SECURITY NUTS TO BOLTS

ILIA ALSHANETSKY
@ILIAA
HTTPS://JOIND.IN/19197
ME, MYSELF & I

• PHP Core Developer
• Author of Guide to PHP Security
• Security Aficionado
• CTO @ Gubagoo Inc. (we are hiring!)
THE CONUNDRUM

USABILITY  SECURITY

YOU CAN HAVE ONE ;-)
OPEN WEB APPLICATION SECURITY PROJECT

• A set of best practices and recommendations around making web applications more secure

• General database of common vulnerability vectors

• A good place to keep yourself up-to-date on security

Not a Bible™
INJECTION

Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something?

In a way -

Did you really name your son Robert?; DROP TABLE Students;-- ?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.

xkcd.com
WHAT NOT TO DO

// $_POST['login'] = "login";
$sql = "SELECT * from users WHERE login='{$_POST['login']}'
    AND password='{$_POST['pwd']}'";

// $_POST['login'] = "' OR 1=1; --";
$sql = "SELECT * from users WHERE login='".$pdo->quote($_POST['login'])."'
    AND password='".$pdo->quote($_POST['pwd'])."'";

// $_POST['login'] = chr(0xbf).chr(0x27)." OR 1=1; --";
// 0xbf27 + addslashes() == 0xbf5c27 == é½œ + "'"
$sql = "SELECT * from users WHERE
    login='".addslashes($_POST['login'])."'
    AND password='".addslashes($_POST['pwd'])."'";

$sql = "SELECT * from users WHERE
    login='".$pdo->quote($_POST['login'])."'
    AND password='".$pdo->quote($_POST['pwd'])."'";

http://hakipedia.com/index.php/SQL_Injection
PREVENT INJECTION

• For databases use prepared statements
• White list inputs whenever possible
• Sanitize inputs (use filter extension)
• Don’t trust and always verify!
BROKEN AUTHENTICATION & SESSION MANAGEMENT
MITIGATION

• Enforce strong password policy
• Require periodic reset of password
• Use 2 factor authentication
• Use SSL and secure flag on cookies
• Don’t forget about auto-logout
• Don’t neglect failed-login detection & tracking
SESSION SECURITY

• Don’t trust new session ids
  session_regenerate_id(true)
  session.use_strict_mode (5.5.2+)

• Use unique session names (not PHPSESSID)

• Only use httpOnly cookies

• Ensure true randomness for session ids
CROSS SITE SCRIPTING - XSS

Attacker posts content that contains embedded JavaScript that is stored by vulnerable application.

User visits the compromised site that transmits expected content along with injected script.

User's browser renders the page and executes injected script.

User's data is transmitted to the attacking party.
PROTECT YOURSELF

• Use filter extension to filter inputs
• Ensure that outputs are HTML encoded
• Don’t reinvent the wheel
• Don’t consider any part of the request as being “safe”
INSECURE DIRECT OBJECT REFERENCES
PREVENTION

• Low level access controls

• Prevent user input in file/URL access commands

• No unsanitized input to execution commands (escapeshellarg() for arguments)

• Non-white-list input shouldn’t dictate logic
SECURITY MISCONFIGURATION
MORE SPECIFICALLY

• Usage of default, un-secure settings
• Not disabling initial accounts (especially those with admin rights)
• Failure to apply latest security patches
• Leaving un-used functions/modules enabled
• Exposed error handling
• Keeping “upgrade” scripts in accessible directories
PREVENTION > CURE

- Perform periodic security checks using automated tools
  - STATIC CODE ANALYSIS
  - NMAP
  - EXTERNAL VULNERABILITY SCANNERS
    https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
    http://sectools.org/tag/web-scanners/
  - DISTRO PACKAGE SECURITY CHECKS
SENSITIVE DATA EXPOSURE

GREAT SUCCESS!
SOME EXAMPLES

- Exposed PHP error messages
- Non-web related files stored inside web-root
- Application version exposure
- Un-encrypted sensitive data storage
- Not using SSL
MISSING FUNCTION LEVEL ACCESS CONTROL

YOU SHALL NOT PASS
WTF??

- Valid input processing without access controls
- Reliance on hidden fields for record identifiers
- Decentralized access control layer
- JavaScript driven access controls
CROSS-SITE REQUEST FORGERY (CSRF)

Attacker tricks the user into following a link to a trusted site with vulnerable payload

User visits the compromised site that renders desired content along with compromised payload

User's data is transmitted to the attacking party
PREVENTION

• Don’t perform data changes on GET
• Use secure (csrf) tokens for POST
• Dynamic Field Names
USING COMPONENTS WITH KNOWN VULNERABILITIES

• Using old vulnerable software
• Not keeping libraries up-to-date
  *cough*OpenSSL*cough*
• Forgetting to update JavaScript libraries
THE CURE

• On server do routine checks for software with known exploits

• Keep libraries up-to-date

• Compare utilized software versions to those listed on http://cve.mitre.org/
UNVALIDATED REDIRECTS AND FORWARDS

• Header Injection
• JavaScript Parameter Injection
• Reliance on HTTP_REFERER
AUTHENTICATION
REQUIRE STRONG PASSWORDS

• Require password length of 8 characters

• Enforce Password Complexity (3 of 4 rules):
  • At least one upper-case letter
  • At least one lower-case letter
  • At least one number
  • At least one special (non-alphanumeric) character
BUT EVEN THAT IS WEAK...

- Rainbow Tables
- GPU optimized hash guessing
- AWS ;-(

The Linkedin list containing 5.8 million unique password hashes is now over 90% cracked. These are the top words users are basing their passwords on.
SECURE PASSWORD HASHES

$password = "@foo1Bar#";

$passwd = crypt($password, 
'$2y'. // BlowFish base
'$10$'. // cryptographic complexity
bin2hex(fread(fopen("/dev/random", "r"), 32)) // random bytes
.'$'
);

if ($passwd === crypt($password, substr($passwd, 0, 29))) {
  // password ok
} else {
  // password check failed
}

This will generate a password hash 60 bytes long
PHP 5.5+ MAKES THIS SIMPLER

```php
$hash = password_hash($password, PASSWORD_BCRYPT,
    ['cost' => 10]);

if (password_verify($password, $hash)) {
    // password ok
} else {
    // password check failed
}
```
WEB BRUTE FORCE ATTACKS

• Limit the number of sequential unsuccessful attempts to 3 - 5

• After that implement one or more of the following:
  • Lockout future attempts for 10-15 minutes
  • Require entry of CAPTCHA for all further attempts
  • Require multi-factor authentication
    • SMS if you have phone number
    • E-mail if you don’t
WEB BRUTE FORCE ATTACKS

• Implement blocks for multiple failed authentication attempts from the same IP address

• Don’t use the standard “login” and “password” form field names

• Re-authorize attempts when login is successful from an unknown IP address and/or Browser.

• If possible randomly generate the field names for authentication forms
<?php
// secret key for encoding form fields
$_SESSION['__form_key'] = $secret = bin2hex(openssl_random_pseudo_bytes(16));
?>
<form>
Login: <input type="text" name="<?= hash_hmac('md5', 'login', $secret); ?>" />
<br />
Password: <input type="password" name="<?= hash_hmac('md5', 'password', $secret); ?>" />
</form>
$secret = $_SESSION['__form_key'];
$input = array();

foreach ($field_names as $v) {
    $hashed_name = hash_hmac('md5', $v, $secret);
    if (isset($_POST[$hashed_name])) {
        $input[$v] = $_POST[$hashed_name];
    }
}
POST AUTHENTICATION PARANOIA

• Ensure Session Expiry Times are enforced at 24 - 30 mins
• Idle time logout after 10 mins of in-activity (JavaScript)
• For long-term session require re-authentication for key actions
  • Profile Changes
  • E-Commerce activities
• Prevent duplicate logins

http://www.erichynds.com/examples/jquery-idle-timeout/example-dialog.htm
CLICKJACKING

• Make sure you have X-Frame-Options header (with DENY or SAMEORIGIN) values

• Avoid GET method to make requests (yes, this includes Ajax)
TRANSPORT SECURITY

• Use HTTP-Strict-Transport-Policy to direct browser to use HTTPS
  • Does not work in IE, yet...
  • Redirect to separate sub-domain after HTTP > HTTPS redirect and restrict cookies to that domain.

Apache:
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"

Nginx:
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains";
SESSION  SECURITY
BASIC PROTECTIONS

- Only use cookies
  
  ```php
  ini_set("session.use_only_cookies", true);
  ```

- Ensure session ID integrity
  
  ```php
  ini_set("session.entropy_file", "/dev/urandom");
  ini_set("session.entropy_length", "32");
  ini_set("session.hash_bits_per_character", 6);
  ```

- Use HTTPOnly cookies for session storage
  
  ```php
  ini_set("session.cookie_httponly", true);
  ```

- Set Secure session bit (when using SSL/TLS)
  
  ```php
  ini_set("session.cookie_secure", true);
  ```
AVOID SESSION FIXATION

ini_set("session.name", "unique name");

session_start();

if (empty($_SESSION['__validated'])) {
    session_regenerate_id(true);
    $_SESSION['__validated'] = 1;
}
DATA ACCESS MANAGEMENT
TYPICAL SITUATION (PRE-MVC)
TYPICAL SITUATION
(POST-MVC)
IDEAL APPROACH

Controller

ACLs

Model
class DataModel {
    private $aclRules = array();

    public function __construct() {
        $this->aclRules['user_id'] = $_SESSION['user_id'];

        switch ($_SESSION['role']) {
            case 'admin':
                break;
            case 'user':
                $this->aclRules['public'] = 1;
                break;
            case 'editor':
                $this->aclRules['category'] = $_SESSION['category'];
                break;
        }
    }

    public function ActionName(array $params) {
        $input = array_replace_recursive($params, $this->aclRules);
        $this->runAction($input);
    }
}
AUDIT TRAIL

Leaving a paper trail is hard work.
Time for a nap.
WHY?

• Makes tracking down user activity easier when there is a security issue...

• All kinds of uses for debugging purposes

• Allows for pattern analysis for “unusual” activity detection

• Creates a “revert” path, versioning on the cheap
HOW?

• Should be done at the lowest level possible to avoid creating a possibility of un-audit-able actions.

• Inside a Model

• Inside Database (via triggers)
class DataModel {
    private function __save() {
        $current = $this->fetch($this->id);
        $changes = array_diff_assoc($this->input, $current);
        $this->pdo->beginTransaction();

        if (($return_val = parent::save())) {
            $this->log(array(
                'user_id' => $_SESSION['user_id'],
                'when' => microtime(1),
                'what' => get_class($this),
                'record' => $this->id,
                'changes' => serialize($changes)
            ));

            $this->pdo->commit();
        } else {
            $this->pdo->rollback();
        }
    }

    return $return_val;
}
“UNUSUAL” PATTERN ANALYSIS
WHAT DOES IT MEAN?

• The best application vulnerabilities are the ones no one knows about.

• But even those usually require some “trial & error” to get to

• Reviewing audit trails and access logs often can let you spot something “unusual” before even knowing what it is...
PATTERNS TO LOOK FOR

• Unusually high number of request per session

• Atypical access pattern (late at night, different browser/IP combinations)

• Frequent accesses to same page within very short span of time, especially so if it is a data modification page.
LOW (MODEL) LEVEL
INPUT VALIDATION
APPLICATION SHOULD VERIFY IT’S OWN INPUTS

• Even at a model level application should verify input for validity

KEEP CALM AND DON'T TRUST ANYONE
class DataModel {
    private $input_config = array(
        'active' => array(
            'filter' => FILTER_VALIDATE_BOOLEAN,
            'flags' => FILTER_REQUIRE_SCALAR),
        'login' => array(
            'filter' => FILTER_VALIDATE_REGEXP,
            'flags' => FILTER_REQUIRE_SCALAR,
            'options' => array('regexp' => '!^[A-Za-z0-9_]+$!'),
        'id' => array(
            'filter' => FILTER_VALIDATE_INT,
            'flags' => FILTER_REQUIRE_SCALAR,
            'options' => array('min_range' => 1)),
        'email' => array(
            'filter' => FILTER_VALIDATE_EMAIL,
            'flags' => FILTER_REQUIRE_SCALAR),
        'blog' => array(
            'filter' => FILTER_VALIDATE_URL,
            'flags' => FILTER_REQUIRE_SCALAR)
    );

    public function save() {
        if (!filter_var_array($this->input, $this->input_config)) {
            throw new validationException('Invalid input');
        }
        // proceed as normal
    }
}
REMOTE URL ACCESS
THINGS TO CONSIDER

• Whenever possible use the API URL sitting behind HTTPS

• Ensure that Peer and Domain verification is enabled

• If you are using cURL know what your settings mean...
$url = 'https://en.wikipedia.org/w/api.php ...';

$context = array(
    'ssl' => array(
        'verify_peer' => TRUE,
        'cafile'      => '/usr/share/ssl/cacert.pem',
        'verify_depth' => 5,
        'CN_match'    => 'en.wikipedia.org'
    ),
    'http' => array(
        'user_agent' => 'My App',
        'ignore_errors' => TRUE
    )
);

file_get_contents($url, NULL, stream_context_create($context));
WITH CURL

```php
$curlh = curl_init($url);

curl_setopt($curlh, CURLOPT_RETURNTRANSFER, TRUE);

curl_setopt($curlh, CURLOPT_CAINFO, '/usr/share/ssl/cert-bundle.crt');

$data = curl_exec($curlh);
```

- Do not set CURLOPT_SSL_VERIFYPEER to FALSE
- Do not set CURLOPT_SSL_VERIFYPEER to FALSE or 1
PHP ERROR HANDLING
HOW TO HANDLE THEM?

• Log all errors

• Logging should not have dependencies
  • Disk is a good target
  • So is syslog

• There are no “trivial” errors
DISABLE ERROR DISPLAY!

```php
ini_set("display_errors", false);
```
THANK YOU FOR LISTENING

HTTP://ILIA.WS
@ILIAA
HTTPS://JOIND.IN/19197