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 > log`[1] ([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:

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

Aufbau ([3]):

```
timestamp protocoll (package-information)
    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:

```
14:01:53.677765 IP (tos 0x0, ttl 127, id 21488, offset 0, flags [none],
    proto UDP (17), length 212)
    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)' > log`[2] ([1])

Output:

```
14:27:10.394893 IP (tos 0x0, ttl 64, id 18592, offset 0, flags [DF], proto
    TCP (6), length 60)
    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:

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

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

⇒ Base-64-verschlüsselt.

- Entschlüsselung ergibt: `alice:sehrgeheim`

3 Sniffing mit dsniff und urlsnarf

3.1 urlsnarf

Befehl:

```
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:

```
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 gesnifften Pakete in lesbarer Tabellenform und zeigt den Inhalt der Pakete an.

4.2

Display-Filter.

Capture-Filter.

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 ICMP[3] 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

.

4.9

.

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.
- 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 gespooft werden soll
- Domain, deren eingehender Datenstream mitgelesen werden soll

[3] Internet Control Message Protocol

5.2

Befehl: `sudo arpspoof 172.16.137.222`. Es wird eine lange Reihe identischer arp-Replys ausgegeben.

5.3

Es wurde der Wireshark-Adapter “any” ausgewählt.

5.4

.

5.5

.

5.6

.

5.7

.

6 Scanning mit nmap

6.1

.

6.2

.

6.3

.

6.4

.

6.5

.

7 OpenVAS

7.1

.

7.2

.

7.3

.

7.4

.

7.5

.

7.6

.

7.7

.

Anhang 1:

Anhang 2

Output des HTTP-Sniffing

```
14:37:51.282324 IP (tos 0x0, ttl 64, id 51836, offset 0, flags [DF], proto TCP (6), length 487)
    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
E....|@.@......,.:.....P......Z.P.%..u..GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:10.0.1) Gecko/20100101 Firefox/10.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: NID=79=
    WlzebisuVRgORNA05jSpuedXCNNs1eBM8yEMd8n30_OluRdkzWbkChEEQ4YgUvHTWB3a64hs
    LjaseRkBrUN1vGIU56_9YOWlq0yWpZRTS4cdFs9-0wKsmJyANZ1uZ7UPnFbMMSPb
```

Anhang 3: Outputs von urlsnarf und dsniff

```
urlsnarf

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"

dsniff

dsniff: listening on eth1
-----------------
05/26/16 15:06:17 tcp 192.168.254.44.56594 -> labservervm.svslab.80 (http)
GET /verysecure/ HTTP/1.1
Host: 10.1.1.2
Authorization: Basic YWxpY2U6c2VocmdlaGVpbQ== [alice:sehrgeheim]
```

Literatur

[1] https://wiki.ubuntuusers.de/tcpdump/

[2] http://danielmessler.com/study/tcpdump/

[3] www.alexonlinux.com/tcpdump-for-dummies#...