

A Benevolent Dictator: Improving web browser HTTPS security with MITM proxy

(C) Alex Smirnoff (arkenoi@gmail.com)



March 2010

Table of Contents

1. [Introduction](#)
2. [Certificate Management UI Problem](#)
3. [Implementation and configuration security issues](#)
4. [X509v3 extensions and trusted path validation](#)
5. [A traditional SSL man-in-the-middle use: HTTPS content inspection](#)
6. [Design, implementation and deployment](#)
7. [PKI problems for “Big Bad Internet”](#)
8. [Summary](#)
9. [Show Me The Code](#)

Introduction

The HTTPS protocol, being standard HyperText Transfer Protocol widely used for World Wide Web augmented with SSL¹ (Secure Sockets Layer, designed, implemented and published by Netscape in 1995) was widespread almost as long as WWW itself (at least, WWW as we know it). It is based on traditional public key cryptography principles and provides mutual (well, mostly unidirectional, server-only in practice) authentication, encryption and tampering protection for all data being sent using that protocol. The Internet's current PKI infrastructure, including root Certificate Authorities, was created together with introduction of this protocol.

The ubiquitous encryption (and HTTPS) is definitely good for us and should be welcome. Despite that², there are still problems that should be resolved:

- Certificate management interface does not meet real life user requirements.
- SSL/TLS implementation and/or configuration may be flawed.
- Contemporary PKI standards are not supported as deserved to be.
- Encrypted data aren't accessible using content inspection tools

This article is based on my RISSPA-12 presentation. The title itself was considered controversial: would «implementation problems» be an appropriate name for user interface inconsistency and fundamental protocol limitations? The major part of the analysis is dedicated to the certificate management which is not a part of HTTPS at all. Anyway, from user's point of view, all these problems belong to HTTPS, so the name is correct (despite it may be not quite technologically correct).

“Man-in-the-middle” decrypting and inspecting proxies are quite common now. The main (and almost only) purpose of those is content analysis - the last of four problems listed above. Usually there are no certificate management functions implemented.

It could be said, arguably, that any deviation from “peer to peer” SSL protocol design and functioning may be considered “violation of ideology” and potential harm to confidentiality, integrity and mutual authentication. Well, real world HTTP deployment has outgrown its 15-year original goal to transfer simple web pages. Protection requirements were changed as well, shifting from protecting low-volume post form data to continuous data transfers as the part of complex web2.0 applications. So if typical “circa 1995” HTTPS application involved “sending this credit

1 We use the «SSL» word referring to HTTPS crypto transport, despite actually it is more likely to be TLS. The TLS itself contains some datagram-specific extensions that are out of scope of this article. The TLS subset used within HTTPS is close enough to SSL 3.0.

2 ..and the fact that HTTPS is almost as old as HTTP itself

card number form over an encrypted channel”, nowadays we have to care if Gmail web interface may become virus attack vector, for example, and if the user is protected from all traditional and emerging web threats (not only confidentiality problems) regardless whether he is using HTTPS or not.

The "Man-In-The-Middle" algorithm, implemented by proxy is quite simple:

1. Client connects to the proxy server (via TCP) and sends “CONNECT” request to open channel to destination server
2. Proxy server establishes TCP and SSL connection to destination server
3. Proxy server retrieves and verifies destination server's certificate¹.
4. If something is wrong (policy violation is detected), connection is dropped and processing ends here.
5. Proxy server generates, and (optionally, if the server certificate is OK) signs proxy certificate using own public key, destination server certificate info and builtin certificate authority.
6. Proxy server accepts client SSL handshake using newly generated proxy certificate.
7. Proxy server accepts client's HTTP request, passes it to content inspection engine and relays to the destination server if it is valid and permitted by the policy.
8. Proxy server accepts server HTTP response, passes it to content inspection engine and relays to the client if it is valid and permitted by the policy.

Certificate Management UI Problem

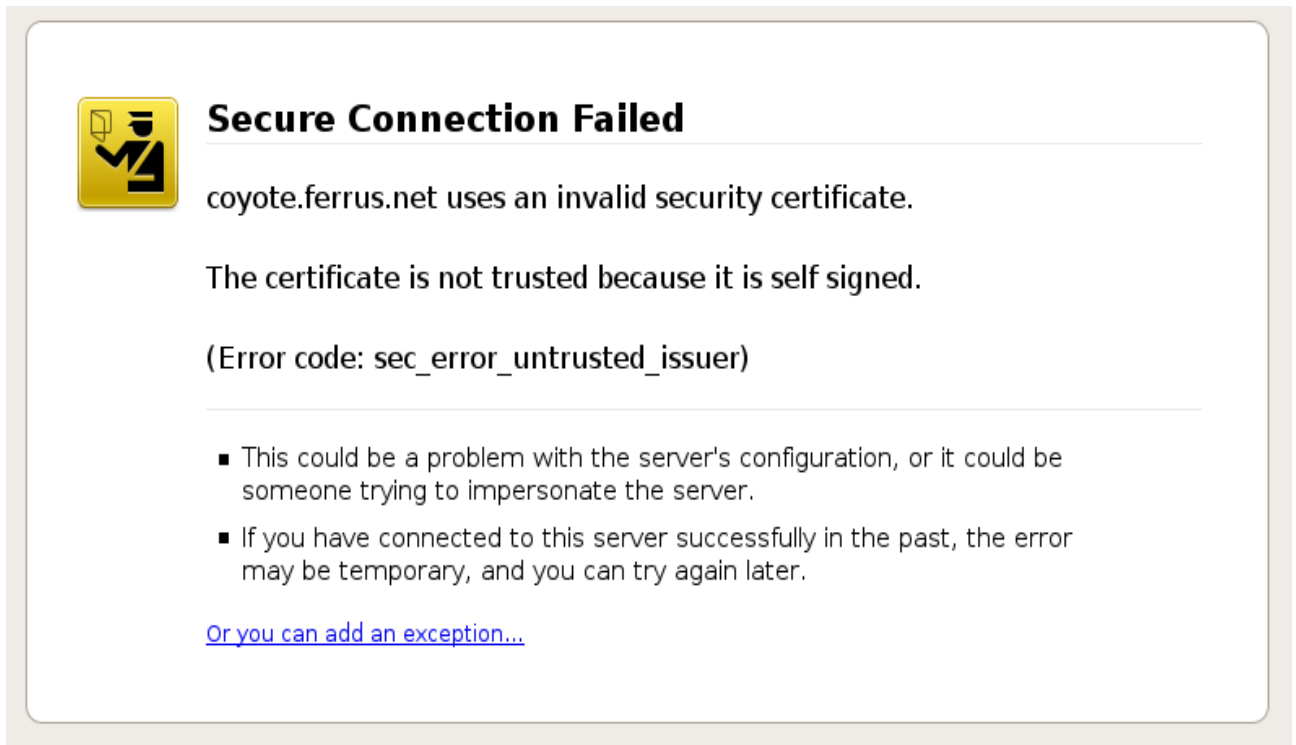
The problems of correct HTTPS deployment and UI aspects are more complex than the format of this article permits us to show. While staying within the boundaries of intranet, we may control all implementation and deployment details that are important for us. Meanwhile, «the Big Dirty Internet» is far from being ideal and it is almost completely beyond of our control. Actually, *most* of the external sites are misbehaving some way way (according to different statistics this is at least 50-70% of all sites). This usually means that browsers we are using will complain about errors encountered there [1].

Those errors have different nature. What is important for us from purely practical point of view is that, in the first place, errors as the user sees it are almost completely useless — people just ignore it; and, in the second place, there is at least one case when *we* can take care of users problems instead. Let's revisit our task again: site authenticity validation. Actually, this is not just *one* task but *two completely different ones*:

- True legal entity authenticity validation by a trusted third party - for high security sites (Internet banking, payment processing)
- Verifying the fact that site identification did not change from our previous visit (so it is still the same site) - for all the rest.

¹ The core web server certificate components are: public key, subject data designating certificate owner, issuer data designating certificate issuer, unique serial number, issue and expiration date, arbitrary extensions as defined by the standard and digital signature. «Self-signed» certificates have «issuer» data being similar to «subject» and to be used if third-party authentication is not required.

Major internet browser vendors, however, disregard this point of view. We have just similar error page (we know, users tend to ignore it and most of the time they are definitely right) for anything that may go wrong:



Secure Connection Failed

coyote.ferrus.net uses an invalid security certificate.

The certificate is not trusted because it is self signed.

(Error code: sec_error_untrusted_issuer)

- This could be a problem with the server's configuration, or it could be someone trying to impersonate the server.
- If you have connected to this server successfully in the past, the error may be temporary, and you can try again later.

[Or you can add an exception...](#)

The example above is Firefox version 3 and it is definitely lesser evil. Chrome does not allow us to define a permanent exception at all. Conventional wisdom says that more warnings just means warnings are more likely to be ignored.

The proper «High Security» solution requires border certificate validation policy enforcement. If certificate validation fails, the connection should not be allowed.

This is the only working way to prevent user from mindlessly ignoring warning messages. If (quite a rare occasion) there is an actual need to bypass the warning message, it is security administrator's responsibility to take proper qualified action (fixing the certificate; notifying the site owner on the certificate error; incident investigation if really an attack was in place; at last adding the exception rule if it is really the proper way to fix the case)

The proper «Medium/Low Security» solution requires keeping server certificate info and issuing a warning if site identification changes.

```
[ark@grave ~]$ ssh shell.sourceforge.net
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for shell.sourceforge.net has changed,
and the key for the according IP address 216.34.181.119
is unknown. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
b4:c5:6c:36:f7:ad:2c:23:52:e1:84:3e:77:61:5d:59.
Please contact your system administrator.
Add correct host key in /home/ark/.ssh/known_hosts to get rid of this message.
Offending key in /home/ark/.ssh/known_hosts:5
RSA host key for shell.sourceforge.net has changed and you have requested strict
checking.
Host key verification failed.
[ark@grave ~]$
```

This «secure shell» screenshot looks familiar for us; nothing prevents the similar functionality from being implemented in SSL proxy. The only non-trivial issue is certificate management UI as legitimate certificates are likely to change from time to time as well. As the SSL proxy is not a standalone program but a part of more complex solution we find it unnecessary to make this feature built in.

Implementation and configuration security issues

Creation of “single control point” to protect misconfigured and unpatched clients is function of traditional application firewall and SSL proxy shines here as expected.

For example, simple and effective mitigation of famous “null byte” vulnerability (shown by Kaminsky and Marlinspike [3] ¹) is just a few lines of proxy code, and the check implemented that way is uniform enough to be effective against other possible ASN.1 data length related parser errors², if there are more of them.

It is even more easy to control the usage of:

1 “Null byte vulnerability”, shown at Black Hat conference in July 2009, is based on inconsistency caused by two different ways to handle text strings: most SSL client software is written in C, thus implying strings are to be terminated with NULL byte, while CA software ignores NULL byte and starts parsing from actual string end; because of that, browser believes common name www.paypal.com[NULL].thoughtcrime.org to belong to www.paypal.com, while certificate authority treats it as belonging to thoughtcrime.org (and is probably willing to sign it if certificate signing request comes from verified thoughtcrime.org source).

2 ASN.1, Abstract Syntax Notation One — ASN.1 is a joint ISO/IEC and ITU-T standard, originally defined in 1984 as part of CCITT X.409:1984 and last revised in 2002. It provides a set of formal rules for describing the structure of objects that are independent of machine-specific encoding techniques and is a precise, formal notation that removes ambiguities. ASN.1 is used for all certificate contents, and string encoding within ASN.1 data format does not impose any limitation on string contents, as all data field size information is encapsulated explicitly.

- Obsolete SSL protocol versions [4]
- Cryptographically insecure certificate digital signatures [5]
- Cryptographically insecure encryption algorithms.

We have to note, though, that there is still no silver bullet. For example, recent (discovered at the end of 2009) TLS renegotiation attack cannot be detected this way [6] ¹ from the client side because of specifics protocol design details. Similar to other firewall systems, the rationale behind SSL “control point” is defense in depth, combined with endpoint patch and configuration management, not creating a single point of failure by relying on a single solution.

We will revisit weak crypto problem later in this article as it affects not just one certificate, but the chain of trust principle itself.

X509v3 extensions and trusted path validation

X509 version 3² extension are well known to everyone who bothered to build advanced PKI as a part of information system service architecture, large VPN deployments and almost any other non-trivial PKI deployment scenario, except, maybe, web services. This strange inconvenience is caused by amazing fact: web browsers don't provide complete support for quite old (IETF RFC2459, 1999) and supposedly ubiquitous standard!

Actually x509v3 browser support is limited to (flawed from usability point of view) OCSP support, Basic Constraints check with ensures server certificate not to be used as intermediate CA³ (and old MSIE versions did miss even that), alternate names extension and basically non-functional (set up manually and thus applicable for intranet use only) CRLs⁴. The validation chain is provided by server, it is rigid, and often insufficient. Let's look more closely on that, examining an arbitrary certificate from “the wild”, for example, certificate of Yandex.Money payment system, which uses extensions that are interesting for us:

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

1a:40:d3:01:00:00:00:00:22:d9

Signature Algorithm: sha1WithRSAEncryption

-
- 1 SSL renegotiation attack is based on abusing protocol design feature that allows SSL session to be “re-negotiated” reusing existing TCP connection. It allows attacker to inject arbitrary data into the session beginning, pretending it to be transmitted from the legitimate client, but does not provide a way to decipher the server response.
 - 2 ITU-T X.509 defines public key infrastructure components: “generic” certificates, certificate revocation lists, authorization (attribute) certificates and trusted path validation. The version 3 defines several new features discussed below.
 - 3 Basic Constraints extension defines if given certificate may be used to sign other certificates (is an authority certificate) and “path length”, describing how many certificate levels “below” are allowed – say, path length “0” means subordinated certificates may not belong, in turn, to more intermediate certificate authorities.
 - 4 CRL — Certificate Revocation List – a standard mechanism defined by X.509 to revoke compromised and obsolete certificates. OCSP – Online Certificate Status Protocol – a networking protocol, allowing requesting side to instantly check given certificate status by simple query, while traditional CRL distribution implies storing full revocation list client-side.

Issuer: DC=ru, DC=yandex, DC=ld, CN=YandexExternalCA

Validity

Not Before: Aug 25 11:55:37 2009 GMT

Not After : Aug 25 11:55:37 2011 GMT

Subject: C=RU, L=Moscow, O=000 Yandex, OU=money.yandex.ru, CN=Yandex.Money

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:bd:bf:56:ca:2d:8a:00:20:6e:a8:24:ca:32:3c:
6c:b9:9b:f2:9c:c2:10:40:01:0d:94:63:8a:10:51:
79:0b:70:29:d5:14:f8:65:7f:ca:36:a9:76:9b:30:
e2:e5:e6:e5:3c:bf:4b:f5:23:7c:c6:c6:7a:c3:de:
fa:76:da:63:56:27:69:e1:53:6c:97:89:9b:19:54:
03:df:c6:45:f6:9c:07:81:c9:45:05:42:de:ff:3f:
25:99:c0:27:6a:2d:4b:02:a0:79:0a:5e:d7:7e:e9:
33:46:ee:bb:28:a0:74:59:c7:56:c8:80:b8:c9:ed:
46:36:4f:de:69:df:d8:69:4f

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraints: critical

CA:FALSE

X509v3 Key Usage:

Digital Signature, Key Encipherment

X509v3 Subject Alternative Name:

DNS:money.yandex.ru, DNS:promo.yandex.ru

X509v3 Subject Key Identifier:

64:95:3A:17:8F:60:C4:42:BE:19:AF:06:9E:3A:50:2B:DA:98:4B:B5

X509v3 Authority Key Identifier:

keyid:DB:41:27:30:4F:1A:F5:5B:3E:84:56:C8:EC:85:98:B3:51:2C:2D:27

X509v3 CRL Distribution Points:

URI:HTTP://crls.yandex.ru/YandexExternalCA/YandexExternalCA.crl

Authority Information Access:

CA Issuers - URI:HTTP://crls.yandex.ru/YandexExternalCA/YandexExternalCA.der

1.3.6.1.4.1.311.21.7:

00.(+.....7.....&...U...-.....7....(.d...

X509v3 Extended Key Usage:

TLS Web Server Authentication

1.3.6.1.4.1.311.21.10:

0.0

..+.....

Signature Algorithm: sha1WithRSAEncryption

99:2d:66:d7:7f:26:6b:69:66:5d:fa:3c:8d:53:5e:e1:0a:e1:

```
68:78:25:d1:e6:b6:94:b6:43:11:90:cb:2b:93:69:fb:f6:82:
26:7b:60:8b:5d:b7:ec:0a:9b:2d:b2:e2:22:5d:04:d8:2b:2a:
04:0d:43:de:e8:b7:2c:e3:91:19:98:e0:4a:53:b6:15:e9:4a:
e2:84:19:11:9e:e2:86:02:74:7a:c4:36:8e:07:fc:b8:f9:c3:
23:22:d5:a9:63:c4:9b:a9:e9:1a:9b:97:0a:0a:94:02:23:5b:
9c:1c:58:14:2d:b7:7f:00:11:07:c0:c3:a1:a9:22:4c:06:09:
11:54:da:6a:76:27:d0:e9:e4:5d:40:f0:54:3f:14:7b:39:f3:
ae:52:83:8d:6b:9e:fb:c1:3b:ef:bb:f6:a4:aa:64:ba:60:f3:
1c:a0:48:da:aa:e3:f5:38:d3:e3:2c:14:49:6e:f3:01:df:04:
ef:62:59:15:a8:18:29:a1:8e:fa:da:38:1d:16:34:55:9a:ce:
e5:1c:d1:7b:db:3f:27:ca:6d:ef:49:b6:0c:3d:64:f0:cb:06:
aa:a9:aa:de:53:f2:2b:02:1f:c4:e7:d6:98:39:5e:cf:bc:9f:
79:55:6e:2f:4e:0e:f9:2f:20:0f:b6:c5:5b:26:c0:ff:c8:97:
39:c8:ab:f1
```

Extensions, that are interesting for us are marked bold. Now a few words about each of these extensions:

- Authority Key Identifier (AKID). This ID identifies issuer's public key, and, thus, provides a way to replace the well-known issuer's Subject lookup.

Why should it be done exactly this way, what's wrong with traditional Issuer – Subject pairing? There are several reasons why AKID use is preferred. The first reason is AKID identifies the issuer's public key distinctively, so if an old certificate was obsoleted by newer one, we know exactly which one do we need, despite both having similar Subject information. The second reason is advanced PKI usage scenarios, where different certificates share similar key information on purpose (cross-certification and alternate validation paths implemented to provide complex trust relationship across multiple domains) and it is important to identify certificates by public key to choose correct one properly if multiple choices are available. [8]

- CRL distribution points. The URL for certificate revocation list, implying HTTP protocol in this example.

CRL via HTTP is traditionally “slow” (a few hours, typically) way to distribute certificate revocation data. As revocation list may be quite long, maybe tens or even hundreds kilobytes, web browsers use only CRLs specified explicitly in browser settings.

- Authority Information Access (AIA). This extension provides a way to retrieve certificate issuer information.

As noted above, web applications either use manually built “certificate chains” provided with SSL/TLS handshake, or just assume all necessary CA certificates are already in the local storage available to the browser.

So what advantage does SSL proxy provide? We may use third-party certificate validation framework (Carillon Pathfinder [9] in our case) to implement features missing from the web browser software:

- Fully featured rfc5280 trust path validation employing AKID information
- Cached (to reduce traffic usage) CRL access
- Automatic missing intermediate CA certificates retrieval with AIA

- Configurable chain of trust policies to deal with “weak” (deploying cryptographically insecure hash functions) certificates properly.

It means we are able to eliminate false certificate error warnings, and, vice versa, additional protection is provided for threats which cannot be detected by generic software.

A traditional SSL man-in-the-middle use: HTTPS content inspection

This basic functionality is the primary goal of all current SSL man-in-the-middle implementations. There are various approaches, though it mostly applies to the interface the proxy provides for external content inspection applications: DLP, antivirus, archival tools, etc. I-CAP (Internet Content Adaptation Protocol [[10](#)]) is the most widespread and feature rich so we chose this one.

Design, implementation and deployment

It is quite obvious that a good security product should be “transparent”, affecting user environment as little as possible (at the first glance, at least). This is the way we designed SSL Proxy. For a typical deployment scenario, besides exceptions specified on purpose, it appears for web browsers (and user), that:

- Self-signed certificates are kept being those, though new certificate key is generated and malicious spoofing attempts may be detected;
- To prevent certificate serial number collision for mobile clients which may use different network entry points, a deterministic algorithm is used to derive “new” serial numbers from “original” ones;
- Minor certificate errors permitted by security policy (expired date, etc) are translated with no change and all diagnostics kept intact;
- Certificates signed by unknown authorities have “UNVERIFIED” string in the issuer name, thus allowing the web browser to diagnose the error quite similar to one occurred while connecting directly;
- A flexible exception policy mechanism is provided (to allow advanced setup not fully supported at the moment, like client certificate deployment, non-standard encryption algorithms or enhanced privacy settings for, say, internet banking)

PKI problems for “Big Bad Internet”

The whole PKI situation is far from being ideal. There is a fundamental issue: if any vital component of the public trust network based on well-respected certificate authorities get compromised, the whole system is flawed and unreliable. And the vital component list is quite long, as it includes not just root CA's, but *all* intermediate ones authorized to issue certificates. Even more, the compromise does not necessary mean a malicious party interfering with core CA functions; a cryptographically insecure hash algorithm may act as a weak link as well. That is purely practical and visible menace. Even at the end of 2009 there are root CA's still using MD5, and even if they do not, there are MD5 signed certificates that are still valid. That's why eliminating such a weak links required special attention. Besides that, a special (noted above) Basic Constraints extension parameter is deployed, limiting possible certificate chain length for arbitrary certificate authority. There is, though, quite widespread false alert, caused by weak algorithm used for root

certificate *self* signing, which does not actually pose any security threat. There are quite a significant number of malformed x509v3 certificates “in the wild”, causing no problem to web browsers as they do not have functionality that may have been affected anyways. All of the above differs much from controlled intranet environment and caused severe problems while adopting Pathfinder framework; thanks to Carillon Information Security specialists who were very helpful all problems turned out to be resolvable.

Summary

As shown above, the use of man-in-the-middle HTTPS proxy server in the corporate environment is perfectly justified, increasing browser security without significant side effects; it is likely to become a de-facto standard component for all enterprise application firewalls. Even if browsers evolve and some features provided with HTTPS proxies become builtin, there is always a place for more security functions implemented in proxy systems.

Show Me The Code

There are two versions of SSL Proxy available:

- Opensource/free under «Old BSD with advertisement clause» license as a part of OpenFWTK project, see http-gw module: <http://openfwtk.sf.net>
- Commercial embeddable module for firewalls and DLP systems, available under request via sales@setere.com

Carillon Pathfinder is GNU LGPL licensed, <http://www.carillon.ca/tools/pathfinder.php>