NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

DOTA - 2023

Defense of the Ancients

The Projects for DOTA2023
(Defense of the Ancients)
Singapore, July 2023.

Table of Contents

Side Channel Timing Attack

Lin Zhiwei
Sichuan University
snakinya@gmail.com

Feng Quanbi
Southern University of Sinence and Technology
fquanbi@gmail.com

Liu Xinmeng
Southern University of Sinence and Technology
12110842@mail.sustech.edu.cn

Shan Tao
Harbin Engineering University
stao5171@gmail.com

Peng Rui
Sichuan University
pr156292834552021@163.com

ABSTRACT

The topic of our project is **Demo of Side-channel Attack**. In this paper, we will provide a brief introduction to side-channel attacks, with a particular focus on timing attacks and their cutting-edge research. Our objective is to demonstrate an attack model based on the HTTP/2 protocol within the existing network environment. Subsequently, we will attempt to extend it to HTTP/1.1 and finally discuss defense strategies.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

Keywords

Timing attacks, Channel measurement, Security, Information leakage

1. INTRODUCTION

In the past few decades, the academic community has been aware of and conducted extensive research on timing attacks in side-channel attacks [1]. When the execution time of an application or algorithm depends on some important information, then we can launch a timing attack. In the past few decades, there has been a wide range of attacks, such as obtaining the private key of an SSL/TLS implementation, or attacking a website to obtain user preferences or personal information. A typical timing attack is: the attacker sends a large number of requests to a service, and then obtains different response times of these requests. Through statistical analysis, the actual execution time of different inputs is learned, and according to this, the privacy information is obtained.

Based on the network architecture of the past few decades, numerous timing side-channel attacks have been successfully executed. In 2005, Brumley and Bonsh successfully carried out a timing side-channel attack, extracting SSL private keys from a local network [2]. In 2007, Bortz and Boneh demonstrated that such timing attacks could be used to obtain sensitive information from network applications [3]. They introduced two possible attack types. One is a direct attack, where the attacker establishes a direct connection with the network server and sends requests to perform the attack. The other is a cross-site timing attack, where the attacker deceives the victim into sending requests to a website controlled by the attacker. Their research showcased how such attacks can be utilized to expose users' private information. Other timing attacks focus on exploiting timing leaks in browsers [4] or utilizing TCP windows to determine response sizes [5]. These types of attacks also pose significant threats.

Timing attacks in the realm of networking face hindrances due to the implementation of both the underlying network infrastructure and the network applications themselves. A crucial aspect of executing such attacks relies on measuring the response times of the server to different requests. The more variability in the measurement results, the more challenging it becomes to accurately determine the execution time. Obtaining this timing information can be greatly affected by the instability of network transmissions, leading to significant variations, or by the server's efficient implementation, resulting in shorter response times.

When these factors converge in a real network environment, such as tiny differences experienced by data packets at each hop (network jitter), if they are of a similar or higher magnitude compared to the differences in server response times, successfully carrying out the attack becomes an extremely challenging task. Although multiple repeated measurements can help mitigate the interference caused by such situations, it is not sufficient to completely overcome the diversity in packet response times introduced by the complexity of the network.

In 2015, Gelernter and Herzberg conducted a renewed investigation into cross-site timing attacks and introduced innovative techniques to overcome the limitations imposed by the victim's unstable network connection. Their approach relied on manipulating the size of responses or the computational overhead required for processing server responses, effectively increasing the differences in response sizes or pro-

cessing times. This enabled them to successfully differentiate between different requests. The starting point of their work aligns with the direction we are pursuing in our project: We focus on making it feasible to detect small timing differences in contrast to increasing these time differences.

In contrast to classic timing attacks, where the attacker obtains many independent measurements through the network and then uses statistical methods to infer the processing time of the request, we try a concurrency based timing attack in this article, where the implementation of the attack relies on the relative timing difference between two requests executed concurrently.

Our report is divided into seven sections. In the first section, we introduce the measurement channel attack and our focus of work. In the second section, we introduce the threat model. In the third section, we describe the detailed principles, problems and implementation details of implementing timing attacks. We will also introduce our implementation in this section. The basic model (simple demonstration), based on the implementation of the HTTP/2 protocol and the extension attempted on HTTP/1.1. In Section IV we discuss defense strategies, possible solutions and analysis. We then conclude our work in Section V and acknowledge in Section VI. Section VII lists the articles we cite and related work.

2. BACKGROUND

2.1 Side-Channel Attack

In computer security, a side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented, rather than flaws in the design of the protocol or algorithm itself (e.g. flaws found in a cryptanalysis of a cryptographic algorithm) or minor, but potentially devastating, mistakes or oversights in the implementation. Timing information, power consumption, electromagnetic leaks, and sound are examples of extra information which could be exploited to facilitate side-channel attacks.

General classes of side-channel attack include:

- Cache attack - attacks based on attacker's ability to monitor cache accesses made by the victim in a shared physical system as in virtualized environment or a type of cloud service.
- Power-monitoring attack — attacks that make use of varying power consumption by the hardware during computation.
- Electromagnetic attack — attacks based on leaked electromagnetic radiation, which can directly provide plaintexts and other information. Such measurements can be used to infer cryptographic keys using techniques equivalent to those in power analysis or can be used in non-cryptographic attacks, e.g. TEMPEST (aka van Eck phreaking or radiation monitoring) attacks.
- Acoustic cryptanalysis — attacks that exploit sound produced during a computation (rather like power analysis).
- Differential fault analysis — in which secrets are discovered by introducing faults in a computation.
- Data remanence — in which sensitive data are read after supposedly having been deleted. (e.g. Cold boot attack)
- Software-initiated fault attacks — Currently a rare class of side channels, Row hammer is an example in which off-limits memory can be changed by accessing adjacent memory too often (causing state retention loss).
- Allowlist — attacks based on the fact that the allowlisting devices will behave differently when communicating with allowlisted (sending back the responses) and non-allowlisted (not responding to the devices at all) devices. Allowlist-based side channel may be used to track Bluetooth MAC addresses.
- Optical - in which secrets and sensitive data can be read by visual recording using a high resolution camera, or other devices that have such capabilities.

In all cases, the underlying principle is that physical effects caused by the operation of a system (on the side) can provide useful extra information about secrets in the system.

2.2 HTTP protocol

The full name of HTTP protocol is Hypertext Transfer Protocol, which is an application layer protocol in the Internet protocol suite model of distributed, writing and hypermedia information system. The HTTP protocol is the basis for data communication on the World Wide Web, where hypertext documents contain hyper connections to other resources that Internet users can easily access. HTTP is the most popular protocol among web services.

HTTP/1.1. It was the most widely used version of HTTP. The keep-alive mechanism was introduced in HTTP/1.1, allowing a connection to be flattered by multiple requests or responses. Such persistent connections reduce request latency because clients do not need to renegotiate TCP's three-way handshake connections in one request service. HTTP/1.1 also adds HTTP pipelines so that clients can send multiple requests before waiting for a response to each request, further reducing latency using persistent connections. However, due to a number of factors, including security and server-side implementation, this feature was disabled in subsequent releases.

A major limitation of timing attacks in HTTP/1.1 is its head-of-line (HOL) mechanism, which allows all requests on the same connection to be processed sequentially.

HTTP/2. It introduced the concept of streams, which are unique bidirectional data flows typically associated with request and response pairs. This eliminates the limitations imposed by Head-of-Line (HOL) blocking. Data is transmitted in the form of frames, with request and response bodies using data frames and message headers using header frames. These frames are compressed using a specific algorithm.

With each frame having a stream identifier, a web server can handle different requests simultaneously using only one TCP connection. As a result, responses can be sent immediately after they are generated on the server. By default, web servers treat requests equally and handle them accordingly.

3. THREAT MODEL

In our attack, we are considering only the direct attack model, where the attacker connects directly to the target server and obtains measurement values based on the responses received (as shown in Figure 1). In this attack model, the attacker can craft these packets as needed. Additionally, the attacker can send multiple packets simultaneously or consecutively, enabling them to measure the server's processing time effectively.

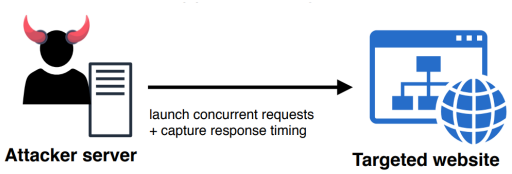


Figure 1: Direct Attack Model

For the network environment in which the attack takes place, we are considering two protocols: implementation under HTTP/2 and implementation under HTTP/1.1. Specifically, due to the differences in features, in HTTP/2, we can leverage the capabilities of streams to request the server to handle multiple requests simultaneously, thereby mitigating the impact of variations in transmission times of data packets in the network environment. On the other hand, in HTTP/1.1, the server processes the packets sent by the attacker sequentially.

4. IMPLEMENTATION

In this section, we will present the implementation and testing of two parts: a traditional timing attack model and a concurrency-based attack model. We tested the former locally, the latter locally and in the cloud (simulating a real network environment)

For readers interested in the specific implementation code, you can refer to our GitHub repository: https://github.com/DiWangShePi/NUS-SUMMER. Additionally, we have deployed a server and a simulated website on the cloud: timeless.snakin.top. Readers can experience a simulated attack process on the website, which will help in better understanding the concepts.

Please note that the purpose of these demonstrations is purely educational and aimed at raising awareness about timing attacks and their potential impact on security. Ethical considerations and legal boundaries must be respected at all times. The intention is to promote knowledge and foster a safer online environment.

We encourage readers to explore the GitHub repository and the simulated website for a hands-on understanding of the attack concepts and their implications. If you have any questions or feedback, feel free to reach out to us through the provided contact information in the report.

4.1 Traditional timing attack

Timing attacks are a type of attack method that infers internal information of a computing system by observing the response time of the system on different inputs. This attack exploits the execution time differences of the computing system when processing different input data, thereby deducing secret information during the system's processing, such as passwords, private keys, and more. Timing attacks are commonly applied to cryptographic algorithms and security systems, and they can be used to steal sensitive information or crack passwords.

Attack Principle. We implemented an example of a timing attack based on the login verification function of the target system. The login verification function compares the user's input password with the correct password and uses time.sleep(0.01) during each character comparison to simulate time differences. Attackers utilize these time differences by repeatedly trying different password characters, gradually deducing each character of the correct password.

Experiment Setup. There are two scripts in the experiment setup: Timing_Attack.py and Victim.py.

Timing_Attack.py: This is the attacker's script. It simulates the process of logging into the target system by calling the login function. The cracked_letter function is used to deduce each character of the correct password based on the inferred partial password characters. By continuously sending requests, it calculates the probability of each character appearing in the correct password and selects the next character based on the probability.

```
1  def cracked_letter(cracked, padding):
2      results = {key: 0 for key in
           ascii_lowercase}
3      for _ in range(3):
4          for letter in ascii_lowercase:
5              input_string = cracked + letter
                   + '-' * padding
6              start = time.time_ns()
7              login(input_string)
8              end = time.time_ns()
9              results[letter] += end - start
10     return sorted(results, key=results.get,
           reverse=True)[0]
```

Victim.py: This is the script of the target system. It uses the Flask framework to build a simple web application that simulates the user login process. The login verification function comp compares the user's input password with the correct password and exposes password characters by simulating time differences.

Experiment Results & Practical Analysis. In the end, in our experiment, the attacker successfully cracked the login password of the target system. This example demonstrates a simple timing attack. In practice, timing attacks are a stealthy and effective attack method, especially for the potential time differences that may exist in many real-world systems. Due to network latency, server load, and other reasons, computing systems may exhibit slight time differences when processing different input data, which can lead to the leakage of sensitive information.

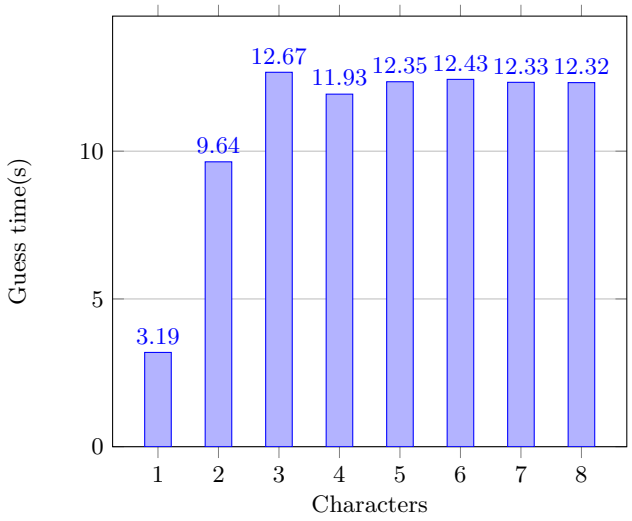


Figure 2: Traditional timing attack

4.2 Timeless timing attack

Timeless timing attacks developed by Van Goethem and Vanhoef[6] eliminate the need for measuring absolute time differences by comparing the order in which simultaneous requests get returned. Since timeless attacks are unaffected by network jitter, they can be executed anywhere globally and can reliably detect much smaller time differences than traditional timing attacks.

Attack Principle. Our goal is to ensure that the server starts processing both requests at exactly the same time, so that the order of responses indicates which request completes processing first. One request is used for a known baseline processing task, and the other request is used for a task that requires the same or different processing times, depending on the secret information we want to know. Due to the request multiplexing feature of HTTP/2, multiple requests sent at the same time will be processed in parallel, and unlike HTTP/1.1, the response will be sent as soon as possible, independent of the order of the requests. Therefore, an attacker can embed two HEADERS frames containing two HTTP/2 requests in a single TCP packet to ensure that they arrive at the same time. Since the headers are compressed and their size is usually around 100-150 bytes, the resulting TCP packet size is significantly smaller than the maximum transmission unit (MTU) observed for all on the Internet .

In this experiment, we utilize the concurrent flow mechanism to construct HTTP/2 messages to achieve the purpose. We use the concurrent streaming mechanism to construct HTTP/2 messages to achieve the purpose, and determine which one is delayed by comparing which of the two AB messages returns first, instead of measuring how much time is spent on which message. At this time, the server, the application has the ability to parallel processing , most of the current can meet this requirement.

In layman's terms, if we can simultaneously send two packets, AB, and they also arrive simultaneously, Timeless Timing attack needs to repeat the operation of sending multiple groups of packets and record their return order. If the server does not experience any delay in processing the two packets, they will be returned immediately because the return order is beyond our control and may be affected by the network during the round trip communication, resulting in a 50% probability of returning in either order. If the server experiences delay in processing packet B, such as performing an additional decryption or query that requires more time than A, B will be returned slightly later than A. Although the response packets will still be affected during communication, we can measure them multiple times to estimate this probability. At this point, the probability of B returning before A is significantly less than 50%, so we can use this probability to determine whether there was a delay in processing the two requests on the server.

Experiment Setup. In our experiments, we set up two machines, an HTTP2 server and an attacker using docker. A flask application called Main.py is running on the HTTP2 server. This code contains a /<secret> route, where the value of <secret> is obtained from the environment variable. The attacker needs to use the timeless timing attack to obtain the <secret> value on the server.

Main.py: The core code for the HTTP2 server is shown in Listing 1, a route handler for a Flask application that takes a parameter called "secret". First, it checks whether the passed "secret" parameter is empty, and if so, returns "WAKUWAKU!". Next, it checks whether the "secret" parameter has the same length as a variable named "FLAG", and if not, also returns "WAKUWAKU!". It then uses the zip() function to compare "secret" with "FLAG" character by character, and returns "WRONG!" if there is a mismatch. If the characters match, a small delay called "TINY_TIME" is added using the time.sleep() function. Finally, if "secret" contains spaces, return "WRONG!", otherwise return "CORRECT!".

```
@app.route("/<secret>")
def check_secret(secret):
    if not secret:
        return "WAKUWAKU!"
    if len(secret) != len(FLAG):
        return "WAKUWAKU!"
    for a, b in zip(secret, FLAG):
        if a == " ":
            continue
        elif a != b:
            return "WRONG!"
        else:
            time.sleep(TINY_TIME)
    if " " in secret:
        return "WRONG!"
    return "CORRECT!"
```

Listing 1: The HTTP/2 Server

Our attack script called Exploit.py is placed on the attacker machine and uses concurrently initiated HTTP2 requests to guess the route on the server. Specifically, it sends a series of GET requests to the target server using an HTTP/2 connection, and then guesses each character of the key based on the difference in response times.

Exploit.py: In the main function exploit(), it first uses the get() function to determine the length of the key, and then uses the time_difference() function to iterate over all possible

characters, guessing each character. In each attack, it uses the asyncio.gather() function to run multiple time_difference() functions concurrently, which use the H2Time class for time difference attacks, and then aggregates the results. After guessing all characters, it uses the get() function to check if the guess was correct and prints out the result.

To demonstrate that the attack is not affected by network jitter, we place the attacker on local and remote hosts for measurements. We set the <secret> string on the server to 11 digits, use the attack script to attack the server, and count the average time of 10 attacks for each character guess in the case of successful guessing.

Experiment Results. The results of our experiments are shown in Fig. 2 and Fig. 3. After comparison, we can find that when guessing the string length of 11, due to the effect of network jitter, the time needed for the local attacker to complete the attack is even higher than that of the remote attacker, but in the end, both of them successfully complete the guessing of the string. This shows that timeless timing attack and not affected by network jitter.

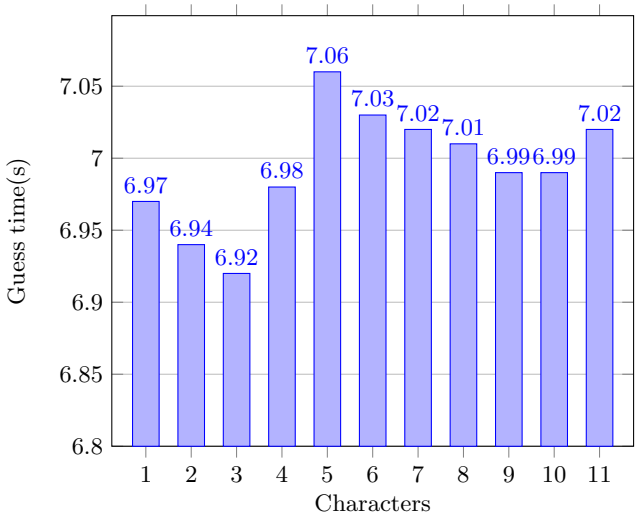


Figure 3: Attacker in local server

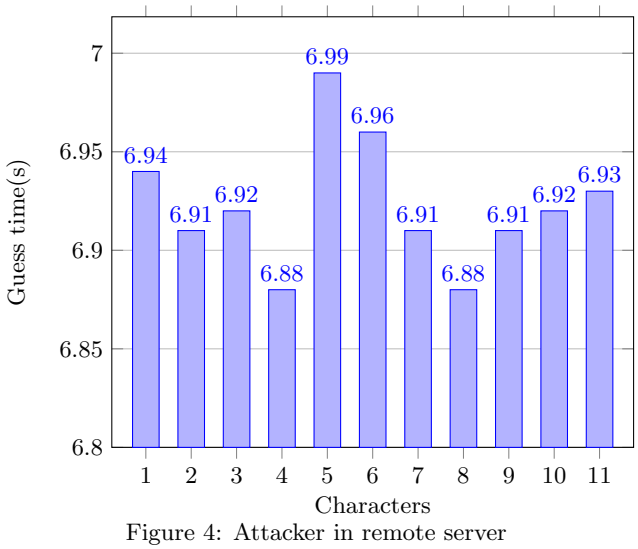


Figure 4: Attacker in remote server

Practical Analysis. In contrast to traditional timing attacks which can be affected by network factors and time differences between the attacker and the server, if the time difference is small it can lead to no attack at all.

Timeless timing attack avoids the effects of network jitter in the communication process by sending out messages at the same time so that they arrive at the same time as much as possible.

4.3 Extension

In the case of HTTP/2 protocol, we can utilize the multiplexed encapsulation protocol to achieve timeless timing attck; however, the current mainstream network environment still uses HTTP/1.1, so in addition to the more restrictive method based on message encapsulation mentioned above, is there any other way to achieve timeless timing attck under HTTP/1.1 protocol?

We can consider the HTTP/1.1 pipeline mechanism, which is one of the ways in which HTTP persistent connections work, and is characterized by the ability of the client to send a new request message before it receives the HTTP response message. So one request message after another arrives at the server, and the server can keep sending back responses.

However, the pipeline mechanism is single-threaded sequential processing, so even if there is a time delay we are difficult to find, in this case we can consider amplification. Since the pipeline is single-threaded, then we use the pipeline single-threaded processing of the same request over and over again, if the difference in execution time between request A and request B is 1ms, then the difference in time between request A*1000 and request B*1000 can be up to 1 second.

But in practice we can't do unlimited scaling. In the actual scenario, the maximum number of requests handled by the pipeline is affected by the configuration of the server middleware, for example, apache will set the maximum number of requests supported by the pipeline to 100 by default if keepalive is enabled.

If the response keepalive only a timeout and no max case means that it does not limit the number of pipeline, that is to say, our amplification scenario exists at this time as long as the unlimited construction of pipeline requests can be unlimited superimposed multiplier.In this way we can use in HTTP/1.1 scenarios.

5. DEFENSE

Now, let's consider defense strategies against this attack. In the previous sections, we have already discussed the basis of this attack: obtaining crucial information by measuring the varying processing times of different requests on the server. Therefore, a very intuitive approach is to eliminate this time difference and ensure that all requests from the same client have the same processing time. This undoubtedly eradicates the foundation of this attack at its core.

However, despite being a highly ideal approach, it is difficult to implement and unacceptable. From the perspective of website service providers, granting all requests the same execution time would significantly increase the server's com-

putational overhead (and might even lead to DDoS attacks under unreasonable settings). Delaying completed requests and waiting for other requests to finish execution would also increase the server's storage space costs. Moreover, such defense strategies would negatively impact the normal user experience. Even if the website owners prioritize security over practicality, achieving uniform execution times for all requests is an extremely challenging task. Especially for complex applications that rely on third-party components.

However, taking a step back and reexamining the premise of this timing attack, we can find a new approach to defense: obfuscating the execution time to prevent attackers from obtaining the actual execution time. One equally intuitive idea is to introduce random perturbations into the execution time, with these perturbations having the same order of magnitude as the original execution time differences. By randomly adding such perturbation times, we can implement a reasonable defense against this attack.

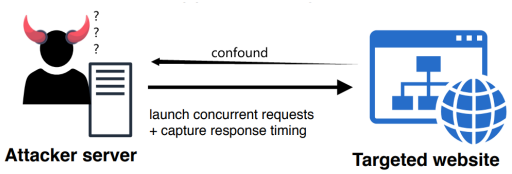


Figure 5: After Using Defense

Due to time and effort constraints, we were unable to measure the real network environment and estimate the cost of this defense strategy. But we managed to implement and prove the effectiveness of it. This part can also be seen in our implementation code.

6. CONCLUSION

In this report, we introduced a side-channel timing attack based on concurrency. In the classical timing attack, an attacker collects a series of response times and uses statistical analysis to try to infer sensitive information. The new example we presented involves inferring sensitive information by measuring the execution time of concurrent tasks. By leveraging the features of the HTTP/2 protocol, two requests can arrive at the target server simultaneously, making the time differences for processing requests completely unaffected by network conditions. This undoubtedly enhances the attack's strength, enabling it to perform as effectively in real-world network environments as it does on local systems.

We implemented a model for the classical timing attack and a model for the new example, and we briefly compared the two approaches. The former can achieve success in a local attack scenario but fails to obtain confidential information in real-world network conditions (On a remote server). On the other hand, the latter is capable of carrying out the attack in both local and real network scenarios, and in the latter case, it does not exhibit significantly lower attack performance compared to the former.

7. ACKNOWLEDGMENTS

In this project, we would like to thank Professor Hugh Anderson for the comprehensive and adequate guidance he provided throughout the course, as well as his full enthusiasm. It was a huge help for us to implement this attack, to try to expand and think about how to defend.

At the same time, I would like to thank my colleagues who worked with me on this project. Even though we started this project from different foundations, I believe we both put in full enthusiasm and energy to complete this project. I think we ended up with a project we can be proud of.

8. REFERENCES

[1] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In CRYPTO, pages 104–113. Springer, 1996.

[2] David Brumley and Dan Boneh. Remote timing attacks are practical. Computer Networks, 48(5):701–716, 2005.

[3] Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In WWW, 2007.

[4] Tom Van Goethem, Wouter Joosen, and Nick Niki forakis. The clock is still ticking: Timing attacks in the modern web. In CCS, pages 1382–1393, 2015.

[5] Mathy Vanhoef and Tom Van Goethem. HEIST: HTTP encrypted information can be stolen through TCP windows. In Black Hat US Briefings, 2016.

[6] Van Goethem, T., Pöpper, C., Joosen, W., Vanhoef, M. (2020). Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections. In 29th USENIX Security Symposium (USENIX Security 20) (pp. 1985-2002).

Exploring Steganagraphy: From LSB to ZKP

Wu Wenhao
School of Computer Science and Technology
Huazhong University of Science and Technology
u202112001@hust.edu.cn

Wang Yibo
College of Computer science
Sichuan University
Chengdu, Sichuan 610207
wangyb0520@gmail.com

Fan Sirui
Faculty of Electronic and Information Engineering
Xi'an Jiaotong University
1714957525@stu.xjtu.edu.cn

Hu Zhuoqi
School of Software Engineering
Beijing University of Posts and Telecommunications
qzhycloud@outlook.com

Pei Haocheng
School of Cyber Science and Engineering
Sichuan University
2020141530105@stu.scu.edu.cn

ABSTRACT

The widespread adoption of images as a medium for information dissemination has given rise to the pertinent issue of image copyright. In response to this evolving scenario, image steganography has emerged as a viable and scholarly approach to address these concerns. LSB is a commonly used steganography algorithm. However, due to its simple steganography strategy, its robustness and security are insufficient. This paper studies the performance of LSB and improves its robustness against noise, cropping, resizing, rotating and grayscale. Experiments showed that the robustness improved greatly. Moreover, we combine cryptography and zero-knowledge proof to improve the security of LSB, which creates a strong steganography system. Our work may play a role in copyright protection, information transmission, etc.

Keywords

Digital Watermark, Robustness, Least Significant Bit, Cryptology, Steganography

1. INTRODUCTION

In the context of the modern internet, the widespread adoption of images as a medium for information dissemination has given rise to the pertinent issue of image copyright. In response to this evolving scenario, image steganography has emerged as a viable and scholarly approach to address these concerns. A steganographic system must provide a method to embed data imperceptibly, allow the data to be readily extracted, promote a high information rate or payload, and incorporate a certain amount of resistance to removal. The objective of digital watermarking is to embed a signature within a digital image to signify origin or ownership for the purpose of copyright protection. Once added, a watermark must be resistant to removal and reliably detected even after typical image transformations such as rotation, translation, cropping, and quantization.[6]

Least Significant Bit (LSB) is a very basic method to embed information in an image, in which least significant bit of the image is replaced with data bit.(See more details in Appendix A) As this method is vulnerable, in this study, we investigated in improving the robustness of LSB against various attacks and tried to improve its security.

2. IMPROVEMENT OF ROBUSTNESS

In the above examination, it can be observed that the LSB algorithm performs poorly in terms of robustness against image manipulation, and in practical applications, image manipulation is a very common attack behavior. Building upon this foundation, we propose improvements to the LSB algorithm to enhance its robustness against image manipulation.

2.1 Noise

Noise is always presented in digital images during image acquisition, coding, transmission, and processing steps. It happens all the time, whether intentionally or unintentionally.[3] There are all kinds of image noise, like Gaussian noise, shot noise and salt-and pepper noise. In our study, noise is added by inverting each bit of each channel of each pixel in an RGB image in a random manner with certain probability. We try to applied techniques to improve the robustness of LSB against this kind of noise attack. Note that everything can be presented in binary format, so we can encode everything into the cover image theoretically. On behalf the brevity of explanation, we apply the techniques to text.

In our study, two methods were combined to improve the robustness against noise: error correcting code and repetition with atatistic analysis.

Error Correcting Code(ECC). Noise causes the data we finally retrieved differs from the original data we encoded. However, Shannon showed that noise need not cause any

degradation in reliability.[8] Error correcting code helps to improve the robustness of the LSB. In our practice, after transforming the text to binary string of length L, we apply Reed Solomon ECC[10] encoder to get a longer binary string and record its length N. This binary string is what we actually write in the cover picture. So we have the following equation:

$$N = S + L \tag{1}$$

, where S is the binary length of all the ECC symbols of Reed Solomon ECC. We can repair $\frac{S}{2}$ errors and S erasures. In our study, S is designated to be 10. When we try to retrieve the original text, we just put the binary string extracted from the stegno image into the ECC decoder and then transform the string to the target format.

Repetition and statistic analysis. Let's say we just write the message only once into the cover image. If more than $\frac{S}{2}$ bits was inverted by the random noise, even with ECC, we would not be able to recover the original information wholly. In fact, this fails to make good use of the whole picture to store information. So, we repeat the same string over and over again, until the image is filled. Since ECC was applied, so the encoded string must be considered as a block, that is, we must record the length of the string for extraction. After extracting the very long binary string written in the image, then we separate it to blocks with length of N. Then we apply the ECC decoder to decode it and record the frequency of each block. Finally, we assume the original string to be the one with highest frequency.

Finally, we analyzed the effect of the proposed methods through experiment.

Evaluation And Results. As for original LSB without ECC and repetition, a single change of a bit can cause the damage. For LSB with ECC only, assume the length of the original binary message to be L, the length of ECC symbols to be S and the noise amount to be p, then we have the probability P_r to retrieve the original message:

$$P_r = \sum_{k=0}^{\frac{2}{S}} C_{(L+S)}^{k} p^k (1-p)^{L+S-k} \tag{2}$$

As for LSB with repetition only and LSB with both repetition and ECC, the algorithm(See Appendix B) is applied to evaluate their robustness against noise.

For each scenario, we varied the length of the message and tried to test the largest noise amount that can be withstood, i.e., can retrieve the original message. Finally, we got the relationship between the message length and the maximum noise that can bear.

Given a message with its length ranging from 8 to 29, we tested the upper bound of the noise amount. As we can see in fig **??**, being equipped with ECC, an improvement of 2%-4% was achieved. With string length ranging from 8 to 29, the upper bound of the noise it can bear ranges from 7% to 18%, this can hardly be achieved by the original LSB or LSB with ECC only. The robustness against this kind of noise was improved significantly.

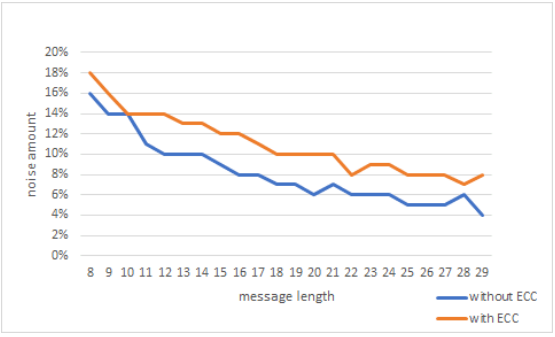


Figure 1: Conparation between repetitive LSB with/without ECC.Given a certain meaasge length, test the noise amount it can bear, the higer the better.

2.2 Cropping

In order to mitigate the potential loss of digital watermark caused by cropping operations, we have devised an algorithm aimed at countering cropping attacks. The crux of this algorithm revolves around leveraging the LSB (Least Significant Bit) technique to iteratively embed the hidden message across the entire image canvas. As a result, even if the image undergoes cropping, as long as a subset of the hidden messages is preserved, they can be extracted, effectively achieving the objective of thwarting cropping attacks.

In essence, each information subset comprises three integral components: the header and footer, the hidden message per se, and the length of the hidden message. Analogous to the concept of packets in network communication, the header and footer serve as markers to delineate the commencement and termination of each information subset during the decoding process. The length component plays the role of a checksum, providing a means to verify the veracity of the extracted hidden message. During the encoding phase, the intended hidden message is amalgamated with the header, footer, and length to form an information subset, which is recurrently inscribed into the entirety of the image utilizing the LSB technique. Careful selection of the header and footer is essential to ensure they do not conflict with the encoding information of the hidden message. For instance, if the hidden message is represented using ASCII codes, the header and footer can be judiciously designated as 0x00 and 0x11, respectively, as these values are devoid of corresponding characters in the ASCII code set. Subsequently, during the decoding process, the Least Significant Bit of the image is extracted, and the header and footer are utilized to demarcate the boundaries of each information subset. Furthermore, the length information is employed to scrutinize the accuracy of the extracted information. In more intricate scenarios, additional sophisticated validation methodologies can be incorporated.

This enhanced LSB algorithm has led to a significant advancement in robustness against cropping attacks. Empirical evidence has validated that even when the image is subjected to a reduction in size by 2%, the hidden information

header	length	hidden message	footer

Figure 2: The format of the information subset

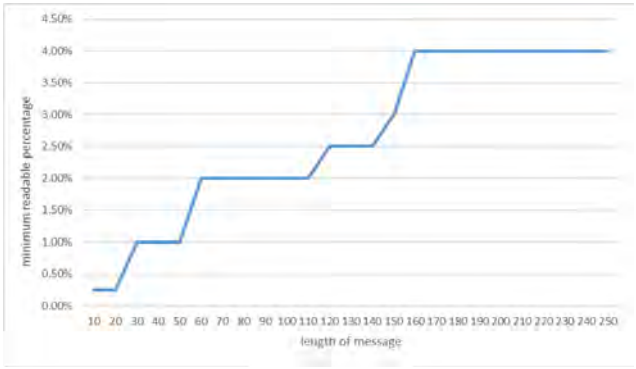


Figure 3: The length of the hidden message affects the minimum legibility percentage.

can be effectively retrieved.

By cropping the image to a certain original size, hidden messages can still be extracted. We refer to this percentage as the readable percentage, while the minimum readable percentage signifies the point where the hidden message cannot be extracted when the image is cropped to a size smaller than this percentage. During the experiment, we observed that the length of the hidden message has an impact on the minimum readable percentage.

By cropping the image to a certain original size, hidden messages can still be extracted. We refer to this percentage as the readable percentage, while the minimum readable percentage signifies the point where the hidden message cannot be extracted when the image is cropped to a size smaller than this percentage. During the experiment, we observed that the length of the hidden message has an impact on the minimum readable percentage.

An experiment was conducted to investigate this phenomenon, and the results are shown in fig 5. From the graph, it can be observed that there is a positive correlation between the length of the hidden message and the minimum readable percentage. This indicates that the longer the hidden message, the weaker its resistance to cropping. However, the experimental results also reveal that even when the length of the hidden message reaches 250 characters (equivalent to 2000 bits), the minimum readable percentage remains below 5%. This suggests that despite the growth in the length of the hidden message, the algorithm's robustness remains strong.

We also combined our improvement of robustness against noise and cropping. In this implementation the length of binary string are not needed, since the recognition of the binary string was realized by recognizing the header and footer. However if we still keep using repetition and statistic to help with decoding, it would take a long time. The code is contained in the appendices.

2.3 Resizing & Rotating

In this part, LSB doesn't seem to work very well. This is mainly due to the following reasons:

1) Changes in Pixel Values: When we scale or rotate an image, the original pixel values change. This is because these operations often involve interpolation or resampling processes, which generate new pixel values to fit the new size or direction of the image. These new pixel values may not retain the value of the original pixel's least significant bit, resulting in the loss of the hidden information.

2) Changes in Image Structure: Scaling or rotating an image changes the arrangement of its pixels. This structural change disrupts the organization of the hidden information in LSB steganography. Even in some situations where the values of the least significant bits can be retained, the hidden information cannot be correctly interpreted due to the changed pixel arrangement.

The Fourier Transform[5] is a mathematical tool that transforms one function of a real variable into another. In the context of an image, it translates the spatial domain representation into the frequency domain. The idea of applying Fourier Transform in steganography, specifically in situations involving scaling and rotation, comes from two primary factors:

1) Robustness against Transformations: Fourier Transform manipulates the frequency representation of an image, which is generally more robust to common image processing operations like scaling and rotation. When information is hidden in the frequency domain, it tends to resist such transformations better than when hidden in the spatial domain (as in the case of LSB steganography).

2) Invisibility of Changes: Changes made in the frequency domain are usually less perceptible in the spatial domain. Thus, hiding information in the frequency domain can potentially be less noticeable to human eyes than making changes directly to the pixels in the spatial domain.

Now let's explain the whole process of how to hide information in images through Fourier transform.

1. First, we perform Fourier transform on the original image to obtain its frequency domain image. Figure 4 shows the spatial and frequency domain plots of this image.

2. We add a readable watermark to the transformed frequency domain image. In this example, we set the watermark text to "Hi, DOTA", and place the watermark in the lower right corner of the frequency domain image. Figure 5 shows the comparison of the frequency domain image of the original image and the watermarked frequency domain image. We can see that there is a "Hi, DOTA" watermark in the lower right corner of the watermarked frequency domain image.

Figure 4: Original image (left) and transformed image (right)

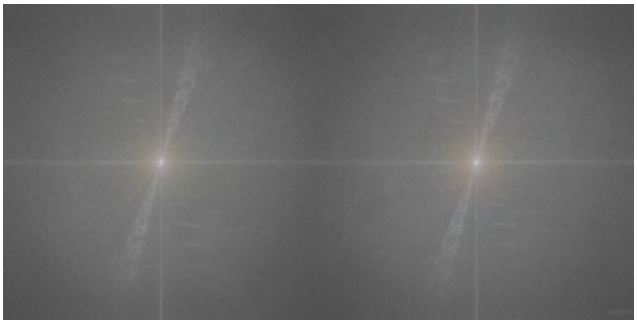


Figure 5: The frequency domain image of the original image (left) and the frequency domain image after watermarking (right)

3. We perform inverse Fourier transform on the frequency domain image with the watermark added and obtain the encrypted image. Figure 6 shows the comparison between the original image and the encrypted image. We can see that since the magnitude spectrum of the original image has been modified by adding the watermark layer, the color of the encrypted image looks darker.

4. Perform Fourier transform on the encrypted image, and we can get the frequency domain image of the encrypted image. In this image, we can see the previously added watermark in the lower right corner, which shows that we have successfully hidden the message "Hi, DOTA" in the encrypted image. Figure 7 shows the previous watermarked frequency domain image and the frequency domain image obtained after performing Fourier transforming on the encrypted image. We can find that the watermark is not as clear as before. This is because some image parameters are lost due to the addition of the watermark layer in the process of inverse Fourier transform and Fourier transform.

Figure 6: Original image (left) and encrypted image (right)

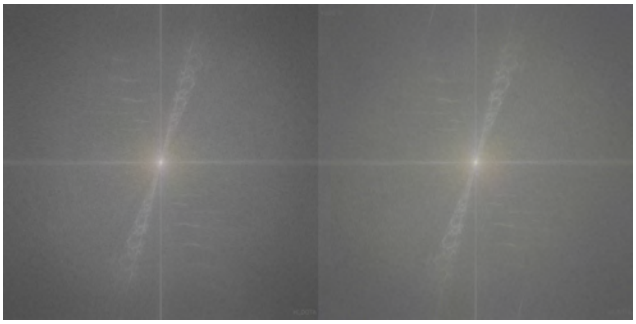


Figure 7: The watermark on the frequency domain of the original image (left) and on the frequency domain of the encrypted image (right)

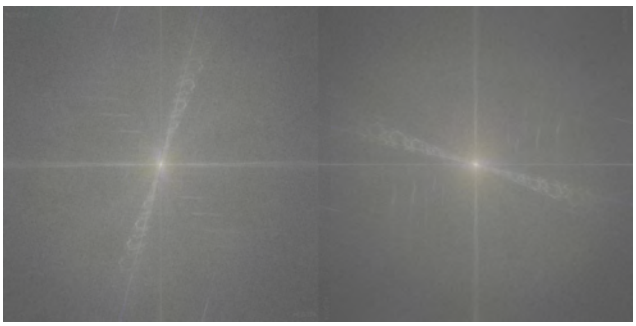


Figure 8: After FFT on the 25%-reduced encrypted image(left) and on the 90-degrees-clockwise-rotated encrypted image(right)

But the good news is that after such processing, no matter whether we resize(not too much) or rotate the encrypted image, we can get the information hidden in it by performing Fourier transform on the encrypted image. Figure 8 shows the result image of Fourier transform when we reduce the encrypted image by 25% and rotate the encrypted image 90 degrees clockwise. We can find that the watermark on the image after the Fourier transform of the encrypted image that has been reduced by 25% is not as clear as the one at the right of Figure 7 . This is because when the encrypted image is resized, we lose some image information, and the lost image information may contain watermark information. However, the clarity of the watermark after the rotation process is the same as the original watermark. This is because the information contained in the image as a whole has little change during the rotation process. We just rotated the image 90 degrees clockwise, so the frequency domain image just also rotated 90 degrees clockwise, and we can see the same watermark "Hi, DOTA" in the lower left corner.

However, the feasibility of using the Fourier Transform in steganography comes with its own challenges:

1) Complexity: The process of converting an image to the

Figure 9: 0.5 partial grayscale (left) and complete grayscale (right)

frequency domain, embedding the information, and then converting it back to the spatial domain is more complex compared to simple LSB steganography.

2) Noise Susceptibility: While the frequency domain representation is more robust against operations like resizing and rotating, it might be more susceptible to noise and compression, which can disrupt the hidden information.

3) Amount of Data: Embedding information in the frequency domain might limit the amount of data that can be hidden, compared to hiding it directly in the pixel values in the spatial domain.

In conclusion, while the Fourier Transform offers a promising avenue for steganography that can resist image resizing and rotating, it also brings its own set of challenges that need to be addressed.

2.4 Grayscale

In sRGB (standard RGB color space) , the grayscale of a original image is such a image that preserve the same perceptual luminance and RGB combination of the original one. Partial grayscale are some of the linear combination of the original image and its grayscale, which preserve the perceptual luminance as well, and are between the original image and grayscale. The image (9) above show examples of grayscale and partial grayscale.

Specifically, we assume that all images are made up of RGB pixels which take an integer value in $[0, 255]$ for every channel. For a pixel, its linear luminance is defined as: $Y = 0.2126R + 0.7152G + 0.0722B$. [1] Holding Y unchanged, make $R = G = B$, we get the grayscale pixel of the original pixel. Obviously, at this time, $R = G = B = Y$, where Y is at $[0, 255]$ (so a grayscale pixel only needs one channel to store). Let $R' = kY + (1-k)R$, $G' = kY + (1-k)G$, $B' = kY + (1-k)B$, and then the $R'G'B'$ is the k grayscale pixel of the original pixel, where k is in $[0, 1]$.

Now, the steganography strategy is divided into two parts.

[1]In fact, the definition of RGB's coefficients is varied, and we only consider this specific linear combination here, which is called Rec.709 standard[1]. But the algorithm we are about to propose will be easily extended to other linear combinations.

Basically, when $k \neq 1$, the information can be restored through a "inverse function" of the grayscale transformation relatively easily. The derivation of all the mathematical formulas are given in the appendix C.

Now, the hardest part is the case of $k = 1$, that is, a complete grayscale image. First of all, we can't restore it because it compresses the three-dimensional information into one dimension in a very rough way, and each channel can only contribute a little influence.

Therefore, we can only retain some information by modifying the original image to try to influence the final grayscale image in a predictable way somehow.

First of all, we have to prove that it is feasible from the perspective of information theory. The question now is, is there a RGB combination that cannot be distinguished after graying, no matter what modification is made to the last bit of the three channels?

If the answer is yes, then our goal of steganography can not be achieved, because once these RGB combinations, which serve as examples for the formula, appear, their performance in the grayscale image is locked and cannot embedd any message to it.

Luckily, the answer is no. We can programe to check every possible combination and see whether there's a modification that influence the grayscale result, and it turned out to be correct. The whole mathematical derivation is also in appendix C.

Therefore, we can manipulate all the pixels as needed so that each pixel can display the message we want in the last bit after it is grayed.

3. COMBINING WITH *CRYPTOGRAPHY*

The above improvements are based on the premise of considering only image manipulation. Taking this into account, we further focus on enhancing the overall security and credibility of the encryption system to withstand decryption and tampering attempts against the digital signature, thereby achieving true security of digital signatures.

3.1 Digital signature system

If we directly embed steganographic information into the carrier without any processing, attackers can eavesdrop on the transmission and potentially retrieve the hidden information by reasonably inferring the steganographic technique used by the sender. This compromises the invisibility of the hidden information, allowing the attacker to gain unauthorized access. Additionally, the attacker could employ relevant techniques to remove our digital watermark and replace it with their own information, falsely claiming that the information was published by them, leaving us powerless against such impersonation.

To address these issues, we develop a two-layer encryption method. This method involves applying different encryption techniques to the data to be hidden, providing dual protection. Firstly, we use AES encryption to ensure the confidentiality of the information. Simultaneously, we apply

digital signatures on top of AES encryption to authenticate the sender's "signature" on the information.

Our encryption scheme consists of two layers: the AES encryption layer and the digital signature layer. The entire encryption process is illustrated in the diagram below, and this part describes each module separately.

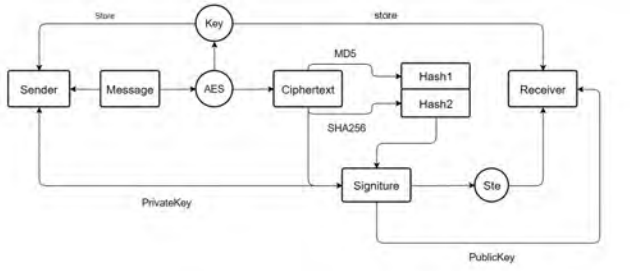


Figure 10: Stucture

In AES layer, we employ the AES-128-CBC encryption method. Both parties will first agree on a 16-character key, which will be kept secure by the information sender and receiver. When the receiver extracts the ciphertext from the image carrier, they can use the key to decrypt the ciphertext and retrieve the original message. It's worth noting that AES is currently considered a relatively secure algorithm with no efficient decryption methods other than brute force (which is infeasible). Hence, our encrypted information is relatively secure.

This implementation employs the Python Crypto.Cipher library to achieve the desired functionality. The program utilizes the AES algorithm in CBC mode to encrypt the plaintext message securely. The sender can readily provide the plaintext message for encryption, alongside the preestablished key and IV, resulting in the generation of ciphertext ready for secure transmission. Similarly, the receiver can decrypt the received ciphertext back to its original plaintext form using the identical key and IV for seamless decryption.

As for digital signature, we start with risk. Potential Risk: While the basic system employs hashing primarily for verification, the most significant risk lies in collision vulnerability, where different input data might generate identical hash values. Attackers could forge files with identical hash values to achieve the same verification effect.

Countermeasure: To mitigate this risk, we employ double hashing by applying both MD5 and SHA256 hash functions to the information, combining the resulting hash values and encrypting them with the sender's private key to form the digital signature. This combined approach significantly reduces the likelihood of collisions, further enhancing the authority of the digital signature.

This module was implemented using the 'cryptography' library to achieve its intended functionality. Upon initialization, the program automatically generates a key pair to facilitate subsequent signature operations. The chosen public exponent of 65537 and key size of 2048 were employed during key pair generation, as they are widely recognized as secure values.

When the sender inputs a message for processing, the program utilizes two distinct hash functions to create digital signatures, which are subsequently returned to the user. The receiver can then employ the message, the sender's public key, and the two signatures for the purpose of verification.

In the implemented solution, the sender can conveniently input the message, AES key, and AES initialization vector (IV). Subsequently, the system generates the ciphertext using AES-128 in CBC mode, alongside a keypair comprising a private and public key. Furthermore, the system generates two versions of a signature for the sender to conceal within a carrier medium. This approach ensures the confidentiality of the message through AES encryption and enables data integrity and authenticity through the use of digital signatures. The receiver is required to have access to the following elements for verification and decryption purposes: the ciphertext, two versions of the signature, and the public key.

Our approach offers a robust solution to protect steganography-based information by combining the strengths of cryptography and steganography. However, we acknowledge that certain aspects, such as comprehensive key management, authorized digital signature key pairs, and secure network transmission, require further research and consideration.

3.2 Zero-knowledge-proof system

Here's another method called zero-knowledge-proof to prove that you are the owner of the image.

Zero-knowledge-proof is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true, while avoiding conveying to the verifier any information beyond the mere fact of the statement's truth.[11]

We are not going to cover this area in detail, but will use it as a tool to prove the ownership of the image. In particular, we are going to use graph isomorphism as the proof basement for the time being.

Given two graphs $G_1(V_1, E_1), G_2(V_2, E_2)$, if there is a bijection m from G_1 to G_2 such that

$$\forall x, y \in V_1,\ (x, y) \in E_1 \leftrightarrow (m(x), m(y)) \in E_2, \qquad (3)$$

and then G_1 and G_2 are said to be isomorphic, written as $(G_1, G_2) \in GI$, as shown in the figure (11).

So far, no polynomial time algorithm for the graph isomorphism problem has been found yet. It is a special case of the hidden subgroup problem just like factoring and discrete logarithm, and is not proved to be NP-hard yet.

Let's assume that one day you get involved in a lawsuit over the ownership of a image. You, the owner of image, are called P. The judge is called Q.

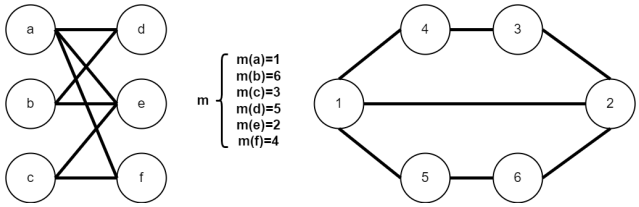


Figure 11: 0.5 partial grayscale (left) and complete grayscale (right)

P embedded two huge isomorphic graphs (G_0, G_1) in advance in the image. P knew exactly about the isomorphic bijection m from G_0 to G_1. P revealed the graphs in the court, and now needs to prove his ownership of the image in front of Q.

Here's the mechanism:

1. P: create a new graph, which is isomorphic to G_1, called H, and show H to Q. P knows the isomorphic bijection n from G_1 to H.
2. Q: pick a random $\alpha \in \{0, 1\}$ and show α to P.
3. P: show
$$\beta = \begin{cases} n & \text{if } \alpha = 0 \\ nm & \text{if } \alpha = 1 \end{cases}$$
to Q.
4. Q: reject if $H \neq \beta G_\alpha$.
5. Repeat 1∼4 until Q trust P. (count as a new round)

In this mechanism, the probability that P really knows m is $1-(\frac{1}{2})^n$ after n rounds of interaction from the perspective of Q. Therefore, Q can trust that P is the owner of the image.

Firstly, this zero-knowledge proof system does not need to publish any public keys like a digital signature system.

Secondly, the same pair of graphs can be reused for several times and it is pretty easy to create more huge isomorphic graphs if you want. Unlike limited primes used as exclusive public key, isomorphic graphs offer much more key resource.

Last but not least, problems like isomorphic graphs has the potential of good robustness. The isomorphic graphs can be directly drawn in the media (e.g., the fourier transform image of every frame of a movie). After malicious transformation, the graphs might lose some edges or vertexs. However, they're still the approximation of the original graphs and hold an approximate isomorphic relation. The image owner can still do the proof within some approximation rate, which is also convincible from the perspective of computational complexity[2].

4. CONCLUSIONS

In this article, we started with the basic LSB steganography technique and explored its performance. We conducted research and improved its robustness in various aspects, including cropping, noise,resize, and grayscale. Meanwhile, we investigated the potential of combining steganography with cryptography, presenting two effective integration schemes to enhance the security of steganography.

Although the LSB algorithm may be considered somewhat outdated, currently, the frontier research direction in digital watermarking revolves around frequency domain-based adaptive watermark embedding. However, we believe that our research can provide insights and references for new steganography algorithms. Furthermore, there is still room for improvement in our work. We envision the potential for combining robustness strategies against various attacks and comprehensively enhancing the algorithm's performance.

5. REFERENCES

[1] Bt.709 : Parameter values for the hdtv standards for production and international programme exchange. 2015. Retrieved July 22, 2023.

[2] V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. Approximate graph isomorphism. pages 100–111, 2012.

[3] A. K. Boyat and B. K. Joshi. A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489*, 2015.

[4] H. S. Fairman, M. H. Brill, and H. Hemmendinger. How the cie 1931 color-matching functions were derived from wright-guild data. *Color Research & Application*, 22(1):11–23, 1997.

[5] J. S. Lim. Two-dimensional signal and image processing. *Englewood Cliffs*, 1990.

[6] L. M. Marvel, C. G. Boncelet, and C. T. Retter. Spread spectrum image steganography. *IEEE Transactions on image processing*, 8(8):1075–1083, 1999.

[7] N. Provos and P. Honeyman. Hide and seek: An introduction to steganography. *IEEE security & privacy*, 1(3):32–44, 2003.

[8] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[9] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[10] S. B. Wicker and V. K. Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[11] Wikipedia contributors. Zero-knowledge proof — Wikipedia, the free encyclopedia. 2023. [Online; accessed 22-July-2023].

APPENDIX

A. LEAST SIGNIFICANT BIT

Least Significant Bit (LSB) is an algorithm used to embed information within an image. Its algorithmic concept is straightforward and widely adopted. In modern images, each bit comprises three channels representing the colors red, green, and blue, with each channel allocated one byte. [4]The fundamental principle of the LSB algorithm involves modifying the least significant bit of each byte to facilitate the insertion of a hidden message.[7] This approach renders

Figure 12: Images before and after embedding using the LSB algorithm.

the information imperceptible to the human eye and indistinguishable through visual inspection before and after applying steganography. It exhibits a high degree of invisibility and ensures a significant likeness between the images before and after the information is embedded using this algorithm.

A good steganography technique requires achieving invisibility, ensuring minimal differences between the images before and after the embedding process, and exhibiting robustness against image manipulation. To assess the consistency between the images before and after Steganography, Peak Signal to Noise Ratio (PSNR) and structural similarity (SSIM)[9] are introduced as evaluation metrics. To test the robustness against image manipulation, the images are subjected to operations such as blurring, rotation, resizing, cropping, and the addition of noise.

Based on the test results mentioned above, it can be observed that the LSB algorithm demonstrates excellent invisibility, as indicated by its outstanding performance in both PSNR and SSIM metrics. This signifies that the differences between the images before and after information embedding are minimal, showcasing the algorithm's remarkable capability in concealing information. However, there is still room for improvement concerning the Robustness against image manipulation. In the subsequent phase, we will propose corresponding enhancement algorithms tailored to address image manipulation challenges.

B. ALGORITHM OF LSB FOR EVALUATION OF ROBUSTNESS AGAINST NOISE

Algorithm 1: Evaluation algorithm

Input: cover image: *img*; message to write: *msg*
Output: stegno image: *s_img*; , maximum noise can withstand: *max_noise*

```
encoded_msg := encode(msg);
for len(b_msg)<len(img) do
    b_msg:=b_msg+ECC_msg
end
s_img:=Write b_msg in img;
while noise_amount:=0 to 100 do
    noised_img:=add_noise(img,noise_amount);
    decoded_msg:=decode(noised_img);
    if decoded_msg!=msg then
        max_noise:=noise_amount;
        break;
    end
end
return s_img, max_noise
```

C. MATHEMATICAL DERIVATION ABOUT GRAYSCALE

C.1 Preparation

Formally,

$$Y(R,G,B) = 0.2126R{+}0.7152G{+}0.0722B, \ abbreviated\ as\ Y, \tag{4}$$

$$gray(k,R,G,B) = (kY{+}(1{-}k)R, kY{+}(1{-}k)G, kY{+}(1{-}k)B), \tag{5}$$

therefore,

$$(R',G',B') = gray(k,R,G,B). \tag{6}$$

Especially,

$$gray(1,R,G,B) = (Y,Y,Y). \tag{7}$$

For an original image, convert all the pixels into their grayscale pixels so that we get a grayscale image, and convert all pixels into their k grayscale pixels so that we get a k grayscale image (a partial grayscale image).

It can be noted that for any pixel, it has the invariance of Y; for all pixels, it has the invariance of k. This inspires us to use this property to store information.

C.2 Derivation

When $k \neq 1$,

the k grayscale image can actually be reversed through a sort of inverse function of $gray()$. (Loosely, we still call it $gray^{-1}()$).

That is to say,

$$\begin{aligned}(R,G,B) &= gray^{-1}(k,R',G',B') \\ &= ((R'-kY)/(1-k), \\ &\quad (G'-kY)/(1-k), (B'-kY)/(1-k))\end{aligned} \tag{8}$$

Obviously we know R'G'B'. By formula (4), we can calculate Y easily. For k, we need a little trick. We introduce an "information pixel", which can be chosen from the first pixel or any other pixels that can be easily detected both before and after grayscale. The RGB of the information pixel in the original image is defined to be $(0, 255, 0)$. Therefore, after the grayscale transformation, the information will turn into $(182k, 255 - 73k, 182k)$. So we check this pixel in the grayscale, and we can calculate the k immediately. Then, we can restore the message embedded in the original image for we have already known all the values we need to calculate the formula (8).

When $k = 1$,

Table 1: LSB performance evaluation

Invisibility	Consistency		Robustness against image manipulation				
	PSNR	SSIM	blurring	rotation	resizing	cropping	noise
low	51.14dB	0.9986	low	low	low	low	low

if $\oplus$ stands for bitwise exclusive OR (which is consistent with the situation in programming languages such as C), then the question is whether

$$\begin{aligned} &\exists\ (R,G,B)\ \forall\ a,b,c \in \{0,1\}, \\ &gray(R,G,B) = gray(R \oplus a, G \oplus b, B \oplus c). \end{aligned} \tag{9}$$

And the answer is no because of a program verification for all all possible (R,G,B,a,b,c) combinations, which implies:

let $low(x) = x\&1$ (which is LSB),

$$\begin{aligned} &\forall\ (R,G,B)\ \exists\ a,b,c \in \{0,1\}, \\ &\{low(Y(R,G,B), low(Y(R \oplus a, G \oplus b, B \oplus c))\} = \{0,1\}. \end{aligned} \tag{10}$$

D. EVALUATIONS ON FOURIER TRANSFORM METHOD

D.1 The value of alpha

In this part, We will illustrate the effect of watermark strength on the watermarked frequency domain image and on the encrypted image.

In our method, we use the variable 'alpha' to control the visibility of the watermark. When alpha=0, the watermark is completely invisible; when alpha=1, the watermark will cover the original image. Figure 13-15 shows the watermarked frequency domain image and the encrypted image obtained after the inverse Fourier transform when alpha is equal to 0, 0.5, and 1. We can see that as the alpha increases, the watermark becomes more and more obvious, while the original image becomes less and less obvious. Therefore, a key point is that we need to find an alpha value that can not only generate a relatively visible watermark, but also ensure that the original image is not disrupted to the greatest extent.

In the case in our article, we use an alpha value of 0.1, a value that produces a visible watermark without disrupting the original image too much.

D.2 Evaluation of the resizing and rotating

We have mentioned in the article that when an image containing a watermark is rotated, the watermark will not change significantly, because the rotation preserves each pixel of the image intact, and only changes their position. However, when we resize an image, the pixels of the image usually increase or decrease, which may destroy the watermark to a certain extent. So we need to evaluate the maximum adjustment range that the watermark added by the Fourier transform method can bear.Take the alpha value of 0.1 in our article as an example. In this case, the image containing the watermark can be reduced to at most 50% of the original size and still show a readable watermark after Fourier

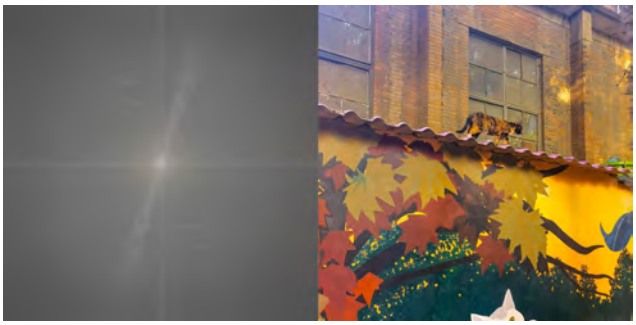


Figure 13: Watermarked frequency domain image(left) and encrypted image(right) when alpha = 0

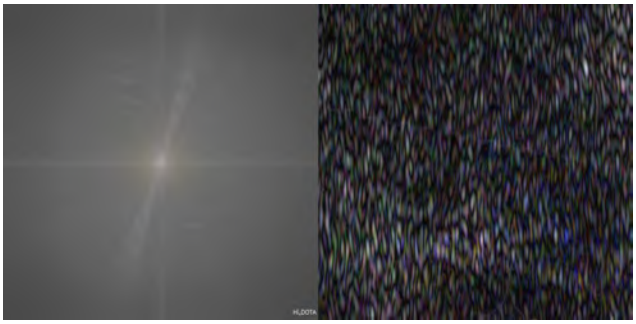


Figure 14: Watermarked frequency domain image(left) and encrypted image(right) when alpha = 0.5

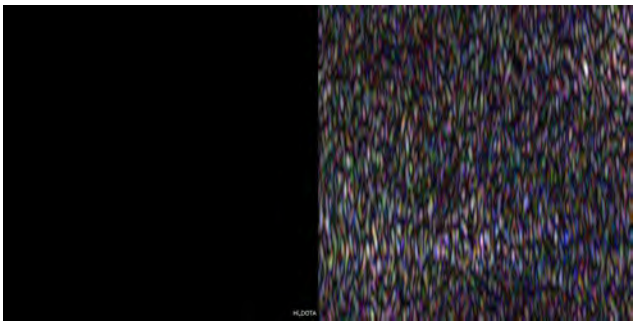


Figure 15: Watermarked frequency domain image(left) and encrypted image(right) when alpha = 1

transform. At the same time, since the pixels are increased when the image is magnified, the watermark will move from the original lower right corner to the center of the frequency domain image after the magnification, and this movement is not linear. At the same time, since the center of the frequency domain image represents low frequency, it is usually very bright, so when the watermark is moved to be covered by the brightness of the center, the watermark will not be visible any more, and the moving distance corresponding to this magnified level is different for each picture, so it is hard to be measured with a specific value. In our case, even though the image is magnified by 4 times, we can still see the watermark in its frequency domain image.

D.3 The Steps of Digital Signature

1. Signature Generation: Firstly, the ciphertext obtained from the first layer is hashed to create a data digest. Then, this digest is encrypted using a private key to generate the digital signature.

2. Signature Transmission: The ciphertext and the generated digital signature are sent together to the recipient.

3. Signature Verification: The data receiver uses the public key to decrypt the received digital signature, obtaining the digest.

4. Digest Comparison: The data receiver hashes the received original data, generating a new digest.

5. Comparison Result: The recipient compares the decrypted digest with the newly computed digest. To check if data has integrity, authenticity, and trustworthy source.

Code Vulnerability Detection using Source Code and AST with Deep Learning

Guan Batu*
Huazhong University of Science and Technology
1037 Luoyu Rd.
Wuhan, China
batuguan@hust.edu.cn

Zhou Junyu†
Sichuan University
Shuncheng Ave 252
Chengdu, China
maxzhou6174@gmail.com

Xiao Yingbo‡
Xi'an Jiaotong University
No. 28 West Xianning Road
Xi'an, China
nightgoodlsan@gmail.com

Zhang Zi'en§
University of Electronic Science and Technology of China
2006 Xiyuan Avenue
Chengdu, China
949911292@qq.com

Liu Dingming¶
Sichuan University
Shuncheng Ave 252
Chengdu, China
1172012877@qq.com

ABSTRACT

This paper presents a comparative study on the use of Python source code and Abstract Syntax Trees (AST) for code detection under deep learning. Traditional code detection methods, such as manual review and rule matching, are inefficient and error-prone. Deep learning, however, can automatically extract features and patterns from code, enabling automated code detection and defect discovery. The paper reviews the status and progress of related fields, the application of deep learning in code detection, and the characteristics of Python source code and AST. It then compares the performance of using Python source code and AST for code detection under deep learning through data pre-processing, model design, and experimental analysis. The study concludes that Python source code is more suitable for training machine learning models than ASTs due to its readability, interpretability, and the semantic information it contains. The research results are significant for improving the accuracy and efficiency of code detection and have certain reference value for promoting the application and development of deep learning in software engineering and security fields. Our project code can be found at vulnerability-test

*This author initiates this idea and writes training code.

†This author parses python source code into AST.

‡This author is responsible for writing the paper.

§This author uses NVIDIA RTX 3060 to train the dataset.

¶This author designs video and poster.

General Terms

Computer security, Machine learning

Keywords

Deep learning, Code vulnerability detection, AST, Software security

1. INTRODUCTION

Code detection is a critical technology in software development and security, which helps developers identify and fix vulnerabilities and defects in code, ensuring the quality and security of software. Traditional code detection methods rely mainly on manual review and rule matching, which are inefficient and error-prone, and cannot meet the growing detection demands and complexity.[3]

In recent years, deep learning, as a neural network-based machine learning method, has been widely used in code detection. Deep learning can automatically extract features and patterns from code, enabling automated code detection and defect discovery. Under the framework of deep learning, converting code into an abstract syntax tree (AST) can better express the structure and semantics of code, improving the accuracy and precision of detection.

This paper aims to compare the advantages and disadvantages of using Python source code and AST for code detection under deep learning. Firstly, we reviewed the research status and progress of related fields and problems, analyzed the application status and problems of deep learning in code detection, as well as the characteristics and applications of Python source code and AST. Secondly, we adopted data preprocessing, model design, and experimental analysis methods to compare the performance and effects of using Python source code and AST for code detection under deep learning. Finally, we evaluated the reliability and practicality of the experimental results, summarized the research results and contributions, and pointed out the limitations

and future work of the research.

The main contribution of this paper is to compare the training methods of Python source code and *AST*, analyze the advantages and disadvantages of deep learning in code detection, and provide prospects for its development direction and application prospects. Our research results are of great significance for improving the accuracy and efficiency of code detection, and have certain reference value for promoting the application and development of deep learning in software engineering and security fields.

2. BACKGROUND

2.1 Related research on code detection

Code detection is an important issue in the field of software engineering and security, and related research covers **code defect detection**, **code security detection**, code performance detection, and code detection tools, which have extensive research and application value.

Code detection contains research on code defect detection and code security detection and other related aspects. Among them, code defects refer to errors, loopholes or bad practices in the program, which may lead to errors in program operation or security problems. Code defect detection refers to the identification and repair of defects in code through static or dynamic analysis techniques. Code security refers to vulnerabilities, risks, or attack surfaces in a program that could lead to an attack or misuse of the program. Code security detection refers to the identification and repair of security problems in code through static or dynamic analysis techniques.

Code inspection is very important for software development and maintenance, which can help to improve code quality, reduce maintenance costs, strengthen code security, and improve team collaboration efficiency.

2.2 Deep Learning in Code Inspection

The application of deep learning in code detection can help developers find problems in code more quickly and accurately, and improve the efficiency and quality of code detection. Deep learning technology can build efficient models to detect errors, defects, vulnerabilities, irregularities, and other problems in the code by learning and training on a large amount of code, and fixing or improving them automatically.

Deep learning can achieve automatic detection of code defects by learning and recognizing the syntax, structure and semantics of the code. Deep learning can realize automatic detection of code security problems by learning and identifying security vulnerabilities in the code, including deep learning for vulnerability detection, deep learning-based code security repair and other aspects. This deep learning-based code detection method can effectively reduce the time and workload of code detection while improving the quality and maintainability of the code. [5]In addition, deep learning can also improve the robustness and reliability of code by analyzing and modeling the code, discovering potential problems in the code, and preventing errors or exceptions in the code in advance.

Therefore, deep learning is significant and useful for code inspection, which can help developers better manage and maintain their code, and improve software quality and user experience.

2.3 Source Code and AST features

An Abstract Syntax Tree (AST) is a tree structure that represents the syntactic structure of code written in a formal language. It provides an abstract representation of the code, capturing its structural or content-related details rather than every detail appearing in the real syntax.

In Python, source code is a textual representation of a Python program, and an AST is a higher-level representation of the code that is generated by the Python compiler during the parsing phase. The AST captures the structure of the code in a machine-readable format, making it easier to analyze and programmatically manipulate the code. The AST contains nodes representing various syntactic structures of the Python language, such as functions, classes, expressions, statements, and literals. Each node in the AST has a type and a set of attributes that provide information about the corresponding syntactic structure.

There are advantages to using source code and using AST for deep learning. Source code can be more directly used for training and learning, making the model closer to the actual application scenario. And AST can be more convenient for code analysis and optimization, avoiding some details in the source code interfering with the training and learning of the model. In the field of automatic code generation, python source code is more suitable for model learning and generation. In the field of code detection and code quality analysis, AST can be more conveniently used for code analysis and detection, as well as code conversion and optimization.

3. METHODOLOGY

3.1 Collection and pre-processing of datasets

Python has a self-contained AST library, and it can parse functions even though it cannot be executed, which is just suitable for our python vulnerability datasets.

The `ast` module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like.[2]

For example, the function `one_plus_two` might look like this:

```
def one_plus_two():
  return 1 + 2
```

one_plus_two.py

and Figure 1 shows one of the version of its AST.[1]

[1]https://i0.wp.com/pybit.es/wp-content/uploads/2021/05/tree-sketch.png

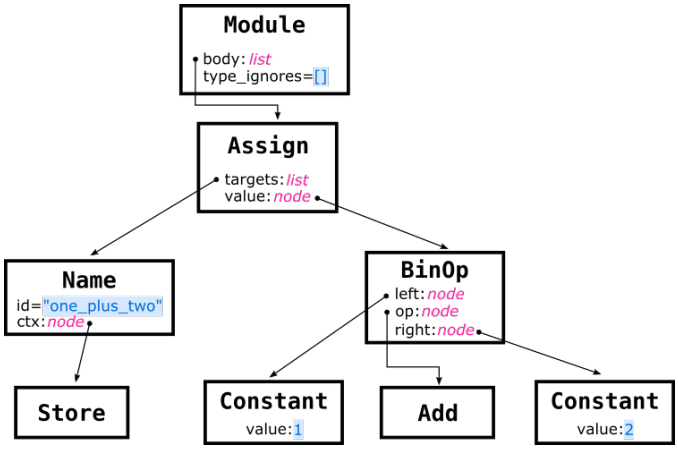


Figure 1: AST of function one_plus_two

However, this type of AST contains too much information about structure, such as `body`, `target`, `value`, which diffuses the dataset and might decrease the efficiency of trained model. Thus, we need to cut off these structure messages to increase entropy of each AST.

In this essay, we keep and print the `Name`, `Str`, `FunctionDef`, `arg`, `Num`, `BinOp`, `Classdef` and `Attribute` information of each node, take this function as an example:

```
def string_to_classes(s):
    if (s == '*'):
        c = sorted(six.iterkeys(
    tracked_classes))
        return c
    else:
        return s.split()
```

string_to_classes.py

By applying the method above, we can generate the AST looks like this:

```
Module
  FunctionDef: string_to_classes
    arguments
      arg: s
    If
      Compare
        Name: s
          Load
        Eq
        Constant: *
      Assign
        Name: c
          Store
        Call
          Name: sorted
            Load
          Call
            Attribute: iterkeys
              Name: six
                Load
              Load
            Name: tracked_classes
              Load
      Return
        Name: c
          Load
      Return
        Call
          Attribute: split
            Name: s
              Load
            Load
```

The last step is to cut off the indentations, substitute the newline character into an actual symbol `\n` to compress the three in only one line, encapsulate the AST in json with other keywords such as `label`, `info`, etc.

The one piece of vanilla dataset looks like this:

```
{"function": "\n\ndef string_to_classes(s)
    :\n    if (s == '*'):\n        c =
    sorted(six.iterkeys(tracked_classes))\
    n        ret
urn c\n    else:\n        return s.split()
    \n", "label": "Correct", "info": "
    dataset/ETHPy150Open PyTables/PyTables
    /tables/utils
.py string_to_classes/original"}
```

The one piece of transferred dataset looks like this:

```
{"function": "Module: \nFunctionDef:
    string_to_classes\narguments: \narg: s
    \nIf: \nCompare: \nName: s\nLoad: \nEq
    : \nConstant: *\nAssign: \nName: c\
    nStore: \nCall: \nName: sorted\nLoad:
    \nCall: \nAttribute: iterkeys\nName:
    six\nLoad: \nLoad: \nName:
    tracked_classes\nLoad: \nReturn: \
    nName: c\nLoad: \nReturn: \nCall: \
    nAttribute: split\nName: s\nLoad: \
    nLoad: ", "label": "Correct", "info":
    "dataset/ETHPy150Open PyTables/
    PyTables/tables/utils.py
    string_to_classes/original"}
```

In this way, we can transfer the whole dataset of vulnerable functions into their AST form, better for training the model in the next step.

In our experiment in a small dataset, 2682 of 3606 of python functions can be correctly generated. [2]

3.2 Model design

This paper explores the potential application of natural language processing (NLP) models in addressing classification

[2]The actual parse code stores here: `https://www.github.com/junyu33/python-ast`

tasks, given the striking resemblance between code and natural language text. Among the available models, RoBERTa [4] stands out as a particularly effective candidate to handle the classification problem at hand.

RoBERTa is a prominent member of the transformer-based family of models, having evolved from the renowned BERT architecture. Through an arduous pre-training phase on vast corpora, RoBERTa acquires a profound understanding of language and encodes it into a series of multiple transformer layers. These layers are characterized by a meticulously designed architecture that facilitates bidirectional learning, avoiding the masked language modeling (MLM) task used by its predecessor. Embracing the essence of the attention mechanism, RoBERTa relies on self-attention to meticulously capture intricate contextual dependencies and generate expressive embeddings for each token in the input sequence. The main structure of RoBERTa can be seen in figure 2.[3]

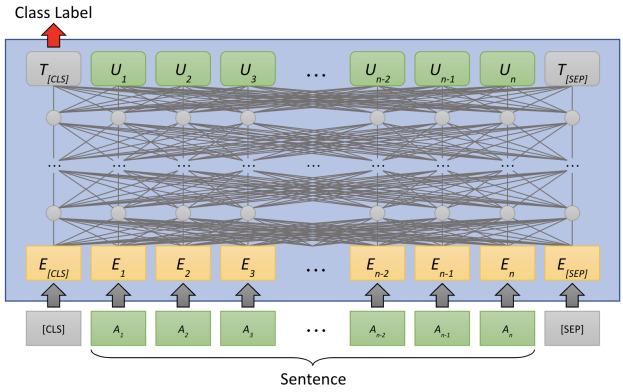


Figure 2: Structure of RoBERTa

In pursuit of our objectives, we adopt two distinct methodologies to accomplish the task at hand. The first approach involves the direct utilization of code segments extracted from the dataset, which serve as input vectors for our model. Subsequent to the model's pretraining phase, it generates a 1×2 tensor, pivotal in establishing the final classification outcome. In a conceptual context, let us denote the tokenized code, resulting in a collection of word vectors as $X = [x_1, x_2, \ldots, x_n]$, and let the model be represented as $\mathcal{M}$. Thus, the model's output, denoted as $y = \mathcal{M}(X)$, leads to the ultimate classification $\mathcal{C}$, which is determined by evaluating $\mathcal{C} = argmax(sigmoid(y))$.

The second approach entails the utilization of pre-extracted Abstract Syntax Trees as input for the classification task. As the RoBERTa model inherently possesses limitations in directly accommodating tree-like structures as input, we choose to leverage the traversal results of the ASTs to ensure model compatibility. Specifically, let us denote the source code as X, and the extracted AST as $T = \text{AST}(X)$. In this particular methodology, the input consists of the traversal sequence $[t_1, t_2, \ldots, t_n]$ derived from the tree T. The evaluation of output results follows the same procedure as employed in the first approach.

[3] https://cs.uwaterloo.ca/~jimmylin/BERT-diagrams-public.pptx

Table 1: Settings of Hyperparameters

Hyperparameters	Value
learning rate	1e-5
max length	206
epoch	181
batch size	16

4. EXPERIMENT

4.1 Expermental Setup

In this experiment, we have selected the Function-docstring classification section of the py150 dataset, which has been re-annotated in the paper CuBERT[1]. This particular subset of data primarily aims to detect instances where functions and docstrings do not correspond appropriately. When such mismatches occur, they are labeled as 'Incorrect'; otherwise, they are labeled as 'Correct'.

Prior to inputting the dataset into the model, a preliminary preprocessing step is commonly employed to segment the textual data into tokens. For this purpose, we have opted for BBPE tokenization. The foundational principle underlying BBPE tokenization entails the utilization of a greedy algorithm to iteratively amalgamate the two contiguous tokens exhibiting the highest frequency. Divergent from traditional word-based tokenization methods, this subword-oriented approach effectively circumvents the manifestation of the long-tail effect. This attribute is especially advantageous when dealing with texts, such as code, where the liberty to arbitrarily select variable names may give rise to an abundance of previously unseen word instances.

In the context of experimental model construction, the RoBERTa model from Hugging Face's transformers library was adopted, benefiting from its highly encapsulated architecture, which allows trainers to concentrate their attention more effectively on critical aspects such as data ingestion and result analysis. The training phase involved the careful configuration of specific hyperparameters, presented in Table 1.

Concerning the matter of AST integration, the AST library in Python is utilized to conduct parsing of an abstract syntax tree, thereby facilitating its seamless ingestion into the model. Following the successful acquisition of the AST, a systematic traversal is performed, engendering a set of tokens that undergo meticulous segmentation. These segmented tokens, comprising discrete identifiers, are subsequently channeled into the formidable RoBERTa model, imbuing it with the essential elements essential for its robust operation.

Throughout the training process, we diligently documented the progressive decline of the loss function, as visually depicted in Figure 3.

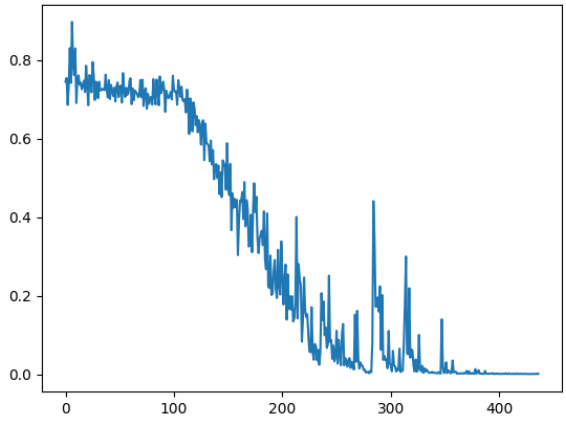


Figure 3: Trend of Loss Variation

4.2 Experimental Results Report

After acquiring the RoBERTa pre-trained models, obtained through distinct training procedures involving source code and Abstract Syntax Trees, the evaluation phase was initiated using a meticulously prepared test dataset. The test dataset comprises both source code samples and their associated labels for evaluation purposes. To ensure consistency, we subject the test dataset to analogous preprocessing techniques employed during the preparation of the training dataset. The preprocessed test samples are then fed into separate models, one trained on source code, and the other on the AST. After obtaining the output results from the test dataset, we proceeded to calculate the accuracy metrics. The formula for calculating accuracy can be represented as: $Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$. The ensuing accuracy outcomes are summarized shown as figure 4.

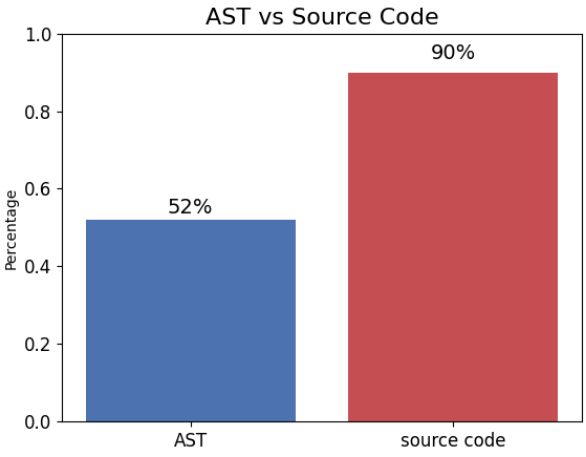


Figure 4: AST vs Source Code

4.3 Experimental Results Analysis

The experimental findings from Section 4.2 reveal that the adoption of the source code-based method yields a significantly higher accuracy (90%) compared to the AST-based technique (52%) for code vulnerability detection. This compelling evidence indicates the superiority of utilizing source code in the aforementioned detection process. Subsequently, a comprehensive analysis of the factors contributing to this outcome will be presented in the subsequent sections.

First of all, the source code contains abundant syntactic information, including identifiers, keywords, operators, and so on. This information can assist the model in better understanding the structure and logic of the code. On the other hand, the Abstract Syntax Tree (AST) represents a more abstract form of the code, which removes some syntactic details and may result in the loss of useful information.

Secondly, contextual information in the source code is crucial for code vulnerability detection. The same piece of code may have different meanings in different contexts, and ASTs often cannot fully retain this contextual information. Models processing source code can capture contextual information more effectively, leading to more accurate identification of vulnerabilities.

Thirdly, for models based on source code, feature engineering may be more straightforward, as the source code itself provides a natural representation. However, for models using ASTs, it may be necessary to process and transform the ASTs to extract meaningful features. Differences in the feature extraction process can influence model performance.

5. FUTURE WORK

First, in this paper, we primarily focus on the binary classification of individual code vulnerabilities in the Python language. However, a more advanced code vulnerability detection system should be capable of pinpointing the exact location of code vulnerabilities and be applicable to multiple programming languages, detecting a wider range of code vulnerabilities.

Furthermore, we conduct experiments in this paper using only one model. In future research, it would be beneficial to explore the utilization of multiple existing models or develop specialized models tailored for the domain of code analysis.

Thirdly, in this paper, we explore the integration of deep learning with Abstract Syntax Trees (AST) in static analysis. However, in the future, it is worth investigating additional code representation methods such as intermediate code to further enhance the detection capabilities.

6. CONCLUSION

In this paper, we address the problem of vulnerability detection in Python code and propose two approaches: utilizing source code and training models using Abstract Syntax Trees (AST). The paper presents the process of parsing Python source code into AST and explains the model training procedure. We ultimately conclude that the source code

approach outperforms the AST method and provide suggestions for future work.

References

[1] Aditya Kanade et al. *Learning and Evaluating Contextual Embedding of Source Code*. 2020. arXiv: 2001.00059 [cs.SE].

[2] Seohyun Kim et al. *Code Prediction by Feeding Trees to Transformers*. 2021. arXiv: 2003.13848 [cs.SE].

[3] Zhen Li et al. "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities". In: *IEEE Transactions on Dependable and Secure Computing* 19.4 (July 2022), pp. 2244–2258. DOI: 10.1109/tdsc.2021.3051525. URL: https://doi.org/10.1109%2Ftdsc.2021.3051525.

[4] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[5] Pengcheng Zhang, Qiyin Dai, and Patrizio Pelliccione. *CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing of Deep Learning Systems*. 2020. arXiv: 1911.07931 [cs.CV].

CodeGuard: A Deep Learning Based Automatic Source Code Vulnerability Detector

Yan Runbang
Huazhong University of Science and Technology
1037 Luoyu Road
Wuhan, China
yrb@hust.edu.cn

Yu Yifei
Huazhong University of Science and Technology
1037 Luoyu Road
Wuhan, China
yuyifei@hust.edu.cn

Han Yiting
University of Electronic Science and Technology of China
Chengdu, China
1730336549@qq.com

Niu Mingcen
SiChuan University
No.2,Section2,Chuanda Road
Chengdu, China
niumingcen@163.com

Tian Ziqi
SiChuan University
No.2,Section2,Chuanda Road
Chengdu, China
245231528@qq.com

ABSTRACT

This paper demonstrates our work to build CodeGuard [1], a deep learning based automatic code vulnerability detector aimed at detecting cross-site scripting(XSS) attack vulnerabilities of web based applications.

We generate about 10,000 PHP source code files containing safe and unsafe code snippets as the dataset, and parse all the codes into Abstract Syntax Trees(ASTs), after which we extract paths from ASTs as inputs to train a *code2vec*[11] neural network model, capturing both semantic and syntactic features of codes, providing a reasonable prediction of vulnerabilities.

What's more, we build a user interface for our model, which allows users to enter their code, then CodeGuard displays the probability of the code being safe or unsafe, together with the crucial paths on an AST graph which contribute to the prediction most, enabling programmers to free their codes from vulnerabilities with a well-defined objective in mind.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Corrections*

General Terms

General literature

[1]The source codes of the project is at: https://github.com/uniqueFranky/php_xss_detector

Keywords

cross-site scripting, code vulnerabilities, source code analysis, deep learning, AST visualization

1. INTRODUCTION

Web applications have revolutionized the way people interact with the Internet, providing a seamless user experience and enabling various online services. However, this increased usage has also led to a rise in cyber security threats, making web applications susceptible to a range of attacks, including cross-site scripting(XSS) attacks which steal sensitive user data, hijack sessions, or even propagate further attacks, threatening the confidentiality and integrity of the web application.

To counter XSS attacks effectively, developers must implement validations on the inputs and outputs of web applications. However, due to the various methods to implement XSS attacks, these validations are highly susceptible to being bypassed by attackers, making it difficult for developers to cover all the situations. These shortcomings are repeatedly detected and exploited by the attackers.

In this paper, we presents a XSS attack vulnerability detector based on abstract syntax trees and deep learning algorithms. It effectively detects whether programmers' source codes include defensive measures against XSS attacks or vulnerability to XSS attacks. Currently, developers can input their PHP source codes into CodeGuard, which automatically informs them whether their source codes are susceptible to XSS attacks. What's more, since CodeGuard is trained to exploit both semantic and syntatic features of source codes, it can be further expanded to address source code analysis for detecting multiple vulnerabilities such as SQL injection, CSRF, and so on.

In summary, we make the following contributions in this paper:

- Convert PHP source codes into ASTs and train a neural network to capture both semantic and syntactic

features of the codes to predict the vulnerabilities of the codes.

- With the help of *attention* mechanism, we extract the most significant paths on ASTs which contribute to the predictions.

- Develop a user-friendly website that enables users to input source codes and obtain the analysis result and a graph of AST with crucial paths colored red in order to help programmers promote their codes for robustness.

2. TECHNICAL BACKGROUND

2.1 Static Code Anaylsis

Static code analysis is a method that relies on the availability of source code to match against predefined rules to identify potential vulnerabilities. Methods of source code analysis applied by developers include: model checking, control and data flow analysis and text-based pattern matching.[3] Oksana proposed three categories for static analysis: detect weakness, detect code that breaks the coding rules and technology for metrics calculation.[8]

This involves identifying the structure and organization of the code, aiding in understanding the logical structure, such as recognizing functions, classes, variables, etc. It can also be broken down into token sequences for easier recognition. Additionally, analyzing the control flow and data flow of the code is essential. Data flow analysis helps track data propagation and changes in the code, aiding in detecting potential data vulnerabilities, such as uninitialized variables or unintended data modifications. Control flow analysis helps understand the code's execution paths and conditions, thereby identifying code segments that may lead to vulnerabilities, such as insufficiently checked user inputs or improper conditional jump statements. During vulnerability rule matching, predefined vulnerability rule libraries are used to match patterns in the source code, seeking potential vulnerabilities and security issues.

2.2 Abstract Syntax Tree

Static analysis typically involves the utilization of ASTs. An abstract syntax tree is a tree-like representation of the program's source code structure. The source code is first tokenized by a lexer, producing various types of tokens, and then parsed and syntax-checked by a parser to create the AST. The root node of the AST represents the entire program, while internal nodes represent abstract syntax structures or tokens. The key feature of the AST is its ability to correspond one-to-one with various syntactic elements in the input source code.

2.3 Code Embeddings

Deep learning algorithms often takes in vectors as inputs. Converting entities into vectors enables neural networks to process a variety of entities in the real-world scenario, rather than the purely mathematical world. Different entities are converted into vectors in different ways, the process of the conversion is called *embed*, because the real-world entity is embedded into the world of multi-dimensional vectors. The vectors converted from entities are called embeddings.

There are several techniques to embed natural languages, such as *word2vec*[10]. Although programming languages share varieties of commons with natural languages, they also differ from each other mainly for the reason that not only semantic features but also syntactic features underlie the programming languages. The structures of codes do matter when comprehensing the intension of the codes. Simply applying the techniques to embed natural languages to embed programming languages does not work efficiently, prompting researchers to come up with embedding methods exclusive to programming languages.

2.4 Code2vec

Code2vec is a neural network model first proposed in *code2vec: Learning Distributed Representations of Code*[11], which represents snippets of codes as continuous distributed vectors (code embeddings). The main idea is to represent a code snippet as a single fixed-length code vector, which can be used to predict semantic properties of the snippet. This is performed by decomposing a code snippet into a collection of paths in its AST, and learning the atomic representation of each path simultaneously with learning how to aggregate a set of them using the attention mechanism.[11]

Firstly, the model decomposes a code snippet into a set of paths on its AST, and divides a single AST path into three parts: two leaf nodes and the corresponding path between them. Then the three parts are embedded separately, and the representation of that AST path is obtained by concatenating the three embeddings, and putting it through a linear network to resize the dimension of the embedding. After that comes the representation of the whole code snippet, which is obtained by calculating a weighted sum of the embeddings of all the paths on the AST of the code snippet, where the weight is calculated by multiplying the vectorized representation of a certain AST path by a global vector called attention vector, which gives a scalar as the result, and the model takes that scalar as the weight for the corresponding AST path. The embedding of the leaf nodes and the paths, together with the global attention vector, are learned simultaneously in the training process.

2.5 Cross-Site Scripting

Cross-site scripting (also known as *XSS*[4]) occurs when dynamically generated web pages display input that is not properly validated.[9] There are three types of cross-site scripting attacks: non-persistent, persistent and DOM-based.[6]

There are several methods to defend against XSS, such as setting up blacklists, whitelists, and disabling certain *GET* methods. With a view to detecting XSS attacks, PMD Nagarjun et al. proposed a model[7] to train a large labeled and balanced dataset by using supervised ensemble learning techniques.

3. WORKS RELATED TO THE PROJECT

Automated software defect prediction is the focus of researchers and developers in recent years. At present, most researches on automatic software defect prediction are based on deep learning technology. Relying on the advantages of deep learning model in semantic analysis and understanding, the structure, syntactic and semantic characteristics of defective codes are mined to realize automatic and intelligent

prediction of software defects, and provide effective support to software quality and reliability.

Chen et al.[2] proposed the use of visualization and transfer learning techniques in software defect prediction. By converting source code into grayscale representation, semantic and structural information is extracted from it, and an end-to-end learning framework based on attention mechanism and transfer learning technology is used to achieve defect prediction. Zeng et al.[12] studied the effectiveness and limitations of JIT (Just-in-Time) defect prediction research, based on CC2Vec tool, and compared it with traditional JIT defect prediction tools. Zhuang et al. [13]have proposed a new JIT defect prediction model based on AST transform embedding, which uses source code AST as its semantic representation and extracts the change sequence of AST by comparing the AST before and after repair. At the same time, combining the artificial features and the semantic features of the code with the gating mechanism, the defect prediction model is constructed. DeepJIT is an end-to-end deep learning framework for JIT defect prediction proposed by Thong et al.[5], which extracts features from Commit messages and code changes and identifies defects based on the proposed features.

4. IMPLEMENTATION

CodeGuard consists of four modules: sample input, AST generation, model analysis, and frontend display. From *Figure* 1, we can see the general architecture, and it's divided into the following three steps:

- Collect PHP source codes, where each code snippet is labeled with 0 or 1 to indicate whether the code is vulnerable or not. Split the collected dataset into training set and testing set.
- Convert each annotated PHP source code snippet into an AST and extract paths from the AST, use the paths and annotations to train a code2vec neural network.
- Extract the global attention vector from the trained model to calculate the contribution of each AST path to the prediction result. Visualize the AST and crucial paths of the AST to help programmers correct their codes.

The rest of this section will demonstrate these steps in detail.

4.1 Finding Datasets

We manage to find a vulnerable PHP source code generator[2]. Using this tool, we generate about 10,000 PHP code snippets with annotations on whether it is vulnerable to XSS attacks or not. We split the dataset: using 90% of them as the training set and 10% as the testing set.

4.2 Generating ASTs and Paths

Our next task is to turn PHP source codes into ASTs. There is a bunch of theories on generating an AST mostly related to analytic methods like lexical analysis and syntax analysis. We choose *PHP-Parser*[3], a tool to parse PHP source codes into ASTs, and use it to convert our PHP code snippets into ASTs and store them in Json files for later use.

[2]source: https://github.com/stivalet/PHP-Vuln-test-suite-generator

[3]source: https://github.com/nikic/PHP-Parser

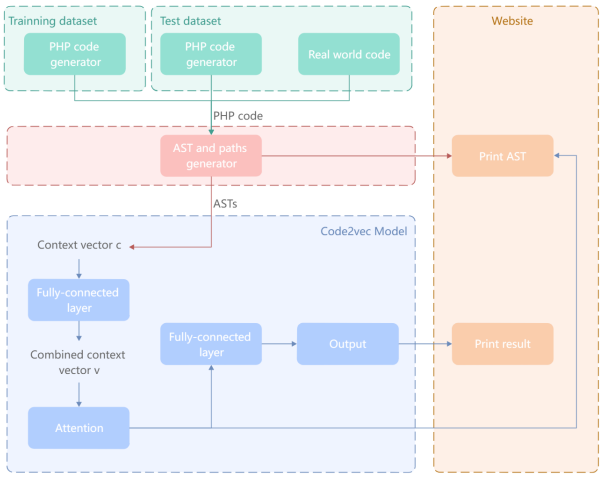


Figure 1: Architecture of CodeGuard

To track all the paths in an AST, we decide to modify the original data structure of ASTs generated by the PHP parser: adding a *parent* element to tree-node which points to its parent and a *depth* element to indicate its depth in tree. With the two new features, we can easily find the paths by looping over the *parent* element. We first traverse over the tree to preprocess the depth of each node, then enumerated every pair of leaf nodes on the ASTs, and then calculate their lowest common ancestor(LCA) to get the paths between them, this can be done using Tarjan's algorithm with a linear consumption of time, or binary lifting algorithm with an extra logarithm time consumption. What we do to search for the paths is that for every pair of leaf nodes, we choose the deeper one to jump to its parent until they are at the same depth, then they will jump together until they meet each other. Recording the nodes and edges passed through the jumping process, the path between them will be found. We use a Python script to modify the ASTs and enumerate every two leaves on a tree and finally get all the paths.

4.3 Building and Training Code2vec Model

As demonstrated in section 2, code2vec takes in the AST paths of a code snippet as input, and outputs a distributed representation of that code snippet. What we do is slightly different from the paper, especially we try to take the positions of the statements in the code snippet into consideration by encoding positional features into vectors. We built the proposed model using PyTorch, and our work includes data preprocessing, neural network constructing and training, and adjusting the hyper parameters in the model, also the failed attempt.

According to the paper[11], after we generate the paths on ASTs, we need to decompose an AST path into three parts including the two leaf nodes on it and the path between the two leaf nodes, which can be done by representing an AST path as a string like *Name↑FieldAccess↑Foreach↓Block↓IfStmt*

$\downarrow Block \downarrow Return \downarrow BooleanExpr$[11], then we separate the string by the first $\uparrow$ and the last $\downarrow$, which will give the three parts.

When it comes to embedding, we firstly encoded the AST paths and leaf nodes into one-hot vectors, where the dimension of the vectors is the different nodes and paths we have in the training set(denote that value as $|V|$), and each vector has exactly one single dimension assigned with 1, while other dimensions are assigned with 0, indicating that the vector represents that corresponding node or path. The embedding process of a given node or path is done by multiplying the one-hot vector with a $|V| \times d$-dimensional embedding matrix , where d is the embedding size, thus obtaining a d-dimension vector as the embedding for that node or path.

After obtaining the representations for the leaf nodes and paths of a given AST path, we concatenated the three vectors, getting a $3d$-dimension vector, and we multiply that $3d$-dimension vector by a matrix of dimension $3d \times d$, giving a d-dimension vector as the representation for the specific AST path.

The vectorized representation is then multiplied by a d-dimension global attention vector A, which gives a scalar as output, acting as the weight for that specific AST path in the whole code snippet. Denoting the embedding for the $i-th$ AST path in the code snippet as E_i, the distributed representation R for the whole code snippet can be calculated by:

$$R = \sum_{i=1}^{n} (E_i \cdot A) E_i \tag{1}$$

This process of calculating the weight of each path and node is called attention mechanism, which was first proposed in *Attention Is All You Need* [1]. It is this attention mechanism that enables us to obtain the crucial paths on the ASTs and visualize them in an AST graph. Since we get the representation of the code snippet, which is a d-dimensional vector, the model can learn the features encoded in the vector, and that can be done by multiplying the d-dimensional embedding of the code snippet by a $d \times 2$-dimensional matrix, resulting in a 2-dimensional vector, and we applied a softmax function[4] to that vector, and we considered the result, which is a 2-dimensional vector, as the probabilities of the code being safe and unsafe.

Training the model is trivial thanks to the interfaces provided by PyTorch, what we need to do is to specify the optimizer and loss function, which are Adam and Cross Entropy Loss, respectively, in our case. We train the model on a remote server equipped with NVIDIA RTX 3090, and the training process takes approximately 1 hour.

4.4 Encoding Positional Features

The code2vec model processes all the AST paths in a parallel way. Although this promotes the efficiency of training the model, it can not take into consideration the order of the statements in the code snippets. What we try to do is that we add positional information into the model, attempting to learn the contributions provided by the relative positions of the statements.

When decomposing an AST path, besides the original three parts, we decomposed it into two extra parts, which are the numbers of lines where the leaf nodes are in the source codes as positional information. We embed the positions forming a $5d$-dimensional context vector instead of $3d$-dimensional context vector, which is used later in learning process.

Although the thought seems reasonable, it did not reach our expectation, resulting in a low performance on testing set. The reason for that will be discussed in *Section* 6.

4.5 Visualizing ASTs and Paths

Merely predicting a code snippet to be safe or unsafe in a real-world scenario seems to be flashy, for the reason that it is the defense to the attacks that really matters. Simply pointing out whether a code snippet is vulnerable or not without hints on how to improve that code does not help much. In order to help programmers to inspect and correct their codes with a well-defined objective in mind, we built a user interface for our model, which allows users to input their code, and our program runs that code against our model, providing the probabilities of the code being safe and unsafe, together with the contribution of each AST path to the prediction result. If the user specifies a rank of the AST paths over the contributions to prediction, we can display a AST graph where the corresponding path is colored red to emphasize that contribution.

This idea was mainly inspired by the attention mechanism used in the code2vec model, where the whole code snippet is represented as a weighted sum of the representations of its AST paths. Treating that weight as the contribution of paths to the prediction seems reasonable. And this part is the most significant work we do beyond the code2vec paper[11].

In order to achieve that, we developed a React[5] based website front end, and a Golang based website back end, where the front end reads the user's inputs and interact with both the user and the back end program, while the back end calls the PHP parser to parse the codes into ASTs, then calls the python script to generate AST paths, after which runs the model to get the the embeddings for each AST path and predictions. The embeddings are used to calculate their contributions to the prediction by multiplying them by the learned global attention vector, and we sort all the paths by their contributions, retrieving the exact one the user specified, which will be emphasized in the AST graph later.

The main work of visualization is to render an AST graph with a specific path colored red. We find an open source tool Graphviz[6] for visualizing graphs, which uses DOT language to define a graph.

We firstly constructed the graph using DOT language by

[4]Softmax function refers to this function:

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2}$$

where x and $f(x)$ are both n-dimensional vectors

[5]A JavaScript library for building user interfaces

[6]https://graphviz.org

depth-first searching(DFS) the ASTs. Searching for the given path on the tree to color it red is accomplished by traversing the tree only passing through the existing nodes on the objective path. We used the following algorithm to search for the given AST path, which is implemented in *draw.py*:

Input: AST, a string representation[7] of the objective AST path
Output: the nodes and edges of the objective path on the rendered graph
Steps: DFS all the nodes matching the first node in the objective path, removing the first node and direction in the objective path. If the current direction indicator is "up", only go from child to parent; if the current direction indicator is "down", only go from parent to child. Recover the original state when backtracing from a recursive call.
We recorded the edge through which every node comes from, and since we are doing a DFS, once we find the objective path, we end the algorithm immediately, so that the records shows the edges of the objective path on the rendered graph. We only need to traverse the path in a reverse order to get the edges.

After getting the nodes and edges on the rendered AST graph, we can modify the color of those edges by the interface provided by Graphviz.

5. RESULTS AND ANALYSIS

5.1 Statistical Results

We run the testing set against our model, and take a conventional RNN model[8] trained over the original source code texts, which treats the source codes as natural languages, as the baseline for comparison. We calculated statistics including accuracy, precision, recall and F1 score of both models shown in *Table* 1. Denoting *True Positives* as TP, *True Negatives* as FP, *False Positives* as FP and *False Negatives* as FN, these statistics are calculated by:

$$Accuracy = \frac{TP + TF}{TP + TF + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

By comparing the statistics shown in *Table* 1, our model beats the RNN model in terms of these criteria. The higher accuracy means our model is more likely to predict the vulnerability of a code snippet correctly, i.e. making more true positives and true negatives. According to *equation* 4 and *equation* 5, the higher precision and recall indicates that our model is making less false negatives and false positives. More specifically, our model gets a recall of 100%, indicating that

[7]e.g. Name↑FieldAccess↑Foreach↓Block↓IfStmt↓Block ↓Return↓BooleanExpr

[8]The code for training both the code2vec model and the RNN model is at *model.py*

Model	Accuracy	Precision	Recall	F1 Score
Our Model	96.8%	93.1%	100.0%	96.5%
RNN Model	93.8%	90.8%	95.4%	93.0%

Table 1: Statistics of the Models

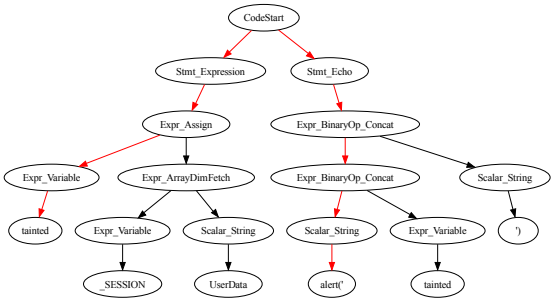


Figure 2: AST graph example for an unsafe code snippet

our model does not make any false negative on the testing set, so our prediction is a *complete* one, where *completeness*[9] is such that if a code is unsafe, the analysis always says it is unsafe. The completeness means our model is capable of predicting the safety of the codes without missing a single unsafe code, which is crucial to vulnerability analysis. Programmers using this tool will not miss any vulnerability in their codes.

The main reason behind this statistical difference should lie in the fact that trained on ASTs of code snippets, our model learns both the semantic and syntactic features of codes, while the RNN model is trained on source codes to only capture the semantic features of codes, omitting the syntactic structures of codes, which is the dominant difference between programming languages and natural languages.

Besides, thanks to the parallelization of processing all the AST paths simultaneously provided by the attention mechanism, training our model takes about 1 hour, while the RNN model, which takes in the tokens in texts in order, takes about 1.5 hours.

5.2 Visualization Analysis

We choose a safe code snippet and an unsafe code snippet from the testing set, and displayed the prediction results and the visualized ASTs using our tool.

Here is a simplified example for an unsafe code snippet:

```
<?php
$tainted = $_SESSION['UserData'];
echo "alert('". $tainted ."')";
```

Our model predicts that this code snippet has a 88.1% chance of being vulnerable, and the model paid most attention to the AST path shown in *Figure* 2, which passes through the

[9]Completeness and soundness are widely used as criterion in analysis.

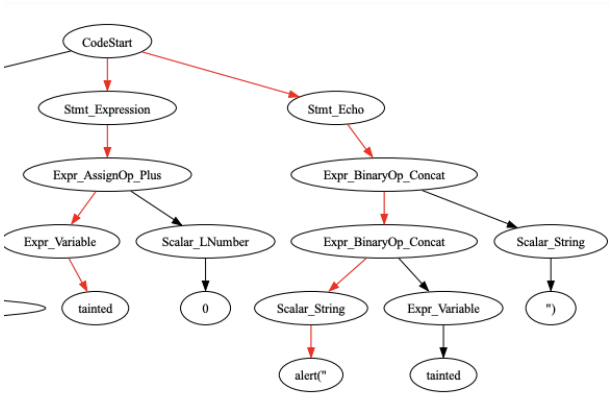


Figure 3: AST graph example for a safe code snippet

assign statement where the variable *tainted* is assigned with a direct fetch from *_SESSION*, and the *alert* function call which directly displays information from *_SESSION* without sanitization. This path does account for the vulnerability of the code snippet.

Here is another simplified example for a safe code snippet:

```
<?php
$tainted = $_GET['input'];
$tainted += 0 ;
echo "alert(\"". $tainted ."\")" ;
```

This code snippet is safe because it implicitly converted the variable *tainted* into a number-type variable by adding zero to that variable. As shown in *Figure* 3, our model does recognize that implicit conversion because it pays most attention to the path which passes through the *add* expression and the *alert* function call.

6. CONCLUSIONS AND REFLECTIONS

In this work, we parsed PHP source codes into ASTs, extracted paths from ASTs, trained a code2vec neural network model, visualized the ASTs and crucial paths on ASTs, and tried to improve the model by encoding positional features.

The model ends up being a *complete* analyst for XSS attack vulnerabilities. With the completeness provided by our tool, programmers can free their codes from XSS attacks since no vulnerability will be omitted by the model, and the crucial paths on the ASTs can equip programmers with a well-defined objective to promte their codes for robustness.

For the reason that the model is trained to learn both the semantic and syntactic features of source codes, given time and corresponding dataset, it can be easily transformed into a model predicting other vulnerabilities of other programming languages.

The reason why the positional encoding failed to play a role in the model may be that the numbers of lines in the source codes are absolute positional features, and the positions of the same statements varies from code to code. If we managed to represent the relative positional features of statements in source codes, we should be able to improve the performance of the model.

There are also drawbacks of our model. The tool we use for generating code snippets is only capable of generating simple code snippets which do not cover the whole PHP language set. When it comes to real-world codes, there are many statements that the model hasn't seen before, which makes it lack the acknowledgement of those features. A solution to that is we translate the real-world code into an identical but simpler form(e.g. turn *switch* statements to *if-elseif* statements), using a subset of PHP which is fully covered in the training set. Alternatively, we can use real-world datasets to train our model.

7. REFERENCES

[1] N. P. J. U. L. J. A. N. G. L. K. I. P. Ashish Vaswani, Noam Shazeer. Attention is all you need. 2017.

[2] Y. Y. e. a. Chen J, Hu K. Software visualization and deep transfer learning for effective software defect prediction. *ACM/IEEE 42nd international conference on software engineering*, 2020.

[3] P. Emanuelsson and U. Nilsson. A comparative study of industrial static analysis tools. *Electronic notes in theoretical computer science*, 217:5–21, 2008.

[4] S. Gupta and B. B. Gupta. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8:512–530, 2017.

[5] K. Y. e. a. Hoang T, Dam H K. Deepjit: an end-to-end deep learning framework for just-in-time defect prediction. *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019.

[6] A. Klein. Dom based cross site scripting or xss of the third kind. *Web Application Security Consortium, Articles*, 4:365–372, 2005.

[7] P. Nagarjun and S. A. Shaik. Ensemble methods to detect xss attacks. *International Journal of Advanced Computer Science and Applications*, 11(5), 2020.

[8] O. V. Pomorova and D. O. Ivanchyshyn. Assessment of the source code static analysis effectiveness for security requirements implementation into software developing process. In *2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, volume 2, pages 640–645. IEEE, 2013.

[9] K. Spett. Cross-site scripting. *SPI Labs*, 1(1):20, 2005.

[10] G. C. J. D. Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space. 2013.

[11] O. L. E. Y. Uri Alon, Meital Zilberstein. code2vec: Learning distributed representations of code. *POPL*, 2018.

[12] Z. H. e. a. Zeng Z, Zhang Y. Deep just-in-time defect prediction: how far are we? *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021.

[13] Z. X. Zhuang W, Wang H. Just-in-time defect prediction based on ast change embedding. 2022.

An exploration of wireless network security

Yu Enbo
Sichuan University
3235952027@qq.com

Wang Siyi
Sichuan University
794688056@qq.com

Shao Jiayang
Wuhan University
2021302141070@whu.edu.cn

Li Huaxuan
Southern University of Science and Technology
2549952274@qq.com

Zhao Miao
Beijing University of Posts and Telecommunications
n@ova.moe

ABSTRACT

The widespread adoption and usage of wireless networks have enabled us to connect to the internet anytime and anywhere, enjoying wireless convenience. However, as wireless networks continue to grow and become more prevalent, network security issues have become increasingly prominent. Hackers and malicious users can exploit the vulnerabilities of wireless networks and infiltrate and disrupt them through various attack methods, posing serious security threats.

This paper aims to conduct an in-depth study of wireless network security issues and explore common wireless network attack methods, as well as how to detect and evaluate network security. By analyzing the security of wireless networks in depth, we can gain a better understanding of the risks that exist in wireless networks and take appropriate measures to protect sensitive information for individuals and organizations.

Keywords

wireless network, security, fake access point, detector, machine learning

1. INTRODUCTION

Wireless network security has always been a major obstacle to the development of the Internet industry. Our team is very interested in the related academic research and experimental operations. Therefore, we have prepared the following research report.

Firstly, we will explore the basic principles and protocols of wireless networks, such as Wi-Fi, Bluetooth, and mobile communication networks. Understanding these basic principles can help us understand how wireless networks work and the potential security risks.

Next, we will provide a detailed introduction to common wireless network attack methods. These include but are not limited to fake AP attacks, man-in-the-middle attacks, fake beacon attacks, and brute force. We will investigate the techniques attackers use to exploit network vulnerabilities and weaknesses to launch attacks, and discuss their potential threats to network security.

After researching the relevant attack methods for wireless network security, we attempted to explore the detection of wireless network security. We mainly used BSS detection and machine learning-based detection methods to investigate whether the wireless network in the surrounding environment is secure. It is the core of our research project

Finally, we summarized the relevant attacks and detections of wireless network security and proposed some defense measures and methods based on the academic research reports. We also summarized all the work and achievements, which deepened our understanding of the academic research related to wireless network security.

2. BACKGROUND INFORMATION

In this section, we will introduce some background information related to wireless network security as well as the key knowledge that we will focus on in this study.

2.1 Wireless network

Wireless networks refer to computer networks that use wireless data connections between network nodes instead of physical cables or wires. These networks allow devices such as computers, smartphones, and tablets to connect to each other and to the internet without the need for physical connections. There are several types of wireless networks, including Wi-Fi, Bluetooth, and cellular networks. While wireless networks provide greater mobility and convenience compared to wired networks, they also pose unique security challenges, such as eavesdropping, unauthorized access, and denial-of-service attacks. To ensure the security of wireless networks, various security measures and protocols have been developed, including encryption, authentication, and access control mechanisms.

2.2 Fake Access Point

Access Point

AP stands for Access Point. In computer networks, an AP is a device in a wireless network that is responsible for connecting wireless devices to a wired or wireless network. APs are commonly used in Wi-Fi networks and are designed to receive data from wireless devices and forward it to wired networks or other wireless

devices. APs typically provide the name and password for the wireless network so that users can connect to the wireless network. In enterprises and public places, APs are commonly used to provide wireless network services so that users can connect to the network and access the internet using wireless devices.

Study of fake AP

A fake AP, also known as a rogue AP or spoofed AP, is a malicious access point created by a hacker or attacker. Fake APs are commonly used for network attacks such as man-in-the-middle attacks and phishing attacks to obtain sensitive information from users, such as usernames, passwords, and credit card information.

Development

The development of fake APs is mainly driven by the development of wireless network technology. As wireless network technology becomes more widespread and prevalent, fake AP attacks are constantly upgrading and evolving.

Early fake AP attacks mainly involved using special wireless devices to create fake Wi-Fi networks to lure users into connecting and steal their sensitive information. This type of attack was particularly common in public places and hotspots.

As wireless network technology continues to develop, fake AP attacks are also constantly evolving and upgrading. Attackers are no longer limited to creating fake Wi-Fi networks but can also use other techniques such as software tools to simulate wireless networks, man-in-the-middle attacks, and more to carry out fake AP attacks. These attack methods can be more covert and effective in carrying out attacks, posing a greater threat to network security.

At the same time, with the development and application of emerging technologies such as the Internet of Things (IoT) and 5G, the application scenarios and methods of wireless networks are also constantly expanding and enriching. This provides more attack targets and opportunities for fake AP attacks, making them an important issue in the field of network security.

2.3 Necessary of security

The importance of wireless network security is self-evident. Paying attention to wireless network security is equivalent to valuing our lives and development. The importance of wireless network security can mainly be reflected in the following aspects:

Data protection: Network security ensures that our sensitive data and personal information are protected. Whether it is individual users or organizations, we store and transmit large amounts of data on the network, including financial information, medical records, personal identity information and more. If this data falls into the hands of hackers or criminals, it can lead to identity theft, financial loss, credit card fraud and other problems.

Economic stability: Network security is crucial for maintaining economic stability and sustained growth. Network attacks can cause trade interruptions, financial system collapses, corporate bankruptcies, and more. Once a country or organization suffers a large-scale network attack, it will have a serious impact and losses on its economy, and may even trigger a global economic chain reaction with unimaginable consequences.

Privacy protection: Network security helps to maintain the privacy rights of individuals and organizations. In the Internet era, we use various online services and social media platforms more and more. If the security of these platforms or services is compromised, our personal privacy will be threatened. Network security ensures that our communication, online activities and personal information are protected, preventing them from being misused or leaked.

Protection of public infrastructure: Many critical infrastructures in modern society, such as electricity, water supply, transportation, communication, etc., are closely connected to the Internet. The lack of network security may lead to the paralysis of these infrastructures, posing a serious threat to public safety and well-being. Therefore, protecting these infrastructures from network attacks is crucial.

3. ENVIRONMENT

In this section, we will introduce the software operating system environment and necessary hardware support that we have built for computer machines in the learning process of our research.

In this study, we mainly used the **Kali Linux** operating system to complete all project experiments, and virtual workstations were used to install virtual machines.

Kali Linux is a Debian-based Linux distribution that is focused on network security testing and penetration testing. It provides a range of tools and programs for security testing and attacks, including network scanning, vulnerability assessment, password cracking, wireless network cracking, reverse engineering, and more. Kali Linux also provides tools for digital forensics and data recovery. Kali Linux is developed and maintained by Offensive Security, and it is a free and open-source software that can be downloaded and installed from the official website.

Figure 1: Kali Linux

Some features of Kali Linux include:

Integration of many network security tools and programs for various security testing and attack scenarios.

Support for multiple hardware platforms, including x86, ARM, and Raspberry Pi.

Provision of an easy-to-use graphical user interface as well as command-line interface.

Provision of comprehensive documentation and community support for learning and problem-solving.

High-security features that support encryption and signing.

In addition, we also need a wireless network card to simulate and reproduce wireless network attack methods.

Figure 2: wireless network card

4. ATTACK ANALYSIS

In this section, we attempted different types of wireless network security attacks and presented various types of threats to wireless network security through explanations of the underlying principles and demonstrations of the attack methods in experiments.

4.1 Fake AP attack

Wireless networks based on the 802.11 protocol are currently mainstream. According to this protocol standard, if there are multiple APs with the same SSID and MAC address in the environment, the client will automatically connect to the AP with the strongest signal. For this reason, attackers can easily set up fake APs by setting up an AP with the same SSID and MAC address as the one they want to forge and maximizing its signal strength. Currently, tools such as AirCrack-ng and Wi-Fi-Pumpkin can be used to quickly set up fake APs. The current attack methods for fake APs are mainly divided into passive attacks and active attacks.

Figure3: fake AP called NUS we created

4.2 Man-in-the-Middle Attack

a type of network attack where an attacker deceives both parties involved in a communication, making them believe that they are communicating directly with each other, when in reality all communication is being monitored and tampered with by the attacker.

A MITM Attack typically involves an attacker inserting a malicious third party between two communicating entities, through which all communication is routed. The attacker can then eavesdrop on sensitive information, deceive the parties involved, and manipulate the contents of the communication.

Specific experimental procedures will be presented in the appendices.

4.3 Fake Beacon Attack

Fake Beacon is a wireless network attack technique where an attacker sends fake Beacon frames to deceive Wi-Fi devices into connecting to a malicious network controlled by the attacker. An attacker can create a fake Wi-Fi network in a victim device by sending fake Beacon frames, which can trick users into thinking it is a legitimate network and connecting to it. The attacker can then steal sensitive information, monitor network traffic, and tamper with network data on this fake network.

4.4 Brute Force

Brute force is a password cracking technique where an attacker continuously tries all possible password combinations until the correct password is found. Brute force attacks are typically used to attack encrypted files, network accounts, email accounts, etc.

Attackers can use computer programs or tools to automate brute force attacks. These tools can generate all possible password combinations and automatically try them until the correct password is found. This type of attack can be very time-consuming, depending on the length and complexity of the password.

Specific experimental procedures will be presented in the appendices.

5. DETECTOR ANALYSIS

In this section, we attempted different types of wireless network detection methods and presented how we detect the security of wireless networks in the surrounding environment through explanations of the underlying principles and demonstrations of the detection methods in experiments.

The detection of wireless network security and the development of functional software is the core of our research project, and therefore, this section is also the most important and crucial one.

5.1 Introduction

Bad AP detector is a tool used to detect malicious or fake wireless access points (APs) in a network. It can scan wireless signals in the network, identify and locate bad APs, and help administrators detect and address network security issues in a timely manner. Bad AP detector can detect malicious or fake APs by monitoring parameters such as wireless signal strength, frequency, and channel in the network. It can identify unauthorized APs and detect those that have been hacked and used for man-in-the-middle attacks or other malicious activities. It can also provide other security features, such as automatically blocking bad APs, alerting administrators, recording information about bad APs, and activating automatic defense systems. These features can help administrators detect and address network security issues in a timely manner, and improve the security and reliability of the network.

5.2 BSS detector

A BSS is a basic unit in a wireless network, consisting of an access point (AP) and the client devices associated with it. BSS detectors are commonly used in wireless network security to help detect and identify malicious access points and Evil Twin attacks.

BSS detectors can scan all BSSs in a wireless network and display their detailed information, such as MAC addresses, channels, SSIDs, and signal strengths.

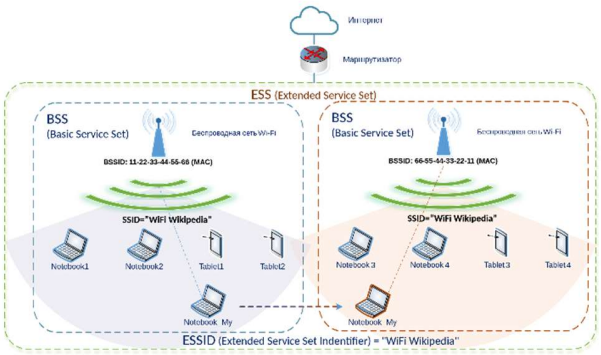


Figure 4: BSSid

BSSid is unique, and according to statistical laws, an organization usually uses the same product to set up the entire network, which means that by analyzing BSSid, we can approximately judge whether an AP is a malicious access point by its differences with normal access points.

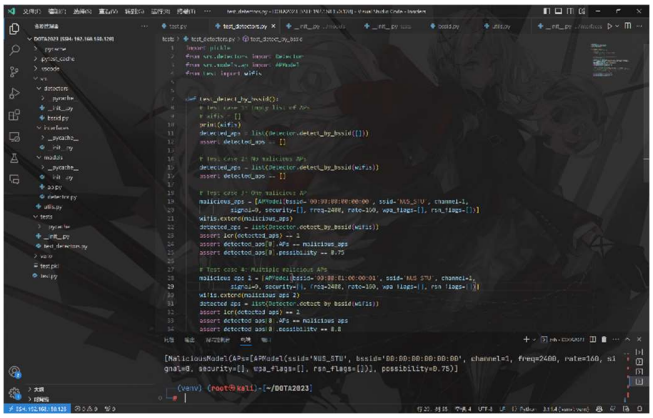


Figure 5: test code and result of BSS detector

Here, we have successfully replicated the process of BSS detection and implemented a wireless network security detector based on it. BSS detection has important applications in wireless network security and can be used to detect unauthorized APs, malicious APs, and other security issues in wireless networks.

5.3 Machine Learning detecting software

Furthermore, we aimed to design a simple yet effective software that utilizes machine learning models to detect wireless network security in the surrounding environment. The software features two functions: one is to detect rogue APs based on their manufacturer, and the other is to detect rogue APs based on 16 features extracted and parsed from 802.11 frames.

To train our model, we needed to search for and collect a wireless network dataset. In our research, we utilized an open-source dataset as the foundation for training our model. Since this dataset was raw, we performed preprocessing and feature extraction work on it.

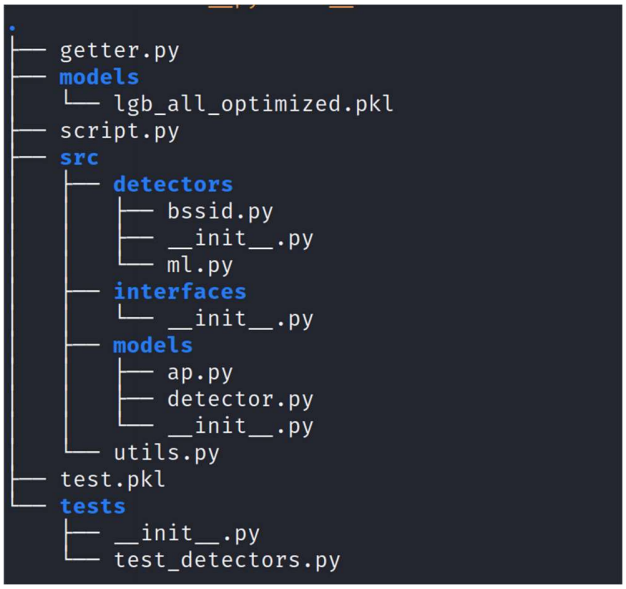


Figure 6: structure of software(code)

The software has two functions: one is to detect rogue APs based on their manufacturer, and the other is to detect rogue APs based on 16 features extracted and parsed from 802.11 frames. After extracting and parsing these 16 features from the 802.11 frames, the trained model is called to generate predictions on whether an AP is rogue or not.

In preprocessing the dataset, we first performed data cleaning and normalization to ensure the accuracy and reliability of the data. We then extracted 16 important features as the basis for subsequent training, including frame.len, radiotap.length, wlan.duration, and more. These features were obtained by parsing the 802.11 frames, therefore requiring parsing and splitting of the 802.11 frames when extracting features. We utilized the Python programming language and corresponding libraries to process the dataset and extract features.

1. frame.len
2. radiotap.length
3. radiotap.dbm_antsignal
4. wlan.duration
5. radiotap.present.tsft
6. radiotap.channel.freq
7. radiotap.channel.flags.cck
8. radiotap.channel.flags.ofdm
9. wlan.fc.type
10. wlan.fc.subtype
11. wlan.fc.ds
12. wlan.fc.frag
13. wlan.fc.retry
14. wlan.fc.pwrmgt
15. wlan.fc.moredata
16. wlan.fc.protected

Figure 7: Required Features

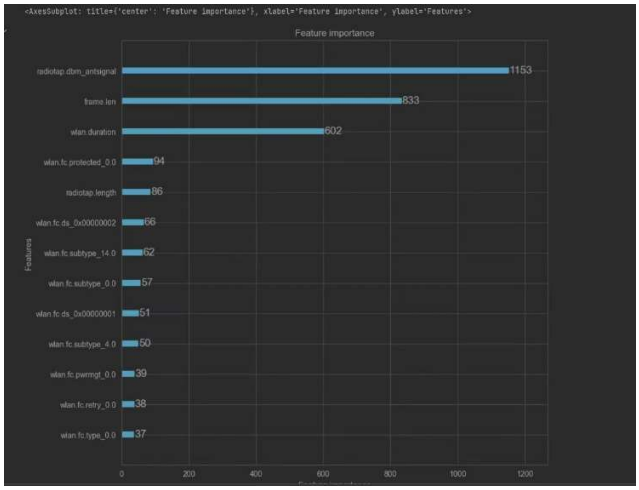


Figure 8: Importance of Features

After preprocessing the data and extracting features, we utilized a the machine learning model. In this research, we selected LGB as the machine learning model. LGB is an efficient gradient boosting decision tree algorithm and is one of the ensemble learning methods based on decision trees. LGB's advantages lie in its efficiency and scalability. It uses multi-threading and histogram-based algorithms to accelerate the training process and can handle large-scale datasets.

Additionally, LGB provides configurable hyperparameters and other advanced features, such as feature importance evaluation and early stopping. We utilized the Python programming language and TensorFlow library to train the model and performed multiple rounds of testing and fine-tuning to achieve optimal detection performance.

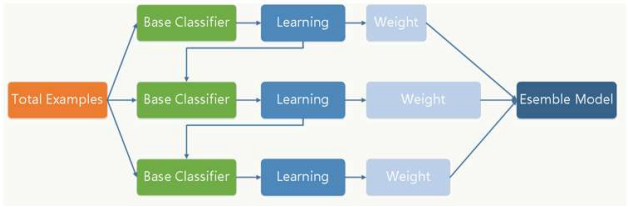


Figure 9: principle of LGB

When applying the experimental results, we utilized the software to detect wireless network security in the surrounding environment and achieved accurate detection results for rogue APs. Through multiple rounds of testing and application of the experimental results, we continuously optimized the parameters and features of the model, thereby improving detection performance and accuracy.

```
└─# python script.py --model ml
NovaMiyu[10:51:07:f8:7f:82] No malicious AP detected.
NUS[f8:1a:67:b0:2e:f6] MALICIOUS AP DETECTED !!!!
HS_PGPR_Family[24:f2:7f:08:80:44] No malicious AP detected.
(Hidden Network)[36:cd:f8:7b:ea:f0] No malicious AP detected.
HS_Conference[24:f2:7f:08:80:42] No malicious AP detected.
Wireless@SGx[24:f2:7f:08:80:43] No malicious AP detected.
HS_PGPR_Family[24:f2:7f:08:93:24] No malicious AP detected.
Wireless@SGx[24:f2:7f:08:93:23] No malicious AP detected.
HS_Conference[24:f2:7f:08:93:22] No malicious AP detected.
NUS_STU[e6:de:b1:ac:f1:b8] No malicious AP detected.
HS_Conference[8e:2d:e2:65:bb:4f] No malicious AP detected.
NUS_STU[7e:06:24:54:b2:8c] No malicious AP detected.
NUS_STU[8e:ce:9c:69:02:4e] No malicious AP detected.
NUS_STU[f8:4d:89:78:8f:4a] No malicious AP detected.
```

Figure 10: Result of detecting software

In summary, our research provides valuable insights and methods for designing and developing software to detect wireless network security. Utilizing machine learning models to detect rogue APs can improve detection accuracy and efficiency, providing reliable technical support for wireless network security management.

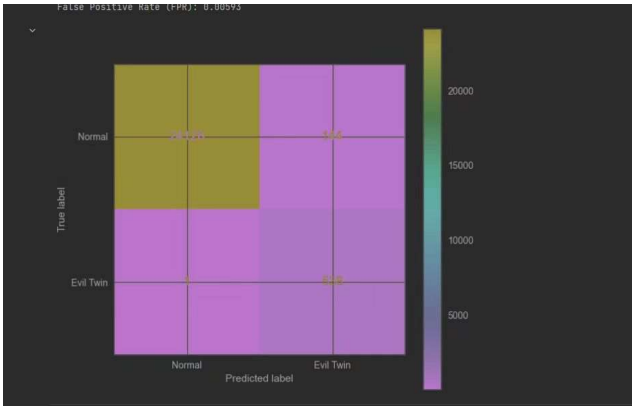


Figure 11: Predicted rate of detecting software

6. DEFENSE

In this section, we will try to summarize the relevant attacks and detections of wireless network security and propose some defense measures and methods based on the academic research reports.

6.1 Detection-based

Detection-based wireless network security defense is a security defense method based on network traffic monitoring and analysis, which protects the security of wireless networks by monitoring abnormal and attack behaviors in network traffic.

Detection-based wireless network security defense typically uses intrusion detection systems (IDS) or intrusion prevention systems (IPS) to monitor network traffic. IDS can monitor abnormal and attack behaviors in network traffic and issue alerts to administrators. IPS not only detects abnormal and attack behaviors but also automatically blocks attack behaviors and protects network security.

It is worth noting that the machine learning-based wireless network security detection software we develop in the detection phase can also be considered a simple defense mechanism. This is because it can provide connection warnings by detecting the security of wireless networks, informing people which networks are unsafe, and reminding them not to connect to these insecure networks. Therefore, this type of detection and warning system can also be viewed as a defense mechanism, although real-world defense mechanisms tend to be more complex and specialized.

6.2 Others

In addition to detection-based wireless network security defense, there are also many other wireless network security defense mechanisms, including:

Encryption: using encryption algorithms to encrypt wireless network data to protect data confidentiality and integrity.

Access control: using access control techniques to restrict access to wireless networks, allowing only authorized users to access the network.

Strong password policies: standardizing and managing access passwords for wireless networks, requiring users to use strong passwords and regularly change passwords.

Segmentation: dividing wireless networks into multiple subnets to limit the impact of attackers.

Firewall: using firewall technology to monitor and filter wireless network traffic to prevent malicious traffic from entering the network.

Virtual Private Network (VPN): using VPN technology to establish a secure wireless network connection to protect data confidentiality and integrity.

Security audit and vulnerability management: conducting regular security audits and vulnerability management to identify and fix security vulnerabilities in wireless networks.

Alert and response: establishing alert and response mechanisms to promptly detect and respond to security incidents in wireless networks.

7. CONCLUSION

Wireless network security has become an increasingly important research area in recent years due to the widespread use of wireless networks and the growing number of security threats. This paper provides a summary of research and experiments related to wireless network security attacks, detection, and defense.

Various types of attacks on wireless networks, such as man-in-the-middle attacks, fake AP attacks, and brute-force cracking, have been analyzed and classified in several research papers. Researchers have also explored various detection and defense techniques, including BSS detection and machine learning-based detection, to counter these attacks.

In summary, research and experiments related to wireless network security attacks, detection, and defense are essential to identify and mitigate security threats in wireless networks. Further research in this area is necessary to develop more efficient and accurate detection and defense techniques that can provide better protection against wireless network attacks.

8. FUTURE WORK

In this section, we will talk about the possible arrow of future work related to wireless network security. Based on our proposed defense methods for wireless network security, including attacks, detection, and extended measures, we believe that the future of wireless network security in the Internet industry can focus on the following aspects:

Develop more efficient and accurate detection methods: Although BSS detection and machine learning-based detection can effectively detect malicious activities in wireless networks, there are still issues with false positives or false negatives. Future works can develop more efficient and accurate detection methods by combining multiple detection algorithms and techniques to improve detection accuracy and efficiency.

Conduct in-depth research on attack mechanisms and features: In-depth research on the mechanisms and features of different types of wireless network attacks can provide a deeper theoretical foundation for developing more efficient and accurate detection methods. Furthermore, in-depth research on attack mechanisms and features can also provide network administrators with more specific and effective defense measures.

Increase security awareness of users and network administrators: Wireless network security issues are not only technical issues but also involve the security awareness of users and network administrators. Future works can strengthen security awareness education for users and network administrators, increase their awareness and understanding of wireless network security, and thereby reduce the risk of network attacks.

Promote new security technologies and standards: With the continuous development of technology, new wireless network security technologies and standards are emerging. Future works can promote new security technologies and standards, such as WPA3, 802.11ax, etc., to improve the security and performance of wireless networks.

9. ACKNOWLEDGEMENTS

We would like to offer our most sincere appreciation and thanks to Professor Hugh Anderson for his guidance, enthusiasm and encouragement during the project. He is always generous in providing all kinds of equipment, we might need and we are equally grateful to the School of Computing for this. And we also thank our teaching assistant Shen Jiamin for his insight and expertise. It is with their careful guidance and attention that we were able to complete this project.

10. REFERENCES

[1] Kali Linux Wireless Network Penetration Testing in Detail, Li Yawei, Tsinghua University Press 067209-01

[2] A Review of Methods for Detecting Rogue Access Points" by Zheng Ruihuan (School of Cybersecurity, Sichuan University, Chengdu 610065, China) Article ID: 1007-1423(2020)14-0031-04

[3] Analysis of Man-in-the-Middle Attack Methods and Risks in Wireless Local Area Networks Based on Rogue APs" by Zhu Haitao and Liu Zhe, Institute of Information Engineering, Chinese Academy of Sciences.

[4]Reserach on Wi-Fi Attack Techniques and Countermeasures, Xu Zhiwei 1 Zheng Baosen2 (1. People's Public Security University of China, Beijing 100038, China; 2. Shandong Police Academy, Shandong 250200, China)

[5] Research on Wireless Network Intrusion Detection Based on Machine Learning" by Ji Yidong, 2019, Beijing University of Posts and Telecommunications.

[6] Research and Optimization of Wireless Network Intrusion Detection Classification Model Based on Recurrent Neural Networks" by Chen Hongsong and Chen Jingjiu, (1. School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China; 2. Beijing Key Laboratory of Knowledge Engineering in Materials Science, Beijing 100083, China).

[7] Application of Machine Learning in Cyberspace Security Research, ZHANG Lei, CUI Yong, LIU Jing, JIANG Yong, WU Jian-Ping '(Department of Computer Science and Technology, Tsinghua University, Beijing 100084) (State Key Laboratory of Network and Switching Technology, Beijing University of Posts and Telecommunications, Beijing,100876)

11. APPENDICES

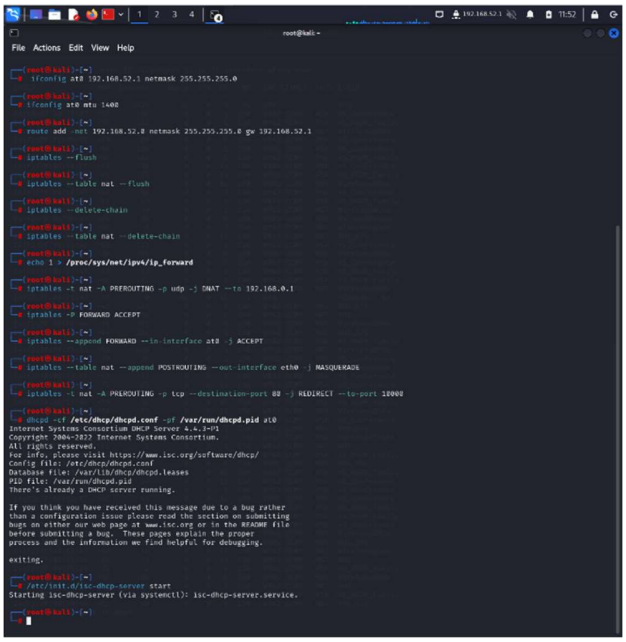
11.1 Implementation Process of fake AP

First, we need to set up the required environment, which is the Kali Linux software platform, to set up a fake wireless access point through the tool called aircrack-ng. We also need a wireless network card that supports monitor mode and packet injection.

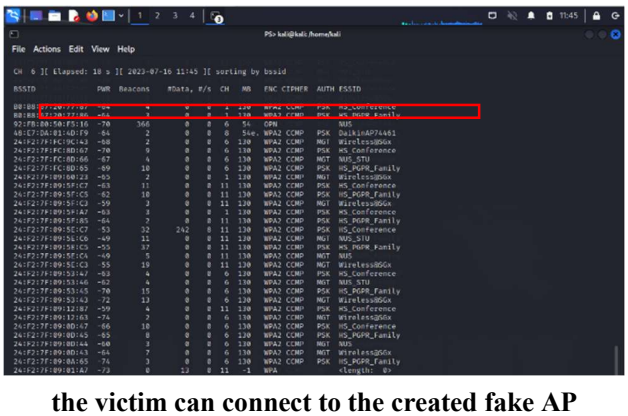
Next, we need to set up the SSID and password for the fake wireless access point to make it look like a legitimate AP. So we can use this fake AP to attack.

Then, we can use tools such as Wireshark to capture and analyze network traffic to obtain sensitive information from users.

fake AP called NUS we created



connect the created fake AP to the network



the victim can connect to the created fake AP

11.2 Implementation Process of fake Beacon

First, we need to et up the Kali Linux software platform and a wireless network card that supports monitor mode and packet injection.

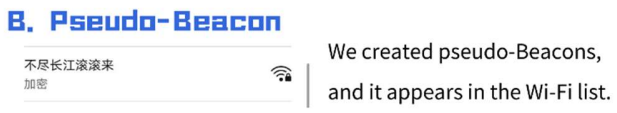
Use the airmon-ng command to enable the monitor mode of the wireless network card.

Use the airodump-ng command to scan the surrounding wireless networks and record the SSID and MAC address of the target network.

Use the mdk3 command to create a fake access point with the same SSID and MAC address as the target network.

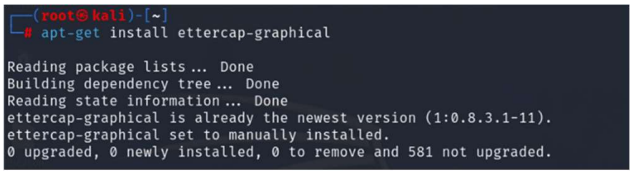
Use the mfdk3 command to start the Fake Beacon Attack and send a large number of fake Beacon frames to deceive users into connecting to the fake access point.

Use tools such as Wireshark to capture and analyze network traffic to obtain sensitive information from users.



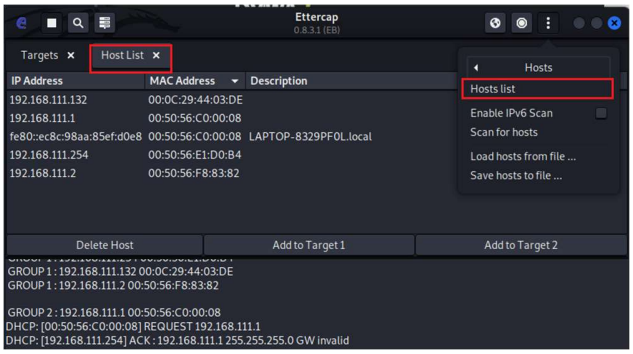
11.3 Implementation Process of MAN IN THE MIDDLE

The installation process is as follows:

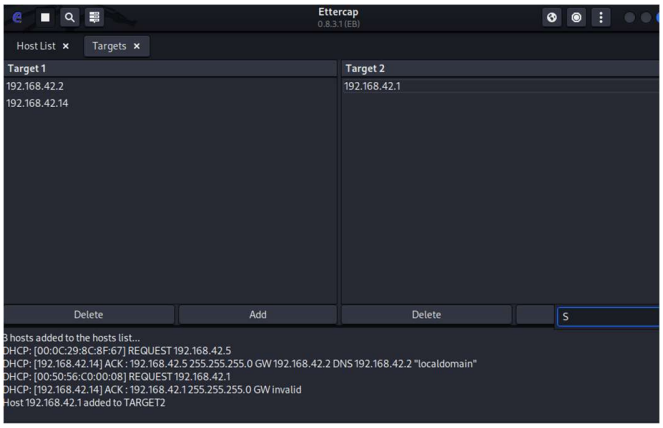


Open the terminal and type "ettercap -G" to open the graphical interface.

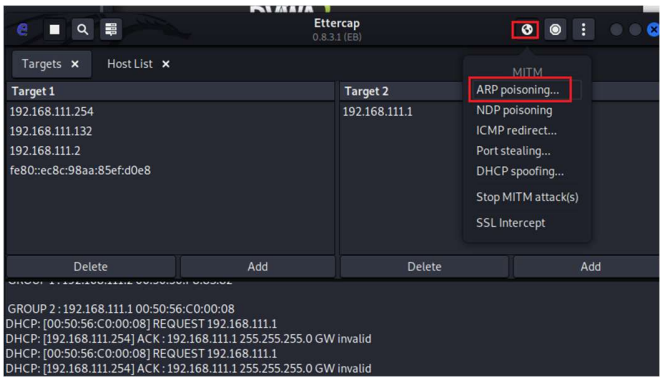
Click on the multi-select button in the upper right corner and select "Targets > current targets" to sniff the hosts on the current network.



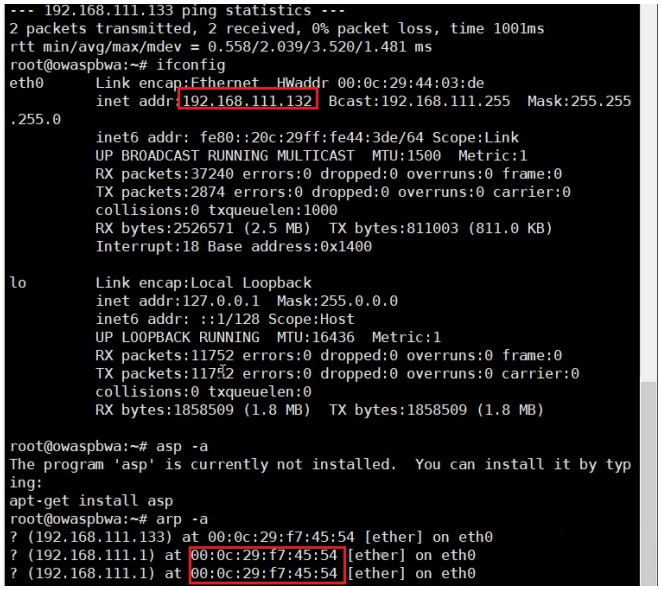
Add the target for the attack by setting all hosts in the network segment as Target1 and the gateway as Target2.



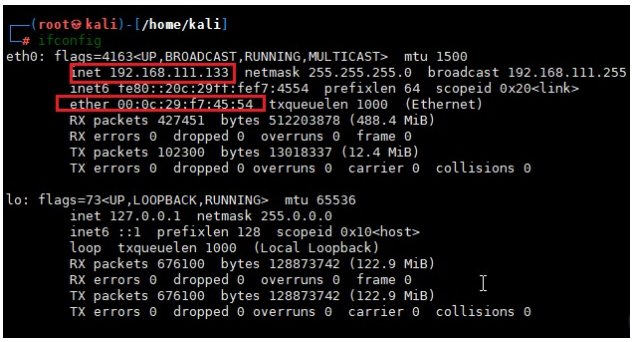
Click on the icon in the upper right corner to perform ARP poisoning attack.



Check the MAC address of Kali and the physical address of the host [192.168.111.132] in the current network. It can be observed that the MAC address of the gateway [192.168.111.1] in the current network will be replaced with the MAC address of Kali [192.168.111.133], allowing the man-in-the-middle to sniff the gateway MAC address of the host within the same network.



Use the MAC address of Ettercap in Kali.

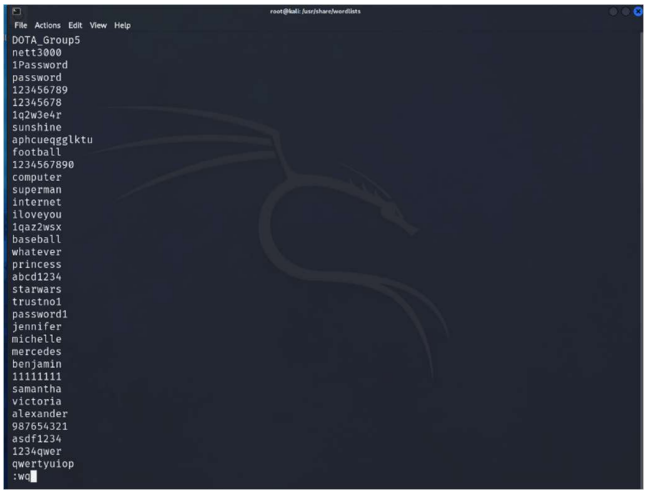


By operating in this network, the data packets will pass through Kali, allowing us to see the plaintext password that is not encrypted.

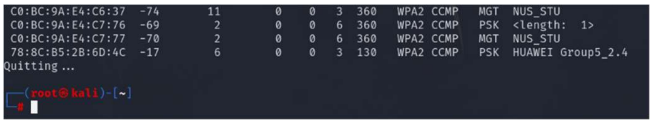
11.4 Implementation Process of Brute Force

Viewing the built-in dictionary files in Kali:

Adding an existing password to the dictionary:

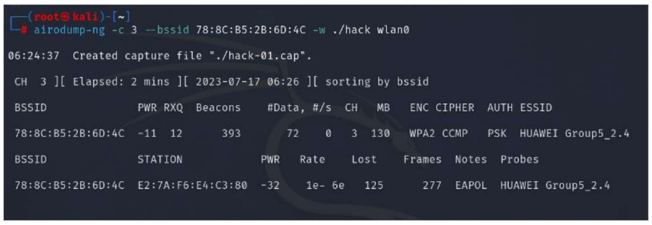


Identifying the experimental WiFi to attack: HUAWEI Group5_2.4

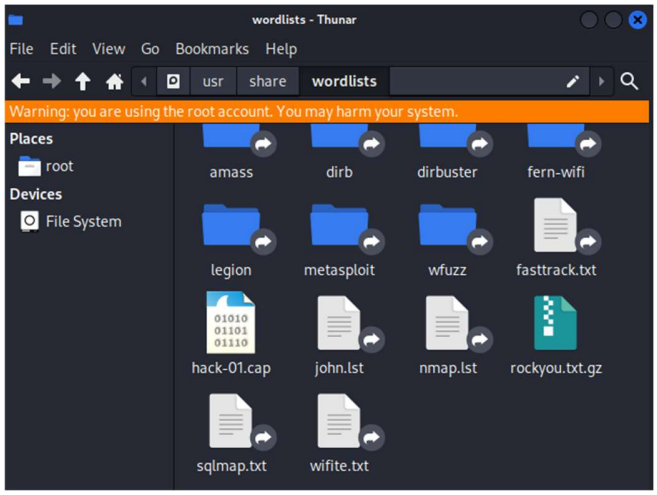


Using the monitor mode of the wireless network card to capture wireless network packets around, we need the handshake packet that contains the WiFi password. This packet is sent when a new user connects to the WiFi.

It can be seen that a handshake packet has appeared and has been captured.



Transfer the captured file "hack-01.cap" to the directory where the built-in dictionary of Kali is located: "usr/share/wordlists/"



Cracking the WiFi password (packet analysis)

It can be seen that the password "DOTA_Group5" has been found.

The Research and Implementation of Phishing AP Detection Techniques

Liu Fazhong
Shanghai Jiaotong University
Shanghai, China
liufazhong@sjtu.edu.cn

Wu Fei
Sichuan University
Chengdu, China
jiashi19@outlook.com

Liang Xiyu
University of Electronic Science and Technology of China
Chengdu, China
UESTC_kk2k4@outlook.com

Lan ZhiQiang
University of Electronic Science and Technology of China
Chengdu, China
lzq1015640307@gmail.com

Li Yaxin
Sichuan University
Chengdu, China
axinli232@gmail.com

ABSTRACT
Sophisticated wireless attacks such as Wifiphishing, Evil twin and so on are a serious threat to Wi-Fi networks. These attacks are tricky enough to spoof users by launching a fake AP pretending to be a legitimate one. The existing intrusion detection schemes are prone to a high rate of false positives as they depend on restricted features. Hence, an efficient intrusion detection system, which considers many more features is needed. This paper describe how we detect wireless network security. It contains how we make attack, defense, and detection.

Keywords
Rogue AP, DNS spoof, snooping, fishing

1. INTRODUCTION

1.1 Background
According to the information released by the WIFI Alliance, in 2018, the global average number of mobile devices per person was 2.4, and experts predict that by 2023, each person will use an average of 3.6 mobile devices. With the continuous increase in the average number of devices per person, the demand for connectivity is also spiraling upwards. To ensure that by 2023, 2.93 billion mobile devices can connect to the internet, the number of public hotspots will multiply. In 2018, there were approximately 169 million public hotspots worldwide, and it is expected that by 2023, this number will rise to a total of 628 million.

While WLAN facilitates people's lives, it also brings about many security issues. The openness of WIFI makes it vulnerable to phishing AP attacks. In a phishing AP attack, attackers set up a rogue AP, which is a network that imitates a legitimate AP and uses the same Service Set Identifier (SSID). In WIFI hotspots, due to the lack of security mechanisms, attackers can exploit vulnerabilities in WIFI client software on laptops or smartphones to launch phishing AP attacks, making such attacks easy to carry out and difficult to defend against. Existing built-in WIFI clients assume that all APs with the same SSID are legitimate and will automatically connect to the AP with the highest Received Signal Strength (RSS) value. If the RSS of the phishing AP is higher than that of the legitimate AP, the client will associate with the phishing AP. There is already software available for laptops that can simulate legitimate APs, posing a severe threat to people's personal privacy and property.

1.2 Contemporary Work
One of the most severe security issues faced by wireless local area network (WLAN) users is the presence of rogue APs. Existing solutions can be categorized into three types: detection techniques based on hardware fingerprint information, time-based measurement, and RSS-based localization detection techniques.

The hardware fingerprint-based detection technique involves using arbitrary information that distinguishes individual devices or a group of devices from others. An authorization list is created by collecting information, including legitimate AP and identity details such as SSID, Media Access Control (MAC) addresses, location, and vendor names. During the detection process, the surrounding APs are scanned, and their fingerprints are collected. Then, these fingerprints are compared with the authorization list to determine the detection result. Szongott et al. [1] used SSID, Basic Service Set Identifier (BSSID), supported authentication, key information, and encryption schemes as fingerprints to detect rogue wireless APs. McCoy et al. [4] classified driver characteristics during the active scanning phase to identify rogue APs based on device distinctiveness.

The time-based measurement technique primarily involves detecting Inter-Arrival Time (IAT), Domain Name System (DNS) resolution delay, Round-Trip Time (RTT), and other parameters in the data packets.Song et al [5] introduced an IAT-based method for detecting rogue APs, utilizing the characteristic that malicious APs typically have at least one extra hop to connect to the internet compared to legitimate APs, using IAT as a detection statistic.

The RSS-based localization detection technique focuses on determining whether the position of the target AP has changed, thereby identifying rogue APs and other attacks. WiFi's RSS values are related to the spatial position of APs. Lim et al. [2] inferred the position of rogue APs by converting received signal strength measurements into distances and using triangulation.

2. ROGUE AP ATTACK IMPLEMENTATION

2.1 Attack Basic Principles

A phishing AP is set up by attackers for illegal purposes. They create a phishing AP and connect it to the internet through nearby legitimate APs. Using readily available tools such as airbase-ng and hostapd, attackers can easily set up a phishing AP. Moreover, attackers can set the phishing AP's SSID to match that of a legitimate AP, enabling evil twin attacks. They can even use wireless sniffing tools to gather information about legitimate APs, making their phishing AP more deceptive and harder to detect. Additionally, attackers can launch attacks against legitimate APs to force normal users to disassociate from them and connect to the phishing AP. Once victims connect to the phishing AP, attackers can execute further attacks using various methods.

2.2 Attack Methods

Attack methods can be broadly categorized into active attacks and passive attacks. In passive attacks, the attacker sets up a phishing AP and falsifies its information based on real information from a legitimate AP, and then does nothing, waiting for victims to connect. Active attacks, on the other hand, involve the attacker actively targeting legitimate APs, preventing users from connecting to the genuine AP, and enticing them to connect to the phishing AP. This paper primarily analyzes active attacks.

2.3 Attack Process

The main attack process involves wireless information detection, phishing AP setup, wireless device attack and Wireless data interception.

(1) Wireless Information Detection: Access points periodically broadcast beacon frames to their surroundings. By monitoring and capturing these beacon frames, one can extract essential information, such as SSID/BSSID. Once the target AP is selected, relevant configuration information can be collected for the next step.

(2) Phishing AP Setup: Using software like hostapd and airbase-ng, a phishing AP can be quickly set up with a wireless network card, forwarding traffic between the internet and the legitimate AP. This method can be implemented using the KaLi Linux system and its equipped tools, requiring only an external wireless network card.

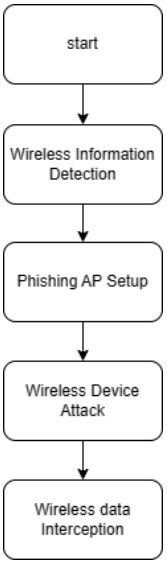


Figure 1: Attack Process

The above description outlines a software-based setup. Alternatively, a hardware device can be used to create an AP, and DNS spoofing can be implemented through a fake DNS server to constitute a phishing AP.This paper utilizes DD-WRT, a Linux-based third-party firmware that can replace the original firmware of certain routers. DD-WRT provides a wide range of advanced features and options that enhance the performance and functionality of the router. By installing DD-WRT on supported routers, users gain access to a plethora of settings and functionality. And with DD-WRT's built-in dnsmasq program, DNS and DHCP configurations can be easily modified via a web interface.

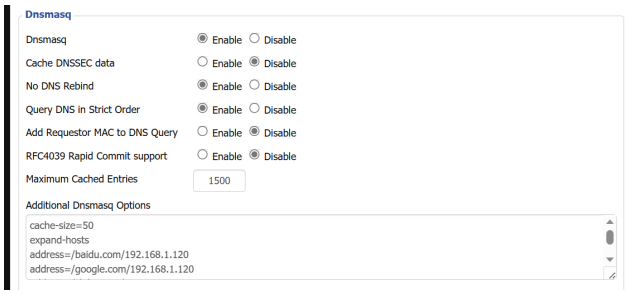


Figure 2: dns on DD-WRT

(3) Wireless Device Attack: Attacking a legitimate AP involves authentication flooding attacks and deauthentication flooding attacks. Authentication flooding attack involves sending a large number of forged authentication request frames to the AP, causing it to disconnect from other users when overwhelmed by the volume. Deauthentication flooding attack involves the attacker sending forged deauthentication packets to the entire network, disconnecting legitimate users from the AP.For Linux systems, the built-in "aireplay-ng" tool is used to achieve the attack effect. By employing this tool, users are forced to disconnect from the legitimate AP, luring them to connect to the phishing AP, thereby achiev-

ing the phishing attack effect.

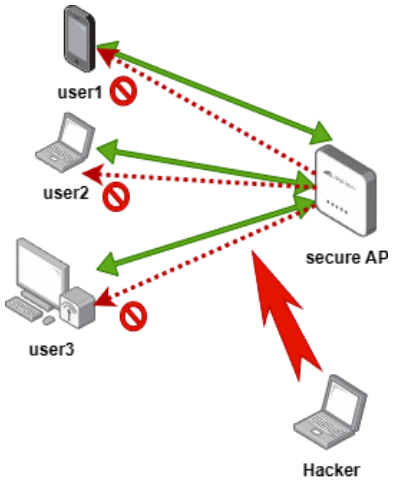


Figure 3: Deauthentication Flood Attack

(4) Wireless Data Interception: Attackers can sniff the interactive data of users connected to the phishing AP. In theory, during the entire eavesdropping process on the targeted user, all users' traffic could potentially be intercepted by the attacker's phishing AP. This means that once users connect to the phishing AP, all their network activities become clear and visible to the attacker who set up the phishing AP.

3. DETECTION RESEARCH AND ANALYSIS

3.1 DNS spoofing Detection Approach

DNS spoofing is also a commonly used technique in phishing APs, where it manipulates the DNS resolution process to redirect users to malicious websites. In order to improve the accuracy of phishing AP detection, this paper proposes a method to identify DNS spoofing by utilizing reliable DNS servers to perform a second query on DNS resolution records and thus confirm their authenticity.

The detection of DNS spoofing involves four main steps: establishing an isolated environment, connecting to a suspicious AP, querying DNS resolution records, and performing a second query through reliable DNS servers.

(1) Establish an isolated environment: Create an isolated environment separate from the host system, which can be achieved through virtual machines. Then, connect to the phishing AP for subsequent testing to ensure the safety of the physical host. Alternatively, you can set up a firewall on your computer to control network traffic, preventing any leakage of private information after connecting to the phishing AP.

(2) Connect to the suspicious AP: Establish a connection to the suspicious AP to prepare for the subsequent steps.

(3) Use a pre-prepared list of common domain names to query DNS resolution records using the DNS server of the network you are connected to. Record the IP addresses obtained from the query. You can use the 'nslookup' command for individual queries or write a Python script for batch queries.

```
nslookup www.example.com
```

(4) Take the obtained IP addresses from the DNS resolution records and perform a second query through one or more reliable DNS servers. These reliable DNS servers should be operated by trusted service providers or organizations. By comparing the results of the first and second queries, you can determine whether DNS spoofing is present.

3.2 Rogue AP Detection Approach

Currently, common Rogue AP attacks configure the same SSID, MAC address, channel, and encryption method as a legitimate AP. They also boost their signal strength to prioritize user connections to the Rogue AP rather than similar secure APs. For users already connected to a secure AP, the attacker may use blocking attacks to force them to disconnect and reconnect to the Rogue AP.

To counter such attacks, this paper proposes a detection method that combines static features, namely wireless device features and configurations, with dynamic features, which involve monitoring the data transmission characteristics between secure APs and users to detect Rogue APs.

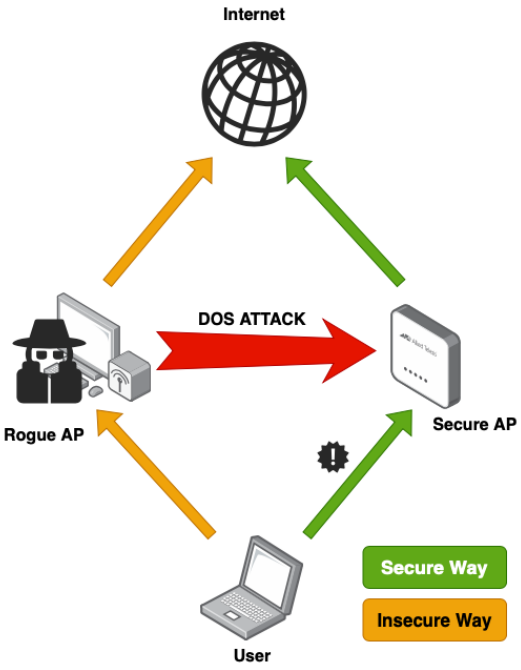


Figure 4: Date Transmission Diagram

3.3 feature extraction

(1) static detection

Static feature extraction involves sniffing the information of the target AP to capture the Beacon frames sent by the corresponding AP to the outside world. From the Beacon frames, static AP information can be obtained.

The Beacon frame header contains BSSID, sequence control, and other information, while the body contains SSID, Vendor, and other information.Through an analysis of the Beacon frame structure, the following information can be identified as static features: SSID, MAC, and Vendor. The details of static feature can be found in appendix part.

In summary, the static features proposed in this paper are shown in the Table 1:

Table 1: Static Features

Feature Name	Feature Meaning	Example
SSID	Service Set Identifier	NUS_STU
MAC	Unique Identifier of AP	24:d6:af:2a:28:c5
Vendor	Supplier of AP in 40,000	00:b5:ca

(2) dynamic monitoring

This paper focuses on the RSS and RTT characteristics of Rogue APs. Rogue APs typically increase their signal strength to entice users to connect to them. Additionally, they act as relays for traffic between secure APs and users, requiring data transmission characteristics between them.

Analyzing these features allows the detection of potentially dangerous APs.The details of dynamic feature can be found in appendix part.

In summary, the dynamic features proposed in this paper are shown in the Table 2:

Table 2: Dynamic Features

Feature Name	Feature Meaning	Example
RSS	Received Signal Strength	-60 dBm
RTT	Round-trip time	65.495 ms

3.4 Rogue AP Detection

When the system detects that more than one AP is using the same SSID, the Rogue AP detection process begins. It starts with static detection, and if it cannot reach a definitive result through static detection, it proceeds to dynamic detection and evaluates AP security based on both results.

(1) Static Feature Verification

The idea of using static features to detect unknown APs is as follows: first, the script is used to monitor the static features (SSID, MAC, Vendor) of multiple APs with the same SSID, and then a comparison is made. Some simple Rogue APs modify only their SSID to match that of a secure AP, without making comprehensive changes to static features. If two APs show differences in their static features, then the existence of a Rogue AP can be determined.

Prioritizing static feature verification improves the efficiency of the system, allowing it to quickly filter out Rogue APs with simpler designs and reducing potential time wastage.

(2) Dynamic Feature Verification

If the system cannot clearly distinguish between two APs during static feature verification, it enters the dynamic feature verification stage. First, the script (or tool) is used to monitor the RSS of the two APs separately to determine their signal strengths.

Simultaneously, the system utilizes the ICMP protocol to send ICMP Echo requests to built-in, common, and stable IP addresses (see the appendix) to request a small data packet. When the target IP receives the Echo request, it immediately sends an ICMP Echo reply message back to the source device, containing the same data packet as in the original request. The system records the time taken for sending the request and receiving the reply, calculating the Round-trip time (RTT). According to the theoretical analysis mentioned earlier, Rogue APs should exhibit RTTs significantly larger than those of secure APs.

After conducting a series of operations, the system compares the RSS and RTT of the two APs and combines the data from static feature detection to give a final judgment result.

3.5 Detector Architecture

In the scenario we have set up, there are two WI-FI networks with the same name in the victim's environment. For victims who have not yet connected to one of the WIFI networks, they are more likely to connect to the Rogue AP due to its stronger signal. For victims who have already connected to a secure AP, the hacker will conduct a deauthentication attack to disconnect the victim from the secure AP.

When the presence of Rogue AP risk is detected, the user can choose to run the detector to determine whether there are Rogue APs in the environment and identify which one is the Rogue AP. When the detector starts running, the first step is static detection. The detector will automatically capture the surrounding Beacon data and further check data with the same SSID. By analyzing the data, it records the static characteristics of different APs, such as SSID, MAC, Vendor, etc. If the static characteristics of two APs are different, it indicates that two different APs are using the same name, and the user will be notified of the existence of Rogue AP risk. If the static characteristics are the same, it may be due to automatic allocation of the same AP signal due to network load, requiring further evaluation.

Then, the dynamic detection phase will be initiated. As this detector involves modifying network environment and system firewall settings, it will request root permissions at this point. The detector will modify the user's firewall settings, restricting their network services during the detection period, allowing access only to pre-defined IPs to prevent potential DNS Spoof attacks. The detector switches between multiple potentially risky AP environments and accesses common IPs using ICMP, records RTT, and compares the results.

4. EXPERIMENT

4.1 Rogue AP Setup

The following provides a detailed description of the process we used to construct a rogue AP and the challenges encountered. We utilized TP-Link's TL-WN722N wireless adapter

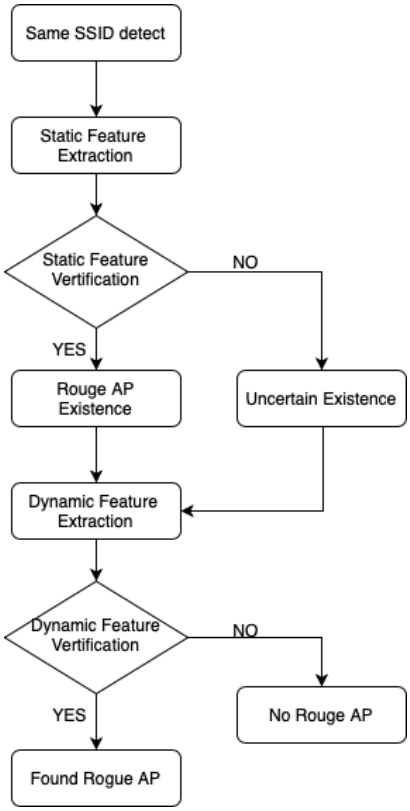


Figure 5: Detection Flow

to connect to the hacker's laptop as the rogue AP. Initially, we needed to put the wireless network card into monitor mode to enable us to sniff data packets within and around the network.

Subsequently, we employed the tool "hostapd" as a software access point, allowing the user to use their wireless adapter to broadcast several access points simultaneously. We modified the "hostapd.conf" file, configuring its SSID, channel, interface, driver, and other settings to make it closely resemble the AP being imitated. After starting the hostapd service, the rogue AP became an open WiFi network that users could connect to.

In the next step, we needed to set up the DHCP service using the "dnsmasq" tool. Dnsmasq acts as a Dynamic Host Configuration Protocol (DHCP) server, resolving DNS requests to or from a machine and also allocating IP addresses to clients. By configuring the "dnsmasq.conf" file, we achieved network connectivity and monitoring functionality for the Rogue AP. After assigning a network gateway and subnet mask to the interface and adding routing tables, we started the DNS server.

Finally, we had to establish the transmission path for victim traffic. This involved forwarding traffic from "eth0," the virtual wireless adapter connected to the internet, to "wlan0mon," the monitoring interface. This enabled the hacker to perform various attacks, gaining complete access to the user's device. IP forwarding was enabled by modifying the "/proc/sys/net/ipv4/ip_forward" file.

Upon completing the above steps, victims could connect to the Rogue AP and access the internet normally. Simultaneously, the hacker could monitor the traffic transmitted by the victim on their terminal and gather information about the websites they visited, enabling further wireless network attacks.

After completing the following steps correctly, when a victim connects to the Rogue AP, the effect shown in Figure 10 will be displayed on the hacker's Kali virtual machine.

```
dnsmasq: cached star-mini.c10r.facebook.com is 31.13.68.35
dnsmasq: query[A] zsecurity.org from 192.168.1.16
dnsmasq: forwarded zsecurity.org to 172.16.152.2
dnsmasq: query[A] www.facebook.com from 192.168.1.16
dnsmasq: cached www.facebook.com is <CNAME>
dnsmasq: cached star-mini.c10r.facebook.com is 31.13.68.35
dnsmasq: query[HTTPS] www.facebook.com from 192.168.1.16
dnsmasq: cached www.facebook.com is <CNAME>
dnsmasq: reply zsecurity.org is 172.66.43.84
dnsmasq: reply zsecurity.org is 172.66.40.172
dnsmasq: query[A] www.google-analytics.com from 192.168.1.16
dnsmasq: forwarded www.google-analytics.com to 172.16.152.2
dnsmasq: query[HTTPS] www.google-analytics.com from 192.168.1.16
dnsmasq: forwarded www.google-analytics.com to 172.16.152.2
dnsmasq: reply www.google-analytics.com is 142.251.10.101
dnsmasq: reply www.google-analytics.com is 142.251.10.100
dnsmasq: reply www.google-analytics.com is 142.251.10.138
dnsmasq: reply www.google-analytics.com is 142.251.10.113
dnsmasq: reply www.google-analytics.com is 142.251.10.139
dnsmasq: reply www.google-analytics.com is 142.251.10.102
dnsmasq: query[A] gitee.com from 192.168.1.16
dnsmasq: forwarded gitee.com to 172.16.152.2
dnsmasq: query[HTTPS] gitee.com from 192.168.1.16
dnsmasq: forwarded gitee.com to 172.16.152.2
dnsmasq: reply gitee.com is 182.255.33.134
dnsmasq: query[A] google.com from 192.168.1.16
```

Figure 6: Attack Effect

4.2 Wireless AP Attack

To launch an attack on a legitimate Access Point (AP), you can use the tool "aireplay-ng" provided with Kali Linux. By using the following command, you can force the target user to disconnect from the AP.

```
aireplay-ng -0 50 -a D2:D2:22:29:79:1D -c 0A
    :88:A3:D7:A5:0E wlan0mon
```

"-a" specifies the MAC address of the target Access Point. '"-c" specifies the MAC address of the target client. After launching the attack, check the test user's device, and you will find that the attacked test user has been disconnected from the AP.

4.3 Rogue AP Detection Experiment

The following provides a detailed description of our testing process for the detector and the challenges encountered. For security and operational considerations, we used an Android mobile hotspot as the secure AP and the same wireless adapter used in the attack experiment as the Rogue AP, and ran the detector for testing.

In the static detection phase, We have developed a script to analyze information obtained from wireless sniffing. Since phishing APs often have the same SSID as legitimate APs, we conducted an analysis by comparing the BSSID of APs with the same SSID. We extracted the first 3 bytes to perform a manufacturer query. As a result, we successfully identified suspicious APs with different manufacturers. However, in real-world scenarios, Attackers can spoof the BSSID and

MAC address to make their phishing AP appear similar to legitimate APs, making it more challenging to distinguish them based solely on the manufacturer information. so this step serves as a preliminary judgment.

```
D:\python\python.exe C:/Users/WuFei/Desktop/detector/info_detect.py
SCAN AP INFO...
INPUT SSID(NAME): NUS_Guest
analyzing manufacturer...
find: 10 different AP with the same name(ssid)
Suspicious AP may exists:check the list
{'Not found', 'Cisco Systems, Inc'}
analyzing signal...
Max RSS: -42  and its manufacturer Not found
check out the list: [-42, -80, -56, -79, -80, -74, -71, -68, -64, -58]

Process finished with exit code 0
```

Figure 7: Detect Vendor And RSS

In the dynamic detection phase, we also predefined a set of IPs and domains as shown in Table 3.

Table 3: Static Features

Domain Name	IP
dbs.com.sg	110.4.47.190
google.com	74.125.68.101
youtube.com	74.125.130.93
twitter.com	104.244.42.129
github.com	140.82.112.4
nus.edu.sg	137.132.84.180
baidu.com	39.156.66.10

After conducting tests on both APs in an isolated environment, the final results are presented in Figure 12. From the analysis, it can be observed that due to the hacker's traffic forwarding, the RTT for accessing common IPs from the Rogue AP is consistently higher than that from the secure AP, showing significant differences. For the common domains such as Github.com and Baidu.com, the RTT is longer due to the distance between their servers and the testing client. However, the time delay caused by the hacker's traffic forwarding is not significantly affected by the server's distance, as expected.

After completing the entire detection process, the detector provides detection results based on the collected data, identifying Rogue APs with risks, informing the user, and restoring the user's network environment and firewall settings.

4.4 DNS Detection Experiment Process

(1) Querying and Saving IP Address Information

Firstly, we read the common domain names stored in the "domain.txt" file and then query the corresponding IP addresses for each domain name. Using Python's 'socket' library, we can easily perform domain name resolution and obtain IP address information.

The following is a partial code snippet for querying IP address information:

```
resolved_ip = socket.gethostbyname(domain)
```

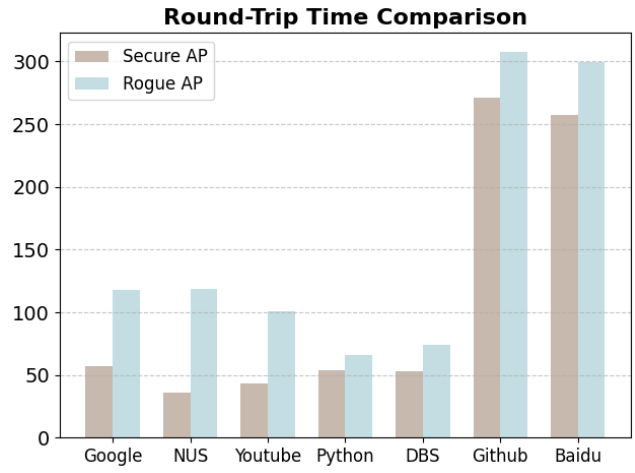


Figure 8: Detection RTT

The final results of this step are shown in the Figure 13. After the queries, the results are stored in the "domain_ip_data.json" file.

```
D:\python\python.exe C:/Users/WuFei/Desktop/detector/dnsInfoGet.py
domain 'google.com' ---> ip: 74.125.68.102
domain 'youtube.com' ---> ip: 192.168.1.120
domain 'bing.com' ---> ip: 192.168.1.120
domain 'twitter.com' ---> ip: 192.168.1.120
domain 'dbs.com.sg' ---> ip: 192.168.1.120

Process finished with exit code 0
```

Figure 9: Dns Record

Therefore, in this step, DNS queries were performed for domain names such as google.com and youtube.com, and the obtained results were stored in a JSON file for further verification.

(2) Verifying the Correspondence between IP and Domain Name

In the second step, we have written another Python script to read the domain names and their corresponding IP address data from the previously saved JSON file and perform verification. We use the same 'socket' library function to re-resolve the domain names and compare the resolved IP addresses with the previously obtained IP addresses to confirm whether they match.

The following is a partial code snippet for verifying the correspondence between IP and domain name:

```
def is_ip_corresponding_to_domain(ip, domain):
    try:
        resolved_ips = socket.gethostbyname_ex(
            domain)[2]
        return ip in resolved_ips
    except socket.gaierror:
        return False
```

The results of this verification process are shown in the Fig-

ure 14 after its completion. By analyzing the experimental results, we can conclude that the resolution result for google.com is correct, but the resolution results for other domain names are all incorrect. Therefore, DNS spoofing exists, indicating that the connected AP is not secure and highly likely to be a phishing AP.

```
D:\python\python.exe C:/Users/WuFei/Desktop/detector/dnsSpoofDetect.py
domain 'google.com' 's ip address '74.125.68.102' is : True
domain 'youtube.com' 's ip address '192.168.1.120' is : False
domain 'bing.com' 's ip address '192.168.1.120' is : False
domain 'twitter.com' 's ip address '192.168.1.120' is : False
domain 'dbs.com.sg' 's ip address '192.168.1.120' is : False

Process finished with exit code 0
```

Figure 10: DNS Verification

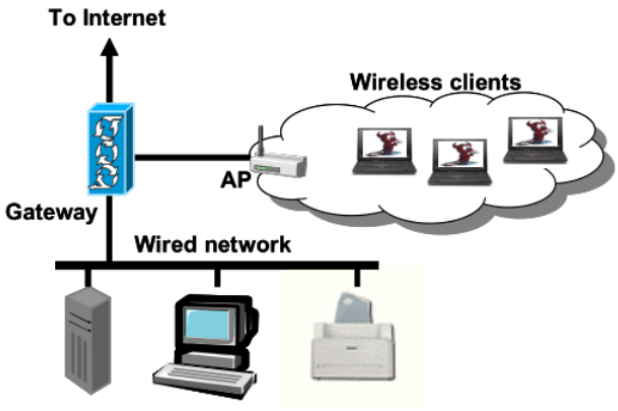


Figure 11: An example network

5. CONCLUSIONS

We investigated the differences between Rogue AP and secure AP. After experiments, we observed that in order to implement attacks, Rogue AP will differ from secure AP in terms of static features such as MAC, SSID, Vendor, dynamic features such as RSS, RTT, and DNS configuration. Based on these characteristics, we have developed and tested a multi feature Rogue AP detection method, which can improve the overall detection accuracy and efficiency by detecting multiple features.[3]

6. REFERENCES

[1] S. C, B. M, and S. M. Metds-a self-contained, context-based detection system for evil twin access points. In *International Conference on Financial Cryptography and Data Security*, pages 370–386. Springer, Berlin, Heidelberg, 2015.

[2] L. H, K. L. C, H. J. C, et al. Zero-configuration, robust indoor localization: Theory and experimentation, 2005.

[3] M. Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.

[4] F. J, M. D, T. P, et al. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security Symposium*, volume 3, pages 16–89, 2006.

[5] S. Y, Y. C, and G. G. Who is peeping at your passwords at starbucks?to catch an evil twin access point. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pages 323–332. IEEE, 2010.

APPENDIX

A. ROGUE AP/DNS SPOOFING THEORY

A.1 Fundamental Theory of WLAN and AP

(1) WLAN structure

WLAN connects all IoT devices through wireless technology without the need for cable media, enabling communication and resource sharing among IoT devices. It usually consists of wireless network cards, Access Controller (AC) devices, AP, computers, and related equipment. The AP is the core device of WLAN, which can connect wired and wireless networks, and provide access and management for wireless terminal devices.

WLAN divides the entire wireless LAN into multiple parts, each of which is called a Basic Service Set (BSS), and each Basic Service Set has a unique identifier called BSSID, which is the well-known physical address (MAC). All components must work together to achieve complementary security.

The Wireless Local Area Network layered model is similar to the wireless network model, but not identical. It is a specific type of wireless network layered model that includes the physical layer, data link layer, network layer, and application layer, among others, which are used to achieve data transmission and application services in WLANs. Compared with the general wireless network layered model, the WLAN layered model focuses more on how to implement specific functions and requirements of WLANs, such as access control, roaming management, and security authentication in WLANs.

(2) Wireless Access Point

Access Point: A wireless access point (WAP), also known as a wireless AP, is a type of wireless communication device that generally serves as a central transmitter and receiver of wireless signals. It is responsible for connecting wireless terminal devices (such as laptops, smartphones, tablets, etc.) to a wireless network and providing network services to them. A wireless AP can also manage access to the wireless network, traffic control, device authentication, security, and other functions.

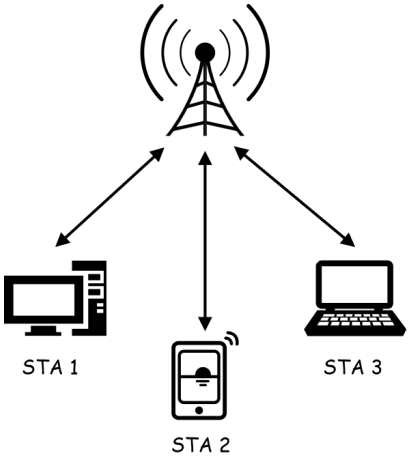


Figure 12: BSS Architecture

Mainstream wireless APs support Wi-Fi, and are used for home, public internet hotspots, and business networks to accommodate wireless mobile devices. A wireless access point can function as a standalone device or as a component of a router. Additionally, if future access requirements increase, wireless access points can be used to extend the wireless coverage range of an existing network.

Service Set Identifier (SSID): usually refers to the name of a wireless AP. Each AP has a unique label that distinguishes it from other APs, allowing users to differentiate between different wireless APs.For example, if you want to connect your laptop or mobile device to the WiFi within a certain area, you may see "NUS_STU", "dotagroup6", "NUS_Guest", and so on in the WLAN list.

Basic Service Set (BSS): as a component of a wireless local area network, it typically consists of an access point and several stations (STAs). Within the BSS, stations can communicate with each other freely.

A.2 DNS Spoofing Theory

The function of DNS (Domain Name System) is to map network addresses (domain names, expressed as strings) to network addresses that can be recognized by computers (IP addresses), so that computers can communicate further and transmit URLs, content, etc.

DNS Spoofing refers to a type of deception in which an attacker impersonates a domain name server. The attacker can use methods such as hacking into the DNS server or controlling a router to resolve the IP address of the victim's target machine to the attacker's controlled machine. As a result, the data originally intended for the target machine is sent to the attacker's machine, allowing the attacker to intercept and even modify the data, thereby collecting a large amount of information. If the attacker only wants to eavesdrop on the conversation data, they will forward all the data to the real target machine for processing and then send the processing results back to the victim machine. If the attacker wants to cause more damage, they will spoof the target machine to return data, so that the victim receives and processes data that is not what they expected but what the attacker expected. For example, if the DNS server resolves the IP of a bank's website to the attacker's machine IP, and the attacker also forges a bank login page on their machine, the victim's real account number and password will be exposed to the attacker.

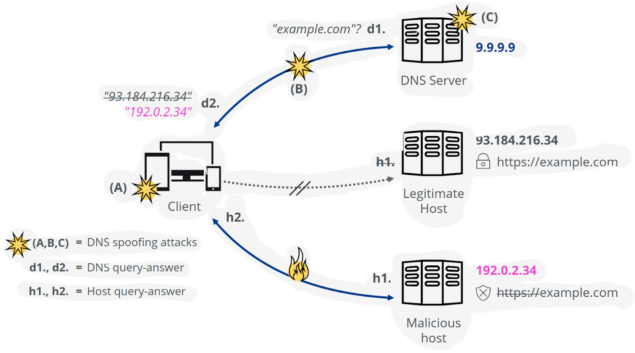


Figure 13: DNS Spoofing Process

B. FEATURE DETAILS

B.1 Static Feature

SSID:

Service Set Identifier. It is important to note that all wireless routers or APs produced by the same manufacturer use the same SSID. In public places such as schools, airports, and stations, public wireless networks provided by operators or schools often have fixed SSIDs, such as "Singtel WIFI," "StarHub," "Wireless@SG," "ChangiWiFi," "NUS_STU," etc. Some attackers set up phishing APs with these names to deceive users into connecting to them. Hence, SSID can be used as a feature for detection.

MAC:

It is the unique identifier of an Ethernet or network adapter on a network. It distinguishes different network interfaces used for various network technologies. In the OSI model, MAC addresses appear in the Media Access Control Protocol sublayer. Unlike IP addresses, MAC addresses are permanent and cannot be changed. Generally, if attackers do not deliberately forge the MAC address to match that of the legitimate AP, the MAC addresses of the phishing AP and the legitimate AP will be different. Thus, MAC address can be used as a feature for detection.

Vendor:

It represents information about the AP's supplier. Each supplier has different characteristics and distinct marks. Attackers find it challenging to modify this data because once wireless devices are manufactured, all supplier data is written into the chips. When attackers use third-party open-source firmware to flash wireless routers, the supplier data of these routers is usually deleted to facilitate the acquisition of permission for the built-in wireless phishing AP, resulting in differences between APs. Thus, the vendor identifier can be used as a feature for detection.

B.2 Dynamic Feature

RSS:

Received Signal Strength reflects the energy received by the probing device from data packets. To lure users to connect, a Rogue AP often sets a high RSS value. Thus, RSS can be used as a feature for detection.

RTT:

Round-trip time (RTT) is an important performance metric in computer networks, measuring the time it takes for data to be sent from the sender to the destination host and then back to the sender. It is a critical indicator of network latency. RTT is calculated by sending a small data packet (usually a ping packet) from the source host to the destination host, and the destination host immediately sends a confirmation response. RTT is the time it takes for this data packet to travel back and forth.

Implementation and Analysis of BREACH Attack System

LIU Jin-jian, GUO Zi-yun, YANG Zong-qi, LI Jun-le and XU An-jun

***Abstract*—In the transmission of network information, compression algorithms such as DEFLATE are commonly used to minimize bandwidth consumption. However, these algorithms pose a risk of secret information leakage. The Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) attack exploits pre and post compression changes in the HTTP response information body to steal information, displaying typical SSL/TLS attack characteristics. This project aims to raise awareness and educate users about the BREACH attack. It involves developing a user-friendly interface and conducting a comprehensive simulation of the BREACH attack, analyzing its characteristics and mechanisms, and comparing it with similar attacks to gain a deeper understanding. The project's findings will be summarized, and prevention suggestions will be provided to enhance network security awareness.**

***Index Terms*—compression, BREACH attack, SSL/TLS, security**

I. INTRODUCTION

Compression is a useful technique for saving the bandwidth and widely used in various systems, while acting as a side-channel[1] without any preventive measures. However, when plaintext data is compressed before encryption, the length of the resulting ciphertext can inadvertently reveal information to potential attackers. This vulnerability has led to the emergence of attacks that exploit this side-channel information.

The BREACH attack is one of them, as a compression side-channel attack, which targets information compressed in HTTP response bodies. This attack can be used to extract login credentials, anti-CSRF tokens and other sensitive, personally identifiable information from SSL-enabled websites.

This paper primarily focuses on give a completely sight in the system we implement, in addition to that, the paper also providing a comprehensive analysis of the BREACH attack, in comparison to other similar attacks. The aim of the paper is to present a detailed examination of the complete BREACH attack, highlighting its intricacies and implications.

A. Overview of CRIME and BREACH

CRIME (Compression Ratio Info-leak Made Easy) and BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) are two security attacks related to SSL (Secure Sockets Layer) and TLS (Transport Layer Security).

The CRIME attack, introduced by Rizzo and Duong in 2012[2] targets SSL/TLS and exploits the compression ratio of HTTP responses to discover session tokens or other sensitive information. It can be triggered when a user visits a website controlled by an attacker, and the attacker manipulates the victim into sending requests to a specific URL, leading to significant information leakage and potential loss[4].

BREACH, on the other hand, is another attack on SSL/TLS that was discovered in 2013[3]. It leverages the compression of HTTPS responses to extract sensitive information, such as CSRF tokens or authentication credentials, from the compressed responses. The attack takes advantage of the way compression algorithms work by analyzing the variations in the compressed data size.

The project mainly focus on BREACH and the process and mechanism in detail will be discussed in the subsequent sections.

L. JJ is a student at the College of Software, Sichuan University, Chengdu, China.

G. ZY is a student at the School of Cyber Science and Engineering, Sichuan University, Chengdu, China.

Y. ZQ is a student at Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China.

L. JL is a student at School of Cybersecurity, Northwestern Polytechnical University, Xian, China.

X. AJ is a student at College of Informatics, Huazhong Agricultural University, Wuhan, China.

B. *Our contribution*

Our project team's objective is to present the characteristics and mechanisms of the BREACH attack for educational purposes. We have successfully implemented the BREACH attack model and constructed a WEB platform with user-friendly pages, considering educational aspects, to make the system accessible and rewarding for any user. Furthermore, we conducted research to analyze and compare the BREACH attack with other popular SSL tools, and we have compiled a comprehensive summary of our findings

II. BREACH ATTACK

A. *DEFLATE Compression Algorithm*

The BREACH attack is mainly focus on DEFLATE compression algorithm, which is a lossless data compression file format that uses a combination of LZ77 and Huffman coding.

For LZ77, it basically Using slide window contains data, moving forward as progress. Main process are as follows[5]:

First, the algorithm begins with an empty buffer and an empty output stream. Then, it searches for patterns of data within the sliding window that match the current input data being processed. When a match is found, it represents it as a pair (length, distance), where "length" indicates the number of matching characters and "distance" shows the offset from the current position to the start of the match within the sliding window.

Next, the algorithm outputs the (length, distance) pairs for matches and the literal characters that do not match any previous data. This output is designed to be stored or transmitted efficiently. As new data is processed, the sliding window slides forward, discarding the old data and adding the new data to the window.

The process repeats itself as the algorithm continues to search, encode matches, output data, and update the sliding window with each iteration.

For Huffman coding, which is the process of finding or using such a Huffman code, which is a particular type of optimal prefix code that is commonly used for lossless data compression.

Mechanism

Breakthrough point：The length of the compressed data is still visible after compressing with the DEFLATE algorithm,

The attacker needs to Inject his guesses of the secrets (using JavaScript) into the HTTP response bodies at the first time, then observe the time when responses would be highly compressed, and the output length differs, means the guess matches, after that, it's time to detect the secret information, and finally the complete secret has been extracted.

III. IMPLEMENTATION

A. *Overview*

In this part, we mainly simulate the attack and display it in webpage, adding functions so that users can easily interact with it.

B. *Implement the Web*

We use html, css, javascript to develop the webpage and interactive functions, the main steps and figures related are shown as follows.

Firstly, we trying to design and finish the layout of the webpage.

Figure 1. Layout of the webpage

Then, design the style of different elements and add appearance elements on the page, find the appropriate background image, then fill in the text and make some adjustments.

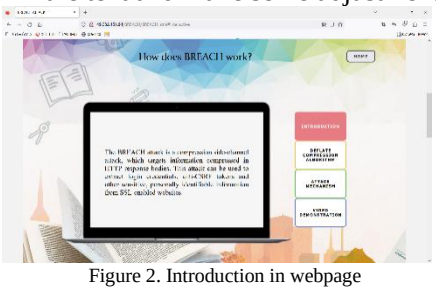


Figure 2. Introduction in webpage

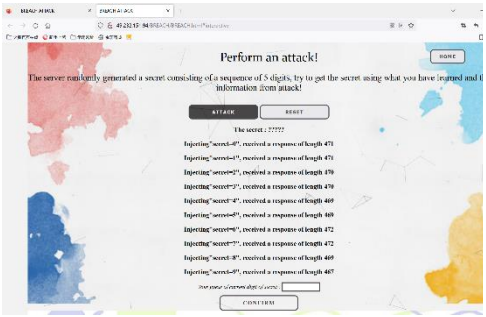


Figure 3. Interaction design in webpage

In order to show more information and raise the concern of prevention, we also give some advice of preventing such an attack.

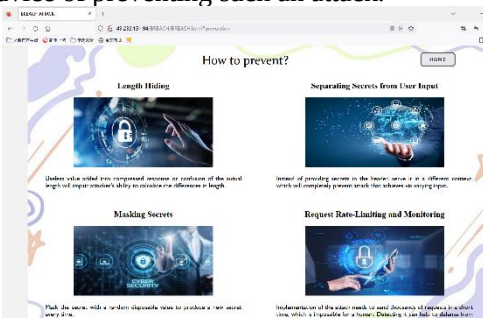


Figure 4. Preventing advice shown in webpage

C. *Implement the Attack*

This lab system is based on the Docker virt ualization platform and Tencent Cloud lightw eight servers. The Docker virtualized containe rs are deployed on Tencent Cloud lightweight servers, simulating a victim host within one Docker container, while using the cloud serve r as the attacking host to launch attacks agains t the Docker container.

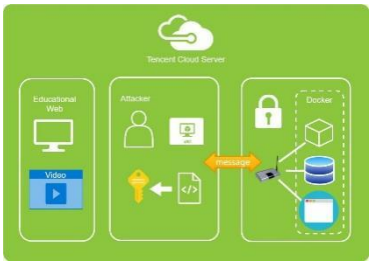


Figure 5. Attack implementation architecture

In detail, we first to gain control of the victim's network. In this system, the attacker is able to inject code to the victim's machine for execution, and note that we primarily assume that we have already gained control of the victim's host network and successfully injected attack script into the victim's host, gaining partial control over the victim's network.

The attack script initiates multiple requests to the target website, which are intercepted and analyzed by the attacker. Since the attacker script runs in a different context from the target website, it is bound by the same-origin policy and unable to access the plaintext or encrypted responses directly. However, the encrypted requests and responses are accessible to the sniffer through direct network access.

By comparing the lengths of the encrypted data, the sniffer can deduce information about the corresponding plaintext lengths and their relationships.

A successful attack completely decrypts a portion of the plaintext. The portion of the plaintext which the attack tries to decrypt is the secret. That portion is identified through an initially known prefix which distinguishes it from other secrets. Each byte of the secret can be drawn from a given alphabet, the secret's alphabet.

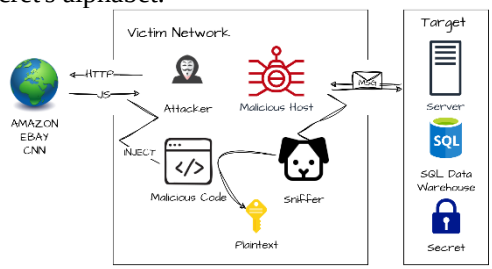


Figure 6. Attack schematic diagram

IV. RESULT AND ANALYSIS

A. *The result of our system*

The system successfully worked. During user operations, the occurrence of BREACH attacks and web page interactions can occur correctly

B. *Analysis*

The analysis of BREACH attack mainly focuses on the advantages and disadvantages compared to other attacks related to it, and given to the characteristic of this attack, methods of preventions are also provided [6]

Advantages of the BREACH Attack are shown as follows:

• Effective Information Extraction: The BREACH attack can effectively extract sensitive information from vulnerable web applications. By manipulating the size of compressed responses, attackers can infer specific parts of the plaintext, enabling them to extract valuable data.

• Low-Cost Attack: Implementing the BREACH attack does not require

sophisticated tools or substantial computing resources. Attackers can use relatively simple scripts or tools to perform the attack, making it accessible to a wide range of adversaries.

• Difficult to Detect: Detecting a BREACH attack is challenging since it leverages legitimate TLS/SSL communications. As a result, traditional intrusion detection systems may struggle to identify the attack, allowing it to remain undetected for extended periods.

Disadvantages of the BREACH Attack are shown as follows:

• Limited Targets: Not all web applications are vulnerable to the BREACH attack. Several conditions must be met for a successful attack, such as the presence of specific types of sensitive data and the usage of HTTP response compression.

• Mitigations Available: The BREACH attack received significant attention upon its discovery, leading to various mitigation strategies being developed. For example, disabling HTTP compression, randomizing secrets, or implementing anti-CSRF tokens can help protect against this attack.

• Complex Execution: While the attack concept is relatively simple, carrying out a successful BREACH attack can be more challenging in practice. Attackers need to find and exploit vulnerable web applications and manipulate the victim's browser to send cross-origin requests.

C. Prevention:

The available prevention methods against this attack are currently limited. The article briefly introduces these methods without delving into extensive details, and they primarily include the following:

The first is length hiding: This method involves adding random padding or extra characters to the plaintext data before compression, making the resulting ciphertext lengths less predictable and harder for attackers to infer sensitive information.

Disabling compression is also a useful method: By disabling compression in the communication protocols, the vulnerability exploited by the BREACH attack can be eliminated. However, this may lead to increased bandwidth usage.

Furthermore, separating secrets from user input can also be useful: Keeping sensitive information, such as session tokens or authentication credentials, separate from user-controllable input can help reduce the attack surface and make it more challenging for attackers to exploit the compression vulnerabilities.

For more approaches, are masking secrets and request rate-limiting and Monitoring: the pre approach involves obfuscating the sensitive data in a way that makes it challenging for attackers to recognize or distinguish specific patterns related to the secrets. For the post one, implementing rate-limiting measures on incoming requests and monitoring for suspicious activity can help detect and mitigate BREACH attacks in real-time.

Up to 2021, the BREACH attack is still very effective, although exact numbers have not been found, there are not a few websites with weaknesses in this regard, and the protection methods are mostly limited to the few described above, which has research prospects.

D. Attacks comparison

This section provides a discussion on some attacks that most of which are stated above. Our discussion is based on specific criteria[7] that are selected to evaluate the current state of the art, which are:

• Weakness: Identifies the vulnerability of the system that has been attacked.

• Effect: Identifies the action that has been enforced by the attack.

• Limitation, to find a limitation on the current solution for the attack.

Such discussions on these criteria are shown in TABLE 1 as follow.

TABLE1: ATTACKS COMPARRISON

Attack	Effects	Weakness
BEAST	Recover the cookies	Predictable IV in CBC mode
CRIME	Discover session token or other secret information	Compressed size of HTTP requests
TIME	Infer the compressed payload's size	Compressed size of HTTP response
BREACH	Extracting the secrets behind the HTTP response	Compressed size of HTTP response
Lucky 13	Get the decrypted message without key	Pad is not include in MAC

V. CONCLUSION

BREACH attack is a typical compression-based side-channel attack that has been widely discussed in the internet security community. Our work is to implement the attack and take it as an educational use for all of the users have a deeply understanding of the attack and hopefully enhancing awareness of network security prevention, which is becoming increasingly important in contemporary times. In addition, our project's analysis and comparison of BREACH attacks can also predict potential forms of future network attacks at a certain level, hoping to be helpful for future research on related network security

References

[1] Kelsey, J: Compression and information leakage of plaintext. In Daemen, J., Rijmen, V., eds.: FSE 2002.

[2] Rizzo, J., Duong, T.: The CRIME attack Presented at ekoparty 12. http://goo.gl/mlw1X1

[3] Gluck, Y., Harris, N., Prado, A.: SSL, gone in 30 seconds: A BREACH beyond CRIME. In: Black Hat USA 2013.

[4] I. Sankalpa, T. Dhanushka, N. Amarasinghe, J. Alawathugoda and R. Ragel, "On implementing a client-server setting to prevent the Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) attacks," 2016 Manufacturing & Industrial Engineering Symposium (MIES), Colombo, Sri Lanka, 2016.

[5] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," in IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, May 1977.

[6] ATTACKS ON SSL A COMPREHENSIVE STUDY OF BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 BIASES, Pratik Guha Sarkar, Shawn Fitzgerald, San Francisco, August, 2013.

[7] A. E. W. Eldewahi, T. M. H. Sharfi, A. A. Mansor, N. A. F. Mohamed and S. M. H. Alwahbani, "SSL/TLS attacks: Analysis and evaluation," 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), Khartoum, Sudan, 2015.

A Study on Malicious Access Point Detection: ARP Spoofing versus DNS Spoofing

Zhao Huanle
Harbin Institute of Technology, Shenzhen
zhaohuanle0@gmail.com

Yu Chenglong
Sichuan University
yuchenglong@stu.scu.edu.cn

Zhang Yiang
Sichuan University
kasemuffin@qq.com

Zhang Jianfan
Wuhan University
1243046148@qq.com

Cai Xinyuan
Huazhong University of Science and Technology
caixinyuan2233@gmail.com

ABSTRACT
ARP request packets are disseminated to all hosts on the network, and ARP responses packets are not authenticated, which means that attackers can take advantage of it by performing ARP spoofing. Similarly, attackers can modify the correspondence between a domain name and an IP address during DNS domain name resolution to launch DNS spoofing. In order to detect the above spoofing attacks on a computer, we purpose two methods respectively, named Resending ARP Packet for detecting ARP spoofing and Polymerized Trusted DNS Verification with Cached IP Range Table to detect DNS spoofing. In our experiments, ARP spoofing via the *arpspoof* tool [10] is detected, while 98.5% of DNS spoofing is detected in 200 times attacks.

Keywords
Computer Networks, Computer Security, ARP Spoofing, DNS Spoofing, MITM,

1. INTRODUCTION
The ARP protocol [1] combines the link and IP layers, and its correctness maintains the proper functioning of the network. But this protocol does not verify the authenticity of ARP packets, so it can be attacked by man-in-the-middle. (U. Meyer and S. Wetzel, 2004 [6]; L. B. Kish, 2006 [7]; K. Hypponen and K. M. Haataja, 2007 [8]; K. Ouafi et al. 2008 [9]).

DNS [5] protocol maintains the mapping relationship between domain names and IP addresses at the application layer, so users can access the Internet directly through domain names. Similarly, DNS servers can also be attacked by man-in-the-middle

In this paper, for the above possible attacks, we have taken two respective methods to target the spoofing attacks, which are the Retransmit Packet Method and Polymerized Trusted DNS Verification with Cached IP Range Tables.

The rest of the paper is organized as follows: Section 2 gives a brief overview of the background information. In Section 3, we describe some related work. We provide our methods to detect ARP spoofing and DNS spoofing respectively in Section 4, and implement and check results in Section 5. We conclude our paper and describe future work in Section 6

2. BACKGROUND
2.1 ARP
Address Resolution Protocol [1], or ARP (Address Resolution Protocol), is a TCP/IP protocol for obtaining physical addresses based on IP addresses. When a host sends a message, it broadcasts an ARP request containing the target IP address to all hosts on the local network and receives a return message to determine the physical address of the target; after receiving the return message, it stores the IP address and the physical address in the local ARP cache and retains them for a certain period of time, so that it can directly query the ARP cache when it is requested next time in order to save resources. The address resolution protocol is based on mutual trust among hosts in the network. Hosts on the local network can send ARP answer messages on their own, and when other hosts receive an answer message, they do not detect the authenticity of the message and then enter it into the local ARP cache;

Assuming that hosts A and B are on the same network segment and host A wants to send a message to host B, the specific address resolution process is as follows:

1. Host A first checks its ARP cache table to determine if it contains an ARP table entry corresponding to host B. If the corresponding MAC address is found, Host A directly utilizes the MAC address in the ARP table to frame the IP packet and sends the packet to Host B.
2. When host A needs to communicate with host B but cannot find the corresponding MAC address in its ARP table, it initiates the ARP resolution process. Firstly, host A caches the data message meant for host B and sends an ARP request message through broadcast. As the ARP request message is broadcasted, all hosts on the network segment receive it. However, only the requested host (host B) processes the request, while others simply ignore it.
3. Host B sends the ARP response message to host A in unicast mode, which includes its own MAC address
4. After receiving the ARP response message, host A adds the MAC address of host B to its own ARP table for forwarding subsequent messages, and

encapsulates the IP data packet before sending it out.

2.2 ARP Spoofing

ARP requests are sent in the form of broadcasts, and hosts on the network can independently send ARP response messages. When other hosts receive the response message, they do not detect its authenticity and record it in the local MAC address translation table. This allows attackers to send fake ARP response messages to the target host, thereby tampering with the local MAC address table. ARP spoofing can lead to communication failure between the target computer and the gateway, but also lead to communication redirection. An example of such an attack is shown in Figure 1.

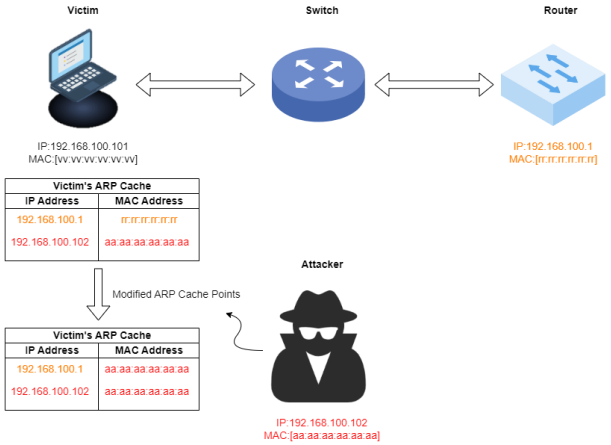


Figure 1. An example of ARP spoofing.

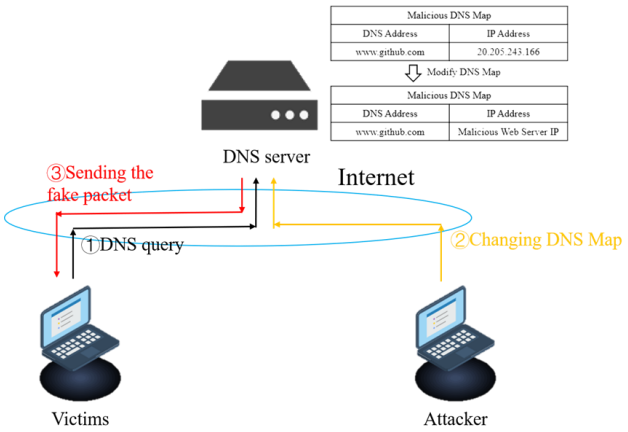


Figure 2. An example of DNS mapping table poisoning.

2.3 DNS

The main function of DNS, also known as Domain Name System, is to translate domain names to IP address so browsers can load Internet resources, making it easier for people to access the Internet.

When a user enters a domain name into a network application, the local DNS resolver checks its cache at first. If cache has the IP address corresponding to the domain name, the resolution is completed immediately. If not, the DNS resolver proceeds to resolve the domain name by querying authoritative DNS servers.

2.4 DNS Spoofing

As with ARP, if an attacker modifies the correspondence between an IP address and a domain name, and the DNS reads an incorrect set of records (Figure 2), it will be affected and will tell others about the incorrect records. Then, for a period of time, all the affected DNS servers will give out incorrect information.

3. RELATED WORK

3.1 Current Detection of ARP Spoofing

In fact, there are already many mature programs for ARP spoofing detection, but some of them are not suitable for our project.

Bruschi et al. proposed the S-ARP protocol [2], but the biggest drawback is that we need to change the network stack, which reduces scalability. Therefore, S-ARP uses DSA and creates an additional computational overhead.

In Passive detection, when we sniff the ARP packet, if there is a change in IP-MAC mappings, we can conclude that the machine is under ARP spoofing. The most famous application is ARPWATCH [3]. Actually, using passive detection is unreliable, due to time lags and being discovered instead of being reported as attack traffic

3.2 Current Detection of DNS Spoofing

As researches go deep in the field of computer networking, hackers are more flexible when launching an DNS spoofing attack. Luckily, we already have field-tested methods to detect those attacks. The research and implementation of proxy-based defense to DNS spoofing attack [4] servers as an inspiration to our project. But there are still drawbacks when it comes to our project. Firstly, static cache in the server can't adapt to the dynamic state of network. In addition, using none-secured socket is a safety hazard itself. Combining passive detection by a challenge-answer model and active detection by maintaining a set of "live" cache updated daily, this model was equipped with the capability to keep in touch with the latest news. However, the time-gap was too long for an attacker to launch an attack. And as cache is stored in the local machine, it could be easily modified. Therefore, we use a set of popular domain's public IP lists to update our local cache to ensure that every query is responded with up-to-date data from authorized entity through secured communication channels.

4. METHODS

Based on related work, we propose two detection schemes for the two spoofing attacks

4.1 Resending ARP Packet

In reality, we can't tell if we are being attacked by comparing the gateway mac address in the ARP cache with the attacker's mac address, because it is possible that the attacker's mac address does not exist in the cache.

Since in ARP spoofing the attacker needs to send ARP packets to the gateway and then to the victim machine, we in fact update the ARP cache with the gateway's IP address corresponding to the attacker's mac address. An obvious way to detect this is to send an ARP packet to the gateway several times to get the gateway's mac address, and compare it with the mac address in the previous

packet, if it is not the same, it means that it has suffered an ARP spoofing attack.

4.2 Polymerized Trusted DNS Verification with Cached IP Range Tables

As shown in the Figure 3, the experiment system consists of four parts - the victim, the hacker's infrastructure, VeriDNS infrastructure and trusted DNS servers. First of all, the hacker enables the personal access point (AP) function of a smartphone and extends its signal with a wireless router. This router acts as a malicious AP waiting for the victims. Then the hacker logins to the router, configures its DHCP server, and points the default local DNS server to the hacker's malicious server. In this case, the malicious server not only acts as the malicious DNS server, but also a malicious web server which will provide the victim with misleading content. When a victim connects to the wireless router, his / her default local DNS will be misconfigured to hacker's malicious server. When the victim browses the Internet, the hacker can forge DNS response messages and direct the victim to malicious websites.

However, such attack can be easily detected by our VeriDNS infrastructure proposed in this paper. This infrastructure is made up of four parts - client, branch server, root server, and trusted third-party DNS server. The client process runs on the victim's machine and sniffs the network traffic endlessly. The client can capture all DNS A resource records (RRs) and send them to a local VeriDNS branch server for verification. In order to achieve that, the client must know the IP address of the local branch server. Therefore, the client process will query the local branch servers' IP address from a global-accessible VeriDNS root server when it starts. This procedure is called branch server address query. Upon receiving the query request from the client, the root server will determine the country or the region where the client locates and sends the client IP addresses of branch servers which locates in the same country / region with the client. This procedure is called branch server address answer. After obtain the local branches' address, the client will send its A RRs to the local branch server continuously and ask the branch server to verify them again. This process is called VeriDNS request. Then the branch server will communicate with some trusted third-party public DNS servers to verify the A RRs. Once the result is determined, the branch server will tell the result to the client. This last procedure is called VeriDNS response. Since branch servers locate in the same country / region with the client, the same A RRs should be obtained even though the load balancers exist. The place to deploy the branch servers should be carefully chosen because the branch server should be reachable from both client's end and trusted DNS server's end, which guarantee the availability of the VeriDNS service. Since any communication is carried through a secure TLS tunnel, VeriDNS can also guarantee the confidentiality and integrity of the data and free from any interception and spoofing.

It is worth noting that local cache strategy is also adopted in our design. There are mainly three concerns about the online verification method. Firstly, DNS query is time consuming. Double queries will slow down the name service infrastructure. Secondly, DNS query has the property of localization. 80% of the queries lies on the 20% of the domain names. These domain names mostly belong to those well-known companies, such as Google, Microsoft, Cloudflare, and Fastly. Furthermore, there is little incentive for hackers to direct victim's traffic to those IP addresses belonging to those well-known companies. Third, since these well-known companies has large quantities of IP address to deploy load balancers, it is very hard to get all IP addresses for one given domain name in one trial, which raises the false positive rate. Therefore, we get the official IP range list from Google, Microsoft, Cloudflare and Fastly and update them periodically to speed up our infrastructure and improve the accuracy of verification.

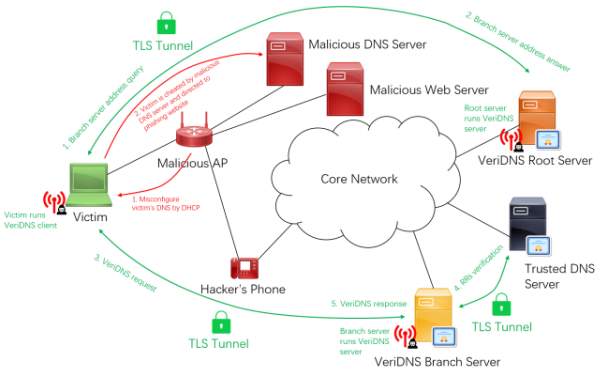


Figure 3. DNS Spoofing Attack flow.

5. IMPLEMENTATIONS and RESULTS

5.1 Detection of ARP Spoofing

5.1.1 Environment Setup

A basic demo of detection of ARP spoofing consists of three parts: the Victim (172.20.10.8), the Attacker (172.20.10.13) and the router (172.20.10.1). The specific network situation that attacker scans is shown in Figure 4.

```
(base) young@r-239-103-25-172 ~ % nmap -sP 172.20.10.1/24
Starting Nmap 7.94 ( https://nmap.org ) at 2023-07-23 18:55 +08
Nmap scan report for 172.20.10.1
Host is up (0.0045s latency).
Nmap scan report for 172.20.10.8
Host is up (0.051s latency).
Nmap scan report for 172.20.10.13
Host is up (0.00047s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 21.28 seconds
```

Figure 4. Using *nmap* tool to scan hosts.

```
PS C:\Users\HECATE> arp -a

接口: 172.20.10.8 --- 0x12
  Internet 地址         物理地址              类型
  172.20.10.1           0a-87-c7-78-60-64     动态
  172.20.10.13          3c-a6-f6-56-69-20     动态
  172.20.10.15          ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

Figure 5. Victim's ARP cache before being attacked.

5.1.2 Performing ARP spoofing and detecting

Before launching an ARP spoofing attack, we use the following command to view the victim's ARP cache: *arp -a*, and the result is shown in Figure 5.

When the attacker finds the IP address of the victim's machine, i.e. 172.20.10.8, the attack is carried out via the *arpspoof* tool [10]. The attacker enters: *sudo arpspoof -i en0 -t 172.20.10.8*

172.20.10.1 on the command line. and then checks the victim's ARP cache, the result is shown in Figure 6. it is very clear that it is indeed being attacked.

At the same time, when we run our own detection program, we can also find that the victim is being attacked, and the victim can resend ARP packets to know the attacker's mac address and router's mac address as shown in Figure 7, which are 3c-a6-f6-56-69-20 and 0a-87-c7-78-60-64.

```
PS C:\Users\HECATE> arp -a

接口: 172.20.10.8 --- 0x12
  Internet 地址         物理地址              类型
  172.20.10.1           3c-a6-f6-56-69-20     动态
  172.20.10.13          3c-a6-f6-56-69-20     动态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

Figure 6. Victim's ARP cache after being attacked.

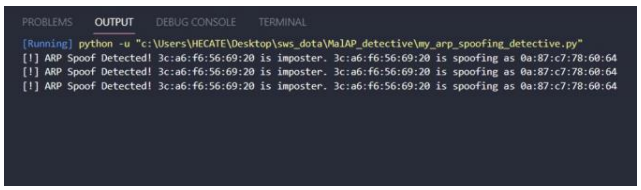


Figure 7. Victim resend packets to detect ARP spoofing.

5.2 Detection of DNS Spoofing

5.2.1 Environment Setup

System Overview: Our system logically consists of three parts: Root VeriDNS server, Branch VeriDNS server and client host.

Root server runs Ubuntu 20.04 LTS (Long Time Support) OS, with a 64GB disk and a 4GB RAM. We set up our root VeriDNS server in Hong Kong at 20.239.49.221, to ensure the access from mainland China to be successful.

Branch server is set up wherever there is convenience generally runs Ubuntu 20.04 LTS OS.

Supporting platform: All the coding works are done on Windows X64 platform with JetBrains tools including PyCharm Professional and IntelliJ IDEA, using Python 3.8.1 and JDK 19.

5.2.2 Establish Hacker's Infrastructure

The researchers enabled mobile data function and personal AP function of a smartphone simultaneously. Then used a wireless router, TP-Link AC1200, to extend its signal. After that, the researchers linked a Raspberry Pi 3 to the router and logged in the router to configure the DHCP server, which made its default local DNS server points to the malicious server, the Raspberry Pi 3, which runs a malicious DNS server and a malicious web server. A permanent DNS cache which used to spoof DNS was added to the DNS server, which made victims who access github.com directed to a fake website.

5.2.3 Establish VeriDNS Infrastructure

First of all, the researchers deployed the VeriDNS client on the victim's Windows 10 virtual machine. Then deployed the VeriDNS server on a Microsoft Azure Ubuntu virtual private server (VPS) and a VPS located in the School of Computing, National University of Singapore separately. The former acted as a root server while the latter acted as a branch server. Start *wireshark* and the processes and left them at the background.

5.2.4 Connect the Victim to Malicious AP

Connected the victim to the malicious AP. Type `ipconfig /all` in the command line prompt to check whether the default DNS has been set to the malicious DNS server.

5.2.5 Access the Poisoned Domain Name

Access *github.com*(Figure 8). Noticed that a forged DNS A RR was sent to the victim and the victim attempted to connected to the malicious web server. At the same time, VeriDNS client alerted the victim that a poisoned A RR was received.

5.2.6 Results

According to the experiments, thanks to the organism of system architecture and the method used in building the system, our VeriDNS server demonstrates good performance both sustainably in offering service and great robustness handling network exceptions. As popular domain public IP lists are deployed as supporting database, we both improves the hit rate of DNS verification query and decreases the time needed to resolve one particular domain name for not have to query online, with network traffic avoided. In our experiment of conducting 200 times query of VeriDNS request, we only experienced less than 3 times of false alarm, which in percentage is less than 1.5%, the same as accuracy of 98.5%. The results are shown in Figure 8.

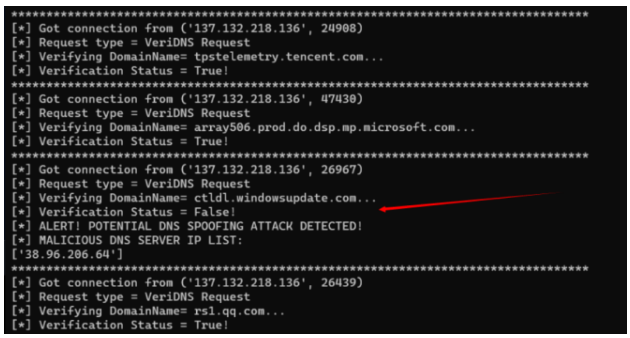


Figure 8. Part of detecting dns spoofing results.

6. Conclusion

To conclude, we use retransmitted packets for ARP spoofing detection and Polymerized Trusted DNS Verification with Cached

IP Range Tables for DNS spoofing detection respectively and we have carried out experimental verification on real computers and routers. The basic idea is to perform verification to get the authenticity of the packet and the experimental results prove that our method is effective.

For more information about our project, visit:

https://github.com/QuantumSecLab/MalAP-Detective

7. REFERENCES

[1] D. C. Plummer. An ethernet address resolution protocol. RFC 826, 1982.

[2] Danilo Bruschi, Alberto Ornaghi, Emilia Rosti , “S-ARP: a Secure Adderess Resolution Protocol” 19th Annual Computer Security Applications Conference, 2003, www.acsac.org/2003/papers/111.pdf

[3] Lawrence Berkeley National Laboratory , “ARPWATCH tool”: ARP Spoofing Detector. ftp://ftp.ee.lbl.gov/ARPwatch.tar.gz

[4] 王虎. 基于代理模式防御 DNS 欺骗攻击的研究与实现(硕士学位论文,北京邮电大学). https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201502&filename=1015585243.nh

[5] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, 1987.

[6] Meyer, Ulrike, and Susanne Wetzel. "A man-in-the-middle attack on UMTS." In Proceedings of the 3rd ACM workshop on Wireless security, pp. 90-97. ACM, 2004.

[7] Kish, Laszlo B. "Protection against the man-in-the-middle-attack for the Kirchhoff-loopJohnson (-like)-noise cipher and expansion by voltage-based security." Fluctuation and Noise Letters 6, no. 01 (2006): L57-L63.

[8] Hypponen, Konstantin, and Keijo MJ Haataja. "“Nino” man-in-the-middle attack on bluetooth secure simple pairing." In Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on, pp. 1-5. IEEE, 2007.

[9] Ouafi, Khaled, Raphael Overbeck, and Serge Vaudenay. "On the security of HB# against a man-in-the-middle attack." In International Conference on the Theory and Application of Cryptology and Information Security, pp. 108-124. Springer, Berlin, Heidelberg, 2008.

[10] YeautyYE, “arpspoof for macOS - intercept packets on a switched LAN”, https://github.com/YeautyYE/arpspoof