Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hacking Web Sites Insecure Direct Object Reference

Emmanuel Benoist

Fall Term 2021/2022

Computer Science Division

Table of Contents

Introduction

Insecure Direct Object Reference

- **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

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)

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).

Principles

Presentation of the Vulnerability

Insecure Direct Object Reference

- **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 access internal resources.**
- **He uses a lack in access control check**

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 class="menu-item"><a href="/index.php?↘
  →account=23456}">account 23456</a></div>
  <div class="menu-item"><a href="/index.php?↘
  →account=23332}">account 23332</a></div>
  <div class="menu-item"><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.

Tampering HTTP parameters

Tampering without browser

- **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 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)
  - **Tamper Data**
    Firefox plugin
    Modify the requests inside the browser
- **Browser - Web Developper mode**
  - Allows to manipulate GET and POST requests.

Vulnerability

Vulnerability?

- **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`

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**
  - In 2000
  - 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(...);
```

  - Work with files `/usr/lib/www/test.txt`
  - But also 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** [1]**:**

  `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`

[1] all requests parameters should be URL encoded

Protection

How to protect yourself?

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

Authorizations

- **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]"↘
→;
```

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

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');
```

Protection (Cont.)

- **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

Conclusion

Conclusion

- **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)

Conclusion (Cont.)

- **Malicious file execution occures when**
  - files can be uploaded
  - Reference for the file (or stream) is based on user input
  - Include can use distant files
- **Malicious file execution is particularly dangerous**
  - When there is no “sandbox”
  - When infected machine can access to resources on the internet (php scripts for instance)
  - Or inside the intranet (SMB for instance)

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
- **A Guide for Building Secure Web Applications and Web Services**
  `http://www.lulu.com/content/1401012`