The Internet Is Insecure!!!
BGP, DNS, Wi-Fi... network infrastructure is insecure at all levels

WEP encryption cracked in 60 seconds (2008)
Kaminsky's DNS poisoning attack (2008)
... but SSL will save us!

Right?
Developed for Web browsers...

... back in 1995
SSL is now widely used by non-browser software for secure networking.
SSL Security Objectives

# Authentication = certificate validation!!
# Confidentiality
# Integrity

Provide end-to-end security even if the network is insecure!
Certificate Validation in SSL

Chain of trust verification
Hostname verification

... and several other important checks described in SSL RFCs

But we don't use RFCs. We use implementations!
The SSL Software Stack

...so, we use middleware libraries

axis  xFire

because modern applications use Web services over SSL.

...instead, we use transport libraries

fsockopen  UtlLib  UrlLib2  HttpLib
HttpsLib  HttpServletRequest
HttpsURIConection

But sometimes even these are too low level for an application.

...but we don't use raw SSL libraries

- Too low level and cumbersome to use
- Wrong level of abstraction
  - e.g., typical SSL implementation works at the socket level, does not understand HTTP
  - Impeccable application portability and maintainability

SSL Libraries Examples

CryptoAPI  ySSL
SSL Libraries: Examples

CryptoAPI

OpenSSL
Cryptography and SSL/TLS Toolkit

yaSSL

GnuTLS

JSSE
... but we don't use raw SSL libraries

- Too low level and cumbersome to use

- Wrong level of abstraction
  - e.g., typical SSL implementation works at the socket level, does not understand HTTP

- Impede application portability and maintainability
... instead, we use transport libraries

fsockopen  Urllib  Urllib2  HttpLib
HttpsClient  HttpsURLConnection

But sometimes even these are too low level for an application!
... so, we use middleware libraries

because modern applications use Web services over SSL
Exhibit 1: Amazon Flexible Payments Service
Amazon FPS: PHP SDK
Enable certificate validation: Yes

CURLOPT_SSL_VERIFYPEER = true
CURLOPT_SSL_VERIFYHOST = true

Verify name in certificate: ???
true

CURL_SSL_VERIFYHOST:

1 to check the existence of a common name in the SSL peer certificate. 2 to check the existence of a common name and also verify that it matches the hostname provided. In production environments the value of this option should be kept at 2 (default value).
How About This Common Name?

AllYourSSLAreBelongTo.us

OK!
Enable certificate validation: Yes

CURLOPT_SSL_VERIFYPEER = true
CURLOPT_SSL_VERIFYHOST = true

Verify name in certificate: ???
What Do We Mean By Insecure?

Can Read And Modify Anything!!!

- credentials
- files
- requests
- etc.

The Internet

Client <-> Attacker <-> Server
Vulnerable cURL-Based Software

[Logos of Amazon Payments, PayPal, osCommerce, Zen Cart, Ubercart, admin, Google, PrestaShop]
Horrible SSL API Design

Gnutools TLS

Python

libcurl

OpenSSL

Java
Overloaded Return Values

gnutls_certificate_verify_peers2()

- Returns 0 on success
- Returns 0 for self-signed certificates
  * Expects the developer to check the return code of _gnutls_x509_verify_certificate()
Horrible SSL API Design

Gnu TLS
Python
libCURL
OpenSSL
Java
Misleading Name/Value Pairs

CURLOPT_SSL_VERIFYPEER: TRUE/FALSE
CURLOPT_SSL_VERIFYHOST: 0 / 1 / 2
Horrible SSL API Design
Multiple Interfaces: Some Secure, Others Insecure

- HttpsClient
- HttpsURLConnection
- SSLSocketFactory
Horrible SSL API Design
Victims of
CryptoAPI, OpenSSL, HttpLib
Vulnerable JSSE-based Middleware

... and Its Victims
Vulnerable Android/iOS Applications

Redeem Groupon Orders Here

GitHub Gaug.es
What The Developers Need:
What SSL Libraries Give Them:
The Result
Open Questions

- Why are these APIs so bad?
- Why must programs that only ever connect to a single domain go through full-blown certificate discovery every time?
- Why do developers not understand the security implications of disabled or broken SSL certificate validation?
- Why didn't developers notice?

Wait, I know this one...
All this software has never been tested with bad certificates!
1. Generate a bunch of bad certificates

2. Test SSL implementations

3. ???

4. PROFIT!!!
x.509 standard.... ugh!
1. Scan the Internet, collect 243,000 certificates
2. Break certificates into grammatically valid pieces
3. Generate 8 million frankencerts from random combinations of certificate pieces
Frankencerts are random, yet grammatical X.509 certificates with ...

- Unusual extensions
- Rare and malformed values of these extensions
- Strange key usage constraints
- Odd issuers

... and many other unusual features
Differential testing: multiple implementations of SSL/TLS should implement the same certificate validation logic... if a certificate is accepted by some and rejected by others, what does this mean?
• Tested several dozen SSL implementations on 8 million frankencerts
• 208 discrepancies due to 15 root causes
• **Multiple bugs**
  • Accepting fake and unauthorized intermediate certificate authorities
  • Accepting certificates not authorized for use in SSL or not valid for server authentication
  • Several other issues
208 discrepancies due to:

- **Multiple bugs**
  - Accepting fake and unauthorized intermediate certificate authorities
  - Accepting certificates not authorized for use in SSL or not valid for server authentication
  - Several other issues

Attacker can impersonate any website
PROFIT!!!

The site's security certificate has expired!

You attempted to reach www.google.com, but the server presented an expired certificate. No information is available to indicate whether that certificate has been compromised since its expiration. This means Google Chrome cannot guarantee that you are communicating with www.google.com and not an attacker. Your computer's clock is currently set to Thursday, February 6, 2014 1:21:14 AM. Does that look right? If not, you should correct the error and refresh this page.

You should not proceed, especially if you have never seen this warning before for this site.

Proceed anyway  Back to safety

Help me understand

OK to click through?
We issued this fake certificate...
... and Chrome wouldn't tell you!