



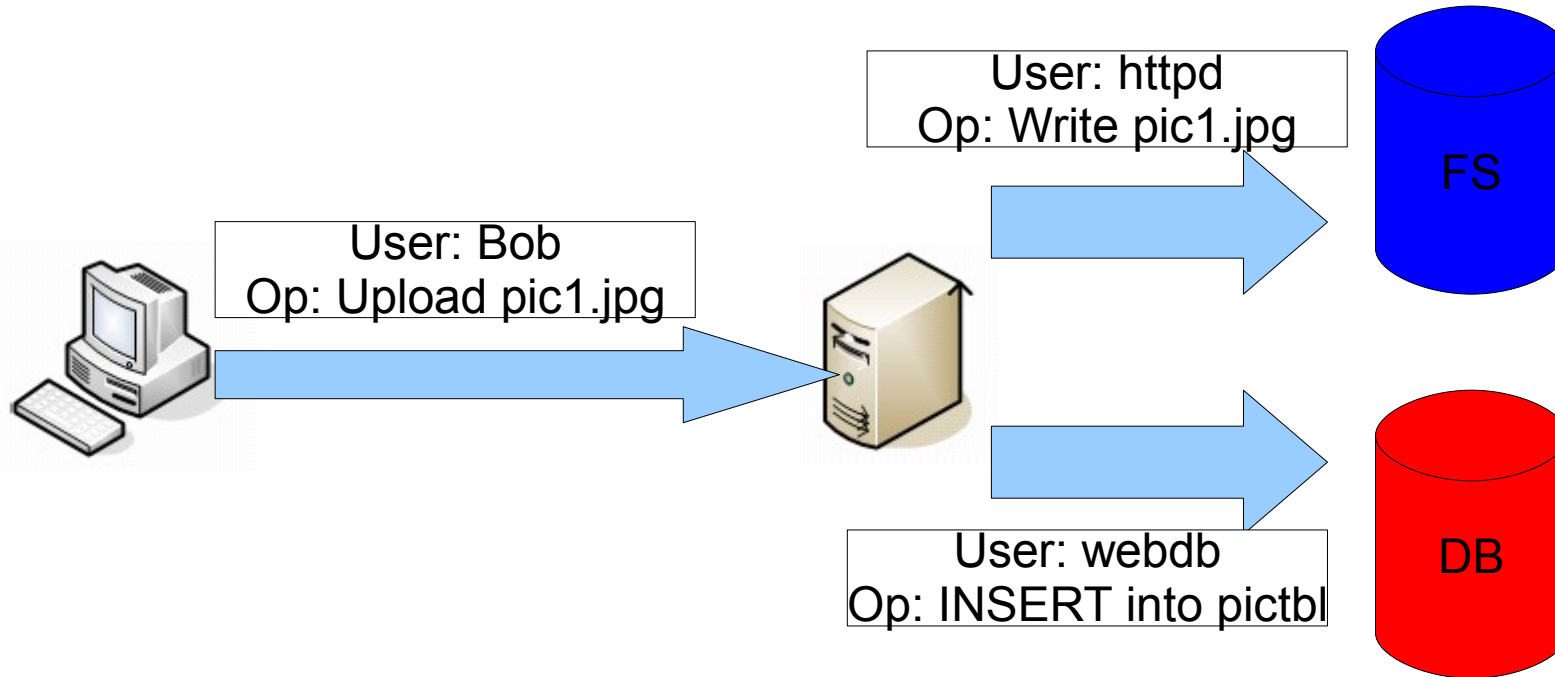
Nemesis: Preventing Web Authentication and Authorization Bypass Attacks

By

Michael Dalton



Web Application Overview





Security in Modern Web Apps

- ❑ Semantic gap between Web App, System Authorization

- ❑ FS, DB have no idea who web client is
 - Only see credentials of web application or web server!

- ❑ To prevent web client from breaking security...
 - Web application creates own authentication system
 - Web Application manually inserts access checks

```
if (client_authorized(c, fileName)
    openFile(fileName)
```



Why is this broken?

- ❑ Every web app rolls own authentication, session mgmt system
 - Bugs, bugs, bugs
 - No common frameworks to prevent attacks
 - URL/cookie validation, session hijacking, etc.

- ❑ Access checks required before all FS, DB ops
 - Miss an access check → application compromise
 - Incorrect check → application compromise

- ❑ Real-world vulns allowing total attacker control of webapp
 - Microsoft IIS, 3Com switches, Banks, lots of webapps...



Authentication Bypass Attacks

- ❑ User authenticated without valid credentials

- ❑ Many ways to mess this up:
 - Cookie, URL parameter validation (not validating pw)
 - Session management flaws
 - Weak cryptography
 - Weak, brute forceable passwords

- ❑ End result: Attacker has unrestricted access to webapp



Authentication Bypass Attack Examples

❑ URL Parameter validation (adapted from phpStat)

```
if (isset($_GET['admin']))  
    $userName = 'admin';
```

- www.vulnerable.com/index.php?admin=1

❑ Session Management (adapted from phpFastNews)

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

- Edit cookie, add name/value pair 'user=admin'



Authorization Bypass

- ❑ User obtains unauthorized access to resource
 - File
 - Database row or table

- ❑ Occurs due to missing or incorrect access control check

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



Nemesis: Stopping Auth Attacks

- ❑ Must **infer** when a user is authenticated safely
 - Instead of trusting (probably insecure) app authentication
 - Track passwords, web client input
 - **Safe authentication** if app compares client input to password

- ❑ Must **automatically enforce** web client ACLs
 - Use an ACL file specifying FS, DB access for web app users

- ❑ Must ensure **compatibility** with legacy apps
 - Crucial for adoption
 - Do not require code re-writes, auth system changes, etc.



Key Insight: Use Taint Tracking

- ❑ Track flow of **untrusted input, passwords**
 - All user input (`$_GET`, ...) has **untrusted** tag bit set
 - Passwords from DB have **password** tag bit set

- ❑ Add tags to each object (String, Integer, etc)

- ❑ Tag bits propagated on all common operations (transparent)
 - Assignment, Function call/return, String concatenation, etc

- ❑ Infer when app **safely authenticates** a user
 - Occurs when user input compared equal to password
 - Does not require knowledge of app auth framework



Authentication Inference Example

```
$user = $_GET['username']
```



Authentication Inference Example

`$user = $_GET['username']`



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```

```
$realpw = $db->query("SELECT pw FROM users WHERE "  
+ "userName = " + $safeuser + ";
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```

```
$realpw = $db->query("SELECT pw FROM users WHERE "  
+ "userName = " + $safeuser + ";
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```

```
$realpw = $db->query("SELECT pw FROM users WHERE "  
                    + "userName = " + $safeuser + ";
```

```
$md5pw = md5sum($pw)
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```

```
$realpw = $db->query("SELECT pw FROM users WHERE "  
                    + "userName = " + $safeuser + ";
```

```
$md5pw = md5sum($pw)
```



Authentication Inference Example

```
$user = $_GET['username']
```

```
$safeuser = mysql_real_escape_string($user)
```

```
$pw = $_GET['password']
```

```
$realpw = $db->query("SELECT pw FROM users WHERE "  
                    + "userName = " + $safeuser + " ;")
```

```
$md5pw = md5sum($pw)
```

```
if ($md5pw == $realpw) {
```

A blue, multi-pointed starburst shape with a black outline, containing the text 'Authenticated!' in black.

Authenticated!



Session Management

- ❑ Problem: how to record auth information for login session
 - Can't trust application authentication info
 - Each application has its own session management framework

- ❑ Solution: create cookie with Nemesis auth info
 - When Nemesis detects safe auth, create a fresh cookie
 - Store Nemesis credentials in cookie
 - Secure with HMAC, server-side key



Authorization Design

- ❑ Authorization bypass – missing or incorrect checks
 - So we apply access checks automatically
 - Apply ACLs using current Nemesis user as principal

- ❑ ACL design
 - Specify file, DB access for web application users
 - Manually generated by sysadmin for now

- ❑ Taint information required for real-world ACLs
 - Restrict based not just on resource, but how resource accessed
 - Webapps often allow restricted access to resource (read-only)
 - Ex: Ensure only trusted queries used to access resource



Nemesis Access Control

- ❑ Simple ACLs map users to allowed resources
 - Bob can access /web/foo.txt
 - Jim can read database table account_reports

- ❑ More complex ACLs must restrict rows
 - Requires rows to report the principal that created row
 - Can be done automatically by rewriting SQL queries

- ❑ Most complex ACLs typically require taint information
- ❑ Ex: Lost password –

```
UPDATE users SET password = $rand()  
WHERE username = 'bob'
```

 - Only safe because \$rand() is **untainted**



Stopping Attacks

- ❑ ACL checks applied to Nemesis user
 - Not web application user!

- ❑ Nemesis user authenticated only if they know pw
 - So authentication attacks don't count

- ❑ Authentication attack
 - Does not change Nemesis user
 - ACLs applied to correctly authenticated user only
 - No privilege escalation via authentication attack

- ❑ Authorization attack
 - Cannot bypass automatically enforced ACLs



Putting it all together..

❑ Admin supplies

- ACL
- Location of user authentication information (table, columns)

❑ Nemesis **infers** authentication using DIFT

- Tracks safely authenticated users
- Stores Nemesis auth information in separate cookie
- Does not require info or modifications to auth framework

❑ Nemesis **authenticates** file, DB accesses against ACL

- Use Nemesis authentication information for current user
- Ensure filesystem, DB ops comply with supplied ACL
- Attack detected when ACL violation occurs



Prototype

- ❑ Modified PHP interpreter to perform DIFT
 - Track taint across all types (String, integer, etc)
 - Leveraged existing open source PHP DIFT work

- ❑ Automatically authenticate on string comparison
 - If user input compared equal to password
 - Check for equality on both == and != operators

- ❑ Create own cookie/session variable for Nemesis user info

- ❑ Automatic SQL query rewriting future work



Experimental Results

| Application | Size (Lines) | Auth Lines Added | ACL Check Lines Added | Attack Prevented |
|----------------|--------------|------------------|-----------------------|---------------------------------|
| Php iCalendar | 13,500 | 3 | 22 | Auth Bypass |
| PhpStat | 12,700 | 3 | 17 | Missing ACL Check |
| Bilboblog | 2,000 | 3 | 11 | Incorrect ACL Check |
| phpFastNews | 500 | 5 | 17 | Auth Bypass |
| Linpha Gallery | 50,000 | 15 | 49 | SQL Injection in Password Check |
| DeluxeBB | 22,000 | 6 | 143 | Missing ACL Check |



Future Work

- ❑ ACL inference engine needed
 - Learn ACLs by monitoring DB, FS accesses
 - Statistical inference, Z-Ranking, etc.

- ❑ Intuitive GUI
 - Allow administrators to understand, edit ACLs

- ❑ SQL query parsing, rewriting



Be sure to read our USENIX Security 2009 paper
“Nemesis: Authentication & Access Control Vulnerabilities in
Web Applications”

Feedback welcome: mwdalton@cs.stanford.edu