Migrating POSIX ACLs to NFSv4 ACLs

<Bob.Sneed@dell.com>

Dell EMC Isilon Corporate Advisory Engineer

Version 2.05 - July, 2018

ABSTRACT 2

DISCLAIMERS 2

1 Introduction 3

1.1 Terminology 4

1.2 OneFS Considerations 4

1.3 Privilege Considerations 4

1.4 Limitations 5

2 Coding Considerations 5

2.1 POSIX-to-NFS4 ACL Translation Algorithm 7

2.2 Additional Coding Notes 8

2.3 Identities 9

3 ACL Utility Programs 10

3.1 ‘xacls’ (Linux: ‘eXtract ACLs’) 10

3.2 ‘wacls’ (OneFS: ‘Write ACLs’) 11

3.2.1 ‘wacls’ Implementation Notes 12

3.3 ‘hacls’ (Portable: Native-to-CHEX ACL formatter) 12

3.4 ‘chexcmp’ (Portable: CHEX compare) 13

4 Summary 14

5 References 15

## ABSTRACT

Details are presented for the design and implementation of working C code for migrating POSIX ACLs from Linux to equivalent NFS4 ACLs on Dell EMC’s Isilon OneFS filesystem. Design and implementation considerations are also given for the various utility programs in which the code has been embedded to facilitate such migrations.

## DISCLAIMERS

This paper describes the design and implementation of **FREE CODE**. Both this paper and its accompanying code are being provided for instructional purposes only. There are no warranties, expressed or implied for this information or the corresponding C code, including any warrantees of correctness or suitability for any particular purpose.

***wacls.c*** is coded specifically to be built under OneFS. Compile and build tools for OneFS are not publicly available, but this code and pre-built OneFS binaries are nevertheless included in this bundle of free code. The building and use of custom C code natively on OneFS are operations that are not officially supported or condoned by Dell EMC Isilon.

As of this writing, this code is in a developmental ‘beta’ state. There may be inconsistencies between this document and the code, including references from the code’s internal comments that refer back to this document.

Opinions expressed herein are purely those of the author, Bob Sneed, and do represent any official views of Dell EMC Corporation or of anyone else.

# Introduction

This paper discusses implementation details of C code for converting POSIX[[1]](#footnote-1) ACL values to equivalent NFS4 NFS4 ACL[[2]](#footnote-2) values on Linux, and the specific use case of then applying the derived NFS4 ACLs to files on an Isilon OneFS target filesystem. The C code discussed here is primarily designed to be embedded in the ***pwalk***(1)[[3]](#footnote-3) tool to implement its ACL-related features, but is also used by some of the additional ACL-related utility programs described herein;

* ***xacls*** - ‘eXtract ACLs’ - a Linux-specific program for eXtracting POSIC ACL and DACL values and formatting them in various ways.
* ***hacls*** - ‘Hexify ACLs’ - a portable utility for converting NFS4 CITI- OneFS-formatted human-readable ACL representations into a common unambiguous hexadecimal representation. (See: discussion of the CHEX format herein.)
* ***chexcmp*** - ‘CHEX compare’ - a portable utility for interpreting CHEX-formatted permissions mask and flag values (defined herein), or comparing two such values.
* ***wacls -*** ‘Write ACLs’ - a OneFS-native utility for translating NFS4 ACLs to OneFS-native Security Descriptors and applying them natively under OneFS, out-of-band with respect to the NFS4 protocol per se.

***pwalk***’s ACL operations are entirely limited to generating NFS4-equivalent ACLs for each non-trivial POSIX ACL encountered, and either dynamically applying them to a target file using the +***wacls=<command>*** (‘Write ACLs’) option, optionally logging various representations of the NFS4 ACLs or showing them in various formats using the ***+xacls=[bin|nfs|hex]*** formatting options. Files and directories bearing no POSIX ACL or DACL, or a POSIX ACL that is determined to be trivial[[4]](#footnote-4) are silently skipped.

## Limitations

There is currently no ability to perform incremental ACL migrations. To address this, future ***pwalk*** refinements might include the ability to migrate not only ACLs, but also file owner, group owner, and POSIX mode bits information, perhaps optionally migrating ‘trivial’ ACLs as mode bits. This could result in a target file’s ACL being removed if the corresponding source file no longer bears an actual ACL.

## Terminology

There are some overloaded terms that can cause great confusion when discussing different ACL models. Here is a summary of the terms used in various contexts;

* CONTEXT: POSIX ACLs
  + ACL - The “Access Control List” for a file of directory; a set of ‘rwx’ ALLOW ACEs plus a possible ‘mask’ value.
  + DACL - Applicable only to directories, the “Default ACL” for files created within the directory, which is expressed as a distinct set of heritable ACEs.
* CONTEXT: Non-POSIX ACLs -- including OneFS native ACLs, NFS4 ACLs, and the Linux ‘Rich ACLs’ project which are yet to be released in standard Linux distributions -- all closely follow the NTFS Security Descriptor[[5]](#footnote-5) model which defines 14 permissions plus a few flag bits for control of ACE inheritance rules.
  + DACL - A “Discretionary ACL” for a file or directory/container; the ACL that controls access and permissions for an object. A DACL on a directory/container may include some ACEs that specify ACE inheritance rules or ACEs which are only for the purpose of ACE inheritance (‘inherit-only’). These are ‘discretionary’ in contrast to a ‘share ACL’, which may allow or deny operations before the DACL is considered.
  + SACL - A “System ACL”, which describes the auditing policy of the security descriptor. (OneFS does not use this mechanism for its auditing implementation. OneFS allows the OneFS root user to read a SACL, but it cannot written, and cannot be read by any non-root user.)

## OneFS Considerations

As of this writing, OneFS (8.1.x.x) does not directly support POSIX ACLs directly in any way.

Since OneFS is a multi-protocol platform, OneFS ACLs may be manipulated over various protocols, including SMB, NFS4, and the OneFS Platform API over HTTP -- and there are instances where illusions and mysteries can arise from such handling, especially in regards to the ordering of ACEs within an ACL.

* ***pwalk***’s POSIX-to-NFS ACL translation imposes its own canonical ordering of ACEs within an ACL, and ***wacls*** preserves that ordering when it applies ACLs.
* Windows clients may cause a OneFS ACL to be re-canonicalized, which could potentially change its meaning.
* On Linux, the CITI tools ***nfs4\_getfacl*** command takes the liberty of canonicalizing ACLs its own way, thus possible creating an inaccurate representation of how the ACL is actually represented and enforced on OneFS.

Strictly speaking, root privilege in not necessary to apply ACLs to any file within the /ifs filesystem. Any user who has the RBAC privilege **ISI\_PRIV\_IFS\_RESTORE** can bypass the entire permission mechanism, which allows designated backup/restore accounts to operate without root privilege.

## Privilege Considerations

Determining all SOURCE filesystem POSIX ACL values by ***pwalk*** or ***xacls*** requires Linux root authority over the SOURCE filesystem, and applying the corresponding NFS4 ACLs to the OneFS TARGET filesystem requires OneFS root authority for ***wacls*** operation. It can be problematic in some scenarios to simultaneously have root authority over both the SOURCE and TARGET filesystems.

When considering allowing OneFS command-line root operations directly from a Linux NFS client, consider the following;

* It is possible to configure ***ssh-agent(1)*** to cache passwords in ***pwalk***’s Linux execution context.
* It is possible to create root-user equivalence between Linux and OneFS and use ***rsh*** instead of ***ssh***. This method is frowned upon in any security-aware production environment.

With these options, it should be possible to migrate ACLs directly from the Linux command-line, with ***pwalk*** invoking a per-worker ***wacls*** job from OneFS root-enabled shell sessions. Alternatively, ACL extraction and translation can be decoupled from the ACL application process by capturing the NFS4 binary ACLs in per-worker .acl4bin files, transferring those files to be visible from OneFS, then applying them from a separate root login session under OneFS.

**BEWARE:** Some NFS client implementations may deny client-side filesystem operations based on client- cached permissions metadata rather than generating an actual RPC to the NFS server. Given that OneFS-returned mode bits are a very lossy approximation of an ACL when an ACL is present, the reason an operation is denied can be highly non-obvious. Client-side permissions enforcement is a violation of the NFS RFC, which requires that all permissions be evaluated by the NFS server. If an NFS client uses the ‘noac’ mount option to suppress such client-side metadata caching as a workaround to this issue, the performance implications will typically be severe.

# Coding Considerations

Given the diversity of ACL types and platform-dependent implementation variances, it is difficult to implement ACL-manipulating code in a neat cross-platform portable coding fashion. Indeed, the amount of detail involved is extraordinarily tedious, including consideration of the inherent complexities of the ACL models and APIs, platform-dependent factors, and the numerous details involved in translating algorithms into working code. Some of those considerations include;

* Modern Linux kernels include ***nfs4\_acl\_posix\_to\_nfsv4()*** and related routines in their NFS client implementation, but these functions are not exposed as application-level APIs. Therefore, ***pwalk*** uses its own ACL translation code as described herein.
* Linux and OneFS vary in the names used for NFS4 permission bitmask values and other NFS4-defined constants. For example, ACE4\_WRITE\_DATA as described in RFC 7530 is called NFS4\_ACE\_WRITE\_DATA in Linux. The actual bitmask values for these values are (and must be) consistent with the values shown in RFC 7530. This code normalizes to the ACE4\_\* names defined in RFC 7530.
* In my opinion, the C API for manipulating POSIX ACLs is quite cumbersome and quixotic. It’s a “*Terrible, Horrible, No Good, Very Bad*” API, so this code is designed to completely insulate callers from its complexities. In particular, all embedded use of dynamically-allocated data structures are consistently freed.
* Inasmuch as the C code described herein was developed to be embedded in the multi-threaded ***pwalk*** program, its entry points are all engineered to be MT-safe.
* In a ***pwalk*** execution context, the ACL translation code may be called billions of times over the course of a multi-day-long ***pwalk*** run, so it is essential that there be no memory leaks which could cause a ***pwalk*** program failure after a long running time.
* For files and directories where no actual ACL has been applied, a ‘synthetic ACL’ value may be returned by the ***acl\_get\_file()*** function, in which case such returned ACLs will always be ‘trivial’. ***pwalk*** makes no attempt to determine if a trivial ACL is real or synthetic, but rather just quietly skips them entirely -- except in the case of a directory that carries a DACL, in which case the trivial ACL is preserved and incorporated into the translated NFS4 ACL.
* The logic described herein is implemented entirely in ***pwalk\_acls.c***, and the header file ***pwalk\_acls.h*** defines the required data structures and forward declarations for including this code in a program such as ***pwalk***.

There are multiple ways an NFS4 ACL may be represented either in binary or human-readable formats. Understanding the diversity and limitations of these formats is essential context for understanding the representations used by the tools described herein;

* NFS4 TOOLS: The human-readable ACL representation used by the Linux ***nfstools[[6]](#footnote-6)*** ***nfs4\_setfacl*** command uses single-letter codes to convey each permission bit. This format has the limitation of being unable to fully represent all possible NFS4 ACL expressions due to its overloading of some of the letters used to represent permission mask and flags values. It is a rather cryptic format which is probably a challenge for most users to interpret accurately. With ‘-sn’, ***xacls*** will show this representation of the translated NFS4 ACLs. ***pwalk*** can also generate this representation with ***+xacl=nfs4***.
* ONEFS: The keyword vocabulary used by ***chmod(1)*** and ***ls(1)*** natively under OneFS uses keyword values to convey ‘standard’ and ‘generic’ compound permissions groupings as well as individual permission bits. The OneFS-specific groupings of bitmask values as described in ***chmod(1)*** and encoded into lookup tables in ***pwalk\_acls.h***. With ‘‑sb’, ***xacls*** will show this representation for each translated ACE, trying to match as closely as possible what ‘***ls -lend***’ would show under OneFS.
* CHEX: This compact format represents the NFS4 *access\_mask* and *flag* bitmask values using a compact and unambiguous dotted notation using hexadecimal values, e.g.; ‘1F01FF.00’. This format was created explicitly in the context of the tools defined herein. CHEX formatted ACE permissions can be generated by ***pwalk*** and ***hacls***, and can subsequently be translated back into human-readable form using ***chexcmp***.

The following binary formats are used in different contexts to represent ACLs in-memory or to transport them across a network connection;

* XATTR: Both POSIX and NFS4 ACLs are normally communicated to and from the filesystem via the get-and set extended attributes (***setattr/getxattr***) APIs. Specifically, the reserved names of extended attributes used for these purposes are;
  + - POSIX: **"system.posix\_acl\_access" (ACL), "system.posix\_acl\_default" (DACL)**
    - NFS4: **"system.nfs4\_acl"**

Neither ***pwalk*** nor ***wacls*** use these representations, but they might someday. This representation has the added complexity of requiring ‘*network-to-host byte ordering*’ coding considerations (ie: BYTEORDER(3) functions htonl(), htons(), ntohl(), and ntohs()). There is also an active Linux project under development called “Rich ACLs”[[7]](#footnote-7) which evidently employs XATTR ACL representations with the name **“system.richacl”.**

* BINARY: Any programmer-defined data structure which retains the required ACL content such that it can be converted to other formats. Specifically, the binary in-memory representation of an NFS4 ACL in ***pwalk*** code is the portable ***acl4\_t*** type defined in ***pwalk\_acls.h***, which includes an array of ***ace4\_t*** ACEs that are congruent with the IETF standard ***nfsace4*** structure. ***pwalk*** uses this format with the ***+wacls=<command>*** option to pass [ACL4,pathname] tuples to a ***wacls*** symbiont, and also with the ***+xacls=binary*** option which captures these values in ***workerNN.acl4*** output files.
* OPAQUE: Any implementation-dependent binary format is called ‘opaque’ when its actual binary representation is not exposed to application programmers, but known only within the implementations of the implementation-specific APIs that use them. ***wacls*** translates binary ***acl4\_t*** values sent from ***pwalk*** to their OneFS-specific OPAQUE representation in order to apply the NFS4 ACLs natively under OneFS.

## POSIX-to-NFS4 ACL Translation Algorithm

The following outline is a canonicalized reference interpretation of the algorithm presented in <https://tools.ietf.org/html/draft-ietf-nfsv4-acl-mapping-05> (“**Mapping Between NFSv4 and Posix Draft ACLs”)**. Outline references from here are referred to by comments in the ***pwalk\_acls*** code (“REF: <reference>”), and **CODING NOTE**s embedded here pertain specifically to the ***pwalk\_acls*** implementation. The original algorithm has been re-ordered and re-phrased in places, and specific NFS4 ACE4 mask values have been substituted for the permissions referred to in the reference document’s algorithm. In the reference document, the original algorithm is preceded by the paragraph; “***We now describe an algorithm which maps any POSIX ACL to an NFSv4 ACL with the same semantics, meeting the above requirements.***”

1. If a POSIX ‘mask ACE’ is encountered, retain its permissions mask value as **posix\_mask\_ace\_mask**; otherwise set **posix\_mask\_ace\_mask** to the value ‘7’ by default (representing ‘rwx’). This value will be considered in step 2.4.1.
2. Translate each POSIX ACE (excepting any mask ACE) to an NFS4 ALLOW ACE as follows;
   1. Set the NFS4 ACE **‘acetype4’** value to ACE4\_ACCESS\_ALLOWED\_ACE\_TYPE.
   2. Set the NFS4 ACE **‘who’** (grantee) value as follows;
      1. Translate the UID and GID values of ACL\_USER and ACL\_GROUP ACEs into NFS4 (“user@realm”) names using directory services, etc., as appropriate. CODING NOTE: we skip this, and just use ASCII representations of the numeric UID and GID values.
      2. Translate ACL\_USER\_OBJ, ACL\_GROUP\_OBJ, and ACL\_OTHER to the special NFS4 names "OWNER@", "GROUP@", and "EVERYONE@", respectively. CODING NOTE: POSIX ‘other’ and NFS4 EVERYONE@ are not conceptually identical, giving rise to the complexities of Step 4.
      3. Take note of the permissions mask from the EVERYONE@ ACE as **posix\_other\_ace\_mask**.
   3. Set the NFS4 ACE **‘aceflag4’** value as follows;
      1. If the ACE specifies a named (i.e.: numeric) group (but not in the GROUP@ ACE, or any of the other special entity ACEs), set the flagval to ACE4\_IDENTIFIER\_GROUP, else set it to zero. CODING NOTE: Since we bind a specific GID to GROUP@ ACEs at the time we process them, we will set ACE4\_IDENTIFIER\_GROUP for GROUP@ ACEs, unless they are DACL-derived ACEs which will be resolved to the special ‘creator\_group’ identity. This logic is marked in the code as ‘REF: 2.3.1-AMENDED’.
      2. If the ACE is part of a POSIX default ACL (DACL) on a directory, add ACE4\_INHERIT\_ONLY\_ACE, ACE4\_DIRECTORY\_INHERIT\_ACE, and ACE4\_FILE\_INHERIT\_ACE flags.
   4. Set the NFS4 ACE **‘acemask4’** from the permission bits of the POSIX ACE as follows:
      1. For all ACL\_USER, ACL\_GROUP, and ACL\_GROUP\_OBJ ACEs, remove any permissions not granted by the POSIX ACE’s **posix\_mask\_ace\_mask** noted in Step 1. Note that the mask does not apply to ACL\_USER\_OBJ (OWNER@) and ACL\_GROUP\_OBJ (GROUP@) ACEs.
      2. Unconditionally set acemask4 bits ACE4\_READ\_ACL, ACE4\_READ\_ATTRIBUTES, and ACE4\_SYNCHRONIZE
      3. If the read bit is set in the POSIX ACE, set ACE4\_READ\_DATA.
      4. If the write bit is set in the POSIX ACE;
         1. Set ACE4\_WRITE\_DATA and ACE4\_APPEND\_DATA. CODING NOTE: We also add ACE4\_DELETE[[8]](#footnote-8) here
         2. If the object carrying the ACL is a directory, set ACE4\_DELETE\_CHILD.
      5. If the execute bit is set in the POSIX ACE, set ACE4\_EXECUTE.
      6. If the ACE is for the special "OWNER@" trustee, set ACE4\_WRITE\_ACL and ACE4\_WRITE\_ATTRIBUTES.

At this point, we've created an NFSv4 ACL with the same number of ACEs as the POSIX ACL, excluding any mask ACE; all of them ALLOW ACEs.

1. Sort the NFS4 ACL’s ACEs into canonical order: OWNER@, users, GROUP@, groups, then EVERYONE@. CODING NOTE: Not implemented. Not necessary because mask comparisons in Step 4 are all relative to the **posix\_other\_ace\_mask** from Step 2.2.3. Canonicalization occurs only once, in Step 5. See “ADDITIONAL CODING NOTES” for details of the canonicalization sort used here.
2. If the bitmasks in the canonicalized NFS4 ACL are non-increasing (so no ACE allows a bit not allowed by a previous ACE), then we can skip this step. Otherwise, we need to insert additional DENY ACE's to emulate the first-match semantics of the POSIX ACL permission-checking algorithm.
   1. If an ACL\_USER, ACL\_USER\_OBJ, or ACL\_OTHER ACE fails to grant some permissions that are granted by the final EVERYONE@ ACE, then that ACE must be preceded by a single DENY ACE;
      1. The DENY ACE should have the same ‘who’ value and flags as the corresponding ALLOW ACE, but the bitmask on the DENY ACE should be the bitwise NOT of the bitmask on the ALLOW ACE.
      2. In each DENY ACE, these bits should be cleared; ACE4\_WRITE\_OWNER, ACE4\_DELETE, ACE4\_READ\_NAMED\_ATTRS, ACE4\_WRITE\_NAMED\_ATTRS, plus the ACE4\_DELETE\_CHILD bit should be cleared on non-directories.
      3. The resulting nfsmask4 bitmask value in each ACE4 must clear all bits which are not defined in the protocol.
   2. All of the ACL\_GROUP\_OBJ and ACL\_GROUP ACEs are consulted by the POSIX algorithm before determining permissions. To emulate this behaviour, instead of adding a single DENY before corresponding GROUP@ or named group ACEs, we insert a list of DENY ACEs after the list of GROUP@ and named group ACEs. Each group DENY ACE is determined from its corresponding group ALLOW ACE exactly as in the previous step. As before, these DENY ACEs should only be added when they are necessitated by a group ACE that is less permissive than the final EVERYONE@ ACE.
3. For directories with both an access ACL and a default ACL (DACL), we translate both the ACL and DACL as above, and then concatenate them. The order of the concatenation is unimportant. CODING NOTE: the ACL and DACL translations are accumulated in the same NFS4 ACL by two consecutive calls to pw\_acl\_xlat\_add\_posix\_acl\_to\_acl4(), then the canonicalization sort groups all the DACL-derived ACEs after the ACL-derived ACEs.

## Additional Coding Notes

All NFS4 ACEs are accumulated sequentially, then, to enforce the ordering constraints of step 4 (the ordering of DENY ACEs), the following ordering is used to canonicalize the NFS4 ACL;

1. OWNER@ DENY
2. OWNER@ ALLOW
3. named user DENY
4. named user ALLOW
5. GROUP@ ALLOW
6. named group ALLOW
7. GROUP@ DENY
8. named group DENY
9. EVERYONE@
10. Although the relative ordering of POSIX ACL-derived ACEs relative to DACL-derived ACEs has no functional impact, all DACL-derived ACEs are ordered after all ACL-derived ACEs using the ordering shown above within each group of ACEs.

Since we will never generate an inherited ACE flag from a POSIX ACL translation (because POSIX ACLs do not track inheritance), our canonicalization does not need to consider that factor.

NFS4 ACLs have no analogue to the ‘mask ACE’ concept found in POSIX ACLs. During ACL translation, the POSIX mask value is simply applied to all non-OWNER@, non-GROUP@ ACE permission mask values. Therefore, unless the POSIX ‘mask ACE’ value is remembered somewhere independent of the ultimate OneFS-native Security Descriptor, it would be impossible for any OneFS-to-POSIX ACL translation algorithm to re-derive its value.

NFS4 ACLs have no direct analog to the POSIX DACL concept on directories, but rather implement similar functionality using NFS4 ACEs that specify NFS4 inheritance flags. All NFS4 ACEs derived from POSIX DACL ACEs will have their ACE4\_INHERIT\_ONLY\_ACE, ACE4\_DIRECTORY\_INHERIT\_ACE, and ACE4\_INHERIT\_ONLY\_ACE flag bits set (which sum to 0x0b) as their defining feature, and some may also carry the ACE4\_IDENTIFIER\_GROUP flag (0x40). This is because POSIX ACLs do not track whether or not a given ACE was inherited, and POSIX ACLs lack the more-robust fine-grained control over inheritance available the NFS4 ACL model.

The original algorithm does not incorporate the semantic intent of the source object’s sticky bit. In the case of directory objects, the sticky bit would suppress ACE4\_DELETE\_CHILD for non-owners of a file. [ **FOR FUTURE IMPLEMENTATION** ].

From ***pwalk***, the normal usage of **pwalk\_acls** module to copy ACLs basically reduces to two function calls to fetch/translate [1] and apply [3] the ACLs, plus possible reporting of errors from the **pwalk\_acls** module [2], plus a final call [4] to close the communications channel or file associated with the ***+wacls*** or ***+xacls*** options. Consider ***wacls.c*** to contain the basic reference coding method, as augmented by its inline comments, and summarizes by this fragment of pseudo-C-code;

int error;

char emsg[64];

char acl4OUTmode; // ‘p’ for pipe or ‘o’ for ordinary file

foreach (pathname) {

// [1] - Fetch ACL4 …

pw\_acl4\_get\_from\_posix\_acls(path, S\_ISDIR(sb.st\_mode), &acl4, emsg, &error);

// [2] - Cope with possible error in fetch …

if (pw\_acl\_errno) printf("ERROR: %s [%d - \"%s\"]\n",

emsg, error, strerror(error));

// [3] - Apply ACL4 to target (pipe) or capture in .acl4 output (file) …

pw\_acl4\_write\_binary(&acl4, path, &acl4OUT, acl4OUTmode, emsg, &error);

}

// [4] - When done iterating files, close down output …

pw\_acl4\_write\_binary(NULL, NULL, &acl4OUT, acl4OUTmode, emsg, &error);

Any errors in ***pwalk*** communicating with ***wacls*** are ***pwalk***-fatal, because the interprocess communication from ***pwalk*** to ***wacls*** is a one-way IPC pipe. Any errors that may occur during the ultimate application of a generated ACL4 to a target file are purely in the realm of ***wacls*** to manage or report in its ***/ifs/wacls/wacls\_<pid>.log*** files.

## Identities

These are the only identities used by ***pwalk***’s ACL-related operations;

*UID/GID - Numerical user and group IDs*

*OWNER@ - NFS4 reserved proxy for file owner*

*GROUP@ - NFS4 reserved proxy for file group owner*

*EVERYONE@ - NFS4 reserved proxy for ‘other’*

*OWNER\_USER - Trustee ID in heritable ACEs for the owner of a newly-created file*

*OWNER\_GROUP - Trustee ID in heritable ACEs for the owner group of a newly-created file*

***pwalk*** is never concerned identity names or name-to-number mapping. Refer to the ***wacls*** utility for more information on how these identities are used.

# ACL Utility Programs

## ‘xacls’ (Linux: ‘eXtract ACLs’)

The Linux ***xacls*** program is a handy command to extract POSIX ACL values, translate them to NFS4 ACL values, and format them into various formats. It also includes logic for debugging the ***pwalk\_acls*** module. ***xacls*** can be built as follows;

gcc xacls.c pwalk\_acls.c -lacl -o xacls

NOTE: Add -dH to enables core dumps; may require 'ulimit -c unlimited'

NOTE: debug cores with 'gdb xacls core.<pid>' ... 'bt'

While subject to change, ***xacls*** uses a fairly primitive command-line syntax;

# ./xacls

Usage: xacls [-d|h|i|o|p|- ...] [-sp] [-so] [-sh] [-sn] [-[o|p] <path|cmd>] <path> [<path> ...] ...

Where: -d -> enable DEBUG trail

-h -> help; show this usage() and exit

-i -> input path names from stdin (ignore non-option args)

-o -> output to <path> as a file (instead of -p)

-p -> output to <cmd> as a pipe (instead of -o)

-- -> end option list; next args are path names

-sp -> show POSIX ACLs input

-so -> show ACL4 values in OneFS format (under development)

-sn -> show ACL4 values as nfs4\_setfacl commands

-sh -> show ACL4 values in CHEX format

NOTE: MUST run as root!

***xacls*** only produces output for files with a non-trivial POSIX ACL or a POSIX DACL. The ***-sp*** option will format POSIX ACL values in a manner similar to the Linux-native ***getfacl(1)***;

# ./xacls -sp acltest.d/acl\_test\_d.1

# -------------------------------------------------

# file: "acltest.d/acl\_test\_d.1"

# owner: 0

# group: 0

# mode: 040750 (dir)

# POSIX ACL has 5 ACEs ...

u::rwx

u:501:r-x

g::---

m::r-x

o::---

# POSIX DACL has 3 ACEs ...

u::rwx

g::r-x

o::--x

# POSIX ACL plus DACL produced 7 NFS4 ACEs

The last line of the ***xacls -sp*** output shows the number of NFS4 ACEs produced by the translation algorithm. This number may be higher than the number of POSIX input ACEs, depending on how many DENY ACEs are produced during the translation. Wherever a POSIX mask ACE is present, there will be no corresponding NFS4 ACE, and all NFS4 ACE values will represent the ‘effective’ rights with the mask value applied.

Multiple options are provided for formatting the translated NFS4 ACL, as in the following examples. The last of these, the ***-sh*** CHEXoption, produces the same format which ***hacls*** should show when post-processing OneFS ‘***ls -lend***’ output for the applied ACLs -- which is particularly useful to simplify validating the end-to-end fidelity of the NFS4 transport and application functions.

# ./xacls -sn acltest.d/acl\_test\_d.1

nfs4\_setfacl A:OWNER@:rwaxtTcCy,A:501:rxtcy,A:GROUP@:tcy,A:EVERYONE@:tcy,A:OWNER@:fdirwaxDtTcCy,A:GROUP@:fdirxtcy,A:EVERYONE@:fdixtcy "acltest.d/acl\_test\_d.1"

# ./xacls -sh acltest.d/acl\_test.1

-rw-rwxr-- 1 0 0 0 acltest.d/acl\_test.1

0: A 160187.00 user:0 <OWNER@>

1: D 040107.00 user:500

2: D 040121.00 user:501

3: A 1200a0.00 user:500

4: A 120086.00 user:501

5: A 120081.00 user:502

6: A 1200a7.00 user:503

7: A 120081.40 group:0 <GROUP@>

8: A 120081.00 everyone

## ‘wacls’ (OneFS: ‘Write ACLs’)

***wacls*** is a OneFS-native binary utility program designed to apply NFS4 binary ACLs generated by ***pwalk*** or ***xacls***. Input ACLs are conveyed to ***wacls*** out-of-band with respect to the NFS4 NAS protocol per se, thus bypassing all identity translation issues that can arise with in-band NFS4 identity conveyance. ACL trustee identities in the input are constrained to be either numeric UID and GID values represented in ASCII or NFS4 special identities; OWNER@, GROUP@, and EVERYONE@.

***wacls*** will completely replace any ACL which may be pre-existing on the target file unless the ***-merge*** option is used, in which case ***wacls*** will merge any pre-existing ACL with the ACL which it will apply.

The usage summary for ***wacls*** is;

# ./wacls -h

Usage: wacls [-d|h|v ...] [-cd=<directory>] [-merge]

Where: -d - Sets DEBUG mode

-h - Prints this usage() and exits

-v - Sets VERBOSE mode

-cd=<directory> - Sets CWD context for passed path names

-merge - Merge applied ACLs with existing ACLs

NOTE: Operation \*requires\* root privilege!

ACL trustee values with NFS4 special identities are processed by ***wacls*** as follows;

* **OWNER@** will be persisted as the owner’s UID on OneFS, except in the context of a heritable directory ACE, in which case it will map to the well-known ‘creator\_owner’ identity (SID:S-1-3-0).
* **GROUP@** will be persisted as the owner’s GID on OneFS, except in the context of a heritable directory ACE, in which case it will map to the well-known ‘creator\_group’ identity (SID:S-1-3-1).
* **EVERYONE@** will be persisted as the well-known ‘everyone’ identity (SID:S-1-1-0) on OneFS.

***wacls*** normally operates silently, with its only output being error messages and optional verbose logging information to a log file (***/ifs/wacls/wacls\_<process\_id>.log***) under OneFS.

**IMPORTANT:** On the target OneFS cluster, the directory ***/ifs/wacls*** must exist prior to using ***wacls***!

**IMPORTANT:** ***wacls*** requires root privilege to do its job! This can be accomplished either by invoking it as the OneFS root user or by installing it as a setuid-root binary which would presumably be restricted to being run only by specific administrators.

**IMPORTANT:** Because root privilege is required, ***wacls*** *cannot* be used on OneFS Compliance-mode SmartLock clusters!

**IMPORTANT:** It is *assumed and required* that all numerical UID and GID values will be congruent between source and target file hierarchies! Otherwise, ***wacls*** would need to modified to map all numeric trustee values appropriately.

### ‘wacls’ Implementation Notes

The BSD-based OneFS system does not include command-line equivalent versions of the Linux NFS4 tools commands ***nfs4\_setfacl*** and ***nfs4\_getfacl***, but rather uses a syntactically distinct set of extensions to ***chmod(1)*** for manipulating OneFS-native ACLs. Therefore, the logic for translating NFS4 ACL values to their OneFS-specific equivalents is entirely contained in ***wacls.c***.

The method by which ***wacls*** will be invoked by ***pwalk*** must be passed to ***pwalk*** with the ***‑wacls=<command>*** option. This will normally be an **ssh** invocation that invokes the remote ***wacls*** binary command. In any event, the ***-wacls=<command>*** program must not cause a password prompt to occur, becasuse ***pwalk*** has no logic for answering a password challenge. Methods for implementing a password-free remote ***-wacls=<command>*** value include;

The C source for ***wacls*** contains additional implementation details, but the three main things to know about it are;

1. ***wacls*** will only operate on files in the /ifs filesystem, so all pathnames passed to ***wacls***, whether in [ACL,path] tuples or via its ***-cd=<directory>*** parameter must be in the '/ifs’ filesystem.
2. All ***wacls*** output goes to ***/ifs/wacls/wacls\_<process\_id>.log*** files, which requires that the ***/ifs/wacls*** directory previously exists and is writable.
3. There are two distinct implementations of ***wacls***; one for OneFS 8.0 (***wacls.c***) or later and the other for OneFS 7.2 (***wacls7.c***) or later. They differ in the method used to translate NFS4 ACLs to OneFS-native opaque ACL values before they are applied. OneFS 8.0 naturally accommodates the NFS4 ACL ‘flavor’, but on prior OneFS releases substantially more complexity is involved in translating NFS4 ACLs first to the OneFS-proprietary IFS ACL format, then to a OneFS Security Descriptor (SD) before being applied. ***As of this writing, only wacls7 has been fully developed, but it works equally well when compiled under OneFS 7 and OneFS 8, but separate binaries are required between OneFS versions due to underlying changes in their respective runtime environments.***

## ‘hacls’ (Portable: Native-to-CHEX ACL formatter)

***hacls*** was developed to aid in the end-to-end validation of ACLs translated and migrated by the tools described herein. It addresses the problem of being able to easily compare post-translation and post-migration NFS4 ACL values by translating various ACL representations into the compact hexadecimal (‘CHEX’) format described above. It is implemented as a simple Unix command-line filter program which ‘stream edits’ *stdin* to *stdout*, converting various platform-specific ACL representations to the CHEX format. Recognized input formats include OneFS ‘ls -lend’ and Linux ‘nfs4\_getfacl’.

Using the same example as the last in previous section, here’s how ***hacls*** reformats OneFS-formatted ACEs;

# ls -lend acltest.d/acl\_test\_d.1 | ./hacls -ogc

# -------------------------------------------------

# file: "acltest.d/acl\_test\_d.1"

# owner: 0

# group: 0

# mode: 040750 (dir)

0: A 1601a7.00 user:0 <OWNER@>

1: A 1200a1.00 user:501

2: A 120080.40 group:0 <GROUP@>

3: A 120080.00 everyone

4: A 1601e7.0b creator\_owner

5: A 1200a1.0b creator\_group

6: A 1200a0.0b everyone

***hacls*** performs the following normalizations for NFS4 ACEs;

* Each NFS4 ACE’s mask and flags are converted to the CHEX format.
* Well-known SID values are converted to their corresponding NFS4 special names.
* UID, GID, and SID trustee values that match the current file owner and group values are tagged with *<owner‑uid>, <owner-sid>, <group-gid>,* and *<group-sid>* tags respectively.
* Since there is no OneFS keyword which explicitly convey that an ACE’s trustee value is a GROUP, ***hacls*** will infer that the ACE4\_IDENTIFIER\_GROUP (0x40) flag is set for any trustee whose value which begins with ‘*group:*’.
* Inherited ACEs will show their ACE4\_INERITED (0x80) flag bit set.

As a simple command-line filter, ***hacls*** usage is quite simple;

# hacls -h

Usage: hacls [-ogc] < <bsd\_formatted\_acls>

Where: -ogc -> suppress output of OWNER:, GROUP:, and CONTROL: lines

## ‘chexcmp’ (Portable: CHEX compare)

The basic usage of ***chexcmp*** is to compare the bits set in two CHEX values, with ‘0’ and ‘1’ representing each bit being set in the first and second argument respectively.  For example;

# chexcmp 160183.00 160181.80

-- Permissions --

1 1 120081 \*GENERIC\_READ              'R' - generic read

1 1 000001 \*READ\_DATA/LIST\_DIRECTORY  'r' - (r)ead file data -or- list directory

1 0 000002  WRITE\_DATA/ADD\_FILE       'w' - (w)rite file's data -or- create file in dir

1 1 000080 \*READ\_ATTRIBUTES           't' - read basic A(t)TRIBUTES (non-ACLs) of a file

1 1 000100  WRITE\_ATTRIBUTES          'T' - write basic a(T)tributes (non-ACLs) of a file

1 1 020000 \*READ\_ACL                  'c' - read A(c)L

1 1 040000  WRITE\_ACL                 'C' - write A(C)L

1 1 100000 \*SYNCHRONIZE               'y' - use object as IPC s(y)nchronization primitive

-- Flags --

0 1 000080  INHERITED\_ACE             '-' - inherited ace (no CITI letter)

Note that a value of -1 can be used for either a mask of flag value to indicate ‘all defined bits’. So, for example, to see all bits that are or are not set for a given CHEX mask value, one could use;

# chexcmp 120081 -1

-- Permissions --

1 1 120081 \*GENERIC\_READ                 'R' - generic read

0 1 160106 \*GENERIC\_WRITE             'W' - generic write

0 1 1200a0 \*GENERIC\_EXECUTE             'X' - generic execute

0 1 1f01ff  MASK\_ALL                  'A' - mask all

1 1 000001 \*READ\_DATA/LIST\_DIRECTORY     'r' - (r)ead file data -or- list directory

0 1 000002 \*WRITE\_DATA/ADD\_FILE          'w' - (w)rite file data -or- create file in dir

0 1 000004 \*APPEND\_DATA/ADD\_SUBDIRECTORY 'a' - (a)ppend file data -or- create subdirectory

0 1 000008 \*READ\_NAMED\_ATTRS             'n' - read (n)AMED attr of file or directory

0 1 000010 \*WRITE\_NAMED\_ATTRS            'N' - write (N)amed attr of file or directory

0 1 000020 \*EXECUTE                     'x' - e(x)ecute file -or- traverse directory

0 1 000040 \*DELETE\_CHILD                'D' - (D)elete file or directory within a directory

1 1 000080 \*READ\_ATTRIBUTES             't' - read basic A(t)TRIBUTES (non-ACLs) of a file

0 1 000100 \*WRITE\_ATTRIBUTES            'T' - write basic a(T)tributes (non-ACLs) of a file

0 1 010000 \*DELETE                       'd' - (d)elete file -or- rmdir directory

1 1 020000 \*READ\_ACL                    'c' - read A(c)L

0 1 040000 \*WRITE\_ACL                   'C' - write A(C)L

0 1 080000 \*WRITE\_OWNER                  'o' - write (o)wner and owner\_group attributes

1 1 100000 \*SYNCHRONIZE                 'y' - use object as IPC s(y)nchronization

# Summary

Files associated with this project may be distributed in various container formats, but will generally include the following files;

src/pwalk.c - Fairly-portable C source for the *pwalk* program

src/pwalk\_acls.c - C source for ACL-handling module used by *pwalk* and *xacls*

src/pwalk\_acls.h - C header file used by *pwalk, xacls, wacls, and hacls*   
src/pwalk\_onefs.[ch] - Functionality that will only build on OneFS  
src/pwalk\_report.[ch] - work-in-progress for generalized reporting capability

src/wacls.c - OneFS-specific *wacls* program

src/xacls.c - Linux-specific *xacls* program

src/hacls.c - Fairly portable *hacls* program

src/chexcmp.c - Very portable *chexcmp* program

src/Makefile\* - Makefiles for various platforms

bin/onefs7 - pre-built OneFS 7.x binaries (from 7.2.1.3)

bin/onefs8 - pre-built OneFS 8.x binaries (from 8.0.1.1)   
bin/linux - pre-built Linux binaries

doc/\*.{docx,xlsx,pptx) - Related Word, Excel, and Powerpoint files

Some distributions may contain additional ancillary files which should be self-documenting.

Here’s a quick crib sheet on how to build and use these tools;

* On Linux;
  + ‘make all’ per the Linux Makefile ('ln -s Makefile.linux Makefile; make all’). That should give you ***pwalk***, ***xacls***, ***hacls,*** and ***chexcmp*** -- all of which might best be installed in ***/usr/local/bin***
  + ***pwalk*** and ***xacls*** are normally operated with root privilege to be able to read the POSIX ACLs on all files. On Linux, either ***xacls*** or ***pwalk*** can generate the binary ACLs outputs required by ***wacls*** on OneFS.
    - For **xacls** usage, see above.
    - For ***pwalk*** usage, see its accompanying Powerpoint deck.
    - Note that ***pwalk*** can take multiple ***-xacls=<option>*** values, so one can obtain matched binary and BSD ACL outputs in a single invocation.
* On OneFS;
  + The binary versions of ***wacls*** and ***hacls*** may be supplied either in uuencoded format or in OneFS version-specific directories in the distribution tarball. These are to be installed on the OneFS cluster by the root user such that they can be invoked from whatever OneFS nodes the user requires.
  + ***mkdir /ifs/wacls*** -- This directory must exist for ***wacls*** to write its log files.
  + ***uudecode*** the respective OneFS binaries into a directory in ***/ifs***, such as ***/ifs/wacls.***
  + Copy OneFS version-specific binaries to ***/usr/local/bin*** on all OneFS nodes from which they will be run. To target all nodes, use the a OneFS command line such as; ‘***isi\_for\_array ‘cp /ifs/wacls/[wh]acls /usr/local/bin***’.
  + Note that ***wacls*** requires root permissions, and will refuse to run without it.
    - On OneFS, ***wacls*** applies binary ACLs from ***xacls*** or ***pwalk***, using ***-cd*=<dir>** to establish the correct CWD relative to the input pathnames;
    - ***cat <binary acls from xacls or pwalk> | wacls -cd=<testdir> [-merge]***
    - See ***/ifs/wacls*** for log files. A log file is only produced if errors occur of if the debug or verbose options were specified.
* On both Linux and OneFS;
  + ***hacls*** and ***chexcmp*** require no special privileges, since they is merely text formatters.
  + ***hacls*** can be used to normalize the ACEs of OneFS- or nfs4\_getfacl-formatted ACLs to the CHEX format described above.
    - OneFS: ***ls -lend <testdir>/\* | hacls -ogc > chex.onefs***
    - Linux: ***find <testdir> | ./xacls -i -sb | ./hacls > chex.linux***
  + ***chexcmp*** can be used either to interpret CHEX-formatted permissions and flags, or to compare two CHEX-formatted values.

# References

The POSIX-to-NFS4 ACL translation algorithm described herein is primarily derived from <https://tools.ietf.org/html/draft-ietf-nfsv4-acl-mapping-05> (“***Mapping Between NFSv4 and Posix Draft ACLs*”)**; the implementation details of which are precisely presented in this document.

A map between NFS4 permissions bits and the human-readable formats used by OneFS ‘ls -le’ and Linux nfs4\_getfacl command are tabulated in the spreadsheet at ==============

<http://www.citi.umich.edu/projects/nfsv4/pnfs/block/CITI+EMC%20report%2020061027.pdf> contains some history of thought from the now-disbanded CITI group.

<http://www.vanemery.com/Linux/ACL/POSIX_ACL_on_Linux.html>

<http://www.bestbits.at/richacl/draft-gruenbacher-nfsv4-acls-in-posix-00.html> includes much history of thought by A. Gruenbacher, who has authored several other ACL-related papers.

<https://wiki.freebsd.org/NFSv4_ACLs> describes many interesting considerations regarding how NFS4 ACLs have been implemented in FreeBSD, including the statements; “*Note that in the FreeBSD implementation, there is no support for translation between POSIX.1e and NFSv4 ACLs. Adding one is not planned either.*”

As a historical reference, Solaris implements NFSv3 POSIX ACL support for through the addition of a sideband protocol extension which was never incorporated into the official NFS standards. <https://docs.oracle.com/cd/E26502_01/html/E29007/gbacb.html> mentions that POSIX ACLs are now automatically translated to equivalent NFSv4 ACLs. From a Linux client, the ‘acls’ mount option may be required to enable this sideband protocol.

[Bug 1161164](https://bugzilla.redhat.com/show_bug.cgi?id=1161164) - nfs4\_setfacl, nfs4\_getfacl ignores DENY ace for DELETE, WRITE\_OWNER, NAMED\_ATTRS

1. POSIX ACLs are defined by IEEE 1003.1e draft 17 ("POSIX.1e"); a standards proposal that was withdrawn before it reached formal adoption as a standard. Nevertheless, the industry has extensively adopted POSIX.1e ACLs, and they are supported by various Linux filesystems and GPFS. (See also; [*https://www.kernel.org/doc/ols/2010/ols2010-pages-19-32.pdf*](https://www.kernel.org/doc/ols/2010/ols2010-pages-19-32.pdf)) [↑](#footnote-ref-1)
2. NFS4 ACLs are defined by RFC 7530 ([*https://tools.ietf.org/html/rfc7530*](https://tools.ietf.org/html/rfc7530)*)* and closely mimic Microsoft’s NTFS Security Descriptor model. [↑](#footnote-ref-2)
3. See [*PowerWalk (pwalk) - a High-Speed Treewalk Utility*](https://inside.emc.com/community/active/isilon_forum/isilon_practice/blog/2015/07/29/powerwalk-pwalk-a-high-speed-treewalk-utility) (EMC internal blog posting). This code is integrated in ***pwalk*** version 1.97. [↑](#footnote-ref-3)
4. An ACL is trivial if its entire effect can be expressed using POSIX mode bits on the octal range of [0000-0777]. However, for directories that carry a POSIX DACL, trivial ACLs must be preserved in the generated NFS4 ACL. [↑](#footnote-ref-4)
5. Microsoft’s Security Descriptors are the foundation for OneFS-native ACLs, NFS4 ACLs, and Linux ‘Rich ACLs. They implement the security model described in <https://msdn.microsoft.com/windows/hardware/drivers/ifs/security-descriptors> [↑](#footnote-ref-5)
6. NFS4 tools (***nfs4\_getfacl***, ***nfs4\_setffacl***, ***nfs4\_editfacl***) were created by now-disbanded Center for Information Technology Integration (CITI) at the University of Michigan, and are the de facto standard tools for working with NFS4 ACLs under Linux. CITI existed between 1986 and 2013, and its work in this area has not yet been supplanted by any ‘official’ standards. (See also: <http://www.citi.umich.edu/projects/nfsv4/linux/>) [↑](#footnote-ref-6)
7. “Rich ACLs” is a development project aimed at bringing NFS4-style ACLs to Linux natively (See also: <http://bestbits.at/richacl/>) [↑](#footnote-ref-7)
8. This ACE4\_DELETE bit was not specified in the reference document, but was added in this implementation. <https://docs.oracle.com/cd/E26502_01/html/E29007/gbacb.html> includes some useful discussion about setting ACE4\_DELETE vis-à-vis setting ACE4\_DELETE\_CHILD on the parent directory.  [↑](#footnote-ref-8)