

## 1. Begriffe

**Port** Der Port ist Teil der Adressierung und definiert einen eindeutigen Zugangspunkt zum jeweiligen Teilnehmer bzw. einen Prozess. So ist z.B. Port 80 für Webserver reserviert und eine Anfrage an diesen Port kann dann vom korrekten Dienst bearbeitet werden.

**Cipher** Cipher ist eine Möglichkeit der kryptographischen Algorithmen, und zwar beschreibt er die Methode einen unverschlüsselten Reintext so umzuwandeln, dass der Inhalt verborgen bleibt. Oft wird Cipher allerdings auch als Synonym zu dem bereits verschlüsselten Text benutzt.

**Public Key Infrastructure** Die PKI ist ein System, welches den Austausch und das Überprüfen von öffentlichen Schlüsseln eines asymmetrischen Verschlüsselungsverfahrens ermöglicht.

**Bastion Host** Ist ein Teilnehmer in einem Netzwerk, der anfälliger für Attacken ist als andere Teilnehmer, z.B. weil auf dem Bastion Host die Webserverdienste laufen. Daher muss er mehr geschützt werden.

**Proxy** Ein Proxy leitet auf Anfrage eines Clients eine nach außen gerichtete Netzwerkanfrage weiter und schickt sie nach dem Bearbeiten wieder an den Client zurück.

**NAT** Steht für Network Address Translation und beschreibt den Vorgang, dass ein Router die Netzwerkadresse in IP-Paketen verändert, um die internen Adressen geheim zu halten.

**IPsec** Mit IPsec werden IP Pakete geschützt. Dies kann auf unterschiedliche Arten passieren, zum einen kann das ganze ursprüngliche Packet eingehüllt werden und somit versteckt, es kann aber auch nur ein zusätzlicher Headerteil eingefügt werden, der nur die Integrität sicherstellt.

## 2. Zwischensysteme

a) Es gibt die Adressklassen A-E im Internet, wobei hier in der Aufgabe nur die Klassen A und B verwendet werden. A umfasst dabei den Adressraum von 1.0.0.0 bis 127.255.255.255 und Klasse B alles zwischen 128.0.0.0 und 191.255.255.255.

Für das private Netz sind speziell Adressbereiche reserviert, und zwar in Klasse A der Bereich von 10.0.0.0 bis 10.255.255.255, in Klasse B alles zwischen 172.16.0.0 und 172.31.255.255 und in Klasse C zwischen 192.168.0.0 und 192.168.255.255.

b) Die ersten Pakete, die über Teilstrecken *a*, *b* und *e* laufen sind ARP-Pakete. Und zwar kennt keine Station IP und MAC Adresse von den anderen bzw. Nachbarn, so dass als erstes ARP Tabellen angelegt werden müssen, um das Netzwerk zu erschließen. Nach dem dann bekannt ist, welche MAC-Adresse zur gefragten IP zuzuordnen ist, kann dann von C1 die DNS Anfrage an C3 stellen (durch ARP wird wieder die MAC rausgefunden) und dann schlussendlich (wenn dann auch bekannt ist welche MAC-Adresse C4 hat) auch die eigentliche

HTTP Anfrage.

Weil ich mir nicht vorstellen kann, dass das alles das gewünschte ist, hier der genaue Ablauf der Anfrage, die mit dem Kreuzchen sind die, die auf a,b oder e liegen.

	Von	Nach	Was?
x	C1	S1	ARP 129.13.37.67 (wegen MAC für DNS Auflösung)
x	S1	C2, R1	ARP 129.13.37.67
	R1	S2, C3	ARP 129.13.37.67
	C3	R1	ARP-Reply 4E37
x	R1	S1	ARP-Reply 3A24
x	S1	C1	ARP-Reply 3A24
	(S2	R2	ARP 129.13.37.67)
x	(R2	C4	ARP 129.13.37.67)
x	C1	S1	DNS Auflösung URL C4 (an C3)
x	S1	R1	DNS Auflösung URL C4
	R1	C3	DNS Auflösung URL C4
	C3	R1	DNS-Reply 129.13.128.200
x	R1	S1	DNS-Reply 129.13.128.200
x	S1	C1	DNS-Reply 129.13.128.200
x	C1	S1	ARP 129.13.128.200 (wegen MAC für HTTP-Anfrage)
	S1	C2, R1	ARP 129.13.128.200
	R1	C3, S2	ARP 129.13.128.200
	S2	R2	ARP 129.13.128.200
x	R2	C4	ARP 129.13.128.200
x	C4	R2	ARP-Reply 345C
	R2	S2	ARP-Reply AE34
	S2	R1	ARP-Reply AE34
x	R1	S1	ARP-Reply 3A24
x	S1	C1	ARP-Reply 3A24
x	C1	S1	HTTP GET 129.13.128.200/3A24
x	S1	R1	HTTP GET 129.13.128.200/3A24
	R1	S2	HTTP GET 129.13.128.200/AE34
	S2	R2	HTTP GET 129.13.128.200/AE34
x	R2	C4	HTTP GET 129.13.128.200/345C
x	C4	R2	HTTP OK 129.13.42.67/B2A5
	R2	S2	HTTP OK 129.13.42.67/5D67
x	S2	R1	HTTP OK 129.13.42.67/5D67
x	R1	S1	HTTP OK 129.13.42.67/AE3F
x	S1	C1	HTTP OK 129.13.42.67/AE3F

c) Es wird davon ausgegangen, dass C2 schon von früheren Anfragen die MAC-Adresse von C3 kennt (bzw. die von R1).

Von	Nach	Quelle/Ziel
C2	S1	1F3D/3A24
S1	C1, R1	1F3D/3A24
R1	C3	7F29/4E37
C3	R1	4E37/7F29
R1	S1	3A24/1F39
S1	C2	3A24/1F39

### 3. Sicherheitsziele

**Vertraulichkeit** Durch Maskieren kann sich eine Einheit als eine andere ausgeben und somit Zugang zu Daten erlangen, des sie normalerweise nicht haben dürfte.

**Erhalt der Datenintegrität** Durch Einschleusen von neuen Daten, wobei man sie jemanden anderes zuschreibt, kann die Datenintegrität angegriffen werden. Diese manipulierten Daten sind dann nicht von echten Daten von dem Sender zu unterscheiden.

**Verantwortlichkeit** Wenn man die Zugangsdaten von jemand anderes benutzt, dann kann man in dem Namen einer anderen Person handeln, ohne das man selbst für diese Aktivitäten identifiziert wird.

**Serviceerreichbarkeit** Durch Sabotage, wie z.B. DoS-Attacken kann die Serviceerreichbarkeit eingeschränkt werden.

### 4. Fifty/Fifty

- *Eine Denial-of-Service-Attacke ist schon dann gegeben, wenn absichtlich so viele erlaubte Anfragen an einen Server gestellt werden, dass die Anfragen anderer Benutzer erst nach langer Zeit ausgeführt werden können.*  
Jein, was heißt schon dann gegeben. Das ist ja quasi die Definition einer DoS-Attacke, dass man bewusst den Host versucht zu überlasten um spezielle Dienste arbeitsunfähig werden.
- *Mit dem Aufkommen von asymmetrischen Codes sind symmetrische Codes überflüssig geworden, da diese einen vorherigen Schlüsselaustausch benötigen.*  
Falsch. Auch bei asymmetrischen Codes muss ja irgendwie der öffentliche Schlüssel ausgetauscht werden. Zusätzlich ist das asymmetrische Verfahren viel zeitintensiver, so dass man dazu greift, einen symmetrischen Schlüssel asymmetrisch verschlüsselt auszutauschen.
- *UDP wird für Multimedia-Dienste mit Echtzeit-Anforderungen eingesetzt.*  
Richtig. Da UDP verbindungslos und nicht-zuverlässig ist, eignet es sich hervorragend um für Multimediaanwendungen, bei denen es auf schnelle Übertragung ankommt, einzusetzen.

## 5. Port-Scanner

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <netdb.h>
5 #include <stdio.h>
6 #include <sys/time.h>
7
8 #define BUFFER 1024
9
10 int portScannen(int port, char* host) {
11     int s = socket(AF_INET, SOCK_STREAM, 0);
12     struct hostent *ip;
13     struct sockaddr_in server;
14     struct timeval time;
15     char buffer[BUFFER];
16
17     time.tv_sec = 1;
18     time.tv_usec = 0;
19
20     ip = gethostbyname(host);
21
22     setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, &time, sizeof(time));
23     // setzt receive Timeout.
24
25     server.sin_addr = *(struct in_addr*)ip->h_addr;
26     server.sin_port = htons(port);
27     server.sin_family = AF_INET;
28
29     if (connect(s, &server, sizeof(server)) == -1) {
30         return -1;
31     }
32     recv(s, buffer, sizeof(buffer)-1, 0);
33     printf("Port: %d sagt %s\n", port, buffer);
34     close(s);
35     return 0;
36 }
37
38 int main(int argc, char *argv[]) {
39     int port1, port2;
40     if (argc < 4) {
41         perror("Zu wenig Parameter");
42         return -1;
43     }
44 }
```

```
43     if((port1 = atoi(argv[2])) != 0 && (port2 = atoi(argv[3]))  
        != 0) {  
44         while(port1 <= port2) {  
45             portScannen(port1, argv[1]);  
46             port1++;  
47         }  
48     }  
49     return 0;  
50 }
```

## 6. IP-Stack

a)

b)

c)