Nemesis: Preventing Web Authentication & Access Control Vulnerabilities

Michael Dalton, Christos Kozyrakis
Stanford University

Nickolai Zeldovich
Massachusetts Institute of Technology
Web Application Overview

FS/DB access executed with full app privileges!
Web Authentication is Broken

- **Semantic Gap** – independent auth sys
  - Web Authentication vs. DB, FS, LDAP, ...

- Webapps are effectively setuid progs
  - All FS, DB ops have privs of webapp
  - Not privs of webapp user (Confused Deputy)

- Programmer must insert auth checks
  - Check web app user before **all** FS/DB op
  - Safe only if programmer is **perfect**
And in the real world…

- Programmers forget auth/ACL checks
  - Authentication/Authorization OWASP Top 10

- Difficult to prevent automatically
  - Each app has its own authentication system
  - Apps have different privilege/ACL systems

- Widespread, highly damaging
  - Vulns usually result in ‘admin’ access to app
Authorization Bypass Vulns

- **Resource access without authorization**
  - Missing authorization check
  - Incorrect authorization check

```php
if(client_authorized($_GET['fileName']))
    openFile($_GET['fileName'])
```

Add URL parameter: `filename=/etc/passwd`
Authentication Bypass Vulns

- Authentication without valid credentials
  - URL/Cookie Validation Error
  - Weak Crypto
  - Ruby on Rails

```php
if (isset($_COOKIE['user'])) {
    $userName = $_COOKIE['user'];
}
```

Edit cookie, add name/value pair: 'user=admin'
Ideal Auth/ACL System

- Only authenticates correctly/safely
  - No authentication bypass attacks

- Always enforces ACLs correctly
  - No authorization bypass attacks

- Existing systems fail on both counts
  - May authenticate unsafely if vulnerable
  - Do not enforce ACLs automatically
Nemesis Overview

- Stops authentication, authorization atks
  - Without requiring app auth code rewrites

- **Infers** when authentication done safely
  - Use DIFT to track auth credentials

- **Enforces** ACLs automatically on file/DB
  - ACLs specify privs for **web clients**
Nemesis System Overview

- Web App 1
- Web App 2
- Web App 3

Core Library
- ACL Enforce

Language Interpreter
- DIFT

- Intercept I/O ops for File ACLs
- Intercept SQL ops for DB ACLs
- Automatic auth inference
- 2 tag bits per object
- Tag prop on all object ops
Safe Authentication Inference

- Propagate user credential, taint bits
  - 2 tag bits per object (String, integer, etc)

- **Infer** when auth occurs safely
  - Tainted info compared equal to auth cred
  - Add check to string or array comparison op

- Record **authentication inferred** user
  - Auth bypass attacks do not change this user
$user = "$_GET['username']"
$user = mysql_real_escape_string($user)
$pw = md5sum("$_GET['password']")

 realtimepw = $db->query("SELECT pw FROM users WHERE userName =" + $user + ";")

if ($pw == $realpw) {
  Authenticated!
  
  Variable   T   P
  $user     R   G
  $pw       R   G
  $realpw   -   G
}
Authorization Enforcement

- Enforce ACLs on FS, DB access
  - Apply to *authentication inferred user*

- Restrict DB table/row, file access
  - Many tables store per-user rows

- Taint information used in some rules
  - New user registration
  - Password change
Attack Prevention

- **Authorization Bypass**
  - Nemesis ACLs enforced automatically
  - Not dependent on any app-enforced checks

- **Authentication Bypass**
  - Auth inference not affected by attack
    - Inference requires user input == password
  - **ACLs check inferred user**
  - Prevents access to any privileged resource!
Configuration Requirements

- **Authentication inference**
  - Table/column info for auth credentials

- **ACL enforcement**
  - ACL from sysadmin for DB, File access

- **Future work**
  - Current configuration provided by admin
  - Log DB, File ops along with inferred user
  - Auto-generate ACLs from logs
Nemesis Prototype

- Added DIFT support to PHP interpreter
  - Password, Taint bits for String, int, etc
  - Assume Raksha checking OS & PHP interpreter for low-level attacks

- Auth inference on string comparison
  - ==, != operators

- Don’t have a full SQL query rewriter
  - Had to manually insert DB checks
Experimental Results

<table>
<thead>
<tr>
<th>Application</th>
<th>Size (Lines)</th>
<th>Auth Lines Added</th>
<th>ACL Check Lines Added</th>
<th>Attack Prevented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Php iCalendar</td>
<td>13,500</td>
<td>3</td>
<td>22</td>
<td>Auth Bypass</td>
</tr>
<tr>
<td>PhpStat</td>
<td>12,700</td>
<td>3</td>
<td>17</td>
<td>Missing ACL Check</td>
</tr>
<tr>
<td>Bilboblog</td>
<td>2,000</td>
<td>3</td>
<td>11</td>
<td>Incorrect ACL Check</td>
</tr>
<tr>
<td>phpFastNews</td>
<td>500</td>
<td>5</td>
<td>17</td>
<td>Auth Bypass</td>
</tr>
<tr>
<td>Linpha Gallery</td>
<td>50,000</td>
<td>15</td>
<td>49</td>
<td>SQL Injection in Password Check</td>
</tr>
<tr>
<td>DeluxeBB</td>
<td>22,000</td>
<td>6</td>
<td>143</td>
<td>Missing ACL Check</td>
</tr>
</tbody>
</table>

No discernible performance overhead
Authentication Bypass: Bilboblog

- Internal login script internet accessible
  - Admin username and password undefined

- PHP + Register Globals = Fail
  - Undefined vars initialized by URL params

- Attacker supplies the admin password!
  - Ensures the ‘submitted’ password is equal
Protecting Bilboblog

- Vulnerable app does not perform auth
  - Compares user input to user input

- Attack has no effect on shadow auth
  - Attacker-supplied admin password is tainted
  - Does not have user credential bit set

- Access to privileged resources denied
  - ACL checks use shadow authenticated user
Authorization Bypass: DeluxeBB

- Forum supports private messages
  - Stored in DB, restricted to sender/receiver

- Invalid access control check
  - Malformed cookies bypass check entirely

- Attacker forges cookies
  - Can read arbitrary user’s private messages
Protecting DeluxeBB

- Nemesis does not parse app cookies
  - Maintains its own shadow auth cookies

- DeluxeBB has row ACL for pm table
  - ‘From’ or ‘To’ field = shadow auth user

- Exploit rendered harmless
  - Only read row if From/To shadow auth user
  - No information leaks can occur
Future Work

- Develop full language for ACLs

- Automate SQL query rewriting for ACLs
  - Database views/triggers (see related work)
  - MySQL Proxy

- Automate ACL generation
  - Parse DB, File access logs
  - Infer authentication rules
Conclusion

- Web authentication is broken
  - Semantic gap between Web App, DB & FS

- Nemesis infers safe authentication
  - When user input compared equal to password

- Nemesis enforces authorization
  - ACLs apply to authentication inferred user

- Validated using real-world PHP Apps
  - Prevented authentication & authorization bypass