Best of Ask the Wizard

Steve Hoffman "Hoff"
OpenVMS Consulting Engineer

Upward Compatibility and OpenVMS Releases

Full upward binary compatibility of all user-mode applications is a central goal of HP OpenVMS Engineering, and a key OpenVMS customer expectation.

On OpenVMS Alpha systems, OpenVMS Engineering assumes, works for, tests for, and targets upward-compatibility of user-mode applications; applications that are specifically using only documented interfaces, that are lacking latent bugs, that are not using position-dependent coding constructs, and that are not specifically dependent on the OpenVMS version number itself or on a version-dependent product.

Like OpenVMS Alpha, OpenVMS VAX releases are expected to provide upward-compatibility of all valid user-mode code. On OpenVMS VAX V6.0 and later systems, OpenVMS Engineering further and additionally assumes that privileged-mode software application code such as device drivers -- again code that is using only documented interfaces, lacking latent bugs, and not otherwise version-dependent nor position-dependent -- will also be upward compatible.

To validate upward-compatibility, OpenVMS Engineering uses a variety of mechanisms. Examples of these mechanisms include source code reviews, statistical quality controls, an extensive regression test suite, tools which verify the values of constants and symbols, and programs for distributing early copies of OpenVMS releases via software developer kits (SDKs) and targeted field test efforts. All of these mechanisms strive for the early identification of compatibility problems, and for the rapid remediation of regressions.

User-mode or privileged-mode application code containing latent bugs, position- or version-dependent code, or that are using undocumented interfaces may well require rework or rebuild as ECO kits are applied or as OpenVMS is upgraded. Application code that fails to consistently check for return status values, that has implicit (and undocumented) assumptions of completion, synchronization, or any of a host of other latent bugs can and occasionally does fail after OpenVMS upgrades. An introduction to some of the more common latent coding errors is

included in topic (1661) of the HP OpenVMS Ask The Wizard site at the following URL: http://www.hp.com/go/openvms/wizard/

Privileged-mode code may require rework across major OpenVMS Alpha releases.

That said, HP OpenVMS Engineering has broken upward compatibility in a few product areas and on a few occasions, the removal of Display Postscript from DECwindows being a salient example. (While not specifically a change in OpenVMS itself, this change has tripped at least one layered product.) Latent problems within an Alpha code generator also triggered compatibility problems with the Alpha Architecture and with associated OpenVMS upgrades; OpenVMS and Alpha implementation upgrades caused the generated (application) code to fail on Alpha 21264 (EV6) and later. (Qv: the SRM_CHECK tool and information available at http://h71000.www7.hp.com/freeware/freeware40/21264/.) There are incompatibilities which can be caused by (deliberately) tightened handling of the file delete permissions starting at OpenVMS V6.0; security enhancements which broke a few applications and which also correctly prevented errant file deletion operations from occurring. But all this written, an application image that was built for the VAX/VMS V1.0 release is and will remain a part of the OpenVMS VAX regression test suite; upward compatibility is a central goal of HP OpenVMS Engineering.

The following lists show the types, contents, and intended customer audiences for various OpenVMS releases:

Note: full upward-compatibility of user-mode code -- that code using documented interfaces and lacking latent bugs -- is expected across OpenVMS upgrades.

Major release ("dot-zero" release)

- OpenVMS Alpha V7.0, OpenVMS VAX V6.0
- Key designator: the "dot zero"
- Contains new features, new capabilities
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration after the upgrade
- Often includes changes to kernel-mode data structures.
- Can require changes (recompile or reassembly, possibly recoding) of kernel-mode (privileged-mode) code

- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

Minor release ("dot" release)

- OpenVMS Alpha V7.1, OpenVMS VAX V6.2
- Key designator: the "dot not-zero"
- Contains new features, new capabilities
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration
- Few (compatible), or no, changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

Maintenance release ("dash" release)

- OpenVMS VAX V5.5-2, OpenVMS Alpha V7.1-2
- Key designator: the "dash"
- Not a vehicle for new features or new capabilities. These releases are intended for new hardware, as a roll-up of one or more maintenance fixes, to establish a common code base, or a combination of these.
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration
- Few (and compatible) or no changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

Limited Hardware Release ("LHR" or "hardware" or "H" release)

- OpenVMS Alpha V6.2-1H3, OpenVMS Alpha V7.1-1H2
- Key designator: the "H"
- Not a release vehicle for new features, new capabilities, nor bug fixes -- the hardware release is intended solely for new hardware and new configuration support
- Includes new hardware support
- Includes only those ECO kits that are directly affected by the work for new hardware support -- only those ECO kits and changes specifically required to prevent regressions are included in this release
- User-mode code using documented interfaces is expected to operate without alteration
- Few (compatible) or no changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- A limited battery of OpenVMS tests are run on this release
- The release is not a general release, and is targeted for use only on configurations including specific new hardware
- Customers must explicitly order this release

Special-Purpose Releases

- V7.2-6C2
- Key designator: other letters
- OpenVMS releases intended solely for specific customers, specific applications, or for specific and targeted purposes
- Can be a predecessor, variant, or descendant. No particular relationship to other OpenVMS releases can be assumed
- Compatibility may or may not be provided during any subsequent OpenVMS upgrades; some OpenVMS upgrades may require full application rebuilds for applications built against these specialized releases
- This release is not a general OpenVMS release, and is typically not shipped to contract support customers

OpenVMS System and Password Security

Various recent posting in the comp.os.vms newsgroup have recalled the attached piece of ancient OpenVMS programming trivia, once published in Phrack. (The attachment is from Phrack, Volume Three, circa June 1, 1989, and potentially also published elsewhere.) This trivia was posted as part of a discussion of the relative sensitivity of lists of usernames on OpenVMS systems.

In this newsgroup context, a Microsoft Windows-based password cracking tool, "John The Ripper", has been referenced. Specifically, the availability of new code within this tool intended to target OpenVMS password security. Depending on the intent of the particular tool user, "John The Ripper" is either intended for the cracking OpenVMS passwords, or for the testing of OpenVMS system password security. This and all similar tools require access to the contents of the OpenVMS password database (SYSUAF.DAT) as the input into the password attack scheme. Password attacks such as those used by "John The Ripper" are not particularly new (though clearly these techniques can and do improve, and the processor cycles that are available to crackers for these brute-force attacks clearly also increase over time), as there are and have been various password-cracking tools targeting OpenVMS over the years -- dictionary attacks, brute-force attacks, and research into the passwords likely used by the target user akin to that of the movie Wargames. ("Joshua" and "CPE1704TKS", for those wishing to collect the relevant movie trivia.) Native OpenVMS-based implementations of password-cracking tools are readily available.

As for this particular "John The Ripper" attack, the OpenVMS break-in evasion mechanisms effectively eliminate direct risks from these tools. Evasion prevents a user from repeatedly guessing passwords, locking out access temporarily or even permanently. Unfortunately, some application programs will also prompt for passwords, and provide either cleartext password storage or will not perform break-in evasion. (See sys$acm on V7.3-1 and later for a relevant and useful programming interface.)

Assuming that the contents of SYSUAF.DAT are not exposed to (untrusted) users on the running OpenVMS system, and evasion is not disabled, direct password attacks are effectively negated. Potential password exposure can be via unsecured or unencrypted BACKUP media libraries, or via unprotected and unencrypted network transfers, or untrusted privileged users. (These considerations are covered in the OpenVMS security manual.)

OpenVMS itself does not store cleartext passwords, only hashed passwords. There is no known way to reverse the password hashing algorithm; a Purdy polynomial. Various tools will perform what are referred to as dictionary attacks; trying combinations of cleartext passwords looking for a matching resulting hash value. Various other tools will search for occurrences where users and or applications have stored cleartext passwords within disk files.

The Purdy polynomial provides for large changes in the output quadword hashed value for small changes in the input. More importantly, the polynomial is also particularly difficult to reverse; to calculate the input values based on the output hash. Further, to prevent tools such as "John The Ripper" from simply building a library of pre-calculated hashed passwords, OpenVMS implements a value known as a salt. This value is unique to each username, and is incorporated into the input into the hash. And OpenVMS implements both a list of previously-used password hash values for each user, and a site-extensible dictionary of prohibited words. These two mechanisms prevent the reuse of passwords, and help the users avoid selecting weak passwords. Site-extensible password filtering is also available, allowing the security manager to further tailor the password processing.

Details of the Purdy polynomial and particularly of the OpenVMS password-hashing algorithm are deliberately available; the strength of the scheme is derived from the mathematics of the algorithm, and not from obscuring the particular implementation. The user interface is the sys$hash_password service, and C and Macro32 implementations of the password hash are widely available.

As for discussions of the exposure of lists of usernames on an OpenVMS system, no major formal security evaluation standards -- the NCSC "Rainbow" series security evaluation standards, the European ITSEC standards, or otherwise -- indicate that or believe that the exposure of lists of usernames is a security problem. Various of these standards do specifically mention such things as exposing classified project names or other sensitive data as usernames. But there is an expectation that usernames can and will be exposed. For instance, the surnames of workers can be available from various sources such as telephone lists or vehicle registrations, and surnames are often used as usernames.

On OpenVMS, usernames are not considered and not maintained as security-relevant objects, and there are various ways to acquire lists of usernames on OpenVMS, including (but not limited to) MAIL messages, PHONE commands, the finger command, SHOW USER and SHOW SYSTEM commands, and various other DCL commands, network applications, and programming interfaces.

OpenVMS as well as all other general-purpose operating systems provide what are known in the security world as covert channels; with ways to acquire or transfer secured information through unsecured channels. The SHOW SYSTEM command is an example of a covert channel on OpenVMS, as individual users can select a process name. As this process name is easily visible, some amount of information can be communicated among users. Accordingly, one the many aspects of an operating system security evaluation is an evaluation of the bandwidth of the covert channels; of the volume of security-critical information that can be transmitted through each covert channel.

In the case of the attached program, the covert channel involves finding entries in the rights list that (often, but certainly not always) match usernames. The attached adduser.pas program uses $idtoasc to look for these matching values. (Unlike John The Ripper, the attached adduser.pas tool provides only lists of usernames (assuming, as is common system management practice, that there are identifiers that match the usernames) and does not attempt to find the password -- nor does the attached example program have any way to find the hashed password value that is stored in the SYSUAF database.)

Another unfortunately common covert channel is the telephone. One of the most common password attacks involves social engineering, not any particular knowledge of the operating system security. Attackers can and commonly do locate and contact system support staff, will falsely identify themselves, will claim that a particular existing user's password has been forgotten, and will request that the password be reset. Shared usernames are another obvious target, and this security attack is one of the reasons why all shared usernames are strongly discouraged. If the attacker's password request is honored, system security is accordingly compromised, whether or not the target username is privileged.

Another potential covert security exposure can be an application that prompts users for passwords, that operates with privileges, or that operates in privileged processor modes. Such application can potentially compromise OpenVMS system security, and these applications can and will be targeted by attackers. This because users will tend to choose the same or similar passwords on a system, meaning that the insecure storage of a password within an application can compromise the OpenVMS password mechanisms. Applications that prompt for and subsequently verify passwords against application-private or against OpenVMS password storage can themselves also be targeted simply for password verification, particularly if the application does not provide for evasion. While OpenVMS break-in evasion prevents "John The Ripper" and similar tools from directly operating against OpenVMS password prompts, similar attacks can be

made against any application that prompts for passwords but that does not implement evasion. (As stated earlier, please see the OpenVMS V7.3-1 sys$acm system service.)

Information related to "John The Ripper" and pointers to the full Phrack article with the attached alluser.pas example program are available on the web.

A discussion of the relative weakness of password-based authentication is available at the OpenVMS Ask The Wizard area, with the core topic on this likely being (4612). Related topics include (1461), (1645), (4303), (4778), (5508), (6080), (6328), (7818), and others. The Ask The Wizard area is available via the following URL:

http://www.hp.com/go/openvms/wizard/

What does all this mean to most OpenVMS users and most customer sites? Please update your password dictionary to contain all words of local, regional, or site-specific significance, and please educate your users on the selection of appropriately-secure passwords. Locally-written password filters are also a potential option, and can reduce the ability of users to select weak passwords. Also train your users and particularly your support staff in related password vulnerabilities, and particularly also to avoid exposing passwords over the telephone or via cleartext email to remote (and thus what must always be assumed insecure) systems. Alternatively, consider configuring and using generated passwords, challenge-response, biometric, or other more secure identification and authorization mechanisms. Please also maintain the current OpenVMS ECOs and (as many security exposures involve a network) also maintain current network product versions and ECOs.

The following program, published in Phrack, was written by "Deep Thought of West Germany:"

```
{
* alluser.pas - get names of all users
* by Deep, 1989
* This program is freely redistributable as long no modifications are made
* DISCLAIMER: I take no responsibility for any use or abuse of this
*             program.  It is given for informational purpose only.
*
* program history:
```

```
* 04-May-89   started
* 02-Jun-89   clean up of code
}
[inherit ('sys$library:starlet.pen')]
program alluser(input,output);

  type $word      = [word] 0..65535;
       $byte      = [byte] 0..255;
       $quadword  = record
                      lo,hi : unsigned;
                    end;
       $uquad  = record
                      lo,hi : unsigned;
                    end;
var
  id: unsigned;
  status, status2: integer;
  length: $WORD;
  attrib,context,context2,context3: unsigned;
  ident, ident2: unsigned;
  name: varying [512] of char;
  holder: $uquad;

begin

writeln('Alluser - use at your own risk!');
status := SS$_NORMAL;
{ id = -1 selects next identifier }
id := -1;
context := 0;
while (status <> SS$_NOSUCHID) do
   begin
   { find next identifier }
   status := $idtoasc(id,name.length,name.body,ident,attrib,context);
   if (status <> SS$_NOSUCHID) then begin
      write(pad(name,' ',16));
      if (ident div (65536*32768) > 0) then
         { it's a rights-list, so print the hex-value of the identifier }
         begin
         writeln(oct(ident,12));
         context2 := 0;
```

```
        context3 := 0;
        { find all holders of this right }
        repeat
          holder := zero;
          status2 := $find_holder(ident,holder,attrib,context2);
          if (holder.lo <> 0) then begin
              ident2 := ident;
              { get UIC and username }
              status := $idtoasc(holder.lo,name.length,name.body,ident2
                  ,attrib,context3);
              write('                ',pad(name,' ',16));
              writeln('[',oct(holder.lo div 65536,3),','
                  ,oct(holder.lo mod 65536,3),']');
             end;
        until (holder.lo = 0);
        end
      else
        { it's a UIC, so translate to [grp,user] }
        begin
        writeln('[',oct(ident div 65536,3),',',oct(ident mod 65536,3),']');
        end;
      end;
   end;
end.
```