DEVELOPMENT OF SECURITY SYSTEMS FOR IIOT DEVICES USING THE PROCESSING OF THE INFORMATION CARRIED BY THE LOCAL IP ADDRESSES

Tudor FERECUȘ[1], Mihalache GHINEA[2]

Rezumat. *De la începuturile industriei IIoT 4.0, este imposibil să te referi la orice echipament modern fără a face referire la modul în care acesta gestionează datele. Cel mai comun răspuns va include conceptul de „transfer wireless de date", care sunt ulterior stocate în baze de date. Această lucrare încearcă să maximizeze informațiile provenite din comunicarea Wi-Fi, propunând modalități de implementare a mentenanței asistate pentru echipamentele industriale.*

Abstract. *Since the IIoT 4.0 industry began, it is impossible to refer to any modern piece of equipment without referencing how it handles data. The most common answer will include the concept of "wireless data transfer", which is later stored in databases. This paper tries to maximize the information that comes from Wi-Fi communication, proposing ways to implement assistive maintenance for industrial equipment.*

Keywords: Industry 4.0, Wi-Fi, maintenance, security

1. Introduction

The paper focuses on exploring new and more efficient ways of using the network connection for its data-fetching properties. To achieve and put into perspective the capabilities of this practice, the implementation consisted of several modules and concepts: automatic local IP fetching, cross-device communication, secure database, automatic attendance recognizer, smart IIoT security, and maintenance web application. Every module and its past iterations will be thoroughly documented in the next chapters, heavily focused on the efficiency, reliability, and security measures of the programming behind it.

The importance of cybersecurity

Even though data security seems to be irrelevant to this conversation and its scope, the paper's structure documents both theoretical and experimental projects, it is impossible to apply the method in a real-life environment without assuring it can provide security of data.

[1] Student, National University of Science and Technology POLITEHNICA Bucharest, ACS Faculty, Spl. Independenței 313, ZipCode 060042. E-mail: tudorfrerecus@gmail.com
[2] Associate Professor, National University of Science and Technology POLITEHNICA Bucharest, IIR Faculty, Spl Independentei 313, ZipCode 060042. E-mail: ghinea2003@yahoo.com

IIoT 4.0 devices use various ways of sharing and getting data. Most of those create and operate securely while their IP is hidden and cannot be exploited. The experimental web applications this paper shows store the local IP to operate, and this process may cause a lot of damage if done incorrectly.

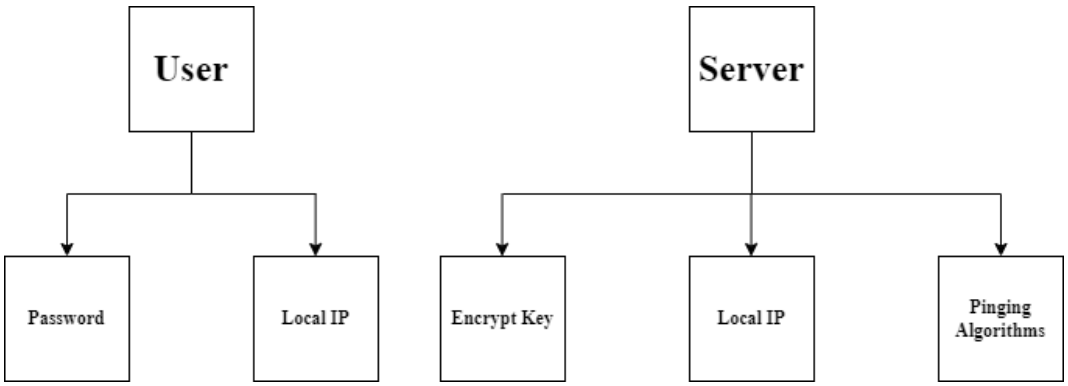
The paper studied the devices from "Smart Pneumatics Lab", FIIR Faculty, and how a leak of IP addresses can damage an entire complex infrastructure of advanced IIoT 4.0 devices. The SSH (secure shell protocol) is often used for accessing a device and relies entirely on an IP address and a password. With an IP address leak, an attacker can use it to start a connection and apply several different methods to find the password. One of the newer methods to achieve this is Machine Learning [15] or even simpler methods like brute force [16][17]. The method used in the laboratory consists of a login cracker, Hydra. With a list of common passwords and the hacked data about the user (IP and password), it took under a minute to crack the Edge of the lab and infiltrate it using the SSH route. This not only raised concerns about the importance of data security, but device security as well. It only took this line of bash code:

hydra -L usernames.txt -P passwords.txt ssh://192.168.1.100

where usernames.txt has the device username, passwords.txt has a list of common passwords, and the link is the SSH address (got from the acquired local IP).

2. Cybersecurity

There are security concerns and sensitive data that must be protected. These modules [2] [5] are shown in Fig. 1



Fig, 1 Modules that require cybersecurity [13]

Each module requires a separate method to ensure no data can be stolen or modified.

One of the main functionalities of any project's database is the security it provides to the most important recipient, the user [9]. The most dangerous database is the one that stores every type of data raw, without hashing or encrypting [20]. This not only makes our users' data accessible in case of a breach but can affect the users who use the same password on multiple websites. One of the most common types of digital account theft is credential stuffing, which takes advantage of smaller, poorly secured applications that get breached and applies the obtained data to other websites, that are harder to hack. The cybersecurity of a website is crucial in this case. [6] [11]

2.1 Password

Password storing in the database is a fairly easy process as it is documented in many instances due to the high frequency of services that use that type of credentials [4].

2.1.1 Data hashing

The way the backend ensures its data is being securely transmitted has more layers. First, the communication from the frontend (user-side) to the backend is hashed so the packets transmitted that may be intercepted make no sense to the intruder. The process can be visualized in Fig. 2. [8]

Fig. 2 JSON Package hashing

2.1.2 Password encrypting

With secure package delivery, the next step is storing sensitive information in the database. Hashing is not a measure that will suffice for saving data, as it can be decrypted easily once retrieved. The logic behind the database processes the data and then encrypts it using the bcrypt algorithm [3]. It uses a key that is only known by the database's admin. The process looks like the one in the Fig. 2, as it is incomprehensible.

2.2 Local IP encryption

The local IP securing process is similar, hashing it in the operation of transmitting it and then encrypting using the same algorithm. The difference occurs when its known structure is taken into consideration. An IP has a typical structure, as it

always has certain characters. Knowing these characters and the fact that the 4 numbers are stored in one byte each (they can have values from 0 to 255) can be a way for the attacker to reduce the complexity of the decryption. It remains a very time-consuming action because of the current computational power, but taking more measures can be a good way of future-proofing the algorithm.

Fig. 3 User data representation in the database

Last byte encrypting

The decision to store and encrypt only the IP's last byte is because it adds another unknown variable for a possible attacker. To obtain the whole local IP address they should get the router's local address as well as the 4th byte, which is fragmented information. The process also clears the stored string of regularities, which restores the bcrypt algorithm's efficiency. [14]

2.3 Server-side data

Modules such as the key, the server's local IP, and algorithms are data that should not be accessed easily. Being stored locally, the server's data is not as exposed to hacking as the user's data, but a layer of processing is good for unwanted glances over the text. [7]

Using the library "pyinstaller", the code is transformed into an executable which we can use for running the scripts, as well as "hashing" the code. The executable is not human-readable, so it makes the important data much safer. Due to this method, the information will be stored incomprehensibly. The process of compiling can be reversed, but the lines of code would look different and harder to understand, taking more time to get important information.

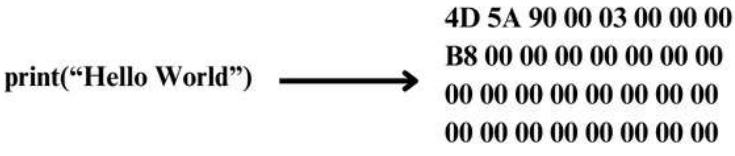


Fig. 4 Visualization of executable's content

3. Web applications and algorithms

The paper proposes an approach to the algorithm's theoretical and experimental contributions. The theoretical part consists of the iterations the method had to go through to achieve the desired efficiency and consistency. The experimental part has two web applications deployed and tested in the "Smart Pneumatics Lab" from the Faculty of Industrial Engineering and Robotics.

3.1 Project overview

The project covers two fully developed web applications. They share the core functionality, each having dedicated files (that the paper refers to as "middleware"), and the same database (the backend), but use them in separate ways to achieve their purpose.

3.1.1 Web applications' overview

The first application, called AllDay, is a tool dedicated to studying attendance in a laboratory or a meeting room, etc., and providing useful data about past analysis, including productivity, the best time for future meetings, and many more.

The second web application, IoTCentre, uses the core functionality more industrially focused, acting as a tool for centralizing the IoT equipment, regardless of the manufacturer, providing instantaneous feedback and an additional level of security for it.

3.1.2 Core functionality

As stated above, each application uses the same framework and builds over it, showing its versatility. This base structure consists of two wireless data manipulation files and a database API for storing them. [1]

The first script, called "pingUsers", scans and requests data to the router to check for the connected devices, data that it stores locally and intersects with all the relevant IPs, and data acquired from the database through a GET request. The script is written in Python.

Because the local IP fetching process can be hard for the end-user to do, the project implements a way to do it automatically.

3.1.3 Limitation workarounds

The applications were developed in the web space to have the most easy-to-use experience and not require any other applications. This brings the opportunities that web development provides, but also its limitations. For security reasons, the JavaScript programming language has no support for interacting with the devices

that access the web application, so the implementation had to find a way to fetch local IP addresses without accessing any device-related data. The way to solve the problem is by hosting a page dedicated to the device data gathering (like a register or device adding page) locally on a server that has access to the designated internet connection. When the user wants to add a device, instead of loading a page hosted remotely, they will be redirected to the local instance of that page. By doing that, it creates a low-level connection between the accessing device and the server, which can be followed to find the data carried by it with this command:

netstat -an | FINDSTR 'X.X.X.X:PORT'

where X.X.X.X:PORT is the local IP of the server. The command works in Windows CLI and Linux-based distributions, which can be run from the local JavaScript-based server using the 'os' library. Thus, the network limitations are ignored by creating a connection between endpoints that only provides what the application needs, without crossing the security concerns that made these types of fetches unallowed.

3.2 Device pinging implementation

The central part of the paper consists of a way to efficiently and reliably get the responsive local IP addresses. Each iteration of the algorithm has its upgrades from its predecessor.

3.2.1 Shell scripting

As a prototype, the first approach of the paper was to make a draft-type script in bash to test its performance and reliability. Even though it would seem an unimportant step in the process, it had the following important results:

- Bash scripting on its own should be used for a lower-level, more system-oriented way of programming, but the process the paper is researching focuses on internet communication and cybersecurity, so the path chosen has restrictions regarding its capabilities.
- The default ping used by the system was too slow to run multiple times per IP checking but too unreliable to check only once.

Fig. 5 Shell Scripting Algorithm

3.2.2 Python script with no custom libraries

With the results of the last iteration in mind, the next scripts were written in Python, a programming language that is faster to code in and has wider support. The defining part of the algorithm was that it used a different OS function, netstat. The following conclusions were made:

- The netstat OS function is a powerful tool that can provide more information than the process needed. Even with parameters such as "-n" or "-a" [18] that save processing time, the command was not efficient enough because of its computation power.
- Moving to Python was a step in the right direction and it was clear that processes that can provide all open and active ports at a time can give faster results than individually pinging every device.

3.2.3 Python script with „scapy" library

„Scapy" is a powerful tool used for pocket manipulation. It was used in this process for creating, sending, and dissecting network packets.[2] The process is more efficient because it does not depend on the tools from 3.2.1 and 3.2.2 but makes use of network protocols. The conclusion that this approach is the best one came from the following analysis:

When a phone enters its „standby" mode affecting the request of network packets from the router, the pings need to be more frequent to catch the exchange of data between the two endpoints. This means that the best approach is the one that can check all IPs of interest in a low enough amount of time to execute the process multiple times a second. For the „ping" function to reliably return the right results it should be executed at a big frequency [10]. The tests showed that 100 pings should give reliable answers. The process would take ~2 minutes per device, an inadequate time.

The last process takes 1.2 seconds to make an iteration that returns the IPs of all devices. The immense time saver comes from the process of returning all the devices in one step, rather than one at a time. The time saved compared to the other approach can be calculated with the following formula:

$$\Delta = \frac{T_{sc}}{T_p \cdot devices}$$
$$Where \begin{cases} T_{sc} = Scapy\,Library\,Method \\ T_p = Ping\,Method \end{cases}$$

Fig. 6 delta time between the 2 approaches

To put the numbers into perspective, for 10 devices (a time of 12 seconds for the ”scapy” method and 20 minutes for the ping method), the resulting time will be 0.01 of the time the first method would take. This is a great alternative to the commonly used „ping” or even “netstat” function.
Due to the algorithm’s efficiency (for 100 packets – 2 minutes and 97.3% accuracy), it is not required to do further iterations, the results being satisfactory.

3.2.4 Final algorithm

The final step in developing the algorithm consists of adding functionality on top of the base one. The data is separated into declarative values such as IP address, number of iterations, and logic. The values are stored in a more secure type of way, an ”.env” variable, that is accessed using the “dotenv” library.
Firstly, it checks if the server has a working Wi-Fi connection by making a get request to google.com (the website does not matter as long as it is accessible), if the request gets any type of data, it means the Wi-Fi connection works as intended.

Secondly, the program checks if the pinging happens in the desired period, as many trackers do not require 24/7 activity. A good case for this functionality is checking devices just during working hours.

Thirdly, after the necessary checks, the logic executes the main algorithm, storing the found devices. After the iterations are done, the program executes a POST request to the database, containing the local IPs of the found devices and the time the request was sent. [9]

4. Local IP fetching

While it may seem redundant to invest time and resources into the developing of an automatic local IP fetcher, the process makes the applications easier to use,

making it easier to set up devices for people who do not know enough about IPs and speeding the process of registering devices, regardless of the experience.
The development was a hard task because of the limitations presented in the overview: a website cannot interact with the device that accessed it on a detailed enough level to get the data the process needs. The way to solve the problem was to create a stronger connection between our server and the devices that use them. Hosting a webpage locally makes use of the core functionalities of hosting and routing, creating a way in which devices connect directly to the server, without needing additional layers like the remote hosting. With this method, all the devices connected to the page can be seen as connected directly to the local server. From this point, the „netstat" command in bash (or Powershell depending on the OS) will list all the connections that the server has with other devices:

netstat -an | FINDSTR 'X.X.X.X:PORT'

This approach is a way to get the desired data without any real performance cost. The way to store the locally red information is by calling the command shown earlier in a local instance of the database, using the „os" library provided by NodeJS.

```
const getUserLocalIp = (callback) => {
    exec(`netstat -an | FINDSTR ` + `"` + process.env.LOCAL_IP + ':' + process.env.LOCAL_PORT + `"`,
    function (error, stdout, stderr) {
    if (error !== null)
        callback("fail");
    callback("fin " + stdout);
    });
}
```

Fig. 7. Code for local IP fetching.

5. The experimental development of the theoretical application

This ping method is a reliable way of determining devices connected to Wi-Fi. It has shown consistency in results compared to the other methods. This accomplishment allows for device-tracking alternatives that otherwise required more resources and were overall pricier.
Two web applications were created to test the method's capabilities, which proves this method is efficient in different real-life situations.

5.1 Hardware requirements

A script's versatility is driven by its requirements, the scope of the paper is to create a method that can be easily run with modern computers, which means that efficiency is one of the main characteristics of the algorithm.

5.1.1 Goal regarding the hardware requirements

Wireless data was optimized over time to work as fast as it could while maintaining low hardware specifications. To preserve the efficiency of this technology, the paper and its techniques require low CPU and memory usage, as well as not overflowing the bandwidth. The used libraries improve standard (but slow) Wi-Fi-related commands (such as “ping”), making them a better use for the implementation’s case. The process went through many iterations to require lower specifications, making it a good fit for every server type.

5.1.2 Resource-consuming modules

The applications for this module are divided into two categories: locally and remotely hosted. The projects that fall in the last case are the application’s frontend (apart from the locally hosted page) and backend. The local server will run the script for the recognition of the local IP and a lightweight instance of the website, being hosted only the IP fetching page.

5.1.3 Hardware analysis

This section will refer only to the parts of the project that are meant to be run locally. The remote hardware usage is not important as the servers of the desired host will run them.
The consumers when it comes to hardware are the „pingUsers.py” script, the backend, and the frontend. Looking at data from before and after calling the scripts shows that the process consumes about 435MB of memory and 16% of the processor power.

Fig. 8 Tracking algorithm’s resources

Fig, 9 Lightweight frontend and backend resources

The processor used for the tests is an Intel i7-10750H.

5.2 Attendance tracker

The attendance tracker, also referred to as “AllDay” for simplicity, is an application that utilizes the methods reminded earlier to track devices such as phones to provide an in-depth analysis of the gathered data.

5.2.1 Main Modules

Both web applications were built so that they use the same backend (with the same encrypting and security measures) and the same IP recognition algorithm. The modularity of both modules comes in handy as it provides an easy way to adapt them to meet our requirements: the backend is focused on storing the pinging algorithm results (referred to as connections), functionality already built-in, and the tracking algorithm, due to the phones' sleeping functionality, should send 100 packets instead of the normal 50, to increase the accuracy from 14.6% to 97.3% (the remaining accuracy error can be resolved from the backend analysis).

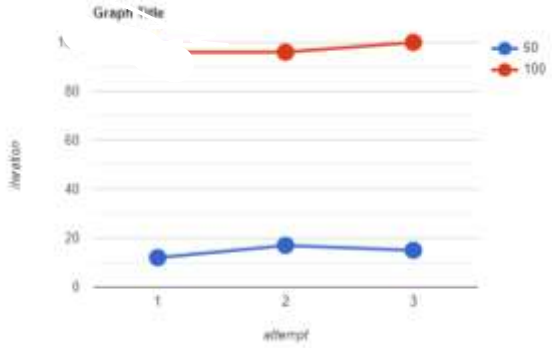


Fig 10 Line chart of accuracy change with different iteration values

5.2.2 Functionality

The data gathered from the network packets [19] is then stored in the database and the tool analyses it (as can be seen in Fig. 11) and provides it to the users. It shows in real-time the devices it recognizes (Fig, 12), a feature that can be particularly useful in laboratories that use shared keys.

Fig. 11 Data analysis on attendance

Fig. 12 Attendance visualization

5.3 Device maintenance and virtual centralization

A more technical approach to the method is to apply it in the IIoT 4.0 industry, by creating a web application that checks regularly which devices (such as sensors or bridges that need to be connected to the internet continuously to provide the desired data) are functional, sending notifications when a tool malfunctions.

5.3.1 Main Modules

The process makes use of the modules developed as part of the paper, using the backend similarly to the attendance tracker, but the tracking algorithm is used differently, checking if all the desired devices are operational and, if they malfunction, sends an email addressing the problem, with detailed time of the occurrence and possible reasons (specifying if the problem came from the router, or the device, etc.). In special cases, the database will store the frequency of data sharing, so the algorithm checks the respective IIoT tool at the right time interval.

5.3.2 Functionality

The application provides a way to add devices, setting their name, type, photo (optional), and frequency (optional). After their addition, they are checked with a frequency chosen by the user. It helps with seeing the state of all devices, regardless of their manufacturer or their drivers, in a minimalistic way that assures the user that it is operational and working as intended. The application also has the option to notify the admin if a device crashes.

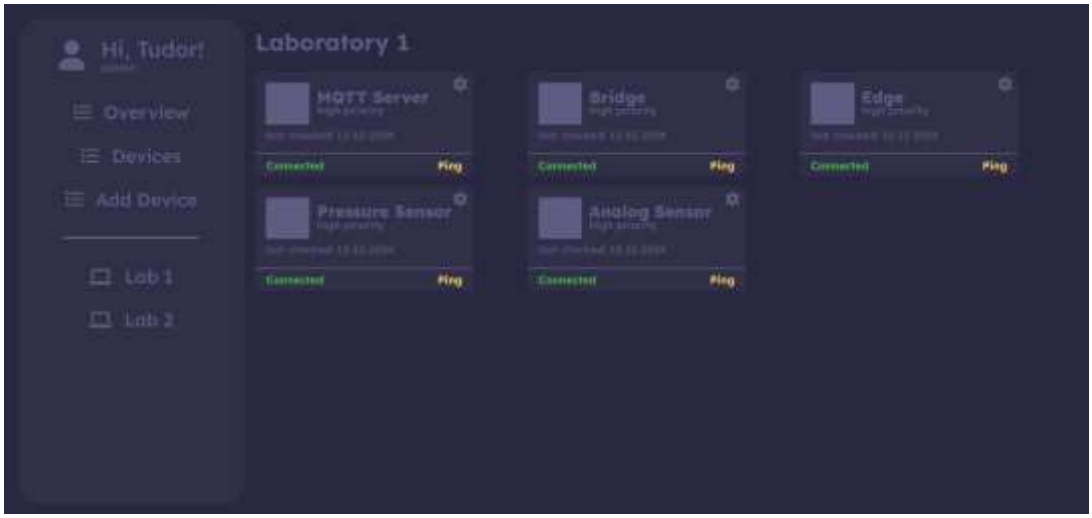


Fig. 13 Device overview

6. Conclusions

The paper contributes new methods and ideas regarding device tracking within a local connection, studying the steps and conclusions drawn along the way, with data to support the architecture's efficiency and reliability.

Theoretical contributions

The theoretical contribution is attributed to the method of fetching local IP in a web application, (using the local server workaround) and the device tracker algorithm which can detect devices connected to the Wi-Fi consistently and effectively. The process's strength comes from its reliability with devices that, after a period of inactivity, access network packets less frequently, an event that makes functions such as "ping" or "netstat" inconsistent.

Experimental contributions

The experimental contribution comes from the two web applications. They provide a new method of tracking and analyzing device data, without other methods' high costs or requirements. The written code's modularity makes the process easy to set up and maintain while providing the possibility for future updates. Each site explores different use cases for the algorithms, one having an industrial approach, while the other has a more general use.

Cybersecurity

When working with personal information it is paramount to store it securely. The paper shows the ways that the database ensures no information is stolen, from

hashing the data transfer between the frontend servers to encrypting the IPs and passwords. The implications of data theft in these web applications' cases can vary from credential stuffing to cyberattacks (such as SSH infiltrations or DDoS). The conclusions drawn are that any type of remote communication can be tracked, and data is not safe in its raw format. The solution in the paper is for it to be hashed and encrypted while keeping in mind the data processed and its security concerns (e.g. local IP) which should be resolved before encryption and storing.

Future goals

The future goals for this approach are to test it in different environments and get enough data to know it will perform in any type of configuration, with any type of IoT device. A future iteration of the algorithm should also include a way to check if the router, because of any malfunction or reset, might regenerate the network identifiers. Another interesting idea that came to mind when finishing the article is to experiment with MAC addresses, instead of local IP addresses, they are unique for every device (which is a plus over the local IPs that might be shared between devices) and have similarities with the IP addresses which means the adaptation of the algorithms should be easy.

R E F E R E N C E S

[1] Istvan Papp, Roman Pavlovic, Marija Antic - WISE: MQTT-based protocol for IP device provisioning and abstraction in IoT solutions
[2] Philippe BIONDI - Network packet manipulation with Scapy
[3] Erna Nababan - Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force
[4] Stephan Wiefling - Usability, Security, and Privacy of Risk-Based Authentication
[5] Saleh Aldaaje - Strategic cybersecurity
[6] Chahak Mittal - An Empirical Study on Cybersecurity Awareness, Cybersecurity Concern, and Vulnerability to Cyber-attacks
[7] Edwin Frank - Industrial Internet of Things (IIoT) Security Challenges: Exploring the trust deficit and security concerns in the context of IIoT deployments
[8] Joshua Cena - Zero-Trust Architecture for Robust IIoT Security
[9] Edwin Frank, Godwin Oluwafemi Olaoye - Industrial Internet of Things (IIoT) Security Challenges: Exploring the trust deficit and security concerns in the context of IIoT deployments
[10] Bernard Sentala, Charles Smart Lubobya, Ackim Zulu - Performance Evaluation and Compression of IP Packets in a Wireless Local Area Network (WLAN)
[11] Tim Murkomen - Performance, Privacy, and Security issues of TCP/IP at the Application Layer: A Comprehensive Survey
[12] Winnie Owoko - Survey on Evolving Threats in TCP/IP Header Attacks: Emerging Trends and Future Directions
[13] Mpekoa, Noluntu - An Analysis of Cybersecurity Architectures. International Conference on Cyber Warfare and Security

[14] Edwin Frank - Cryptographic Solutions for IIoT Security: Analyzing the role of cryptographic techniques in addressing security challenges in IIoT environments
[15] Amer Ali Hamza Jumma s Urayh Al-Janabi - Detecting Brute Force Attacks on SSH and FTP Protocol Using Machine Learning: A Survey
[16] Gain SSH Access to Servers by Brute-Forcing Credentials - https://null-byte.wonderhowto.com/how-to/gain-ssh-access-servers-by-brute-forcing-credentials-0194263/
[17] Namje Park - Network Log-Based SSH Brute-Force Attack Detection Model
[18] Giovanni Vigna, Richard A. Kemmerer - NetSTAT: A Network-based Intrusion Detection Approach
[19] Emilio Paolini, Luca Valcarenghi, Luca Maggiani, Nicola Andriolli - Real-Time Network Packet Classification Exploiting Computer Vision Architectures
[20] Pavani Wijegunawardhana DATABASE SECURITY THROUGH ENCRYPTION