the wire (ethereal)

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>TCP</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>172.16.0.1</td>
<td>172.16.0.2</td>
<td>UDP</td>
<td>5000</td>
<td>5001</td>
</tr>
<tr>
<td>192.168.0.1</td>
<td>192.168.0.2</td>
<td>ICMP</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OpenVPN

- Open source solution to VPN (windows/linux/*bsd)
- Authenticate with shared secret or public key (openssl)
- Tunnel IP or other frames over UDP or TCP
- Operates as user-space daemon (doesn’t use IPsec, no OS mode)
- Establishing a tunnel (a little routing/device trickery)
  - Client connects to server daemon and authenticates
  - New subnet addresses (temp) negotiated for tunnel endpoints
  - Tunnel provides secure (encrypted/authenticated) path
  - Client/server can use tunnel for NFS/login/print etc.

OpenVPN vs IPsec VPNs

- IPsec VPNs
  - Complex (IKE, SA)
  - Kernel support or mode to IP
  - Costly
  - Early problems with NAT
  - + enterprise implementations
  - Higher performance
  - scale
  - SecurID

- OpenVPN
  - /4 Can attach from any host
  - + operate at user level
  - + based on ISL
  - - doesn’t scale

Both provide pre-shared key or PKI authentication. Both suffer from allowing a foreign host/NET into the “inside” probably need IDS/firewall at VPN border.

PPTP

- Microsoft’s Point-to-Point Tunneling Protocol for VPN (’94)
  - Windows 95, 98, NT (Nice Try)
- Don’t need no standards or industry review invented their own
- Authentication protocol (broken)
- Hash function (weak)
- Key generation algorithm
- Used a known encryption algorithm, but effective key length reduced by user-chosen ASCII passwords
- Plus other implementation bugs
- The implementation was badly flawed, some later patches
- IPsec uses standard crypto and widely reviewed ...

Wireless

- wireless devices: VCR remote, PDA, cell phone, wireless ether
- wireless threats
  - sniffing content
  - user location
  - traffic analysis
  - becoming a mote (new/improvised)
  - becoming a base station
  - replay/offset
  - jamming (Dead)
- wireless defenses
  - spread spectrum
  - authentication
  - encryption at link level (LFSR, ECC, RSA)
  - access control list (allowable MAC addresses)
  - wireless protocols: WPA, 802.11i/WEP, GSM, CDMA, Bluetooth (weak security?), WiMAX
  - use IPsec or application security (advised) for end-to-end security
Wireless ethernet security (802.11i)

- Wired equivalent privacy (WEP) has some known flaws
  - Short IV, CRC, no key init.
- WPA (WIFI protected access)
  - Builds on WEP
  - Still uses RC4
  - temporal key integrity protocol TKIP
  - Longer IV (48 instead of 24 bits)
- Better message authentication:
  - Key hierarchy instead of single master key
  - Key refresh/update protocol
- Support for Extensible Authentication Protocols (EAP)

Cellular security

GSM (invented cryptosystem)

- compress, encrypt
- subscriber ID module (SIM) provides authentication
- challenge/response (AS)
- AS generates AS key
- encryption (AS)/brokered
  - AS is a stream cipher (3-64 bits)
- AS/2 only
- most cell phone security is weak

Link layer encryption applies only between cell tower and your cell phone, landline transmission is not encrypted

Network security

VULNERABILITIES

- denial of service
  - ICMP flood, DoS, unreachability
- impersonation
  - hosts, logins, etc.
- session capture
  - TCP key negotiation
  - TCP hijacking
- Server/application attacks
  - application flooding, IP spoofing
- buffer overflows
- Denial of service
- Improper configuration

COUNTERMEASURES

- denial
- configure properly
- denial by locking
- Denial of service
- session capture
  - TCP key negotiation
- Denial of service
- Server/application attacks
  - application flooding, IP spoofing
- buffer overflows
- Denial of service
- Improper configuration

Setting up kerberos

- get kerberos from MIT (cyphers)
- designate secure authentication server machine (KDC)
  - maybe place authentication servers for redundancy
  - build applications (p-utils, login, ftp, passwd, krb5, etc., kadmin)
  - PAM (plugable authentication modules) may make it easier
  - register principals (user, servers)
  - data base (principal/password) is encrypted with master key
  - install each server’s key on server
  - client-only easy (PC/MAC versions, Linux)
  - Can’t do RAS with ‘secure host’ needs your password to be known
  - How to install kerberos with is easy
  - Also a public-key version of Kerberos login, also smart card support

Kerberos components

- Key Distribution Center
  - KDC (Key Distribution Center)
    - Performs Authentication Service (AS)
    - Performs Ticket Granting Service (TGS)
    - Alternate/secondary KDC’s
- Servers
  - Server software (print, speaker, login, ftp) on various machines must be registered and have access control securely stored on the machine
  - Users
    - Account and Kerberos password
Authentication protocols

- See lecture 3, need confidentiality and timeliness
- Authenticaions: shared secret and/or public keys
- Worry about replay attacks
  - Malicious copies messages and replays it later
  - Replay a time-stamped message within the time "window"
  - Suppress original message, send it later
  - Reflect message back to sender
- Replay defenses:
  - Sequence numbers (bookkeeping problem)
  - Timestamp (hard to keep distributed clocks aligned, NTP)
  - Challenge/response (nonce) (hard for connectionless transactions)
- Kerberos (Needham-Schroeder) uses timestamps

Kerberos session

- User logs in, kerberized login sends <client name, TGS server name> to Kerberos AS
- Kerberos AS generates random session key (SK) and replies
  < SK_{tgs}, <client name, machine name, server name, SK_{tgs} > >
- On client, user's password is used to encrypt message
- To get ticket for another service, client sends a message to TGS, with authenticator encrypted with SK_{tgs}, the sealed TGS ticket, and the server name
- TGS generates a random session key (SK_{tgs}) and replies with
  < SK_{tgs}, <client name, machine name, server name, SK_{tgs} > >
- The client can send a request to the server consisting of the server's encrypted ticket, and an authenticator encrypted with SK_{tgs}
- The server can decrypt the ticket and get the session key SK_{tgs} and decode and verify the ticket (check for replay)
- Server adds 1 to timestamp and sends to client encrypted with SK_{tgs} (mutual authentication)

Kerberos exchanges

- Authentication of client to server
- Optional authentication of server to client
- Secure exchange of random session key
- Avoid plain-text passwords, permit a single signon
- Need Kerberos authentication

Kerberos credentials

- Authenticator
  - Name/instance/realm of the client
  - Timestamp
  - Encrypted with server's session key
  - Shows that the sender of the ticket is the same party to whom the ticket was issued
- Ticket
  - Server
  - Client
  - Time stamp
  - Lifetime
  - Encrypted with server's key
  - Generated by TGS
  - Good for a single client and server
  - TGS's voucher for the identity of the requestor of the service

Kerberizing

- You can add Kerberos calls to your own client/servers
- Need Kerberos database, authenticator, ticket-granting server, and administrative programs
- Can use klogon, but better if you have kerberized BSD utilities
- Kerberos calls added to login, r-utils, NFS
- "klogin -x" sets up encrypted session, every packet is encrypted
- Kerberos API (later)
Kerberos v4
- typical client/server application
- library requests just UDP packets
- Kerberos servers listening on well-known ports (UDP)
- encryption: modified DES-CBC
  - PLXK, hash weaknesses
- MAC: Juneman checksum on (key, msg)
- Random numbers (session keys)
  - Random time (syscall - time.tv_sec - "genunix4()" - counter
  - key = random()

Kerberos v5
- support of DCE
- more functionality
- new encodings
  - ASKIA data representation (v4: byte order data)
  - address encoding (v4: IPv4 only)
  - stronger random numbers (yarrow with /dev/random)
  - AB and TGS exchanges include a nonce instead of timestamp
  - selectable encryption/MAC
    - MAC, DES of md5/md4/DES-CBC
    - encryption/MAC, DES or md5/DES-CBC
- principal name multi-component
  - v5 uses name/instance/realm (4D names)
  - v5 name/realm
- new ticket flags (delegation) and longer lifetimes
- v5 will handle v4 requests
- more recently
  - public key for initial authentication (DNS using smart card to hold public key, need PIN)
  - one-time password support, kerberos auth
  - Kerberos v5 RFC1510

Kerberos v5 random numbers
- KDC generates random session keys
  - yarrow using /dev/random and packet interarrival times for random input
  - Initial seed from master key and (other realm keys if available)
  - yarrow keeps a fast and slow pool of random bits mixed with SM/1
  - 16 and 8 key, an entropy guess from random input
- Output bits are generated from 3DES using a key from the fast pool
- Wipes memory and saves pool to file on exit

v5 tickets
- proxiable TGT -- can be used to request tickets for a different net address (Alice can let Bob use her printer)
- forwardable TGT -- can be presented to a remote TGS
- lifetimes
  - longer lifetimes (v4: 21 hrs, v5: start/end)
  - renewable (by KDC)
  - postdated (good a week from now for 2 hrs, KDC clears INVALID flag)

Why not Kerberos?
- every network service must be modified
- Kerberos server must be physically secure with hardened OS
- export restrictions
- off-the-wire password attack on message from KDC to client
- if password is declined, causekrpr can decrypt other tickets
- spoof servers and users
- steal kerberos ticket

Still better than anything else:
- also used in DCE and Windows XP
- part of DNS single-sign-on common access card
- public key (on smart card) login to kerberos
Other variations

SESAME
- European project
- based on Kerberos
- uses public key
  - ticket encrypted with user's public key
  - AS stores only public keys, not as vulnerable

CORA - technology for distributed applications
- set of specs
- object request broker (ORB)
- security spec released '95
- authentication, access control, audit, message protection

Secure operating systems

- design issues
  - hardware features
  - OS features
  - design principles
- Examples
  - Microsoft, TMACH, SELinux, OpenBSD
- evaluation methods
  - Orange book, common criteria, FIPS
- Why secure applications are not enough, NSA -- really need secure OS (required reading)

features for security

Hardware
- privileged state
- privileged instructions
- memory protection (RO,NX)
- timer interrupt
- system call (trap/SVC)
- hardware RNG?
- TPM?

OS
- user authentication
- memory protection
- file/device protection
- access control (DAC, MAC)
- scheduling
- interprocess communication

Secure OS design

- design security in from the beginning
- define access rules for subjects/objects and allowable info flows (security policy)
- label objects (top secret, secret, unclassified, read-only)
- least privilege (fine grained)
- small kernel (TCP)
- separation (temporal, logical, cryptographic), layers or compartments
- complete mediation -- every access checked
  - mandatory access control (MAC)
- default is deny
- ease of use
- apply software engineering principles

Trusted platform module (TPM)

- Window's Vista, Linux?, coming to MAC OS
- Attached processor for storing keys, doing crypto, RNG
- Tamper resistant
- Authenticate user/system (digital rights)
- Secure startup, trusted OS query, disk encryption (stolen laptop)
- Code signing
- Trusted computing group "standard"
- Number of vendors making TPMs

Access control

- access matrix
  - (subject,object,access)
  - by subject: access control list (ACL)
  - what the subject/user has access to
  - maintained by OS
- role-based access control (RBAC)
  - Each process/user has a "role"
  - Requires new administrator
  - A user may have several "roles"
  - Certain files specify which domains
    a role has access
  - Can associate a role with a "method"
  - Easy to change privileges of a role
  - Principal of least privilege
**TCB**

**Trusted Computing Base**
- reference monitor
- implement and enforce security
- authentication and access control
  - No read-up or write-down
  - Labeled objects, supervisor clearance
  - “need to know” role-based
- tamper resistant
- can’t be bypassed
- small/easy to correctly implement
- verifiable?

---

**Secure OS’s**

- Multics
- Sicom
- KSOS (3-layer, PDP-11)
- kernelized VM/370
- secure UNIX/Linux
  - UCLA Secure UNIX
  - TMACH/DOS
  - CMWs, NetTop
  - Trintel/Linux, NSA secure Linux (SELinux), Grsecurity
  - OpenBSD

---

**Multics**

- B2 multi-level secure OS (mid 60’s)
- multi-layered (rings)
- extension of 2 layer systems
- upper rings implemented in software (processor: GE645)
- kernel, ring O
- “traps” for requesting ring services (gate)
- uses capabilities
- segmented address space
- design review (PL/I)
- need more hardware assist (ring-crossing)

---

**UCLA secure UNIX (1979)**

- three layers
  - user/supervisor/kernel (PDP-11)
- small secure kernel
- kernel interface layer
- application layer
- all security functions in one place
- capability list maintained by kernel
- objects: processes, pages, devices

---

**TMACH**

**Trusted MACH**
- based on UNIX/MACH kernel
- with OSF UNIX personality
- sponsored by DARPA and NFL
- available for 486/586

**TMACH goals**
- B3 security — data integrity
- portable to many platforms
- competitive performance
- extensible
- UNIX personality

**TMACH architecture**
- kernel, small, secure
- layering, modularity, abstraction, data hiding
- TCB servers, multi-level secure servers
- non-TCB code, the OS user interface

---

**TMACH security**

- labeled objects (mandatory model)
- user clearance
- mandatory access control
- access mediated by reference monitor
- auditing
- ACLs
- trusted path for user authentication
- subdivided privileges (least privilege)
- task communication (portes)
- covert channel analysis
- trusted startup and recovery
- security model (Bell and LaPadula)
Secure Linux

- NSA's Security-Enhanced Linux
  - Mandatory access control (Type-enforcement and RBAC)
  - Separation of information based on confidentiality and integrity
  - Based on Linux Security Modules (LSM)
- Latest Enterprise Fedora has SELinux option, file ACL's, firewalling
- Hardened Linux
  - D/selective (script to harden configuration)
  - Expert (Secure Linux)
  - Openwall Linux (Own)
  - PAM/LSM
  - Grsecurity
  - Support for TPM

Security validation (assurance)

- demonstrate the security of an OS or application or crypto device
  - verification
  - penetration analysis
    - test security services
    - try to bypass security
    - tiger team
    - demonstrate presence of errors, not absence
  - informal validation
    - requirements checking
    - design and code reviews
    - module and system testing
    - certification (Orange Book or itsec or Common Criteria)

Example ACL

Printer daemon ACL (Grsecurity)

```
Subject /usr/sbin/cupsd a {
  /b
  /etc/cups/cupsd
  /etc/cups/cupsd/D wod
  /etc/group c
  -CAP_ALL +CAP_CHOWN +CAP_DAC_OVERRIDE
  bind disabled
  connect disabled
}
```

- Configure services to have certain capabilities and access rights
- Configure users to be part of certain groups/roles
- Principle of least privilege — even if you buffer overflow cupsd ...

Secure the Linux kernel

- Need lots of "hooks" in kernel source to check permissions
  - File open/read/write, process create, sockets, ...
  - LSM model (modules) SELinux
- Grsecurity others require kernel patches
  - Non-trivial configuration files to establish "policy"
  - Grsecurity has a learning mode to establish what privileges a process needs
  - User roles can be restricted
    - You can only read mail (can't compile)
    - Different administrative privileges
  - SELinux SELinux SELinux SELinux or or or or Grsecurity Grsecurity Grsecurity Grsecurity

OpenBSD

- Most secure of the open source Linux
- Developed in Canada, so crypto software included
  - Libresc v5
  - opencrypt /opencrypt
  - Support for crypto hardware
- Immutable and append-only files, no writing to /dev/mm / dev/kmem
- Actively audit source code looking for vulnerabilities as well as timely patches for bugs discovered by "others"
- Default installation — non-essential services disabled
- PRNG (/dev/random) (/dev/random is for hardware RNG)

Hardening Linux

- Disable net services
- Reduce setuid programs
- Active password mgc/removal
- Default mask
- Limit root login locations
- Set BSD password
- Get rid of r-utils
- Login banners
- Remote logging
- More detailed event logging
- User-specific /tmp areas
- ... read the books

Latest RedHat/Fedora has SELinux option, file ACL's, firewalling

Security

Different administrative privileges

Hardened
- SELinux SELinux SELinux SELinux or or or or Grsecurity Grsecurity Grsecurity Grsecurity

Support for TPM
- Active password mgt/renewal
- Firewal enabled
- Enable NTP
- Enable port scan detector
- Incorporate StackGuard/PaX
- Support no-exec data areas (NMA)
- Auto patch/update
- Immutable/append files (chattr)
- Encrypted file systems
- SELinux or Grsecurity

Incorporate Incorporate Incorporate Incorporate StackGuard/PaX StackGuard/PaX StackGuard/PaX StackGuard/PaX

- Securing the Linux kernel
- Demonstrate presence of errors, not absence
- Firewall enabled
- Enable NTP
- Support no-exec data areas (NMA)
- Auto patch/update
- - Encryption file systems
- - SELinux or Grsecurity

Developed in Canada, so crypto software included

Read the books
DoD certification (historical)

Requirements for a secure OS
- security policy
- authentication
- labeling of objects and subjects
- accountability (audit log)
- assurance (enforce/evaluate)
- tamper resistant
- documentation

Orange book ratings
- D -- minimal (D for DOS)
- C1 -- discretionary
- C2 -- controlled access
- B1 -- labeled
- B2 -- structured
- B3 -- security domain
- A1 -- verified

Might be able to add features to an OS to qualify for C1-B1
- B2 requires security part of OS design
- B3/A1 provable model of security

Ratings
C1 -- discretionary access
- memory protection (user vs OS)
- object access control (ACL)
- user authentication (password)
- discretionary access control
- penetration testing
- e.g., MVS with RACF

C2 -- controlled access
- single user access controls (ACL)
- audit logs (tamper resistant)
- object reuse, protect memory, files, swap
- e.g., MVS/ACF2, VMS, DEC UNIX

Ratings
B1 -- labeled
- mandatory access controls (privacy)
- labeled objects (incl. devices)
- label printer output
- prevent read up and write down (Bell-LaFakula)
- analysis and testing of design and source code
- informal model of security policy e.g., DMW's (compartmentalized mode workation)

B2 -- structured protection
- test and review of design
- principle of least privilege
- trusted path (user/_root/OS)
- security kernel (TCP)
- programs must report security level changes
- covert channels identified and bandwidth estimated, e.g., Multics

Covert channels
- ways to reveal info to lower level
- low-bandwidth
- difficult to eliminate
- use memory/file or timing
- existence of a file (bit 0 or 1)
- mode of a file (O or 1)
- high CPU load (O or 1)
- process name (O or 1)
- network connection (O or 1)

Ratings
B5 -- security domain
- stronger design
- ACL supports user denial
- ACL supports read/write (integrity)
- full access control
- active audits (alerts)
- penetration resistant
- secure startup and crash, e.g., TMACH (applied for B5)

A1 -- formal verification
- formal, provable security model
- top-level specification
- demonstrate spec follows model
- implementation consistent with spec
- formal analysis of covert channels, e.g., Biome
Orange book shortcomings

- Miss one feature, lose rating
- Local software invalidates?
- OS patches invalidate?
- Mandatory access controls cumbersome
- Write-up would allow virus

"If get a C2 rating (96)
But epoxy/shut floppy no network
"Compaq 386"

UK’s itsec

UK software certification ("99), fast-track assessment, list of certified products

- Assurance levels
  - E0 – no assurance
  - E1 – informal architecture design, some testing, secure dot
  - E2 – penetration tested, audit trail
  - E3 – source code/drawings, acceptance procedure, re-test
  - E4 – formal security model, configuration control
  - E5 – independent configuration control
  - E6 – formal architecture description and correlation with design and testing

Common Criteria (ISO 15408 '99)

Combine US and EU criteria

Functional requirements: desired security behavior

Assurance requirements: assure claimed security measures are effective

Assurance levels
  - EAL1 – functionally tested
  - EAL2 – structurally tested
  - EAL3 – methodically tested and check
  - EAL4 – methodically designed, tested, and reviewed
  - EAL5 – semi-formally designed and tested
  - EAL6 – semi-formally verified design and tested
  - EAL7 – formally verified design and tested

FIPS 140-1

Crypto module security (hardware, e.g. encrypers, crypto cards)

- Level 1 – uses FIPS approved algorithms
- Level 2 – tamper-evident seals, role-based authentication (C2)
- Level 3 – self-destruct (zero its self) if tampered with, identity based authentication (B1)
- Level 4 – sophisticated tamper detection and zeroing, resist environmental attacks (state loss), (B2)

* The problem of establishing that a program, application, or OS meets any particular security requirement is known to be fundamentally unsolvable!
Next time ...

Writing secure code
Crypto API's
Secure applications

Lectures:
1. Risk, viruses
2. UNIX vulnerabilities
3. Authentication & hashing
4. Random #s, classical crypto
5. Block ciphers DES, AES
6. Stream ciphers RC4, LFSR
7. MIDTERM
8. Public key crypto RSA, D-H
9. ECC, PKCS, ssh/ftp
10. PKI, SSL
11. Network vulnerabilities
12. Network defenses, IDS, firewalls
13. IPsec, VPN, Kerberos, secure OS
14. Secure coding, crypto APIs
15. Review

CS594 project due: December 1

CS594 project due: December 1