Hacking Web Sites
Broken Access Control

November 24, 2023

Emmanuel Benoist | BFH-TI

Table of Contents

Introduction

Broken Access Control

- **Access control should**
  - force users to act inside the intended permissions
- **Failures lead to**
  - unauthorized information disclosure
  - modification or destruction of data
  - performing a business function outside the limits of the user
- **Missing function level access control:**
  - access pages without authorization.
- **Insecure direct object references:**
  - access data directly.
- **Path traversal**
  - Access to files outside of the limits

Introduction

- **Possibility to access pages without necessary privileges**
- **Vertical escalation**
  - Anonymous users access private functionalities
  - Regular users access administrator functionalities
- **Horizontal escalation**
  - Users access data of other users
- **Possibility to access resources without required privileges**
  - Access Files intended for other users
  - Access directories and files outside the scope of the web server.

Insecure direct object references

- **Occurs when developer uses HTTP parameter to refer to internal object**
  - For instance `http://mysite.com/program.php?lang=fr`
  - And in the program:

    ```
    require_once($_REQUEST['lang']."lang.php");
    ```
- **Can also access to identifiers**
  - For instance `http://mysite.com/program.php?page=124`
  - It may be possible to change the page ID. The rights to see the page have to be tested.
- **Which objects are subject to attacks?**
  - Files: for upload and/or for reading, or accessing
  - Identifiers : for showing them, or changing them

Principles

Presentation of the Vulnerability

- **Developer does not test if a user is legitim for accessing a page**
  - Attackers just access the page / the resource
  - Just need to know (or guess) the URL or URL parameters
  - can access to data reserved for another user (privilege escalation)
- **Developer exposes a reference to an internal implementation object, as URL or form parameter**
  - A file
  - Directory
  - Database record
  - Key
- **The attacker manipulates one of the parameter and accesses internal resources.**
  - uses a lack in access control check

Expose internal object references

- **Applications expose their internal object references to users**
  - Attacker use parameter tampering to change references
  - they can violate security policy if it is unenforced
- **Example**
  - If the application uses a parameter which contains a filename or a path
  - It can be changed to access other resources

viewpage.php?document=mydoc

displays the content of the file `/home/bie1/myfiles/mydoc.pdf`
the input could be manipulated into accessing another file

viewpage.php?document=../../frc1/otherDocument

will show the file `/home /frc1/otherDocument.pdf`

- `Also known as : Path Traversal`

Tampering HTTP parameters

Which parameters can be tainted?

- **HTTP GET parameters**
  - Directly inside the URL
  - Example: `http://www.mysite.com/index.php?viewaccount=23456`
  - Change the URL : `http://www.mysite.com/index.php?viewaccount=1234`
  - Gives access to the account `1234`
- **HTTP POST parameters**
  - In the body of the POST request,
  - Often URL encoded (other encoding are possible).
- **Other HTTP headers**
  - The cookies (are included in each HTTP header)
  - Languages (in the header: Accept-Language)
  - User-Agent (to determine which browser is reading the page)
- **API parameters**
  - Access API without access control for POST, PUT and DELETE

Internal objects?

- **Data Base objects**
  - Reference to records in a table
  - Primary key used to refere to a page
  - `www.vicitim.com/page?pageId=345` to access page with Id 345
- **File (or directory)**
  - Reference to a file is done in the parameter
  - The file may be loaded

```
include($_GET['language'].'.php');
```

  - Or it may be moved, copied, etc.
- **Keys**
  - a key may be shown as a parameter (or cookie).

Tampering manually

- **HTTP is an open protocol**
  - Requests can be generated manually
  - Parameters can be set arbitrary
- **GET Request**
  - Insert parameters in the Query string
  - Parameters need to be URL encoded

```
GET /program.php?param=fake&param2=new+text%00 HTTP/1.1
Host: www.vicitim.org
```

- **POST Request**
  - Parameters are in the body of the request

```
POST /program.php HTTP/1.1
Host: www.victim.org
Content-Length: 52
Content-Type: application/x-www-form-urlencoded

username=bie1&email=bie1@bfh.org&submit=Submit+me%21
```

Tampering with cURL and Wget

- **HTTP is deprecated**
  - Requests are now always Secure
  - Need a TLS connection for HTTPS
- **Use cURL or Wget to generate GET requests**
  - cURL : output on stdout
  - Wget : downloads a new file

```
$ curl https://www.benoist.ch/softSec
$ wget https://www.benoist.ch/softSec
```

- **POST Request**
  - Parameters are in the body of the request

```
$ wget --post-data 'username=bie1&email=bie1@bfh.org' \
https://www.victim.org/program.php
$ curl -X POST -d 'name=linuxize' -d 'email=linuxize@example.com' \
https://example.com/contact.php
```

Tampering inside a browser

- **Some web applications generate complicated requests**
  - Cookies, Url referer,
  - AJAX requests
  - JSON requests
- **Not so easy to generate manually**
- **Use tools to manipulate Requests generated by the browser**
  - **ZAP Zed Attack Proxy**
    OWASP tool
    Proxy of the browser,
    intercepts requests that can be manipulated (and much more)
- **Browser - Web Developper mode**
  - Allows to manipulate GET and POST requests.

Examples

Example

- **View the account of a client**
  - Suppose we have the following html in the menu of a client
  - The client can see each of his or her accounts

```
<div class="menu">
  <div><a href="/index.php?account=23456}">account 23456</a></div>
  <div><a href="/index.php?account=23332}">account 23332</a></div>
  <div><a href="/index.php?account=12231}">account 12231</a></div>
</div>
```

- **When the client clicks on the link:**

```
GET https://www.mybank.com/index.php?account=23456 HTTP/1.1
Host: www.mybank.com
...
```

Example (Cont.)

- **What appends if he replaces** `23456` **with** `121212`**?**
  - It may display the required account: If authorization is not checked.

Another Example

- **Access secret content**
- **Suppose you have a JavaScript application with the following request for a JSON object**

```
GET /resource?item=12345 HTTP/1.1
Host: www.mysite.com
Cookie: SESSIONID=239e98d32c98b23a
....
```

The application will respond with the following kind of answer:

```
{ id: 12345,
  name: 'benoist',
  firstname: 'emmanuel',
  accountnumber: '1234543245900',
  balance: '2090',
  currency: 'CHF'}
```

Another Example (Cont.)

- **What if we send the following?**

```
GET /resource?item=11111 HTTP/1.1
Host: www.mysite.com
Cookie: SESSIONID=239e98d32c98b23a
....
```

- **It may not be protected**
  - maybe the application just verifies that the user is loged in.

Example: Upload form

- **Suppose we found the following upload form**

```
<form action="upload.php" method="post" enctype="multipart/form–data">
    Select image to upload:
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
    <input type="hidden" name="homedir" value="uploaded/">
</form>
```

File to upload is copied into the directory `uploaded`

Example: Upload form (Cont.)

- **We may upload a file anywhere** Replace `uploaded` with :
  - `otherDirectory` (just to test)
  - `/var/www/htdocs/` if you have a Ubuntu server
  - `/var/www/htdocs/anysubdirectory/with777` Because user can write in this directory
- **Limitation for upload**
  - Only where the user has the right to write.
  - Very interesting in the directory accessible throw the web server.

Example: Angular

- **Programs verifies the role client side**
  - Done in Angular : Roles
  - Access Data throw an API

Example: Angular (Cont.)

- **Example of code (w3schools.com)**[1]

```
 <div ng–app="myApp" ng–controller="customersCtrl">
<table>
  <tr ng–repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
    $http.get("customers_mysql.php")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

[1] https://www.w3schools.com/angular/angular_sql.asp

Example: Angular (Cont.)

- **No verification is done on the server.**

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");
$result = $conn->query("SELECT CompanyName, City, Country FROM Customers");
$outp = "";
while($rs = $result->fetch_array(MYSQLI_ASSOC)) {
    if ($outp != "") {$outp .= ",";}
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
    $outp .= '"City":"'   . $rs["City"]        . '",';
    $outp .= '"Country":"'. $rs["Country"]     . '"}';
}
$outp ='{"records":['.$outp.']}';
$conn->close();
echo($outp);
?>
```

Example: Angular (Cont.)

- **Problem 1:The user (and/or roles) are not known on the server**
  - Authentication MUST take place server side.
  - Roles MUST be stored server side in a session.
  - Access to the API is restricted to logged-in users.
- **Problem 2: Each access to the Data Base must verify that the user is allowed to access data**
  - SQL must verify that user has access to data.

Possible also for modern client side applications

- **Ajax or JSON functions are called from within the JS application**
  - It is your code
  - Requests are generated in JavaScript
  - Are sent to the server
- **Ajax/JSON Functions**
  - Can be administration level
  - URL's (server side) may not verify the rights
  - A user could force the browser to send requests
  - And access functions without the required privileges

Examples of Attacks

Access tax declaration

- **Account number is often the primary key**
  - Tempting to use the key direct in the web interface
- **Developers have used paramtrized SQL to prevent Injection**
- **If no other check is done, Attacker could access all accounts**
  - Replacing his account with the one of the victim
- **This attack was conducted in Australian Taxation Office's GST Start Up Assistance**
  - Attacker visited the details of 17'000 companies
  - And sent an email to each of them.

Modify internal keys

- **Modify Database Key**
  - If the attacker receives a URL:
    `http://www.attacked.com/resource.php?messageID=123`
  - It is tempting to try if the next user exists:
    `http://www.attacked.com/resource.php?messageID=124`
- **Need to know some internal IDs**
  - Can be brute forced
  - Can be found in other pages : View source

Read files

- **Accessing a file**
- **File directly accessible**
  - Suppose you have the following URL
  - `www.victim.org/resources/BIE1.pdf`
  - You know that `BIE1` is your ID in the system.
  - You know your boss's ID is `DUE1`,
  - `www.victim.org/resources/DUE1.pdf`
  - more easy if `www.victim.org/resources/` shows the index of the directory

Read files (Cont.)

- **Use a program to read a file**
  - Using commands like:

```
require(...);
include(...);
fopen(...);
file_get_contents(...);
```

  - Works with files `/usr/lib/www/test.txt`
  - Works also sometimes (depending on configuration) with URL's `http://192.168.1.25/secretFile.txt`

Read file

- **Program**

```
<?php
if(isset($_SESSION['userID'])){
  $homepage = file_get_contents($_GET['file']);
   echo $homepage;
}
?>
```

- **Works with the following URL** [2]**:**
  `http://www.victim.org/file=bie1.pdf`
- **But also with** `http://www.victim.org/file=/etc/passwd`
- **Could also work with**
  `http://www.victim.org/file=http://192.168.1.24/restrictedResource`

[2] all requests parameters should be URL encoded

Access to admin page

- **Admin page is reserved for administrators**
  - The page itself just tests if a session is started
  - The link to this page is only presented to users with admin status.
  - The page does not test if the user is admin (since normally only admins can see the link)
- **The attacker will access the admin page**
  - Suppose the attacker has a normal user account
  - He/she can see the program structure:
    where are the programs
    what is the directory structure
    which suffix is used for programs
  - Tests all the possible admin pages:
    `/admin.php`,`/admininstrator.php`, `/admin/`, `/root.php`, ...
  - Once the url is found: access is granted

Vulnerability

Threat agents

- **Anyone with network access**
  - Can send your application a request
- **Anonymous users may access restricted parts of the application**
  - Vertical escalation
- **Regular users use privileged functions**
  - Vertical escalation
- **Regular users access data from other users**
  - Horizontal escalation

Security Weakness

- **Prevalence: Average**
- **Applications do not always protect application functions properly.**
  - System may be misconfigured
  - Code is needed on each page: developers forget it
  - Code is needed by every access to internal data.
- **Detectability: Average**
  - Detection is easy
  - Hardest part: identifying URLs
- **For testing**
  - Ask the client for different level accounts: save a lot of time
  - Or use automatic testing using rules with default set of possible names

Protection

Authorizations

- **Verify authorization to all pages**
  - User must be loged-in
  - User must have the right privileges to access that page
- **Verify authorization to all referenced objects**
  - Verify in the SQL that the person is authorized:

```
$query = "select * from guestbook, user where guestbookID=$number ";
$query .= "AND guestbook.author=user.userID";
```

To restrict the access only to the destinatairs of the file, we should add:

```
$query .= "AND guestbook.dest=$_SESSION[userid]";
```

Avoid Direct Object References

- **Avoid exposing direct object references to user**
  - DB Primary Keys
  - Or filenames
- **Validate any private object references**
  - *“Accept known good”* approach

Protection against malicious file execution

- **Careful Planning**
  - Desigining architecture
  - Designing the program
  - Testing the program
- **A well written application does not use user-supplied input for**
  - Accessing server based resource
  - Images
  - XML and XSLT
  - Scripts
- **Application should have firewall rules preventing**
  - new outbound connections the the internet
  - or internally back to any other server
- **However, legacy applications may need to accept user supplied input**

Protection

- **Strongly validate user input**
  - use “accept known good” as a strategy
- **Add firewall rules**
  - Prevents your server to connect other web sites
  - or internal systems
- **Check user supplied files and filenames**
  - and also: tainting data in session object, avatars and images
  - PDF reports, temporary files, etc.
- **Considere implementing a chroot jail**
  - or other sandbox mechanisms to isolate applications from each other
  - Example: Virtualization

Use explicit taint checking mechanisms

- **If included in language**
  - JSF or Struts
- **Otherwise, consider a variable naming scheme**

```
$hostile = &$_POST;
$safe['filename'] = validate_file_name($hostile['unsafe_filename']);
```

- **So any operation based upon hostile input is immediately obvious:**

```
// Bad:
require_once($_POST['unsafe_filename'].'inc.php');
// Good:
require_once($safe['filename'].'inc.php');
```

Indirect object reference map

- **Where a parital filename was used, prefere a hash of the partial reference**
- **Instead of**

```
<select name="language">
  <option value="english">English</option>
```

- **Use**

```
<select name="language">
  <option value="2c8283b7743646a2a72e626437484">
     English
  </option>
```

- **Alternatively, use 1, 2, 3 as array reference**
  - check array bounds to detect parameter tampering

Indirect object reference map (Cont.)

- **Example of code**

```
<?php
include('pdo.inc.php');
try {
    $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username, $passwor
    echo '<h1>List of patients</h1>';
    $sql = "select * from patient";
    $result = $dbh->query($sql);
    while($line = $result->fetch()){
      echo "<a href='patients3.php?id=".$line['patientID']."'>";
      echo $line['first_name']." ".$line['name'];
      echo "</a><br>\n";
    }
    $dbh = null;
}
catch(PDOException $e)
{    echo $e->getMessage(); }
?>
```

Indirect object reference map (Cont.)

- **Generates something like:**

```
<a href='viewPatient.php?id=1'>Hugh Laurie </a><br>
<a href='viewPatient.php?id=2'>Jesse Spencer </a><br>
<a href='viewPatient.php?id=3'>Peter Jacobson </a><br>
<a href='viewPatient.php?id=4'>Lisa Edelstein </a><br>
```

- **Direct access to the Database IDs**
  - Can be easy manipulated,
  - need to be checked on the server side.

Indirect object reference map (Cont.)

- **Better:**

```
while($line = $result->fetch()){
  $rand_number = ...;
  $_SESSION['map'][$rand_number]=$line['patientID'];
  echo "<a href='patients3.php?id=".$rand_number."'>";
  echo $line['first_name']." ".$line['name'];
  echo "</a><br>\n";
 }
```

- **Modification in** `patients3.php`

```
if(isset($_GET['id'])){
  $patientID = $_SESSION['map'][ (int)($_GET['id'])];
}
```

Conclusion

Conclusion I

- **OWASP Top 10 - A1:2021**
  - Broken Access Control
  - most important web security risk in 2021
- **Dangerous**
  - Easy to exploit
  - Flow is quite common
  - Business impact is proportional to the functions that can be discovered.

Conclusion II

- **Web Parameters are easilly spoofed**
  - GET can be manipulated in the URL string
  - POST needs more sophisticated tool but is very easy too
- **Giving access to internal resource**
  - Allows modification, and illegal access
  - But gives also usefull information about your site (even if access is prohibited)
- **Should include authorization check in every page and resource**
  - Protects against privilege escalation (horizontal and vertical).
- **Hard to protect the access to internal resources**
  - Even harder in Client Side applications (JSON / AJAX).

References

- **OWASP Top 10 - 2013**
  `http://www.owasp.org/index.php/Top_10_2013`
  It has been merged back into *Borken Access Control* in OWASP Top 10 - 2017
- **OWASP Top10 2013**
  `https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control`
- **A Guide for Building Secure Web Applications and Web Services**
  `http://www.lulu.com/content/1401012`