Passwortsicherheit allgemein: + SSL (((X.509 Zertifikat) digitale signatur sz: authenzität, integrität, verschlüssl sz: vertrauen

Geben Sie den folgenden Befehl auf der *vmLF1* ein, um die Nachrichten vom Intranet zum Server in der DMZ mitzulesen:

```
tcpdump -i green0  \
        tcp        \
    and net  192.168.210.0/24  \
    and host 192.168.220.12    \
        -A -l | perl -ne 'print if /\d{20}\s+\w+/'
```

```
echo '12345678901234567890 ping' | nc 192.168.220.12 7777
```

Kombinationsmöglichkeiten von Zeichen

Jedes Passwort kann grundsätzlich über diese zwei Merkmale beschrieben werden:

- **Zeichensatz**: Ein Passwort wählt einige Zeichen aus einer vorgegebenen Liste aus. Beispiel: die Zeichen «qwert» werden aus dem Alphabet von Kleinbuchstaben als Passwort ausgewählt.
- **Länge**: Jedes Passwort hat eine Länge. Beispiel: Das Passwort «qwert» hat die Länge fünf.

Diese beiden Merkmale sind jeweils nicht geheim und damit grundsätzlich auch einem Angreifer bekannt. Obwohl ein Angreifer das Passwort eines Einzelnen nicht kennt, kann er bereits einige Aussagen treffen.

Ein einfaches Beispiel:

- **Zeichensatz**: Alphabet A-Z (Gross- & Kleinschreibung) sowie Ziffern 0-9.
- **Länge**: 8 bis 12 Zeichen.

Ein Angreifer kann nun bereits die Anzahl aller theoretisch möglicher Passwörter berechnen. Wenn die Grösse des Zeichensatzes (Anzahl Zeichen) z ist und die Länge des Passwortes l, dann lautet die Formel für die Anzahl möglicher Kombinationen k:

$$k = z^l$$

Wortkombinationen

Passwörter bestehen oft aus zusammengesetzten Wörtern statt zufällig gewählten Zeichen. Solche Passwörter verlieren aus verschiedenen Gründen schnell Ihre Komplexität und sind damit einfacher knackbar.

Ein Beispiel:

Ein Hacker weiss, dass der Benutzer Passwörter aus jeweils drei Wörtern bildet. Jedes Wort wird am Anfang gross geschrieben. (Der Benutzer bildet sich ein, das Passwort so sicherer zu machen, da auch Grossbuchstaben verwendet werden.) Hier drei Beispiel nach diesem Schema:

RegenschirmApfelBerg, SahneMilchButter, DerDieDas

Merkmale: Alphabet A-Z Gross- & Kleinschreibung. 9 - 20 Zeichen.

Nach vorherigem Vorgehen wären dies 52 Zeichen insgesamt. Dies gäbe folgende Rechung:

$(2x26)^{20} - (2x26)^9 = 52^{20} - 52^9 = 2.09x10^{34}$ = *knapp 21 Quintilliarden.*[7] *Eine Zahl mit 34 Nullen!*

Also ziemlich sichere Passwörter? Nicht wirklich!

Sobald der Angreifer das Wissen über den Aufbau hat, kann er die Rechnung vereinfachen. Da die Passwörter aus Wörtern statt Buchstaben aufgebaut werden, sind Wörter die kleinste Einheit. Damit hat jedes Passwort nur eine Länge von drei! Der Zeichensatz wird dafür grösser, da es viel mehr Wörter gibt als Buchstaben im Alphabet. Der zentrale Wortschatz der Deutschen Sprache besteht gemäss Duden aus 70'000 Wörtern. Dies ergäbe folgende Anzahl möglicher Passwörter:

$70000^3 = 3.43x10^{14}$ = *343 Billionen. Eine Zahl mit 14 Nullen, also bedeutend weniger!*

Wie Sie sehen sind diese Passwörter unsicher, obwohl bis zu 20 Buchstaben verwendet werden!

? Erklärung

Was passiert? Die Algorithmen «zerhacken» die Passwörter («geheim» & «sicher») in eine zufällig wirkende Zeichenkette. Solange die gleiche Eingabe «zerhackt» wird, kommt immer das gleiche Ergebnis als Hash heraus. Für jede andere Eingabe soll aber (möglichst) immer ein anderer Hash heraus kommen. Daher lassen sich statt der Passwörter, die Hashes der Passwörter vergleichen. Es genügt also den Hash des Passwortes in der Datenbank zu speichern, statt das Passwort selbst. Vorteil: Wenn jemand die Passwort-DB stiehlt, bekommt er nur Hashes und keine Passwörter.

in irgend einer Form für die Authentifizierung vorhanden sein. Es gibt hier also einen Widerspruch:

- Sicherheit: Das Passwort darf nicht dauerhaft gespeichert werden.
- Praxis: Das Passwort muss dauerhaft gespeichert werden, damit man beim Login das eingegebene Passwort vergleichen kann.

Hashcat Commands:

Block Chiffre Mode (bzw. Zufälligkeit des Initialisierungsvektors)

Hier ist ein beispielhafter Aufruf von `hashcat`:

```
hashcat -m 0 -a 3 geheim.txt ?l?l?l?l --force
```

- `-m 0`: Der Hash-Typ wird auf `md5` eingestellt.
- `-a 3`: Der Bruteforce-Modus wird mit diesem Parameter eingestellt.
- `geheim.txt`: Dateiname der Datei mit den Hashwerten.
- `?l?l?l?l`: Maske/Muster für die zu prüfenden Wertebereiche. Ein Fragezeichen steht für eine Stelle im Passwort, der Buchstabe für den Zeichentyp. Diese Maske bedeutet: «Teste alle Passwörter mit der Länge vier (4 mal `?`) und kleinen Buchstaben (`l`). Sie finden eine detailliertere Beschreibung dazu auf der Webseite des Programms.»[7]
- `--force`: Führt den Angriff auch dann aus, wenn die Treiber nicht optimal konfiguriert sind.

```
- [ Attack Modes ] -

  #  | Mode
 ====+======
  0  | Straight
  1  | Combination
  3  | Brute-force
  6  | Hybrid Wordlist + Mask
  7  | Hybrid Mask + Wordlist

- [ Built-in Charsets ] -

  ?  | Charset
 ====+=========
  l  | abcdefghijklmnopqrstuvwxyz
  u  | ABCDEFGHIJKLMNOPQRSTUVWXYZ
  d  | 0123456789
  h  | 0123456789abcdef
  H  | 0123456789ABCDEF
  s  |  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
  a  | ?l?u?d?s
  b  | 0x00 - 0xff
```

-a 0 = Wörterbuch

```
- [ Basic Examples ] -

  Attack-          | Hash- |
  Mode             | Type  | Example command
 ==================+=======+==================================================================
  Wordlist         | $P$   | hashcat -a 0 -m 400 example400.hash example.dict
  Wordlist + Rules | MD5   | hashcat -a 0 -m 0 example0.hash example.dict -r rules/best64.rule
  Brute-Force      | MD5   | hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a?a
  Combinator       | MD5   | hashcat -a 1 -m 0 example0.hash example.dict example.dict
```

Einbinden in die Applikation

Der Fulla-Server ist so programmiert, dass der Wechsel auf SSL/TLS sehr einfach ist. Sie müssen dem Verbindungs-Objekt (*socket*) einfach die Dateipfade zu den Zertifikaten mitteilen!

Dem Server müssen Sie den öffentlichen und den privaten Schlüssel angeben:

```
use IO::Socket::SSL 'inet4';

my $socket = IO::Socket::SSL->new (
    LocalAddr     => '0.0.0.0',     # local server address
    LocalPort     => '7777',        # local server port
    Listen        => 5,             # queue size for connections
    Proto         => 'tcp',         # protocol used
    SSL_cert_file => 'cert.pem',    # SSL certificate
    SSL_key_file  => 'key.pem',     # SSL certificate key
);
```

```
use IO::Socket::SSL 'inet4';

my $socket = IO::Socket::SSL->new (
    PeerHost     => '192.168.220.12',
    PeerPort     => '7777',
    Proto        => 'tcp',

    SSL_ca_file => .............
);
```

Client stellt eine zweite Anfrage und nutzt die Session-ID:

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
```

? Erklärung

Was ist passiert? Der Angreifer konnte sich auf dem System einloggen und beliebige Abfragen tätigen. Er konnte sich sogar selbst einen eigenen Benutzer anlegen. In Zukunft wird er also seinen eigenen Nutzer verwenden können.

Wichtig: Für diese Attacke musste der Angreifer keine Passwörter kennen oder knacken!

Hinweis

Um solche Attacken zu verhindern (bzw. zu erschweren), sollte sämtliche Kommunikation zwischen Client und Server verschlüsselt werden. Damit können Router nicht mehr den Inhalt der Nachrichten auslesen und die Session-ID bleibt geheim.

Hinweis

Die Implementation der Session-ID im Fulla-Server ist sehr einfach. Wie sieht das in der Industrie aus? Wie sind Cookies in HTTP umgesetzt? Hinweise dazu auf der letzten Seite.

Implementieren Sie SSL/TLS im Server und Client. Testen Sie die Verbindung. Beobach Sie die Daten-Pakete auf dem Router mit dem folgenden Befehl:

```
tcpdump -i green0 tcp and net 192.168.210.0/24 and host 192.168.220.12 -A
```

Was beobachten Sie nun auf dem Router? Was bedeutet das für jemanden, der Zugriff auf

Mit dem folgenden Befehl können Sie selbst einen öffentlichen (hier `cert.pem`) und einen privaten Schlüssel (`key.pem`) generieren:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

Einbinden in den Browser

Es gibt verschiedene Standards für das Speichern von öffentlichen und privaten Schlüsseln. Um unseren Schlüssel im Browser nutzen zu können, müssen wir diesen in ein anderes Format umwandeln. Umwandeln des Public-Keys in ein Web-Zertifikat (hier `modul183.p12`):

```
openssl pkcs12 -export -in cert.pem -inkey key.pem -out modul183.p12
```

Client eröffnet eine erste Anfrage an Server:

```
GET /index.html HTTP/1.1
Host: www.example.org
```

Server antwortet und setzt eine Session-ID:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

Lösung: Salt

Damit Wörterbücher nicht mehr funktionieren, werden die Passwörter vor dem Hashen und Speichern noch etwas «gesalzen». Das heisst: Statt das eigentliche Passwort zu hashen, wird dem Passwort ein möglichst zufälliger String angehängt und dies dann gehasht.

Ein Beispiel mit dem Passwort `4lf2000` und dem Salz `z4Tms01lQz`:

```
ohne Salz:            4lf2000           -> b9177c2058a7a07c41086609bd07005c

mit Salz z4Tms01lQz:  4lf2000z4Tms01lQz -> f2b7cb8144fb1adcfdbd949799293e1f
```

Wenn das Passwort des 18 jährigen Alfred ohne Salz gehasht wurde, hat er Pech gehabt: Das Wörterbuch enthält seinen Hash `b9177c...` und somit ist sein Passwort geknackt.

Der Hash `f2b7cb...` wird aber nicht im Wörterbuch auftauchen! Denn wie soll ein Hacker auch schon vorher wissen, dass ein komplettes Wörterbuch mit dem Salz `z4Tms01lQz` erstellt werden soll? Wenn nun jeder Benutzer ein anderes Salz hat, wird der Aufwand zu gross, um die Passwörter eines Systems zu knacken.

Hinweis

Wenn Sie Ihre Passwörter mit einem Salt absichern, müssen Sie unter Umständen die Struktur der Datenbank abändern! Denn neben dem Hash muss nun auch das Salt in der Datenbank abgelegt werden.

Wenn jemand sich einloggen möchte, benötigen Sie das Salt, um den Hash zu berechnen um so das Passwort zu verifizieren.

Achtung

Es gibt Standards, welche das Salt direkt zusammen mit dem Hash in einem langen String kombiniert ablegen. Dann benötigen Sie keine eigene Spalte für das Salt. `php` macht dies in der Standard-Verwendung z.B. so.

Perl-Snippets (Hilfestellung)

Erzeugen eines zufälligen Salt aus 10 Zeichen:

```
my $random_salt = String::Random->new->randregex('[a-zA-Z0-9]{10}');
```

Erzeugen eines MD5-Hashes aus einem Passwort:

```
my $hash = Digest->new('MD5')->add($password)->hexdigest;
```

Erzeugen eines MD5-Hashes aus einem Passwort mit einem Salt:

```
my $hash = Digest->new('MD5')->add($password . $salt)->hexdigest;
```

Abfragen einer Information aus der Datenbank:

```
my $sth = $dbh->prepare("SELECT x FROM a WHERE y = 'z'");
$sth->execute();
my ($x) = $sth->fetchrow_array();

if ($x) {
    # etwas gefunden
}
else {
    # nichts gefunden
}
```

Speichern einer Information in der Datenbank:

```
my $sql = "INSERT INTO user (name, pw_algo, pw_hash, berechtigung, "
        . "algo_param, pw_salt) VALUES ('$user', 'md5', '$pw_hash', "
        . "0, 0, '');";
my $sth = $dbh->prepare($sql);
$sth->execute();
```

Verwenden des Algorithmus Eksblowfish/bcrypt:

```
use Digest;
use Digest::Bcrypt;
my $hash = Digest->new ( 'Bcrypt' )
                 ->cost( 10 )
                 ->salt($salt)
                 ->add ($password)
                 ->hexdigest;
```

Verwenden des Algorithmus SHA:

```
use Digest;
use Digest::SHA;
my $hash = Digest->new ( 'SHA-256' )
                 ->add ($password . $salt)
                 ->hexdigest;
```

Einfache Gegenmassnahmen sind:

- die Quelle der «Störanfragen» sperren (z.B. per Firewall-Regel)
- die Leistung des Servers und dessen Netzanbindung erhöhen

Mit einem einfachen Skript können Sie Anfragen spammen:

```
i=0; while true; do ziu artikel a; ((i++)); echo "Request $i done"; done
```

Dies macht immer sofort wieder eine Anfrage, sobald die vorherige beantwortet wurde. Öffnen Sie mehrere Terminals und kopieren das Kommando in jedes Terminal, entsprechend dem Screenshot unten. Beachten Sie:

- Zuerst noch ein `ziu login` absetzen im ersten Terminal.

IP auf Applikationsebene blockieren

Eine Möglichkeit besteht darin, ein Limit einzuführen, wie viele Anfragen ein Client senden darf. Wenn der Client zu viele Anfragen sendet, werden diese nicht mehr bearbeitet. Als eine einfache Methode der Identifikation des Absenders kann dessen IP-Adresse verwendet werden. Kommen viele Anfragen von der selben IP, werden Anfragen von dort ignoriert.

Aufgabe 2

Bauen Sie die unteren Code-Fragmente in die Datei `bin/fulla` ein, so dass der Server schnelle Abfragen von einer IP blockiert. Testen Sie das Verhalten in Ihrem Setup, mit Kunde und Hacker.
Ps: Der Code liegt auf der Modulablage, damit Sie ihn nicht abtippten müssen.
Pps: Es ist nicht verboten ein paar Kommentare einzufügen 😀

```
# Globale assoziative Liste um Zugriffszeit pro IP abzuspeichern
my $dos_list = {};

# DOS-Überprüfung im Server-Loop
my $current_time = time();
my $last_access  = $dos_list->{$client_address};

$dos_list->{$client_address} = $current_time;

if ($last_access) {
    my $span = $current_time - $last_access;
    if ($span < 2) {
        $log->warn("client blocked: $client_address:$client_port");
        print $client_socket
            "0000000000000000000 dos from $client_address";
        shutdown($client_socket, 1);
        next;
    }
}
```

User blockieren

Das Blockieren von IP-Adressen hat einige Schwachstellen:

- IP-Adressen sind keine fälschungssichere Information über den Absender.
- Gegen grosse DDOS-Angriffe hilft dies nur bedingt.
- Wegen `NAT` können mehrer Kunden über die selbe IP kommen und sich damit gegenseitig ausschliessen.

Eine Alternative wäre es, die Abfragen von Sessions zu überwachen.

Aufgabe 3

Implementieren Sie eine Begrenzung für Sessions.

Gehen Sie analog zum oberen Beispiel vor. Sie müssen Ihren Code dort einfügen, wo bereits eine Session vergeben wurde.

Nun haben wir den Vorteil, dass wir wirklich pro Kunde überwachen. Der Nachteil ist allerdings, dass wir nun DDOS aushalten müssen, welche mit Login-Anfragen kommen!

Gegenmassnahmen

Das Problem ist Grundsätzlich nicht einfach zu lösen. Es besteht ein Zielkonflikt:

- Einerseits soll der Service gut erreichbar sein.
- Andererseits soll er nicht erreichbar sein, wenn sinnlose Anfragen kommen.

Gegenmassnahmen lassen sich auf verschiedenen Ebenen treffen. Grob lassen sich diese wie folgt Unterteilen:

Applikation, Betriebsystem, Netzwerk

Wenn der Dienst ernsthaft vor externen Angriffen geschützt werden soll, kommen Sie nicht darum herum, Massnahmen auf allen Ebenen umzusetzen. Als Applikationsentwickler sind Sie hier auf die Mitarbeit von Systemtechnikern angewiesen! Wir werden den Schwerpunkt nun auf einige mögliche Massnahmen innerhalb der Applikation legen und die anderen Bereiche nur kurz touchieren.

Behandlung auf Systemebene

Für das Blocken von *DOS*-Angriffen auf Ebene des Betriebsystems gibt es weitreichende Möglichkeiten. Ziel ist es, eine böswillige Anfrage möglichst frühzeitig zu erkennen und dann deren Absender zu blockieren.

Ein Beipiel hierfür aus der Unix-Welt ist das quelloffene Programm `fail2ban`. Damit lässt sich zum Beispiel der SSH-Zugang für Absender blocken welche massenhaft Login-Versuche auf den SSH-Port machen. Wenn Sie keinen solchen Schutz haben und einen Root-Zugang per SSH erlauben, können Sie davon ausgehen, dass der Zugang per Bruteforce innert einiger Monate geknackt ist! Sie können `fail2ban` auf beliebige Logfiles ansetzen. D.h. Ihre Applikation kann Logfiles erzeugen und `fail2ban` überprüft dann, ob Auffälligkeiten auftreten.

Hinweis

Sich vor vielen Anfragen schützen, verhindert nicht nur DOS-Angriffe sondern auch Brutforce-Angriffe!

Behandlung im Netzwerk

Viele Angriffe beruhen nicht auf Applikationsprotokollen (OSI-Layer 5-7). Oft werden direkt bekannte Schwachstellen von Netzwerkprotokollen ausgenutzt (OSI-Layer 1-4). Dem können Sie nur direkt auf Netzwerkebene entgegen treten. Sie benötigen dafür spezialisierte Router-/Firewall-Software für das Monitoring (Überwachen) des Netzverkehrs. Dies benötigt spezialisiertes Fachwissen und bildet einen eigenen Markt mit Anbietern von Produkten. Aus Sicht des Applikationsentwicklers «funktioniert» es einfach... vergessen Sie daher nicht, dass es viele Leute gibt, welche dafür sorgen, dass dem so ist!