System Security Capabilities and Uses

SE001

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

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:

VMS, Unix, WNT all use passwords for local system
Networks pose a threat.

Threat:
Wiretaps (including ethernet "hardware assisted tap")

(also Tempest types, for completeness)
Untrustworthy hosts
Untrustworthy protocols
Untrustworthy programs

Ethernet packet grabbers are the most common wiretap class threat.

Countermeasures:
TDR (to find unauthorized taps)
routers
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.
Kerberos or similar authentication suites

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.)

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

Access to files via NFS is via Unix-style UID/GID. An NFS client learns the UID/GID that a user is supposed to use by (1) a local file which contains username to UID/GID mappings (such as passwd on Unix, or the NFS configuration information on a MultiNet for OpenVMS client), or (2) by sending a username/password to a PCNFSD server, which authenticates the user-name/password and replies with the UID/GID.

Now, if someone with lots of time on their hands (a student, let's say) pokes around on their NFS client enough, they'll eventually find some way of causing the NFS client 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 clients can access 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.

Untrustworthy programs:

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

Countermeasures:
KNOW YOUR SOURCES !!!

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 $setprv optionally), or system service intercept by Brian Schenkenberger (see S96 sigtapes, upcoming) which is suitable for $setprv, $gettim, and other intercepts for 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.

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. 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 months. Polycenter has one.

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 modules
-> 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
        -> EXTREME care in proxy access with privs
        -> Use FAL$LOG=1/disable=8
    * "R" utilities (trusted users or hosts)
        -> EXTREME care in this for anyone who can
        -> become superuser
    * 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***
    VMS, Unix, and NT check user access, not user/program.
        -> Extended checking software features can be
        -> helpful with the most critical data.

"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.

Threats:

* Idle account reuse
  -> Clean out all residues of accounts (including
  -> ACEs) promptly. NT user ID is never reused.
  -> (DISUSERing an acct can be better sometimes to
  -> 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 used to
  -> permit safe access by category to data you want
  -> to share internally but not all over.
  -> Firewalls internally can be very helpful if work
  -> community topology allows.

* Weak default protections
    -> Run checkers like COPS or any of several on
    -> the sigtapes for VMS, or the Polycenter tools
    -> to test.
    -> Volume protections to isolate classes of users
    -> work like mandatory controls. If someone can’t
    -> access the volume, they can’t get any file on
    -> it. Virtual disks help here. On VMS you can
    -> use volume ACLs to isolate volumes from, say,
    -> network access.
    -> If information is private, use a cryptodisk and
    -> keep the key value locked in a safe.

Access Control:

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

VMS, 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.

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

* Both VMS 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. VMS and Unix have notions of specially-trusted applications (install with privilege or with suid/sgid privilege). (VMS also has an "install with identifier" notion.)

File Access:

* File access in VMS and Unix can be specified with a "UIC" shorthand (descended from pdp10 Monitor conventions) giving a user or group number. VMS 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.

* VMS 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. VMS 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. VMS has the notion of protected ACE which may also be hidden. These VMS ACEs are fully general, not only for auditing.

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

> VMS 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.

> 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.

System Security Capabilities

* Access controls:

    Unix: Read, Write, Execute, SUID, SGID
    VMS: Read, Write, Execute, Delete, Control
    WNT: Read, Write, Execute, Delete, Control, TakeOwnership, Read-ACL

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

* Correct setup rules of thumb, VMS 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 VMS 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.

WNT Security Under VMS

* An experiment I have done...not a product nor even a project
* Done to see what was needed to handle a "foreign" model in VMS

* Uses my I/O intercept technology:

> For files, intercept disk driver FDT ahead of file system
> Where appropriate read tags on files in a kernel thread and use kernel data to ensure nothing that should be examined is omitted
> Notify a server (which winds up acting as registry or talking to it) when open/create/delete/close etc. are done and pass it the current request info, where to send a SKAST. Make the proces wait for reply.
> Server runs, does additional checks as needed and returns "yea" or "nay"
> Server sends SKAST to the address (in the process) and the SKAST causes the process to be unwaited, and its control thread then resumes context and either does the I/O or fakes failure

> Similar schemes for generic system services are feasible and would work much the same.

* The server contains logic to read a user database containing groups, default ACLs (user and system) and also a default command (an added hook, to be replaced later by a more general notification scheme).
* The server reads a file database containing NT information on file security ACLs etc., or chains up to parent directories (uses backpointers) to find any default ACL with inheritable ACEs if any
* Access is tested according to NT type rules.
* Create handled by access for all NT - security users and will flag a file as using default ACL for user or inherited from above if appropriate. Does not need to propagate records, but will use user default ACL if present. Also will fill in a fully general system ACL, which is tested before user ACL.
* User and system databases are only permitted to be edited by an editing app, which enforces permissions on who can do what. The intercept itself is used to control this, "knows" where these databases are and what apps may access them.

In addition the intercept allows access controls based on

* User Id
* What time of day it is
* Where the user is physically
* What image is being used
* What privileges are present (denies access if EXCESS privs are present)
* Whether the file has been tampered with
* Whether the file is tagged for passwords and a password has been pre-entered
* What kind of access is requested
* Replacement privilege masks (to lower privileges while doing access)

... In addition to normal VMS security.

Access failure is logged, and files can be hidden from unauthorized access.

Note that the access controls here are not the usual military model. The idea is to MACHINE ENCODE "need to know" so that normal activities tend to be permitted, but access outside those activities can be controlled and audited. Control of excess privileges is important since privileges otherwise can become a total security bypass.

This sort of control is appropriate to major corporate data bases, customer lists, bank account records, payroll files, future plans, etc., where the files need to be shared, but where loss or damage can harm the data owner seriously.

Audit

*VMS and WNT have facilities for audit of security relevant events

* VMS facilities are well developed

* Problem of audit is volume of data.

* Several products exist to use the "listener mailbox" feature of VMS Audit to report suspicious actions in close to real time.

* Audit can tell when the horse leaves the barn and what was going on. This is only useful for some threats, not all.

* Pattern matching in console input watchers can also give clues to snooping

* 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, commercial 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.