Labreport 06

Julian Deinert, Tronje Krabbe

14. Juli 2016

Inhaltsverzeichnis

1. TODO

1.2

TODO

2. TODO

2.1

2.2

Um ein selbstsigniertes Zertifikat zu erzeugen, sind die folgenden Befehle notwendig:

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Mit dem ersten Befehl erzeugen wir einen neuen, 2048-bit langen privaten Schlüssel. Der zweite Befehl erstellt die *certificate signing request*, in der die Informationen enthalten sind, die das Zertifikat später umfassen soll. Diese werden interaktiv abgefragt. Als Common Name verwenden wir `vmsrv12.svslab`.

Im dritten Befehl verwenden wir schließlich den erstellten Private Key, um aus der CSR ein Zertifikat zu generieren, das 365 Tage gültig ist.

Um dieses Zertifikat für den Apache-Server einzusetzen, ersetzen wir in der Konfigurationsdatei die Pfade hinter `SSLCertificateFile` und `SSLCertificateKeyFile` mit den Pfaden zum Zertifikat und dem private key.

Um einen Missbrauch von Zertifikaten und damit auch ihre Wirksamkeit sicherzustellen, werden selbsterstellten Zertifikaten nicht vertraut. Vertrauenswürdige Zertifikate werden von *Certificate Authorities* ausgestellt, deren Aufgabe es ist, rechtmäßig angefragte Zertifikate auszustellen und mit Glaubwürdigkeit zu versehen sowie im Umkehrschluss unglaubwürdige Zertifikatserstellungen zu erschweren.

2.3

Das Apache-Modul *mod_ rewrite* ermöglicht Weiterleitungen einzurichten. Wir wollen alle Requests an HTTPS weiterleiten und erweitern die config daher um folgende Zeilen. [1]

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

Mit dem ersten Befehl wird die Bearbeitung der Rewrite-Regeln eingeschaltet. Danach wird geprüft, ob nicht bereits HTTPS verwendet wird, da damit die Weiterleitung hinfällig wird. Schließlich wird die Rewrite-Regel definiert, die dann weiterleitet. Die Optionen am Ende sind die Flags. `L` gibt an, dass die Regel die letzte ist, die ausgeführt wird. `R` spezifiziert, dass ein HTTP Redirect passieren soll.

Die Regel selbst ist ein regulärer Ausdruck, bei dem alles nach einem optionalen "`/`" matched und in die HTTPS-URL eingesetzt wird.

2.4

Über die verschlüsselte Verbindung können wir die Zugangsdaten selbstverständlich nicht mitlesen. Verwenden wir *sslstrip* als Proxy, so wird die Umleitung auf die verschlüsselte Verbindung verhindert und wir können die Zugangsdaten recht komfortabel im Log-File nachlesen.

Die Sicherheit der Weiterleitung ist damit maximal als "gut gemeinte" Maßnahme für Endanwender gedacht, taugt aber nicht, ernsthafte Angriffe abzuwehren. Hierzu sind weitere Maßnahmen nötig, wie beispielsweise HSTS.

HTTP Strict Transport Security (HSTS) ist ein in RFC 6797[2] beschriebener Standard, der einen solchen Angriff verhindern kann. Wird HSTS aktiviert, so können Seiten unter einer Domain, die HSTS setzt, nicht über eine unverschlüsselte Verbindung abgerufen weden. HSTS wird über einen Header aktiviert, der bei der Antwort mitgesendet wird und etwa so aussieht:

```
Strict-Transport-Security: max-age=15768000
```

[1] Quelle: `https://wiki.apache.org/httpd/RewriteHTTPToHTTPS`

[2] `https://tools.ietf.org/html/rfc6797`

Damit wird HSTS im Browser strikt forciert und lässt sich auch für den Endnutzer nicht wieder abschalten. Mit dem `max-age`-Parameter wird das maximale Alter in Sekunden angegeben. 15768000 Sekunden entsprechen etwa einem halben Jahr, danach wird der Browser eine unverschlüsselte Verbindung prinzipiell wieder erlauben, wenn nicht erneut ein HSTS-Header mitgesendet wird.

3. Unsichere selbstenwtickelte Verschlüsselungsalgorithmen

3.1 BaziCrypt

Die letzten 10 Bytes des Ciphertexts sind exakt der Key. Oder, genauergesagt, der Key XOR Null, was den Key unverändert lässt. Jetzt ist es sehr einfach, die Dateien zu entschlüsseln. Siehe unser Skript im Appendix. Die drei Plaintexte sind:

[n01.txt.enc] Hallo Peter. Endlich koennen wir geheim kommunizieren! Bis bald, Max

[n02.txt.enc] Hi Max! Super, Sicherheitsbewusstsein ist ja extrem wichtig! Schoene Gruesse, Peter.

[n03.txt.enc] Hi Peter, hast du einen Geheimtipp fuer ein gutes Buch fuer mich? Gruss, Max

3.2 AdvaziCrypt - Denksport

Wenn wir davon ausgehen, dass der Key vollständig (wenn auch 'verschlüsselt') in der Datei vorhanden ist, dann sind auf jeden Fall die letzten 10 Bytes des Ciphertexts die verschlüsselten Key-Bytes. Jetzt kann man auf die Anzahl der Padding-Bytes schließen, indem man darauf achtet, wann die hinteren Bytes aufhören, Teil des vermeintlichen, verschlüsselten Keys zu sein. Man zählt also mit und schaut, wann ein Byte nicht mehr passt. Für die genaue Implementation, siehe unser Skript im Appendix.

3.3 AdvaziCrypt

Die Plaintexte lauten:

[n04.txt.enc] Hi Max, natuerlich: Kryptologie von A. Beutelspacher ist super. Gruss Peter

[n05.txt.enc] Hi Peter, worum geht es in dem Buch? Ciao, Max.

[n06.txt.enc] Hi Max, das ist ein super Buch, das viele Krypto-Themen abdeckt. Gruss Peter

Appendix

3.4 3. Skripte

Bazi Crack

```
#!/usr/bin/env python
def crack(filename, bsize):
    """Decrypt a bazi-encrypted file given the file's
    name and the blocksize (usually 10)
    """

    result = ''
    counter = -1
    with open(filename, "rb") as f:
        f.seek(-bsize, 2) # seek last n bytes
        key = [int(b) for b in f.read(bsize)]
        f.seek(0, 0) # seek beginning of file again
        while True:
            counter += 1
            counter = counter % bsize
            b = f.read(1)
            if b == b'':
                break
            num = int.from_bytes(b, 'little')
            result += chr(num ^ key[counter])
    return result


if __name__ == "__main__":
    import sys
    print(crack(sys.argv[1], 10))
```

Advazi Crack

```
#!/usr/bin/env python
def crack(filename, bsize):
    """Decrypt an advazi-encrypted file given the file's
    name and the blocksize (usually 10)
    """

    result = ''
    with open(filename, "rb") as f:
        f.seek(-bsize, 2) # seek last n bytes

        # initialize key as last n bytes
        key = [int(b) for b in f.read(bsize)]

        f.seek(0, 0) # seek beginning of file again
        whole_file = f.read() # read in the entire file

        # compute padding
        padding = 0
        counter = bsize - 1
        for i in range(len(whole_file) - 1, 0, -1):
            if whole_file[i] != key[counter]:
                break
            else:
                padding += 1
            counter -= 1
            counter %= 10

        # decrypt key with padding value
        for i in range(len(key)):
            key[i] ^= padding

        counter = -1
        f.seek(0, 0) # seek beginning of file again

        # decrypt plaintext
        while True:
            counter += 1
            counter %= 10
            b = f.read(1)
            if b == b'':
                break
            num = int.from_bytes(b, 'little')
            result += chr(num ^ key[counter])
    return result[:len(result) - padding]

if __name__ == "__main__":
    import sys
    print(crack(sys.argv[1], 10))
```