Part V

Session Management Security

Why Session Management Security?

- HTTP protocol doesn't offer session management
- web applications rely however on browsersessions, users and their data
- therefore they must implement their own session management
- PHP offers ext/session which is the basis of a session management that can be used by PHP applications
- securing it is the job of the application

Session ID

The session ID is a string that identifies a user session an the data contained in it

Secure Session ID Generation

- not suited as session id are
  - current timestamp (in seconds)
  - the user's IP address
  - sequential numbers
  - simple combinations/hashes of these values
- suited are combinations of
  - microseconds
  - random numbers
  - process IDs

Session ID Generation in PHP

- PHP generates a new session id from
  - the current timestamp in microseconds
  - the process ID, the IP address of the user
  - a random number from the LCG
- combination gets hashed
  - MD5 / SHA1 / ext/hash
- alphanumerical encoding (4-6 bit)

➡ PHP generated session id is considered safe

Session ID Transport

- PHP supports differen session id transports
  - in cookie
  - in query string
  - in form field
- preferred is transport by cookie
- session id in query string or form field is more complicated to handle
- session id in query string leaks through referrer

Session ID Transport - Cookie Security

- session name
  - to stop applications to influence each other
  - session_name('myApplicationX');
- httpOnly cookies
  - to stop JavaScript from accessing the cookie
  - ini_set('session.cookie_httponly', true);
- secure Flag important for SSL sites
  - to stop cookie from leaking on port 80
  - ini_set('session.cookie_secure', true);

Session Lifetime

- users want to stay logged in forever
- from a security point of view sessions should be deactivated after some inactive idle time
  - ini_set("session.gc_maxlifetime", 60*15); // 15 minutes
- by changing the cookie parameters the maximum session lifetime can be controlled
  - ini_set("session.cookie_lifetime", 60*15); // 15 minutes
  - ini_set("session.cookie_lifetime", 0); // until browser is closed

Permissive vs. strict Session-Systems

- Permissive session-systems
  - accept arbitrary session ids
  - only refuses session ids containing illegal characters
  - creates a new session, if none exists with the choosen id
- strict session-systems
  - accept only session id created by themself
  - will refuse a session id if it does not belong to a started session

➡ PHP session management is permissive

Strict Session System in PHP

```
session_start();

// Accept only sessions with strict flag
if (!isset($_SESSION[`strict`])) {

    // Generate new session id
    session_regenerate_id();

    // set strict flag
    $_SESSION = array(`strict` => true);
}
```

Session Storage

- PHP saves sessions serialized
- PHP supports differen session storage modules
    - session.save_handler - storage-module: files, mm, user, sqlite
    - session.save_path - configuration of storage-module
- Default-configuration
    - session.save_handler - files
    - session.save_path - /tmp

Session Storage - Data Mixup (I)

- Default */tmp* often not changed
- all applications share session data
- very bad in shared hosting situations
- can lead to inter-application-exploits

Session Storage - Data Mixup (II)

- Example 1 - Setup
  - Customer runs two applications on own server
  - both applications consist of multi-step forms
  - both application store previous steps in the session
  - application 1 copies all user input in the session - validation/ filtering occurs after the last step
  - application 2 copies only validated and filtered data into a session

Session Storage - Data Mixup (III)

- Example 1 - Exploit
  - Attacker enters malicious data into application 1
  - Attacker copies session id from cookie of application 1 into the cookie of application 2
  - Attacker uses application 2 that trusts blindly the unfiltered data that was stored by application 1 in the session

➡ unfiltered malicious data from application 1 results in a security problem in application 2

Session Storage - Data Mixup (IV)

- Example 2 - Setup
  - Customer runs two applications on his own server
  - both applications are for separate user groups
  - both applications are developed by the same developers
  - both applications share parts of their implementation

Session Storage - Data Mixup (V)

- Example 2 - Exploit
  - Attacker is a user of application 1 (maybe even a moderator / admin)
  - Attacker logs into application 1
  - Attacker copies session if from the cookie of application 1 into the cookie of application 2
  - because both applications share the implementation of the user object, application 2 finds a compatible, valid and logged in user object in the session
  - Attacker is logged into application 2

Session Storage - Data Mixup Prevention

- store session data always in separate places
  - ini_set('session.save_path', '/tmp/application_1');
  - userspace session storage module
- add application marker to session
- encrypt session data

Session Storage - Applicationmarker

```
session_start();

if (!isset($_SESSION[‘application‘])
    || ((string)$ SESSION[‘application‘] !== ‘application 1‘)) {
  session_regenerate_id();
  $_SESSION = array(‘application‘ => ‘application_1‘);
}
```

Session Storage - Userspace Session Storage

- PHP supports userspace session storage
  - set_session_save_handler("o","r","w","c","d","g");
- six functions must be implemented
  - `open    - storage module init`
  - `read    - loading session data`
  - `write   - storing session data`
  - `close   - storage module shutdown`
  - `destroy - delete a session`
  - `gc      - garbage collector`

Session Storage - Insecure Transactions (I)

- usual implementation
  - `open` - `gets ignored`
  - `read` - **`SELECT * FROM tb_sess WHERE sid=?`**
  - `write` - **`INSERT/UPDATE tb_sess SET data=? WHERE sid=?`**
  - `close` - `gets ignored`
  - `destroy` - `gets ignored`
  - `gc` - `gets ignored`

Session Storage - Insecure Transactions (II)

- common implementations ignore that reading, modifying and writing back the session data is a transaction
- most userspace session storage handlers are vulnerable to race conditions

Session Hijacking

- Attacker retrieved the session id of a user and takes over the session
- possible take over paths
  - sniffing HTTP connections
  - leak of session id in query string through referer
  - XSS

Session Hijacking - Countermeasures

- do not transport session id in query string
- mark session id cookie as httpOnly
- use SSL and mark session cookies as secure
- add additional safeguards: one time URL tokens

Session Hijacking - One Time URL Tokens

- all links must include the one time URL token
- current valid URL tokens must be stored in session
- used one time URL tokens are deleted from session
- requests without valid one time tokens are ignored
- session hijacking becomes more work because one time URL tokens must be retrieved, too

Session Fixation

- Attacker forces the victim to surf with a session id chosen by the attacker
- possible attack vectors
  - session id in query string
  - cookie infection
- because session id is known there is no need to guess or steal it

Session Fixation - Invalid Countermeasures

- bind session to content of HTTP headers
  - ➡ session fixation becomes only minimally harder
  - ➡ browser compatibility problem
- bind session to user's IP address
  - ➡ leads to problems with big ISPs with changing IP addresses
  - ➡ doesn't protect against attacks from the LAN / same route
  - ➡ but works against attacks from the outside

Session Fixation - Working Countermeasures

- Changing the session id after each change in status
  - ➡ session_regenerate_id() + session_destroy()
  - ➡ stops abuse of fixated sessions
- Re-authentication before sensitive actions
  - ➡ requesting the password
  - ➡ stops fixation because valid requests require user's password

Questions ?