Netzsicherheit 1, WS 2020/2021
Übung 6: IKEv1 Cracking

Martin Grothe

Abgabe: Freitag, 18. Dezember 2020, 14:00h
Die Übungen sind im Moodle-Kurs zur Vorlesung elektronisch abzugeben.

1 Einleitung

Die Sicherung von Verbindungen im Internet kann über verschiedene Protokolle von statten gehen. Eines der weit verbreitetsten ist Transport-Layer-Security (TLS). Jedoch ist das Kerneinsatzgebiet von TLS eher die Sicherung von Daten auf der Anwendungsebene. Möchte eine Organisation oder Unternehmen jedoch einem Benutzer/Mitarbeiter den Zugang zu Teilen eines Netzwerks geben, so ist ein Virtual Private Network (VPN) eine bewährte Methode. Es gibt viele verschiedene Technologien die für die Umsetzung eines VPNs in Frage kommen. Eine Technologie ist IPSec.

Grundlegen besteht IPSec aus zwei Bestandteilen, dem Internet Key Exchange (IKE) und IPSec. Damit Nutzerdaten auf der IPSec Ebene verschlüsselt werden können, müssen zuvor Schlüssel und Algorithmen ausgehandelt werden. Dazu wird das Internet Key Exchange (IKE) Protokoll eingesetzt, welches aus den folgenden 3 Protokollen besteht:

1. Oakley Key Determination Protocol (Oakley)
2. Internet Security Association and Key Management Protocol (ISAKMP)
3. Secure Key Exchange Mechanism for Internet (SKEME).

2 IKE

IKE sorgt dafür, dass sich Kommunikationsparteien gegenüber einander Authentifizieren, Schlüsselmaterial erstellen und *Security Associations* aushandeln. Eine ausgehandelte *Security Associations* besteht zumeist aus definierten Algorithmen (z.B.: AES), Parametern (z.B.: Bitlänge), Modi (z.B.: CBC) und den Schlüsseln für die entsprechenden Algorithmen[1].

In der Übung wollen wir uns IKE in Version 1 anschauen.

2.1 Version 1

Generell besteht ein Durchlauf von IKE immer aus 2 Phasen (1. und 2. Phase), diese erfüllen unterschiedliche Ziele. Diese Phasen sind zusammen mit IPSec in Abbildung 1 dargestellt.

[1] https://tools.ietf.org/html/rfc2409#page-5

Phase 1 Dient der Authentifizierung der Parteien und zum Aushandeln, sowie bestätigen des gemeinsamen Schlüssels (Diffie-Hellmann) und frischer Zufallswerte (Nonces).

Phase 2 In dieser Phase wird der zuvor ausgehandelte Schlüssel aktualisiert und weitere Parameter festgelegt (z.B.: IPSec Modus: AH[2] oder ESP[3]). Ebenfalls wird in dieser Phase festgelegt, welche IP-Adressen über die IPSec Verbindung erreichbar sein sollen und von welcher IP-Adresse die Verbindung aufgebaut wird.

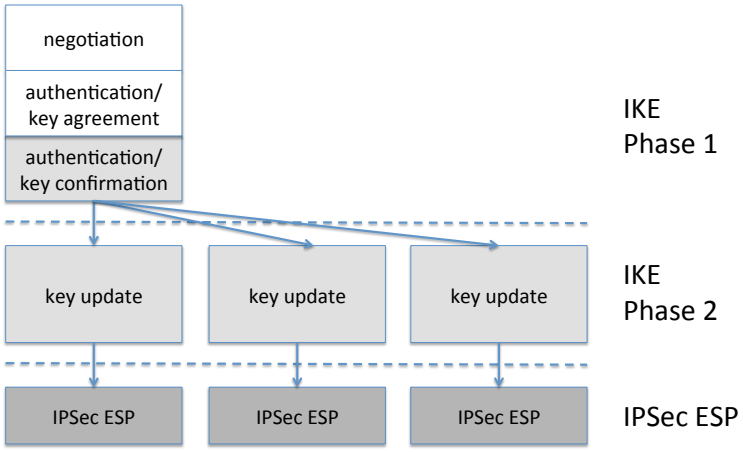

Abbildung 1: IPSec und IKE Phasen

Modi von Phase 1 Bei der Verwendung von IKE können in der Phase 1 zwei Modi eingesetzt werden, der *Main Mode* und der *Aggressive Mode*. Ersterer benötigt 6 Nachrichten und der Letztere 3 Nachrichten zum Aushandeln der Parameter von Phase 1. Der *Aggressive Mode* benötigt zwar weniger Zeit/Nachrichten für das Aushandeln der Parameter ist dadurch jedoch nicht so sicher wie der *Main Mode*.

Authentifizierung Varianten In IKE gibt es 3 verschiedene Möglichkeiten wie sich die Teilnehmer authentifizieren:

1. Signatur
   - Der Teilnehmer ist in der Lage eine Signatur zu erstellen, die mittels hinterlegtem Zertifikat verifiziert werden kann.
2. Passwort
   - Der Teilnehmer kennt ein geheimes Passwort, dass er vor beginn des IKE Durchlaufs eingibt.
3. Public Key Verschlüsselung von Zufallswerten
   - Der Teilnehmer kann einen Wert entschlüsseln, welcher mit seinem öffentlichen Schlüssel verschlüsselt worden ist.

In dieser Übung geht es um die Sicherheit der Authentifizierung mittels Password.

[2]https://de.wikipedia.org/wiki/IPsec#Authentication_Header_.28AH.29
[3]https://de.wikipedia.org/wiki/IPsec#Encapsulating_Security_Payload_(ESP)

Password Authentifizierung im Aggressive Mode Wie zuvor bereits erwähnt, besteht die *Aggressive Mode* Authentifizierung aus 3 Nachrichten, diese sind in Abbildung 2 dargestellt.

Der Initiator startet die Kommunikation und erstellt dazu einen *Header*, welcher zufällig gewählten *Cookie Wert* des Initiator enthält und den *Cookie Wert* des Responders mit 0 initialisiert. Als nächstes wird die Nachricht m_1 erstellt. Diese besteht aus sogenannten *proposals*, welche eine Zusammenstellung von unterstützten Algorithmen und Parametern usw. ist. Ferner wird ein zufälliger *Key Exchange Werts* (g^x) für den Diffie-Hellmann Schlüsselaustauschs generiert. Als vorletzter Wert wird die *Nonce* (Number used once) n_I generiert. Anschließend hängt der Initiator seinen Identifikationswerts ID_I ans Ende der Nachricht. Danach sendet er den *Hdr* und die Nachricht m_1 an den *Responder*.

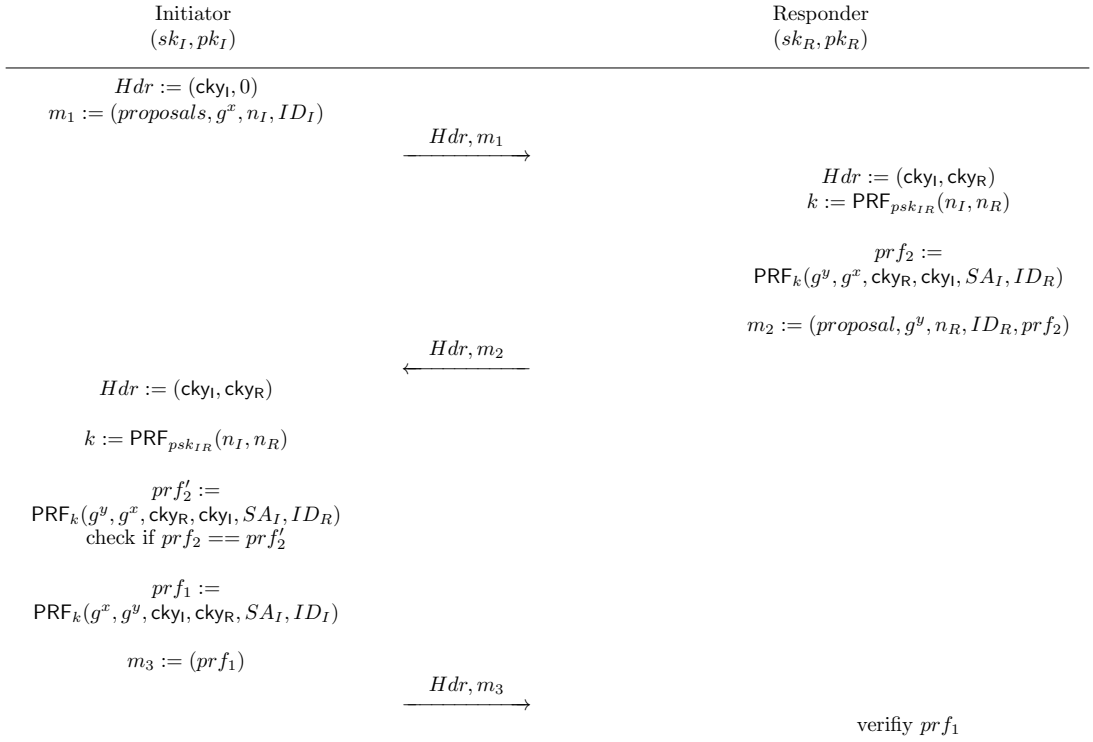

Abbildung 2: IKEv1 Aggressive Mode mit Password

Der *Responder* empfängt die Nachricht m_1 und verarbeitet diese. Er initialisiert seinen *Cookie Wert* (cky_R) im *Hdr* und generiert ebenfalls *proposals*, *Key Exchange Wert*, und *Nonce* für die Nachricht m_2. Zur Erstellung des Schlüssels K verwendet der Responder die beiden *Nonces* zusammen mit dem Passwort (psk_{IR}) in einer PRF. In IKE wird als PRF ein HMAC[4] verwendet. Das Passwort dient dabei als Schlüssel. Der dadurch generierte Schlüssel wird anschließend als Schlüssel in der zweiten PRF verwendet, welche den Wert prf_2 generiert. Die Werte die dabei eingehen sind die Key Exchange Werte, Cookies, die Security Association Payload und die ID des Responders. Im letzten letzten Schritt sendet der Responder die konstruierte Nachricht m_2 an den Initiator zurück.

Der Initiator überprüft den Hashwert (prf_2) in Nachricht m_2, in dem er die gleichen Werte in die HMAC Funktion eingibt und den Hashwert berechnet, stimmen beide überein, so fährt er mit dem Protokollablauf fort, andernfalls bricht er den Verbindungsaufbau ab. Im weiteren Verlauf von Phase 1 generiert der Initiator den Hashwert prf_1 und vertauscht dabei fast alle Werte, außer SA_I und ID_R, letzterer wird durch ID_I ersetzt.

[4]https://de.wikipedia.org/wiki/Keyed-Hash_Message_Authentication_Code

Anschließend sendet der Initiator den Hashwert mit einem *Hdr* an den Responder, welcher diesen Wert ebenfalls verifiziert.

3 Aufgabe: Angriff auf IKEv1 Aggressive Mode mit Password Authenfizierung

Ihre Aufgabe ist es, den Password Authentifizierungsmodus von IKEv1 im Aggressive Mode anzugreifen. Dazu wird Ihnen ein Wörterbuch und eine PCAP Datei gestellt. In der PCAP Datei finden Sie einen Handshake, welche den oben beschriebenen Ablauf mit den Nachrichten m_1 bis m_3 enthält.

Ihre Aufgabe Schreiben Sie ein Python 3 Script, welches die relevanten Werte aus der PCAP Datei extrahiert und für jedes Passwort im Wörterbuch den Hashwert berechnet und diesen gegen den in Nachricht m_2 enthalten Hashwert (prf_2) überprüft. Stimmen diese Werte überein, gibt das Script, dass Passwort aus und beendet sich selbst. Überlegen Sie sich dazu, welche Werte Sie jeweils benötigen und welche Werte das Passwort beeinflusst. Das Script soll so generisch sein, dass auch eine andere Pcap Datei ohne Fehler verarbeitet werden kann. Das heißt, vermeiden Sie überall da wo es möglich ist direkte Angaben von Byte-Positionen. Jede Missachtung führt zu Punktabzügen.

Ihnen stehen zum Testen zwei Pcap Dateien und zwei Wörterbücher zur Verfügung.

1. *pcaps/ikev1-psk-aggressive-mode.pcapng* enthält den zu knackenden IKE Handshake
2. *pcaps/ikev1-psk-aggressive-mode-simple.pcapng* enthält einen IKE Handshake, beim welchem ein schwaches Passwort verwendet worden ist.
3. *dict/list.txt* beinhaltet ein Wörtbuch mit mehreren 100 Tausend einträgen, welches sie verwenden sollen, um die erste Pcap Datei zu brechen.
4. *dict/list-simple.txt* ist zum testen Ihres Scripts gedacht, mit welchem Sie den einfachen IKE Handshake brechen können.

Hilfen Damit Sie schneller starten können, sind 3 Python Dateien bereits vorgegen, welche Ihnen bei der Arbeit eine Struktur und Unit-Tests zur Verfügung stellen, gegen die Sie ihre Implementierung testen können.
Bitte beachten Sie, dass die Unit-Tests gegen die *simplen* Test Werte (siehe oben) und nicht gegen die zu lösenden Werte der Aufgabenstellung erstellt wurden Starten Sie also erst mit den simplen Werten, um Ihre Implementierung zu überprüfen, im Anschluss können Sie damit die Aufgabe lösen. Um Ihnen eine einfache Parameter und Return Werte übergabe zu ermöglichen sind alle Funktionen mit sogenannten *Type-Hints* versehen. Dadurch sehen Sie den Typ des Parameters bzw. des Return Werts.

Scapy Verwenden Sie in Ihrem Script Scpay. Sollten Sie noch nicht mit Scapy gearbeitet haben hier eine kurze Erklärung der Entwickler, was scpay ist:[5]:

> Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.). It also performs very well at a lot of other specific tasks that most other tools can't handle, like sending invalid frames, injecting your own 802.11 frames, combining technics (VLAN

[5] http://www.secdev.org/projects/scapy/

hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel, ...), etc. See

Um die Aufgabe zu bearbeiten, benötigen Sie Scapy v2.x. Eine Installationsanleitung finden sie hier.

Alle relevanten Dateien, welche Sie verwenden sollen finden sie als Zip Datei. Verwenden Sie unbedingt die Python Template Dateien, andernfalls wird die Aufgabe mit 0 Punkten bewertet.

Geben Sie das Passwort welches in der *ikev1-psk-aggressive-mode.pcapng* verwendet worden ist auf Ihrer entsprechenden E-Learning Plattform ein. Laden Sie im Anschluss Ihren Ordner mit den unten stehenden Dateien und Ordner auf die E-Learning Plattform als ZIP Datei:

```
requirement.txt (do not touch)
default.nix (do not touch)
pcaps/ikev1-psk-aggressive-mode-simple.pcapng
pcaps/ikev1-psk-aggressive-mode.pcapng
dict/list-simple.txt
dict/list-simple.txt
main.py
ikev1_payloadParser.py
ikev1_pcapReader.py
```

4 Hinweise

IKE Werte in PCAP Dateien Bei der Variable SA_I in der Abbildung 2 handelt es sich um die *ISAKMP Security Association Payload* ab dem *DOI* (Domain of Interpretation) Wert bis zum Ende der *ISAKMP Security Association Payload*. Der Identifikationswert (ID_R) besteht aus den Werten (ID Type || Protocol ID || Port || Identification Data).

4.1 Entwicklungsumgebung

Eine gute graphische Umgebung mit der ich selber arbeite ist PyCharm, welche von JetBrains entwickelt wird und als Community Edition kostenlos zur Benutzung frei verfügbar ist. Sollten Sie PyCharm nicht verwenden wollen, finden Sie hier eine Auflistung anderer Entwicklungsumgebungen.

4.2 Python

Umwandeln von Bytes in HEX Zeichen Sie werden in dieser Übung einige Konvertierungen vornehmen müssen. Für das Umwandeln von Bytes in Hex Zeichen kann `binascii` und dessen Funktion `hexlify` verwendet werden.

HMAC und Algorithm Einen HMAC finden Sie in der Python Bibliothek `hmac` und den dazu passenden Algorithmus in `hashlib`.

4.3 Scapy

Eine erste Einleitung für die Verwendung von Scapy finden Sie online. Weitere Schritte werden in Grundlagen erklärt. Wie PCAP Dateien verarbeitet werden finden Sie hier. Ein Cheat Sheet für scapy ist ebenfalls online verfügbar.

Weitere nützliche Kommandos sind folgende:

Es muss jeweils vorher scapy importiert werden, dies kann mittels `from scapy.all import *`.

- Auflisten alle unterstützten Protokolle `ls()`
- Auflisten alle unterstützten Kommandos `lsc()`
- Auflisten alle Layer im Packet `packet.summary()`
- Parsen und Anzeigen eines Packets und seines Inhalts `packet.show()` oder `packet.show2()`
- Packet Layer durchiterieren `packet[0]` ist das erste Layer, `packet[1]` entspricht dem zweiten Layer. Es können sind auch multiarray Zugriffe mittels Indexangabe möglich `packet[0][1]`
- Sollte man wissen, dass ein Packet ein layer enthält, so kann direkt darauf Zugegriffen werden, mittels `packet[layername]`. Wichtig dabei ist, dass der Namen ohne Anführungszeichen eingeben wird, auch wenn die Entwicklungsumgebung dies als Fehler ansieht, es ist richtig, sofern der Layername richtig ist.

- Ist ein direkter Zugriff auf bestimmte Bytes gewollt, so kann dies mittels `packetbytes[4:]`, `packetbytes[:4]` oder `packetbytes[2:4]` realisiert werden. Erstere gibt alle Bytes ab der 4. Position zurück, der zweite Befehl, alle Bytes bis zum 4. Byte aus, und letzterer alle Bytes zwischen der 2. und der 4. Position.