# Results

My project was a network attack visualisation tool that simulates various network attacks like SYN flooding, DNS tunneling and Port scanning. The final product of this project allows users to observe and interact with simulated traffic attacks (See Appendix 1 subheadings).

I used modern  Javascript libraries like Recharts to build a dynamic line chart to showcase the network's traffic during SYN flood (refer to Appendix 1 - SYN Flood) and Xarrows to animate directional flow between phases of attack like DNS tunneling and Port Scan (refer to Append 1 - DNS tunneling and Port Scan). Some outputs I was most proud of were the interactive features of SYN Flood **(Appendix 1 A2)**,DNS tunneling **(Appendix 1 A4)** and Port Scan **(Appendix 1 A6)**.

In the SYN Flood simulation (**Appendix A2)**, I implemented an interaction that mimics the experience of registering on a website under high traffic conditions. When the simulated SYN Flood is active, the registration process becomes noticeably slower, representing the server's overwhelmed state caused by numerous half open TCP connections. This simple interaction helps users visually and practically understand how SYN Floods exploit the TCP handshake process to degrade server performance.

In the DNS Tunneling simulation (**Appendix A4)**, I designed the interactivity to reflect both the attacker's and victim's perspectives, similarly to the approach used in the SYN Flood simulation. On the attacker's side, users can input a malicious domain, which is then registered and handled by a trusted DNS server, successfully bypassing a simulated firewall. When the victim inputs this malicious domain, the attacker can comprise the victim's system and utilise commands to exfiltrate data. From the victim's side, the user experiences no immediate feedback or indication of compromise reflecting the stealthy nature of DNS tunneling.

In the Port Scan simulation (**Appendix A6)**, , I implemented a more straightforward interaction compared to the other simulations. Users simulate scanning a set of predefined ports and observe their responses, whether they are open, closed, or filtered. I also included follow up interactions with open ports that allows users to simulate basic exploitation on these open ports. While simple, they fulfill my goal of helping users connect the idea of open ports to the types of vulnerabilities and data exposure that attackers often seek during the probing phase of a cyber attack.

These 3 interactions were the best features inside my technical project since despite being so simple, it effectively conveys the severity of network attacks. Overall, I believe that my project has adequately taught all kinds of users about these 3 different types of network attacks.

# What I did

Before building the codebase for the project, I've researched common network attacks, specifically SYN floods, DNS tunneling and Port scanning and identifying how to visualise these attacks to someone with no prior networking background. I managed my project across five weeks, starting with research and site planning in Week 4. Week 5 focused on structuring the site and writing the attack content, while Week 6 was spent developing interactive simulations. In Weeks 7 and 8, I completed the report, video, and final refinements.

I spent around 5 hours summarising the information and mapping out what my pages would look like and what visualisation I would go with. All my quick notes and information that I was planning to add to my page was documented inside my github repository (**Appendix 1 A7).** My research was culminated and simplified into the yellow section seen in Appendix 1 A1, A3 and A5. This component alongside the very basic implementation and outline of my network attack pages were initially done first  (seen in my github repository commits, commits were a bit lazy). My entire codebase didn't follow the best practices such as not using reusable components and having deeply nested code, however, it does get the job done.

When creating this project, I've encountered several technical issues. Within the SYN Flood page, I had trouble syncing up the traffic graph and the actual SYN Flood packets occurring. Since I didn't rapidly send multiple packets at once, I wasn't able to accurately sync the packets being sent with the traffic graph. So my best alternative was to just manually test out different ranges of numbers to find the closest resemblance. The most challenging issue was inside the DNS tunneling page where the progressive animation was used. My design for this page was that I had 6 phases which contained each step of a DNS tunneling attack. Then I would go through each phase and then annotate each phase. The difficult component was the transitioning part where I had to keep track of the index to use for displaying the arrow and the messages. I initially consulted websites such as stack overflow and youtube videos, however, I couldn't exactly find what I required, so I resorted to using AI.  ChatGPT initially gave me the codebase (**Appendix 1 A8**) for the progressive animation. With this as the main structure, I've added hovering mechanisms, arrows and labels on each arrows to further reinforce the actual animation.

Overall, I was able to finish my technical project in weeks 6 -7. There were moments where I had to do some shortcutting with my codebase (i.e messy, redundant code) so that I was able to finish on time and start on the writing component. Even when I faced setbacks, I was able to adapt by researching documentations and seeking out alternative solutions. While my implementation could be cleaner and more optimised in terms of code structure, the final result still successfully delivered an interactive and educational experience.

# How I was challenged

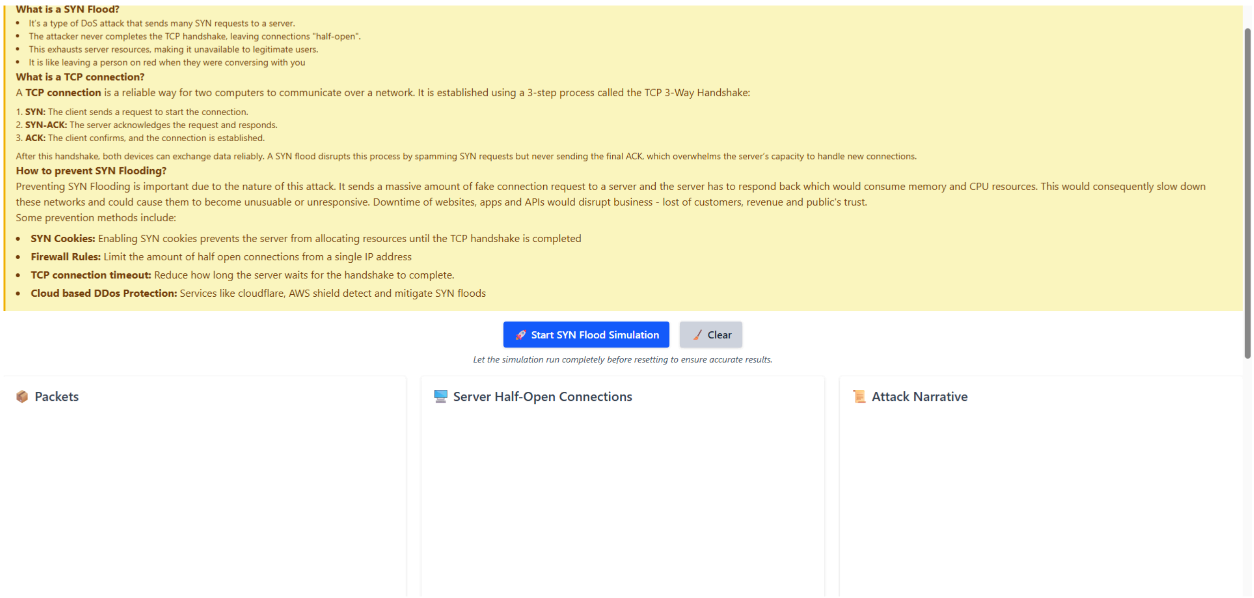
One of the biggest challenges was simplifying complex network attacks so that they were understandable and accessible to all users, especially those without a programming background. This required me to thoroughly learn how SYN Floods, DNS tunneling and Port Scanning work, not just conceptually but deep enough so that I can visualise their interactions.

From this project, I've learnt to prioritise functionality over visuals and that engagement is vital to a simulation. Having functionality and engaging activities assist users in learning the actual content (an important component of UI/UX design) while the visuals would reinforce the functionality and interactions making it more pleasing to look at and navigate.  Initially, I planned to include five network attacks, but I underestimated the time needed for building clean interactivity. I spent too much time refining visuals early on, which limited the number of attacks I could finish. Next time, perhaps I would simplify my codebase a bit more - focusing less on the visual so I can incorporate more network attacks so there would be more diversity and more exposure to all types of network attacks.

Overall, this project deepened my knowledge of these network attacks. It helped me become more confident in understanding network attacks, methods to prepare for these attacks both at home and in a professional environment and consequences of these attacks. In addition to this, my frontend skills were further honed with all the new React Libraries I've used. The main takeback from this assignment is that sometimes "good enough" is better than perfection as good quality with good quantity is better than perfect quality with bad quantity when it comes to simplifying multiple network attack concepts.

# Appendix 1

## A1



**What is a SYN Flood?**

- It's a type of DoS attack that sends many SYN requests to a server.
- The attacker never completes the TCP handshake, leaving connections "half-open".
- This exhausts server resources, making it unavailable to legitimate users.
- It is like leaving a person on red when they were conversing with you

**What is a TCP connection?**

A **TCP connection** is a reliable way for two computers to communicate over a network. It is established using a 3-step process called the TCP 3-Way Handshake:

1. **SYN:** The client sends a request to start the connection.
2. **SYN-ACK:** The server acknowledges the request and responds.
3. **ACK:** The client confirms, and the connection is established.

After this handshake, both devices can exchange data reliably. A SYN flood disrupts this process by spamming SYN requests but never sending the final ACK, which overwhelms the server's capacity to handle new connections.

**How to prevent SYN Flooding?**

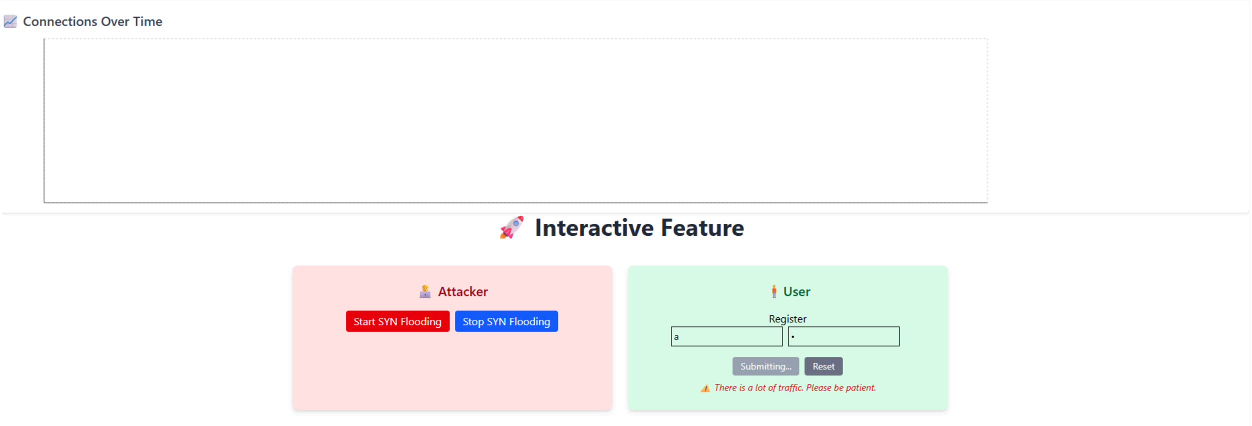
Preventing SYN Flooding is important due to the nature of this attack. It sends a massive amount of fake connection request to a server and the server has to respond back which would consume memory and CPU resources. This would consequently slow down these networks and could cause them to become unusuable or unresponsive. Downtime of websites, apps and APIs would disrupt business - lost of customers, revenue and public's trust.

Some prevention methods include:

- **SYN Cookies:** Enabling SYN cookies prevents the server from allocating resources until the TCP handshake is completed
- **Firewall Rules:** Limit the amount of half open connections from a single IP address
- **TCP connection timeout:** Reduce how long the server waits for the handshake to complete.
- **Cloud based DDos Protection:** Services like cloudflare, AWS shield detect and mitigate SYN floods

*Let the simulation run completely before resetting to ensure accurate results.*

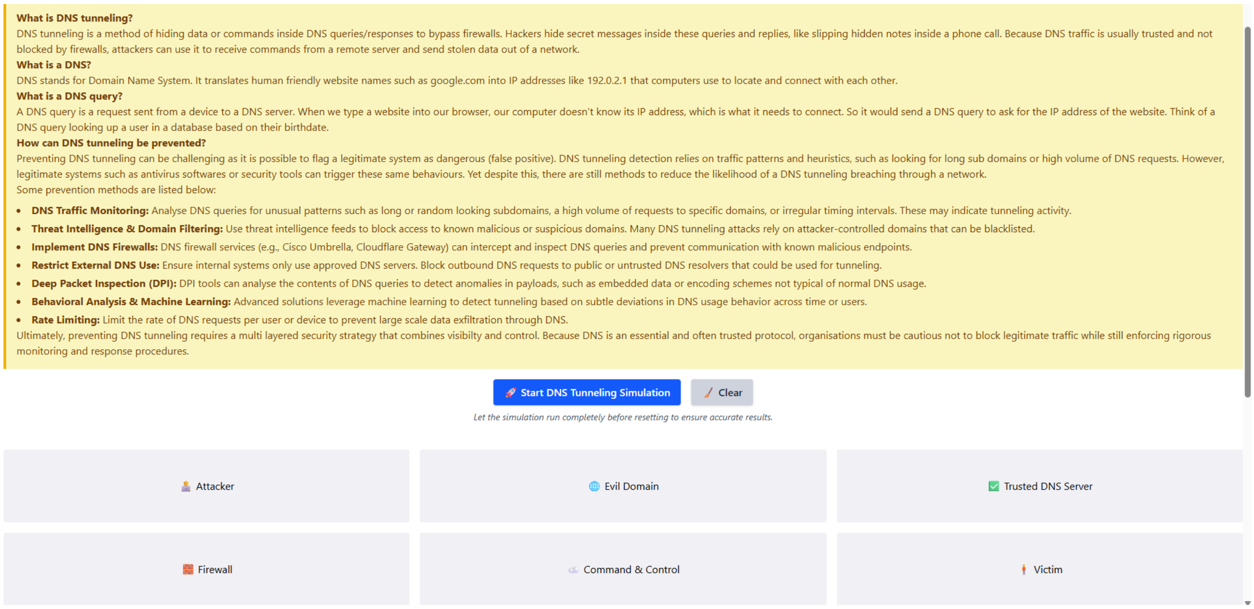
## A2



a •

*There is a lot of traffic. Please be patient.*

A3



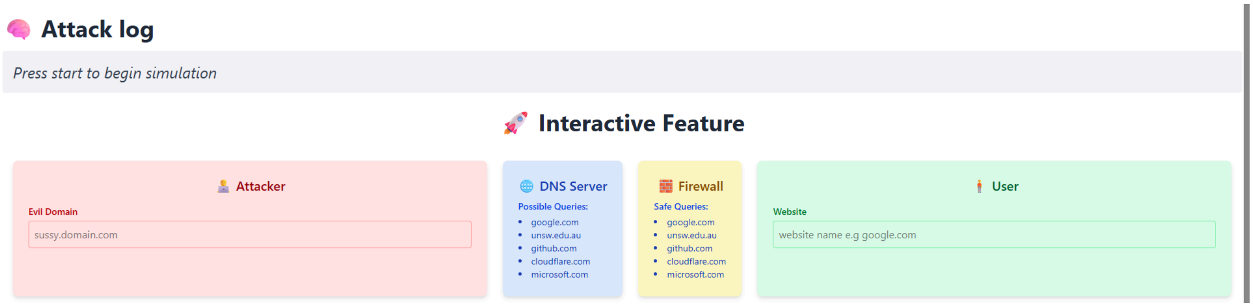
**What is DNS tunneling?**
DNS tunneling is a method of hiding data or commands inside DNS queries/responses to bypass firewalls. Hackers hide secret messages inside these queries and replies, like slipping hidden notes inside a phone call. Because DNS traffic is usually trusted and not blocked by firewalls, attackers can use it to receive commands from a remote server and send stolen data out of a network.
**What is a DNS?**
DNS stands for Domain Name System. It translates human friendly website names such as google.com into IP addresses like 192.0.2.1 that computers use to locate and connect with each other.
**What is a DNS query?**
A DNS query is a request sent from a device to a DNS server. When we type a website into our browser, our computer doesn't know its IP address, which is what it needs to connect. So it would send a DNS query to ask for the IP address of the website. Think of a DNS query looking up a user in a database based on their birthdate.
**How can DNS tunneling be prevented?**
Preventing DNS tunneling can be challenging as it is possible to flag a legitimate system as dangerous (false positive). DNS tunneling detection relies on traffic patterns and heuristics, such as looking for long sub domains or high volume of DNS requests. However, legitimate systems such as antivirus softwares or security tools can trigger these same behaviours. Yet despite this, there are still methods to reduce the likelihood of a DNS tunneling breaching through a network.
Some prevention methods are listed below:

- **DNS Traffic Monitoring:** Analyse DNS queries for unusual patterns such as long or random looking subdomains, a high volume of requests to specific domains, or irregular timing intervals. These may indicate tunneling activity.
- **Threat Intelligence & Domain Filtering:** Use threat intelligence feeds to block access to known malicious or suspicious domains. Many DNS tunneling attacks rely on attacker-controlled domains that can be blacklisted.
- **Implement DNS Firewalls:** DNS firewall services (e.g., Cisco Umbrella, Cloudflare Gateway) can intercept and inspect DNS queries and prevent communication with known malicious endpoints.
- **Restrict External DNS Use:** Ensure internal systems only use approved DNS servers. Block outbound DNS requests to public or untrusted DNS resolvers that could be used for tunneling.
- **Deep Packet Inspection (DPI):** DPI tools can analyse the contents of DNS queries to detect anomalies in payloads, such as embedded data or encoding schemes not typical of normal DNS usage.
- **Behavioral Analysis & Machine Learning:** Advanced solutions leverage machine learning to detect tunneling based on subtle deviations in DNS usage behavior across time or users.
- **Rate Limiting:** Limit the rate of DNS requests per user or device to prevent large scale data exfiltration through DNS.

Ultimately, preventing DNS tunneling requires a multi layered security strategy that combines visibilty and control. Because DNS is an essential and often trusted protocol, organisations must be cautious not to block legitimate traffic while still enforcing rigorous monitoring and response procedures.

*Let the simulation run completely before resetting to ensure accurate results.*

A4



## Attack log

*Press start to begin simulation*

## 🚀 Interactive Feature

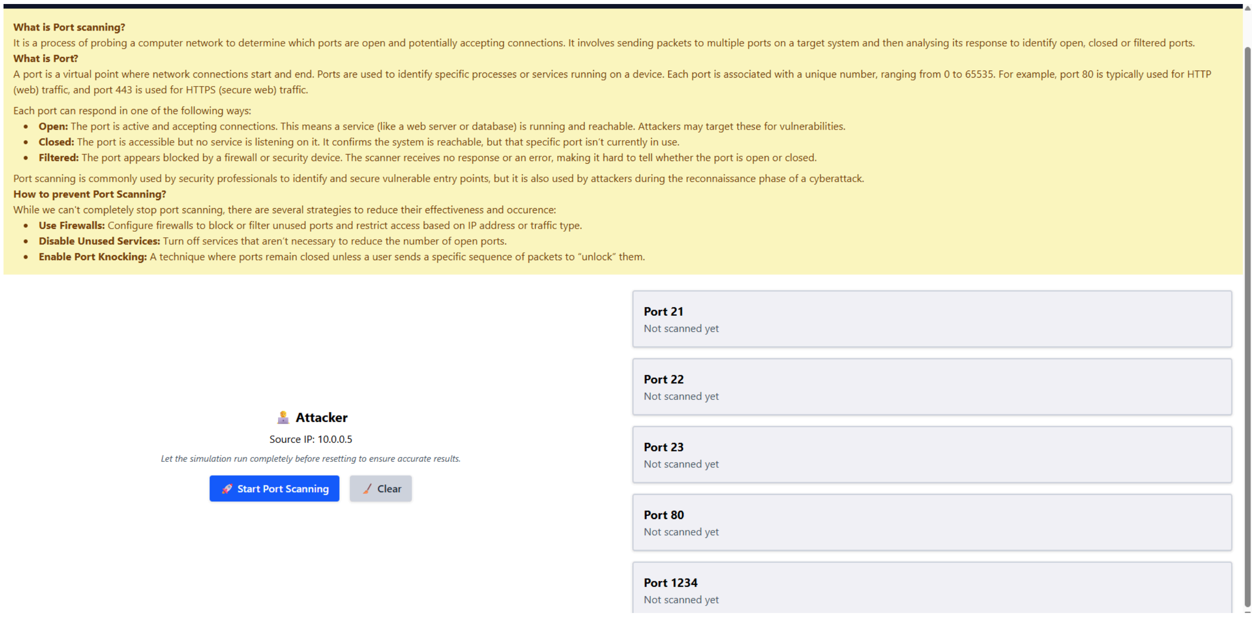
Evil Domain: sussy.domain.com

Possible Queries:
- google.com
- unsw.edu.au
- github.com
- cloudflare.com
- microsoft.com

Safe Queries:
- google.com
- unsw.edu.au
- github.com
- cloudflare.com
- microsoft.com

Website: website name e.g google.com

A5



**What is Port scanning?**

It is a process of probing a computer network to determine which ports are open and potentially accepting connections. It involves sending packets to multiple ports on a target system and then analysing its response to identify open, closed or filtered ports.

**What is Port?**

A port is a virtual point where network connections start and end. Ports are used to identify specific processes or services running on a device. Each port is associated with a unique number, ranging from 0 to 65535. For example, port 80 is typically used for HTTP (web) traffic, and port 443 is used for HTTPS (secure web) traffic.

Each port can respond in one of the following ways:

- **Open:** The port is active and accepting connections. This means a service (like a web server or database) is running and reachable. Attackers may target these for vulnerabilities.
- **Closed:** The port is accessible but no service is listening on it. It confirms the system is reachable, but that specific port isn't currently in use.
- **Filtered:** The port appears blocked by a firewall or security device. The scanner receives no response or an error, making it hard to tell whether the port is open or closed.

Port scanning is commonly used by security professionals to identify and secure vulnerable entry points, but it is also used by attackers during the reconnaissance phase of a cyberattack.

**How to prevent Port Scanning?**

While we can't completely stop port scanning, there are several strategies to reduce their effectiveness and occurence:

- **Use Firewalls:** Configure firewalls to block or filter unused ports and restrict access based on IP address or traffic type.
- **Disable Unused Services:** Turn off services that aren't necessary to reduce the number of open ports.
- **Enable Port Knocking:** A technique where ports remain closed unless a user sends a specific sequence of packets to "unlock" them.

**Attacker**

Source IP: 10.0.0.5

*Let the simulation run completely before resetting to ensure accurate results.*

**Port 21**
Not scanned yet

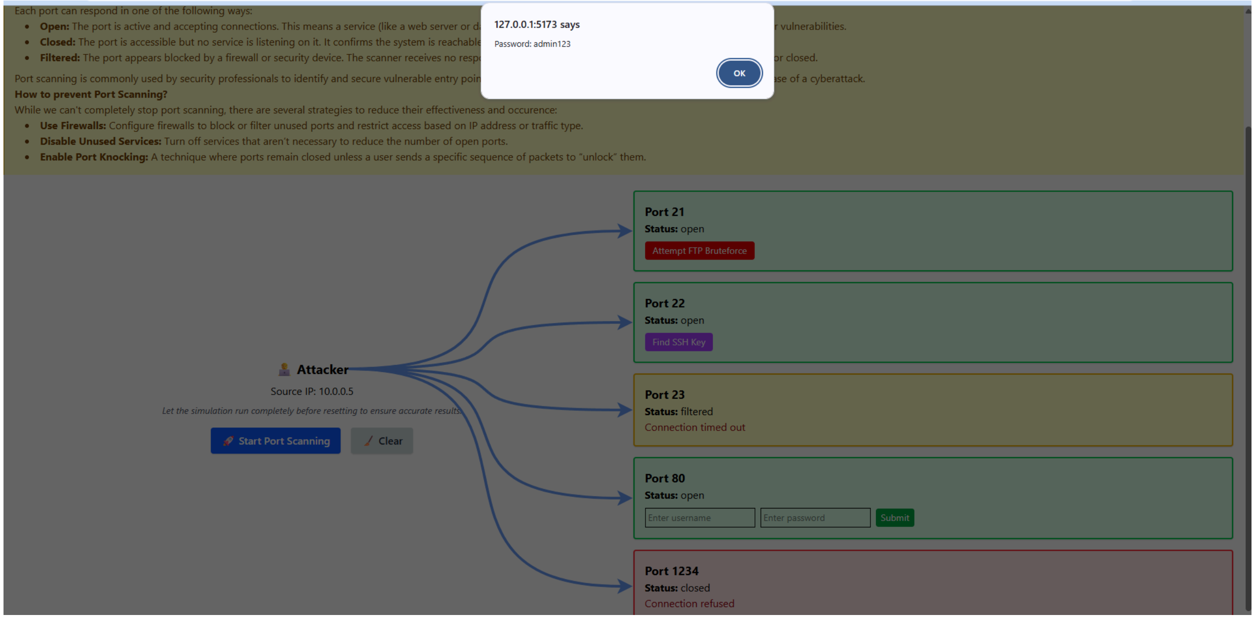
**Port 22**
Not scanned yet

**Port 23**
Not scanned yet

**Port 80**
Not scanned yet

**Port 1234**
Not scanned yet

A6



127.0.0.1:5173 says

Password: admin123

Each port can respond in one of the following ways:

- **Open:** The port is active and accepting connections. This means a service (like a web server or d r vulnerabilities.
- **Closed:** The port is accessible but no service is listening on it. It confirms the system is reachable
- **Filtered:** The port appears blocked by a firewall or security device. The scanner receives no resp or closed.

Port scanning is commonly used by security professionals to identify and secure vulnerable entry poin ase of a cyberattack.

**How to prevent Port Scanning?**

While we can't completely stop port scanning, there are several strategies to reduce their effectiveness and occurence:

- **Use Firewalls:** Configure firewalls to block or filter unused ports and restrict access based on IP address or traffic type.
- **Disable Unused Services:** Turn off services that aren't necessary to reduce the number of open ports.
- **Enable Port Knocking:** A technique where ports remain closed unless a user sends a specific sequence of packets to "unlock" them.

**Attacker**

Source IP: 10.0.0.5

*Let the simulation run completely before resetting to ensure accurate results.*

**Port 21**
**Status:** open

**Port 22**
**Status:** open

**Port 23**
**Status:** filtered
Connection timed out

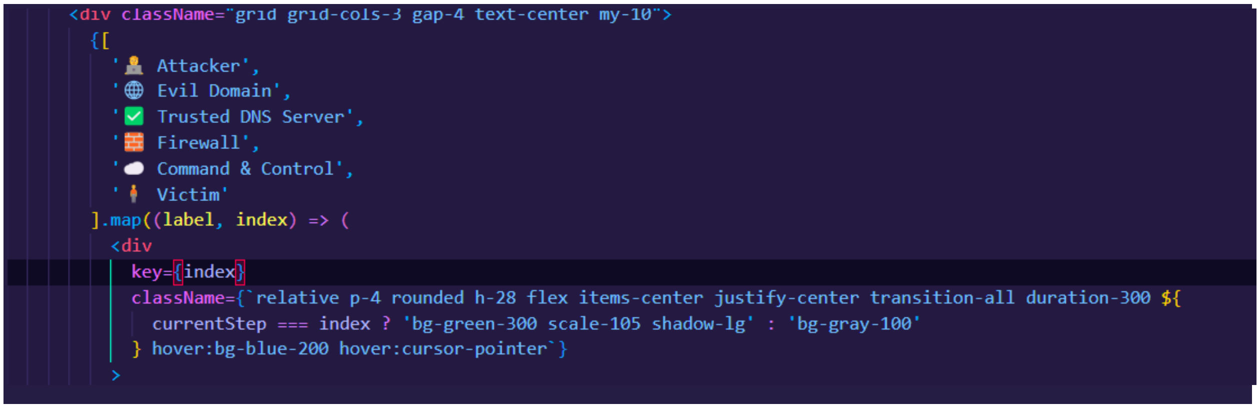
**Port 80**
**Status:** open

**Port 1234**
**Status:** closed
Connection refused

# A7

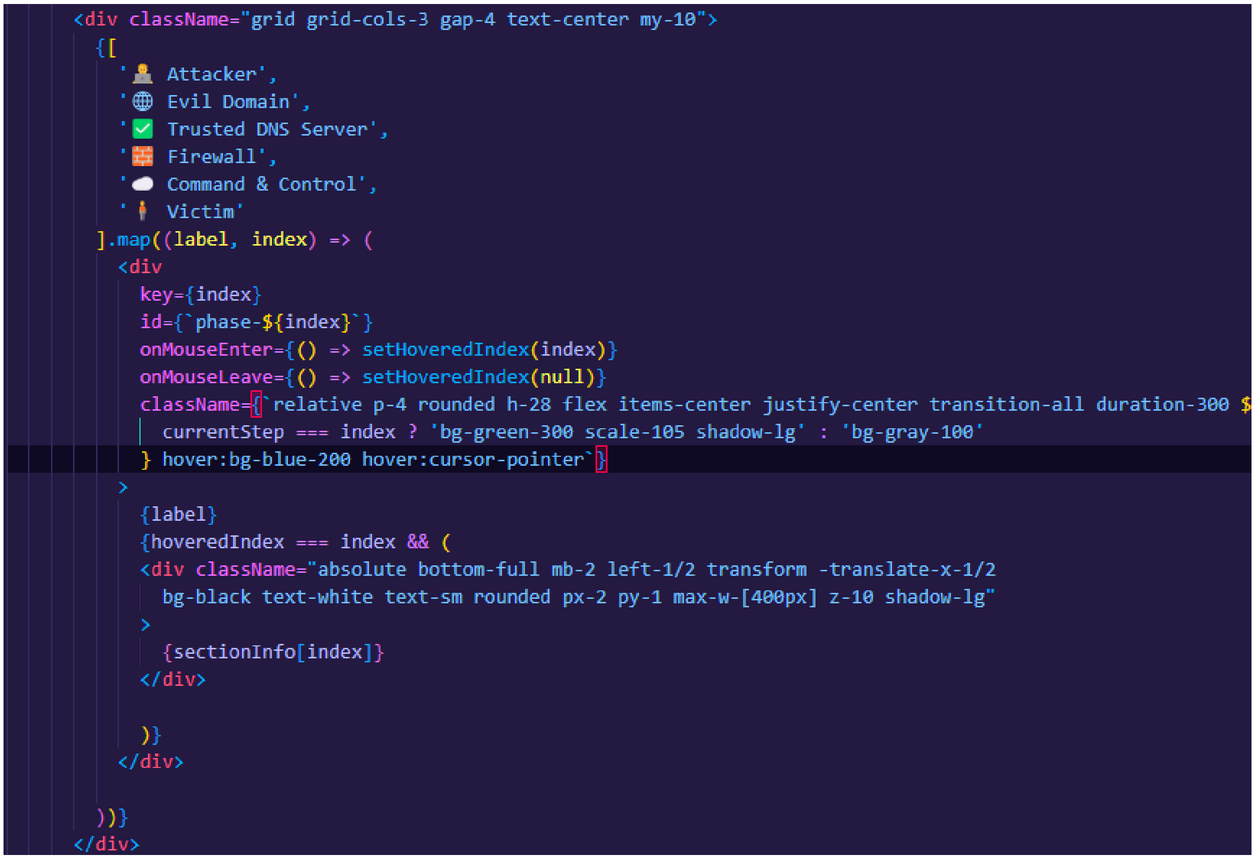
https://github.com/AaronthiccThai/NetworkAttackVisualiser
https://github.com/AaronthiccThai/NetworkAttackVisualiser/commits/main/

# A8



```
<div className="grid grid-cols-3 gap-4 text-center my-10">
  {[
    '🧑‍💻 Attacker',
    '🌐 Evil Domain',
    '✅ Trusted DNS Server',
    '🧱 Firewall',
    '☁️ Command & Control',
    '🧍 Victim'
  ].map((label, index) => (
    <div
      key={index}
      className={`relative p-4 rounded h-28 flex items-center justify-center transition-all duration-300 ${
        currentStep === index ? 'bg-green-300 scale-105 shadow-lg' : 'bg-gray-100'
      } hover:bg-blue-200 hover:cursor-pointer`}
    >
```

# A9



```
<div className="grid grid-cols-3 gap-4 text-center my-10">
  {[
    '🧑‍💻 Attacker',
    '🌐 Evil Domain',
    '✅ Trusted DNS Server',
    '🧱 Firewall',
    '☁️ Command & Control',
    '🧍 Victim'
  ].map((label, index) => (
    <div
      key={index}
      id={`phase-${index}`}
      onMouseEnter={() => setHoveredIndex(index)}
      onMouseLeave={() => setHoveredIndex(null)}
      className={`relative p-4 rounded h-28 flex items-center justify-center transition-all duration-300 $
        currentStep === index ? 'bg-green-300 scale-105 shadow-lg' : 'bg-gray-100'
      } hover:bg-blue-200 hover:cursor-pointer`}
    >
      {label}
      {hoveredIndex === index && (
      <div className="absolute bottom-full mb-2 left-1/2 transform -translate-x-1/2
        bg-black text-white text-sm rounded px-2 py-1 max-w-[400px] z-10 shadow-lg"
      >
        {sectionInfo[index]}
      </div>

      )}
    </div>

  ))}
</div>
```