SSL und Perl

Überblick

- kurze Einführung in SSL
- Stand der Unterstützung von SSL in Perl

was ist überhaupt SSL?

- "Secure Socket Layer", Nachfolger TLS (Transport Layer Security), welches fast das gleiche ist
- stellt verschlüsselte Verbindung zwischen zwei Endpunkten her
- optionaler Austausch von Zertifikaten, um Identität zu überprüfen

was sind Zertifikate?

- "elektronischer Ausweis"
- enthält Informationen über einen Endpunkt (z.B. Hostnamen, IP-Adressen...)
- sollte (elektronisch) unterschrieben sein von einer glaubwürdigen Instanz (CA = Certificate Agency, Herausgeber)
- hat eine Gültigkeitsdauer
- kann widerrufen werden
- Spezialfall selbstsignierte Zertifikate: nur der Inhaber des Zertifikates hat unterschrieben ("Glaube mir, ich bins")

Überprüfung von Zertifikaten

- passt das Zertifikat zu demjenigen, der es mir präsentiert (Identität)?
- vertraue ich der CA (dem Herausgeber), die es unterschrieben hat? Dazu haben die Browser eine Liste verbreiteter CAs eingebaut.
- bei selbstsignierten Zertifikaten: Glaube ich, das der mir Unbekannte mich nicht anlügt?
- ist das Zertifikat abgelaufen?
- ist das Zertifikat widerrufen worden?

Hintergrund dieses Vortrages

- 11/07: CVE-2007-5162: "Ruby Net::HTTPS library does not validate server certificate CN"
- Perl macht es doch auch nicht anders und keiner regt sich auf? Wo ist das Problem?
- Perl-Workshop 02/2008: "Bist du der IO::Socket::SSL Maintainer?"
- Ja, aber eigentlich ist das doch nicht die Aufgabe von IO::Socket::SSL sondern des übergeordneten Layers (HTML, SMTP...)
- Aber: die Nutzer von IO::Socket::SSL checken die Zertifikate selber nicht

Warum überhaupt überprüfen?

- "ich will doch nur verschlüsseln, warum brauche ich die Unterschrift einer CA?"
- "Mozilla SSL policy bad for the Web": Abschreckende Firefox3 Warnungen bei selbstsignierten Zertifikaten hindern Verbreitung von Verschlüsselung
- Problem: kein Schutz vor Man-In-The-Middle Attacken ohne Überprüfung von Zertifikaten

was ist Man-In-The-Middle?

- Endpunkte: Alice, Bob. Angreifer Mallory
- Alice will SSL Verbindung zu Bob aufbauen, landet aber bei Mallory.
- Mallory baut SSL Verbindung weiter zu Bob auf und bekommt von diesem ein Zertifikat präsentiert.
- Um die Verbindung unverschlüsselt lesen oder manipulieren zu können muss er die Daten von Bob entschlüsseln und für Alice erneut verschlüsseln.
- Da Mallory das Secret von Bobs Zertifikat nicht hat, erstellt er ein neues, was wie Bobs aussieht (aber andere Unterschrift)
- Nur wenn Alice das Zertifikat gründlich verifiziert findet sie die falsche Unterschrift.

ist das wirklich eine Gefahr?

- Verbindungen umleiten mittels ARP Spoofing im lokalen Netz
- Verbindungen umleiten mittels DNS Spoofing im Internet (z.B. CVE-2008-1447)
- Exit Node im Tor Netzwerk
- Tools wie z.B. Ettercap zur Automatisierung des Angriffs sind frei verfügbar
- wird offensichtlich auch praktiziert: Bugreport, der auf MITM in the wild schließen läßt

Überprüfung der Identität

- Check gegen commonName und/oder subjectAltNames
- RFC2818 (http), RFC3920 (xmpp) - wenn subjectAltNames existieren wird commonName nicht benutzt, subjectAltName kann auch Wildcard, zB *.example.com aber auch host*.example.com sein.
- RFC4513 (ldap), RFC2995 (pop,imap...), RFC4642 (nntp) - sowohl commonName wie auch subjectAltName werden überprüft, subjectAltName kann auch *.example.com sein, aber nicht host*.example.com
- RFC3207 (smtp) - man soll checken ob man dem Zertifikat vertraut, ohne Details zum Vorgang zu geben
- RFC4217 (FTP über TLS) empfiehlt check wie RFC2818, legt sich aber nicht wirklich fest

Zertifikate checken hilft nicht immer

- "(Un)trusted Certificates": andere vertraute CA stellt Zertifikat für gleiche Webseite aus
- Tricks mit subjAltNames zum Akzeptieren von ungültigen Zertifikaten
- SSL umgehen durch Umschreiben der Webseiten bei MITM: sslstrip
- MD5 Kollisionen bewusst nutzen um neue Zertifikate zu erzeugen: "MD5 considered harmful today" (25C3)
- Zertifikat knacken wegen schwachem Zufall: CVE-2008-0166

SSL und Perl

Crypt::SSLeay/Net::SSL

- nur bei LWP benutzt
- liefert bei Bedarf commonName zum Check
- wird bei LWP genutzt um commonName gegen Hostname zu checken, sofern man Pseudo-Header 'If-SSL-Cert-Subject' setzt
- weitere Entwicklung wohl nur, um es compilierbar zu halten

IO::Socket::SSL

- basiert auf Net::SSLeay (d.h OpenSSL)
- unterstützt
  - Checks gegen commonName und subjectAltNames
  - Berücksichtigung von Wildcards
  - verschiedene Verifikationsschemas
  - internationale Domainnames
  - Checks gegen IP
- checkt aber auch nichts, wenn nicht gesagt was genau gescheckt werden soll (da keine klare Regel, wie in allen Fällen zu checken)
- braucht daher offizielle Unterstützung im Caller oder Hacks, um diese Unterstützung nachträglich einzubauen

IO::Socket::SSL Client

```
# ohne Verifikation
$cl = IO::Socket::SSL->new( 'signin.ebay.de:443' );

# mit Verifikation
$cl = IO::Socket::SSL->new(
  PeerAddr => 'signin.ebay.de:443',
  SSL_verify_mode => 1,
  SSL_ca_path => '/etc/ssl/certs',
  SSL_verifycn_scheme => 'www',
);
print $cl "GET / HTTP/1.1\r\n...."
```

IO::Socket::SSL Server

```
$srv = IO::Socket::SSL->new(
  LocalAddr => 'signin.ebay.de:443',
  Listen => 10,
  SSL_cert_file => ...,
  SSL_key_file => ...,
);
$cl = $srv->accept;
<$cl>
```

weiterhin

- Crypt::NSS, Net::NSS::SSL Layer über Netscape Security Services (v0.4 11/08)
- Crypt::MatrixSSL Layer über MatrixSSL.org Bibliothek (v1.86 01/09)

LWP (https)

- Problem: LWP nimmt via Net::SSL entweder Crypt::SSLeay oder IO::Socket::SSL, wobei es ersteres bevorzugt
- Daher Hack: Net::SSLGlue::LWP
  - forciert Nutzung von IO::Socket::SSL, da nur dieses richtiges Zertifikatschecking bietet
  - forciert Zertifikatscheck entsprechende RFC2818
  - leider bietet LWP keinen Mechanismus Optionen an den Socket zu übergeben, daher verbindungsspezifische Optionen nur über `local`

Net::SSLGlue::LWP - Beispiel

```
use Net::SSLGlue::LWP SSL_ca_path => '/etc/ssl/certs';
use LWP::Simple;
get( 'https://signin.ebay.de' );
```

LWP und https Proxy

- LWP hat ein einzigartiges Verständnis von https_proxy
- statt einem CONNECT wird ein plain text Request für eine https:// Resource an den Proxy gesendet
- die Verbindung mit dem Proxy ist dann unverschlüsselt
- daher keine Verifikation von Zertifikaten möglich
- nur wenige Proxies unterstützen das
- Crypt::SSLeay/Net::SSL hatte Workaround mittels HTTPS_PROXY Environmentvariable (dazu muss env_proxy in LWP aus sein)
- IO::Socket::SSL hat diesen Workaround nicht
- Net::SSLGlue::LWP fixt stattdessen https_proxy, sodaß die Variable wie in anderen Programmen funktioniert

SSL und SMTP

- Net::SMTP::TLS, startet plain und macht dann mit STARTTLS weiter.
  - Seit 01/2006 keine Änderungen mehr (aber auch keine Bugreports), letzte Version 0.12.
  - Nutzt IO::Socket::SSL, macht aber keine Zertifikatschecks und beruht nicht auf Net::SMTP sondern reimplementiert alles :(
- Net::SMTP::SSL, welches einfach die Abhängigkeit von IO::Socket::INET gegen eine Abhängigkeit von IO::Socket::SSL austauscht und auch keine Zertifikatschecks macht.
- Daher Hack: Net::SSLGlue::SMTP
  - erweitert Net::SMTP um STARTTLS Kommando (wie Net::SMTP::TLS)
  - und um SSL von Beginn an (wie Net::SMTP::SSL)
  - in beiden Fällen mit korrektem Zertifikatscheck

Net::SSLGlue::SMTP Beispiel

```
use Net::SSLGlue::SMTP;
$smtp_ssl = Net::SMTP->new(
  'mail.gmx.net', SSL => 1 ); # SSL Port 465
$smtp = Net::SMTP->new(
   'mail.gmx.net' ); # Plain Port 25
$smtp->startssl; # switch to SSL
```

- Achtung! Da für SMTP kein Standard für das Checken der Zertifikate existiert akzeptieren Mailclients evtl. andere Zertifikate als Net::SSLGlue::SMTP. In diesem Falle `SSL_verifycn_scheme` anpassen.

```
# mail.gmx.de in subjectAltName, in cn mail.gmx.net
$smtp = Net::SMTP->new( 'mail.gmx.de',
   SSL => 1,
   SSL_verifycn_scheme => 'http' )
```

Net::LDAP

- nur hier aufgenommen, weil der ursprüngliche Request zur Unterstützung sich auf Net::LDAP bezog.
- nutzt IO::Socket::SSL, aber macht natürlich auch keinen Zertifikatscheck
- Daher Hack: Net::SSLGlue::LDAP
  - forciert SSL_verify_scheme ldap, sofern über den Net::LDAP Mechanismus verify auf True gesetzt ist
  - ungetestet

weitere Module

- Net::POP, Net::NNTP könnte man analog zu Net::SMTP mit einem Net::SSLGlue::* aufbohren
- implicites FTPS (d.h Connection mittels SSL von Anfang an über speziellen Port, analog zu HTTPS, Ports 990/989) könnte über Aufbohren von Net::FTP gemacht werden, allerdings gibt es wohl keine RFC dazu und kein Schema, wie das Zertifikat zu verifizieren ist.
- explizites FTPS (RFC4217) ist wesentlich schwieriger, da hier einiges an neuen Kommandos unterstützt werden muss. Es scheint einen Versuch Net::FTPSSL zu geben, der seit ein paar Jahren mal wieder auflebt (v0.6, 02/09)

noch offene Fragen

- Standard-CAs
  - unter Linux i.A. /etc/ssl/certs
  - unter BSD ähnlich
  - unter Windows ??
- zurückgerufene Zertifikate
  - kann mit heruntergeladenen CRLs umgehen
  - keine Unterstützung von OCSP, welches in den neueren Zertifikaten benutzt wird

Perl im Vergleich

Ruby

- nach Fix wegen CVE-2007-5162 wird gewarnt, wenn keine Überprüfung gemacht wird (das ist der Default)
- Net::HTTPS benutzt OpenSSL::SSL::SSLSocket::post_connection_check, welcher das http scheme (RFC2818) implementiert.
- Net::SMTP benutzt die gleiche Routine, d.h. es wird das http scheme zur Verifikation bei SMTP benutzt.
- Net::POP und Net::IMAP benutzen ebenfalls das http scheme, das ist falsch.

Python

- httplib, http.client: "This does not do any certificate verification!"
- scheint nicht mal die Möglichkeit zu geben, einfach eigene Checks einzubauen (d.h man kann zwar Clientzertifikat übergeben, aber nicht beeinflussen, ob und wie Serverzertifikat verifiziert wird)
- imaplib, poplib... scheinen ebenfalls keinerlei Checks vorzunehmen
- ssl: stellt nötige Infos (subjectAltNames, commonName) zur Verfügung, macht aber selber nichts damit und bietet auch keine Convinience Funktion zum Check an

Java

- in Applets/JRE nur commonName checking
- aber in JSSE (Java Secure Socket Extension) wohl für HTTPS korrekt implementiert
- aber keine Ahnung, wie es mit den anderen Verifikationsarten (LDAP..) aussieht

weitere Features von IO::Socket::SSL

- nonblocking Sockets
- start_SSL (Upgrade von unverschlüsselt) und stop_SSL (Downgrade auf unverschlüsselt)
- transparente IPv6 Unterstützung
  - führt dazu, das evtl IPv6 bevorzugt wird, wenn DNS AAAA und A Records liefert
  - kann Problem geben, wenn aber IPv6 Verbindung nicht funktioniert oder DNS spinnt
- Unterstützung für Windows (alle Tests laufen, aber keine Unterstützung für nonblocking Sockets)

Danke!