Intrusion Detection System

Network Security

9/30/2024

Ashkenazi Menashe, Klein Matan

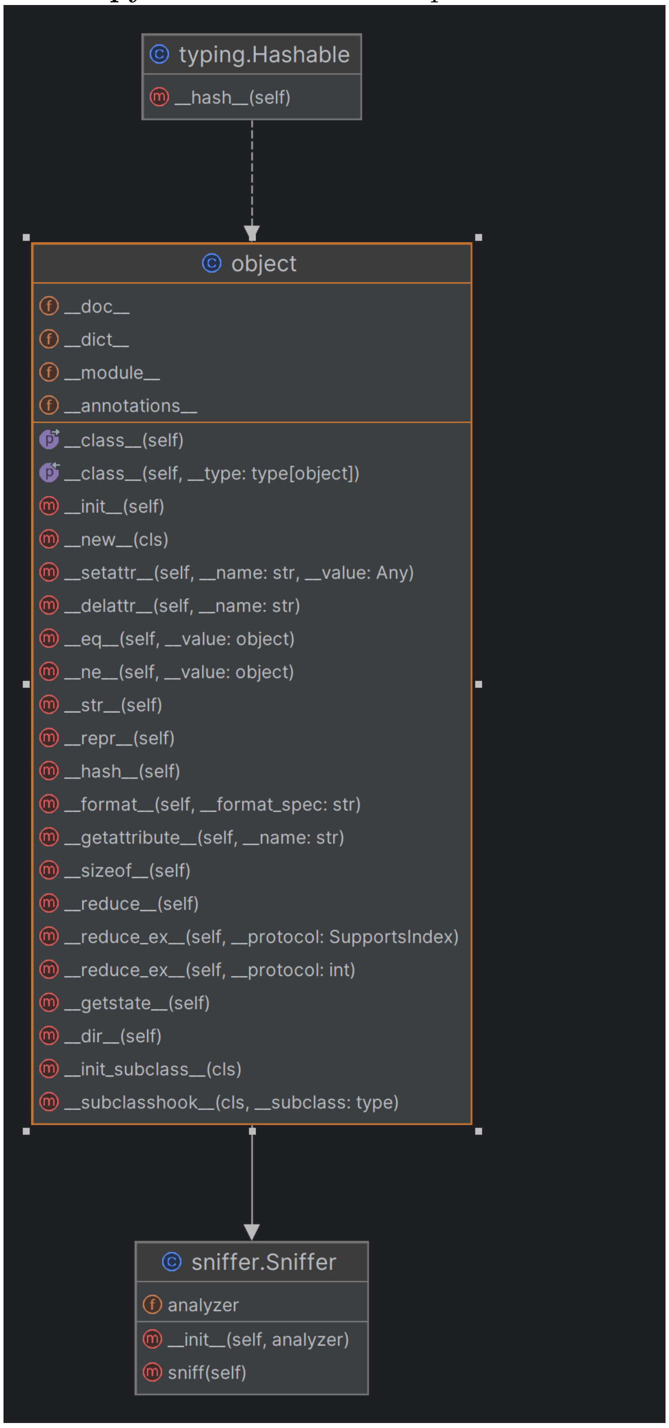
Contents

1 Objective

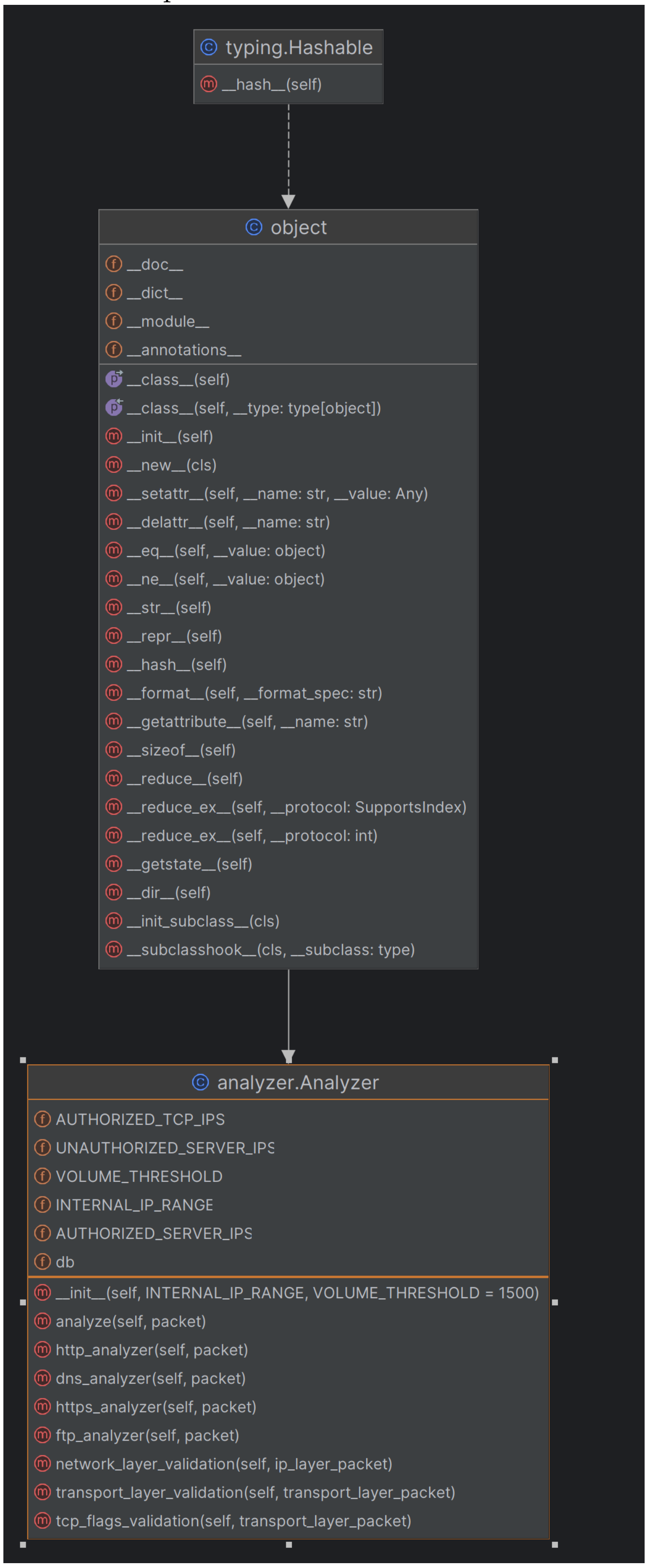
The objective was to build an IDS that specialized in data leak. The system gets traffic on interface, returns statistics and looks for attacks of data leak.

2 Architecture

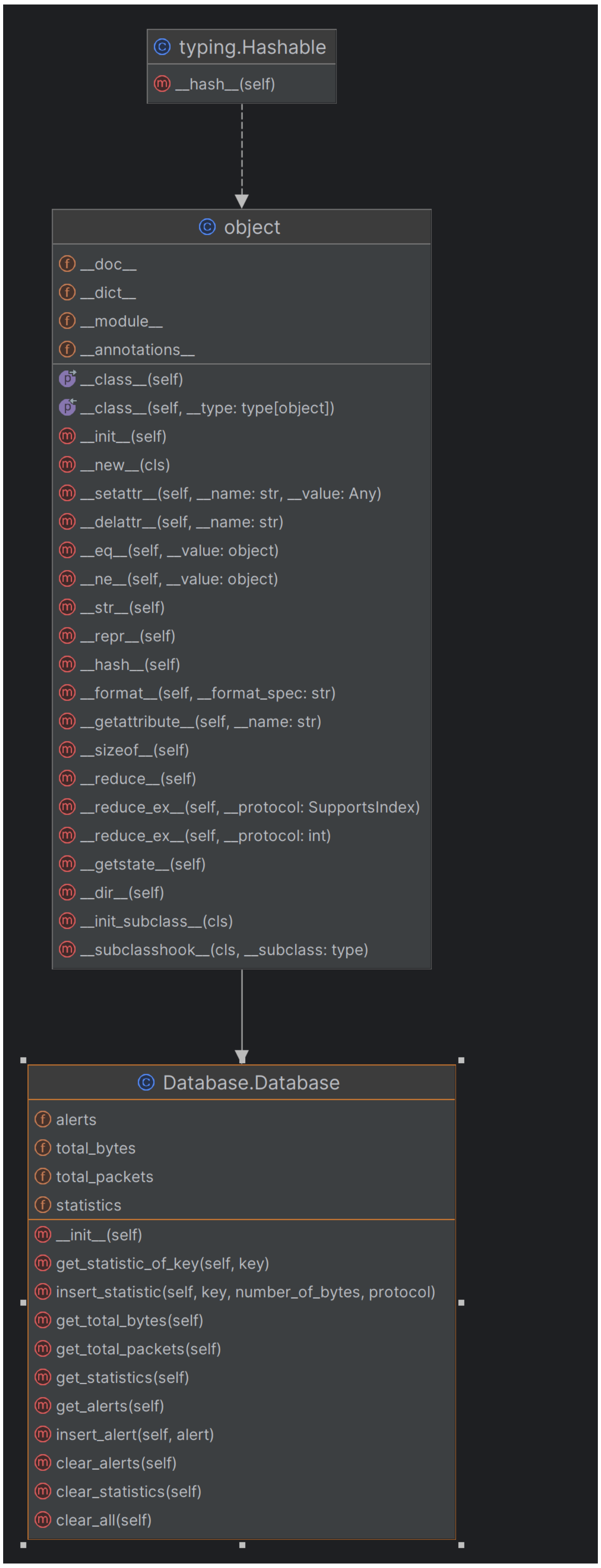
sniffer.py - This module is responsible for sniffing packets from the network.


typing.Hashable
__hash__(self)
object
__doc__
__dict__
__module__
__annotations__
__class__(self)
__class__(self, __type: type[object])
__init__(self)
__new__(cls)
__setattr__(self, __name: str, __value: Any)
__delattr__(self, __name: str)
__eq__(self, __value: object)
__ne__(self, __value: object)
__str__(self)
__repr__(self)
__hash__(self)
__format__(self, __format_spec: str)
__getattribute__(self, __name: str)
__sizeof__(self)
__reduce__(self)
__reduce_ex__(self, __protocol: SupportsIndex)
__reduce_ex__(self, __protocol: int)
__getstate__(self)
__dir__(self)
__init_subclass__(cls)
__subclasshook__(cls, __subclass: type)
sniffer.Sniffer
analyzer
__init__(self, analyzer)
sniff(self)

analyzer.py - This module is responsible for analyzing the packets and checking if they meet the requirements.

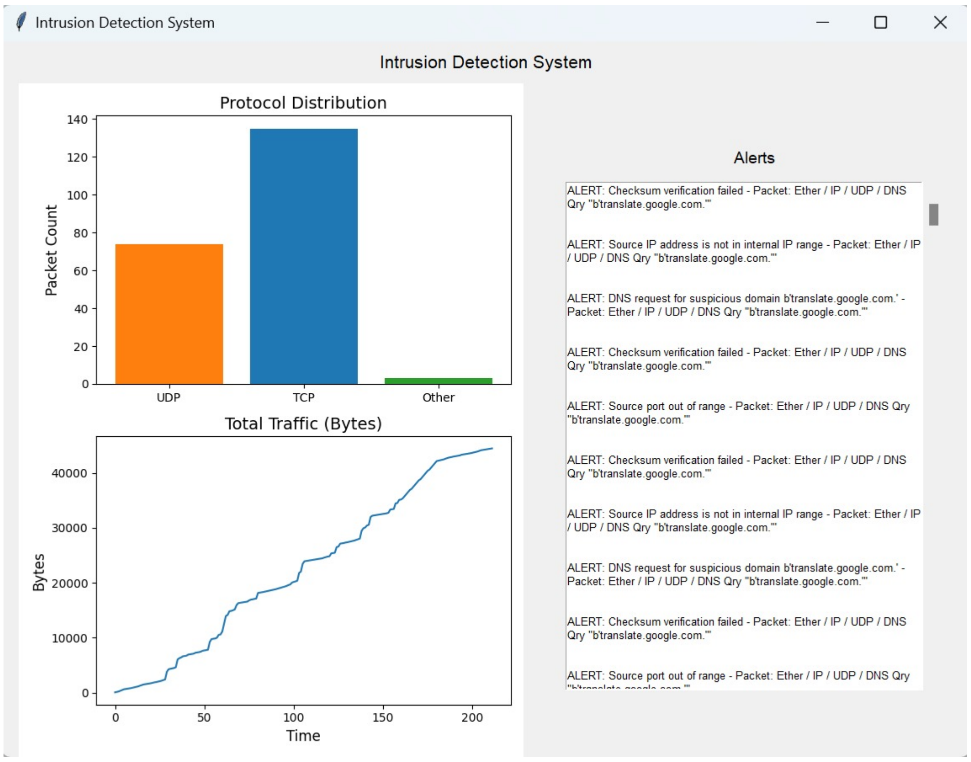

typing.Hashable
__hash__(self)
object
__doc__
__dict__
__module__
__annotations__
__class__(self)
__class__(self, __type: type[object])
__init__(self)
__new__(cls)
__setattr__(self, __name: str, __value: Any)
__delattr__(self, __name: str)
__eq__(self, __value: object)
__ne__(self, __value: object)
__str__(self)
__repr__(self)
__hash__(self)
__format__(self, __format_spec: str)
__getattribute__(self, __name: str)
__sizeof__(self)
__reduce__(self)
__reduce_ex__(self, __protocol: SupportsIndex)
__reduce_ex__(self, __protocol: int)
__getstate__(self)
__dir__(self)
__init_subclass__(cls)
__subclasshook__(cls, __subclass: type)
analyzer.Analyzer
AUTHORIZED_TCP_IPS
UNAUTHORIZED_SERVER_IPS
VOLUME_THRESHOLD
INTERNAL_IP_RANGE
AUTHORIZED_SERVER_IPS
db
__init__(self, INTERNAL_IP_RANGE, VOLUME_THRESHOLD = 1500)
analyze(self, packet)
http_analyzer(self, packet)
dns_analyzer(self, packet)
https_analyzer(self, packet)
ftp_analyzer(self, packet)
network_layer_validation(self, ip_layer_packet)
transport_layer_validation(self, transport_layer_packet)
tcp_flags_validation(self, transport_layer_packet)

Databse.py - This module is responsible for storing the statistics of the packets and the alerts.


typing.Hashable
__hash__(self)
object
__doc__
__dict__
__module__
__annotations__
__class__(self)
__class__(self, __type: type[object])
__init__(self)
__new__(cls)
__setattr__(self, __name: str, __value: Any)
__delattr__(self, __name: str)
__eq__(self, __value: object)
__ne__(self, __value: object)
__str__(self)
__repr__(self)
__hash__(self)
__format__(self, __format_spec: str)
__getattribute__(self, __name: str)
__sizeof__(self)
__reduce__(self)
__reduce_ex__(self, __protocol: SupportsIndex)
__reduce_ex__(self, __protocol: int)
__getstate__(self)
__dir__(self)
__init_subclass__(cls)
__subclasshook__(cls, __subclass: type)
Database.Database
alerts
total_bytes
total_packets
statistics
__init__(self)
get_statistic_of_key(self, key)
insert_statistic(self, key, number_of_bytes, protocol)
get_total_bytes(self)
get_total_packets(self)
get_statistics(self)
get_alerts(self)
insert_alert(self, alert)
clear_alerts(self)
clear_statistics(self)
clear_all(self)

main.py - This module is responsible for starting the network monitoring and detecting system.

GUI.py - This module is for the user interface of the network monitoring system. The GUI class provides a graphical user interface for displaying network statistics and alerts.



ALERT: Checksum verification failed - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Source IP address is not in internal IP range - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: DNS request for suspicious domain b'translate.google.com.' - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Checksum verification failed - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Source port out of range - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

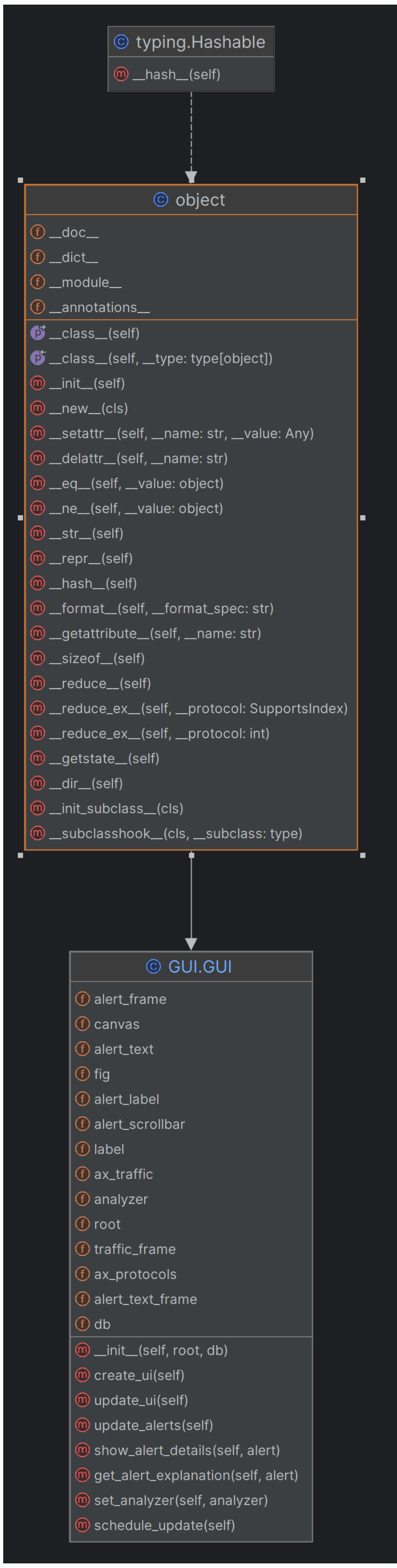
ALERT: Checksum verification failed - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Source IP address is not in internal IP range - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: DNS request for suspicious domain b'translate.google.com.' - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Checksum verification failed - Packet: Ether / IP / UDP / DNS Qry "b'translate.google.com.'"

ALERT: Source port out of range - Packet: Ether / IP / UDP / DNS Qry



3 Detection Approaches

In this section, we explore various techniques employed for the detection of network anomalies and potential data exfiltration activities. These approaches focus on multiple characteristics of network traffic such as packet length, URLs or domains, anomalies in headers and flags, HTTP and FTP-based information leaks, checksum validation, port analysis, and HTTP traffic behavior without associated DNS queries.

3.1 Packet Length and URL/Domain Analysis

One of the primary detection approaches involves monitoring the length of network packets, particularly in HTTP requests and responses. Abnormally long URLs or unusually large payload sizes within small packets may indicate attempts to exfiltrate data covertly. By analyzing the domain names or URLs accessed, especially those that deviate from typical patterns or use suspicious characters, the system can detect potential anomalies. For example, domains with excessively long names or those using unusual characters may suggest the use of DNS tunneling for data exfiltration.

3.2 Anomalies in Headers or Flags

Monitoring packet headers and flags is another critical aspect of anomaly detection. TCP/IP flags, such as SYN, ACK, or FIN, are expected to follow specific patterns during normal network communication. Unusual combinations, such as packets with both SYN and FIN flags set or missing expected flags, may indicate malicious activity. Additionally, anomalies in the structure or content of the headers, such as missing fields or incorrect values, can also point to data exfiltration attempts or abnormal network behavior.

3.3 HTTP GET Requests Leaking Data

HTTP GET requests that include sensitive information in the query parameters or URL may be a vector for data leakage. By monitoring for unusually long URLs in GET requests or the inclusion of personal or sensitive information such as usernames, passwords, or credit card details, the detection system can identify potential threats. This kind of leakage is often subtle and might occur through legitimate-looking requests, making it a key target for detection.

3.4 Sensitive Information in FTP Traffic

FTP is an older but still commonly used protocol for file transfers. The presence of sensitive information in FTP traffic, particularly during the login process or in file transfers, presents a significant risk for data exfiltration. Monitoring for cleartext passwords or sensitive files in FTP requests and responses can help detect such activities. Additionally, unusual FTP commands or anomalies in the file transfer process may suggest attempts to exfiltrate data.

3.5 Checksum Validation

Another important detection technique involves validating the checksum of packets. Each network packet contains a checksum, which is used to verify the integrity of the data. If the calculated checksum does not match the expected value, this may indicate packet tampering, potentially pointing to a malicious actor attempting to modify or inject data into the network stream. Routine validation of checksums helps ensure the authenticity of the traffic and prevents unauthorized modifications.

3.6 Port Range for Source and Destination

Network traffic is often categorized based on the source and destination ports being used. Most protocols utilize a specific range of ports, and deviations from these ranges can be a sign of anomalous behavior. For example, HTTP traffic typically occurs over port 80 or 443, while FTP uses ports 20 and 21. Traffic appearing on unusual ports, or ports reserved for other services, might indicate port scanning, tunneling, or other forms of network abuse.

3.7 HTTP Traffic Without DNS Queries

Finally, HTTP traffic that is not preceded by a corresponding DNS query is another indicator of potential malicious activity. In most legitimate scenarios, web requests are preceded by DNS lookups to resolve domain names into IP addresses. The absence of such DNS lookups may indicate that the HTTP request is being directed to an IP address directly, possibly pointing to a malicious script or a data exfiltration attempt bypassing normal name resolution. In conclusion, these detection approaches provide a robust framework for identifying various forms of data exfiltration and network anomalies. By analyzing network characteristics such as packet lengths, header anomalies, sensitive information leakage, port usage, and checksum validation, the detection system can safeguard against a wide range of threats.

4 Functions descriptions

4.1 Analyzer Class

The `Analyzer` class is responsible for inspecting network packets and checking them against various security policies. It contains methods to analyze HTTP, HTTPS, DNS, FTP packets, as well as validations at the network and transport layers. The class also maintains sets of authorized IPs and uses a database object to log alerts and statistics.

The method `analyze` is used to parser the packets for 5-tuple flow identifiers and analyze the packets based on the network layer, transport layer, DNS, HTTP, HTTPS, and FTP.

The method `network_layer_validation` is used to analyze the packets in the network layer and check various things like the checksum of the packet, the source IP address should be in internal IP range, and the destination IP address should not be unauthorized.

The method `transport_layer_validation` is used to analyze the packets in the transport layer and check various requirements like checksum verification, source and destination port within range, establishing and closing connections, and flags set in the packet.

The method `tcp_flags_validation` is used to analyze the flags set in the TCP packets and check that anomalies are not present.

The method `dns_analyzer` is used to analyze the DNS packets and check if they meet the requirements like DNS request with long domain names, non-standard ports, not well known domain names, and anomalies in the DNS headers.

The method `http_analyzer` is used to analyze the HTTP packets and check if they meet the requirements like HTTP request with long URLs, missing headers, sensitive information in the raw data and missing DNS requests before sending an HTTP request.

The method `https_analyzer` is used to analyze the HTTPS packets and check if they meet the requirements like not sending a DNS request before sending an HTTPS request and unauthorized ports in the destination.

The method `ftp_analyzer` is used to analyze the FTP packets and check if sensitive information is present in the raw data.

4.2 Sniffer Class

The `sniff` method is used to sniff packets from the network with the help of the Scapy library and analyze them using the analyzer object.