MACE: Detecting Privilege Escalation Vulnerabilities in Web Applications

Maliheh Monshizadeh, UIC
Prasad Naldurg, IBM Research India
V. N. Venkatakrishnan, UIC

mmonsh2@uic.edu, venkat@uic.edu, pnaldurg@in.ibm.com
Introduction
Typical Web Application Architecture

- Applications run at the privilege of the **administrator** of database
  - With highest privileges
- Typically, **no** privilege minimization
  - Susceptible to **privilege escalation**
Privilege Escalation

- **Vertical Privilege Escalation**
  - e.g.:
    - XSS
    - Authentication Bypass

- **Horizontal Privilege Escalation**
  - Example: Banking Website
  - Can be caused by:
    - XSS, Session Fixation
    - IDs/Passwords Prediction
Vertical Priv. Escalation - Example

**verifyUser();**

```php
if($_GET['action'] == 'deleteAll')
    query("DELETE FROM tbl_articles");
```

Any logged-in user can delete all articles

**function verifyUser()**

```php
function verifyUser()
{
    if(!isset($_SESSION['userID']))
        header('Location: /login.php');
    else $userID = $_SESSION['userID'];
    return;
}
```
verifyUser();

if($_SESSION[$userID]['canWrite'] && $action == 'delete')
query("DELETE FROM tbl_articles WHERE article_ID = '' + sanit($_GET['article_ID']) + ''");

- Guessing a valid article_ID
- Hidden field in HTML code
- Auto-increment

Any logged-in user can delete other users’ articles
Problem & Challenges

- Missing/Improper authorization checks
  - A serious security threat for web applications
  - Facilitates an attacker to perform sensitive operations
- Problem: How to identify missing authorization checks?
  - Challenge 1: No authorization policy specification available
    - The only documentation is source code
  - Challenge 2: Manual analysis difficult or tedious
    - Applications are large, many thousand LOC
- Goal: must infer authorization policy from code
Inferring Access Control Rules

• How to infer policy from application, if policy implementation is not complete?
  • Intuition: For the same resources, in different access locations
    • Same access control rules

• Which conditions should hold for the user to be able to access a resource?
  • Code includes:
    • Path to the query
    • Query
Approach

- Identify **Authorization Context**
  - **Conditions** along the path to the query: Authorization checks
  - **Conditions** in the query: **Access parameters**

- Resource Access Analysis
  - Building an independent access model for **each resource**
  - Compare authorization contexts within each group
  - Detect **inconsistent** accesses
Why MACE?

- Focuses on authorization vulnerabilities
- A more precise and flexible authorization model for resources
- Detects both
  - Vertical privilege escalation
  - Horizontal privilege escalation
- **First tool** for automatic detection of Horizontal Priv. Escalation
MACE
MACE

Annotations

Computing Authz Contexts → Resource Access Analysis → Authz Context Comparison

- Find all resource accesses (DB queries)
- Compute Authz Contexts for them

Conflicts

PHP Application Code
Computing Authorization Contexts

- Application Code
- Annotations

Computing Authorization Contexts

- Finds Authorization Contexts

- Finds Authorization Contexts

Variables/Values
user
session
role/group
permissions
...

Control Flow Analysis → Dependency Analysis → Path Enumeration → Resource Access Analysis → Authz Context Comparison

Conflicts
Control Flow Analysis

- What authorization **conditions** hold at the query location?
  - **Traverse** Control Flow Graph to gather the conditions related to authorization
- Intra-procedural/Inter-procedural
- Context-sensitive, flow-sensitive

```
if($action == 'delete')
if(isset($_SESSION['userID']))
if($permission['canWrite'])
```

**Annotation**
- $_SESSION['userID']
- $userid['level']
- $permission['canWrite']
Control Flow Analysis

if ($action == 'delete')

if (isset($_SESSION['userID']))

if ($permission['canWrite'])

isset($_SESSION['userID'])

Annotation

$_SESSION['userID']
$userid['level']
$permission['canWrite']

Authorization Context
Control Flow Analysis

if($action == 'delete')

if(isset($_SESSION['userID']))

$permission['canWrite']

Authorization Context

isset($_SESSION['userID'])

$permission['canWrite']

Annotation

$_SESSION['userID']

$userid['level']

$permission['canWrite']
if($action == 'delete')

if(isset($_SESSION['userID']))

if($permission['canWrite'])

isset($_SESSION['userID'])

$permission['canWrite']

_annotation

$_SESSION['userID']

$userid['level']

$permission['canWrite']
Computing Authorization Contexts

- Application Code
- Annotations

**Computing Authorization Contexts**

- Propagates Authorization Context (Values)
- Builds Symbolic Queries

**Flow**

1. Control Flow Analysis
2. Dependency Analysis
3. Path Enumeration
4. Resource Access Analysis
5. Authz Context Comparison

Conflicts
Data Dependency Analysis

- Propagates Authorization Values for
  - If-statements
  - Symbolic Queries
  - Intra-procedural/Inter-procedural

```php
$userID = $_SESSION['userID'];
query("INSERT INTO tbl_comments VALUES ($userID , ...)");
```

**Annotation**

- $_SESSION['userID']
- $userid['level']
- $permission['canWrite']
- $userID
Computing Authorization Contexts

- Application Code
- Annotations

- Finds All Execution Paths

Control Flow Analysis → Dependency Analysis → Path Enumeration → Resource Access Analysis → Authz Context Comparison → Conflicts
Path Enumeration

- Authorization Contexts are **path-sensitive**
- Path Enumeration enumerates all paths leading to queries
- Therefore we have authorization contexts for all \(<\text{query}, \text{path}>\) pairs
- Intra-procedural/Inter-procedural

**Authorization Context -1**

\[
\text{isset}($_\text{SESSION}['\text{userID}'])
\]
\[
\text{$permission['canWrite']}
\]

**Authorization Context - 2**

\[
\text{isset}($_\text{SESSION}['\text{userID}'])
\]

\[
\text{if}($\text{permission['canWrite']}))
\]

\[
\text{if}(\text{isset}(\$_\text{SESSION}['\text{userID}']))
\]
MACE

Annotations → Computing Authz Contexts → Resource Access Analysis → Authz Context Comparison → Conflicts → PHP Application Code
Resource Access Analysis

- For each resource (DB table)
  - Gather resource access information
    - Authorization Context (Query-Path)
    - Access Parameters (Query WHERE Clause)
  - Compare authorization information
    - Insert-Insert, Insert-Update, Insert-Delete

```php
query("INSERT INTO tbl_articles VALUES ( sanit($_GET['article']),
  $_SESSION['userID'], ..."));
```

```php
query("DELETE FROM tbl_articles WHERE article_ID =" + sanit($_GET['article_ID'])
  + "," + "'');
```

<table>
<thead>
<tr>
<th>Authorization Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>isset($_SESSION['userID'])</td>
</tr>
<tr>
<td>$permission['canWrite']</td>
</tr>
<tr>
<td>authorID = $_SESSION['userID']</td>
</tr>
</tbody>
</table>

≠

<table>
<thead>
<tr>
<th>Authorization Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>isset($_SESSION['userID'])</td>
</tr>
<tr>
<td>$permission['canWrite']</td>
</tr>
</tbody>
</table>
Detecting Conflicts

- What is the rationale for comparison?
  - INSERT queries: typically the creator of the data owns the data

- Different type of conflicts show different vulnerabilities
  - Authorization Context Conflicts (conflicts in the paths)
    - Vertical privilege escalation (VPE)
  - Access Parameter Conflicts (conflicts in the WHERE clause)
    - Horizontal privilege escalation (HPE)
Evaluation
Summary of Vulnerabilities

- 7 popular small to large PHP applications (1K to 90K LOC)
- **Overall 71 vulnerabilities found**
  - **66 previously unknown** vulnerabilities detected
- **5 out of 7** applications found to be vulnerable
- **Zero false positives**

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>VPE</th>
<th>HPE</th>
<th># of Vulns</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpns</td>
<td>News System</td>
<td>✔</td>
<td>✔</td>
<td>7</td>
</tr>
<tr>
<td>DCPPortal</td>
<td>CMS</td>
<td>✔</td>
<td>✔</td>
<td>46</td>
</tr>
<tr>
<td>DNScript</td>
<td>Domain Trading</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>MyBloggie</td>
<td>Blog</td>
<td>✔</td>
<td>✔</td>
<td>6</td>
</tr>
<tr>
<td>MiniBloggie</td>
<td>Blog</td>
<td>✔</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>SCARF</td>
<td>Conference Management</td>
<td>✔</td>
<td>✔</td>
<td>11</td>
</tr>
<tr>
<td>WeBid</td>
<td>Online Auction</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

- Our results demonstrate that the approach used by MACE is effective, scalable
Identified Vulnerabilities

- DCPPortal
- CMS
- VPE: Use of cookies for authorization
  ```php
  if (UserValid($_COOKIE["dcp5_member_id"])) {
    ...
  }
  ```
  Cookies are used to validate the users

- HPE: Users can delete any agenda items from the shared calendar
  ```php
  if ((isset($_REQUEST["action"])) && ($_REQUEST["action"] == "delete")) {
    $sql = "DELETE FROM $t_agenda WHERE id = " . $_REQUEST["agid"] . "";
  }
  ```
  untrusted source
Precision

- False Positives?
  - **Multiple** INSERT Queries with **inconsistent** Authorization Contexts
    - Which one is correct?

- False Negatives
  - SELECT Queries
    - Disclosure policy is not available
  - INSERT queries with empty authorization contexts
    - Subsequent DELETE & UPDATE queries undetected
Performance & Scalability

- Application size: 1K to 90K
- Analysis Time
  - 95%: Dependency Analysis/Path Enumeration
  - Resource Access Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>SLOC</th>
<th># DB Tables</th>
<th>Analysis Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpns</td>
<td>4.5K</td>
<td>13</td>
<td>8220</td>
</tr>
<tr>
<td>DCPPortal</td>
<td>89K</td>
<td>34</td>
<td>982</td>
</tr>
<tr>
<td>DNScript</td>
<td>1.5K</td>
<td>7</td>
<td>35093</td>
</tr>
<tr>
<td>MyBloggie</td>
<td>6K</td>
<td>5</td>
<td>373</td>
</tr>
<tr>
<td>MiniBloggie</td>
<td>1.5K</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>SCARF</td>
<td>1K</td>
<td>7</td>
<td>54</td>
</tr>
<tr>
<td>WeBid</td>
<td>27K</td>
<td>47</td>
<td>1492</td>
</tr>
</tbody>
</table>
Related Work

• Prevention of Authorization Vulnerabilities
  • New Development Frameworks: Capsules.
  • Run-time Frameworks: Nemesis[5], Diesel[9], Swaddler[4], CLAMP[10].
  • MACE is focused on static detection of authorization vulnerabilities

• Finding Authorization Bugs in Legacy Web Applications
  • e.g.: RoleCast[11]
  • MACE uses a fine-grained access model

• General Vulnerability Analysis in Applications
  • Using Program Invariants, Behavioral Patterns
  • e.g.: Waler[7], AutoISES[14], NoTamper[1]
  • MACE focuses on authorization vulnerabilities → better precision
Conclusion

- **Missing/improper** authorization a serious problem for Web applications

- **MACE:**
  - Best-effort tool for **automatic** detection of HPE, VPEs
  - 71 total, 66 unknown vulnerabilities were found
  - Reduces **weeks of testing** effort

- Lack of documentation about access control policies makes it challenging to detect vulnerabilities

- MACE demonstrates that it is possible to infer policies from code and use them for vulnerability detection
Questions?


• How to identify the authorization variables?
• MACE Extracts of Globals/SuperGlobals
• Developers Confirm
• Examples:
  • $_SESSION['user_id']
  • $userid['level']
  • $permission['can_write']
Annotation Example - MyBloggie

- $HTTP_SERVER_VARS[REMOTE_ADDR]
- $_SERVER[REMOTE_ADDR]
- $_SERVER[HTTP_HOST]
- $_SERVER[SERVER_NAME]
- $_FILES[userfile][name]
- $_FILES[userfile][type]
- $_FILES[userfile][size]
- $_FILES[userfile][tmp_name]
- $_SESSION[username]
- $_SESSION[user_id]
- $_SESSION[passwd]
Precision - False Positives?

- **Multiple** INSERT Queries with **inconsistent** Authorization Contexts

- Which one is correct?
  - Logged-in users can insert
  - Guest users can insert

query("INSERT INTO tbl_comments VALUES ( 
  $_SESSION['userID'], 
  ...)");

query("INSERT INTO tbl_comments VALUES ( 
  $_SERVER['REMOTE_ADDR'], 
  ...)");

- Mitigation technique: Comparison with the strongest context
Precision - False Negatives

- SELECT Queries
- Missing Authorization Information in INSERTs

query("INSERT INTO tbl_articles VALUES ( sanit($_GET['article_code']), 
$_GET['userID'],
sanit($_GET['article_msg']))");

query("DELETE FROM tbl_articles WHERE article_ID =" + sanit($_GET['article_ID']) + "'\n";

- MACE reports the insert queries which have empty contexts
## Detected Violations

<table>
<thead>
<tr>
<th>Application</th>
<th>Number of Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insert-insert</td>
</tr>
<tr>
<td>phpns</td>
<td>0</td>
</tr>
<tr>
<td>DCPPortal</td>
<td>0</td>
</tr>
<tr>
<td>DNScript</td>
<td>0</td>
</tr>
<tr>
<td>MyBloggie</td>
<td>0</td>
</tr>
<tr>
<td>MiniBloggie</td>
<td>0</td>
</tr>
<tr>
<td>SCARF</td>
<td>1</td>
</tr>
<tr>
<td>WeBid</td>
<td>0</td>
</tr>
</tbody>
</table>