Global Information Assurance Certification Paper

Fundamentals for Securing OpenVMS Systems

Mario Babineau

GSEC – Securing OpenVMS; Fundamentals. V1.4b

Table of Contents

Executive Summary

The purpose of this document is to define the fundamentals of securing an OpenVMS system. It defines the OpenVMS security model and how it can be applied to analyze and define security measures required for an OpenVMS system.

OpenVMS is based on a very different model than Unix, thus the knowledge base required to attack or secure an OpenVMS system is very different than the one used for Unix. Most Unix based attacks will fail on an OpenVMS system.

OpenVMS security management also uses a different approach than Unix. This document will show how security management is done in OpenVMS.

Securing an OpenVMS is not like securing a Unix system. It requires a different "mind set". It is like learning a new language; trying to translate every word from one to the other will in most cases make the sentence completely incomprehensible, and some words just do not exist in the other language. Direct translation of security practices from Unix to OpenVMS is not practical, nor will it give the expected result.

Understanding Computer Security

The first step in computer security is to understand how the Operating System (OS) works. Read the vendor internal and management manuals. The security administrator should be familiar with the command structure used by this OS. The second requirement is to know in a structured way where the computer compromises could come from. Then a model can be developed on which security rules can be based.

Computer systems are usually compromised in one of these four ways:

- **User[1] irresponsibility**: This is where a user purposely or accidentally leaves open access to the system. This can be done by leaving a session logged on and leaving the terminal unlocked. This could also be a user who writes down his password in an easily accessible location.
- **User probing**: This is a user trying to find unprotected access to a system restricted area. Some users view this as an intellectual game between them and the security administrator.
- **User penetration**: This is when a user actually breaks through the security barrier to gain access to the system restricted area.

[1] A user is someone who has authorized or unauthorized access to a system

- **Social Engineering**: This is when someone accesses system resources by non-technical means. This is usually done by coercing a user to give information regarding the system security or facility security. This could also be done by convincing the user to actually perform commands on the system to give them access.

The second step is to define the security level required. The following factors must be considered when defining the security level:

1. **Cost**: The higher the security level the more expensive it will be to maintain security on the system.
2. **Accessibility:** The higher the security level the more difficult it will be to access the system. Too much security may render a system unusable.
3. **Risk**: What are the risks of intrusion on this system? No access to Internet and only one account would be a very low risk system.
4. **Asset**: What is the asset that needs protecting? How much monetary value does it have for the company? How much interest does it have for others?

Security needs will be defined in one of the following three levels:

1. **LOW**: Straight out of the box configuration is good enough.
2. **MEDIUM**: Security tools built in the OS with some configuration is acceptable
3. **HIGH:** Third party software is required to secure this system.

OpenVMS Security Model

"OpenVMS and UNIX use rather different security models, and as a result, the two platforms tend to require rather different security attacks and tools. OpenVMS and UNIX also have rather different management "mind-sets" " [1]

In the late 1960s, as multi-user systems were emerging, a lot of research was done on how to secure such a system.

The first option was to find every security vulnerability on a system and block them one by one. This proved to be an impossible task, since the blocks themselves hold some security vulnerability.

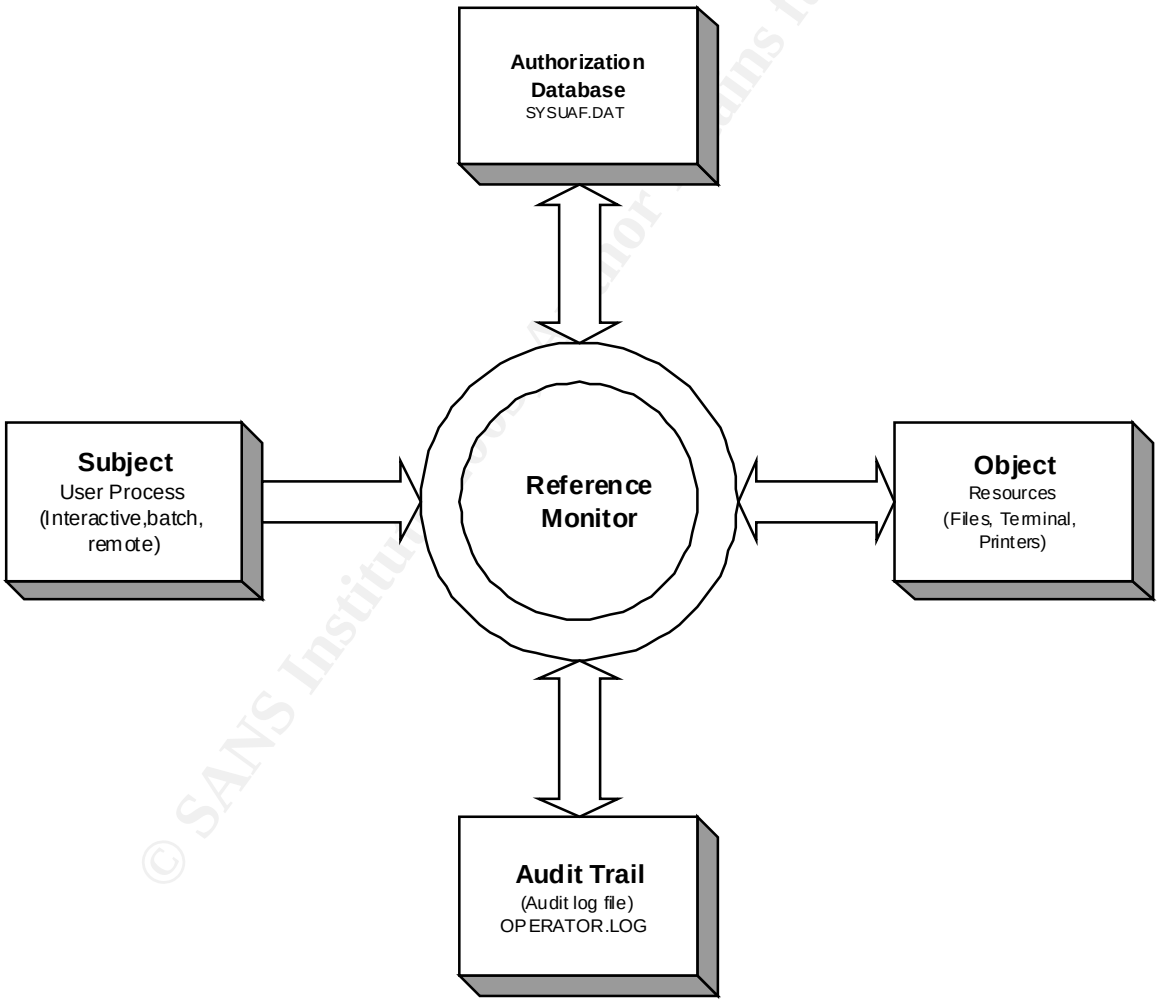
The second option came as the Reference Monitor Concept. This concept was first introduced in 1972 in the *"Anderson report" [2].* a computer system is defined into four parts: a **subject**, an **object**, an **authorization database**, and an **audit trail**. The Reference Monitor process is the controller of the flow among all of these parts. The Reference Monitor model is only an abstraction; it does not

define security rules but the model on which to base security rules. For the model to be perfect the following criteria must be met:

- Every access to objects MUST be mediated.
- The Authorization database and the audit trail MUST be tamperproof.
- The code that performs these tasks MUST be small and easy to analyze.

The concept is that any subject that wants to access an object MUST be authorized by the authorization database and a record of the access (granted or denied) MUST be recorded in the audit trail. The authorization database must be dynamic so that it can adapt to new subjects or objects created on the system.

Figure 1: Reference Monitor



Computer systems that conform to this concept are very secure. OpenVMS does not conform to this model, but the user interface and system management processes do mirror the basic processes of the Reference Monitor model.

Now let's look at how these four parts are implemented in OpenVMS and how we can secure these with some simple configuration changes.

Reference Monitor SUBJECT Security

In the OpenVMS world, the subject could be any of the following:

- Interactive process
- Batch job process
- Remote/network process

All of these, except batch jobs, require authentication to the system. This authentication is done via the LOGINLOGOUT process. Every action done on an OpenVMS system MUST be done under a user identification, thus a batch job cannot be submitted unless the user is already authenticated. The LOGIN LOGOUT process creates a user process which defines the user rights on the system. The user rights are stored in the system **User Authorization File** (UAF) SYSUAF.DAT. The SYSUAF.DAT file is controlled and updated via the Authorize utility. This utility can only be run by users that are part of the system group.

Most access to the system requires a password as an authentication mechanism. OpenVMS offers multiple ways to protect and enforce password policy. The password policy and the password itself are also stored in the UAF. OpenVMS offers the following features for password security:

- Primary and Secondary password
- Auto generate password
- Password expiration
- Password dictionary
- Password history
- Password length
- One-way encryption storing
- System password for terminal access
- Template account
- Access limiting descriptors

Primary and Secondary Passwords

Passwords are NOT case sensitive. Actually nothing is case sensitive in OpenVMS; everything typed in lower case is automatically converted to upper case by OpenVMS.

Password restrictions in OpenVMS are:

1. Passwords can be from 1 – 31 characters
2. Valid characters are: A-Z, 0-9, $ (dollar sign), and _ (underscore)

To set a primary password with the Authority Utility:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY  SANS[2]  /PASSWORD=THISISLIFE
```

The user can modify his/her password via:

```
$ SET PASSWORD
      or
$ SET PASSWORD/GENERATE=n
```

The secondary password is where a user enters a username/password and the system asks for a secondary password before creating the user process. A primary password is mandatory prior to setting the secondary password.

To set primary and secondary passwords with the Authority Utility:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY  SANS /PASSWORD=(THISISLIFE,THISISLIFE2)
```
(in this case the primary password is: THISISLIFE, and the secondary password is: THISISLIFE2)

The user can modify his/her secondary password via:

```
$ SET PASSWORD/SECONDARY
      or
$ SET PASSWORD/SECONDARY/GENERATE=n
```

Auto generate password

OpenVMS can generate a password. The auto generate password utility only requires one parameter: the number of characters the password should have. The first column is the actual proposed password and the second column is the pronunciation for the password.

```
$ SET PASSWORD/GENERATE=8
Old password:
afneapyva     af-nea-py-va
calfjoyn      calf-joyn
alotansha     a-lo-tan-sha
vadfoarwoa    vad-foar-woa
todaskma      to-dask-ma
```

[2] All examples in this paper will use a user account call SANS

```
Choose a password from this list, or press RETURN to get a new list
New password:
```

This policy can be enforced on a user basis via the Authorize utility.

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY SANS /GENERATE_PASSWORD [=keyword]

     BOTH          Generate primary and secondary passwords.
     CURRENT       Do whatever the DEFAULT account does. This could mean
                   to generate primary, secondary, both, or no
                   passwords.
                   This is the default keyword.
     PRIMARY       Generate primary password only.
     SECONDARY     Generate secondary password only.
```

Password expiration

The default expiration period for an account is 90 days. When the account gets created the password is pre-expired; the user must enter a new password on the first successful login.

To set the expiration to 90 days, 30 hours, 15 minutes, and 7 seconds the syntax is:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY SANS /PWDLIFETIME=90-30:15:07
```

Passwords can be set to expire immediately as follows:

```
UAF> MODIFY SANS /PWDEXPIRED
```

Password Dictionary:

By default the SET PASSWORD command compares the new password against the standard OpenVMS dictionary. OpenVMS will modify the case as necessary to find a match in the dictionary file. The security administrator can create an unacceptable password list and add it to the standard dictionary file as follows:

```
$ CREATE LOCAL_PASSWORD_DICTIONARY.DATA
StarWar
Skywalker
nameofcompany
usernames
Crtl/Z

$ SET PROCESS/PRIV=SYSPRIV
$ CONVERT/MERGE/PAD LOCAL_PASSWORD_DICTIONARY -
_$ SYS$LIBRARY:VMS$PASSWORD_DICTIONARY.DATA
```

The dictionary check is not performed if the Authorize utility is used to set a user password.

Password History

By default the password history is ON and keeps 60 names for 365 days. This can be changed by modifying the following logical names:

SYS$PASSWORD_HISTORY_LIFETIME (value=1 to 28000 days)
SYS$PASSWORD_HISTORY_LIMIT (value=1 to 2000 names)

```
$ DEFINE/SYSTEM/EXEC SYS$PASSWORD_HISTORY_LIMIT 100
```

The history check is not performed if the Authorize utility is used to set a user password.

Password Length

The minimum length of the password is defined on the UAF via the following command:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY  SANS /PWDMINIMUM=n
```

The maximum length is fixed to 31 characters.

One-way encryption

Every password gets encrypted and only the encrypted password is stored on the UAF. OpenVMS does provide a default encryption algorithm but a user/programmer can create a site-specific algorithm for added security. The new algorithm is enabled via the Authorize utilities.

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZED
UAF> MODIFY  SANS /ALGORITHM=PRIMARY=CUSTOMER=128
```

The *OpenVMS Programming Concepts manual* provides directions on how to write and use customer algorithms.

System Password

A system password is a password that is not attached to an account but to a terminal. This terminal can be a Local Area Terminal (LAT) or remote terminal (created via a telnet session), or the console terminal. This password forces the

user to enter this password prior to being able to use this terminal to connect to the system. The user will still be asked to provide a username/password to access the system.

This is a two-step process; first set the system password and then attach this password to a terminal.

First set the password:

```
$ RUN AUTHORIZE
UAF> MODIFY /SYSTEM_PASSWORD=<password>
      or
$ SET PASSORD/SYSTEM
```

Then attach this password to a terminal:

```
$ SET TERMINAL/SYSPWD/PERMANENT TTY1
```

Template account

OpenVMS has a special account called DEFAULT. This account is the account that gets used to create the other accounts. Any modification to this account will be transferred to all new accounts created after the change was made. Any account that was previously created MUST be modified one by one with the new setting.

```
UAF> show default
Username: DEFAULT                           Owner:
Account:                                    UIC:    [200,200]
([DEFAULT])
CLI:      DCL                               Tables: DCLTABLES
Default:  SYS$SYSDEVICE:[USER]
LGICMD:   LOGIN
Flags:  DisUser
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                 Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:      (pre-expired)
Last Login:            (none) (interactive),           (none) (non-
interactive)
Maxjobs:       0  Fillm:       100  Bytlm:         8192
Maxacctjobs:   0  Shrfillm:     0  Pbytlm:           0
Maxdetach:     0  BIOlm:        18  JTquota:     100000
Prclm:         2  DIOlm:        18  WSdef:          150
Prio:          4  ASTlm:        24  WSquo:          256
Queprio:       0  TQElm:        10  WSextent:       512
CPU:        (none)  Enqlm:      100  Pgflquo:      10240
Authorized Privileges:
  TMPMBX NETMBX
Default Privileges:
  TMPMBX NETMBX
```

Access Limiting Descriptors

The following UAF parameters will limit the access of a user to the system. These are set via the Authorize utility.

/[NO]ACCESS = range	Allow access to system in specified range
/[NO]BATCH = range	Allow batch access in specified range
/[NO]DIALUP = range	Allow interactive dialup access in specified range
/[NO]INTERACTIVE = range	Allow interactive access in specified range
/[NO]LOCAL = range	Allow interactive access via local terminal in specified range
/[NO]NETWORK = range	Allow remote batch access in specified range
/[NO]REMOTE = range	Allow remote interactive access in specified range

range= `([PRIMARY], [hr - hr], [SECONDARY], [hr - hr])`

Reference Monitor OBJECT Security

An Object in OpenVMS can be any of the following:

- File
- Batch queue
- Printer, disk, tape
- Terminal
- Any device locally or remotely connected to the system

OpenVMS has two security control mechanisms: User Identification Code (UIC) base and Access Control List (ACL) base. OpenVMS also has security privileges that give a user the ability to bypass all of OpenVMS's security, a description of these parameters will be given in this section.

UIC Base Security

When an account gets created the System Administrator also gives it a unique UIC. The UIC has two numbers, a Group number (from 1 to 37776 octal), and a member identifier (0 to 177776 octal). Multiple users can belong to the same group, but every user will have a unique member identifier within that group. A user cannot belong to multiple groups in OpenVMS. This identifier [group,member] is stored in the UAF.

When an object gets created, the owner is the account that created the object. Only the owner has control access when using the UIC base security. The owner

can set four securities groups on the object: System access, Owner access, Group access, and World access. Each group has six types of access levels that can be set:

1. **R**ead, Gives the right to read the data from the object.
2. **W**rite, Gives the right to write data to the object.
3. **E**xecute, Gives the right to execute binary object.
4. **D**elete, Gives the right to delete an object.
5. **P**hysical Gives the right to perform physical I/O.
6. **L**ogical Gives the right to perform logical I/O.

Depending on the type of object being looked at, (device, file, etc.) some of the types may not apply. The object protection can be displayed with the following command:

```
$ DIR /FULL (OR /SECURITY) MYFILE.TEXT

MYFILE.TEXT;1                File ID:  (6422,1,0)
Size:            1/4         Owner:    [100,100]
Created:   22-DEC-2002 13:55:40.29
Revised:   22-DEC-2002 13:55:40.40 (1)
Expires:   <None specified>
Backup:    <No backup recorded>
Effective: <None specified>
Recording: <None specified>
File organization:  Sequential
Shelved state:      Online
Caching attribute:  Writethrough
File attributes:    Allocation: 4, Extend: 0, Global buffer count: 0
                    No version limit
Record format:      Variable length, maximum 0 bytes, longest 19 bytes
Record attributes:  Carriage return carriage control
RMS attributes:     None
Journaling enabled: None
File protection:    System:RWED, Owner:RWED, Group:RE, World:
Access Cntrl List:  None
Client attributes:  None


$ SHOW DEVICE/FULL DKA0:

Disk AXPVMS$DKA0:, device type RZ29B, is online, file-oriented device,
    shareable, available to cluster, error logging is enabled.

    Error count                    0    Operations completed                0
    Owner process                 ""    Owner UIC                      [SYSTEM]
    Owner process ID        00000000    Dev Prot            S:RWPL,O:RWPL,G:R,W
    Reference count                0    Default buffer size                512
```

The protection can be set with the following command:

```
$ SET FILE/PROTECTION=(S:RWED,O:RWED,G:RWED,W:RWED) MYFILE.TEXT
$ DIR/SECURITY MYFILES.TEXT

MYFILE.TEXT;1         [100,100]                    (RWED,RWED,RWED,RWED)
```

By default, files get their protection from the parent directory and devices get their protection from the device template.

ACL Base Security

ACL is a more granular way to control access to objects. ACL are built with multiple or single Access Control Entry (ACE). ACE can be based on UIC or an identifier.

An identifier gets created with the Authorization utility, and then users are granted the identifier. A user may have multiple identifiers granted. Access is granted to the identifier via the ACE.

ACL/ACEs are stored in the object header. To use ACL the user must have UIC Read access to an object.

Prior to granting access to an object the following checks are performed:

- UIC access is checked; if there is no ACL on the object, access is granted as per the UIC access level.
- If the Object has ACL and UIC Read access is granted, then the ACEs are checked. Access is granted as per the first matching ACE in the list.
- If the Object has ACL and UIC Read access is granted, and NO matches are found in the ACL, then the UIC access level is granted.

Because of the access rules, the order in which ACEs are entered on an object is very important.

To add ACE "Control access" to the object is required. Access levels given via ACE are as follows; Read, Write, Execute, Delete, Control, None.

The following example creates an identifier call "GIAC" and then grants the user SANS this identifier.

```
UAF> ADD /IDENTIFIER/ATTRIBUTES=(RESOURCE) /VALUE=IDENTIFIER:%X80011 GIAC
%UAF-I-RDBADDMSG, identifier GIAC value %X80080011 added to rights database

UAF> GRANT/IDENTIFIER GIAC SANS
%UAF-I-GRANTMSG, identifier GIAC granted to SANS

UAF> SHOW SANS

Username: SANS                          Owner:   SANS User
Account:  USERS                         UIC:    [100,100] ([USERS,SANS])
CLI:      DCL                           Tables: DCLTABLES
Default:  [USER]
LGICMD:
Flags:
Primary days:   Mon Tue Wed Thu Fri
```

```
Secondary days:                     Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:  22-DEC-2002 14:36
Last Login:            (none) (interactive),            (none) (non-
interactive)
Maxjobs:         0  Fillm:       100  Bytlm:        64000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:       150  JTquota:       4096
Prclm:           8  DIOlm:       150  WSdef:         2000
Prio:            4  ASTlm:       250  WSquo:         4000
Queprio:         4  TQElm:        10  WSextent:     16384
CPU:        (none)  Enqlm:      2000  Pgflquo:      50000
Authorized Privileges:
  NETMBX       TMPMBX
Default Privileges:
  NETMBX       TMPMBX
Identifier                      Value           Attributes
  GIAC                          %X80080011
```

Now we add the ACE GIAC to the file "MYFILE.TEXT" which will give Read, Write, Execute access to any user who has this identifier

```
$ SET SECURITY /ACL=( -
_$ (IDENTIFIER=GIAC,ACCESS=READ+WRITE+EXECUTE), -
_$ (IDENTIFIER=*,ACCESS=NONE)) MYFILE.TEXT

$ DIR/FULL MYFILE.TEXT

Directory DKA600:[SANS]

MYFILE.TEXT;1                File ID:  (6422,1,0)
Size:            1/4         Owner:    [100,100]
Created:   22-DEC-2002 13:55:40.29
Revised:   22-DEC-2002 14:56:52.26 (3)
Expires:   <None specified>
Backup:    <No backup recorded>
Effective: <None specified>
Recording: <None specified>
File organization:  Sequential
Shelved state:      Online
Caching attribute:  Writethrough
File attributes:    Allocation: 4, Extend: 0, Global buffer count: 0
                    No version limit
Record format:      Variable length, maximum 0 bytes, longest 19 bytes
Record attributes:  Carriage return carriage control
RMS attributes:     None
Journaling enabled: None
File protection:    System:RWED, Owner:RWED, Group:RWED, World:RWED
Access Cntrl List:  (IDENTIFIER=GIAC,ACCESS=READ+WRITE+EXECUTE)
                    (IDENTIFIER=*,ACCESS=NONE)
Client attributes:  None
```

Now user SANS has read, write, and execute access to the file MYFILE.TEXT. But SANS will not be able to delete the file.

```
$ DELETE/LOG MYFILE.TEXT;1
%DELETE-W-FILNOTDEL, error deleting DKA600:[SANS]MYFILE.TEXT;1
-RMS-E-PRV, insufficient privilege or file protection violation
```

UAF Security Privilege

Some privileges will allow a user to totally destroy the system. These system privileges should NOT be given to users unless absolutely necessary. Privileges are granted via the Authorization utility. The system privileges are:

SETPRV	Can set any privilege
SECURITY	Can perform security-related functions
READALL	Can provide read access to anything on the system
SYSMAN	Can insert object in system logical name table
DETATCH	Can create detached processes
LOG_IO	Can do logical I/O
PHY_IO	Can do physical I/O
PFNMAP	Can map to specific physical pages
SYSPRV	Can access object via system protection
CMEXEC	Can change mode to exec
CMKRNL	Can change mode to kernel
BYPASS	Can bypass UIC checking

```
UAF> MOD SANS/PRIV=BYPASS
%UAF-I-MDFYMSG, user record(s) updated
UAF> SHOW SANS

Username: SANS                          Owner:   SANS User
Account:  USERS                           UIC:    [100,100] ([USERS,SANS])
CLI:      DCL                             Tables: DCLTABLES
Default:  [USER]
LGICMD:
Flags:
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                 Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:  22-DEC-2002 14:36
Last Login: 22-DEC-2002 15:03 (interactive),            (none) (non-
interactive)
Maxjobs:         0  Fillm:       100  Bytlm:        64000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:       150  JTquota:       4096
Prclm:           8  DIOlm:       150  WSdef:         2000
Prio:            4  ASTlm:       250  WSquo:         4000
Queprio:         4  TQElm:        10  WSextent:     16384
CPU:        (none)  Enqlm:      2000  Pgflquo:      50000
Authorized Privileges:
  BYPASS       NETMBX       TMPMBX
Default Privileges:
  NETMBX       TMPMBX
Identifier                      Value           Attributes
  GIAC                          %X80080011
```

Reference Monitor Authorization Database

OpenVMS stores its security and user information in distributed databases. These databases are index files and should be protected using the same tools as described in the object security section. The UIC base access should be set as follows: "System:RWED, Owner:RWED, Group: nothing, World: nothing"

The following files are part of the Authorization database:

SYSUAF.DAT	Holds the username, passwords, and UICs
NETPROXY.DAT	Holds username for remote access translations
RIGHTSLIST.DAT	Holds rights identifiers
VMS$OBJECTS.DAT	Holds UICs, Protection codes, ACL
VMS$AUDIT_SERVER.DAT	Holds security configuration parameters

These files are normally located in the SYS$SYSTEM directory, but can be located elsewhere if logical pointers are used to define their new location. It is strongly recommended for security reasons that these files be moved to a new location.

This can be done via the following steps:

```
$ COPY SYS$SYSTEM:RIGHTLISTS.DAT  DKA600:[SANS]
$ COPY SYS$SYSTEM:SYSUAF.DAT  DKA600:[SANS]
$ COPY SYS$SYSTEM:NETPROXY.DAT  DKA600:[SANS]
```

Edit the SYS$MANAGER:SYLOGICALS.COM and add the following lines

```
$ DEFINE/SYSTEM/EXEC RIGHTSLIST DKA600:[SANS]:RIGHTSLIST.DAT
$ DEFINE/SYSTEM/EXEC SYSUAF DKA600:[SANS]:SYSUAF.DAT
$ DEFINE/SYSTEM/EXEC NETPROXY DKA600:[SANS]:NETPROXY.DAT
$!
```

These file should never be edited directly. OpenVMS provides utilities to modify these files.

Reference Monitor Audit Trail

Auditing is the ability to monitor and report on security events. The auditing capability of OpenVMS is very extensive and fully configurable. Auditing information can be generated in three different ways:

1. Event category auditing
2. Object access auditing via the Access Control Entry (ACE)
3. User auditing by modifying the user authorization record

The audit event messages can be directed to two places: the /ALARM parameter sends the events to an operator terminal for live monitoring via the OPCOMM manager, and the /AUDIT parameter sends the events to an audit log file.

Event Category Auditing

Event categories are defined in classes. Each class can have subclasses. Auditing can be enabled for any class or subclass within a class. There are twenty classes of events (see table 1: Event Classes). It is not recommended to enable all event classes and subclasses, since this will generate a very large number of events and require a lot of resources from the system. By default OpenVMS will audit five classes (see table 1: Event Classes). To enable auditing for a specific class use the following command:

```
$ SET AUDIT /AUDIT /ALARM /ENABLE=TIME
```

To disable auditing use the following command:

```
$ SET AUDIT /ALARM /DISABLE=TIME

(Notice that in this command the TIME class was not disabled for the
AUDIT output)
```

To view which auditing is enabled on the system use the following command:

```
$ SHOW AUDIT
System security alarms currently enabled for:
  ACL
  Authorization
  Audit:         illformed
  Breakin:       dialup,local,remote,network,detached
  Logfailure:    batch,dialup,local,remote,network,subprocess,detached

System security audits currently enabled for:
  ACL
  Authorization
  Time
  Audit:         illformed
  Breakin:       dialup,local,remote,network,detached
  Logfailure:    batch,dialup,local,remote,network,subprocess,detached
```

To view the ALARM auditing live from any terminal session issue the following command:

```
$ REPLY/ENABLE=SECURITY

%%%%%%%%%%  OPCOM  30-DEC-2002 13:54:21.74  %%%%%%%%%%
Operator _AXPVMS$TNA6: has been enabled, username SYSTEM

%%%%%%%%%%  OPCOM  30-DEC-2002 13:54:21.74  %%%%%%%%%%
```

```
Operator status for operator _AXPVMS$TNA6:
SECURITY
```

From this point on all security events will be displayed to this terminal. Multiple terminals can be enabled for security. Security auditing can only be enabled from an account that has the security privilege set into its UAF record.

Table 1: Event Classes

Event Class	Default	Description
ACCESS		Specifies the type of access on which to generate events (i.e.: failure, success etc.)
ACL	X	Access to a file with a Security Access Control List events
AUDIT	X	Generate an event each time the SET AUDIT command is use; This class can not be disabled
AUTHORIZATION	X	Access to the System Authorization file
BREAKIN	X	All types of intrusion attempts
CONNECTION		Connection or termination of a connection to the system
CREATE		Creation of an object
DEACCESS		Deaccess of an object
DELETE		Deletion of an object
IDENTIFIER		Use of an identifier to access an object
INSTALL		Installation of new program on system
LOGFAILURE	X	Failed login attempts
LOGIN		Successful login attempts
LOGOUT		All logout
MOUNT		Mount and dismount operation
NCP		Access to the Network Configuration Databases
PRIVILEGE		Successful or unsuccessful use of privilege
PROCESS		Access to the system service processes
SYSGEN		Modification of any of the kernel parameters
TIME		Modification of the system time

Object Access Auditing Via the Access Control Entry (ACE)

It is possible to monitor specific objects for auditing using a security ACE within the object ACL. Monitoring can be done on specific access types and the auditing event can be sent to a live terminal (/ALARM) or to the audit log file (/AUDIT). The access types are the same as the one defined in the UIC base security plus two new types: SUCCESS and FAILURE.

Security auditing ACEs are attached to an object via the SET SECURITY/ACL command or the Access Control List editor.

There are three files that are not monitored with the default auditing mechanism.

SYSALF.DAT	Automatic login file for window based domains
OPERATOR.LOG	Operator log file use by the OPCOMM utility
ACCOUNTING.DAT	System accounting file

These files should be monitored for security auditing, thus this can be done by attaching a security ACE to the file.

To attach an Alarm ACE to the OPERATOR.LOG file enter the following command:

```
$ SET SECURITY/ACL=(ALARM=SECURITY,ACCESS=DELETE+CONTROL+SUCCESS+FAILURE) -
_$ SYS$MANAGER:OPERATOR.LOG
```

To attach an Audit ACE to this file the command is:

```
$ SET SECURITY/ACL=(AUDIT=SECURITY,ACCESS=DELETE+CONTROL+SUCCESS+FAILURE) -
_$  SYS$MANAGER:OPERATOR.LOG
```

If a terminal is enabled for security, the following message will be displayed:

```
%%%%%%%%%%  OPCOM  31-DEC-2002 08:39:48.12  %%%%%%%%%%
Message from user AUDIT$SERVER on AXPVMS
Security alarm (SECURITY) on AXPVMS, system id: 65534
Auditable event:          Object access
Event time:               31-DEC-2002 08:39:48.12
PID:                      00000224
Process name:             _TNA6:
Username:                 SYSTEM
Process owner:            [SYSTEM]
Terminal name:            TNA6:
Image name:               AXPVMS$DRA0:[SYS0.SYSCOMMON.][SYSEXE]SETSHOSECUR.EXE
Object class name:        FILE
File name:                _AXPVMS$DRA0:[SYS0.SYSMGR]OPERATOR.LOG;4
File ID:                  (6375,53830,0)
Access requested:         CONTROL
Sequence key:             002C3D3F
Status:                   %SYSTEM-S-NORMAL, normal successful completion
```

The ACL attached to this file should look like the following:

```
$ DIR/SEC SYS$MANAGER:OPERATOR.LOG

Directory SYS$SYSROOT:[SYSMGR]

OPERATOR.LOG;4       [SYSTEM]                          (RWED,RWED,RE,)
          (ALARM=SECURITY,ACCESS=DELETE+CONTROL+SUCCESS+FAILURE)
          (AUDIT=SECURITY,ACCESS=DELETE+CONTROL+SUCCESS+FAILURE)
```

User Auditing by Modifying the User Authorization Record

It is sometimes necessary to monitor a specific user activity on a system. This can be done by setting the AUDIT flag in the user record using the Authorize utility:

```
$ SET DEF SYS$SYSTEM
$ MC AUTHORIZE
UAF> MODIFY SANS /FLAGS=AUDIT
%UAF-I-MDFYMSG, user record(s) updated
UAF> EXIT
%UAF-I-DONEMSG, system authorization file modified
%UAF-I-RDBNOMODS, no modifications made to rights database
$
```

If a terminal is enabled for security the following message will be displayed:

```
%%%%%%%%%%  OPCOM  31-DEC-2002 09:00:03.68  %%%%%%%%%%
Message from user AUDIT$SERVER on AXPVMS
Security alarm (SECURITY) and security audit (SECURITY) on AXPVMS,
system id: 65
534
Auditable event:          System UAF record modification
Event time:               31-DEC-2002 09:00:03.68
PID:                      00000224
Process name:             _TNA6:
Username:                 SYSTEM
Process owner:            [SYSTEM]
Terminal name:            TNA6:
Image name:
AXPVMS$DRA0:[SYS0.SYSCOMMON.][SYSEXE]AUTHORIZE.EXE
Object class name:        FILE
Object name:              SYS$COMMON:[SYSEXE]SYSUAF.DAT;1
User record:              SANS
Flags:                    New:      AUDIT
                          Original: (none)
```

The operating system will generate an event for every action that is taken by SANS (i.e.: Login, Logout, files accessed, program executed, remote connection, etc.). The AUDIT flag generates an extremely large number of audit events.

To extract the information from the audit log use the following commands:

```
$ SET DEF SYS$MANAGER
$ ANALYZE/AUDIT/SELECT=(FLAGS=MANDATORY,USERNAME=SANS)-
_$ SECURITY.AUDIT$JOURNAL
```

Administering the Audit Log File

OpenVMS writes all audit events to the latest version of the file SYS$COMMON:[SYSMGR]SECURITY.AUDIT$JOURNAL. The audit file has the following characteristics:

- Binary Uses less disk space
- Clusterwide Same file for multiple nodes in cluster
- Sequential record Ease of access for user-written programs

This file will grow without limit, thus it can easily fill up the system disk when many events are monitored. It is good practice to rotate the audit file every day. This can be done via the following command:

```
$ SET AUDIT /SERVER=NEW_LOG
```

The same problem exists with the SYS$MANAGER:OPERATOR.LOG file. This file can be rotated with the following command:

```
$ REPLY/LOG
```

To save disk space, the previous day's audit and log files should be moved to another disk. The easiest way to accomplish this is to write a simple DCL program to perform these tasks and then submit it via a batch queue to execute at 23:58 every day (see example in Appendix A).

Logging security events is useless if the data stored in the audit log file is not analyzed. In OpenVMS this is done via the Analyze Audit utility. This utility allows extracting of information from the binary audit log files and storing them in a readable format. The command has the following syntax:

$ ANALYZE/AUDIT [file-spec[,...]]

This utility has many filtering qualifiers as described in the following table:

Table 2: AUDIT Filtering Qualifiers

Qualifier	Filtering	Description
/BEFORE	Content	Extract events messages generated prior to the specified time
/SINCE	Content	Extract events messages generated after the specified time
/EVENT_TYPE	Content	Extract events messages with the specified class
/SELECT	Content	Extract events messages based on data in the event message
/IGNORE	Content	Exclude events messages based on data in the event message
/BRIEF	Format	Produce a single-line record format report with basic information (this is the default)
/FULL	Format	Produce a multi-line record format report with all possible data for each record selected
/SUMMARY	Format	Produce a summary report of all records selected
/BINARY	Format	Produce a binary file to be further analyzed by a user-written program

/OUTPUT	Destination	Specify the report destination (default is the user terminal)

It is also recommended that the audit file be analyzed on a daily basis with the following command:

```
$ ANALYZE/AUDIT/BRIEF/SUMMARY/SINCE=TODAY/OUTPUT=AUDIT$DISK:20021227_AUDIT.LOG
-
_$ SYS$MANAGER:SECURITY.AUDIT$JOURNAL
$ MAIL/SUBJECT="Security audit" AUDIT$DISK:20021227_AUDIT.LOG SYSTEM
```

These steps can be added to the daily batch job (see example in Appendix A).

Remember, unless it is part of the routine task of the security administrator to look at this file, it will not be possible to detect any security issues on the system. Someone should be looking at this log file on a daily basis.

If a security issue is found in the security summary file, it is possible for the security administrator to drill down into the audit file using the Analyze Audit utility.

Conclusion

"Military contractors also accept OpenVMS as the one and only operating system for providing maximum security with maximum numerical throughput" [3]

A VMS team participated in the DEFCON9 competition. DEFCON9 is an annual computer underground conference for hackers held in Las Vegas, Nevada. At this conference hackers try to compromise systems to gain points. This is a statement made at this conference on OpenVMS.

"After 52 hours of playing, the DEFCON judges (a.k.a Goons) placed a note in the Scoreboard file that said that the Green Team's VMS box was "Virtually Unhackable" and that hackers might want to move on to another target." [4]

OpenVMS is a very structured Operating system. It has been around for over 25 years without any major security issues. Understanding the security model used in OpenVMS will greatly help in defining the rules required for setting a specific security level on a system.

OpenVMS has many security parameters, all configurable and tunable. Digital (or now HP) writes very good and comprehensive management manuals. I recommend that all security Administrators read the OpenVMS security Management manual (part of the OpenVMS management manual set).

This paper gives a fundamental overview of OpenVMS security. More information and utilities can be found at the following links:

www.openvms.org
www.pointsecure.com

Appendix A: Auditing program

Example program for maintaining the security audit file in OpenVMS.

```
$ SAY :== WRITE SYS$OUTPUT
$! ************************************
$! *  MAINTAINING_SECURITY_AUDITING.COM  *
$! ************************************
$!
$!  This procedure will perform the following tasks:
$!      1: Create a new daily SECURITY.AUDIT$JOURNAL file
$!      2: Create a new OPERATOR.LOG file
$!      3: Copy the previous day's Security audit file to a new disk and
$!         and rename it in the format: YYYYMMDD_SECURITY.JOURNAL
$!      4: Generate a summary report for today's security activity
$!         and save it with the name YYYYMMDD_SECURITY.REPORT
$!      5: Mail the report to the system account.
$!      6: Copy the previous day's operator log file to a new disk and
$!         and rename it in the format: YYYYMMDD_OPERATOR.LOG
$!     7: Resubmit itself for 23:58 tonight
$!
$!
$! PRIOR to start using this procedure:
$!     1: Define a logical for AUDIT$DISK: in SYS$MANAGER:SYLOGICALS.COM file
$!        Execute de file: $ @SYS$MANAGER:SYLOGICALS.COM
$!     2: Create 3 directories under the AUDIT$DISK:
$!           $ CREATE/DIR AUDIT$DISK:[LOG]
$!           $ CREATE/DIR AUDIT$DISK:[AUDIT]
$!           $ CREATE/DIR AUDIT$DISK:[COM]
$!     3:  Copy this procedure in the AUDIT$DISK:[COM] directory
$!
$!  To start this process run this procedure once manually
$!  by executing the following command from a privilege account:
$!
$!   $ SET DEF AUDIT$DISK:[COM]
$!   $ @MAINTAINING_SECURITY_AUDITING.COM
$!
$!
$! -------------- Start of procedure ---------------
$!
$! Check if the AUDIT$DISK logical exists
$ AUDIT_DISK = F$TRNLNM("AUDIT$DISK")
$ IF AUDIT_DISK .EQS. "" THEN GOTO LOGICAL_ERROR
$!
$! Get the date for the filenames
$!
$!
$! Determine the current day
$ date_string = F$CVTIME("TODAY",,"YEAR") + -
                F$CVTIME("TODAY",,"MONTH")+ -
                F$CVTIME("TODAY",,"DAY")
$!
$!
$! Create a new Audit Journal file
$ SET AUDIT/SERVER=NEW_LOG
$!
$! Create a new operator log file
$ REPLY/LOG
$!
```

```
$! Copy the Audit journal file to the audit disk
$ COPY/LOG SYS$MANAGER:SECURITY.AUDIT$JOURNAL;-1 -
          AUDIT$DISK:[AUDIT]'date_string'_SECURITY.JOURNAL
$!
$! Analyze today audit Journal file
$ ANALYZE/AUDIT -
         /BRIEF -
         /SUMMARY -
         /OUTPUT=AUDIT$DISK:[AUDIT]'date_string'_SECURITY.REPORT -
         AUDIT$DISK:[AUDIT]'date_string'_SECURITY.JOURNAL
$!
$! Mail the audit file to the SYSTEM account
$ MAIL /SUBJECT="Security audit" -
       AUDIT$DISK:[AUDIT]'date_string'_SECURITY.REPORT -
       SYSTEM
$!
$! Copy the operator log file to the audit disk
$ COPY SYS$MANAGER:OPERATOR.LOG;-1 AUDIT$DISK:[LOG]'date_string'_OPERATOR.LOG
$!
$! Resubmit the job for tomorrow at 23:58
$ SUBMIT /TIME=23:58 -
       /NOLOG -
       /NOPRINT -
       /QUEUE=SYS$BATCH -
         AUDIT$DISK:[COM]MAINTAINING_SECURITY_AUDITING.COM
$!
$ EXIT
$!
$LOGICAL_ERROR:
$ SAY " "
$ SAY "*** ERROR: AUDIT$DISK logical name not defined! ***"
$ SAY " "
$ SAY "****** MAINTAINING_SECURITY_AUDITING Terminating **********"
$ SHOW TIME
$ SAY " "
$ SAY " "
$ EXIT
$!
```

References

[1] Hoffman, S. H., *CGI Security.* Newsgroups: comp.os.vms, date: 1997/03/04

[2] Anderson, J. P., *Computer Security Technology Planning Study*. Technical report ESD-TR-73-51, Air Force Electronic System Division, Hanscom AFB, Bedford, MA, 1972

[3] Ceculski, B., *Why OpenVMS is better than linux.* Newsgroups: comp.os.linux.security Date: 2002/10/29

[4] Wisniewski, J. R., *Virtually Unhackable DEFCON9.* Point Secure Inc, White Paper 2002

[5] Wayne Sauer, *Applied VMS Security*. DECUS Symposium 1996. Presentation

[6]……http://www.cs.nps.navy.mil/people/faculty/irvine/publications/1999/wise99_RMCUnifySecFd.pdf

[7]……http://www.computer.org/proceedings/s&p/7828/7828toc.htm

[8]……http://www.openvms.compaq.com/doc/731FINAL/DOCUMENTATION