genua

Debugging SSL/TLS

Perl Workshop 2015

Steffen Ullrich

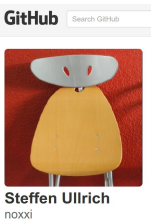
http://noxxi.de/howto/ssl-debugging.html

Slides: http://noxxi.de/talks.html

about:me

genua

- genua mbH seit 2011
  Application Level Firewalls, Forschung IT-Sec
- Maintainer IO::Socket::SSL seit 2006
  Non-Blocking I/O, sichere Defaults, SSL-Interception
- Sehe SSL-Probleme als interessante Knobelaufgaben, sofern es nicht immer wieder die gleichen sind.



Ablauf

- keine Konfiguration, (fast) kein Code
  findet man genug woanders
- Grundlagen zum Verständnis
- nützliche Tools
- Typische und exotische Fehler
- Ursache finden und Problem fixen
  - anhand Symptom/Fehlermeldung
  - geht woanders, ging gestern noch ...
- Beispiele aus dem Alltag
- Entwicklerecke

Grundlagen

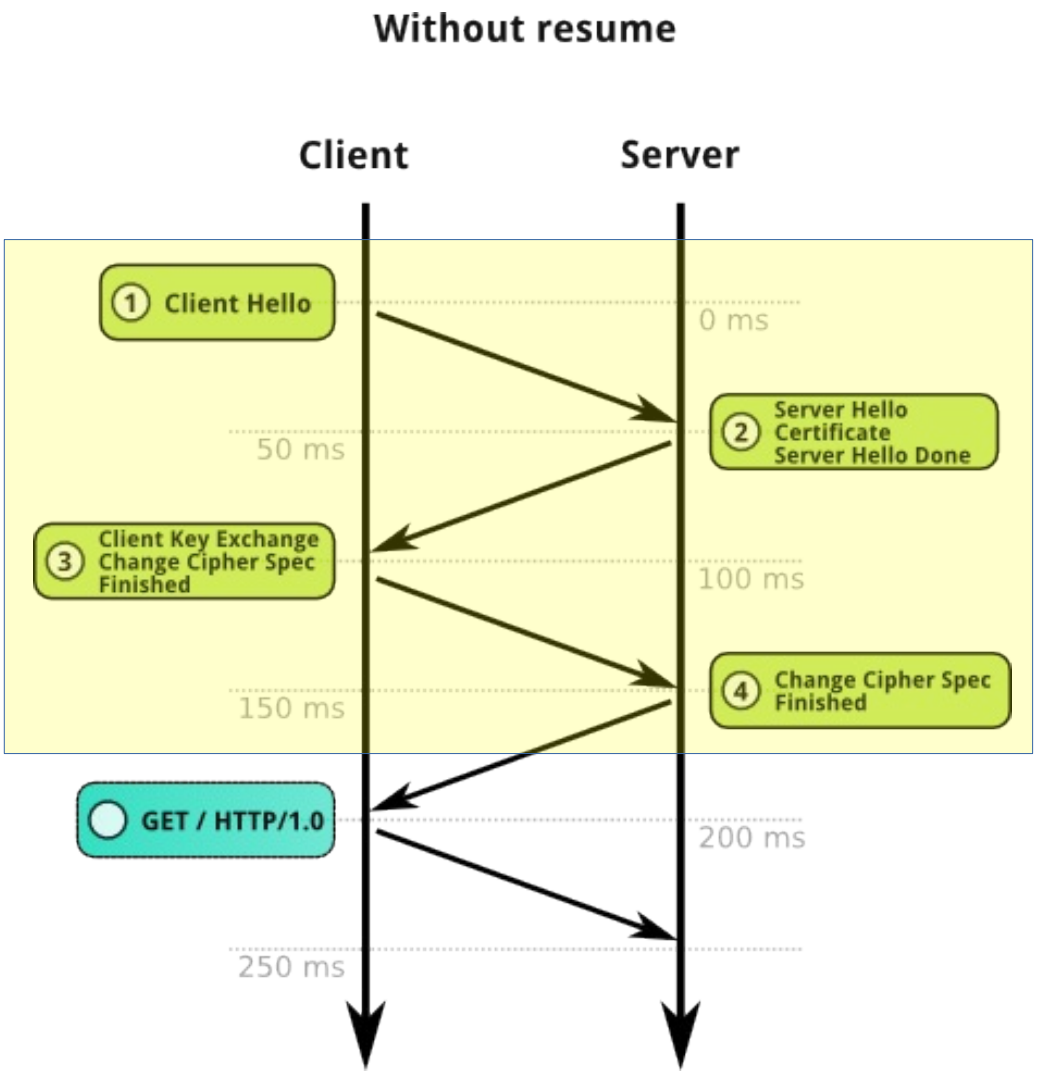
wie funktioniert TLS
häufige Mißverständnisse

Basics TLS

- verschlüsselter Transport zwischen Client und Server
- keine sichere Verschlüsselung ohne sichere Identifikation des Servers
  - kaputte Identifikation -> MITM
  - Identifikation i.A. über Zertifikate
- Handshake:
  - Client offeriert Protokollversion und Ciphers
  - Server offeriert Zertifikate, wählt Cipher und Version
  - Key Exchange

Offer:
Protokollversion
Cipher-Suites
TLSExt: SNI*
...

Clientzertifikat



Protokollversion
Cipher-Suite
Serverzertifikat
Zertifikat-Request
...

http://vincent.bernat.im/en/blog/2011-ssl-session-reuse-rfc5077.html

* SNI – Server Name Indication

häufige Missverständnisse 1

- will TLS nicht SSL
  - TLS 1.0 == SSL 3.1 ...
  - Oft im Zusammenhang mit IMAP, SMTP, FTP..: SSL == implizit, TLS == explizit (STARTTLS)
- SSL 3.0 abschalten (POODLE) durch Abschalten aller SSL 3.0 Ciphern
  - Protokoll != Cipher-Suite
  - Resultat: nur die für TLS 1.2 definierten neuen Ciphern bleiben übrig, d.h. TLS 1.0 und TLS 1.1 gehen nicht mehr

häufige Missverständnisse 2

- will nur Verschlüsselung, keine Zertifikate..
  - ohne saubere Identifikation MITM möglich
  - prinzipiell möglich ohne Zertifikate (ADH), von Browsern i.A. nicht unterstützt
  - Firefox 37 – Versuche von Opportunistic Encryption (erstmal Rollback)
  - SMTP – mehr Probleme als nur saubere Identifikation

Scheinbar **kein Problem**
Schließlich funktioniert ja alles !!

genua

- fehlerhafte Zertifikatsvalidierung
  - unsichere Defaults:
    - keine Prüfung
    - nur Trust Chain, nicht Hostnamen
    - oft bei C, Python, PHP, Ruby, Perl... (wird besser)
  - sichere Defaults extra abgeschalten
    - oft bei Android, iOS, Java – VU#582497,...
  - Bugs
- Heartbleed, goto fail, Winshock, ...
- schwache Protokolle und Ciphern

es funktioniert, aber...

genua

1,500 iOS apps have HTTPS-crippling bug. Is one of them on your device?

Apps downloaded two million times are vulnerable to trivial man-in-the-middle attacks.

by **Dan Goodin** - Apr 20, 2015 9:08 pm UTC

AFNetworking Strikes Back: 25,000+ Apps

April 24, 2015

Remember back when 1,500 vulnerable ap

A quick check found that apps from Bank of America, Wells Fargo, and JPMorgan Chase were likely affected, although some of those reports may be false positives. It's possible that some apps flagged by SourceDNA use custom code or secondary measures such as certificate pinning that prevents attacks from working. Apps from Microsoft, meanwhile, remained vulnerable to the HTTPS-crippling bug reported earlier.

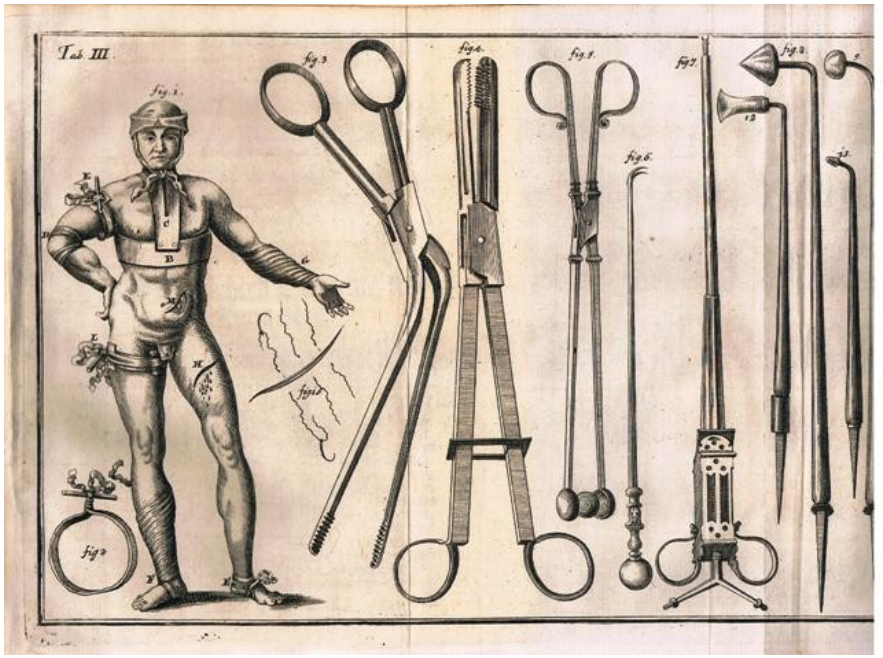
Android apps *still* suffer game-over HTTPS defects 7 months later

Apps with >350 million downloads fail to detect simple man-in-the-middle attack.

by **Dan Goodin** - Apr 27,

The vulnerable apps include OKCupid Dating, Dish Anywhere, ASTRO File Manager with Cloud, CityShop – for Craigslist, and PicsArt Photo Studio, which collectively have commanded from 170 million to 670 million downloads, according to official Google Play figures. Most of the titles have been updated regularly, but they continue to contain a game-over vulnerability that fails to detect fraudulent transport layer security (TLS) certificates, according to a blog post published Sunday by Sam Bowne, a

Tools



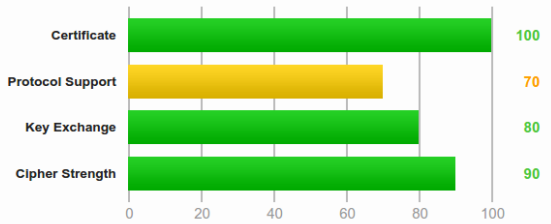
Tools für Debugging - SSLLabs

https://www.ssllabs.com/ssltest/analyze.html

Summary

Overall Rating

C



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server is vulnerable to the POODLE attack. If possible, disable SSL 3 to mitigate. Grade capped to C. MORE INFO »

Intermediate certificate uses SHA1. Upgrade to SHA256 as soon as possible to avoid browser warnings. MORE INFO »

The server supports only older protocols, but not the current best TLS 1.2. Grade capped to B.

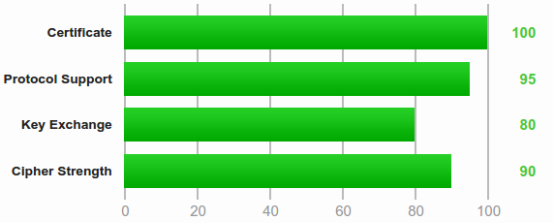
The server does not support Forward Secrecy with the reference browsers. MORE INFO »

This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks.

Summary

Overall Rating

A



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

Certificate uses SHA1. When renewing, ensure you upgrade to SHA256. MORE INFO »

This site works only in browsers with SNI support.

This server is not vulnerable to the POODLE attack because it doesn't support SSL 3. MORE INFO »

This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks.

This server supports HTTP Strict Transport Security with long duration. MORE INFO »

Additional Certificates (if supplied)

Certificates provided	4 (5036 bytes)
Chain issues	Incomplete, Extra certs, Contains anchor

Path #1: Trusted

1	Sent by server	*.sierrawireless.com SHA1: 5eb9489fec718a25c0633a0490c539c4c71f8ebd RSA 2048 bits / SHA256withRSA
2	Extra download	Go Daddy Secure Certificate Authority - G2 SHA1: 27ac9369faf25207bb2627cefaccbe4ef9c319b8 RSA 2048 bits / SHA256withRSA
3	In trust store	Go Daddy Root Certificate Authority - G2 SHA1: 47beabc922eae80e78783462a79f45c254fde68b RSA 2048 bits / SHA256withRSA

IE 6 / XP No FS [1] No SNI [2]	Protocol or cipher suite mismatch		Fail[3]
IE 7 / Vista	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
IE 8 / XP No FS [1] No SNI [2]	Protocol or cipher suite mismatch		Fail[3]
IE 8-10 / Win 7 R	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
IE 11 / Win 7 R	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) FS	128
IE 11 / Win 8.1 R	TLS 1.2	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x9f) FS	256
IE Mobile 10 / Win Phone 8.0	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
IE Mobile 11 / Win Phone 8.1	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) FS	128
Java 6u45 No SNI [2]	TLS 1.0	TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x33) FS	128
Java 7u25	TLS 1.0	TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x33) FS	128

Tools für Debugging - analyze.pl

https://github.com/noxxi/p5-ssl-tools/blob/master/util/analyze-ssl.pl

genua

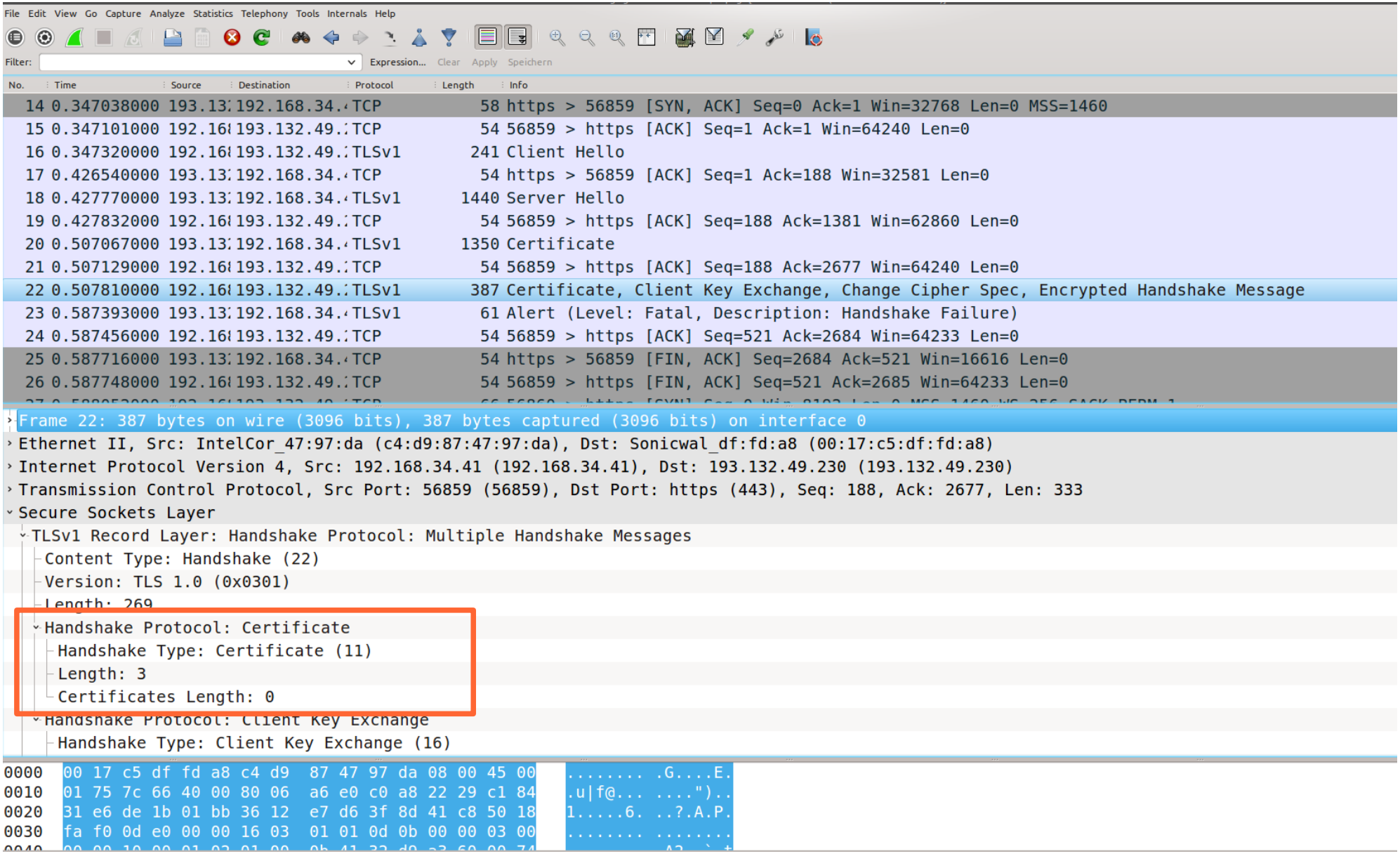
- SMTP, IMAP, POP3, FTP, HTTP Proxy, PostgreSQL
- IPv4 und IPv6
- checkt ob SNI unterstützt, optional oder nötig

```
$ perl analyze-ssl.pl --all-ciphers --show-chain --starttls smtp mx1.genua.de
 ...
 * maximum SSL version  : TLSv1_2 (SSLv23)
 * supported SSL versions with preferred cipher:
   * TLSv1   ECDHE-RSA-AES256-SHA
   * TLSv1_1 ECDHE-RSA-AES256-SHA
   * TLSv1_2 ECDHE-RSA-AES128-GCM-SHA256
 * cipher order by      : server
 * SNI supported        : ok
 * certificate verified : ok
   * [0] bits=2048, ocsp_uri=http://ocsp.comodoca.com, /OU=Domain Control Validated
         /OU=PositiveSSL Wildcard/CN=*.genua.de SAN=DNS:*.genua.de,DNS:genua.de
   * [1] bits=2048, ocsp_uri=http://ocsp.usertrust.com, /C=GB/ST=Greater Manchester/L=Salford
         /O=COMODO CA Limited/CN=PositiveSSL CA 2
 * supported ciphers
   * TLSv1_2 ECDHE-RSA-AES128-GCM-SHA256
   * TLSv1_2 ECDHE-RSA-AES256-GCM-SHA384
   * TLSv1_2 ECDHE-RSA-AES256-SHA384
   * TLSv1_2 ECDHE-RSA-AES128-SHA256
   * TLSv1_2 DHE-RSA-AES256-GCM-SHA384
   ...
   * TLSv1_2 RC4-SHA
```

Tools für Debugging
- weitere

- OpenSSL Kommandozeile
  - openssl s_client -connect ...
    -cipher, -starttls, -debug, -servername ...
  - openssl s_server ...
    -cipher, -debug, -www ...
  - openssl x509 -text ...
- SSLyze

Tools für Debugging
- Wireshark



Problem#1: Zertifikate

Zertifikat fehlerhaft

- Self-Signed
- abgelaufen
- zu lange gültig (CAB, Chrome 42+):
  ab 1.4.2015: max 39 Monate
  ab 1.7.2012: max 60 Monate
  davor: max 120 Monate bzw. max 1.7.2019
- Client kennt Signaturalgorithmus nicht
  https://support.globalsign.com/customer/portal/articles/1499561-sha-256-compatibility
- Revoked
- Weiterhin (meist alte oder selbsterstellte):
  - unsichere Signatur oder RSA Keylänge ...
  - keine extKeyUsage

Zertifikat fehlerhaft - Name passt nicht

- falsches Zertifikat wegen SNI
- oder Zertifikat fehlerhaft: RFC 2818, RFC 6125, CA Browser Forum Baseline Requirements:
  - Wildcards:
    ```
    *.example.com !~ foo.bar.example.com
    *.example.com !~ example.com
    ```
  - Wildcards nur in SAN (Subject Alternative Names) erlaubt (Safari)
  - CN ignorieren (nur) wenn SAN DNS (Safari)
  - IP als SAN IP und nicht als CN oder SAN DNS. *Für IE zusätzlich als SAN DNS anlegen!*
  - Einige CA scheinen nicht so genau zu wissen, was erlaubt ist. Browser sind unterschiedlich relaxt.

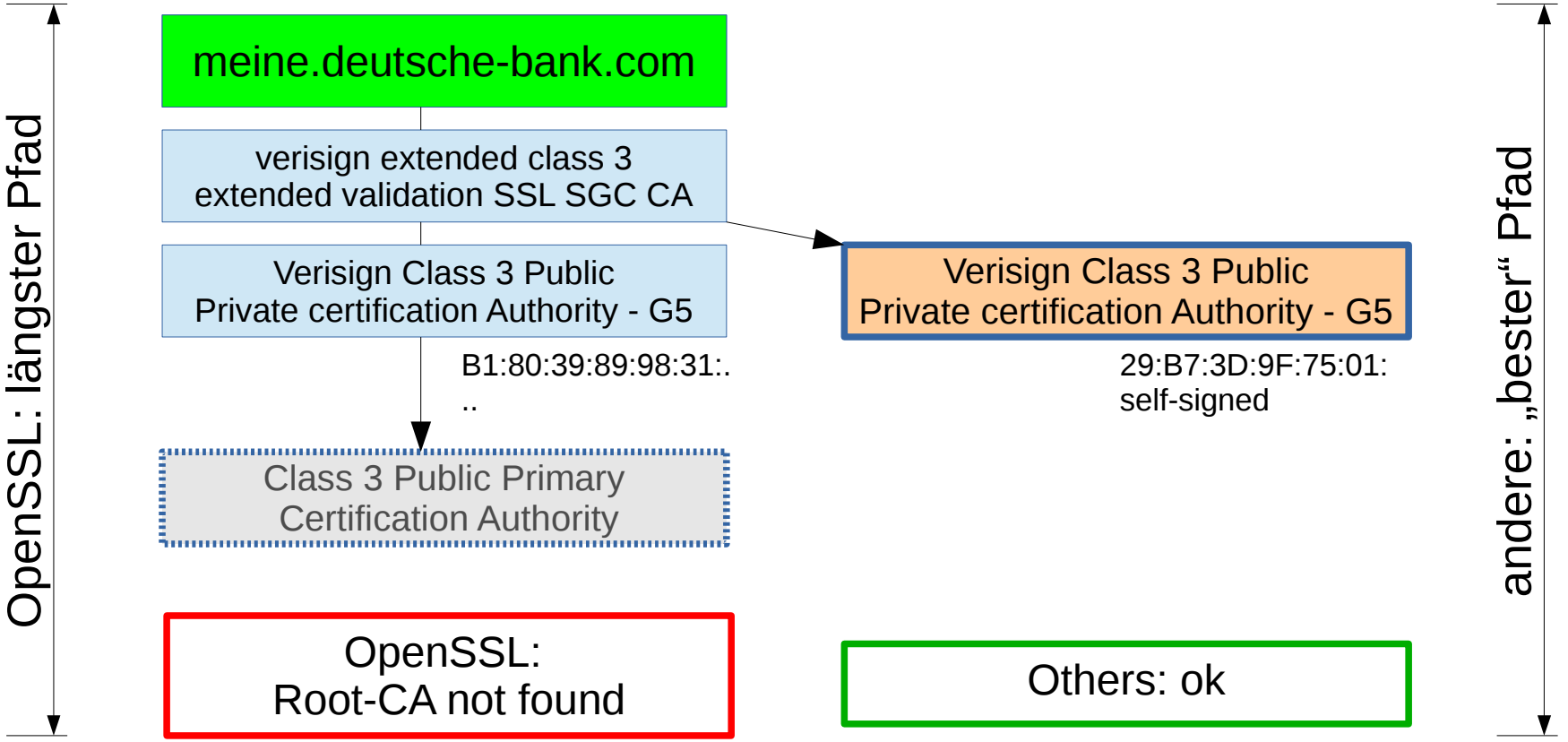
Trust chain 1
Zwischenzertifikate

genua

Chain issues	Incomplete, Extra certs, Contains anchor

- Fehlkonfiguration Server:
  - fehlende Zwischenzertifikate
  - falsche Zwischenzertifikate
  - falsche Reihenfolge
- Desktop Browser füllen oft auf mit
  - gecachten Zertifikaten (FF)
  - Download fehlender Zertifikate (Chrome, IE)
- andere Applikationen bügeln Fehler nicht aus
- Trust-Path u.U. abhängig von TLS Stack

Trust Chain - OpenSSL Multi-Path Probleme



OpenSSL #2732 von 2012 – noch offen
OpenSSL 1.0.2: X509_V_FLAG_TRUSTED_FIRST
OpenSSL 1.0.x: fixed (#3621,#3637 von 2015)

Betroffen alle Nutzer von OpenSSL (Perl, Python, Ruby, curl, wget...)

Trust Chain 2
Root-CA

genua

- Wo sind die Root-CA
  - Firefox: per Profile
  - MSIE, Safari: System
  - Chrome: System (Windows, Mac) bzw. Datadir
  - Java: per Benutzer
  - Python, PHP, Perl, Ruby:
    mal so mal so und mal anders
- Was beinhalten Root-CA
  - MSIE, Chrome, Firefox: 100+
  - Firefox: Ende 2014 viele 1024bit CA entfernt
  - Rest: Auswahl, oft veraltet.
    Teilweise basierend auf Mozilla-CA.

genua

- Wo und wie müssen eigene Zertifikate eingefügt werden (z.B. für SSL Interception):
  - Browser: GUI
  - Java, Mac: keytool
  - Curl: je nach Backend (OpenSSL, NSS, ...)
  - Perl, Python...: je nach Module
  - Linux: `/etc/ssl/certs` + rehash
  - *BSD: `/etc/ssl/cert.pem`
  - `...`

genua

Problem#2: alte Implementationen

alte Implementationen

- fehlender Support für SNI
  - Nutzung von SNI nimmt zu, z.B. Cloudflare Free SSL
  - kein SNI in MSIE/XP, Android (Apache HTTPClient), Java 6, Python 2.7.8-, älteren IO::Socket::SSL (Perl), SSL 3.0...
- div. Server nur mit RC4-* unterwegs
  - curl 7.35+ hat diese standardmäßig disabled
  - MSIE 11 nur bei Protokolldowngrade
- SSL 3.0 zum großen Teil abgeschalten
  - viele Clients (PhantomJS u.a.) default SSL3.0

alte und kaputte Implementationen

- wget < 1.12 checkt nur gegen CN nicht SAN
- F5 Big IP Loadbalancer < 10.2.4 (2012) verschluckt ClientHello 256..512 Bytes
  - wird getriggert von vielen Ciphern (TLS1.2) und/oder Nutzung von SNI
  - Workaround in 1.0.1g `SSL_OP_TLSEXT_PADDING` führt zu Problemen mit IronPort
- ECC Probleme auf Redhat, Suse
  elliptic curve routines: EC_GROUP_new_by_curve_name: unknown group
- Perl LWP <6.06 Proxy-Unterstützung
  400 Bad Request

Debugging

http://commons.wikimedia.org/wiki/File:Innards_of_a_G._Seifert_mechanical_gold_watch_-b.jpg

Schwierigkeiten beim Debugging

genua

- wenig Informationen
  - Client/Server versenden höchstens Alerts, die nur eine Nummer haben ohne Zusatzinfos.
  - Zusatzinfos nur in eventuellen lokalen Logmeldungen.
  - SChannel macht oft einfach Verbindung zu statt Alert zu verschicken.
  - Einige Alerts eher verwirrend, da im falschen Zusammenhang benutzt.

Schwierigkeiten beim Debugging

genua

- jeder machts anders
  - Browser versuchen um Fehler herumzuarbeiten mit Protokoll-Downgrades und Blacklists, andere Applikationen nicht.
  - Zertifikate werden an verschiedensten Stellen konfiguriert.
  - Gleiches OS, Service Pack, Software... sind kein Garant für gleiches Verhalten. Ciphern und Root-CA können z.B. maschinen-, applikations oder nutzerspezifisch sein.
  - verschiedene Backends für gleiche Applikationen (Chrome, curl...)

Schritte zum erfolgreichen Debugging

- Informationsbeschaffung
  - Fehlermeldungen checken auf beiden Seiten. Debugging aktivieren.
  - Tools benutzen zum Check von SNI, Chain, Ciphers, Protokollversion, Inkompatibilitäten...
- Problem eingrenzen
  - Client, Server, Middlebox, bestimmte Software
  - Variationen der Verbindung testen: andere Clients, Server, Netzwerke...
  - erneutes Testen mit Protokoll, Cipher die bei anderen Variationen erfolgreich waren

Fehlermeldungen und Ursachen 1
Server mag Client nicht

genua

- (TCP) Connection failed
  - Ist der Server wirklich online, an diesem Port und dieser IP?
  - Evtl. Firewalls dazwischen.
- No shared ciphers
  - Fehlkonfiguration auf Client oder Server, z.B. nur alte Ciphers, TLS 1.0 aber keine SSL 3.0 Ciphers erlaubt ...
  - Fehlende Zertifikat auf Server erlaubt nur ADH, was Clients i.A. nicht unterstützen

Fehlermeldungen und Ursachen 2
Server mag Client nicht

- unknown protocol
  - Server kann nicht Protokoll was Client anbietet, z.B. Client nur SSL 3.0 oder Server nur TLS 1.2
- unknown name, handshake failure
  - SNI wird gebraucht. Server hat kein Default-Zertifikat oder Client hat falschen Namen benutzt.
- SSL Handshake timed out, „want read“
  - F5 Big IP oder Server spricht kein TLS und wartet auf mehr Daten

Fehlermeldungen und Ursachen 3
Server mag Client nicht

- tlsv1 alert decode error, bad record mac, unknown protocol, record too long, handshake failure ...
  - Server spricht kein TLS oder erst nach STARTTLS. Client interpretiert Message vom Server irgendwie als TLS.
  - HP ILO2 o.ä

Fehlermeldungen und Ursachen 4
Server mag Client nicht

genua

- connection closed, connection reset by peer, handshake failure, error 40, SSL_connect SYSCALL
  - Crash von Peer
  - unerwartetes Client-Zertifikat (Winshock fun)
  - Server kann Client-Zertifikat nicht verifizieren
  - ...
  - eigentlich alles, da einige Server lieber die Verbindung zu machen statt Alerts zu schicken (SChannel)

Fehlermeldungen und Ursachen 5
Certificate verify failed

- Mögliche Ursachen:
  - self-signed
  - fehlende Zwischenzertifikate
  - falscher CA-Store benutzt oder Root-CA nicht im Store (auch unknown_ca Alert)
  - falscher Name
  - falsche lokale Uhrzeit
  - falsches Zertifikat, da ohne SNI geholt
  - SSL Interception im lokalen Netzwerk, evtl. kombiniert mit Pinning

Fehlermeldungen und Ursachen 6
Client mag Server nicht

- ECC Probleme
  elliptic curve routines:EC_GROUP_new_by_curve_name:unknown group
  Client announced ECC Kurven die er garnicht unterstützt und hat ein Problem, wenn Server diese dann wählt (OpenSSL in bestimmter Config: Redhat, CentOS)
- OCSP Probleme
  Ungültige oder fehlende OCSP-Antworten sind leider normal. Die meisten Clients ignorieren Fehler außer bei EV-Zertifikaten. Manche Nutzer schalten sie aber ein.

Aber es funktioniert(e)

gestern noch | woanders

Aber es funktionierte... 1

- vor dem Update des Browsers, System...
  - Mozilla löscht CAs mit RSA1024.
    Betrifft auch LWP (Mozilla::CA) u.a.
  - SSL 3.0 disabled
  - Python, PHP, Ruby, Perl.. fangen an sichere Defaults zu benutzen, d.h. Validierung wo vorher keine war
  - Curl 7.35.0 RC4 disabled
  - Perl LWP IO::Socket::SSL statt Crypt::SSLeay
    Crypt::SSLeay checkt Hostname nicht, benutzt andere Defaults für Ciphers

Aber es funktionierte... 2

- vor Änderungen der Serverkonfiguration
  - SNI jetzt benötigt
  - Ciphers, Protokollversion eingeschränkt
  - Zertifikat geändert,
    Zwischenzertifikate vergessen oder falsch
- gestern, letzte Woche ...
  - irgendwas von den vorhergehenden Sachen
  - Zertifikat abgelaufen bzw. zurückgezogen

Aber es funktioniert ... 1

- in anderen Browsern/Applikationen
  - SNI benötigt
  - fehlende Zwischenzertifikate (gecached oder geholt bei anderen)
  - Desktopbrowser machen Protokoll-Downgrades, andere Applikationen nicht
  - andere Root-CA
- zu Hause
  - blockiert durch Firewall in Firma
  - andere CA wegen SSL Interception

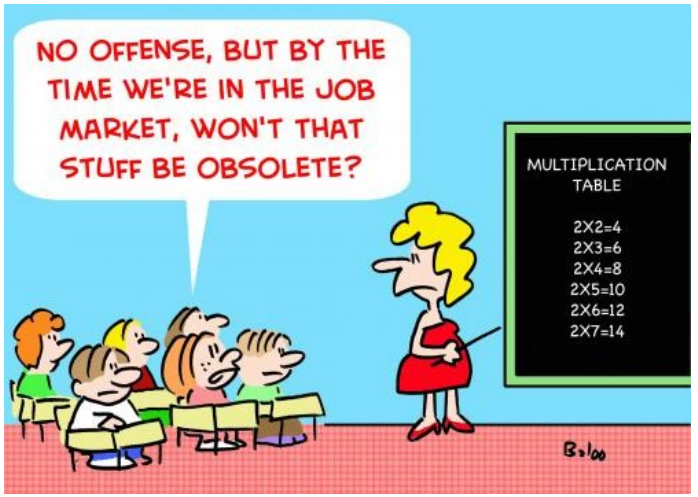
Aber es funktioniert ... 2

- auf (fast) den gleichen Systemen
  - andere Einstellungen zu erlaubten Ciphern oder Protokollen
  - andere Root-CAs
- in anderen Browsern
  - andere Root-CAs (SSL Interception?)
  - andere Ciphern, Protokolle erlaubt
  - unterschiedliche Proxy-Einstellungen

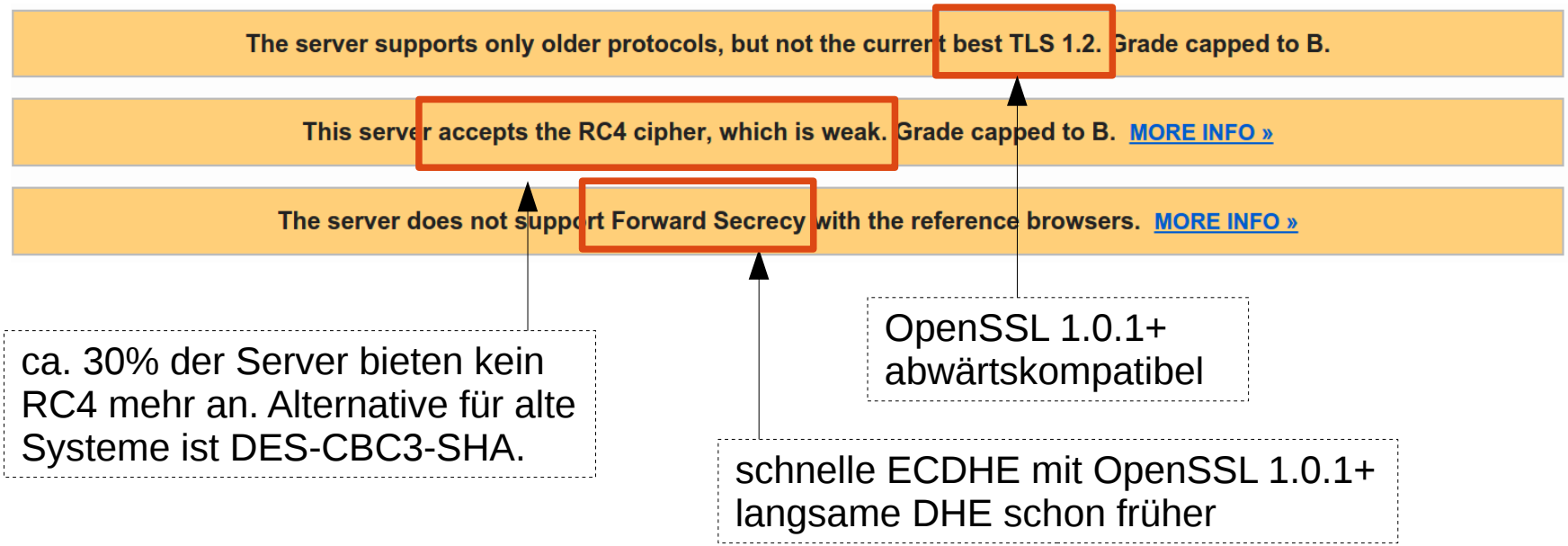
genua

Obsolete Cryptography

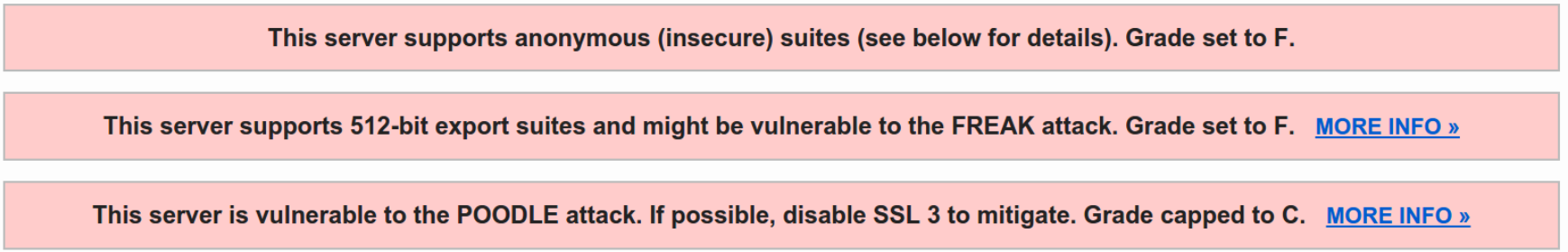
Was in Zukunft kaputt gehen könnte



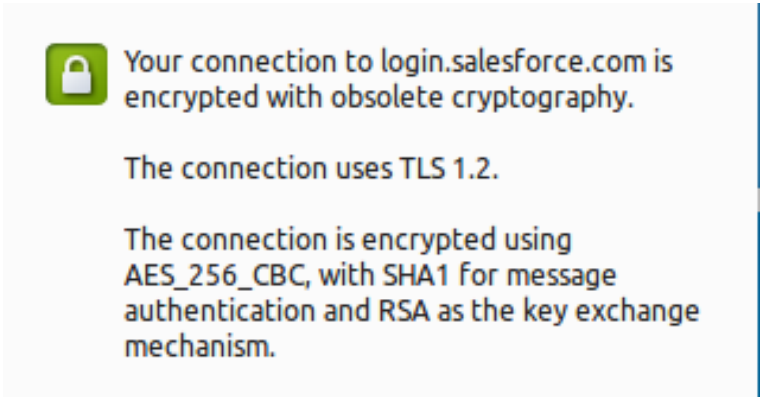
Obsolete Cryptography

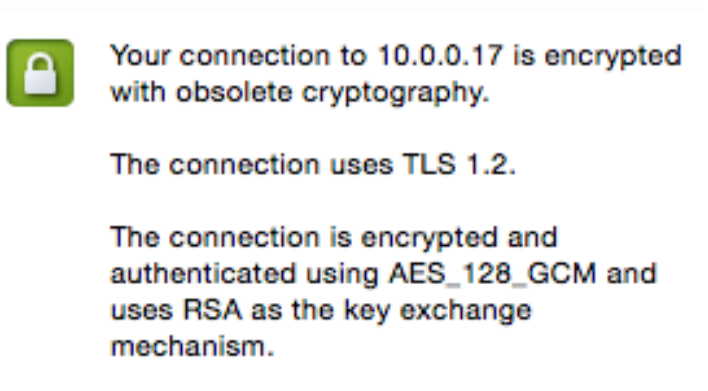


mehr als obsolete, bereits gefährlich (MITM)



Obsolete Cryptography Ciphers (Chrome)





> For a very long time, the first element in the security section of the Connection dialog says something like:
> "Your connection to example.com is encrypted with 128-bit encryption"
>
> This is useless because it makes people think that 256 bits must be better than 128. But AES-256-CBC is 256 bits and it's far worse than AES-128-GCM.

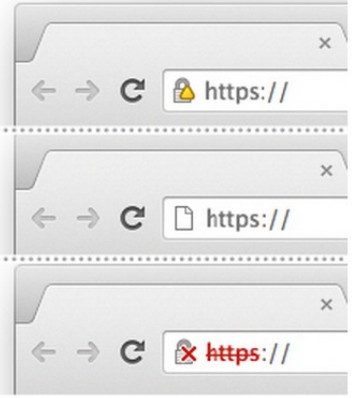
```
net/ssl/ssl_cipher_suite_names.cc  IsSecureTLSCipherSuite

KeyExchg: DHE_RSA | ECDHE_ECDSA | ECDHE_RSA
Cipher:   AES_128_GCM | AES_256_GCM | CHACHA20_POLY1305
Mac:      AEAD

CipherOrder: AES128-GCM-SHA256 vor AES256-SHA384 !!
```

Obsolete Cryptography
SHA-1 Signaturen

genua

Certificate has a weak signature and expires after 2016. Upgrade to SHA2 to avoid browser warnings. MORE INFO »

Chrome Version	Earliest Release Date	SHA-1 Expires Jan.- May 2016	SHA-1 Expires June - Dec. 2016	SHA-1 Expires After 2016
39	3 Nov. 2014	No Change	No Change	Yellow Triangle Over Lock
40	15 Dec. 2014	No Change	Yellow Triangle Over Lock	Blank Page, No Lock
41	26 Jan. 2015	Yellow Triangle Over Lock (sub resources will also trigger icon)	Yellow Triangle Over Lock (sub resources will also trigger icon)	Red X Over Lock (sub resources trigger yellow icon)



SSL

For SSL certificates, Windows will stop accepting SHA1 end-entity certificates by 1 January 2017. This means any time valid SHA1 SSL certificates must be replaced with a SHA2 equivalent by 1 January 2017.

We plan to add a security warning to the Web Console to remind developers that they should not be using a SHA-1 based certificate. We will display an additional, more prominent warning if the certificate will be valid after January 1, 2017, since we will reject that certificate after that date. We plan to implement these warnings in the next few weeks, so they should be appearing in released versions of Firefox in early 2015. We may implement additional UI indicators later. For instance, after January 1, 2016, we plan to show the "Untrusted Connection" error whenever a newly issued SHA-1 certificate is encountered in Firefox. After January 1, 2017, we plan to show the "Untrusted Connection" error whenever a SHA-1 certificate is encountered in Firefox.

Beispiele

Beispiel 1

After much troubleshooting (most of which determined that the mail server only supports TLSv1), I've found that I can connect to the server using openssl:

```
openssl s_client -connect mail.calpoly.edu:993 -tls1
```

as well as with the sockets package in Python 2.7:

```
<ssl.SSLSocket object at 0x7fbab6e7aed8>
```

When I try to connect in Python 3.4, however, I get a Handshake error:

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
```

```
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> ssl_sock = ssl.wrap_socket(sock=sock, ssl_version=ssl.PROTOCOL_TLSv1)
>>> ssl_sock.connect(('mail.calpoly.edu', 993))
```

```
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: SSLV3_ALERT_HANDSHAKE_FAILURE] sslv3 alert handshake failure (_ssl.
```

It seems like Python 3.4 tries to use sslv3 even though I tell it not to.

Kaputter Server?
Versteht kein TLS1.1+?

So unspezifisch wie es nur geht :(

Irrtum:
TLS1.x nutzt SSLv3-Funktionen für den Handshake

Beispiel 1 cont.
- Analyse mit analyze.pl

kein SSL 3.0 (ok)
korrekte Fehlermeldung

```
$ perl analyze-ssl.pl --all-ciphers mail.calpoly.edu:993
 ...
 ! using SSL_version SSLv23:!TLSv1_2:!TLSv1_1:!TLSv1, default ciphers
   -> SSL connect attempt failed error:14077102:
      SSL routines:SSL23_GET_SERVER_HELLO:unsupported protocol
 ...
 ! using SSL_version SSLv23:!TLSv1_2, default ciphers
   -> SSL connect attempt failed because of handshake problems
 * maximum SSL version   : TLSv1 (SSLv23:!TLSv1_2:!TLSv1_1)
 * supported SSL versions with preferred cipher:
   *   TLSv1 RC4-MD5
 * cipher order by       : unknown
 * SNI supported         : ok
 * certificate verified : ok
 * supported ciphers
   *  TLSv1 RC4-MD5
```

Handshake problems
bei TLS 1.1+ statt Antwort mit TLS 1.0
-> BROKEN

nur RC4-MD5 (naja)

Beispiel 1 cont.
- Analyse mit analyze.pl -v3

```
$ perl analyze-ssl.pl -v3 ...
 ...
 <3> handshake failed with HIGH:ALL:eNULL:!RC4-MD5:
     SSL connect attempt failed error:14077410:
     SSL routines:SSL23_GET_SERVER_HELLO:
     sslv3 alert handshake failure
```

Handshake failure statt
no shared ciphers -> BROKEN

ssl.py aus python 3.4,
nicht in python 2.7

```
#   * Disable NULL authentication, NULL encryption, and MD5 MACs for security
#     reasons
_DEFAULT_CIPHERS = (
    'ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+HIGH:'
    'DH+HIGH:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+HIGH:RSA+3DES:ECDH+RC4:'
    'DH+RC4:RSA+RC4:!aNULL:!eNULL:!MD5'
)
```

Fix:
ssl.wrap_socket(...,
    ssl_version = ssl.PROTOCOL_**TLSv1**,
    ciphers = '**RC4-MD5**'

Beispiel 2

I got those commands on `irb`

```
require 'mechanize'
agent = Mechanize.new
agent.get('https://monabo.lemonde.fr/customer/account/forgotpassword/')
```

I got this error:

```
:SSLError: SSL_connect returned=1 errno=0 state=unknown state: sslv3 alert handshake failure
```

I tried on mac, and it works I don't have this error. However, it doesn't work on my computer (running Linux Mint 17).

Wir haben den Hostnamen, wichtig zum Reproduzieren des Problems.

Generischer Fehler :(

ertificates.crt

- Setting this variable:

```
agent.agent.http.ca_file = '/etc/ssl/certs/ca-certificates.crt'
```

- Setting this:

```
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE
```

- Changing ruby version (I'm currently using ruby 2.1.5p275)

Sinnvoll für Debugging von Zertifikatsproblemen, hilft nicht bei Handshake Failure.

Wichtige Info zur genutzten Softwareversion.

Test mit anderem Host geht.

Please notice that I can request `https://google.com` with `mechanize` and that it works, but not `https://monabo.lemonde.fr/customer/account/forgotpassword/`. The fact that it works on mac may suppose that I got a wrong config.

Test mit anderem Client auch.

Beispiel 2 cont.

genua

/_2_1/ext/openssl/lib/openssl/ssl.rb

```
module OpenSSL
  module SSL
    class SSLContext
      DEFAULT_PARAMS = {
        :ssl_version => "SSLv23",
        :verify_mode => OpenSSL::SSL::VERIFY_PEER,
        :ciphers => %w{
          ECDHE-ECDSA-AES128-GCM-SHA256
          ECDHE-RSA-AES128-GCM-SHA256
          ECDHE-ECDSA-AES256-GCM-SHA384
          ECDHE-RSA-AES256-GCM-SHA384
          DHE-RSA-AES128-GCM-SHA256
          ECDHE-RSA-AES256-SHA
          DHE-RSA-AES128-SHA256
          DHE-RSA-AES256-SHA256
          DHE-RSA-AES128-SHA
          DHE-RSA-AES256-SHA
          DHE-DSS-AES128-SHA256
          DHE-DSS-AES256-SHA256
          DHE-DSS-AES128-SHA
          DHE-DSS-AES256-SHA
          AES128-GCM-SHA256
          AES256-GCM-SHA384
          AES128-SHA256
          AES256-SHA256
          AES128-SHA
          AES256-SHA
          ECDHE-ECDSA-RC4-SHA
          ECDHE-RSA-RC4-SHA
          RC4-SHA
        }.join(":"),
        :options => -> {
```

```
perl analyze-ssl.pl  --all-ciphers monabo.lemonde.fr
...
* SSLv23     TLSv1     DES-CBC3-SHA
* TLSv1_2    FAILED: ...:wrong version number
* TLSv1_1    FAILED: ...:wrong version number
* TLSv1      TLSv1     DES-CBC3-SHA
* SSLv3      SSLv3     DES-CBC3-SHA
...
* supported ciphers with SSLv23 handshake
*   TLSv1 DES-CBC3-SHA
```

The server supports only SSLv3 and TLSv1 and only with the cipher DES-CBC3-SHA. This cipher is not included in the default cipher set used by your version of ruby, as you can see in https://github.com/ruby/ruby/blob/ruby_2_1/ext/openssl/lib/openssl/ssl.rb. This setting is strange because from what I know DES-CBC3-SHA (i.e. DES3) is considered more secure then RC4-SHA which they have in their cipher set.

Beispiel 3

I recently got an SSL certificate for my site:

```
https://ram.rachum.com/
```

It works great in browsers. But it fails for `requests`:

```
>>> import requests
>>> requests.get('https://ram.rachum.com')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    requests.get('https://ram.rachum.com')
  File "C:\Python27\lib\site-packages\requests\api.py", line 55, in get
```

Python 2.7, kein SNI < 2.7.9

aber SNI nötig für korrektes Zertifikat

```
$ perl analyze-ssl.pl  ram.rachum.com
 ...
 * SNI supported         : ok
 * certificate verified : ok
 * chain on 184.172.15.234
   * [0/0] ...CN=ram.rachum.com...
 ...
 * chain on 184.172.15.234 without SNI
   * [0/0] ...CN=*.webfaction.com...
 ...
```

Beispiel 4

- Added an ssl certificate from godaddy. It looks like the SSL certificate is installed correctly: https://www.sslshopper.com/ssl-checker.html#hostname=tabularasa-ny.com
- Everything still works, with one caveat.

Issue:

On some machines, people receive the following:

> Unable to make a secure connection to the server. This may be a problem with the server, or it may be requiring a client authentication certificate that you don't have. Error code: ERR_SSL_PROTOCOL_ERROR

Pattern:

The only real pattern I can see is that people with Time Warner in their office cannot load the site.

Hostname gegeben um Problem zu reproduzieren

WTF?

```
perl analyze-ssl.pl  tabularasa-ny.com
 ...
 ! failed SSL upgrade on IP 2600:3c03::f03c:91ff:fe84:5fd
 * certificate verified : ok
 * chain on 104.237.144.128
```

Time Warner Cable was an inaugural participant in World IPv6 Launch.

TWC has rolled out IPv6 to over 90% of its residential network.

genua

Entwicklerecke

genua

- Sichere Defaults nutzen (so vorhanden):
  - Validierung Trust Chain und Hostname, evtl. OCSP/CRL
    Perl: SSL_fingerprint für einfaches Pinning
  - Ciphers, Protokollversion
- SNI Unterstützung sicherstellen
- Vorsicht mit Threads und OpenSSL
  nur ein Thread per SSL-Objekt!
- Achtung: SSL-Sockets sind User-Space!

Kernel-Socket vs. SSL-Socket - fork: I/O in client und server

close(kernel-socket): TCP close
close(ssl-socket):

- sendet SSL close notify alert
- verändert State der SSL Verbindung auf beiden Seiten

```
while (1) {
    my $sslcl = $sslsrv->accept;
    if (fork()) {
        # parent
        close($sslcl);                  # close SSL layer + socket
    } else {                            # besser: undef $sslcl
        # child
        close($sslsrv);
        handle_request($sslcl);         # SSL layer evtl. geschlossen
    }
}
```

Kernel-Socket vs. SSL-Socket - fork und exec

genua

dup(kernel-socket): zwei Ids für den gleichen File-Descriptor
dup(ssl-socket):

- keine Assoziation des neuen FD mit SSL-State
- Schließen des originalen FD verwirft SSL-State

```
my $sslcl = $sslsrv->accept;
if (fork()) {
    undef $sslcl;
} else {
    # child
    open(STDIN,'<&',$sslcl);      # dup kernel socket only
    open(STDOUT,'>&',$sslcl);
    undef $sslcl;
    exec ....                     # read/write to kernel socket
}
```

Kernel-Socket vs. SSL-Socket - fork, start_SSL

accept(kernel-socket) – TCP Accept im Kernel
accept(ssl-socket)

- TCP-Accept im Kernel
- SSL-Handshake im User-Space
- besser SSL-Handshake im forked Prozess machen, insb. bei event-basierten Programmen

```
my $srv = IO::Socket::SSL->new(
    Listen => 10,
    LocalAddr => ...,
    SSL_cert_file => ...
);
while (1) {
    my $cl = $srv->accept;
    if (fork() == 0) {
        # child
        handle_request($cl);
    }
}
```

```
my $srv = IO::Socket::IP->new(
    Listen => 10,
    LocalAddr => ...,
);
while (1) {
    my $cl = $srv->accept;
    if (fork() == 0) {
        # child
        IO::Socket::SSL->start_SSL(
            $cl,
            SSL_server => 1,
            SSL_cert_file => ...
        );
        handle_request($cl);
    }
}
```

Kernel-Socket vs. SSL-Socket - select vs. SSL-Wrapping

genua

- SSL liest in Frames
- Kernel-Read unabhängig von Framegröße: mehrere Frames, Teilframes oder Overlapping sind möglich
- SSL_read liefert max. eine Frame
- Reste können im SSL-State zurückbleiben, d.h. Daten sind verfügbar, auch wenn kein Daten am Kernel-Socket

↓ Kernel-Socket

```
vec($ri,fileno($sslcl),1)=1;
while (select(my $ro = $ri,undef,undef,undef,undef)) {
    sysread($sslcl,$buf,8192) or last;
}
```

↑ SSL-wrapped Socket

SSL_read liest komplette SSL-Frame (max 16k)
-> `$sslcl->pending` benutzen
-> oder immer 16k lesen

Noch Fragen?