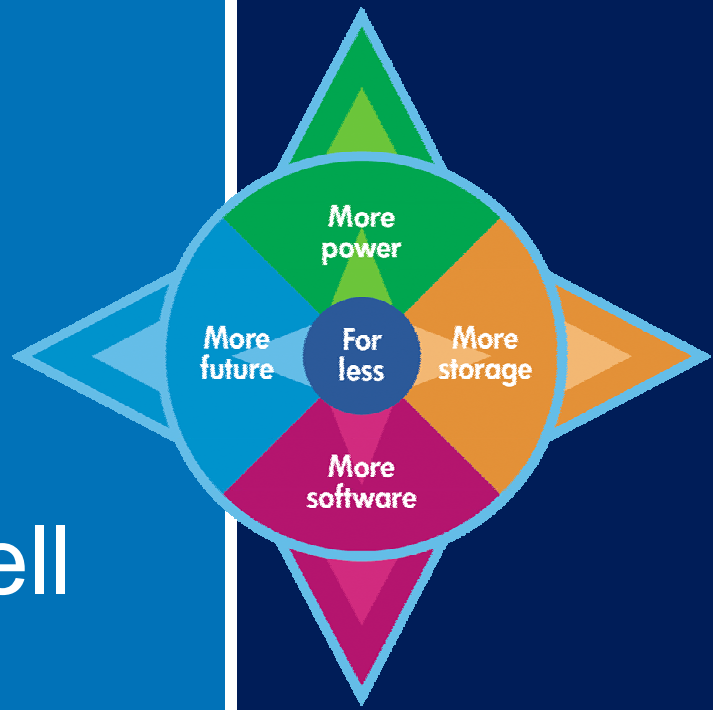
How To Harden Your NonStop Server: A Security Show and Tell

part 2 : OSS



Roland Lemoine /Joachim Schmitz
Support specialists
HP Global Customer Support Center

Session overview

- Cover ALL security aspects within ONE track
- Talk and show
- Demonstrate hot spots and present solutions

Session Overview

- Part 1 of 3 – GUARDIAN
  Carl Weber (GreenHouse)

  - How secure is NSK?
  - Can be broken in? Easily?
  - Is there an easy way to prevent it?
  - Solutions!

Session Overview

- Part 2 of 3 - OSS
  Roland Lemoine / Joachim Schmitz (HP)
  - Are we like Unix?
    11 common Unix security holes
  - OSS security features:
    Leverage Safeguard features for OSS
  - SSL - enable your middleware
    (iTP Webserver, Java and WebLogic,etc...)

Session overview

Part 3 of 3 - LAN

- Arrigo Triulzi (K2Defender)
- Thomas Burg (comForte)

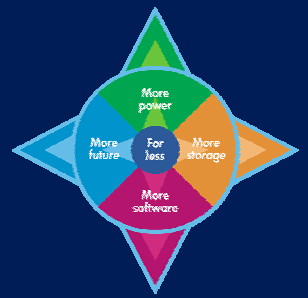
- TCP/IP: Extending the reach of NonStop security requirements
- Are there only "script-kiddies" out there?
- Why a firewall is not enough
- Best practices in network security

Agenda

- Unix® security applied to OSS
  - Are we like Unix?
  - 10 common Unix security holes reviewed
- OSS security features
  - Basic security refresh
  - Safeguard volume ACL applied to OSS
  - OSS to Guardian interoperability
  - Auditing
- Middleware security
  - File sharing security: NFS and SMB
  - SSL enabling middleware (iTP Webserver, Java)
  - Open source and third party products

Are we like Unix? Where to start?

- Read the existing security chapter of each product:
  - OSS System Services Management and Operations Guide, Managing security.
- Analyze existing Unix security checklist and compile the items that may apply to OSS.
  - Exhaustive checklists from http://www.cert.org/ :
    - http://www.cert.org/tech_tips/usc20.html
    - http://www.cert.org/tech_tips/Unix_configuration_guidelines.html
    - http://www.cert.org/tech_tips/intruder_detection_checklist.html



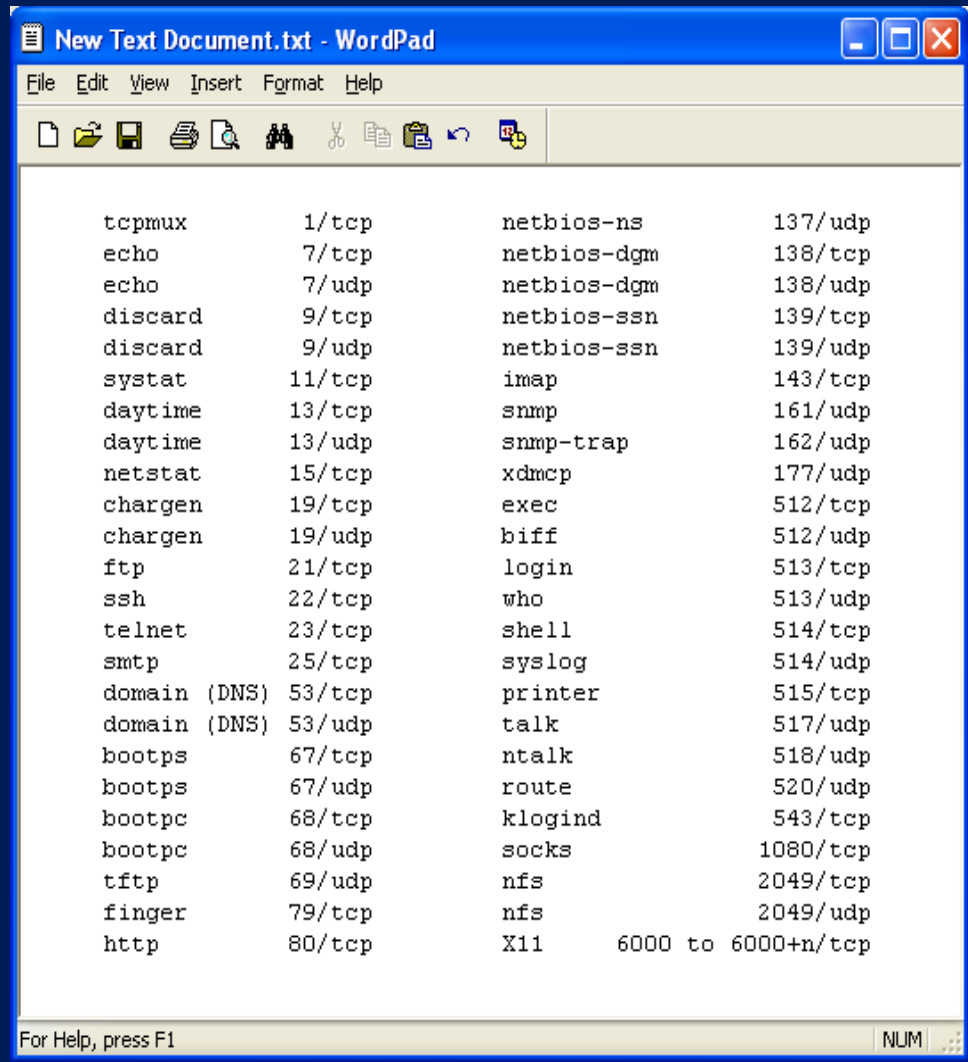
- Read “Practical Unix and Internet Security” book (3rd Edition by Simson Garfinkel, Gene Spafford, Alan Schwartz)

Are we like Unix? Good news…

- Reading those references will show you that OSS is by default immune to a majority of Security issues that are encountered on most Unix systems.
- Why? …Because:
  - OSS is delivered with no network services enabled.
  - Limited number of administration files/commands.
  - OSS is delivered with no system task started (more later).
  - Complete OSS environment is simple and has a reasonable number of files.
- OSS applies one of the first security rules: deny all, at first then allow one by one as needed.

Are we like Unix? Many services?

- Most Unix systems have a multitude of services ready to use (sendmail, NFS, rexec, syslog, snmp,…).
- Securing a complex system means securing each service, which in turn means you need to know how each of them work, when they are used, and by which application.
- This makes security difficult to implement when you're being asked:
"Consider disabling the following services:"
Do you know all of them?

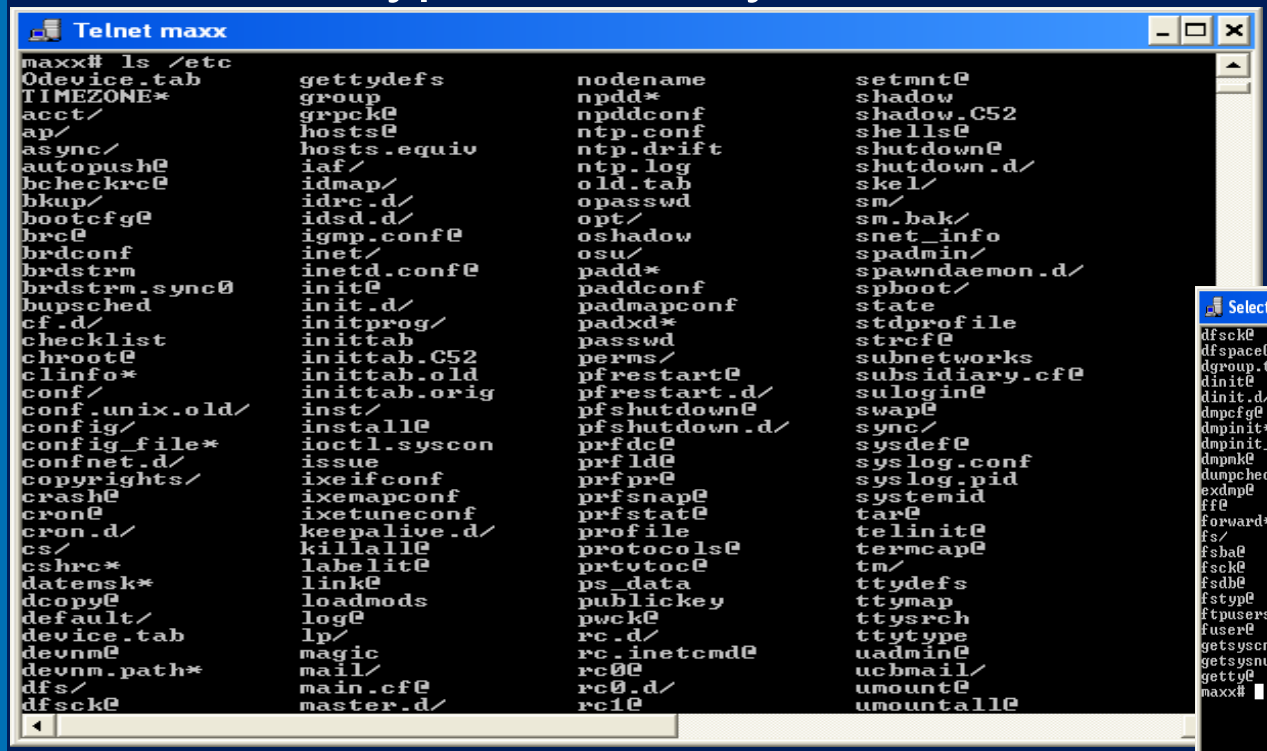


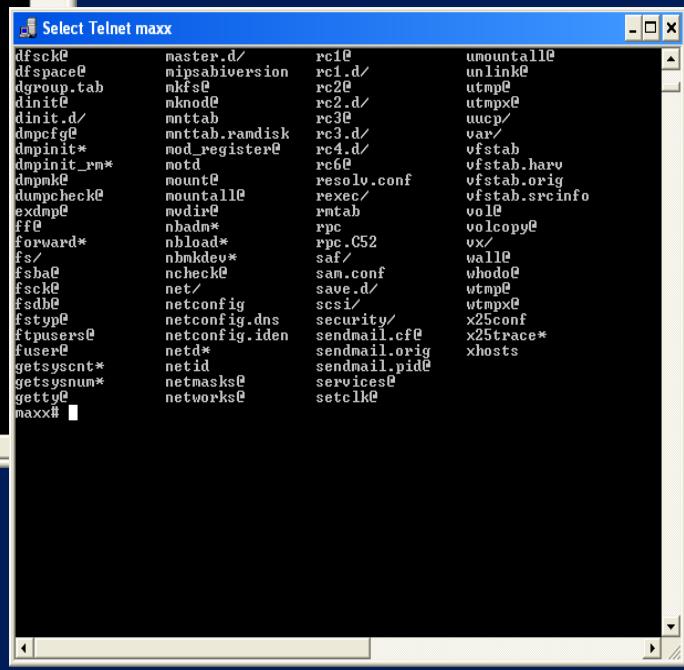
Are we like Unix? Many tools?

- A typical unix system has more admin files/commands:
  - Passwd and group files on Unix (but not OSS) expose user information in a readable format:
    - roland:x:1001:1001:R Lemoine:/home/roland:/bin/bash
    - ckulick:x:1002:1002:Christina Kulick:/home/ckulick:/bin/sh
    - jzimsky:x:1003:1003:John Zimsky:/home/jzimsky:/bin/sh
  - Admin commands like netstat, share, etc… give details which helps potential hackers.
  - Many commands you probably don't know so you can't make any decision on keeping them or removing them.

Are we like Unix? Many files?

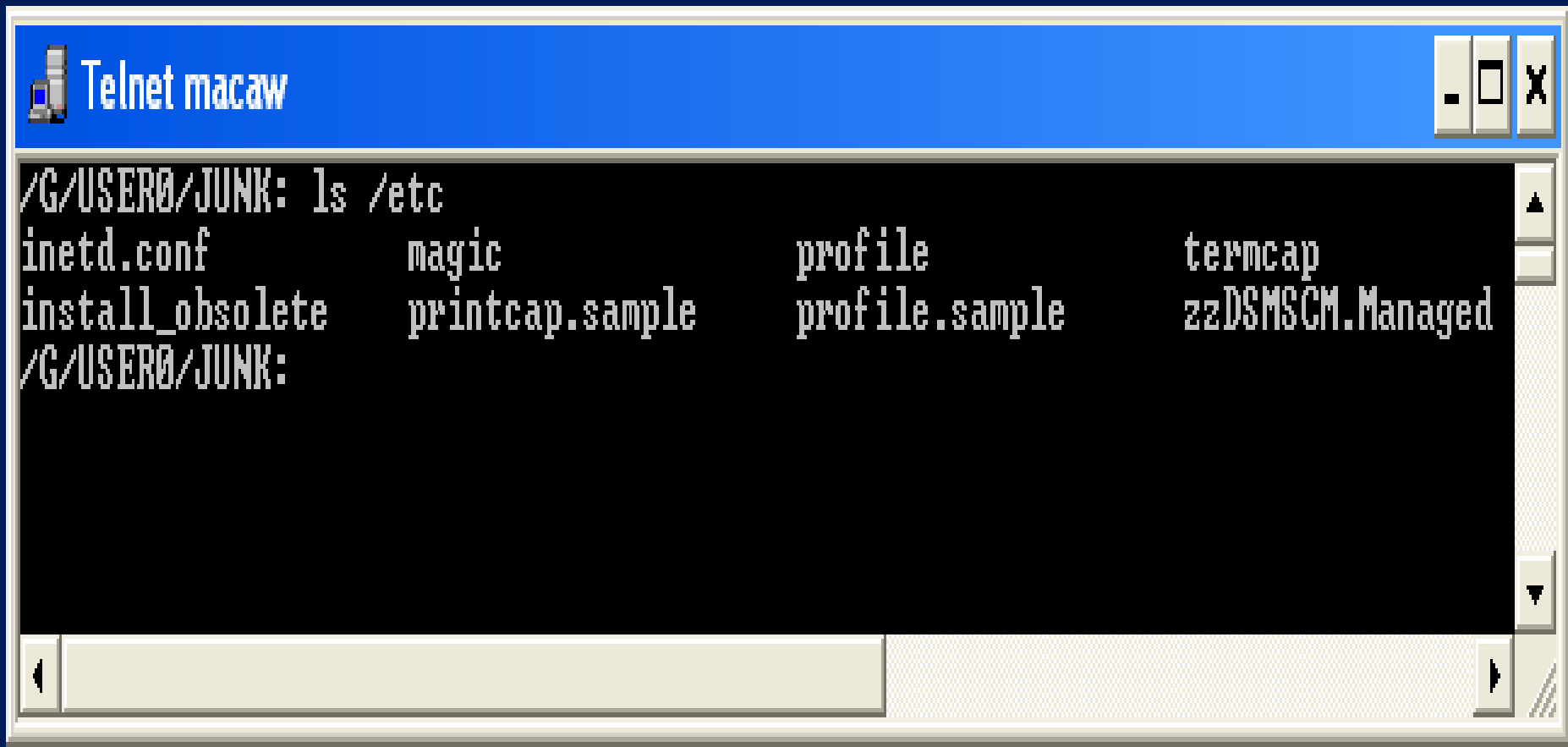
- /etc on a typical Unix system:





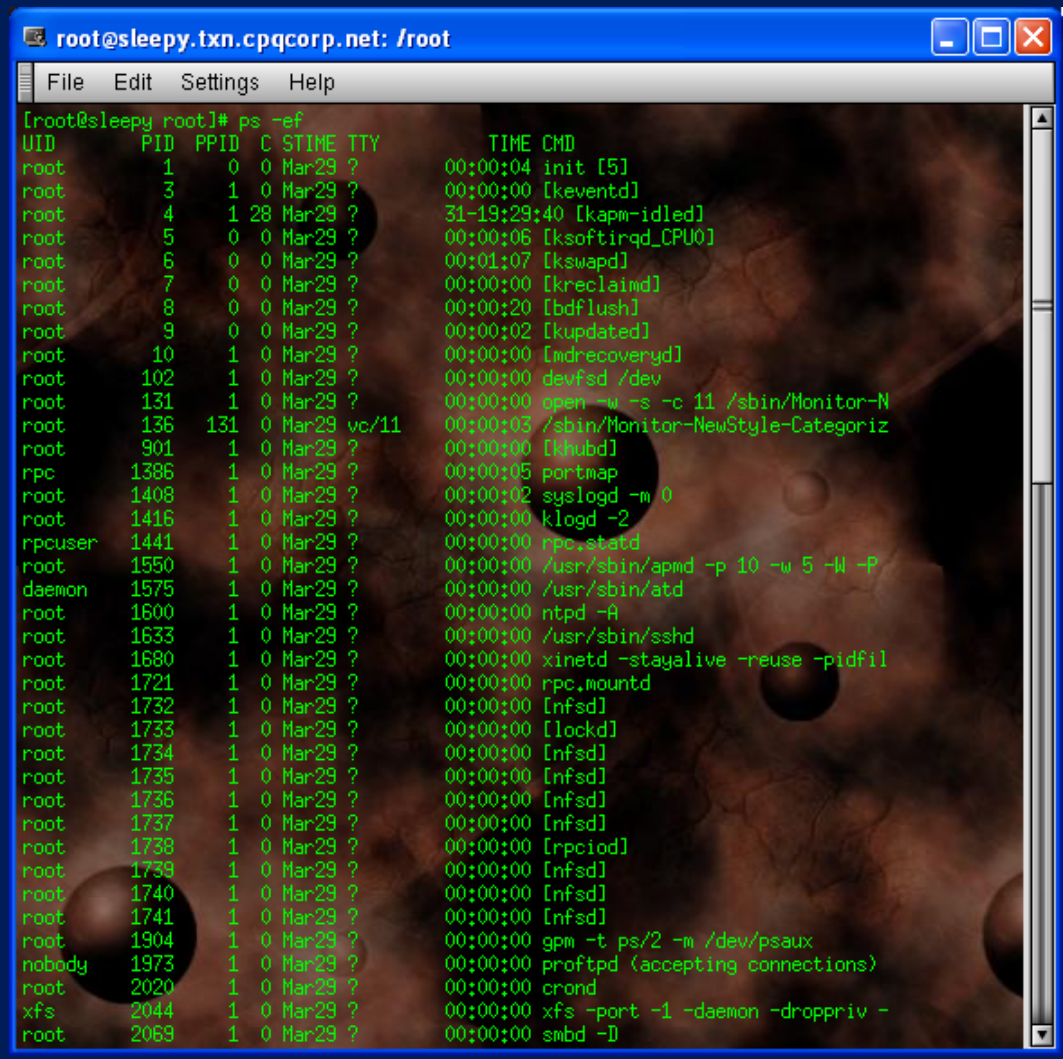
Are we like Unix? Not many files!

- /etc on OSS system:



Are we like Unix? Many processes?

- OSS is delivered with no system task started
- The window on the right shows a typical Unix system process list
- What processes need to be secured?
- Which one can be stopped?



Are we like Unix? Maybe better!

- Simplicity is security's best friend. This makes OSS less vulnerable to common Unix attacks than other Unix platforms!
- This also means that to secure OSS you don't need a complex set of skills but only need to consider those services that you need to configure for your application.
- Doing nothing, your OSS environment is already much more secure than any Unix platform!

11 common Unix security holes.

1. Remote hosts trust (.rhosts and hosts.equiv)
2. Inetd.conf services enabled.
3. SUID shell scripts or commands
4. Cron (Unix job scheduler)
5. File permissions
6. .netrc
7. Check all your system call arguments
8. System calls to avoid
9. File opens and return codes
10. Don't depend on environment variables
11. Current directory in PATH

1. Remote hosts trust

- Commonly used to allow rsh (Remote Shell) to access your server from a client, without the client being prompted for a password.
- You define a user/host allowed to connect:
  icebat.txn.cpqcorp.net jsmith
- This system though, using TCP/IP addressing, is opened to “spoofing”. Hackers just have to place a system with the same IP address on the network and will instantly get access to the system without having to use a password.
- Consider using ssh instead.
- If you still need to use rsh, make sure to apply checklist items applying to .rhosts and hosts.equiv

2. inetd services

- inetd is not started by default on OSS
- If started the following services can be disabled:
  echo, discard, daytime, chargen, time.
  To avoid possible Denial of Service attacks.
- inetd is protected against Denial of Service attacks.
  - -R rate

    Specifies the maximum number of times per minute a service can be invoked. The default value is 40.

3. SUID/SGID commands and scripts

- SUID (means Set User Id) allows a user to execute a program under the identity of another user.
  - NOTE: The set user id (SUID) flag is like NSK progid in that when a process is created from the file, the process runs under the user id of the file's owner.
- Do not allow SUID files other than system ones if possible.

  Hint: you can influence/restrict them via /etc/suid_profile
- Avoid Shell scripts with SUID.
- A malicious user can leave a file with SUID for later access. Solution:
  Create a cron job to look for SUID files
  - find / -WNOE –WNOG -type f –perm -04000 –print
  - Usual ones in OSS are:
    at, atq, atrm, cron, ipcs, newgrp, rsh, expreserve, exrecover
  - If you find another one, consider it suspect.

3. SUID commands and scripts

- More on how to locate SUID programs:
  - find / -W NOG –W NOE –type f –user SUPER.SUPER –perm –04000
    - The parts of this command are as follows:
    - / - search all directories under the root “/”
    - -W NOG – but don’t search /G
    - -W NOE – and don’t search /E
    - -type f – look only for regular files
    - -user SUPER.SUPER – files owned by the super user
    - -perm –04000 – look for files that have the SUID bit set

4. Cron: The Unix job scheduler

- For the cron make sure that:
  - /var/spool/cron/crontabs/SUPER.SUPER is read/write only for the owner.
  - Owner is SUPER.SUPER
  - Consider disallowing cron for regular users:
    - Create /var/adm/cron/cron.allow
    - Add the name you authorize to use cron.
    - Any other name is instantly denied to use cron.
  - ENSURE Super user cron job files do not source any other files not owned by the Super user or which are group or world writable.

5. File permissions

- Ensure those files are Read-Write only for their owner: /etc/inetd.conf, /etc/hosts.equiv, $HOME/.rhosts
- Ensure /etc/profile is owned by SUPER.SUPER and 644 (r/w for the owner, read for the rest).
- Ensure that /, /etc, /usr/etc, /bin, /usr/bin, /sbin, /usr/sbin, /nonnative are owned by root and not writable to ‘other’ and that the sticky-bit is set on /tmp, /var/tmp and on /usr/tmp (and that they are world writable as an exception):
  # chmod 1777 /tmp /var/tmp /usr/tmp
- Use ‘umask’ in /etc/profile to setup a default security for new files

6. .netrc

- Allows user to automate an ftp session without being prompted.
- Very flexible but usually contains password in clear text and therefore very easy target for hackers.
- Fairly save to use if permissions on this file as well as on the users home directory is properly secured.

7. Check your arguments

- Arguments passed to your program on the command line
- Arguments that you pass to system functions
- Do bounds checking on every variable
- Arguments you get from the environment
- Arguments/input you read from a file
- Hackers are taking advantage of unexpected input to a program to make them fail.

8. System calls to avoid

- replace:
  gets(); strcpy(); strcat(); sprintf(); vsprintf();
  scanf(); sscanf();
  with:
  fget(); strncpy(); strncat(); snprintf(); vsnprintf(); bcopy();
  bzero(); memcpy(); memset();
- avoid execlp() and execvp(): they use PATH, so you may not run what you expect...
- NEVER use system() or popen() calls

9. File opens, return codes

- To open a file, which should already exist
  - lstat() the path, check that lstat() succeeded
  - check that it's acceptable (e.g., not a symlink)
  - open() (without O_CREAT), check that the open succeeded
  - fstat() the fd returned by open
  - if the lstat and fstat st_ino and st_dev fields match, accept.
  - If possible use access() instead, it performs a whole lot better
- don't assume a system call will succeed (fork(), etc..)
- make consistency checks - like asserts in C

10. Don't depend on environment variables

- It’s easy for a hacker to alter those and therefore change the behavior of a program.
- Log time, UID, GID, terminal, PID, arguments,hostname with syslog()
  - For tracing capabilities.
- Use full (absolute) pathnames for any filename argument
  - prevents a hacker to substitute a file from a location where he has write capability.

11. Don't put the current directory into your PATH

- Never ever do it for the super user
- Avoid it, if possible, for ‘mere mortal’ users
  - If necessary append it at the end of PATH, never at the beginning.
- Why?
  - There are some publicly writable directories, e.g. /tmp. Consider someone dropping malicious code there under some innocent looking name like ‘ls’, but actually it is a script doing a ‘rm –r /’. Now some user, or in worst case the super user, comes along, ‘cd /tmp’ and performs an ‘ls’… Let’s hope for a recent backup now!

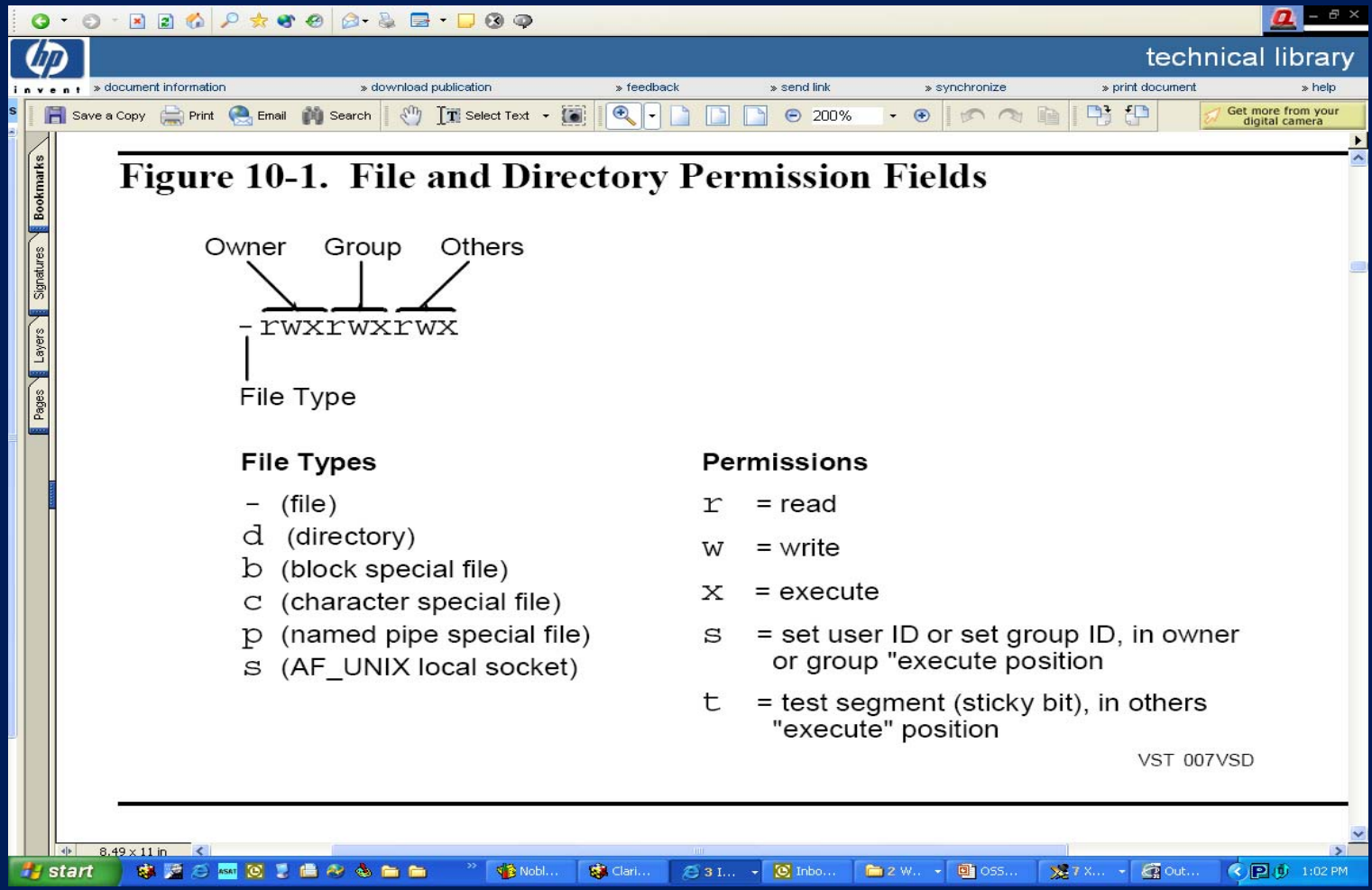
11 common Unix security holes

- This was just a sample list of common security mistakes to avoid.
- Please use the 3 references listed on slide 3:
  - OSS manuals
  - Practical Unix and Internet Security 3rd edition.
  - Internet references like the one provided by the CERT which include exhaustive Unix checklist.

OSS security features

- OSS basic security refresh
- Security inherited from the guardian volume (Limited ACL support).
- Security aspects implied by the interoperability between OSS and Guardian.
- Safeguard auditing to enhance OSS security.

Basic security refresh



OSS security inherited from the volume

- Within OSS a file may appear as writeable by all: i.e. : rw-rw-rw
- But this may not be sufficient as the volumes where the OSS files are really located (the ZYQ sub-volume) may have a security restrictions set.
- It is possible to use some of the Safeguard ACL features applied to OSS:
  - Volume ACLs restricting Create can be applied to an OSS dedicated volume
    - Only Create, not Read, Write, Execute or Purge! And even that has been dropped as of G06.26!
  - ACLs are not supported at the sub-volume (ZYQnnnnn & ZX0nnnnn) level.
  - ACLs are not supported at the OSS file level.

OSS security : Guardian interoperability

- OSS as the POSIX personality to Guardian is providing extremely useful bridging capability between OSS to Guardian.
- ‘gtacl’ allows to start a TACL or a Guardian program from OSS.
- /G provides access to all Guardian files from the OSS environment.
- Pathway servers are accessible though the same APIs available in Guardian
- Those capabilities are important to preserve existing developments but should be known whenever opening access to OSS users.

OSS security : Guardian interoperability

- There are no “OSS users”. Users going into OSS environment are the same as the Guardian ones declared in safeguard and will therefore have the same power as they have in the guardian environment.
- There is no need for a “root” Safeguard alias to run OSS.
- root is the equivalent of SUPER.SUPER in the Unix world, but since most system administration is done on the Guardian side, this “SUPER” user is rarely needed in OSS.
- It’s still good practice to use aliases for accountability
  - For example the different aliases will appear in an audit trail.
  - Aliases may have different passwords.

OSS security : Guardian interoperability

- If root is used, remember to not allow easy remote access (spoofing exposure):
  - Through a .rhosts file ➔ Don't add "root" in it
  - Through OSS NFS ➔ Don't allow root user access
- SUPER.SUPER should only be needed for a limited set of actions:
  - Installing middleware
  - Initial creation and setting of permissions on filesets and top directories.
  - Change ownerships of files or directories
  - Setting SUID/SGID bits

OSS security : Guardian interoperability

- Restricting access to Guardian for OSS users isn't possible. But see Knowledge base solution "How to restrict access for an OSS user". The result will be:
  1. Can logon only to the OSS environment via telnet service;
  2. Can NOT logon to Guardian environment directly;
  3. Can NOT logon to Guardian by means of OSS gtacl command;
  4. Can NOT execute Guardian command from OSS environment by means of OSS gtacl command and the -c option. The -p option still works.
  5. Have access to Guardian file space (/G directory)
  6. Have FTP access to Guardian file space ("quote guardian" FTP command)

OSS security : Guardian interoperability

- ‘gtacl’ access can be limited further by changing permission from 755 (owner, group and the whole world can execute) to 700 (only owner (SUPER.SUPER) can execute gtacl.
- /G can be accessed by OSS users but as soon as the user access a file in /G, then Guardian/Safeguard security takes over.
  - i.e.: /G/system/sys00/tacl is “NUNU”
- Guardian file security settings are not visible from OSS:
  - ‘ls’ will show only “emulated” OSS security settings and has no notion of Safeguard ACLs at all.

OSS security feature: Auditing

- Safeguard auditing is the process of logging to an audit trail a configurable selection of events such as open(), unlink(), chown(), mkdir(), setuid(), fork(), etc…
  (complete list in the OSS programmer's guide).

- Examples:
  - It allows you to go back and check when a file has disappeared and who deleted it.
  - It allows you to see that a new file has been created
  - It allows you to see that someone issued a kill command
  - Etc…

- Audit trails can be used to discover resource usage.

OSS security feature: Auditing

- OSS
  - Specify those existing filesets to be audited
    - SCF ALTER FILESET $ZPMON.HOME, AUDITENABLED ON
    - Only the super ID (255,255) can change the AUDITENABLED attribute
- Safeguard
  - AUDIT-CLIENT-SERVICE must be enabled (default)
  - Alter default audit pool configuration to accommodate increase in audit activity due to OSS security auditing
  - Enable AUDIT-PROCESS-ACCESS-PASS and/or AUDIT-PROCESS-ACCESS-FAIL for:
    - Auditing of process creation calls by exec(), fork(), and so forth
    - Auditing of calls to setuid(), setgid(), setsid(), setpgid(), and kill

OSS Auditing in a few steps

1. in safecom
   alter safeguard, audit-client-service on
   alter safeguard, audit-process-access-pass all
   alter safeguard, audit-process-access-fail all
2. in SCF, assume process $ZPMON
   alter fileset <fileset>,auditenabled on
   Stop and start fileset.
   Starting with G06.24 this can be done without stopping the fileset!
3. In OSS: touch /<fileset>/newfile
4. in SAFECOM
   info audit service (get the name of the current audit file).
5. in SAFEART
   audit file <audit file name>
   set destination file $USER0.ROLAND.MYAUDIT
   set where objectname = "/<fileset>/newfile“
   start

OSS security feature: Auditing

- Output produced extracts:

```
Operation   =Create        Outcome     =Denied
ObjectType =OssDiskfile   Veracity      =Tr
OwnerUsername              =OSC.ROLAND
OwnerUsernumber             =121,240
....
ObjectName
=/crash/newfile=$OSS4.ZYQ00002.Z00009PS:866
2211
```

File sharing security: NFS and SMB

- NFS and SMB are 2 protocols allowing file sharing and therefore frequently used when a server needs to share files with workstations.
- NFS is predominant in the Unix world as any Unix server or client will implement the protocol.
- On OSS, the product OSS NFS allows server capability so OSS files can be shared with NFS clients.
- PC can install NFS client software to behave like a Unix workstation.

File sharing security: NFS and SMB

- NFS has one main security issue:
  Unix clients are granted access based on the User Id they present at authentication time.
- No password authentication is performed!
- It is therefore possible to simulate a user by adding the user on any Unix client and therefore get access to the files of this user on the server.
- Check the security features in the Security section of the NTL manual OSS NFS Management and Operations manual.

File sharing security: OSS NFS checklist.

- Don’t allow root (SUPER.SUPER) access
  Hint: set ROOT USER OK to FALSE
- Export only selected filesets.
- Use ADDR-CHECK (on the LAN object) to enforce client to have a matching hostname/IP address.
- PC clients go through a real password authentication using PCNFSD.
- Unix workstation may not allow mount operations without root access.
- Use NETGROUP feature to allow only specific clients hostnames to access OSS NFS.

File sharing security: Samba

- SMB (Server Message Block) is native to Microsoft Windows.
- Samba is a famous freeware which emulates SMB protocol on the server side.
- A Samba port is available on OSS (not supported) which therefore allows PCs to access OSS files transparently from the windows environment as if this was an additional PC drive.
- No need to install any software on the PC to access the OSS resources.

File sharing security: Samba

- A real user authentication is happening (not UID based).
- A great integration: If your user/password on the PC matches the user/password in Samba then you can access the files without having to enter your PC password again.
- UID or host spoofing won't work since a password is needed.
- Passwords are stored encrypted within samba.

SSL : Terminology and concepts

- **Asymmetric encryption:** 2 different keys (one public/one private), i.e. RSA (512/1024 bit keys).
- **Symmetric encryption:**1 key known by both sides only. i.e. DES, RC2, RC4. (40/128 bit keys).
- **Public (encryption) key:** Everybody can encrypt messages with a "public key" BUT only the server who distributes it can decrypt the messages that were encrypted with it. So it allows you to make a private connection to the server (nobody else can read your message).
- **Private (decryption) key**: is owned by the server to decrypt messages encrypted with the public key.

SSL : Terminology and concepts

- **Digital signature:** allows your recipient to check that the message he receives has not been altered by anybody.
- **Private (encryption) key:** used to encrypt a hashed message (message digest) and add it to digitally signed message.
- **Public (decryption) key:** it is used to decrypt "encrypted message digest".

SSL : Combining technologies

- symmetric and asymmetric, which one to use?
- Reality is, **both need to be used** to achieve a safe encryption capability.
- Asymmetric encryption is used for the key management.
- Symmetric encryption used for the remaining data exchange.

SSL : Combining technologies

- No man in the middle attack ➔ certificate will bind server data to the associated public key.
- No forgery of the public key ➔ The whole certificate will use a digital signature to avoid the contents to be altered.
- The certificate signature is encrypted using a private key from the Certificate Authority.
- Using the CA's public key embedded in the client, the client makes sure the contents of the message have not been altered.

iTP Webserver

- Don’t run iTP Webserver as SUPER.SUPER (root). A user in the SUPER group is recommended.
- Use ITP Secure Webserver to encrypt data using SSL.
- Use Region directive to place a password to access html files:
  RequirePassword realm {-userfile userfile | -safeguard}
  - And it is a good idea to use this feature in association with SSL so that the password circulates encrypted on the wire.
- Use the Region AllowHost/DenyHost directives.
- Use the Deny directive
- Enable the AccessLog to see who is connecting.
- Consider disabling Directory browsing
- Disable the TRACE method (iTP Webserver ABS and up)

SSL-Enabling iTP Webserver

- Configure the SSL transport: httpd.stl.config:

```
AcceptSecureTransport -transport /G/ZB018 -port 443 -address
16.74.49.30 -cert {CN=Secure Transport Bootstrap Certificate,
OU=Testing Only - Do Not Trust for Secure Transactions, OU=No
Assurance - Self-Signed, OU=Generated Tue Dec 30 15:13:33 CST 2003,
O=bbq}
```

- Enforce a directory to be secured:

```
Region /*/ssl-sample-dir {
    RequireSecureTransport
}
```

- Access the page using https:

```
https://<host>:<port>/samples/ssl-sample-dir
```

- And you get….

SSL-Enabling iTP Webserver

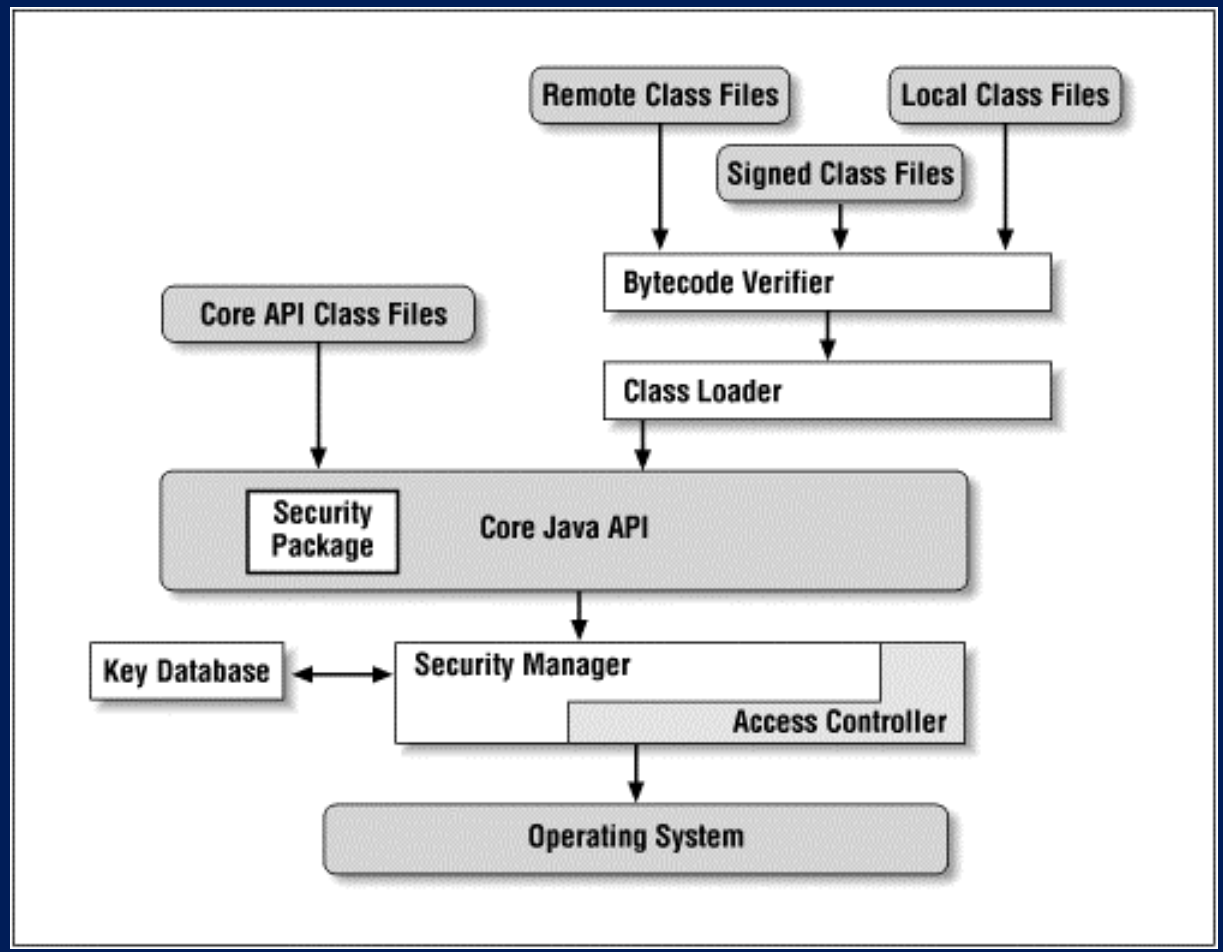
SSL-Enabling iTP Webserver

- To use a real certificate:
  1. Generate a key pair:
     ./keyadmin -keydb database -mkpair -dn 'CN=compaq.us.tandem.com,OU=work,O=cpq,L=austin,ST=texas,C=US' -length 1024
  2. Generate a CSR (Certificate Signing Request).
     ./keyadmin -keydb database -mkreq <certificate request filename> -dn 'CN=compaq.us.tandem.com,OU=work,O=cpq,L=austin,ST=texas,C=US‘
  3. Obtain a certificate for the public key part of the pair from a Certificate Authority (CA) by e-mailing the certificate-request file to the CA.
  4. Add the certificate to the Key database
     bin/keyadmin -keydb conf/mykeys -addcert my-cert.txt

Java: A Language ready for security

- Java was written from the ground up with security in mind.
- Language design: bounds checking on arrays, legal type conversion only, no pointer arithmetic, ...)
- Access control (Using the security manager): file access, network access, resource access, etc...
- code signing: cryptographic authentication.

Java security: Architecture



Java security: Bytecode Verifier

- Check that the bytecode has not been altered after compilation
- Illegal data conversion
- Private data access
- run time overflow, etc…
- All done before the code can execute

Java security: ClassLoader

- Provide the infrastructure so that only correct classes are loaded.
- Untrusted classes are restricted from executing dangerous instructions
- Untrusted classes cannot access protected system resources
- You can implement your own class loader to make specific checks when loading a class. For example: accept only encrypted classes loading, check if code is run by authorized users. The class loader also check that system classes are not replaced by user classes.

Java security: Security manager

- The java security manager: checks for example if current thread can:
  - create a new class loader
  - create a sub process
  - halt a JVM
  - load a DLL
  - read/write/delete a file
  - Connect/accept a socket connection,.
- The access Controller is used by Security Manager to allow or prevent access from the API to the OS.

Java security: Security Manager

- If no security manager is used then all privileges are granted.
- To add it:

  java -Djava.security.manager
  or
  System.setSecurityManager(new SecurityManager());

  (uses default policy in $JAVA_HOME/jre/lib/security/java.policy)
- Dangerous access is allowed by permissions mapped to an identity which can be a “CodeSource” or a “Signer”.

Java security: Security Manager

- Default policy JAVA_HOME/jre/lib/security/java.policy
- grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;};
  classes from the ext directory to be executed.
- grant codeBase "file:/home/roland/*" {
  permission java.security.AllPermission;};
  classes from /home/roland to be executed
- grant {
  permission java.net.SocketPermission "localhost:1024-",
  "listen";};
  Post a listen on a socket as long as the port number is higher than 1024

Java security: Security Manager

- To grant permissions based on a signer:
- Generate a key pair:
  keytool -genkey -alias signFiles -keypass kpi135 -keystore susanstore -storepass ab987c
- Sign the code (delivered in a jar file)
  jarsigner -keystore susanstore -signedjar sCount.jar Count.jar signFiles
- Publish the public key:
  keytool -export -keystore susanstore -alias signFiles -file SusanJones.cer
- Install the certificate on the user side:
  keytool -import -alias susan -file SusanJones.cer -keystore raystore
- Grant Permissions:
  grant SignedBy “susan" { permission …

Java security: JAAS

- So far we granted privileges based on who provided the code, but Java also has an API to allow execution based on who is running the code.
- Java Authentication and Authorization Service
- for *authentication* of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet; and
- for *authorization* of users to ensure they have the access control rights (permissions) required to do the actions performed.

Java security: JSSE

- So we have secured who is providing the code and who is running it, one last thing… How to secure the data on the "Java wire":

  ➔ Java Secure Socket Extension

- A java implementation of the SSL protocol
- As for all Security products, for global deployment make sure to check U.S. Exports restrictions which may apply for some countries.

More middleware security…

- NS/Corba 2.6 provides encryption of the IIOP wire using SSL (Openssl).
- NS/Tuxedo provides LLE (Link Level Encryption).
- BEA WebLogic Server:
  - Full SSL implementation.
  - All Java security features
  - Server SSL ready without coding.
  - Security features available in the IDE (workshop).

Open Source security products

- The very famous openssl and openssh are available on NonStop:
  - openssl-0.9.7a: Secure Sockets Layer and Transport Layer Security
  - openssh-3.7.1p2: Port of OpenBSD's excellent OpenSSH
- sudo-1.6.7p5: Give limited root privileges to users and log activity

Third party security products

- If you are a partner and want to be added to this list, contact Ron.LaPedis@hp.com
- Baker Street Software
- Bowden Systems
- CAIL
- comForte
- Cross-EL
- Crystal Point
- CSP
- GreenHouse
- Gresham Software Labs
- Insession Technologies
- K2Defender
- Unlimited Software Associates
- XYPRO

NonStop delivers