genua

IO::Socket::SSL

SSL leicht gemacht (so gut es geht)

genua

- Steffen Ullrich
- seit 1997 primär Perl
- seit 2001 genua mbH – Entwicklung und Forschung IT-Security
- IO::Socket::SSL, Net::SSLGlue*, Net::INET6Glue*, Mail::SPF::Iterator, Net::SIP, Net::PcapWriter, Net::Inspect, Net::IMP*, App::DubiousHTTP, Devel::TrackObjects...

genua

- IO::Socket::SSL:
  SSL-Bibliothek mit Interface analog zu IO::Socket*
- seit 2006 Maintainer:
  non-blocking, SNI Client/Server, sichere Defaults, SSL Interception ...
- meistgenutzte SSL-Bibliothek in Perl
  baut auf Net::SSLeay auf

genua

- Einführung in SSL/TLS
- einfache Nutzung von IO::Socket::SSL in eigenen Programmen: Client, Server
- Bibliotheken, die IO::Socket::SSL nutzen
  und ihre Probleme
- Wichtigkeit von sicheren Defaults
  auch im Vergleich zu Python, Ruby...
  Hall of Shame: explizites Abschalten der Sicherheit

genua

- fortgeschrittene Nutzung
  non-blocking, SNI, Session Caching, SSL Interception...
- typische Fehler bei Nutzung
- praktische Probleme
  Warnings, Bugs in Server und Middleboxes...
- prinzipielle Probleme von SSL-Nutzung
  Tauglichkeit des aktuellen Vertrauensmodells
- Ausblick auf weitere Entwicklung

genua

Einführung in SSL/TLS

SSL: Secure Socket Layer
TLS: Transport Layer Security

SSL 2.0 - SSL 3.0 - TLS 1.0 - TLS 1.1 - TLS 1.2
1995 – 1996 – 1999 – 2006 - 2008

- Ende-zu-Ende Verschlüsselung der (TCP)-Verbindung
  Ciphern werden während Handshake ausgehandelt
- Identifikation der Gegenstelle(n)
  - durch Zertifikate (X.509)
  - Zertifikatshierachien (Vertrauensketten) bis zur Root-CA
  - Root-CA (Certificate Authority) wird vom Browser vertraut
  - Manuelles Vertrauen in self-signed Zertifikat wird ersetzt durch Vertrauen in signierende Root-CA.

genua

- „Aber ich will doch nur Verschlüsselung“
  Häufiges Argument gegen Verifikation des Zertifikats
  - Verschlüsselung warum?
    Damit keiner mithört.
  - Verschlüsselung mit wem?
    Alice: „Mit Bob natürlich.“
  - Und wie kannst du sicher sein, dass du mit Bob und nicht mit Mallory sprichst, der dann weiter mit Bob redet?
    Alice: „Achso.“ (Identifikation)

genua

- Client:
  Hallo [servername], ich kann folgende Ciphern ...
- Server:
  Hallo, ich habe mir folgende Cipher gewählt ...
  Und hier ist mein Zertifikat (+Zertifikatskette)
  optional: Und wer bist du?
- Client:
  Überprüft das Zertifikat: gefällt mir
  optional: und hier ist mein Zertifikat
- Server: Super, lasst uns kommunizieren

Nicht beschrieben: Key Exchange

genua

- wird Herausgeber des Zertifikats vertraut:
  - wenn Root-CA ok
  - wenn Zwischen-CA: erneut Vertrauensfrage
- ist das Zertifikat abgelaufen
- passt der Hostnamen zum Zertifikat
- ist das Zertifikat zurückgezogen
  (ganz schwieriger Punkt)

Einführung in SSL/TLS
- Check Root-CA/Zertifikatskette

genua

```
$ openssl s_client -connect www.google.de:443 \
    -CApath /etc/ssl/certs/
```

Server-Zertifikat

```
0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=google.com
  i:/C=US/O=Google Inc/CN=Google Internet Authority G2
```

Zwischen-CA google

```
1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
  i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
```

Zwischen-CA GeoTrust

```
2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
  i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
```

Einführung in SSL/TLS
- Vergleich Hostnamen mit Zertifikat

genua

```
$ openssl s_client -connect www.google.de:443 ...
$ openssl x509 -text -noout
-----BEGIN CERTIFICATE-----
....

Subject: C=US, ST=California, L=Mountain View,
   O=Google Inc, CN=google.com
...
X509v3 Subject Alternative Name:
  DNS:google.com, DNS:*.android.com,...DNS:*.google.de...
```

genua

Einfache Benutzung* in eigenem Code

*) Transparenter Support für IPv6

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

IO::Socket::SSL – einfache Nutzung - Client

```
use IO::Socket::SSL;
my $cl = IO::Socket::SSL->new('www.google.de:443');

# Alternativ
my $cl = IO::Socket::SSL->new(
    PeerAddr => 'www.google.de:443',
    SSL_version => 'SSLv23:!SSLv2',         # default
    SSL_verify_mode => SSL_VERIFY_PEER,     # default
    SSL_ca_file..|SSL_ca_path => ..         # useful default
    SSL_verifycn_name => 'www.google.de', # default
    SSL_verifycn_scheme => ...,             # useful default
    SSL_cipher_list => ...                  # useful default
    SSL_hostname => 'www.google.de',        # SNI default
);

# Upgrade
my $cl = IO::Socket::INET->new('www.google.de:443');
IO::Socket::SSL->start_SSL($cl,
    SSL_hostname => 'www.google.de',        # für SNI
    SSL_verifycn_name => 'www.google.de', # default zu SSL_hostname
    ...                                     # Rest default ok
);
```

IO::Socket::SSL – einfache Nutzung - Server

```
my $ssl_server = IO::Socket::SSL->new(
    LocalAddr => 'www.example.com:443',
    Listen => 10,                   # setzt implizit SSL_server
    SSL_cert_file => 'cert_chain.pem',
    SSL_key_file => 'key.pem',
    SSL_cipher_list => ...,                 # useful default
    SSL_honor_cipher_order => 1,            # default
    SSL_dh => ...,                          # 2048bit default
    SSL_ecdh_curve => 'prime256v1',         # default
);

# Upgrade Socket
my $cl = $inet_server->accept;
IO::Socket::SSL->start_SSL($cl,
    SSL_server => 1,
    SSL_cert => ..., SSL_key => ...,
    ...                                     # Rest default ok
);
```

IO::Socket::SSL – einfache Nutzung - Clientzertifikat

```
my $cl = IO::Socket::SSL->new(
    PeerAddr => 'want-client-cert.example.com:443',
    SSL_cert_file => ...,
    SSL_key_file => ...,
    SSL_passwd_cb => sub { 'secret-password' }
);
```

genua

Benutzung
durch Bibliotheken

www smtp ftp pop3 imap

Packages mit IO::Socket::SSL - WWW

- LWP: seit 6.0 Default IO::Socket::SSL
  vorher Net::SSL/Crypt::SSLeay
  - sichere Defaults (CA, verifycn_scheme..)
  - nutzt Mozilla::CA als CA-Store
  - Proxy Support kaputt, repariert seit 12/2013.
    Keine Ahnung, wann neue Release 6.0.6 kommt.
    Debian hat die Fixes drin.

Packages mit IO::Socket::SSL - WWW 2

genua

- Furl: sichere Defaults (von IO::Socket::SSL)
  aber eigenwillige Ideen zu Proxies
- HTTP::Tiny: unsichere Defaults
  Manuell verify_SSL=1 und evtl. SSL_ca_file setzen
  Kann nicht mit SSL_ca_path umgehen, versucht CA-File selbständig aus eingeschränkter Liste zu finden
- Mojo::UserAgent: unsichere Defaults
  verifiziert nur, wenn 'ca' (d.h. SSL_ca_file) explizit übergeben, kein Support SSL_ca_path

Packages mit IO::Socket::SSL - SMTP

genua

- Net::SMTP::SSL – SSL direkt
- Net::SMTP::TLS – STARTTLS
  alt, kaputte SSL_version
- Net::SMTP::TLS_ButMaintained
  hardcoded Verifikation ausgeschaltet
- Net::SMTPS – direkt und STARTTLS
  OK, aber kein SSL_verifycn_scheme smtp
- **Net::SSLGlue::SMTP** - vollständig

Packages mit IO::Socket::SSL - FTP

genua

- Net::FTPSSL – nur Verifikation wenn SSL_Client_Certificate gesetzt
- **Net::SSLGlue::FTP** - vollständig

Packages mit IO::Socket::SSL - POP3

- Net::POP3::SSLWrapper – kein STLS
- Net::POP3S
  OK, aber kein SSL_verifycn_scheme
- **Net::SSLGlue::POP3** - vollständig

Packages mit IO::Socket::SSL - IMAP

- Net::IMAP::Client – default Verifikation
  sofern CA zu finden (Linux), sonst keine
- Net::IMAP::Simple – keine Verifikation
  manuelle Verifikation möglich
  kaputte default SSL_version incl. fehlerhafte Doku
- IMAP::Client – benutzt IO::Socket::SSL Defaults
  überschreibt SSL_version mit TLSv1

Sichere Defaults

Sichere Defaults
- wozu ?

genua

The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software

http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf

The real change is that it now **validates** certificates rather than simply displaying their status; previously, it **did not validate the certificate at all**, let alone display its status. The Steam client would happily display HTTPS content from *any* server, regardless of whether the provided SSL certificate had expired, was for the correct domain name, or was signed by a trusted certificate authority. There have been

http://www.highseverity.com/2012/03/valve-fixes-https-vulnerability-in.html

Unsichere Defaults

by web browsers. 40% of iOS-based banking apps tested by IOActive are vulnerable to such attacks because they fail to validate the authenticity of SSL certificates presented by the server.

http://www.theregister.co.uk/2014/02/14/fake_ssl_cert_peril/

Überschreiben sicherer Defaults mit unsicheren Werten

Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security

We introduce MalloDroid, a tool to detect potential vulnerability against MITM attacks. Our analysis revealed that 1,074 (8.0%) of the apps examined contain SSL/TLS code that is potentially vulnerable to MITM attacks. Various

http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf

Sichere Defaults
- Ignoranz, Falschinformationen...

Similarities Between Signed and Self-Signed Certificates

Whether you get your certificate signed by a certificate authority or sign it yourself, there is one thing that is exactly the same on both:

- Both certificates will generate a site that cannot be read by third-parties. The data sent over an https connection or SSL, will be encrypted regardless of whether the certificate is signed or self-signed.

In other words, both types of certificates will encrypt the data to create a secure website.

You can also use self-signed certificates for situations that require privacy, but people might not be as concerned about. For example:

- Username and password forms
- Collecting personal (non-financial) information
- On forms where the only users are people who know and trust you

http://webdesign.about.com/od/ssl/a/signed_v_selfsi.htm

http://xkcd.com/1181/

Sichere Defaults
- andere Programmiersprachen

genua

- PHP: sichere Defaults mit PHP5.6 (alpha)
  sofern OpenSSL CA-Defaults nutzbar (unix)
  oder globale CA per php.ini konfiguriert
  https://wiki.php.net/rfc/tls-peer-verification
- Python: unsichere Defaults mit builtin
  keine Verifikation in urllib, urllib2, urllib3 – in Diskussion:
  http://lwn.net/Articles/582065/
  fehlerhafter Wildcard-Match (*.*.com), gefixt in 3.4
  Sichere Defaults in requests Library (Mozilla CAs)
- Ruby: sichere Defaults, aber..
  Check gegen OpenSSL CA-Defaults, Windows :)
  fehlerhafter Wildcard-Match (*.*.com)
- Java: sichere Defaults (eigener CA-Store)

Sichere Defaults
- History IO::Socket::SSL

genua

- 2008
  1.14 SSL_verifycn_* (default off, buggy bis 1.25)
- 2009
  1.23 Warnung, wenn keine nutzbaren CA, statt silent pass
- 2012
  1.67 SSL_honor_cipher_order gegen BEAST (opt.)
  1.68 per Default kein SSL 2.0
  1.79 Warnen, wenn Default SSL_VERIFY_NONE genutzt wird
- Q3.2013
  1.950 SSL_VERIFY_PEER per Default
  1.951 Default CA = OpenSSL-CA builtin
- Q4.2013
  1.956 kein MD5 per Default, kein ADH, 3DES vor RC4, TLS1.[12]
  1.956 Server: Forward Secrecy mit DHE und ECDHE default
  1.956 Server: SSL_honor_cipher_order default 1
  1.956 Client: hostname Checks überarbeiten (RFC6215)
- Q1.2014
  1.967 Client: neue Option SSL_fingerprint (für self-signed)
  1.967 Client: Default Hostname Verification scheme
  1.968 Client: Default CA auch auf Windows (Mozilla::CA)
  1.969 set_args_filter_hack zum Überschreiben kaputter Nutzung

Sichere Defaults
- Hall Of Shame

genua

- OTRS, FosWiki
  Bugreports wegen Warnings bei Nutzung insecure Defaults (verify_none) -> Empfehlen Downgrade IO::Socket::SSL, spätere Versionen schalten Verifikation beim Mailzustellung explizit aus
- Explizites Ausschalten Verifikation bei:
  Heineko Mail API (0.2.16, 2014)
  File::HTTP (0.91, 2013)
  Net::SMTP::Bulk (0.16, 2014)
  Net::TLS::SMTP_ButMaintained (0.24 2013)

Sichere Defaults
- Hall of kleinere Übel

genua

- Mail::Sender (0.8.22 2012), IMAP::Client (2006)
  SSL_version fix auf TLSv1 d.h. kein TLS 1.1 oder TLS 1.2
- perl-ldap (0.60 2014)
  SSL_version Default SSLv23 d.h. auch SSL 2.0
  default SSL_cipher_list 'ALL', also auch ADH und LOW
- keine Verifikation, wenn nicht explizit CA angeben
  d.h. keine Nutzung IO::Socket::SSL/OpenSSL Defaults
  Event::RPC (1.05 2014)
  Mojolicious (4.88 2014)
  HTTP::Tiny (0.043 2014, sucht nach eigenen CA)
  Net::IMAP::Client (0.9504 2014)

Sichere Defaults
- Erzwingen

genua

```
use IO::Socket::SSL 1.969;
use Module::Overriding::Secure::Defaults;

# alle Defaults erzwingen
IO::Socket::SSL::set_args_filter_hack(
   'use_defaults');

# Alternativ: selber filtern
IO::Socket::SSL::set_args_filter_hack( sub {
   my ($is_server,$args) = @_;
   $args->{SSL_verify_mode} = SSL_VERIFY_PEER;
   ...
});
```

Sichere Defaults
- Einbinden in eigene Module

genua

```
package MyModule;
use IO::Socket::SSL;

# am besten sich IO::Socket::SSL um gute Defaults
# kümmern lassen
sub startssl {
    my ($self,%args) = @_;
    my %sslargs = map { $_ => delete $args{$_} }
        grep { /^SSL_/ } keys %args;
    IO::Socket::SSL->start_SSL( $self->{socket},
        %sslargs );
}

# evtl. Checken, ob IO::Socket::SSL CA-Store hat
if ( ! defined &IO::Socket::SSL::default_ca
    or ! IO::Socket::SSL::default_ca()) {
    # selber auf die Suche nach CA-Store machen
    ...
}
```

Fortgeschrittene Nutzung

genua

- Problem:
  self-signed Zertifikat oder
  CA-Trust egal, wir wollen genau dieses Zertifikat
  (Hinzufügen self-signed non-CA zu CA-Store hilft nicht)

```
# Identifikation gegen Fingerprint checken
IO::Socket::SSL->new(
    PeerAddr => '192.168.0.1',
    SSL_fingerprint => 'sha256$31453a32393a31303...',
);

# Fingerprint ermitteln
IO::Socket::SSL->new(
    PeerAddr => '192.168.0.1',
    SSL_verify_mode => SSL_VERIFY_NONE, # hab ja keine CA
)->get_fingerprint(['sha256'|'sha1'|'md5'...]);
```

Fortgeschrittene Nutzung:
- SNI: mehrere Server hinter einer IP

genua

```
# Client - implizit
IO::Socket::SSL->new('www.google.de:443');

# Client – explizit
IO::Socket::SSL->start_SSL( $socket,
    SSL_hostname => 'foo.example.com'
);


# Server
IO::Socket::SSL->new(
    LocalAddr => '1.2.3.4:443',
    Listen => 10,
    SSL_cert_file => {
        'foo.example.com' => 'foo.pem',
        'bar.example.com' => 'bar.pem',
        ''                => 'default.pem',
    },
    SSL_key_file => { ... }
);
```

Fortgeschrittene Nutzung:
- forking Server

genua

```
# Am besten startet man einen INET-Server und macht
# den SSL-Upgrade erst nach dem fork(). Auf diese Weise
# findet der blockierende SSL-Upgrade im Kind statt,
# während der Masterprozess weiter arbeiten kann

my $server = IO::Socket::INET->new(
    LocalAddr => '0.0.0.0:1234',
    Listen => 10
);
while (1) {
    my $cl = $server->accept or next;
    defined( my $pid = fork()) or die “fork failed: $!“;
    $pid and next; # Master

    # Kind
    close($server);
    IO::Socket::SSL->start_SSL($cl,
        SSL_server => 1,
        SSL_key => ..., SSL_cert => ...
    ) or die “SSL accept failed: $SSL_ERROR“;
    ...
}
```

genua

DON'T !

- wenn es wirklich sein muss:
  - neuere Net::SSLeay Versionen sollten halbwegs thread-safe sein
  - Laden von IO::Socket::SSL global, nicht in jedem einzelnen Thread

Fortgeschrittene Nutzung:
- non-blocking

genua

- Analog zu IO::Socket, aber...
  - Lesen kann vorheriges Schreiben benötigen: `SSL_WANT_WRITE`
  - Schreiben kann vorheriges Lesen benötigen: `SSL_WANT_READ`
  - Connect und Accept lesen und schreiben beim SSL-Handshake: `SSL_WANT_{READ,WRITE}`
  - Lesen liest evtl. aus vorhandenem Buffer, nicht vom Socket: `pending()` oder >= 16k lesen
- perldoc IO::Socket::SSL “Non-blocking I/O“
- nicht in Windows :(

Fortgeschrittene Nutzung:
- non-blocking read/write

genua

```
# nonblocking read
my $rv = $socket->sysread(...);
if ( ! defined $rv ) {
    die “$!,$SSL_ERROR“ if ! $!{EAGAIN};   # Fehler
    if ( $SSL_ERROR == SSL_WANT_READ ) {
        ... warten auf socket readable (select loop...)
    } elsif ( $SSL_ERROR == SSL_WANT_WRITE ) {
        ... warten auf Socket writable ...
    }
}
# ACHTUNG! es können Daten vorliegen, ohne das Socket
# readable in select-Maske. Checken mit
# $socket->pending() oder nur readsize >= 16k benutzen

# nonblocking write
my $rv = $socket->syswrite(...);
... Rest analog zu read, pending nicht relevant ...
```

Fortgeschrittene Nutzung:
- non-blocking connect (Client)

genua

```
# nonblocking connect
# erst non-blocking INET-Connect machen
# dann non-blocking SSL-Upgrade

# Upgrade Socket
IO::Socket::SSL->start_SSL( $socket,
    SSL_startHandshake => 0 );
while (1) {
    $socket->connect_SSL && last; # connect fertig
    die “$!,$SSL_ERROR“ if ! $!{EAGAIN};   # Fehler
    if ( $SSL_ERROR == SSL_WANT_READ ) {
        ... warten auf socket readable (select loop...)
    } elsif ( $SSL_ERROR == SSL_WANT_WRITE ) {
        ... warten auf Socket writable ...
    }
}
```

Fortgeschrittene Nutzung:
- non-blocking accept (Server)

genua

```
# nonblocking accept
# erst non-blocking INET-Accept machen
# dann non-blocking SSL-Upgrade

# Upgrade Socket
IO::Socket::SSL->start_SSL( $socket,
    SSL_server => 1,
    SSL_key => ... , SSL_cert => ..
    SSL_startHandshake => 0 );
while (1) {
    $socket->accept_SSL && last; # accept fertig
    die “$!,$SSL_ERROR“ if ! $!{EAGAIN};   # Fehler
    if ( $SSL_ERROR == SSL_WANT_READ ) {
        ... warten auf socket readable (select loop...)
    } elsif ( $SSL_ERROR == SSL_WANT_WRITE ) {
        ... warten auf Socket writable ...
    }
}
```

Fortgeschrittene Nutzung:
- Session Caching Server

genua

```
# Session Caching verringert Overhead SSL-Handshake
# bei wiederholten Verbindungen zum gleichen Server

# Session Cache Server – implizit per Context
my $ctx = IO::Socket::SSL::Context->new(
    SSL_server => 1, SSL_key => .., SSL_cert => ...,
    # Cachegröße bei Bedarf beeinflussen
    SSL_create_ctx_callback => {
        my $c = shift;
        Net::SSLeay::CTX_sess_set_cache_size($c,128);
    }
);
...
IO::Socket::SSL->start_SSL($socket,
    SSL_server => 1,
    SSL_reuse_ctx => $ctx );

# funktioniert nicht, wenn Server forked !
```

Fortgeschrittene Nutzung:
- Session Caching Client

genua

```
# Session Cache Client – per Context
my $ctx = IO::Socket::SSL::Context->new(
    SSL_session_cache_size => 1000, ...
);
...
IO::Socket::SSL->start_SSL($socket,
    SSL_reuse_ctx => $ctx,
    SSL_session_key => ...  # default IP:Port
);

# Session Cache Client – über Context hinweg
my $cache = IO::Socket::SSL::Session_Cache->new(1000);
...
IO::Socket::SSL->start_SSL($socket,...,
    SSL_session_cache => $cache );

# globale Nutzung des gleichen Cache
IO::Socket::SSL::set_default_session_cache($cache)
```

genua

MAN-IN-THE-MIDDLE

```
use IO::Socket::SSL::Intercept;
my $mitm = IO::Socket::SSL::Intercept->new(
    proxy_cert_file => 'proxy_cert.pem',
    proxy_key_file  => 'proxy_key.pem',
    ...
);
...
my $to_client = $to_client_listener->accept;
my $to_server = IO::Socket::SSL->new(...);
my ($cert,$key) = $mitm->clone_cert(
    $to_server->peer_certificate);
IO::Socket::SSL->start_SSL($to_client,
    SSL_cert => $cert, SSL_key => $key );

# jetzt Daten zwischen $to_client und $to_server
# austauschen und dabei mitlesen, manipulieren...
```

genua

Typische Fehler und Probleme

Typische Fehler

```
# SSL_verify_mode erwartet Zahl, nicht String
- SSL_verify_mode => 'SSL_VERIFY_PEER'
+ SSL_verify_mode => SSL_VERIFY_PEER
```

```
# SSL_version verträgt nicht mehrere Versionen
#    SSL_version => 'TLSv1 SSLv3'
#    -> “invalid SSL_version specified at ....“
# DER BUG IST NICHT IN IO::Socket::SSL, der benutzte
# Syntax verstieß schon immer gegen die Dokumentation,
# wurde nur früher nicht angemeckert
# Ergebnis früher: nur TLSv1 benutzt, SSLv3 nicht!

- SSL_version => 'TLSv1 SSLv3'
+ SSL_version => 'SSLv23:!SSLv2:!TLSv1_1:!TLSv1_2'
```

Typische Fehler

 # impliziter SSL_verify_mode von none bis 1.950

```
*****************************************************************
 Using the default of SSL_verify_mode of SSL_VERIFY_NONE for client
 is depreciated! Please set SSL_verify_mode to SSL_VERIFY_PEER
 together with SSL_ca_file|SSL_ca_path for verification.
 If you really don't want to verify the certificate and keep the
 connection open to Man-In-The-Middle attacks please set
 SSL_verify_mode explicitly to SSL_VERIFY_NONE in your application.
*****************************************************************
```

-> Upgrade oder SSL_verify_mode explizit setzen

Typische Probleme

`SSL3_GET_SERVER_CERTIFICATE:certificate verify failed`

- CA nicht im CA-Store
- Self-signed -> SSL_fingerprint
- Hostname falsch:
  www.foo.bar.com matcht nicht auf *.bar.com
  bar.com matcht nicht auf *.bar.com
- Gegencheck mit Browser und openssl s_client
  openssl verhält sich bei speziellen Chains anders als NSS
- Debugging:
  perl -MIO::Socket::SSL=debug100
  wireshark
  stackoverflow, mail sullr@cpan.org :)

`Verbindung hängt, Connection reset o.ä`

- diverse kaputte Server, Accelerator... (F5 BIG-IP u.a.)
- Versuch SSL_version Downgrade: SSLv23:!TLSv1_1

genua

Prinzipielle Probleme

mit

derzeitigem SSL

SSL – prinzipielle Probleme

- Firefox derzeit 172 CA, Windows 377, OS X 207
  Will ich denen wirklich allen vertrauen?
  http://www.heise.de/newsticker/meldung/Ein-Drittel-aller-Zertifikats-Herausgeber-nur-Security-Ballast-2139451.html
- 2011 Komprimittierung DigiNotar, Comodo
  DigiNotar kaputt, Comodo too big to fail
  Alle Browser aktualisiert, da Revocation nicht klappt
- jede Zwischen-CA kann beliebige Zertifikate ausgeben
- 2012 Trustwave verkauft Zwischen-CA für Interception
  wir tuns auch nie wieder
  http://www.heise.de/security/meldung/Trustwave-verkaufte-Man-in-the-Middle-Zertifikat-1429722.html
- 2013 französische Zwischen-CA gibt Zertifikate für google etc aus
  Tschuldigung, war aus Versehen
  Alle Browser aktualisiert, da Revocation immernoch nicht klappt

Ausblick

Ausblick, TODOs

- wichtig:
  - OCSP
- vielleicht:
  - Prefixcheck bei Wildcards (*.co.uk verbieten)
  - CA-Store von Windows
  - Zertifikatspinning, Blacklist
  - opt. Support in Net::{SMTP,FTP,POP3}
  - SSL_cipher_list => ':DEFAULT:!RC4'
  - SSL_verify_mode => 'I-DONT-CARE-ABOUT-SECURITY'
-

genua

ENDE

Danke auch an Net::SSLeay,
Bugreporter, cpantesters und Nutzer

Bugreports, Fragen, Patches.. willkommen:
https://github.com/noxxi/p5-io-socket-ssl
stackoverflow.com
sullr@cpan.org
rt.cpan.org