System Security Capabilities and Uses

(Some Things To Look Out For)

Glenn C. Everhart
(Everhart@star.enet.DEC.Com, Everhart@GCE.Com)
Digital Equipment Corp.

Securing A System

Facets:

Authentication (being sure a user is who he says he is)
Access Control (What the user may do)
Audit (Record keeping in case "something happens")
...all in accordance with a ***policy and security model***.
(Dominant security model is military.)

Authentication:

OpenVMS, UNIX, WNT all use passwords for local system. Networks pose a threat; knowing your user over a large area is vulnerable to many problems.

Threat: (a short list of general types)
Wiretaps (including ethernet "hardware assisted tap")
(also Tempest types, for completeness)
Untrustworthy hosts
Untrustworthy protocols
Untrustworthy programs

Wiretaps:
Ethernet packet grabbers are the most common wiretap class threat.

Countermeasures:
TDR (to find unauthorized taps)
routers (keep traffic off wires where it isn’t needed)
Use of FDDI (which has no promiscuous mode)
Specialty boxes like DESNC

Untrustworthy hosts:

Any host with a dishonest person who can control it. Can spoof addresses or protocols. Many attacks on IP exist at the routing level. DECnet host names can be forged:

Countermeasures:

For DECnet, node SEND and RECEIVE passwords for each node (and default access NONE; set access for each machine.)

Kerberos or similar authentication suites

NOTE HOWEVER that published TCP attacks exist which threaten even Kerberos traffic if someone in the middle can sniff the net between two points trying to use it.

Untrustworthy protocols:

Many common protocols like NIS (YP), NFS, have holes which permit anyone who can access them to gain system access. The X windows protocols can be particularly vulnerable (depending on transport and authentication options.) Freely available programs can connect to an X server and ask it to report keyboard events. ANY access to the node (if tcp/ip) means all key strokes can be read (passwords...). X has no built in security; OS is expected to do it. Note again that TCP/IP reports only source machine. DECnet reports machine and user, a bit better. It has been reported that NetBEUI allows NO security control under NT.

Know Your System Limitations.

Remember that NT, OpenVMS, UNIX are only C2 certified as standalone systems; NOT on a net. Many protocol problems have been reported (e.g. no security in MS Access, at least at one time, NT Netware implementation doesn't ask for passwords for nonexistent accts).

Systems that have been around longer have fewer problems. (For some details on MS OSs, http://www.c2.org/hackmsoft, among other sites.) Microsoft is working on a crypto based authentication hierarchy system. Also, various other sets of protocols exist for authentication, file sharing, etc., but the ones below are most commonly used.

A little detail on some of these.

NFS (Network File System)

NFS is "the" filesharing method in UNIX. Used also in OpenVMS and NT.

Vulnerability #1:

NFS checks its authorization to move data only at open, and does not check the integrity of the transfer. Anyone who can trick TCP/IP into routing through them (or who can send data fast enough on a LAN) can corrupt files. This has been used to insert Trojan horses in critical files, corrupting even critical Kerberos code.

Note that DECnet FAL transfers have an end to end CRC which prevents this, and DECnet+ has an option to do end-end checksums on everything.

NFS authentication is usually via UNIX style" IDs which require all systems to have the same ID (number) to name mapping.

(from a recent vmsnet.networks.tcp-ip.multinet article:)

```
Access to files via NFS is via UNIX-style UID/GID. An NFS client
       learn the UID/GID that a user is supposed to use by (1) a
       local file which contains username (such as passwd on UNIX,
       or the NFS configuration information on (e.g.,) a Multinet
       client), or (2) by sending a username/password to a PCNFSD
       server, which authenticates the username/password and replies
       with the UID/GID.
Now, if someone with lots of time on their hands (a student, let’s
       say their NFS client enough, they’ll eventually find some way
       of causing NFS to emit any UID/GID they wish. Exact methods
       are left as an exercise for the reader. The "solution" is to
       setup mount restrictions, so only "trusted" NFS c certain
       mountpoints (directory trees).
```

Countermeasures:

Cryptographic authentication options and systems, to handle authentication. Securing data really requires encryption if any part of the data travels over untrusted wires. Ethernets are very frequently not trustworthy.

Some protocols have optional authentication which is not widely used.

Try AFS (or DCE/DFS), which were designed for much stronger security.

NIS (See D. Hess, D. Stafford, U. Pooch, Texas A&M)

NIS (formerly Yellow Pages) is a protocol that propagates user ID to number mappings (and others) around a net and does authentication.

NIS is based on RPC (Remote Procedure Call. RPC authentication defaults to none. It may use "UNIX" authentication or DES (which encrypts timestamps).

NIS config files are world readable, since NIS doesn’t alter the maps. Client security can be very poor.

Any machine on the net can pretend to be the NIS server (easy if UDP is used for transport, as is usually done). The fake NIS server need only beat the real one to the draw and can claim anyone is a legitimate, privileged user on another machine.
With the "R" utils this can get any access a client allows to a trusted user (for example). The client can’t find out.

Since EVERYTHING tends to use NIS, this means all authentication gets broken. Even breaks DES (since the server is where the keys come from).

Countermeasures:

Kerberos is stronger.

DNS (Domain Name System)

Many TCP/IP programs use authentication by name, not IP address. The DNS does this mapping. It’s distributed.

The problem is that one can fake being a DNS server, and cache is used to store DNS names. Current algorithms can be lied to, and one DNS response can "piggyback" enough extra info so the IP to name AND the name to IP maps can be corrupted at one time in the local DNS cache. This means name based authentication can be compromised, and any "distributed trust" authenticated only in this way can be defeated.

Defences:

No generic defenses exist. If the DNS code would prefer authoritative cache data over non-authoritative, and caches didn’t replace old data with any new data, the attack would be harder. Crypto DNS systems may help. Someday.

Note also that attacks on most IP protocols exist and have been reported years ago. Some just haven’t gotten into the wild yet. The recent spate of SYN attacks, mailbots, cancelbots, etc. is the tip of the iceberg.

Be very careful what you trust off your machine. Again, this is TCP/IP specific. DECnet, for example, has other issues with people faking nodenames, and you fight that by ALWAYS using link passwords for access (and set default access to none).

This sort of attack is most common in TCP/IP in part because of its ubiquity and wide source availability.

Untrustworthy programs:

This refers to programs with covert behavior,
particul "Trojan horse" apps that do secret things when they ar run with privs.

Countermeasures:

KNOW YOUR SOURCES !!! This is not a panacea for **all** security problems, but makes vandalism much less likely. Something like Internet Explorer, for example, is very complex and has been reported to have bugs able to disclose anything on its machine, though this was doubtless **not** the authors intent.

Run in a nonprivileged environment, and use tools like SET WATCH (shows file accesses tried), FTS (on VAX) to show what system services are used (and fake $setpr optionally), or system service intercept by Brian Schenkenberger (see S96 sigtapes) which is suitable for $setprv, $gettim, and other intercepts fo selected processes. Watch for $getjpi, $setprv calls to set privs, or ask why $gettim may be called. In hard cases, DISM32 or similar tools can be used to recreate MACRO-32 if on VAX. ($Getjpi priv read or $gettim bomb timer)

Try to arrange never to run untrusted code under a priv'd process. An access ACE and an audit listener mailbox client can be built to attempt to enforce this. (A pre-activation checker is better still.)

Authentication of scripts for tampering is a good idea if you can do it. In OpenVMS, Network-1 has a tool. Run a system checker periodically to look for tampered with code. Won't stop access, but will tell you about it before you're exposed for long. Polycenter has one. UNIX tools at CERT help for UNIX.

Limit what unknown programs can do. The subsystem facility of OpenVMS can limit a program's access if set up right. A 3rd party product can further limit access by privs, time, image, user, location, and "trustworthiness rating" and prevent accidental privilege grants and check file corruption. With such a system in place, files protected cannot be accessed by Trojan code, since the Trojan code cannot be given privs to do damage.

Know the capabilities of your system.

User Authentication:

The key to system access is to have an identity the system knows.

* Passwords when interactive. (Challenges, secure channels...)
  USE the "evasion" features in your system
  if you have them. USE password policy model
  and checkers to be sure passwords are strong.
  Pick good pass-phrases; do not force frequent
  change (becomes algorithmic or is written down).

* Some distributed trust mechanisms exist. Examples:
* DECnet proxies, "R" utilities
  EXTREME care in proxy access with privs
  Use FAL$LOG=1/disable=8 (DECnet IV)
* "R" utilities (trusted users or hosts)
  EXTREME care in this for anyone who can
  become superuser, system, or administrator (UNIX, OpenVMS, NT)

* Routing information (network firewalls).
  Several CERT advisories about this. Some
  documented IP attacks still exist that have
  not yet been seen in the wild.
  Trust hosts sparingly.
* Most systems assume *programs run under a user's name* are to
be treated as *agents of **that** user*. For the most part
OpenVMS, UNIX, and NT check user access, not user/program.
  Extended checking software features can be
  helpful with the most critical data.
  The "military model" is inadequate for
    control of "insiders" and does not
    encode "need to know". Only username
    is generally used, but location,
    program used, priv level, time, and
    various other info exists, could be
    used to specify what's allowed.
* Automatically downloaded software (Java, ActiveX, etc.)
    needs an extended definition of security. The OpenVMS
    "subsystem" facility begins to address this. (The 3rd
    party "Safety" tool "paranoid mode" goes further.)

MANY holes have been reported in Java (see Dean, Felton and Wallach, Princeton Univ. CS dept. 1996). The issue is that an automatically downloaded app cannot sensibly be called "agents" of the user running a web browser. To provide security, ActiveX attempts to ensure that the sender's identity is known. This (if do-able) makes virii less likely, but does not mean that a program might not be used to copy or alter data on a machine. Thus the ActiveX scheme helps the common "PC" problem but not the commercial security major issues. *From the Java FAQ:*

```
Q: Won't digital signatures solve all the problems?
A: No, they'll only help a little. Digital signatures let you know
      who wrote an applet, but they don't decide if you can trust
      the author.
```

Nor do they help you decide how secure the author's work is against things like information theft using it. (Recall the IE example; holes in a complex package may allow misuse even from the best known sources.)

Java holes in implementation have been reported. However, deeper problems exist, among them:

* There is no formal security policy for Java.
* There is no logging of security events in Java.
* There is no trusted computing base.
* There is no single line of defense, but security functions are spread all over the code.
* The bytecode verifier is impossible to check for security.
* Java is a language and a security barrier. The policies for these two objectives are not the same.
* At least a few covert channels exist which can be used to send information or subvert the Java security system.

It seems likely that a rework of Java would be needed to provide a secure technology. Javascript is also vulnerable.

The stronger the underlying OS's abilities to guard your data, the safer you are using tools like Java or ActiveX, if those tools are used. OpenVMS can be set up more robustly than most.

The most common security threat:

"Social Engineering" (i.e., plain old deception of humans) is a serious threat. (Watch someone type a password, or get a password changed over the phone or print & mail a document...)

Some forms of these threats can be helped where clerks are duped, IF the clerks are themselves limited in how they can get information. Education and auditing are needed too.

Responses To Threats:

* Educate your users.
* Idle account reuse

  Clean out all residues of accounts (includimg ACEs) promptly. NT user ID is *never* reused. (DISUSERing an acct can be better sometimes detect attempts at reuse.)
* Spoofing or untrustworthiness of something on another machine. Authenticators can be added for some types

of access, to ensure you’re dealing with the user and machine you think. The better ones encrypt their traffic. They can also be use permit safe access by category to data you need to share internally but not all over. Firewalls *internally* can be very helpful if community topology allows.

* Weak default protections
Run checkers like COPS or any of several on the sigtapes for OpenVMS, or the Polycenter too, to test. (The WNT security FAQ is on the OpenVMS/LT sigtapes too.)
Volume protections to isolate classes of users work like mandatory controls. If someone cannot access the volume, they can’t get any file on it. Virtual disks help here. On OpenVMS you can use volume ACLs to isolate volumes from, say, network access.
If information is private, use a cryptodisk, and for safety keep the key value locked in a safe. (You might get run over by a truck, after all.)

Access Control:

Access control means the facilities the system uses to control who gets what access to what objects.

OpenVMS, UNIX, and WNT all rely on a user name which becomes a magic cookie.

* In UNIX and WNT, a user can belong to many groups (sometimes limited to a fixed maximum number like 8 or 16). Access to objects can be as the user, or as a group member.

* OpenVMS allows a user to hold many rights identifiers (as many a desired). Access can be as the user or as a holder of an identifier.

* Both OpenVMS and WNT have notions of privileges which bypass security checks; details are different and some WNT privileges are relevant to certain utilities only. UNIX has a single privilege, "Superuser" for all functions. OpenVMS and UNIX have notions of specially-trusted applications (install with privilege or with suid/sgid privilege). (OpenVMS also has an "install with identifier" notion, the subsystem facility.)

File Access:

* File access in OpenVMS and UNIX can be specified with a "UIC" shorthand (descended from pdp10 Monitor conventions) giving a user or group number. OpenVMS and some modern UNIX versions also support ACLs.

* WNT has no "UIC" heritage but inherits the MSDOS file protection scheme in addition to its ACLs. This overrides ACL protections if present.

* OpenVMS allows files in a directory to have a default ACL copied to files as they are created. WNT allows default ACLs in parent directories as well as a default user ACL. OpenVMS does not have a default per-user ACL. File ACLs in WNT can be inherited from parent directory OR user default.

* WNT allows some limited "SYSTEM" ACL entries to exist for limited purposes (i.e., audit). OpenVMS has the notion of protected ACE which may also be hidden. These OpenVMS ACEs are fully general, not only for auditing.

* Major difference: OpenVMS stores security information **with** the file (in the header, actually). WNT stores it separately in a "registry".

OpenVMS approach means the security information comes "along for the ride" during file access. Thus access is fast. When trying to remove a user, removing all ACLs that mention that user can be a chore. (Tools on the SIG tapes exist to help).

WNT approach makes management easier. Cache can attempt to reduce overhead of lookup of security information. Where a file has no security information, its ancestors are examined. Thus only one ACL might cover a large part of a volume, many directory levels deep.

* Access controls:

UNIX: Read, Write, Execute, SUID, SGID
OpenVMS: Read, Write, Execute, Delete, Control
WNT: Read, Write, Execute, Delete, Control, TakeOwners, Read-ACL

* Impersonation services exist in both systems, for use by trusted servers. Capabilities are generally similar

* Correct setup rules of thumb, OpenVMS OR WNT:

Back up data often enough and GUARD the BACKUPS
Protect what is important most carefully
Be sure default accesses correspond to policy

Important Differences of Detail between OpenVMS and WNT

* Registry vs. security info with object
* ACLs only, with inheritance and user defaults, vs. ACLs and UICs
* Details of ACLs and privileges differ
* WNT default if no ACLs are present is completely open access.
* WNT also defaults to a "guest" account requiring no prior authentication at all.

Audit

* OpenVMS and WNT have facilities for audit of security related events.

* OpenVMS facilities are well developed

* Problem of audit is volume of data.

* Several products exist to use the "listener mailbox" of OpenVMS Audit to report suspicious actions in c to real time.
* Audit can tell when the horse leaves the barn and what is going on. This is only useful for some threats

* Pattern matching in console input watchers can also respond to snooping for info around the system.

* A "security patrol" is worth having if you can.

Keep audit trails, offsite if possible. Keeping audits on write-once configurations worth thinking about.

Conclusion:

* Know and control your user population and its permissions
* Avoid untrusted protocols
* Check your security against local policy with tools, commerc or free
* "Walk the beat" to look for strange things
* Set up access permissions, especially on critical data, to allow only what is needed for people to do their jobs
* Consider resetting default permissions if they are too loose for your policy.