SVS Bachelor-Projekt Network Security

Blatt 6: Kryptographie

Louis Kobras
6658699

Utz Pöhlmann
6663579

1 Absicherung des TCP-Chats mit SSL

2 CAs und Webserver-Zertifikate

2.2 Selbstsignierte Zertifikate

Es wurde in mehreren Läufen die Fallstudie durchgearbeitet. Und zwar mehrmals und sowohl zusammen als auch einzeln und unabhängig voneinander. Mit dem Ergebnis, dass der Apache2-Server nicht funktioniert. Die Gruppe neben uns, denen wir inzwsichen bestimmt mega auf den Keks gehen und denen ich als Wiedergutmachung ein Eis mitgebracht habe, konnte uns leider auch nicht helfen. Unsere certs und pems und reqs und keys und csrs wurden alle ordnungsgemäß erstellt und Schritt für Schritt, Wort für Wort nach der Fallstudie erzeugt und bearbeitet. Es ist kaputt. Selbst Reboots und Reinstallationen helfen nicht. Folglich funktionieren Aufgabe 2.1ff nicht. Mal wieder. Sind wir echt so blöd oder liegt vielleicht ein Fehler auf unserer Maschine vor?

2.3 HTTPS-Weiterleitung

.

2.4 sslstrip

sslstrip wurde nach [1] installiert und gestartet.
Die Verbindung wurde am "(svs.informatik.uni-hamburg.de)" in der Logdatei erkannt. (IP: 134.100.15.55)
Die Browsereinstellungen wurde unter `Edit` → `Preferences` → `Advanced` → `Network` → `Connection` → `Settings...` auf `localhost` und Port `8080` gesetzt. Zudem wurden die `No Proxy for`-Einstellungen entfernt.
Der Inhalt der Datei: vgl. [ssllog (S. 3)] Die Lösung aus Aufgabe 2.3 ist somit definitiv ein Plus an Sicherheit. Der Sinn von HSTS ist, sich vor sog. "downgrade Attacks" zu schützen. Hierbei wird der Client dazu gezwungen, statt einer "modernen" sicheren Verbindung eine "alte" unsichere Verbindung aufzubauen. (Bsp.: HTTP statt HTTPS) Ein weiterer Nutzen ist, "Session Hijacking" zu unterbinden. Hier wird ein Authentifikationscookie abgefangen und so ein Man-In-The-Middle-Angriff gestartet. Da HSTS Webservern erlaubt, auf Browser den Zwang einer sicheren Verbindung (via HTTPS) auszuüben, sind alle Server, die diese Möglichkeit nutzen, auch sicher vor SSL-Stripping-Angriffen.

3 Unsichere selbstentwickelte Verschlüsselungsalgorithmen

3.1 BaziCrypt

Quellcode siehe [3.1: Knacking BaziCrypt (S. 3)]

3.2 AdvaziCrypt - Denksport

.

3.3 AdvaziCrypt - Angriff implementieren

.

4 EasyAES

5 Timing-Angriff auf Passwörter (Bonusaufgabe)

Literatur

[1] https://moxie.org/software/sslstrip/

ANHANG

ssllog

Anmerkung: Aufgrund der Länge der Zeilen wurden nachträglich manuell Zeilenumbrüche eingefügt.

```
2016-06-23 15:45:58,171 POST Data (safebrowsing.clients.google.com):
goog-malware-shavar;a:239451-244530:s
    :234828-234868,234872-234874,234876-234888,234890-234895,
     234897-234901,234903-234904,234906,234910-234915,234917-234927,
     234929-235103,235105-235168,235170-235176,235178-235198,235200-235256,
     235258-235274,235276-235307,235309-235473,235477,235479-235485,
     235487-235807,235809-235974,235976-236189,236191-236363,236365-236366,
     236368-237764,237766-238163,238165-238181,238183-238196,238198-239044,
     239046-239996:mac
goog-phish-shavar;a:448570-450992:s
    :268799-268858,268860-269092,269096-269182,
     269184-269212,269214-269222,269224-269265,269267-269291,269293-269295,
     269297-269311,269313,269316-269347,269349-269351,269353-269356,
     269359-269360,269362-269368,269370,269390-269392,269408-269513,
     269515-269518,269521-269575,269577-269606,269608-269615,269617,
     269619-269628,269630-269694,269696-269734,269736-269843,269845,
     269847-269902,269904-269926,269928-269937,269939-269995,269997-270023,
     270025-270092,270094-270115,270117-270151,270153-270163,270165-270190,
     270192-270205,270207-270234,270236,270238-270275,270277-270374,
     270376-270402,270404-271027,271030-271036,271038-271049,271051-271118,
     271120-271194,271196-271212,271214-271227,271229-271391,271393-271394,
     271398-271434,271436-271442,271444-271449,271452-271472,271475-271526,
     271528-271595:mac

2016-06-23 15:47:15,606 SECURE POST Data (svs.informatik.uni-hamburg.de):
username=admin&password=password
```

3.1: Knacking BaziCrypt

```
import sys


def xor(first_string, second_string):
    """
␣␣␣␣given␣two␣HEX␣strings,␣uses␣the␣XOR␣function␣between␣them␣by␣converting
    ␣them␣to␣lists␣of␣integers
␣␣␣␣:param␣first_string:
␣␣␣␣:param␣second_string:
␣␣␣␣:return:␣the␣result␣of␣XOR
␣␣␣␣"""
    # convert the first HEX string to integers
    first_list = []
    for c in first_string:
        b = int(c, 16)
        first_list.append(b)
    # converts the second HEX string to integers
    second_list = []
    for c in second_string:
        b = int(c, 16)
        second_list.append(b)
    result = []
    # XORs the lists
```

```
23      for i in range(0, len(first_list)):
24          j = first_list[i] ^ second_list[i]
25          j = hex(j)  # converts the XORd integer to a HEX
26          j = j[2:]  # cut the HEX '0x' notation given by the built-in
                function cast hex(1)
27          result.append(j)
28      result = ''.join(result)  # joins the XORd list to a string
29      return result
30
31
32  def task_three_point_one(msg):
33      """
34  ␣␣␣␣task_three_point_two␣function␣of␣the␣program.␣takes␣a␣HEX-encrypted␣
        message␣and␣decrypts␣it
35  ␣␣␣␣:param␣msg:␣the␣message␣to␣decrypt
36  ␣␣␣␣:return:␣the␣decrypted␣message
37  ␣␣␣␣"""
38      # find the key that was used to encrypt
39      """
40  ␣␣␣␣absuses␣the␣nature␣of␣the␣encryption␣method␣used␣by␣taking␣the␣last␣10␣
        chars␣of␣the␣message,␣which␣are␣the␣key
41  ␣␣␣␣"""
42      key = msg[-20:]
43      # decrypt the message using the found key
44      """
45  ␣␣␣␣due␣to␣the␣nature␣of␣the␣encryption␣method␣used,t␣he␣message,␣being␣
        symmetrically␣encrypted,
46  ␣␣␣␣can␣be␣decrypted␣by␣XORing␣the␣message␣with␣the␣encryption␣key.
47  ␣␣␣␣So,␣bring␣the␣key␣to␣the␣same␣length␣as␣the␣message␣and␣xor␣them
48  ␣␣␣␣"""
49      key *= 10  # takes advantage of the fact that each key had a length of
            10 chars
50      # and each message had a length of 100 chars, setting the length factor
            to 10
51      msg = xor(msg, key)
52      msg = get_chars(msg)
53      return "message:␣{}".format(''.join(msg))
54
55
56  def get_chars(lst):
57      """
58  ␣␣␣␣given␣a␣list␣of␣HEX␣values,␣this␣function␣returns␣the␣char␣values␣of␣
        each␣position
59  ␣␣␣␣:param␣lst:␣a␣list␣of␣hex␣values
60  ␣␣␣␣:return:␣a␣list␣of␣char␣values␣as␣integer
61  ␣␣␣␣"""
62      res = []
63      for i in xrange(0, len(lst) - 1, 2):
64          a = int(lst[i], 16)
65          a *= 16
66          b = int(lst[i + 1], 16)
67          b += a
68          res.append(chr(b))
69      return res
70
71
72  for i in range(1, len(sys.argv)):
73      f = open(sys.argv[i], 'r')
74      f = f.read()
```

```
75      f = f.encode('hex')
76      f = task_three_point_one(f)
77      print f
```