

SVS Bachelor-Projekt Network Security

Blatt 4: Sniffing und Scanning

Louis Kobras
6658699

Utz Pöhlmann
6663579

1 Vertrautmachen mit der Umgebung

1.1

- VMs in unser Verzeichnis kopiert und geöffnet
- Angegeben, dass VMs kopiert wurden
- Bootreihenfolge: MysteryVM, SurfingVM, RoutingVM
- RoutingVM hat 2 Netzwerkadapter: NAT und Host-Only

1.2

- SurfingVM hat keine IP auf eth1
- Zurücksetzen der Datei `/etc/udev/rules.d/70-persistent-net.rules` mit root-Rechten
- Rebooten der SurfingVM
- **Standartgateway: 192.168.254.1**
- IP: 192.168.254.44
- **DNS-Nameserver: 10.1.1.1** (ermittelt mit `route -n`, bestätigt mit `nslookup ubuntu.com`)

1.3

- Netzwerkkarte 1: eth0, 172.16.137.222
- Netzwerkkarte 2: eth1, 192.168.254.1
- VMWare-Standart-Gateway: 172.16.137.2

1.4

- Ping an 10.1.1.2 aus beiden VMs erfolgreich (0% Package loss)

2 Sniffing mit tcpdump

2.1

- `tcpdump` listet alle Pakete auf, die über die Netzwerkkarte laufen
- Capture-Filter zum Filtern und Sortieren der gefangenen Packages

2.2

- Kommando: `sudo tcpdump -p -i eth1 -s 0 -vvv udp port 53 > log1 ([1], [2])`

¹-p: weil Aufgabe. -i ethX: Adapter, der gelistened werden woll. -s 0: Größe des Capture in Bytes (0=alle). -vvv: alle Paketinformationen ausgeben. «schnittstelle» port «port». > log: in die Datei 'log' echoen, die ggf. im \$(pwd) angelegt wird.

2.2.1 Anfrage

Output:

```
1 14:01:53:677232 IP (tos 0x0, ttl 64, id 1258, offset 0, flags [DF], proto
  UDP (17), length 60)
2 192.168.254.44.35616 > server.svslab.domain: [udp sum ok] 19679+ A? www
  .google.com. (32)
```

Aufbau ([3]):

```
1 timestamp protocol (package-information)
2 nameserver > local-domain checksum-check some-number Question? target.
  (num)
```

Anmerkung: tcpdump kennt nur wenige Protokolle und gibt, wenn er ein Protokoll nicht erkennt, IP an.

2.2.2 Antwort

Output:

```
1 14:01:53.677765 IP (tos 0x0, ttl 127, id 21488, offset 0, flags [none],
  proto UDP (17), length 212)
2 server.svslab.domain > 192.168.254.44.35616: [udp sum ok] 19679 q: A?
  www.google.com. 1/4/4 www.google.com. [2m33s] A 216.58.213.228 ns:
  google.com. [1d21h4m48s] NS ns1.google.com., google.com. [1
  d21h4m48s] NS ns3.google.com., google.com. [1d21h4m48s] NS ns2.
  google.com., google.com. [1d21h4m48s] NS ns4.google.com. ar: ns1.
  google.com. [3d21h12m22s] A 216.239.32.10, ns2.google.com. [3
  d21h12m22s] A 216.239.34.10, ns3.google.com. [3d21h12m22s] A
  216.239.36.10, ns4.google.com. [3d21h12m44s] A 216.239.38.10 (184)
```

Wieder ist die erste Zeile Meta-Information. Die zweite Zeile ist eine Anfrage unserer Domain an unseren Nameserver, welcher dann an Google weiterfragt, wo die Nameserver von Google die Anfrage durch reichen.

2.3

- Kommando: `sudo tcpdump -p -i eth1 -s 0 -vvv '(tcp port 80) or (tcp port 443)' > log2` ([1])

Output:

```
1 14:27:10.394893 IP (tos 0x0, ttl 64, id 18592, offset 0, flags [DF], proto
  TCP (6), length 60)
2 192.168.254.44.35453 > ham04s01-in-f4.1e100.net.www: tcp 0
```

Fazit: Unser Nameserver brennt mit einem Hamster durch.

2.4

Neuer Befehl: `sudo tcpdump -p -i eth1 -s 0 -vvv -A 'tcp port 80'` Output vgl. Anhang 2

2.5

- Aufrufen der URL `http://10.1.1.2/verysecure/`
- Eingabe der Login-Daten `alice:sehrgeheim`
- Login-Daten im Package:

²s.o., tcp port 80 für HTTP, tcp port 443 für HTTPS

```
1 Authorization: Basic YWxpY2U6c2V0cmdlaGVpbQ==
```

⇒ Base-64-verschlüsselt.

- Entschlüsselung ergibt: `alice:sehrgeheim`

3 Sniffing mit dsniff und urlsnarf

3.1 urlsnarf

Befehl:

```
1 sudo urlsnarf -i eth1 > log
```

Aufbau des Output: IP - Timestamp - Adresse - Protokoll - Browser - Systemdaten
Befehl greift alle HTTP-Pakete vom angegebenen Adapter ab und zeigt ihre Daten an.

3.2 dsniff

Befehl:

```
1 sudo dsniff -i eth1 > log
```

Aufbau des Output: Timestamp - Senderadresse - Empfängeradresse - Adresse - Protokoll - Host - Paketinhalt (decoded)
Liest den Inhalt von HTTP-Paketen aus und decodiert (zumindest Base-64).

4 Sniffing mit Wireshark

4.1

Wireshark liefert eine graphische Darstellung der gesniffen Pakete in lesbarer Tabellenform und zeigt den Inhalt der Pakete an.

4.2

Display-Filter. Bestimmt, welche der aufgefangenen Pakete angezeigt werden.

Capture-Filter. Bestimmt, welche Pakete aufgefangen werden.

4.3

Wireshark wurde geöffnet.

4.4

- `eth1` liegt nahe, da dieses Interface das Gateway für die SurfingVM bereitstellt (Capture-Filter).
- Alternativ zur Interface-Wahl kann ein Display-Filter zur Steuerung des Outputs erstellt werden.

4.5

Es wird nur ein Ping gesendet. (#Easteregg). Der Server pingt zurück. Die Pings werden über ICMP³ übertragen.

Der Klient DARF die Daten so lange behalten, wie er will. Jedes Paket hat einen time-to-live-Eintrag; ist dieser überschritten, wird das Paket erneut angefordert.

Weil Linux den DNS nicht cached, erwarten wir die gleiche Antwort.

Wir bekommen die gleiche Antwort, was bedeutet, Linux cached den DNS nicht.

Der Browser sendet Pings über TCP und anschließend HTTP. Dies wechselt sich stetig ab.

Es würde erwartet, dass in beiden Fällen das Gleiche passiert

4.6

Erstellen des Filters:

1. Kontextmenü eines HTTP-Eintrages
2. Klick auf "Apply as filter"
3. Fertig

4.7

Funktion liegt unter Menüreiter "Analyze".

Ausgabe eines HTTP-Response öffnet Popup, in welchem der Content des Package angezeigt wird. Es kann zwischen verschiedenen Darstellungen gewählt werden (Raw/ASCII, HexDump, C Arrays)

4.8

- Server starten auf RoutingVM
- Auf SurfingVM mit telnet auf Server einwählen
- Auf RoutingVM Wireshark starten
- Auf SurfingVM Dinge tun
- Auf der RoutingVM kann der gesamte Chat nun als TCP-Packages ausgelesen werden (u.a. auch die Login Daten)

4.9

- Aufrufen von `https://de-de.facebook.com`
- Verwendete Protokolle: TCP, TLSv1
- Es wurden nicht alle Pakete in Wireshark angezeigt (Nummerierung nicht durchgehend). Kein Filter eingestellt. Theorie: HTTPS wird verborgen.

5 ARP-Spoofing

5.1

Ablauf des ARP-Spoofings:

- 'Angreifer' sendet gefälschtes ARP-Paket mit seiner eigenen MAC-Adresse an Entity A, sodass Entity A ihre Pakete an den 'Angreifer' schickt statt an Entity B.

³Internet Control Message Protocol

- Gleiches geschieht mit Entity B in Bezug auf Entity A.
- 'Angreifer' liest die Pakete von Entity A aus und leitet sie an Entity B weiter, und umgekehrt.

Funktionsweise von `arpspoof`:

- Abzufangender Adapter wird angegeben
- Entity, die gespoof werden soll
- Domain, deren eingehender Datenstream mitgelesen werden soll

5.2

Befehl: `sudo arpspoof 172.16.137.2`. Es wird eine lange Reihe identischer arp-Replys ausgegeben.

5.3

Es wurde der Wireshark-Adapter "any" ausgewählt.

5.4

- Display Filter auf ICMP
- IP-Adresse 172.16.137.146 durch Ping-Intervall ermittelt

5.5

- Display-Filter `ip.addr==172.16.137.146 && pop`
- beliebigen Eintrag ausgewählt und per Rechtsklick "Follow TCP Stream"
- Nutzerdaten: `USER bumblebee, PASS Optimus Prime`
- hat eine ungelesene Mail von `root@labservervm`
- Alternativen: Einhalten von Verdecktheit und Verborgtheit (GSS Sicherheitsziele :P)

5.6

- Browser/Version: Mozilla/5.0
- URL: `http://10.1.1.2/secure/secret.html`
- Login-Daten: Base-64 encoded im Kopf des Paketes; Daten: `admin:geheim`

Widersprüche zwischen Erkenntnissen:

- None

5.7

`arpspoof` wurde am Ende des Experiments mit Ctrl-C abgebrochen.

6 Scanning mit nmap

6.1

Die 5 coolsten NMAP-Funktionen (nach [4])

1. Security audits
2. network inventory
3. managing service upgrade schedules
4. monitoring host uptime
5. monitoring service uptime

6.2

- Skript vgl. Anhang 4
- gewählte Sprache: Bash
- Liste aktiver Rechner:

- 10.1.1.1
- 10.1.1.2
- 10.1.1.5
- 10.1.1.11
- 10.1.1.21
- 10.1.1.31
- 10.1.1.41
- 10.1.1.51
- 10.1.1.61
- 10.1.1.71
- 10.1.1.81
- 10.1.1.91
- 10.1.1.101
- 10.1.1.111
- 10.1.1.121
- 10.1.1.131
- 10.1.1.181
- 10.1.1.186
- 10.1.1.218
- 10.1.1.222
- 10.1.1.235
- 10.1.1.238
- 10.1.1.254

6.3

- `nmap` sagt, alle sind up.
- `ping` sagt, fast alle sind down.
- `nmap` lügt. >.<
- `nmap` erzeugt:
 - Ping (ICMP)
 - Senden eines HTTPS-Package (TCP)
 - Senden eines HTTP-Package (TCP)
 - Timestamp anfragen (ICMP)
 - Antwort auf HTTP von Remote Host (TCP)
- `nmap` erzeugt bei einem Host, den `ping` gefunden hat:
 - Ping (ICMP)
 - Ping Response vom Remote Host (ICMP)
 - Senden eines HTTPS-Package (TCP)
 - Senden eines HTTP-Package (TCP)
 - Timestamp anfragen (ICMP)
 - Antwort auf HTTP von Remote Host (TCP)
- Ermittlung des Up-Status durch Erhalt der HTTP-Antwort

6.4

- Three-Way-Handshake: SequenceNumber(SYN) (x) von Client and Host, Rücksenden von SequenceNumber (y) und AcknowledgeNumber(ACK) (x+1) von Host an Client, Rücksenden von AcknowledgeNumber (y+1) Client an Host. ([5])
- TCP-Connect-Scan durch `sudo nmap -sT 10.1.1.2` (vollständiger 3-way-handshake, (SYN)->(SYN+ACK)->(ACK))
- TCP-SYN-Scan durch `sudo nmap -sS 10.1.1.2` (nur halber handshake, (SYN)->(SYN+ACK))

6.5

- Scannen aller Ports mit `sudo nmap -p- -sV 172.16.137.146 -oG logs.txt`
- Output enthält `5288/open/tcp//http//Apache httpd 2.2.14 ((Ubuntu))`
- Apache-Webserver im Browser aufgerufen mit `172.16.137.146:5288` (Secret Site)

7 OpenVAS

7.1

Wir haben uns ein wenig mit OpenVAS vertraut gemacht.

7.2

Start des OpenVAS-Servers mit `/etc/init.d/openvas-server start`
FEHLGESCHLAGEN, weil der VAS-Daemon Dinge nicht laden konnte.

7.3

Startet

7.4

Login auf dem Server als user@localhost:user

7.5

Es wurde auf das Fragezeichen geklickt und der Assistent durchgearbeitet. Währenddessen wurde als Name "localhost" und als IP-Adresse die eigene IP-Adresse gewählt. Danach wurde auf das Stecker-Symbol geklickt, die Daten eingegeben und "ok" betätigt.

Es wurden keine Schwachstellen gefunden (exportierter Report enthält "Number of security holes: 0; Number of security warnings: 0")

Note: OpenVAS-Scan-Server ist veraltet.

Eingabe: File -> Scan Assistant -> Task: \$name -> Scope: \$name -> Targets: IP der MysteryVM (172.16.137.146) -> Execute

7.6

MysteryVM hat eine Sicherheitslücke und zwei Sicherheitswarnungen.

Login-Daten: root:password

7.7

Anhang 1:

Anhang 2

Output des HTTP-Sniffing

```
1 14:37:51.282324 IP (tos 0x0, ttl 64, id 51836, offset 0, flags [DF], proto
   TCP (6), length 487)
2 192.168.254.44.35465 > ham04s01-in-f4.1e100.net.www: Flags [P.], cksum
   0xbb75 (correct), seq 311797790:311798237, ack 398350995, win 9648,
   length 447
3 E....|@.@.....,:.....P.....Z.P.%..u..GET / HTTP/1.1
4 Host: www.google.com
5 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:10.0.1) Gecko/20100101
   Firefox/10.0.1
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
7 Accept-Language: en-us,en;q=0.5
8 Accept-Encoding: gzip, deflate
9 Connection: keep-alive
10 Cookie: NID=79=
   WlzebisuVRgORNA05jSpuedXCNNs1eBM8yEMd8n30_OluRdkzWbkChEEQ4YgUvHTWB3a64hs
   LjaseRkBrUN1vGIU56_9YOWlq0yWpZRTS4cdFs9-0wKsmJyANZ1uZ7UPnFbMMSPb
```

Anhang 3: Outputs von urlsnarf und dsniff

```
1 urlsnarf
2
3 192.168.254.44 - - [26/May/2016:15:03:07 +0200] "GET http://10.1.1.2/
   verysecure/ HTTP/1.1" - - "-" "Mozilla/5.0 (X11; Ubuntu; Linux i686; rv
   :10.0.1) Gecko/20100101 Firefox/10.0.1"
4
5 dsniff
6
7 dsniff: listening on eth1
8 -----
9 05/26/16 15:06:17 tcp 192.168.254.44.56594 -> labservervm.svslab.80 (http)
10 GET /verysecure/ HTTP/1.1
11 Host: 10.1.1.2
12 Authorization: Basic YWxpY2U6c2VocmdlaGVpbQ== [alice:sehrgeheim]
```

Anhang 4: Chat und so Stuff dem seine Dateien

Anhang 5: Ping-Skript (Aufgabe 6.2)

```
1 #!/bin/bash
2 COUNTER=0
3 LIMIT=255
4 while [ $COUNTER -lt $LIMIT ]; do
5     echo "pinging 10.1.1.1.$COUNTER"
6     ping -c1 10.1.1.1.$COUNTER
7     let COUNTER=COUNTER+1
8 done
```

Literatur

- [1] <https://wiki.ubuntuusers.de/tcpdump/>
- [2] <http://danielmessler.com/study/tcpdump/>
- [3] www.alexonlinux.com/tcpdump-for-dummies#...
- [4] <https://nmap.org/book/man.html>
- [5] <https://de.wikipedia.org/wiki/Drei-Wege-Handschlag#/media/File:Three-way-handshake-example.gif>