Permission Layout and Mappings

Xxx

DFA Perms v1-v4

Only the file dfa and the file permission layout existed. Permissions were stored in the Accept1 table and the Accept2 dfa table did not exist meaning control audit and quieting at the rule level did not exist.

DFA Perms v5-v9

The v5 dfa format was introduced with AppArmor 2.3. It introduced the Accept2 dfa table, and the PolicyDB dfa to provide more permissions for new mediation classes.

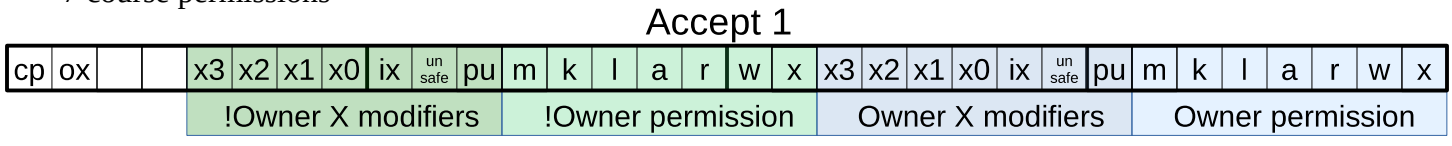
Permissions are stored in the Accept tables of the dfa. For both the file and PolicyDB dfa there are two 32 bit accept entries for each state, providing 64 bits of permission information per state. The encoding of the file dfa permissions is different than the encoding of the PolicyDB dfa.

Implicit Deny

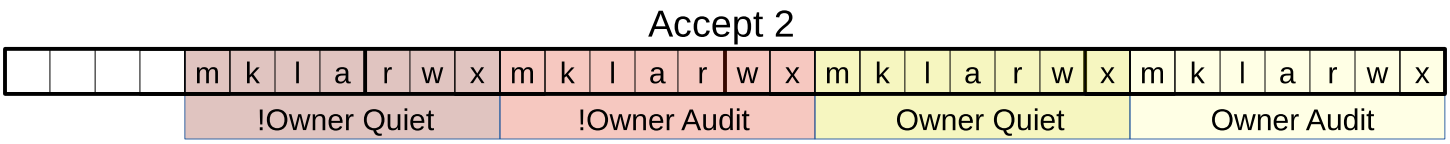
To save space because of the limited encoding bits explicit deny information was discarded after compile before minimization; only allow permissions and their related control bits were encoded. This meant that kernel side, even if userspace had an explicit deny rule, all deny information was implicit by allow permissions not being present. For regular policy enforcement this worked, but when in complain mode the difference between **audit deny** rules and not the profile not containing a rule could not distinguished, so in complain mode audit deny rules will result in ALLOWED audit messages the same as if the profile did not contain a matching rule.

File Permissions

7 course permissions



xx

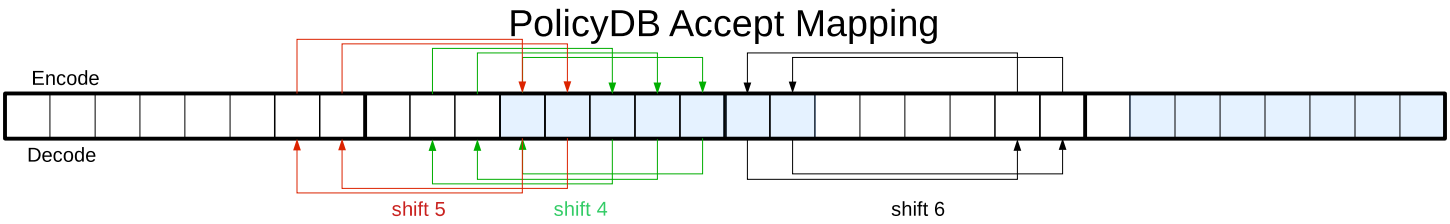


Policy DB Permissions

PolicyDB introduced a new flatter permission layout with more than seven permissions by throwing away the user conditional. However early PolicyDB layouts were still limited by the x modifiers as the backend dfa computation would mangle bit entries in the x modifiers permissions regardless of whether dfa was file or PolicyDB, so understanding the file dfa permission layout also helps with understanding PolicyDB permission layout.

Basic Accept Permission Mappings

The accept1 and accept2 table permission are mapped the same way from the front end encoding to a storage encoding that does not use the X modifier positions. The kernel will then reverse the mapping before it checks the permission bits. It is a little odd that it is done this when, but it was done with plans for the extended permissions (covered later) and then had to be cut back for staged implementation.



This provides 14 bits for permissions. The encoded permissions are shown in blue, in the diagram.

Network socket rules

xxx

v5-v6

Did not use the dfa. Table lookup based on family and type. Single bit in

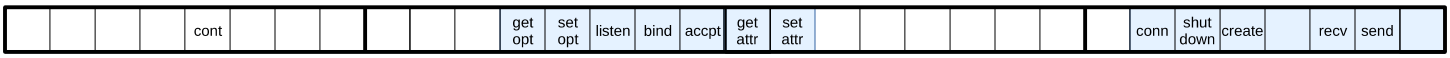
v7 af_unix-v8 net

Moved to encoding network permissions in the PolicyDB.

Accept1 Pre-encode

After mapping with map_perms() network permissions will be mapped such that none of what are the xmods positions in the file dfa are used.

Accept1 Post-encode



The kernel will reverse the encoding before the permissions are used, by using compute_perms() either at runtime, or during unpack in policy_compat.c

v7 – aa_class_net_compat
v8 – aa_class_net

newer versions of parser don’t mangle policydb, xmods positions. Still maps

link bit
cont

Extended Permissions – Perms Table 32

What both File and PolicyDB are mapped to in kernel

Allow userspace to use current kernel mapping

V1 – dev, unused

V2

Permissions extended to 32 bit field, with clean path to extend to 64 bit
Finer permissions

Additional information like audit, quiet, stored in separate 32 bit fields.
Explicit deny
Implied owner conditional, requires split file and PolicyDB

With the expansion of permission information a single permission entry became quite large. Instead of storing the permission information directly in the dfa accept entries. The permission information is stored in a separate table, that has at least 1 entry (no-permissions) and at most as many entries as there are states in the dfa.

The dfa accept1 entry is an index into the permission table, while the accept2 table is dropped. This allows the permission table to be deduplicated so that it only stores the unique permissions. Despite the expanded permission information it is possible for extended perms dfa + permissions table to take up less space than the old encoding with 64 bits of permission information stored on each state.

V3

V3 of extended permissions re-introduced the accept2 table, to provide conditional information and in so doing allowed for the dfa and PolicyDB dfas to be unified into a single dfa.