Nutzerauthentifizierung

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 1.3.7

Folien: [HTML] https://delors.github.io/sec-nutzerauthentifizierung/folien.de.rst.html
[PDF] https://delors.github.io/sec-nutzerauthentifizierung/folien.de.rst.html.pdf
Fehler melden: https://github.com/Delors/delors.github.io/issues

1. Grundlagen

2

Challenge-Response Authentifizierung

Beobachtung

Die Verwendung einer (kryptographischen) Hashfunktion alleine ist nicht ausreichend zur sicheren *Benutzerauthentifizierung über eine nicht-sichere Verbindung*.

Eine einfache Replay-Attacke ist möglich.

Ein einfaches Protokoll basierend auf einer Hashfunktion f wäre:

Challenge-Response Protokoll ≙ *Herausforderung- und Antwortprotokoll*

Alice	unsicherer Kanal	Bob
Gibt Benutzerkennung ID ein	→ sendet ID	sucht zu ID Schlüssel K in der Datenbank
		↓
Gibt Passwort K' ein	sendet r ←	wählt zufällige Zahl r
		↓
berechnet: $Res' = f(K', r)$	→ sendet Res'	$f(K, r)? = Res'$

Frage

Wie bewerten Sie die Sicherheit genau dieses Protokolls/Ansatzes?

Zero-Knowledge Protokolle

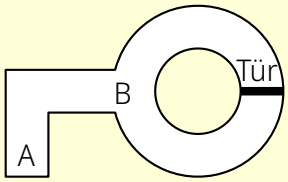
Die Idee ist, dass man jemanden davon überzeugen möchte, dass man eine bestimmte Information hat, ohne diese Information zu offenbaren.

Beispiel

Der geheimnisvolle Geheimgang

Peggy möchte Victor überzeugen, dass Sie den Code zur Tür kennt, ohne ihn zu offenbaren.

- Peggy wählt einen der Wege zur Tür, während Victor an der Stelle A steht und darauf wartet, dass Sie bei der Tür ist.
- Sobald Peggy Bescheid gibt, dass Sie an der Tür angekommen ist, geht Victor zu Punkt B und sagt Peggy auf welchem Weg sie zurückkommen soll.
- Kommt sie auf dem falschen Weg zurück, dann kennt sie den Code der Tür (offensichtlich) nicht. Kommt sie auf dem richtigen Weg zurück, könnte es noch immer Zufall gewesen sein mit Wahrscheinlichkeit $\frac{1}{2}$.



Bei n Spielen ist die Wahrscheinlichkeit, dass Peggy immer zufällig den korrekt Weg gewählt hat $\frac{1}{2^n}$.

Viele Zero-Knowledge Protokolle basieren darauf, dass man im Prinzip ein Spiel spielt, das man auch zufällig gewinnen kann. Durch die Wiederholung des Spiels wird die Wahrscheinlichkeit jedoch für permanentes zufälliges Gewinnen sehr schnell sehr klein (exponentiell). Somit kann man für praktische Zwecke hinreichend sicher sein, dass der Beweisführende (im Beispiel Peggy) über das Wissen verfügt, das er vorgibt zu besitzen, wenn er immer gewinnt.

Nach 20 Runden ist die Wahreinschlichkeit nur noch $1/2^{20} = 1/1048576$.

Mit 128 Runden erreicht man ein Sicherheitsniveau, das vergleichbar ist mit anderen kryptographischen Verfahren (AES-128, SHA-256, ...).

Fiat-Shamir Protokoll

Voraussetzungen

- gegeben zwei zufällige Primzahlen p und q und $n = p \cdot q$
- Peggys geheimer Schlüssel s wird dann zufällig bestimmt; $s \in \mathbb{Z}_n^*$ und s coprimal zu n.
- Der öffentliche Schlüssel wird dann wie folgt berechnet: $v = s^{-2} \mod n$. Der öffentlichen Schlüssel besteht dann aus den zwei Zahlen v und n.

Protokoll

1. Peggy berechnet x unter Verwendung einer beliebigen Zufallszahlen $r \in \mathbb{Z}_n^*$:

   $x = r^2 \mod n$

2. Peggy sendet die Zahl an Victor.
3. Victor wählt zufällig ein Bit $b \in \{0, 1\}$
4. Peggy berechnet $y = r \cdot s^b \mod n$
5. Peggy sendet y an Victor.
6. Victor testet: $x \cdot v^{-b} \mod n \stackrel{?}{=} y^2 \mod n$

Übung: klassisches Fiat-Shamir-Protokoll

1.1. Beispiel: Fiat-Shamir-Protokoll mit kleinen Zahlen

Zwei Parteien, Peggy (die sich authentifizieren möchte) und Victor (der Prüfer), führen das Fiat-Shamir-Protokoll durch. Verwenden Sie die folgenden Werte:

- $p = 3, q = 7 \rightarrow n = p \cdot q = 21$
- Peggys geheimer Schlüssel ist $s = 2$
- Peggy wählt die Zufallszahl $r = 4$
- Victor wählt die Herausforderung $b = 1$

Beantworten Sie folgende Fragen:

1. Berechnen Sie den öffentlichen Schlüssel v.
2. Berechnen Sie den Wert x, den Peggy an Victor sendet.
3. Berechnen Sie die Antwort y, die Peggy an Victor sendet.
4. Führen Sie die Verifikation als Victor durch

2. Einmalpasswortsysteme

7

Password Sniffing

In der Anfangszeit: unverschlüsselte Übertragung von Passwörtern (telnet, ftp, ...)

In der Übergangszeit:
Verwendung von Einmal-Passwörtern (S/Key, ...)

Heute: Passwörter werden verschlüsselt übertragen (ssh, https, ...)
Zusätzliche Absicherung durch Zwei-Faktor-Authentifizierung (basierend auf Einmalpassworten: TOTP, ...)

Unverschlüsselte Passworte können leicht mittels eines Sniffers, der den Netzwerkverkehr mitschneidet (z. B. Wireshark), abgefangen werden.

Einmal-Passwörter

Die Idee ist, dass Passwörter nur genau einmal gültig sind und nicht wiederverwendbar sind.

- Tokens (z. B. RSA SecurID)
- Codebuch: Liste von Einmal-Passwörtern, die das gemeinsame Geheimnis sind.
- S/Key: Passwort „wird mit einem Zähler kombiniert“ und dann gehasht.

Das S/Key Verfahren

Einmal-Passwort-System nach Codebuch-Verfahren.

Initialisierung

1. Der Nutzer gibt sein Passwort W ein; dies ist der geheime Schlüssel.
   (Sollte W bekannt werden, dann ist die Sicherheit des Verfahrens nicht mehr gewährleistet.)
2. Eine kryptografische Hash-Funktion H wird n-mal auf W angewandt, wodurch eine Hash-Kette von n einmaligen Passwörtern entsteht. $H(W), H(H(W)), \ldots, H^n(W)$
3. Das initiale Passwort wird verworfen.
4. Der Benutzer erhält die n Passwörter, die in umgekehrter Reihenfolge ausgedruckt werden: $H^n(W), H^{n-1}(W), \ldots, H(H(W)), H(W)$.
5. Nur das Passwort $H^n(W)$, das an erster Stelle der Liste des Benutzers steht, der Wert von n und ggf. ein Salt, wird auf dem Server gespeichert.

Anmeldung

Identifiziere das letzte verwendete Passwort n.

- Der Server fragt den Nutzer nach dem Passwort $n-1$ (d. h. $H^{n-1}(W)$) und übermittelt ggf. auch den Salt.
- Der Server hasht das Passwort und vergleicht es dann mit dem gespeicherten Passwort $H^n(W)$.
- Ist das Passwort korrekt, dann wird der Nutzer angemeldet und der Server speichert das Passwort $H^{n-1}(W)$ als neues Passwort $H^n(W)$ und dekrementiert n.

Im Original basiert S/Key auf der kryptographischen Hashfunktion MD4. Ein Austausch wäre aber selbstverständlich möglich!

Intern verwendet S/KEY 64-bit Zahlen. Für die Benutzbarkeit werden diese Zahlen auf sechs kurze Wörter, von ein bis vier Zeichen, aus einem öffentlich zugänglichen 2048-Wörter-Wörterbuch ($2048 = 2^{11}$) abgebildet. Zum Beispiel wird eine 64-Bit-Zahl auf "ROY HURT SKI FAIL GRIM KNEE" abgebildet.

HMAC-based one-time password (HOTP)[1]

- ermöglicht die Erzeugung von Einmal-Passwörtern auf Basis eines geheimen Schlüssels und eines Zählers; Parameter:
  - Ein kryptografisches Hash-Verfahren H (Standard ist SHA-1 mit 160 Bit)
  - einen geheimen Schlüssel K, der eine beliebige Bytefolge ist
  - Ein Zähler C, der die Anzahl der Iterationen zählt
  - Länge des Passworts: d (6-10, Standardwert ist 6, empfohlen werden 6-8)
- Zur Authentifizierung berechnen beide das Einmalpasswort (HOTP) und dann vergleicht der Server den Wert mit dem vom Client übermittelten Wert.

  Berechnung aus dem Schlüssel K und dem Zähler C:

  $HOTP(K, C) = truncate(HMAC_H(K, C))$

  $truncate(MAC) = extract31(MAC, MAC[(19 \times 8 + 4) : (19 \times 8 + 7)])$

  $HOTP\ value = HOTP(K, C)\ mod\ 10^d$ (führende Nullen werden nicht abgeschnitten)

$truncate$ verwendet die 4 niederwertigsten Bits des MAC als Byte-Offset i in den MAC. Der Wert 19 kommt daher, dass ein SHA-1 160 Bit hat und $160/8 = 20$ Byte.

$extract31$ extrahiert 31 Bit aus dem MAC. Das höchstwertig Bit wird (wenn es nicht 0 ist) entsprechend maskiert. Eine Schwäche des Algorithmus ist, dass beide Seiten den Zähler erhöhen müssen und, falls die Zähler aus dem Tritt geraten, ggf. eine Resynchronisation notwendig ist.

---

[1] https://www.rfc-editor.org/rfc/rfc4226

Time-based one-time password (TOTP)[2]

- Erzeugung von zeitlich limitierten Einmal-Passwörtern (z. B. 30 Sekunden)
- Basierend auf einem vorher ausgetauschten geheimen Schlüssel und der aktuellen Zeit
  Z. B. Unix-Zeit in Sekunden (ganzzahlig) und danach gerundet auf 30 Sekunden.
- Es wird das HOTP Verfahren mit der Zeit als Zähler verwendet und entweder SHA-256 oder SHA-512 als Hashverfahren, d. h. TOTP $value(K)$ = HOTP $value(K, C_T)$, wobei T die „aktuelle Zeit“ ist.
  $C_T = \lfloor \frac{T - T_0}{T_X} \rfloor$
  T_X ist die Länge eines Zeitintervalls (z. B. 30 Sekunden)
  T ist die aktuelle Zeit in Sekunden seit einer bestimmten Epoche
  T_0 ist bei Verwendung der Unix-Zeit 0
  C_T ist somit die Anzahl der Dauern T_X zwischen T_0 und T

Das Verfahren verlangt somit, dass die Uhren von Server und Client (hinreichend) synchronisiert sind.

---

[2] https://www.rfc-editor.org/rfc/rfc6238

Übung

2.1. S/Key

1. Welche Vorteile bieten Einmalpasswortsysteme gegenüber Systemen mit mehrfach zu verwendenden Passworten?
2. Welchen Angriffen sind Einmalpasswortsysteme weiterhin ausgesetzt?
3. Generieren Sie eine Liste von Einmalpassworten mit Initialwert $r = 769$. Generieren Sie $H(r)$ bis $H^6(r)$ wenn die Einwegfunktion hier der Einfachheit halber $H(x) = x^2 \ mod \ 1000$ ist.
4. Wie oft kann sich der Benutzer anmelden? Wie sieht seine Liste aus?
5. Welchen Wert speichert der Server vor dem ersten Anmeldevorgang?
6. Spielen Sie zwei Anmeldevorgänge durch.
7. Wenn ein Passwort $H^L(W), 1 < L < N$ bekannt ist, welche Auswirkungen hat dies auf die Sicherheit des Verfahrens?

Übung

2.2. HOTP

Gegeben sei der folgende MAC:

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Mac	bc	9f	aa	ae	1e	35	d5	2f	3d	ea	96	51	da	12	cd	36	62	7b	84	03

Berechnen Sie den HOTP Wert für $d = 6$.

Übung

2.3. TOTP

Identifizieren Sie die Vor- und Nachteile von TOTP gegenüber S/Key und fragen Sie sich an welcher Stelle es (aus Sicherheitsperspektive) mögliche Schwächen gibt?

Die Standardzeitspanne ist 30 Sekunden. Welcher Konsequenzen hätte eine deutliche Verlängerung bzg. Verkürzung der Zeitspanne?