

# Vorlesung Sicherheit

Dennis Hofheinz

ITI, KIT

29.06.2017

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

## 1 Benutzerauthentifikation

### ■ Motivation

- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- **Szenario:** Nutzer  $U$  möchte sich auf Server  $S$  einloggen
- **Frage:** Was zeichnet  $U$  aus?
  - Üblich:  $U$  kennt Passwort  $pw$
  - Möglich: biometrische Merkmale (Fingerabdruck, Irisscan)
  - Etwas exotischer: Wissen/Fähigkeiten von  $U$
- **Annahme zunächst:**  $U$  kennt Passwort  $pw$
- **Frage:** Wie könnte/sollte Überprüfung aussehen?

- Erster Versuch:

$$U_{pw} \xrightarrow{pw} S_{pw}$$

(Server vergleicht  $pw$  mit für  $U$  gespeichertem Passwort)

- **Frage:** Warum nicht optimal?
- **Antwort:** Server kennt  $pw$ 
  - Problematisch, weil Server *alle*  $pw$  kennt
  - Server korrumpiert  $\Rightarrow$  alle  $pw$  öffentlich
  - Heikel: Nutzer verwenden  $pw$  evtl. auch bei anderen Diensten

# Zweiter Versuch

- Zweiter Versuch (schon angerissen bei Hashfunktionen):

$$U_{pw} \xrightarrow{H(pw)} S_{H(pw)}$$

- **Idee:** Server speichert nur  $H(pw)$  für Hashfunktion  $H$
- **Frage:** Verbesserung?
  - Server muss nur  $H(pw)$  (und nicht  $pw$ ) speichern
  - Bringt keine Verbesserung, weil  $H(pw)$  schon hinreichend!
- Besser:

$$U_{pw} \xrightarrow{pw} S_{H(pw)}$$

(Server wendet  $H$  auf Nutzereingabe an und vergleicht)

- **Ziel:** Das Verfahren

$$U_{pw} \xrightarrow{pw} S_{H(pw)}$$

näher beleuchten (Angriffe, Verbesserungen,  
Implementierungen)

## 1 Benutzerauthentifikation

- Motivation
- **Szenario**
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- **Szenario:** Server korrumpiert, (Liste von)  $H(\text{pw})$  bekannt
- **Angriffsziel:** Finde  $\text{pw}$
- Bei sicherer Hashfunktion  $H$  schwierig,  $H$  zu invertieren. . .
- . . . wenn  $H$ -Eingabe  $\text{pw}$  gleichverteilt gewählt
- **Aber:** Passwörter üblicherweise nicht gleichverteilt

# Häufige Passwörter

- **Passwort-Hitliste:** (englischsprachig, 2012, Quelle: SplashData)

10 baseball

9 111111

8 dragon

7 letmein

6 monkey

5 qwerty

4 abc123

3 12345678

2 123456

1 password

- **Kommen diese Passwörter oft vor?** (Quelle: Mark Burnett)

- 91% nutzen eins der 1000 häufigsten Passwörter

- 99,8% nutzen eins der 10000 häufigsten Passwörter

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- **Wörterbuchangriffe**
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- **Gegeben:**  $H(\text{pw})$  und ein Wörterbuch
- **Ziel:** Finde  $\text{pw}$
- **Vorgehen:** Vergleiche  $H(\text{pw})$  mit  $H(\text{pw}')$  für alle Wörterbucheinträge  $\text{pw}'$
- **Problem:** Sehr langsam, wenn Wörterbuch riesig
- **Hier: einige Annahmen über Passwort**
  - Server erzwingt nichttriviale Passwörter
  - Passwort nicht trivial ratbar ( $\Rightarrow$  Wörterbuch groß)

- **Problem:** Wörterbuchangriff langsam
- **Lösung:** *Sortiere* Wörterbuch nach Hashwert:

$$\begin{array}{ccc} ( H(pw_1) & , & pw_1 ) \\ ( H(pw_2) & , & pw_2 ) \\ & \dots & \dots \end{array}$$

mit  $H(pw_1) < H(pw_2) < \dots$

- Sortieren einmal nötig, danach binäre Suche möglich
- Allerdings: hoher Speicherplatzbedarf, wenn Liste lang

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- **Komprimieren des Wörterbuchs**
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- **Problem:** Wörterbuch zu groß
- **Idee:** Tausche Speicherplatz gegen Rechenzeit (Time-Memory-Tradeoff)
- **Genauer:** Verlagere Arbeit
  - Wörterbuch verkleinert
  - Fehlende Teile beim Angriff selektiv rekonstruiert
  - Schwierigkeit: nur *benötigte* Teile sollen rekonstruiert werden

- **Grundidee:** betrachte *Ketten* von Hash-Passwort-Paaren

$$(H(\text{pw}_{1,1}), \text{pw}_{1,1}) \xrightarrow{f} (H(\text{pw}_{1,2}), \text{pw}_{1,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{1,m}), \text{pw}_{1,m})$$

$$(H(\text{pw}_{2,1}), \text{pw}_{2,1}) \xrightarrow{f} (H(\text{pw}_{2,2}), \text{pw}_{2,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{2,m}), \text{pw}_{2,m})$$

...

$$(H(\text{pw}_{n,1}), \text{pw}_{n,1}) \xrightarrow{f} (H(\text{pw}_{n,2}), \text{pw}_{n,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{n,m}), \text{pw}_{n,m})$$

- Hier  $f$  Funktion, die nächstes Tupel aus vorherigem ableitet
- Ergibt matrixartige Anordnung, **spart so noch keinen Platz**
- Tatsächlich geht Sortierung verloren, **Angriff unklar**

- **Reparieren:** so funktioniert Angriff jetzt

$$\begin{aligned} & (H(\text{pw}_{1,1}), \text{pw}_{1,1}) \xrightarrow{f} (H(\text{pw}_{1,2}), \text{pw}_{1,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{1,m}), \text{pw}_{1,m}) \\ & \dots \\ & (H(\text{pw}_{n,1}), \text{pw}_{n,1}) \xrightarrow{f} (H(\text{pw}_{n,2}), \text{pw}_{n,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{n,m}), \text{pw}_{n,m}) \end{aligned}$$

- Sortiere Liste initial nach Endpunkten  $H(\text{pw}_{i,m})$
- Wähle  $f$ , das nur von Hashwert abhängt
- Angreifer kann nun gegebenes  $H(\text{pw}^*)$  mit  $f$  weiterentwickeln:

$$H(\text{pw}^*) \xrightarrow{f} (H(\text{pw}_1^*), \text{pw}_1^*) \xrightarrow{f} (H(\text{pw}_2^*), \text{pw}_2^*) \xrightarrow{f} \dots$$

... und in der Liste der Endpunkte (effizient binär!) suchen

- Angenommen, Angreifer findet gemeinsamen Endpunkt:

$$\begin{aligned} (H(\text{pw}_{i,1}), \text{pw}_{i,1}) &\xrightarrow{f} (H(\text{pw}_{i,2}), \text{pw}_{i,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{i,m}), \text{pw}_{i,m}) \\ H(\text{pw}^*) &\xrightarrow{f} (H(\text{pw}_1^*), \text{pw}_1^*) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{m^*}^*), \text{pw}_{m^*}^*) \end{aligned}$$

mit  $\text{pw}_{m^*}^* = \text{pw}_{i,m}$  und  $m^* \leq m$

- Angreifer geht  $i$ -te Kette in Liste von vorn durch
- Hoffnung:  $(H(\text{pw}^*), \text{pw}^*)$  taucht auf (dann  $\text{pw}^*$  gefunden)
- Problem bei  $f$ -Kollision  
(dann gemeinsames Kettenende bei unterschiedlichen Ketten möglich)

# Wie die Liste komprimiert wird

- **Beobachtung:** nur Anfangs- und Endpunkte der Liste

$$(H(\text{pw}_{1,1}), \text{pw}_{1,1}) \xrightarrow{f} (H(\text{pw}_{1,2}), \text{pw}_{1,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{1,m}), \text{pw}_{1,m})$$

...

$$(H(\text{pw}_{n,1}), \text{pw}_{n,1}) \xrightarrow{f} (H(\text{pw}_{n,2}), \text{pw}_{n,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\text{pw}_{n,m}), \text{pw}_{n,m})$$

benötigt (benötigter Rest kann on-the-fly generiert werden)

- **Konsequenz:** Liste kann auf

$$\begin{array}{ccc} (H(\text{pw}_{1,1}), \text{pw}_{1,1}) & , & (H(\text{pw}_{1,m}), \text{pw}_{1,m}) \\ & & \dots \\ (H(\text{pw}_{n,1}), \text{pw}_{n,1}) & , & (H(\text{pw}_{n,m}), \text{pw}_{n,m}) \end{array}$$

komprimiert werden (spart Faktor  $m/2$  an Speicherplatz)

# Anforderungen an $f$

- Funktion  $f$  sollte einige Anforderungen erfüllen
  - $f$  sollte „sinnvolle“ Passwörter (mit Hashes) ausgeben
  - $f((H(\text{pw}), \text{pw}))$  sollte nur von  $H(\text{pw})$  abhängen  
(damit Angreifer  $f$  auf gegebenes  $H(\text{pw}^*)$  anwenden kann)
  - $f$  sollte „hinreichend“ kollisionsresistent sein
  - Insbesondere:  $f$  sollte Passwortraum „gut abdecken“
  - Spagat:  $f$  willkürlich genug, um Kollisionen zu vermeiden, aber nicht zu willkürlich, um häufige Passwörter zu finden
- Mögliches  $f$ :
  - $f$  nimmt die ersten 40 Bits von  $H(\text{pw})$ , „arrangiert“ diese in eine sinnvolle Buchstaben-Sonderzeichen-Zahlenkombination  $\text{pw}'$  und gibt  $(H(\text{pw}'), \text{pw}')$  aus

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- **Rainbow Tables**
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- Problem komprimierter Listen: „Überlappung“ von Ketten

$$\begin{aligned} & (H(\mathbf{pw}_{i,1}), \mathbf{pw}_{i,1}) \xrightarrow{f} (H(\mathbf{pw}_{i,2}), \mathbf{pw}_{i,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\mathbf{pw}_{i,m}), \mathbf{pw}_{i,m}) \\ & (H(\mathbf{pw}_{j,1}), \mathbf{pw}_{j,1}) \xrightarrow{f} (H(\mathbf{pw}_{j,2}), \mathbf{pw}_{j,2}) \xrightarrow{f} \dots \xrightarrow{f} (H(\mathbf{pw}_{j,m}), \mathbf{pw}_{j,m}) \end{aligned}$$

- Sobald  $\mathbf{pw}_{i,u} = \mathbf{pw}_{j,v}$  (bel.  $u, v$ ) ist, sind Restzeilen gleich
- Viele Überlappungen  $\Rightarrow$  schlechte Abdeckung der Liste
- Andererseits sollte  $f$  nicht „zu zufällig“ sein
- **(Mögliche) Lösung:** benutze verschiedene  $f_u$  ( $u = \text{Spalte}$ )

- **(Mögliche) Lösung:** benutze verschiedene  $f_u$  ( $u = \text{Spalte}$ )

$$(H(\text{pw}_{1,1}), \text{pw}_{1,1}) \xrightarrow{f_1} (H(\text{pw}_{1,2}), \text{pw}_{1,2}) \xrightarrow{f_2} \dots \xrightarrow{f_{m-1}} (H(\text{pw}_{1,m}), \text{pw}_{1,m})$$

$$(H(\text{pw}_{2,1}), \text{pw}_{2,1}) \xrightarrow{f_1} (H(\text{pw}_{2,2}), \text{pw}_{2,2}) \xrightarrow{f_2} \dots \xrightarrow{f_{m-1}} (H(\text{pw}_{2,m}), \text{pw}_{2,m})$$

...

$$(H(\text{pw}_{n,1}), \text{pw}_{n,1}) \xrightarrow{f_1} (H(\text{pw}_{n,2}), \text{pw}_{n,2}) \xrightarrow{f_2} \dots \xrightarrow{f_{m-1}} (H(\text{pw}_{n,m}), \text{pw}_{n,m})$$

- **Effekt:**  $\text{pw}_{i,u} = \text{pw}_{j,v}$  zieht nur bei  $u = v$  Kollision der „Restzeilen“ nach sich  $\Rightarrow$  größere Abdeckung der Liste
- **Namensgebung:**  $f_j$  und  $j$ -te Spalte der Liste haben „Farbe“  $j$
- **Nachteil:** Angriff muss alle Farben von  $H(\text{pw}^*)$  durchprobieren

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- **Angriffe abwehren**
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- Interaktive Nutzerauthentifikation

- Einfache (und seit langem bekannte) Gegenmaßnahme:

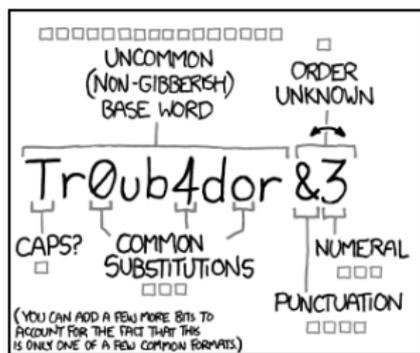
$$U_{pw} \xrightarrow{pw} S_{(\text{salt}, H(\text{salt}, pw))}$$

mit Zufallszahl `salt` „gesalzener“ Hashwert

- „Individualisiert“ `pw`, verhindert Wörterbuchangriffe (einschließlich Angriffen mittels Rainbow Tables)

- Weitere einfache Gegenmaßnahme: Key Strengthening
- **Idee:** erschwere H-Auswertung (für Angreifer *und* Server)
- Beispiel:  $H'(pw) = \underbrace{H(H(\dots H(pw) \dots))}_{1000 \text{ Mal}}$
- Oder:  $H'(pw) = H(pw, i)$  für kleinstes  $i \in \mathbb{N}$ , so dass  $H(pw, i)$  mit 20 Nullen beginnt (benötigt Durchprobieren  $\approx 2^{20}$  vieler  $i$ )
- Verlangsamt Angriff, verlangsamt aber auch Authentifikation
- Benutzt zum Beispiel von Truecrypt

# Bessere Passwortwahl (?)



~28 BITS OF ENTROPY

$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$

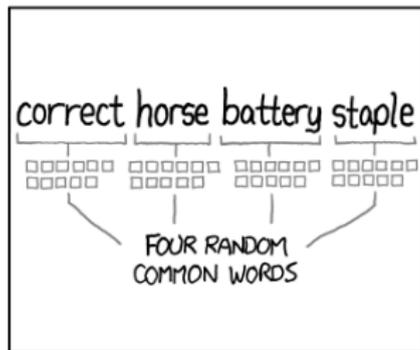
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE O's WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER: **HARD**



~44 BITS OF ENTROPY

$2^{44} = 530 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$

DIFFICULTY TO GUESS: **HARD**

THAT'S A BATTERY STAPLE.

CORRECT!

DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- **Parallelisierte Brute-Force-Angriffe**
- Die Realität
- Interaktive Nutzerauthentifikation

# Parallelisierte Brute-Force-Angriffe

- **Idee:** suche systematisch „plausible“ Passwörter ab
- Lange Zeit praktisch aufwändiger als z.B. Rainbow Tables
- **Aber:** bei geschickter Aufteilung des Passwortraums parallelisierbar
  - Insbesondere interessant für Graphics Processing Units (GPUs)
  - Viele kleine *programmierbare* Einheiten auf GPU, die einfache Aufgaben (z.B. H-Auswertung) übernehmen können
  - Mittlerweile (2013) effizienter als Rainbow Tables
- **Frage:** welche Gegenmaßnahmen (Salt, Key Strengthening, „bessere Passwörter“) helfen hier?

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- **Die Realität**
- Interaktive Nutzerauthentifikation

- Windows (seit 1980er,  $\leq$  ME) verwendet „LM-Hashes“  $H(\text{pw})$

$$U_{\text{pw}} \xrightarrow{\text{pw}} S_{H(\text{pw})}$$

mit DES-basierter Hashfunktion  $H$ :

- $\text{pw}$  muss 14 Zeichen lang sein (ggf. aufgefüllt/abgeschnitten)
- Kleinbuchstaben in  $\text{pw}$  durch Großbuchstaben ersetzt
- $\text{pw}$  wird in 7-Zeichen-Blöcke  $\text{pw}_1, \text{pw}_2$  aufgeteilt, dann:

$$H(\text{pw}) := ( \text{Enc}(\text{pw}_1, \text{KGS!@#\$\%}), \text{Enc}(\text{pw}_2, \text{KGS!@#\$\%}) )$$

- Erster Teil hängt nur von  $\text{pw}_1$  ab, zweiter nur von  $\text{pw}_2$
- **Frage:** Was könnte Angreifer tun, der nur  $H(\text{pw})$  kennt?

- Windows ( $\leq$  XP) unterstützt und speichert noch LM-Hashes
- Ab Windows Vista LM-Hashes standardmäßig deaktiviert
- Neuere Windows-Versionen verwenden NTLM oder Kerberos
  - NTLM auch problematisch (nutzt schwache Verschlüsselung)
- Unix-artige Systeme (OS X, Linux, \*BSD) verwenden seit etwa 1980 gesalzene Hashes:

$$U_{pw} \xrightarrow{pw} S_{(\text{salt}, H(\text{salt}, pw))}$$

- Hashfunktion H variiert (SHA-512, MD5, DES-basiert, ...)
- **Probleme:** frühe Versionen beschränkt auf 8 Zeichen, Probleme mit Nicht-ASCII-pw

## 1 Benutzerauthentifikation

- Motivation
- Szenario
- Wörterbuchangriffe
- Komprimieren des Wörterbuchs
- Rainbow Tables
- Angriffe abwehren
- Parallelisierte Brute-Force-Angriffe
- Die Realität
- **Interaktive Nutzerauthentifikation**

- Für manche Anwendungen (E-Banking) Passwörter nicht sicher genug
  - Kommunikation zwischen U und S unsicher (Internetcafe)
  - Wichtigstes Ziel: Replay-Angriffe vermeiden
- **Möglich:** *interaktive* Authentifikation zwischen U und S
- **Problem:** U allein nicht sehr berechnungsmächtig
- **Möglich:** U nutzt sichere Hardware (Chipkarte/Lesegerät)
  - 1 U  $\xrightarrow{\text{pw/PIN}}$  HW<sub>K,pw</sub>  $\xleftarrow{R}$  S<sub>K</sub>
  - 2 U  $\xrightarrow{\text{HW}_{K,pw}}$  S<sub>K</sub>  $\xrightarrow{\sigma := \text{Sig}(K,R)}$
- Manchmal ohne pw/PIN (Nutzer setzt nur Chipkarte ein)