

Übungen zu Systemsicherheit

Jürgen Kleinöder,
Michael Gernoth, Reinhard Tartler
Universität Erlangen-Nürnberg, Informatik 4

B.1 Passwörter zur Anmeldung

- Es werden keine Klartextpasswörter, sondern kryptographische Hashes verwendet.
- Ein Salt hilft gegen Wörterbuch-Angriffe
- Passworthash: $f(\text{salt}, \text{pw})$
- Salt und Hash werden gemeinsam abgespeichert.
- Bei der Anmeldung wird wieder Salt und Passwort gehasht und mit gespeichertem Ergebnis verglichen.

B.2 Gängige Verfahren

- Unix
 - ◆ traditionelles Unix crypt() (modifiziertes DES)
 - ◆ MD5
 - ◆ Blowfish (OpenBSD)
 - /etc/shadow
- Windows
 - ◆ Windows Lan Manager Hash (LM Hash)
 - ◆ DES 2*7 Zeichen
 - ◆ Konversion zu Grossbuchstaben
 - Sicherung in SAM (Security Account Manager)

B.3 Netzwerkanmeldung

- Hashes über das Netzwerk
- Challenge-Response Verfahren
- Unix
 - ◆ NIS
 - ◆ LDAP
 - ◆ Kerberos
- Windows
 - ◆ NTLM (NT Lan Manager)
 - ab NT 3.1
 - Challenge Response mit NT Password Hash (MD4)
 - ◆ Active Directory (LDAP + Kerberos)

B.4 Passwort Hash Algorithmen

■ Grundlegender Aufbau

```
$<CODE>$<SALT>$<deadbeefHASHdeadbeef>
```

■ Gängige Codes sind

- ◆ MD5: \$1\$
- ◆ Blowfish: \$1b\$

■ In dieser Übungsaufgabe

- ◆ Syssec: \$syssec1\$

B.5 Der syssec1 Hash

■ Beispiel

```
$syssec1$gcXj981u$hWP8hUh2K+xSYWdmxK1RnQ
```

■ Erklärung

- ◆ fixes Trennzeichen: \$
- ◆ fester Identifier: syssec1
- ◆ <SALT> 8 BASE64 Zeichen, also 48 bit zufälliger Salt
- ◆ <HASH> 22 BASE64 Zeichen, also 128 bit für MD5

■ Generierung: 1 mal MD5 über <SALT><PASSWORT>

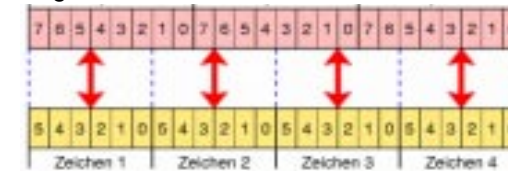
B.6 BASE64 Kodierung

■ <http://de.wikipedia.org/wiki/Base64>

■ Randbedingungen

- ◆ Es werden grundsätzlich 8-bit lange Worte kodiert
- ◆ Es wird ein 64 Zeichen Alphabet verwendet (6bit)
- ◆ Es werden (eigentlich) nur 4*n Byte lange Worte erzeugt

■ Padding



- ◆ Es wird nach Standard mit '=' aufgefüllt
- ◆ ABER: Nicht bei uns!

B.7 Die OpenSSL Bibliothek

- Projekt, um eine SSL2/3 und TLS1-Bibliothek zur Verfügung zu stellen
- Aus der SSLeay-Bibliothek hervorgegangen
- Zugriff auf die intern genutzten Kryptografiefunktionen möglich
- Mitgeliefertes Tool „openssl“ bietet unter anderem:
 - ◆ SSL-Verbindungsaufbau
 - ◆ X509 Zertifikatsmanagement
 - ◆ CA Funktionalität
 - ◆ Ver-/Entschlüsselung

B.8 OpenSSL EVP_MD*- Schnittstelle

- Initialisierung eines Kontextobjekt:

```
EVP_MD_CTX_init()
```

- Einstellen des Algorithmus (z.B. MD5)

```
EVP_DigestInit_ex()
```

- Inkrementeller Input:

```
EVP_DigestUpdate()
```

- Finalisierung

```
EVP_DigestIFinal_ex()
```

- Aufräumen

```
EVP_MD_CTX_cleanup()
```

B.9 Weitere OpenSSL Funktionen

- Zufallszahlen

```
RAND_bytes()
```

- OpenSSL BIO Interface

- ◆ Abstraktion für I/O (Speicher, Datei, Socket, BASE64, etc)
- ◆ Verkettbar

```
int len;
char buf[256];
BIO *in = BIO_new_fp(stdin, BIO_NOCLOSE);
BIO *out = BIO_new_fp(stdout, BIO_NOCLOSE);
BIO *filter = BIO_new(BIO_f_null());

out = BIO_push(filter, out);
while((len = BIO_read(in, buf, sizeof(buf))) > 0)
    BIO_write(out, buf, len);

BIO_flush(out);

BIO_free_all(out);
BIO_free(in);
```

B.10 Aufgabe

- Implementierung des syssec1-Hashes

```
$ ./genpw
Enter Password: geheim
$syssec1$3wWnBd0/$056896/DNOxrKwkgDMNv2A
```

- Benutzung von OpenSSL-Funktionen

- ◆ BIO stellt BASE64 Filter bereit
- ◆ Flag `BIO_FLAGS_BASE64_NO_NL` schaltet zeilenbasierte Verarbeitung ab
- ◆ `BIO_s_mem` erstellt einen speicherbasierten BIO
- ◆ `BIO_get_mem_data` liefert Pointer auf Rohdaten im BIO und deren Länge

- Abgabe mit Vorführung in der nächsten Übung:

- ◆ 25./27.11.