Agenda: The Big Questions

• What is Node Access?
• How does Node Access work?
• What modules provide Node Access?
• Why does Node Access not do [insert feature X here]?
• How can we improve Node Access for Drupal?
Agenda: Node Access APIs

• Node Access terminology
• Defining access realms
• Defining access grants
• Development tools
• Best practices
What is Node Access?

• Drupal’s system for regulating which users can see which content.

• A function in the core Node module.

• An API for defining access to node content.
Site with Domain Access
Domain Access with OG on an allowed domain
DA and OG with private node on invalid domain
Access allowed by Domain Access
Both modules deny access
None shall pass
How does Node Access work?

- Define security permissions.
- Define blanket permissions.
- Define node module permissions.
- Define node access module permissions.
- Return TRUE or FALSE.
Security and global checks

```php
function node_access($op, $node, $account = NULL) {
    global $user;

    if (!$node) {
        return FALSE;
    }
    // Convert the node to an object if necessary:
    if ($op != 'create') {
        $node = (object)$node;
    }
    // If no user object is supplied, the access check is for the current user.
    if (empty($account)) {
        $account = $user;
    }
    // If the node is in a restricted format, disallow editing.
    if ($op == 'update' && !filter_access($node->format)) {
        return FALSE;
    }

    if (user_access('administer nodes', $account)) {
        return TRUE;
    }

    if (!user_access('access content', $account)) {
        return FALSE;
    }
}
Specific node module checks

```php
// Can't use node_invoke(), because the access hook takes the $op parameter
// before the $node parameter.
$module = node_get_types('module', $node);
if ($module == 'node') {
    $module = 'node_content'; // Avoid function name collisions.
}
$access = module_invoke($module, 'access', $op, $node, $account);
if (!is_null($access)) {
    return $access;
}

function blog_access($op, $node, $account) {
    switch ($op) {
    case 'create':
        // Anonymous users cannot post even if they have the permission.
        return user_access('create blog entries', $account) && $account->uid ?
    case 'update':
        return user_access('edit any blog entry', $account) || (user_access('e
    case 'delete':
        return user_access('delete any blog entry', $account) || (user_access(}
Node Access checks

// If the module did not override the access rights, use those set in the // node_access table.
if ($op != 'create' && $node->nid && $node->status) {
  $grants = array();
  foreach (node_access_grants($op, $account) as $realm => $gids) {
    foreach ($gids as $gid) {
      $grants[] = "(gid = $gid AND realm = '$realm')";
    }
  }

  $grants_sql = ''; if (count($grants)) {
    $grants_sql = 'AND (.implode(' OR ', $grants) .')';
  }

  $sql = "SELECT COUNT(*) FROM {node_access} WHERE (nid = 0 OR nid = %d) $grants_sql
  $result = db_query($sql, $node->nid);
  return (db_result($result));
}

// Let authors view their own nodes.
if ($op == 'view' && $account->uid == $node->uid && $account->uid != 0) {
  return TRUE;
}

return FALSE;
Come see the violence inherent in the system!
All node access systems are not created equal: 1

- Node Access modules cannot grant ‘create’ privileges.

```php
// If the module did not override the access rights, use those set in the node access table.
if ($op != 'create' && $node->nid && $node->status) {
    $grants = array();
    foreach (node_access_grants($op, $account) as $realm => $gids) {
        foreach ($gids as $gid) {
            $grants[] = "(gid = $gid AND realm = '$realm')";
        }
    }
}
```

- These are restricted to node modules and hook_perm.
All node access systems are not created equal: 2

- Node Access modules cannot act on unpublished nodes.

// If the module did not override the access rights, use those set in the
// node_access table.
if ($op != 'create' && $node->nid && $node->status) {
    $grants = array();
    foreach (node_access_grants($op, $account) as $realm => $gids) {
        foreach ($gids as $gid) {
            $grants[] = "(gid = $gid AND realm = '$realm')";
        }
    }
}

- These are restricted to administrators and super-users.
All node access systems are not created equal: 3

- The `{node_access}` table is not designed for CRUD.

- Design intent dictates database schema and enforces a limitation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nid</td>
<td>int(10)</td>
<td></td>
<td>UNSIGNED</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>gid</td>
<td>int(10)</td>
<td></td>
<td>UNSIGNED</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>realm</td>
<td>varchar(255)</td>
<td>utf8_general_ci</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grant_view</td>
<td>tinyint(3)</td>
<td></td>
<td>UNSIGNED</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>grant_update</td>
<td>tinyint(3)</td>
<td></td>
<td>UNSIGNED</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>grant_delete</td>
<td>tinyint(3)</td>
<td></td>
<td>UNSIGNED</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>
All node access systems are not created equal: 4

- Multiple node grants can cancel each other out.
All node access systems are not created equal: 5

- Node Access rules are collapsed by priority.

```php
function node_access_acquire_grants($node) {
    $grants = module_invoke_all('node_access_records', $node);
    if (empty($grants)) {
        $grants[] = array('realm' => 'all', 'gid' => 0, 'grant_view' => 1,
    } else {
        // retain grants by highest priority
        $grant_by_priority = array();
        foreach ($grants as $g) {
            $grant_by_priority[intval($g['priority'])][] = $g;
        }
        krsort($grant_by_priority);
        $grants = array_shift($grant_by_priority);
    }
    node_access_write_grants($node, $grants);
}
```
Many (happy) returns

• Eight returns.

• Defaults to FALSE == good.

• Finding the conflicts in your code can be a burden.

• No hooks to alter other access grants.
Node modules should not restrict ‘view’

<table>
<thead>
<tr>
<th>Module</th>
<th>create</th>
<th>view</th>
<th>update</th>
<th>delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>blog</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>forum</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>node</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>poll</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>project</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>image</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Making it work

- Crucial concepts
- Realms
- Grant Id [GID]
- grant_view
- grant_update
- grant_delete
node_access} defaults

- The default row in the table must be present unless other node access modules are in use.

<table>
<thead>
<tr>
<th>nid</th>
<th>gid</th>
<th>realm</th>
<th>grant_view</th>
<th>grant_update</th>
<th>grant_delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>all</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Otherwise the queries all return null.

```sql
SELECT COUNT(*) FROM node_access WHERE nid = 0 AND ((gid = 0 AND realm = 'all')) AND grant_view >= 1

SELECT COUNT(*) FROM node n WHERE n.promote = 1 AND n.status = 1

SELECT n.nid, n.sticky, n.created FROM node n WHERE n.promote = 1 AND n.status = 1 ORDER BY n.sticky DESC, n.created DESC LIMIT 0, 10
```
**hook_node_access_records()**

- Defines the rules that are saved to the `{node_access}` table. This routine is run just after `node_save()`.

```php
function hook_node_access_records($node) {
    if (node_access_example_disabling()) {
        return;
    }

    // We only care about the node if it's been marked private. If not, it is treated just like any other node and we completely ignore it.
    if ($node->private) {
        $grants = array();
        $grants[] = array(
            'realm' => 'example',
            'gid' => TRUE,
            'grant_view' => TRUE,
            'grant_update' => FALSE,
            'grant_delete' => FALSE,
            'priority' => 0,
        );
        return $grants;
    }
}
```
What to return for each record

- Positional (not keyed) array, containing:
  - ‘realm’ --> A unique name for your grant. Multiple realms are allowed.
  - ‘gid’ --> A numeric identifier for the grant, indicating the context.
  - ‘grant_view’ --> TRUE or FALSE that users can view the node.
  - ‘grant_update’ --> TRUE or FALSE that users can edit the node.
  - ‘grant_delete’ --> TRUE or FALSE that users can delete the node.
  - ‘priority’ declarations are frowned upon, as they disable other modules.
Writing to `{node_access}`

- Never write directly to `{node_access}` when saving a node. Let the API handle it for you.

- You might need to insert default data here, but only in special cases.
Challenges to overcome
Storing your records

- `node_access_rebuild()` will empty and rebuild `{node_access}`.

```php
function node_access_rebuild($batch_mode = FALSE) {
  db_query("DELETE FROM {node_access} ");
  // Only recalculate if the site is using a node_access module.
  if (count(module_implements('node_grants'))) {
    if ($batch_mode) {
      $batch = array(
        'title' => t('Rebuilding content access permissions'),
        'operations' => array(
          array('_node_access_rebuild_batch_operation', array()),
          ),
        'finished' => '_node_access_rebuild_batch_finished'
      );
      batch_set($batch);
    }
  }
}
```

- Be prepared!
Module access records

- Store your data in a safe place -- your own table.

```php
$schema[\'domain_access\'] = array(
    \'
```
```
fields\' => array(
    \'
```
```
    \'nid\' => array(\'type\' => \'int\', \'unsigned\' => TRUE, \'not null\' => TRUE, \'default\' => 0),
    \'gid\' => array(\'type\' => \'int\', \'unsigned\' => TRUE, \'not null\' => TRUE, \'default\' => 0),
    \'realm\' => array(\'type\' => \'varchar\', \'length\' => \'255\', \'not null\' => TRUE, \'default\' => \''),
    \'primary key\' => array(\'nid\', \'gid\', \'realm\'),
    \'indexes\' => array(
        \'nid\' => array(\'nid\'),
    ),
);`
```
```
- Store whatever data you need to rebuild your grants in the \{node_access\} table.
```
Declaring node grants
hook_node_grants()

• Determines the access rights for an individual user. These values are used to write the {node_access} SQL statement.

```php
function hook_node_grants($account, $op) {
  if (user_access('access private content', $account)) {
    $grants['example'] = array(1);
  }
  $grants['example_owner'] = array($user->uid);
  return $grants;
}
```

• $op may be view, update or delete.

• Your return value may vary based on the $op.
What to return for each grant

- An associative (keyed) array of grants, where the `realm` is the key and the value is an array of `grant ids`.

- `realm` --> A unique name for your grant. Multiple realms are allowed.

- `gid` --> A numeric identifier for the grant, indicating the context.

```php
$grants['my_grant'] = array(1, 2, 3);
-or-
$grants['user_grant'] = 10;
$grants['user_grant'] = 20;
```
How grants are applied

- When a page is requested, the $grants array is transformed into a JOIN query from the {node} to {node_access} table.
The system in action
Query #1: Should we bother?

- If a NULL count is returned, access is denied.

```sql
node_access_view_all_nodes()

SELECT COUNT(*) FROM node_access
WHERE nid = 0
  AND (.realm = 'all' AND gid = 0)
  OR (realm = 'domain_site' AND gid = 0)
  OR (realm = 'domain_id' AND gid = 16)
  )
  AND grant_view >= 1;
```
Query #2: Count the pages

- Send a pager_query() to count the output.

```
pager_query()

SELECT COUNT(*) FROM node n
    INNER JOIN node_access na
    ON na.nid = n.nid WHERE
    (na.grant_view >= 1
    AND ((na.realm = 'all' AND na.gid = 0)
    OR (na.realm = 'domain_site' AND na.gid = 0)
    OR (na.realm = 'domain_id' AND na.gid = 16)))
    AND ( n.promote = 1 AND n.status = 1 );
```
Query #3: Build the pages

- Send a pager_query() to build the output.

```sql
pager_query()

SELECT DISTINCT(n.nid), n.sticky, n.created
FROM node n INNER JOIN node_access na
ON na.nid = n.nid WHERE (na.grant_view >= 1
AND (((na.realm = 'all' AND na.gid = 0)
  OR (na.realm = 'domain_site' AND na.gid = 0)
  OR (na.realm = 'domain_id' AND na.gid = 16)))
AND ( n.promote = 1 AND n.status = 1 )
ORDER BY n.sticky DESC, n.created DESC LIMIT 0, 10
```
Tips from the wizard
The default grant

- Notice that the default grant is always checked.

- This is why node_access_rebuild() removes it if other node access modules are present.
Troubleshooting

• If no node access modules, check for ‘row zero’ in {node_access}

<table>
<thead>
<tr>
<th>nid</th>
<th>gid</th>
<th>realm</th>
<th>grant_view</th>
<th>grant_update</th>
<th>grant_delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>all</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• If multiple node access grants (or modules), check for conflicts.

```
SELECT COUNT(*) FROM node_access WHERE nid = 0 AND ((realm = 'all' AND gid = 0) OR (realm = 'domain_site' AND gid = 0) OR (realm = 'domain_id' AND gid = 16) OR (realm = 'og_user' AND gid = 2)) AND grant_view >= 1
```

• Remember the -OR- factor.
Developer tools: Devel Node Access

<table>
<thead>
<tr>
<th>node</th>
<th>realm</th>
<th>gid</th>
<th>view</th>
<th>update</th>
<th>delete</th>
<th>explained</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>domain_all</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Domain Access -- False: Only allows content from the active domain (ken.example.com) or from all affiliates.</td>
</tr>
<tr>
<td>Cui</td>
<td>domain_id</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Domain Access -- Viewable on example.com.</td>
</tr>
<tr>
<td>Cui</td>
<td>domain_site</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Domain Access -- Viewable on all affiliate sites.</td>
</tr>
<tr>
<td>Paratus</td>
<td>Lucidus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vel Eum</td>
<td>Vulputate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Developer tools: hook_node_access_explain()

- Tell people what your module does in everyday language!

```php
/**
 * Implements hook_node_access_explain for devel.module
 */
function domain_node_access_explain($row) {
    global $_domain;
    $active = $_domain['subdomain'];
    $domain = domain_lookup($row->gid);
    $return = t('Domain Access -- ');
    switch ($row->realm) {
        case 'domain_all':
            if (domain_grant_all() == TRUE) {
                $return .= t('True: Allows content from all domains to be shown.');
            } else {
                $return .= t('False: Only allows content from the active domain (%domain) or from all affiliates.', array('%domain' => $domain['subdomain']));
            }
            break;
        case 'domain_site':
            $return .= t('Viewable on all affiliate sites. ');
            break;
        case 'domain_id':
            $return .= t('Viewable on %domain.', array('%domain' => $domain['subdomain']));
            break;
    }
    return $return;
}```
Developer tools: build a debugger

**Debugging status:**
- [ ] Do not show debugging output
- [x] Show debugging output on node view

If set, users with the `set domain access` permission will be able to view the node access rules for each node. See the README for more details.

**Ibidem Luptatum Vel Ille Letalis Interdico**
Fri, 03/28/2008 - 20:47 — mubravitasw


**Subdomains**
- Drupal 2
- All affiliates

**Source domain:** Drupal 2

mubravitasw's blog  gichuwrawori  kado  chutheslovop  kethikucli  swehoba
Help us build a better system

• Saturday at 13:30

• Cisco BOF room
And there was much rejoicing!