Grundpraktikum Netz- und Datensicherheit

Thema:

Entwickeln eines automatisierten Schwachstellenscanners

Lehrstuhl fi¿½r Netz- und Datensicherheit
Ruhr-Universitï¿½t Bochum

Versuchdurchfi¿½hrung: Raum ID 2/168

Betreuung: Florian Feldmann
Zusammengestellt von: Martin Grothe
Stand: 13. Dezember 2017
Version: 0.5

Inhaltsverzeichnis

1 Einführung (1h)

Für die Erstellung eines automatisierten Schwachstellenscanners, müssen verschiedene Programme zusammen arbeiten.

- Portscanner
- Packetsniffer
- Analyzer

Der Portscanner übernimmt dabei die Aufgabe des aufspürens offener Ports. Parallel untersucht der Paketsniffer die Pakete auf Informationen, die auf Schwachstellen hindeuten. Anschliessend kann eine so gefundene Schwachstelle ausgenutzt werden.
Dieser Versuch baut auf vielen Informationen auf, die in den letzten Semestern des Studiums vermittelt worden sind.

1. OSI Schichten System oder einem vergleichbaren Modell
2. C Programmierung insbesondere Strukturen und Pointer

Sollten Sie in einem oder allen der oben genannten Themen unsicher sein, so lesen Sie sich in das/die Themengebiet(e) wieder ein und fahren Sie danach mit den Kontrollfragen fort.

1.1 Kontrollfragen der Einführung

Die Kontrollfragen/aufgaben sollen sicherstellen, dass Sie das nötige Hintergrundwissen haben, um die Grundlagen für den Versuch zu verstehen.

OSI Schichten

1. Beschreiben Sie den Aufbau eines Ethernet Pakets.
2. Wie gross kann ein Ethernet Paket maximal werden?
3. Aus welchen Bestandteilen besteht ein IP-Header/TCP-Header?
4. Welche Funktionen haben die IP-/TCP-Headerfelder?
5. Wie wird die Grösse eines IP Headers bestimmt?
6. Wie wird die Grösse einer TCP Paket Payload bestimmt?

C Programmierung

1. Welche Funktion hat eine Struktur?
2. Was sind die Bestandteile eine Struktur?
3. Wie können Array Elemente mit Pointern adressiert werden?
4. Wie wird unter C dereferenziert?
5. Kann man auf ein Char Array eine Struktur casten?
6. Was ist eine Callback Funktion und was ist ihre Aufgabe?
7. Wann lohnt sich der Einsatz einer Callback Funktion?
8. Wie kann man eine IP Adresse aus der Struktur *in_addr* in eine String umwandeln?
9. Wie findet man einen Substring in einem String?

2 Grundlagen

Für die Entwicklung des Vulnerabilitie Scanners sind Grundlegende Kentnisse auf folgenden Gebieten notwendig.

1. Libpcap
2. Arbeiten mit Netcat (nc)
3. IRC

2.1 Libpcap (1,5h)

Libpcap ist eine Bibliothek die das Abfangen und Verarbeiten von Netzwerkpaketen unter Win/Linux und Mac sehr leicht macht und ist der quasi Standard für "Packet Capture". Auf Grund der sehr guten Qualität der Dokumentation, sei an dieser Stelle auf die Man Pages und HowTos der benötigten Funktionen verwiesen. Sie müssen wissen wie die Signaturen aussehen, wofür die Bestandteile notwendig sind und welche Aufgabe die Funktion erfüllt.

Folgen Sie bitte der vorgegeben Reihenfolge,schlagen Sie Techniken und Fachwörter nach, die Ihnen unbekannt sind:

1. Programming with pcap
2. pcap_findalldevs()
3. pcap_setdirection()
4. pcap_set_promisc()
5. pcap_open_live()
6. pcap_loop()
7. Alle Man Pages der Libpcap
8. **Besonders hilfreich ist dieser Artikel**:
   Hakin9 Building a Sniffer with Libpcap

Es gibt einen wichtigen Fakt, der in der Dokumentation nicht auftaucht, der hier allerdings betrachtet werden soll. Die Callback Funktion wird bei Libpcap immer iterativ aufgerufen. Dies bedeutet, dass ein Paket erst dann verarbeitet werden kann, wenn die Verarbeitung des vorherigen abgeschlossen worden ist. Daraus ergeben sich zwei Konsequenzen für die Entwicklung eines Sniffers.

1. Pakete müssen effizient verarbeitet werden.
2. Alte noch nicht verarbeitete Pakete werden von neu ankommenden Paketen überschrieben.

Weiterhin gibt es keine Garantie, dass Pakete nach dem *return* der Callback Funktion noch im Speicher vorhanden sind. Die Speicherung eines Pakets ist deshalb die vorrangige Aufgabe der Callback Funktion Im Anschluss kümmert sich eine andere Funktion um deren Verarbeitung. Dazu werden *POSIX Threads* unter Linux eingesetzt. Da der Versuch so einfach wie möglich gestalltet sein soll, werden diese hier nur erwähnt und spielen in der späteren Aufgabenstellung keine Rolle. Dem interessierten Leser werden folgende Webseiten empfohlen:

1. Linux Posix Threads Tutorial
2. Advanced Introduction into Pthreads

Kontrollfragen für Libpcap

1. Welche Struktur speichert den Handler eines Devies, dass zum Abhören von Paketen verwendet wird?
2. Wie kann die maximale Anzahl der zu empfangenden Pakete eingestellt werden?
3. Wie und Wo wird die Callback Funktion für die Verarbeitung der Netzwerkpakete übergeben?
4. Wie werden die Funktionsparameter der Callback Funktion übergeben?
5. Welche C Funktion eignet sich zum schnellen Speichern von Paket-Header und Paket-Payload?
6. Wie kann auf Headerinformationen schnell zugegriffen werden?
7. Muss man dazu die Headerinformationen kopieren?
8. Was ist der Unterschied zwischen *pcap_next()* und *pcap_loop()*?
9. Wofür wird die Struktur *pcap_pkthdr* verwendet, welche Aufgaben erfüllen dessen Bestandteile?
10. Wie heissen die Standard C Header für Paket Header Strukturen unter Linux?

2.2 Netcat (0,5h)

Netcat wird häufig als das schweizer Taschenmesser der Hacker und Sysadmins bezeichnet. Im folgenden werden Sie einige Gründe dafür kennen lernen. Sie können den Installations und Konfigurationsteil des Dokuments überspringen, da Ihnen ein System mit vorkonfiguriertem Netcat zur Verfügung gestellt wird.

1. Introduction to Netcat

Kontrollfragen für Netcat

1. Wie überträgt man eine Datei mittels Netcat?
2. Wie gibt man den Port an, auf dem Netcat auf eine Verbindung wartet?
3. Wie kann ein Befehl über Netcat gesendet werden?
4. Werden Daten über Netcat verschlüsselt übertragen?
5. Wie kann man verschiedene Shell Kommandos auf einmal ausführen?

3 Aufgaben

Dieser Versuch ist in 2 Hauptaufgaben mit mehreren Unteraufgaben unterteilt. Dadurch wird die Komplexität des Versuchs verringert und ist leichter zu kontrollieren.

1. Entwickeln eines Sniffers:
   a) der die Pakete auf der Console ausgibt (ASCII Hex Codierung)
   b) der die Pakete nach TCP Paketen filtert und die Paket Payload ausgibt
   c) der die TCP Paketpayload nach einem String durchsucht
2. Zusammenarbeit zwischen Scanner und Sniffer. Der Sniffer lauscht nach Paketen, die auf einen verwundbaren IRC-Server hindeuten.

Der Versuch stellt ein vereinfachtes Szenario dar. Schwachstellen-Scanner wie Nessus[1] oder OpenVAS[2] sind in der Realitï¿½t wesentlich komplexer.

Alle benï¿½tigten Dateien sind in **/home/student/Praktikum/libpcap** zu finden.

3.1 Wichtiger Hinweis zum Bericht

Geben Sie in Ihrem Praktikums-Bericht als Lï¿½sung jeder Aufgabe ausschlieï¿½lich die **wichtigen Passagen** in ihrem Code an. Programmteile, die sich seit der letzten Aufgabe nicht verï¿½ndert haben, brauchen Sie nicht erneut zu zeigen. Sachen wie *includes*, welche bereits in den Templates stehen, sind irrelevant.

Erklï¿½ren Sie ihren Code. Ein reines Quellcode-Listing ist nicht ausreichend, um den Versuch zu bestehen!

Fï¿½gen Sie **zusï¿½tzlich** das komplette Programm zu jeder Unteraufgabe in den **Anhang** ihres Berichtes ein.

Seien Sie weiterhin darauf vorbereitet, den Quellcode auf Anfrage vorzeigen zu kï¿½nnen. Lï¿½schen Sie ihre Dateien also nicht vor Erhalt Ihrer Korrektur.

[1] `http://www.tenable.com/products/nessus-vulnerability-scanner`
[2] `http://www.openvas.org/`

3.2 Aufgabe 1: Entwickeln des Sniffers

In dieser Aufgabe entwickeln Sie ein C-Programm, welches die PCAP Bibliothek zum abhï¿½ren des lokalen IP-Verkehrs benutzt. In den Unteraufgaben werden Sie ihr Programm schrittweise erweitern.

3.2.1 Aufgabe 1.1

Im Ordner "libpcap"finden Sie ein Template für die erste Aufgabe. Sie sollen mit Hilfe von Libpcap jedes Paket vom Netzwerkinterface *eth0* abfangen und den kompletten Paketinhalt in Hex Zeichen auf der Konsole ausgeben. Folgende Informationen sollen zusätzlich ausgegeben werden:

- Einen Zähler, der für jedes Paket erhöht wird.
- Die Gesamtgrösse des Pakets, so wie es von Libpcap empfangen worden ist.
- Fügen Sie nach jedem Byte eine Leerstelle und nach 16 Zeichen einen Zeilenumbruch ein.

Hinweis

- Kompilieren Sie Programme die auf libpcap zugreifen immer mit dem Flag *-lpcap*
- Um immer nur zwei Hexzeichen auszugeben, verwenden Sie folgenden Befehl:

```
printf("%02x", packet[i] & 0xff);
```

3.2.2 Aufgabe 1.2

Entwickeln Sie in einer neuen C-Datei, basierend auf dem Quelltext aus Aufgabe 1.1, einen Sniffer, der nur den Inhalt der TCP-Paket Payload ausgibt. Es sollen nur Pakete ausgegeben werden, die das *ACK*-Flag gesetzt haben. Verfahren Sie ansonsten analog wie in Aufgabe 1.1 und geben die Payload als Hex Zeichen aus. Weiterhin sollen folgende Informationen separat mit ausgegeben werden:

- Empfänger IP-Adresse
- Empfänger TCP-Port
- Sender IP-Adresse
- Sender TCP-Port

Hinweis

- Sie können dazu die Strukturen aus *netinet/ip.h* und *netinet/tcp.h* verwenden.

3.2.3 Aufgabe 1.3

Erstellen Sie nun in einer neuen C-Datei, basierend auf dem Quelltext aus Aufgabe 1.2, einen Sniffer, der in der TCP-Paket Payload nach einem String sucht. In der Datei *suchstring.c* im Ordner "Vulnerability Scanner"finden Sie den zu verwendenen Suchstring. Falls der String gefunden worden ist, soll eine Meldung auf der Konsole ausgeben werden, dass ein exploitable IRC Server gefunden wurde. Zusätzlich sollen der Port und die IP-Adresse[3] des Servers ausgegeben werden.

[3] Pakete, welche keine IP-Pakete sind, also keine IP-Adresse enthalten, müssen nicht gedumpt werden.

3.3 Aufgabe 2: Suchen nach dem verwundbaren Server

Sie werden Ihren Scanner nun benutzen, um die Antworten eines Servers auf den verwundbaren String zu untersuchen. Dazu stehen Ihnen zwei Hilfsmittel zur Verfi¿½gung:

- Die Datei **fake_irc_server.sh** startet einen Server auf Port 6667, welcher auf Anfrage die Antwort des verwundbaren IRC-Servers liefert. Es handelt sich nicht um den echten IRC-Server, sondern lediglich um die Antwort dessen.
- Das Programm **scan.sh** startet einen einfachen Port-Scan auf allen 65535 TCP-Ports.

3.3.1 Aufgabe 2.1

i¿½ffnen Sie die Datei **scan.sh** im Text-Editor und verstehen Sie den Quellcode. Erklï¿½ren Sie in Ihrem Bericht die einzelnen Parameter des netcat-Aufrufs.

Hinweis: Falls Sie das Programm ausfi¿½hren und wi¿½hrenddessen mit Wireshark sniffen, kann es zum zeitweisen Einfrieren ihres Computers kommen. Das liegt an der i¿½berdurchschnittlich hohen Datenï¿½bertragungsrate des loopback-Interfaces.

3.3.2 Aufgabe 2.2

Fï¿½hren sie folgendes in genannter Reihenfolge durch:

- i¿½ffnen Sie eine Konsole und starten Sie den gefi¿½lschten IRC-Server.
- Starten Sie nun ihren Sniffer auf dem Interface **lo**. Dies ist das *loopback interface*.
- Starten sie den Scan.

Ihr Programm sollte nun den verwundbaren IRC-Server auf Port 6667 melden. Dokumentieren Sie ihre Ergebnisse mittels Screenshots oder Programmausgaben.

Literatur

[1] Adobe. PS language specifications. http://partners.adobe.com/public/developer/ps/index_specs.html.

[2] Dobbertin, Bosselaers, and Preneel. The hash function RIPEMD-160. http://homes.esat.kuleuven.be/ bosselae/ripemd160.html.

[3] Eastlake. US Secure Hash Algorithm 1, 2001. http://tools.ietf.org/html/rfc3174.

[4] Klima and Dufek. MD5-Collisions - Algorithm from Pavel Dufek: Modification of Klima´s program, 2008. http://cryptography.hyperlink.cz/MD5_collisions.html.

[5] Lucks and Daum. The Story of Alice and her Boss, 2005. http://www.cits.rub.de/imperia/md/content/magnus/rump_ec05.pdf.

[6] Rivest. The MD4 Message-Digest Algorithm, 1992. http://tools.ietf.org/html/rfc1320.

[7] Rivest. The MD5 Message-Digest Algorithm, 1992. http://tools.ietf.org/html/rfc1321.

[8] Wang and Yu. How to break MD5 and other hash functions. *Advances in Cryptology - EUROCRYPT 2005*, pages 19–35, 2005.

sources

http://www.programming-pcap.aldabaknocking.com/

http://www.tcpdump.org/manpages/pcap.3pcap.html
http://www.tcpdump.org/manpages/pcap_findalldevs.3pcap.html
http://www.tcpdump.org/manpages/pcap_setdirection.3pcap.html
http://www.tcpdump.org/manpages/pcap_set_promisc.3pcap.html
http://www.tcpdump.org/manpages/pcap_open_live.3pcap.html
http://www.tcpdump.org/manpages/pcap_loop.3pcap.html

http://linux.die.net/man/3/system