DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Passwortwiederherstellung

Dozent: Prof. Dr. Michael Eichberg

Kontakt: michael.eichberg@dhbw.de

Version: 2.0

Folien: [HTML] https://delors.github.io/sec-passwortwiederherstellung/folien.de.rst.html
[PDF] https://delors.github.io/sec-passwortwiederherstellung/folien.de.rst.html.pdf

Fehler melden: https://github.com/Delors/delors.github.io/issues

Was ist Passwortwiederherstellung?

Definition

Passwortwiederherstellung ist der Prozess, der dazu dient, ein nicht (mehr) vorhandenes Passwort wiederzuerlangen.

- Wer hat schon einmal Passworte wiederhergestellt?
- Wer hat Erfahrung mit Linux?
- Wer hat Erfahrung mit Linux Kommandozeilenwerkzeugen für die Verarbeitung von Dateien mit Text?
- Wer hat Erfahrung mit regulären Ausdrücken?
- Wer hat Erfahrung mit Python?
- Wer hat Erfahrung mit Java (Reverse Engineering)?

Passwortwiederherstellung - Haftungsausschluss

Wer unbefugt sich oder einem anderen Zugang zu Daten, die nicht für ihn bestimmt und die gegen unberechtigten Zugang besonders gesichert sind, unter Überwindung der Zugangssicherung verschafft, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

—§ 202 a Abs. 1 StGB

Gericht sieht Nutzung von Klartext-Passwörtern als Hacken an

[...] Vor dem Amtsgericht wurde ein Prozess verhandelt, der die Gefahren verdeutlicht, denen sich Menschen mitunter aussetzen, die versuchen, Sicherheitslücken in der Software deutscher Firmen zu finden. Das Amtsgericht hat einen Programmierer verurteilt, der im Auftrag eines Kunden eine Software analysiert und darin eine Sicherheitslücke gefunden hatte, welche die Daten von Einkäufern in Online-Shops im Internet offengelegt hatte. Der Programmierer kontaktierte [...] die betroffene Firma, die daraufhin die Sicherheitslücke schloss und ihn anzeigte. Aufgrund dieses Umstandes wurde der Programmierer nun wegen unbefugten Zugriffs auf fremde Computersysteme und Ausspähens von Daten – welches unter dem sogenannten Hacker-Paragrafen 202a StGB unter Strafe gestellt ist – [...] verurteilt[...].

—Heise.de - 19.01.2024 12:54 Uhr

Wiederherstellung von Passwörtern

Welches Vorgehen müssen wir wählen, um Passworte der folgenden Art wiederherzustellen?

1. Donaudampfschifffahrt
2. Passwort
3. ME01703138541
4. 2wsx3edc4rfv
5. Haus Maus
6. iluvu
7. Emily18
8. MuenchenHamburg2023!!!!
9. password123

!! Wichtig

Es gibt „einfach sichere Passworte“, die in vernünftiger Zeit mit angemessenen Ressourcen nicht wiederhergestellt werden können.

1. Passwortwiederherstellung 101

Passwortwiederherstellung

01 Wissen wo/in welcher Form der Passworthash zu finden ist.

02 Extraktion des Hashes

03 Wiederherstellung des Passwortes durch das systematische Durchprobieren aller Kandidaten.

Beispiel - Wiederherstellung eines Linux Login Passwortes

```
~% sudo cat /etc/shadow
[...]
john:$6$zElzjLsMqi36JXWG$FX2Br1/[...]. ↩
RxAHnNCBsqiouWUz751crHodXxs0iqZfBt9j40l3G0:19425:0:99999:7:::
[...]
```

```
% echo -n '$6$zElzjLsMqi36JXWG$FX2Br1/[...]. ↩
RxAHnNCBsqiouWUz751crHodXxs0iqZfBt9j40l3G0' > hash.txt

% hashcat -m 1800 hash.txt -a 3 '?d?d?d?d?d?d'
```

Finden eines Hashes

Im Falle von Linux Login Passworten ist genau spezifiziert wo die Passworte (`/etc/shadow`) und in welcher Form die Passworte gespeichert werden. Nach dem Namen des Nutzers (im Beispiel `john`) ist der verwendete Hashingalgorithmus vermerkt. Dieser unterscheidet sich zwischen den Distributionen. Aktuell setzen die meisten Distributionen auf `yescrypt`. Danach folgen die Parameter. Insbesondere der Salt.

ID	Mode
5	Sha256crypt (veraltet)
6	SHA512crypt (in Ablösung)
y (or 7)	yescrypt

Systematisches Testen aller Kandidaten

Konzeptionell führt die Software Hashcat die folgenden Schritte durch:

```
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000000") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000001") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000002") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000003") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000004") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000005") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000006") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000007") ✘
<extracted_hash> =? SHA512crypt("zElzjLsMqi36JXWG","000008") ✘
...
<extracted_hash> == SHA512crypt("zElzjLsMqi36JXWG","123456") ✔
```

Der folgende Code könnte als Grundlage genutzt werden, um das Passwort wiederherzustellen.

(Linux nutzt standardmäßig 5000 Runden.)

```python
from passlib.hash import sha512_crypt

sha512_crypt.hash(
    "123456",
    salt="zElzjLsMqi36JXWG",
    rounds=5000)
```

2. Werkzeuge und Methoden zur Wiederherstellung von Passwörtern

Passwortwiederherstellung

initiale Quellen für Passwortkandidaten

- Wörterbücher
- Verzeichnisse (z. B. Postleitzahlen, Städte, Straßennamen)
- Leaks (z. B. Rockyou, LinkedIn, Sony)

Werkzeuge zum Generieren von weiteren Passwortkandidaten

- Prince
- OMEN (Markov-Modelle
- Hashcat
- PCFG

Werkzeuge, um Passworthashes zu extrahieren bwz. anzugreifen

- John (the Ripper)
- Hashcat

Grundlegende Werkzeuge

- Linux Shell
- Reguläre Ausdrücke

Extraktion von Hashes

Hinweis

Im Folgenden diskutieren wir nur exemplarisch die Extraktion einiger Hashes, um das grundlegende Vorgehen zu besprechen. Im Allgemeinen gibt es für weit(er) verbreitete Software häufig bereits Lösungen zur Hashextraktion. Falls nicht, dann muss man Googeln und/oder Reverse Engineering betreiben.

Quellen für Werkzeuge & Anleitungen

- Hashcat Tools
- John (the Ripper)
- Googeln
- KI Assistent (z. B.) ChatGPT

Konversation mit ChatGPT (23. August 2024)

Disclaimer

Im Folgenden wurde Anpassungen der Formatierung vorgenommen, zwecks Integration in den Foliensatz. Die Inhalte sind jedoch vollständig von ChatGPT erstellt.

Prompt

Wie extrahiere ich den Hash eines Passworts eines Nutzers unter Ubuntu Linux?

Antwort

Um den Hash eines Passworts eines Nutzers unter Ubuntu Linux zu extrahieren, müssen Sie Zugriff auf die Datei /etc/shadow haben, da diese Datei die gehashten Passwörter enthält. Folgende Schritte sind notwendig:

1. Root-Rechte erlangen:

   Da die Datei /etc/shadow nur von root gelesen werden kann, müssen Sie entweder als root angemeldet sein oder sudo verwenden.

2. Öffnen der /etc/shadow Datei:

   Sie können die Datei mit einem Texteditor oder einem Kommandozeilen-Tool wie cat, less, oder grep öffnen.

   Beispiel mit cat:

   ```
   sudo cat /etc/shadow
   ```

3. Suchen des entsprechenden Benutzers:

   Suchen Sie nach der Zeile, die mit dem Benutzernamen beginnt, dessen Passwort-Hash Sie extrahieren möchten. Diese Zeile hat das folgende Format:

   ```
   benutzername:$id$salt$hash:weitere_felder
   ```

   Der Hash des Passworts ist der Teil nach dem zweiten $ Zeichen. Hier ein Beispiel für eine Zeile aus der /etc/shadow Datei:

   ```
   username:$6$salt$hashed_password:date:0:99999:7:::
   ```

   In diesem Fall ist hashed_password der Passwort-Hash des Benutzers.

4. Extraktion des Passwort-Hashes:

Wenn Sie nur den Hash extrahieren möchten, können Sie den Befehl grep zusammen mit cut verwenden:

```
sudo grep 'username' /etc/shadow | cut -d: -f2
```

Dieser Befehl extrahiert den gesamten Hash einschließlich des ID- und Salt-Teils, also z.B.:

```
$6$salt$hashed_password
```

Falls Sie nur den Hash ohne den ID- und Salt-Teil [...]

Hinweis

Der Zugriff auf und das Bearbeiten von /etc/shadow sollte sehr vorsichtig und verantwortungsbewusst durchgeführt werden, da unsachgemäße Änderungen zu Sicherheitsproblemen oder sogar zum Verlust des Zugangs zum System führen können.

Relevante Linux Kommandozeilenwerkzeuge

- `file` dient der Ermittlung des Typs einer Datei.
- `binwalk` durchsucht Binärdateien in Hinblick auf das Vorkommen bekannter Muster (insbsondere Dateiheader, aber auch Kryptokonstanten etc.)

  `-E` kann zur Visualisierung der Entropie verwendet werden.
- `dd` kopiert Daten blockweise von einem Startpunkt in einer Datei in eine andere Datei. Wird ggf. zum Extrahieren von Hashes benötigt.
- `xxd` und `hexdump` erstellen beide einen Hexdump einer Datei.

Verschlüsselte PDF Dateien

Extraktion erfolgt (zum Beispiel) mit den John Tools:

```
$ pdf2john Document.pdf > Document.pdf.john
$ cat Document.pdf.john
Document.pdf:$pdf$4*4*128*-3392*1*16*861da8b9c1672ddc3953dee025
5d622d*32*301d0810078c5698ab17b286e21230700000000000000000000000
00000000000*32*c038ddb8fbdaeb67b6e80e2d936108fc851ff40c5b652c71
97bda4f797939532
```

Danach kann der Hash entweder direkt mit John angegriffen werden, oder nach dem Entfernen des Headers mit Hashcat.

```
$ pdf2john Document.pdf \
  | sed -E "s/^[^:]+://"        # Dateiname entfernen
  > Document.pdf.hashcat
```

Libreoffice Dateien

Extraktion des Basishashes erfolgt auch hier (zum Beispiel) mit den John Tools. Danach muss sowohl der Prefix als auch der Suffix, der für die Entschlüsselung nicht relevant ist, abgeschnitten werden, wenn im Folgenden Hashcat verwendet werden soll.

```
$ libreoffice2john Document.odt
  | sed -E -e 's/[^:]+://' -e 's/:::::[^:]+$//'
  > Document.odt.hashcat
```

Um zu verstehen, wie der Hash genau auszusehen hat, ist es im Allgemeinen hilfreich sich die erwartete Struktur für einen Hash anzusehen: Hashcat - Example Hashes

Verschlüsselte Mac Disk Images (.dmg)

In diesem Fall hat nur John (the Ripper) Unterstützung für den konkreten Hash.

```
$ dmg2john Container.dmg > Container.dmg.john   # Extraktion

$ john Container.dmg.john \                     # Angriff
  --wordlist=/usr/share/wordlists/rockyou.txt
```

Verschlüsselter USB Stick (APFS Volume)

Es liegt ein normaler USB Stick vor auf dem eine Partition vom Typ `Apple APFS` ist.

```
Disk /dev/sda: 14.45 GiB, 15518924800 bytes, 30310400 sectors
Disk model: Flash Disk
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 1D63D8AE-7CBC-47BE-9093-8469B0786EAF

Device      Start      End  Sectors  Size Type
/dev/sda1      40   409639   409600  200M EFI System
/dev/sda2  409640 30310359 29900720 14.3G Apple APFS
```

Verschlüsselter USB Stick (APFS Volume)

1. Installation von apfs2hashcat (umfasst das Kompilieren der Sourcen)
2. Hash extrahieren durch „Copy-and-Paste“ aus dem Logfile/der Konsole.

```
$ sudo ./apfs-dump-quick \
  /dev/sda2 \   # /dev/sda2 ist die Ziel APFS Partition
  /tmp/log.txt
```

3. Hash angreifen

```
$ hashcat -m 18300 fv2.hashcat \
  /usr/share/wordlists/rockyou.txt
```

Hashcat - Einführung

Hashcat ist – Stand 2024 – das Tool zum Wiederherstellen von Passwörtern.

Liest ein(e Liste von) Hash(es) ein und prüft, ob einer der angegebenen Passwortkandidaten nach dem Hashen mit einem gegeben Hash übereinstimmt.

- unterstützt über 350 Hash-Typen (mit einigen automatischen Erkennungen)
- unterstützt mehrere Angriffsmodi, z. B.,
  - Wörterbuch (ggf. mit Regeln)
  - Masken
  - Kombinationen aus Wörterbüchern und Masken
  - <Lesen von Passwortkandidaten aus stdin>
- Open-Source
- Kann zum Generieren von neuen Kandidaten verwendet werden.
- ist CUDA/OpenCL basiert und **auf entsprechenden Grafikkarten extrem schnell**.

Hashcat - relevante Parameter

Angriffsmodi:

```
-a0 Angriff mit Wörterbuch
    (ggf. mit Regeln -r)

-a1 Kombinationsangriff
    Angriff mit dem Kreuzprodukt
    zweier Wörterbücher.

-a3 Brute-force Angriff

-a6 Hybridangriff
    Wörterbuch und Maske
```

Brute-force - Eingebaute Zeichensätze:

```
?l = abcdefghijklmnopqrstuvxyz

?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ

?d = 0123456789

?s = !"$%&'()*+,-./:;<=>?@[]^_{|}~

?a = ?l?u?d?s
```

Definition von bis zu 4 eigenen Zeichensätzen (?1,...,?4) ist möglich.

Hashcat - Ausgewählte Regeln

(Die Regeln sind teilweise John kompatibel.)

Name	Function	Description	Input	Output
Nothing	:	Do nothing (passthrough)	`p@ssW0rd`	`p@ssW0rd`
Lowercase	l	Lowercase all letters	`p@ssW0rd`	`p@ssw0rd`
Uppercase	u	Uppercase all letters	`p@ssW0rd`	`P@SSW0RD`
Capitalize	c	Capitalize the first letter and lower the rest	`p@ssW0rd`	`P@ssw0rd`
Toggle Case	t	Toggle the case of all characters in word.	`p@ssW0rd`	`P@SSw0RD`
Reverse	r	Reverse the entire word	`p@ssW0rd`	`dr0Wss@p`
Duplicate	d	Duplicate entire word	`p@ssW0rd`	`p@ssW0rdp@ssW0rd`
Append	$X	Append X to the end	`p@ssW0rd`	`p@ssW0rdX`
Prepend	^X	Prepend X at the beginning	`p@ssW0rd`	`Xp@ssW0rd`
...	...	...	...	...

Szenario 1: eine Pin Angreifen

Ausgangssituation

Gegeben sein ein mit SHA256 gehashter 5-stelliger Pin in der Datei: `5_digits_pin.sha256`.

Hashwert:

```
79737ac46dad121166483e084a0727e5d6769fb47fa9b0b627eba4107e696078
```

Angriff mit Maske

```
hashcat -m 1400 5_digits_pin.sha256 -a3 "?d?d?d?d?d"
```

-m 1400:	Modus für einen einfachen SHA256 Hash.
-a3:	bezeichnet einen Maskenangriffe
"?d?d?d?d?d":	Beschreibt die Maske. Hier 5 Ziffern ( *digits*).

Szenario 2: Ein (hoffentlich) einfaches Loginpasswort angreifen

Ausgangssituation

Ein mit SHA512crypt gehashtes Passwort in der Datei: `password.sha512crypt`.

Angriff mit Wörterbuch

```
hashcat password.sha512crypt -a0 /usr/share/wordlists/rockyou.txt
```

-a0: bezeichnet einen Wörterbuchangriff.

/usr/share/wordlists/rockyou.txt:
Das zum Angriff verwendete Wörterbuch; der Pfad ist der Standardpfad zum Rockyou Wörterbuch in Kali Linux.

Szenario 3: ein komplexeres Passwort angreifen

Ausgangssituation

Ein mit MD5 gehashtes Passwort in der Datei: `password.md5`. Ein erster Angriff mit Rockyou war nicht erfolgreich.

Angriff mit Wörterbuch und Regelsatz

```
hashcat -m 0 password.md5 \
           -a0 /usr/share/wordlists/rockyou.txt \
           -r /usr/share/hashcat/rules/best64.rule
```

-a0: bezeichnet einen Wörterbuchangriff.

/usr/share/wordlists/rockyou.txt:
Das zum Angriff verwendete Wörterbuch.

-r /usr/share/hashcat/rules/best64.rule:
Der zum Beugen der Passwortkandidaten verwendete Regelsatz. Der Regelsatz best64 hat sich in einem Wettbewerb als „bester“ Regelsatz erwiesen.

Szenario 4: ein Passwort mit Salt angreifen

Ausgangssituation

Ein MD5 Hash ist gegeben: `c84b5c34c9ff7d3431018d795b5975e5`. Weiterhin ist bekannt, dass der verwendete *Salt* `SALT` ist.

Angriff

1. Modus für MD5+Salt heraussuchen (`-m10`); ggf. Beispielhash ansehen, um zu verstehen, wie der Hash aufgebaut ist.
2. Erzeugen des Hashes für Hashcat:

   ```
   echo -n "c84b5c34c9ff7d3431018d795b5975e5:SALT" > salted.m5.hash
   ```
3. Mit Hashcat angreifen:

   ```
   hashcat -m10 salted.md5.hash  -a3 '?a?a?a?a'
   ```

Lösung

Das Passwort ist `Test`. In diesem Fall wäre es auch möglich gewesen direkt zu Prüfen ob das Passwort `Test` ist, indem man Hashcat im Modus -m0 (für reinen MD5) startet und als Kandidaten `TestSALT` vorgibt.

Szenario 5: Kombination von Wörterbuch mit eigenem Regelsatz

Ausgangssituation

Wir greifen einen sogennanten langsamen Hash an und können deswegen nur wenige Passworte gezielt testen.

Aufgrund von Social Engineering/Ermittlungen wissen wir, dass die Person häufig kurze Worte (max 4 Buchstaben nimmt) diese aber oft verdoppelt und häufig die Worte mit einem Großbuchstaben anfangen lässt.

Angriff

1. Erstellen eines fokussierten Wörterbuchs: `candidates.txt`.
2. Erstellen des Regelsatzes: `case.rule`.
3. Angriff mit den erstellten Wörterbuch und dem Regelsatz.

Szenario 5: Kombination von Wörterbuch mit eigenem Regelsatz

Angriff

1. Generierung von `candidates.txt`

   Um sicherzustellen, dass wir keine Duplikate testen, wandeln wir alle Worte in Kleinschreibung um und filtern entsprechende Duplikate. Die Beachtung aller Varianten in Hinblick auf die Groß- und Kleinschreibung wird durch die Regeln sichergestellt.

```
$ grep -Po "^[a-zA-Z]{3,4}(?=[^a-zA-Z])" \
       /usr/share/wordlists/rockyou.txt \
  | tr [:upper:] [:lower:] \
  | sort -u \
  > candidates.txt
```

Zu Bedenken

Die gezeigte Operation löst die Ordnung in der Datei auf und sortiert diese alphabetisch. Dies ist aber häufig nicht gewünscht – insbesondere wenn der Leak nach Verwendungshäufigkeit sortiert ist!

Szenario 5: Kombination von Wörterbuch mit eigenem Regelsatz

Angriff

1. Erstellen des Regelsatzes: `case.rule`

   Um sicherzugehen, dass wir alle Varianten abdecken, brauchen wir drei Regeln.

cd	Erst Groß-Kleinschreibung anpassen und dann duplizieren.
dc	Erst duplizieren und dann Groß-Kleinschreibung anpassen.
d	Einfach nur duplizieren.

2. Angriff mittels Hahcat

```
hashcat -m 1700 hash.sha125 candidates.txt -r case.rule
```

Tips

Das beherrschen von regulären Ausdrücken ist bei der Passwortrekonstruktion sehr hilfreich.

Der folgende Ausdruck liefert zum Beispiel alle 4stelligen Worte aus Rockyou mit Hilfe eines Lookheads, dass längere Worte filtert.

```
$ grep -Po "^[a-zA-Z]{3,4}(?=[^a-zA-Z])" \
     /usr/share/wordlists/rockyou.txt
```

Das Passwort `TreeTree` würde sich damit erfolgreich wiederherstellen lassen.

Szenario 6: Kartesiche Produkt von zwei Wörterbüchern

Ausgangssituation

Aufgrund von Social Engineering/Ermittlungen wissen wir, dass die Person sehr gerne zwischen deutschen Großstädten pendelt. Nachdem andere Versuche nicht zum Erfolge geführt habe, wollen wir jetzt Passworte der Art: "BerlinHamburg" testen.

Angriff

1. Erstellen eines fokussierten Wörterbuchs durch *Googeln* von großen Städten.
2. Angriff durch Kombination des Wörterbuchs mit sich selbst.

```
$ hashcat -m 1400 hash.sha256 -a 1 big_cities2.txt big_cities2.txt
```

Szenario 7: Wörterbuch mit Maske

Ausgangssituation

Es ist bekannt, dass die Passwörter der Gruppierung häufig mit vier Zahlen und zwei Sonderzeichen aus einer sehr kleinen Mengen von Sonderzeichen (`$!.`) enden. Davor kommt ein Wort mit ca. 4-8 Stellen in den typischerweiser "liebe/love/luv" vorkommt.

Angriff

1. Erstellen eines fokussierten Wörterbuchs: `candidates.txt`
2. Angriff mit passendem Maskenangriff

Szenario 7: Wörterbuch mit Maske

Angriff mit Hybridangriff

`candidates.txt` enthält alle Begriffe aus rockyou, die die Anforderung erfüllen:

```
$ grep -oE "[a-zA-Z]*[Ll]((uv)|(ove)|(iebe))[a-zA-Z]*" \
      /usr/share/wordlists/rockyou.txt \
 | sort -u \
 > candidates.txt
```

Angriff mit Hashcat:

```
$ hashcat -m 1400 hash.sha256 candidates.txt \
    -a 6 -1 '$.!' '?d?d?d?d?1?1'
```

Beispiel

In diesem Falle verwenden wir einen Hybridangriff, der eine Wordliste mit einer Maske kombiniert. Hier definieren wir unseren eigenen „Zeichensatz“ mit dem Parameter `-1 '$.!'` und referenzieren diesen in unserer Maske später mit `?1`.

Ein Beispielpasswort, dass wir mit dem Ansatz ermitteln könnten, wäre:

SHA256	Passwort
b9cace43df57bc694498bf4d7434f45a8466c4a924f608d54fd279d24b3dc937	ILuvU2023!!

Szenario 8: Passwörter mit Muster

Ausgangssituation

Wir möchten ein Wörterbuch erstellen mit „Wörtern“, die Buchstabenvervielfältigungen enthalten, aber nicht länger als 16 Zeichen sind. Zum Beispiel: `aaaaBBBBcccc` oder auch `AFFFFFE`. Weiterhin soll die Liste nach der Länge der gefundenen Einträge aufsteigend sortiert sein und Zeichen, die keine Buchstaben sind, einfach gelöscht werden.

Lösung

Heraussuchen entsprechender Wörter aus rockyou mittels Linux Kommandozeilenwerkzeugen.

```
$ grep -E "([a-zA-Z])\1{3,}" /usr/share/wordlists/rockyou.txt
  | grep -E "^.{4,16}$"
  | sed -E 's/[^a-zAZ]//g'
  | sort -u
  | awk '{print length " " $1}'
  | sort -n
  | sed -E 's/^[0-9]+ //'
```

Alternative Aufgabenstellung

Sortierung der finalen Liste nach nach der Häufigkeit der Muster, angefangen mit dem häufigsten Mustern.

Szenario 9: Passwörter bestehend aus Fragmenten

Ausgangssituation

- Einer gegebenen Liste können wir nur entnehmen, dass alle Passwörter zusammengesetzt sind aus den Fragmenten: `ab`, `mem`, `1i` und `xy`.
- Darüber hinaus ist immer eine Zahl vorangestellt und am Ende kommt ein Punkt (`.`) oder ein Ausrufezeichen (`!`).
- Die Länge scheint zwischen 6 und 16 Zeichen zu sein und Fragmente können sich wiederholen.

Beispiel: `1ab1ixyxy`.

Vorgehen

1. Erstellen eines Basiswörterbuchs (`base.txt`) mit den Fragmenten als Einträge.
2. Erstellen von Regeln für das Voranstellen und Anhängen der entsprechenden (Sonder)zeichen.
3. Aus Basiswörterbuch das finale Wörterbuch für den Angriff generieren.
4. Mit dem finalen Wörterbuch und entsprechenden Regeln angreifen.

Szenario 9: Generierung von Wörterbüchern aus Fragmenten

Lösung

Zu Generierung aller Kombinationen aus den Fragmenten verwenden wir den Princeprocessor. Der Princeprocessor ist sehr schnell und ermöglicht es in Fällen die Ausgabe direkt an Hashcat durchzureichen und das Zwischenwörterbuch nicht explizit speichern zu müssen.

Angriff

```
$ princeprocessor --pw-min=6 --pw-max=16 base.txt \
  | hashcat -m 1400 hash.sha256 \
     -r number_prepend.rule \
     -r sc_append.rule
```

Aufbau von `number_prepend.rule`:

```
^0
^1
...
^9
```

Aufbau von `sc_append.rule`:

```
$.
$!
```

Mit dem obigen Ansatz könnte zum Beispiel das folgende Passwort ermittelt werden:

SHA256	Passwort
8b11f8e8d487266a791d6d723a3e380c 38f49679735a7f3395ace4302e83dd0e	8abxylixy.

In diesem Falle wäre es auch möglich gewesen nur einen Regelsatz zu erstellen mit den passenden Regeln (zum Beispiel: `^1$.`, `^1$!`, ...) der Aufwand wäre hier jedoch höher gewesen und hätte keinen Nutzen gehabt.

Im Allgemeinen ist jedoch bei der Verwendung des Kreuzproduktes von Regeln immer darauf zu achten, dass keine (oder zumindest keine relevante Anzahl von) Regeln dupliziert werden. Ein Beispiel wäre das Kreuprodukt aus einem Regelnsatz für das optionale Anhängen einer Ziffer mit sich selbst. Sei der Regelsatz:

```
:
$1
$2
```

und würde man diesen mit sich selber kombinieren, um alle Fälle des Anhängens von keiner, einer bzw. zwei Zahlen abzudecken, dann würden folgende Regeln entstehen:

```
::
:$1
:$2
$1$1
$1$2
$2$1
```

```
$2$2
$1:
$2:
```

Wie zu erkennen ist, führen zum Beispiel die Regeln `$1:` und `:$1` jeweils zum gleichen Ergebnis und wären deswegen nicht effektiv.

Szenario 10: Hashcat als Werkzeug zur Wörterbuchgenerierung

Ausgangssituation Gegeben sein 3 Wörterbücher [1]: `base1.txt`, `base2.txt` und `base3.txt`. Gesucht ist ein Wörterbuch, dass alle Kombinationen aus den drei Wörterbüchern enthält und bei dem alle Teilworte immer mit Sonderzeichen (-) voneinander getrennt sind.

Beispiel Sei `base1.txt`: *Kuh*, *Schwein*; `base2.txt`: *Haus*, *Villa* und `base3.txt`: *Baum*, *Busch*. Dann wäre das gesuchte Wörterbuch: *Kuh-Haus-Baum*, *Kuh-Haus-Busch*, ..., *Schwein-Villa-Busch*.

Vorgehen

1. Erzeugen des Kreuzprodukts der ersten beiden Wörterbücher.

```
$ hashcat --stdout base1.txt base2.txt -j '$-' > base1-base2.txt
```

1. Erzeugen des finalen Wörterbuchs durch Bildung des Kreugprodukts der Ergebnisse aus Schritt 1 mit dem dritten Wörterbuch.

```
$ hashcat --stdout base1-base2.txt base3.txt -j '$-' > final.txt
```

Die Hashcat Utilities Bibliothek hat auch noch weitere Werkzeug zum Kombinieren von Wörterbüchern, die viele Fälle sehr effizient abdecken (auch den besprochenen). Jedoch ist es gerade in Fällen, in denen komplexere Regeln zur Anwendung kommen sollen, häufiger sinnvoller/nowendig direkt Hashcat im "stdout" Modus zu verwenden, um die Zwischenwörterbücher zu generieren.

---

[1] Die selbe Vorgehensweise lässt sich auch anwenden, wenn man ein Wörterbuch mit sich selber kombinieren möchte.

Passwörter angreifen - Zusammenfassung

- Passwörter können vielfach effizient angegriffen werden.
- (gute bis exzellente) Kenntnisse über die Zielpersonen sind häufig notwendig.
- Viele Werkzeuge sind verfügbar (siehe auch Hashcat Werkzeuge, Princeprocessor, John the Ripper, etc.)
- Kleine etablierte Kommandozeilenwerkzeuge (`tr`, `greb`, `sed`, `awk`, ...) oder selbstentwickelte Werkzeuge (zum Beispiel in Python) sind häufig ergänzend notwendig und führen oft schneller zum Ziel als die Suche nach *dem* Tool.
- Insbesondere wenn es um die semantische Anreicherung von Wörterbüchern geht, dann sind (bisher) keine etablierten Werkzeuge vorhanden.
- Häufig führen nur Kombinationen von etablierten und eigenen Werkzeugen zum gewünschten Ziel.

Übung

Wörterbuch basierte Wiederherstellung eines Passworts

2.1. MD5 Hash eines einfachen Passworts

`7c6a180b36896a0a8c02787eeafb0e4c`

Hinweise: Das Passwort besteht aus Buchstaben gefolgt von Ziffern und ist sehr häufig.

Sie können Hashcat (https://hashcat.net/hashcat/) verwenden oder ein Bash-Skript schreiben oder eine kleine Lösung in einer Programmiersprache Ihrer Wahl entwickeln. Verwenden Sie ggf. eine Liste bekannter Passwörter (z. B. Rockyou).