The Developer Meeting Agenda
- Advanced Security Features -

NEC OSS Promotion Center
KaiGai Kohei
<kaigai@ak.jp.nec.com>
My Topics in v9.1

• External Security Providers
  – Step.1: Reworks existing access controls
  – Step.2: Add security label support
  – Step.3: Add SELinux support

• Row-level Access Controls
  – A few issues to be resolved here
    • using VIEWs for row-level access controls
    • PK/FK constraints with RLS
    • And so on...
External Security Providers (0/3)

Step.0: Current implementation

Routine to do ‘something’ on xxx object

Database Objects

ACL
Owner

Logic

pg_xxx_aclcheck()

pg_xxx_ownercheck()
External Security Providers (1/3)

Step.1: Reworks existing access control

Routine to do ‘something’ on xxx object

Logic

check_xxxx_something()

Is it allowed?

Yes,

pg_xxx_aclcheck()

pg_xxx_ownercheck()

Database Objects

ACL

Owner
External Security Providers (2/3)

Step.2: Add security label support

Routine to do ‘something’ on xxx object

- Logic
  - check_xxxx_something()
  - Is it allowed?
  - Yes,

ACL
- pg_xxx_aclcheck()
- pg_xxx_ownercheck()

Owner
- Classified
- Public

Database Objects
Step.3: Add SELinux support

Routine to do ‘something’ on xxx object

Is it allowed?

Yes,

check_xxxx_something()

pg_xxx_aclcheck()

pg_xxx_ownercheck()

check_xxx_something()

SELinux

ACL

Owner

Classified

Public

Database Objects
Benefits

• Clear code separation between PostgreSQL and SELinux part
  – Loadable module may be an option?

• Allow to accept various security models
  – Not only SELinux
Step.1: Reworks existing access control

• Policy for reworking
  – At the execution stage
  – All the checks at once
  – Invocation as soon as possible, after all the needed informations are gathered

• Naming convention
  – check_<object class>_<action> (args, ...)
  – E.g) void check_relation_alter(Oid relOid, ...);
Example: creation of a new table

DefineRelation(....)
{
    namespaceId
        = RangeVarGetCreationNamespace(...);
        :
    pg_namespace_aclcheck(...);
        :
tablespaceId
    = get_tablespace_oid(...);
    if (OidIsValid(tablespaceId))
        pg_tablespace_aclcheck(...);
    MergeAttributes()
        :
    :        heap_create_with_catalog(...);
}
Example: creation of a new table

```
DefineRelation(...)
{
    namespaceId
        = RangeVarGetCreationNamespace(...);
    tablespaceId
        = get_tablespace_oid(...);
    MergeAttributes(&supOids)
    : check_relation_create(namespaceId, tablespaceId, supOids);
    heap_create_with_catalog(...);
}
```

```
check_relation_create(...)
{
    pg_namespace_aclcheck(...);
    if (OidIsValid(tablespaceId))
        pg_tablespace_aclcheck(...);
    foreach (l, supOids)
        pg_class_ownercheck(...);
}
```
Example: creation of a new table

DefineRelation(....) {
  namespaceId
      = RangeVarGetCreationNamespace(...);
  tablespaceId
      = get_tablespace_oid(...);
  MergeAttributes(&supOids)
      :
  check_relation_create(namespaceId, tablespaceId, supOids);
  heap_create_with_catalog(...);
}

cHECK_RELATION_CREATE(....)
{
  pg_namespace_aclcheck(...);
  if (OidIsValid(tablespaceId))
     pg_tablespace_aclcheck(...);
  foreach (l, supers)
     pg_class_ownercheck(...);
#ifdef HAVE_SELINUX
   sepgsql_relation_create(...);
#endif
}

CheckRelationCreate(...)
{
  pg_namespace_aclcheck(...);
  if (OidIsValid(tablespaceId))
     pg_tablespace_aclcheck(...);
  foreach (l, supers)
     pg_class_ownercheck(...);
  ifdef HAVE_SELINUX
     sepgsql Relation Create(...);
  endif
}
Issue: scale of reworks

• If we try to rework anything at once, the patch will too large to commit.

• The patch should be divided into per object class basis.
  – About 200-500 Line/Patch in most cases
Step.2: Security label support

- Security label
  - A text identifier used to MAC security
    - In DAC, similar to owner-id and ACLs
    - E.g) “system_u:object_r:postgresql_db_t:s0”
- Requirement
  - Capability to assign a text label on an object
    - Note: massive number of objects tends to share small number of security labels.
Access control decision

- The default PG permissions
  - S: Database User-Id
  - T: Ownership/ACLs of the object
  - A: defined in the model (ACL_SELECT, ...)

- Labeled based MAC (such as SELinux)
  - S: Label of the client
  - T: Label of the object
  - A: defined in the model (db_table:{select}, ...)
Plan: The way to store labels

- A tuple has a security identifier (4-bytes), if HEAP_HASSECEID is set
  - Similar to OID management
- Text representation is on pg_seclabel system catalog
- Service routines translate them each other

**HeapTupleHeaderData**

- uint16 t_infomask;
- uint16 t_infomask2;
- uint16 t_hoff;

**Oid Security Identifier**

**Oid Object Identifier**

**Data contains**

**seclabelTransIn()**

**seclabelTransOut()**

**Other PostgreSQL routines**

<table>
<thead>
<tr>
<th>relid</th>
<th>secid</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1259</td>
<td>16384</td>
<td>'system_u:object_r:sepgsql_table_t:s0'</td>
</tr>
<tr>
<td>1259</td>
<td>16385</td>
<td>'system_u:object_r:sepgsql_ro_table_t:s0'</td>
</tr>
<tr>
<td>1259</td>
<td>16386</td>
<td>'system_u:object_r:sepgsql_sysobj_t:s0'</td>
</tr>
<tr>
<td>2615</td>
<td>16408</td>
<td>'system_u:object_r:sepgsql_schema_t:s0'</td>
</tr>
</tbody>
</table>
Plan: Statement for management

• ALTER TABLE `<name>`
  SET WITH/WITHOUT SECURITY LABEL
  – add/remove ‘security_label’ system column
  – If no MAC, no storage needed for labels

• ALTER xxx `<name>`
  SECURITY LABEL TO ‘<label>’
  – It changes security label of the object
Plan: Step to label database

In the single user mode...
- ALTER TABLE xxx SET WITH SECURITY LABEL
- Execute Initial Labeling

No security label support in this path.
An Alternative (simplified) Idea

• Add “seclabel text[]” for labeled catalogs
  – Similar to reloptions

• Merits
  – Design is simple (suitable for the 1st phase)

• Demerits
  – Needs to redesign when RLS with MAC
  – Waste of storage, and unignorable performance loss

• Issues
  – Multiple security providers should be supported concurrently?
Step.3: Add SELinux support

• We need to do
  – Put SELinux hooks on the new security functions (at step.1)
  – SELinux code makes access control decision using security labels (at step.2)

• Which is more preferable?
  – SELinux code in #ifdef ... #endif block
  – SELinux code in Loadable-module
Row-level access controls

• Issues on the wikipage
  – Covert channel
  – Order to evaluate row-level policy
    Same issue with "using VIEWs for RLS"
      ➡️ Need helps from optimizer experts
  – TRUNCATE, COPY TO statement
  – Table inheritance
  – FK constraints
  – New grammar for RLS setup
Issue: Using VIEW for RLS (1/2)

- Order to evaluate scan qualifiers
  - x=1 should be earlier than user_func()
  - **order_qual_clauses()** sort the node within quals for the given scan plan

- Idea
  - FuncExpr should remember nestlevel?
Issue: Using VIEW for RLS (2/2)

- User defined function comes into JOINs
  - $a = x$ should be earlier than $\text{user_func()}$
  - $\text{distribute_qual_to_rels()}$ tries to chain the qual node on the scan node with least dependent

- Idea
  - Also, FuncExpr should remember nestlevel?
Trusted and Untrusted nodes

• Trusted nodes
  – Operator, Index access method, Type In/Out methods, Conversion, ...

• Untrusted nodes
  – User defined functions, others?

• Point of idea
  – If we can ensure the node is harmless, it can come into more deep nestlevel.
  – Index scan with user given condition, instead of SeqScan
RLS and FK Constraints

• Covert channels
  – No major RDBMS handles CC with RLS

• RLS and FK Constraints
  – FK is implement with secondary query
  – Using two modes
    • Filter mode
      – In normal, violated tuples are filtered
      – policy functions should be checked at first.
    • Abort mode
      – In FK checks, violated tuples cause an error
      – policy functions should be checked at last
Thanks for the discussion