SSL Validation Checking vs. Go(ing) to Fail

Thomas Brandstetter

Presentation Outline

- Bio
- Background for research
- Architecture & Testing Concept
- Results

BIO

- Founder and GM of Limes Security
- Independent security consulting company, part of Softwarepark Hagenberg, Austria
- 2 major focus areas
  - Secure Software Development
  - Industrial Security
- Strong industrial background: Former head of Siemens ProductCERT & manager of Hack-Proof Products Program

- Associate Professor at University of Applied Sciences St. Poelten, Austria
- Classes:
  - Web- & Application Security
  - Penetration Testing
  - Industrial Security
  - Botnets & Honeypots
  - CERTs & Incident Response
- Research Interests: Industrial Security & Application Robustness

BACKGROUND FOR RESEARCH

Starting Point: Apple's security-related update iOS 7.0.6

About the security content of iOS 7.0.6

This document describes the security content of iOS 7.0.6.

iOS 7.0.6

- **Data Security**

  Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

  Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

  Description: Secure Transport failed to validate the authenticity of the connection. This issue was addressed by restoring missing validation steps.

  CVE-ID

  CVE-2014-1266

Source: http://support.apple.com/kb/HT6147

What was wrong in Apple's SSL code?

- According to public analysis, the problem resided in a file called sslKeyExchange.c (version 55741) of the source code for SecureTransport, Apple's offical SSL/TLS library
- Buggy code comes as a sequence of C function calls, starting off in SecureTransport's sslHandshake.c:
  - SSLProcessHandshakeRecord()
- -> SSLProcessHandshakeMessage() dealing with different aspects of SSL handshake:

-> SSLProcessClientHello()

- -> SSLProcessServerHello()
- -> SSLProcessCertificate()
- -> SSLProcessServerKeyExchange()
- Last function is called for various TLS connections, notably where forward secrecy is involved

Source: http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

What was wrong in Apple's SSL code?

- Here, the server uses its regular public/private keypair to authenticate the transaction, but generates an ephemeral keypair for the encryption (forward secrecy)
- Benefit of forward secrecy is that if the server throws away the ephemeral keys after each session, then you can't decrypt traffic from those sessions in the future, even if you acquire the server's regular private key by different methods (e.g. demand from law enforcement, bribery or break-in theft)
- To continue: SSLProcessServerKeyExchange() lead to function call
  -> SSLDecodeSignedServerKeyExchange()
  -> SSLDecodeXXKeyParams()
  IF TLS 1.2 -> SSLVerifySignedServerKeyExchangeTls12()
  OTHERWISE -> SSLVerifySignedServerKeyExchange()

Source: http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

Tracing the bug further to its root cause in sslKeyExchange.c

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...
fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

First fail is correctly bound to if statement, but the second isn't conditional:
Code always jumps to the end from that second goto, err will contain a successful value because SHA1 update operation was successful and so the signature verification will never fail!

Source: http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

Analyzing what the code probably should have done

Code should calculate cryptographic checksum of three elements - the three calls to SSLHashSHA1.update(), then call the critical function sslRawVerify().

If sslRawVerify() succeeds, then err ends up with the value zero "no error“

That's what the SSLVerifySigned-ServerKeyExchange function returns to say, "All good."

The first goto fail happens when the if statement succeeds, e.g. if there has been a problem and therefore err is non-zero, causing an immediate "bail with error," and the entire TLS connection fails.

In C, the second goto fail, which shouldn't be there, always happens if the first one doesn't

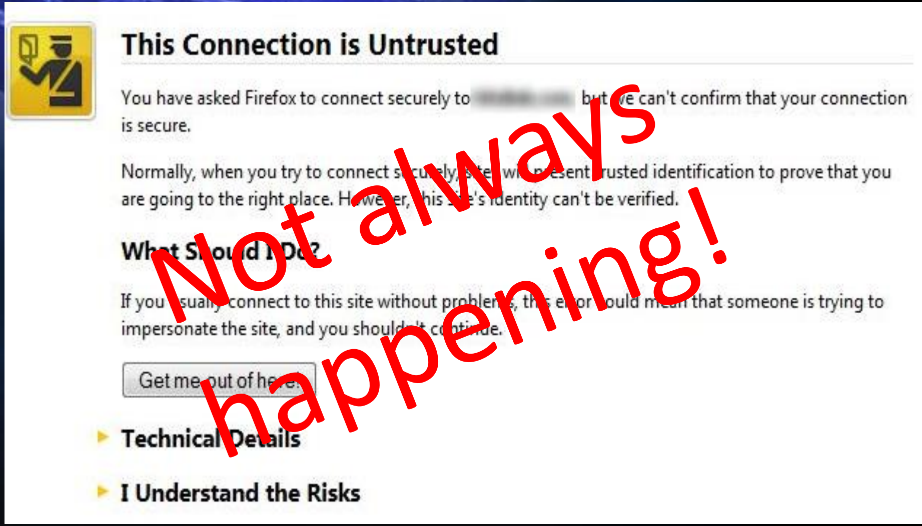
The result is that the code jumps over the call to sslRawVerify(), and exits the function.

This causes an immediate "exit and report success", and the TLS connection succeeds, even though the verification process hasn't actually/fully taken place.

Source: https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

What did it mean?

- SSL Validation not working properly:
- Link between ephemeral key and certificate chain is broken
- Possible to send a correct certificate chain to client, but sign handshake with wrong private key, or not sign it at all
- No proof that the server possesses the private key matching the public key in its certificate
- Forged certificates should lead to error message/warning are omitted
- Thereby making man-in-the-middle (MITM) attacks easier



Source: https://www.imperialviolet.org/2014/02/22/applebug.html

Our thoughts at this point

How is it possible that this critical bug in a security function went unnoticed for a long time?

Could it have been detected?

If source code was available: Yes! By Apple conducting source code scans/reviews, indicating that code fragment is never reached	If source code was not available (Most of the time): Maybe, only if SSL validation checks can be somehow assessed from the outside systematically

This lead to our main research questions

When having the source code, detecting a bug like goto fail seems possible, but:

- To which degree can SSL validation checks of 3rd party apps be systematically assessed if source code is not available?
- What is the overall state of SSL validation checks conducted by app(lication) developers currently, are developers doing the right things?

SSL VALIDATION FUZZER
CONCEPT & ARCHITECTURE

Our designated approach

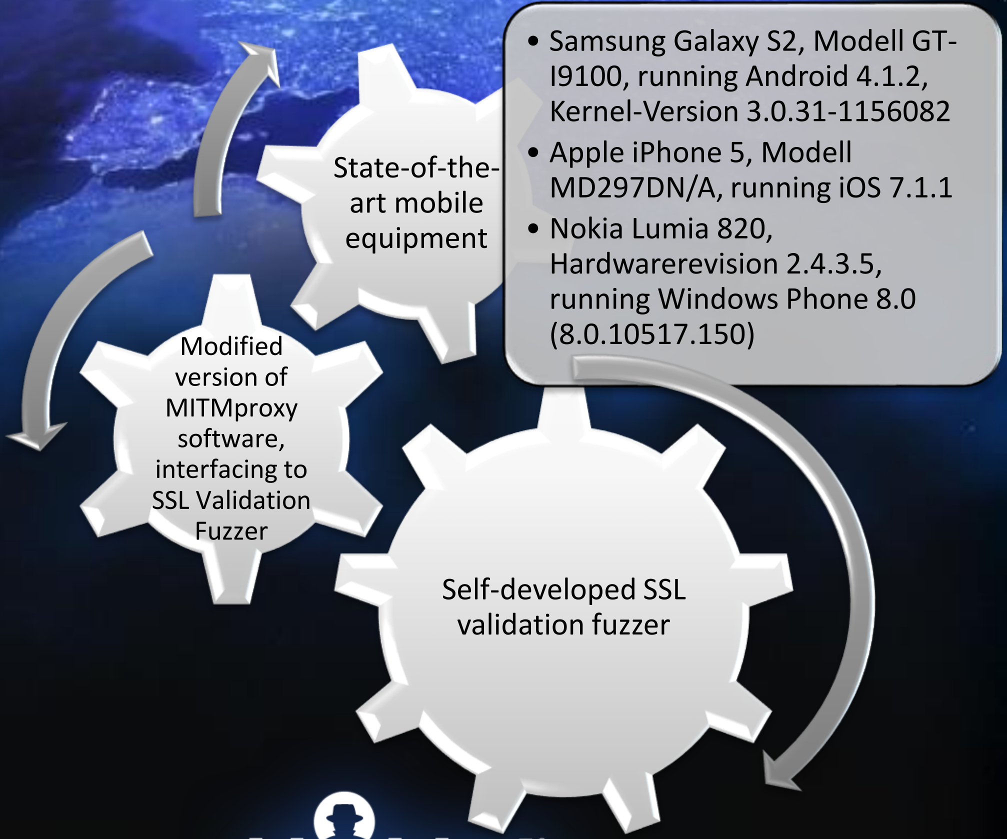
Derive a testing methodology which allows us to assess whether SSL validation checks in different (mobile) applications have been implemented properly by the app's developers – without having access to the source code

Create a tool which helps us in this assessment

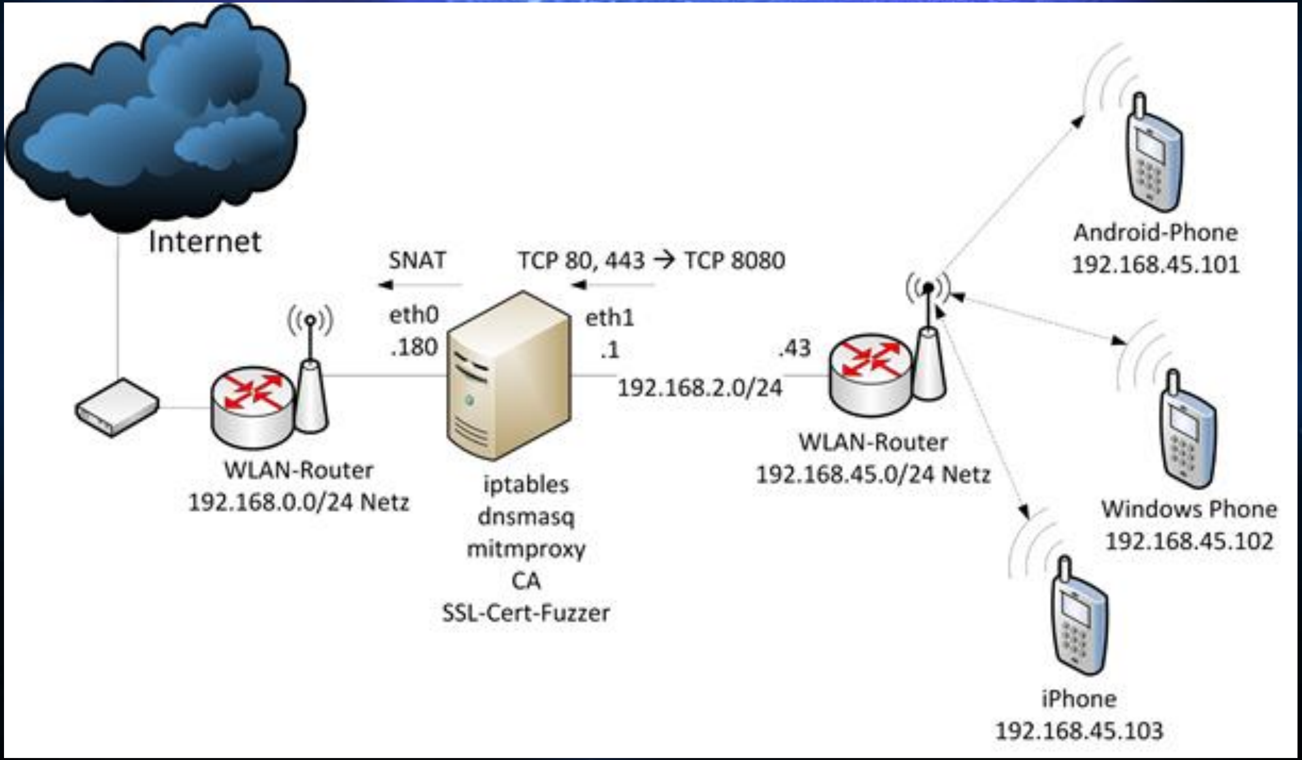
Run this tool against a number of apps which are likely to have SSL validation implemented: Candidate group one: Critical EBANKING/payment apps!

Check the same app on the 3 main mobile platforms iOS, Android and Windows Phone to look for interesting patterns

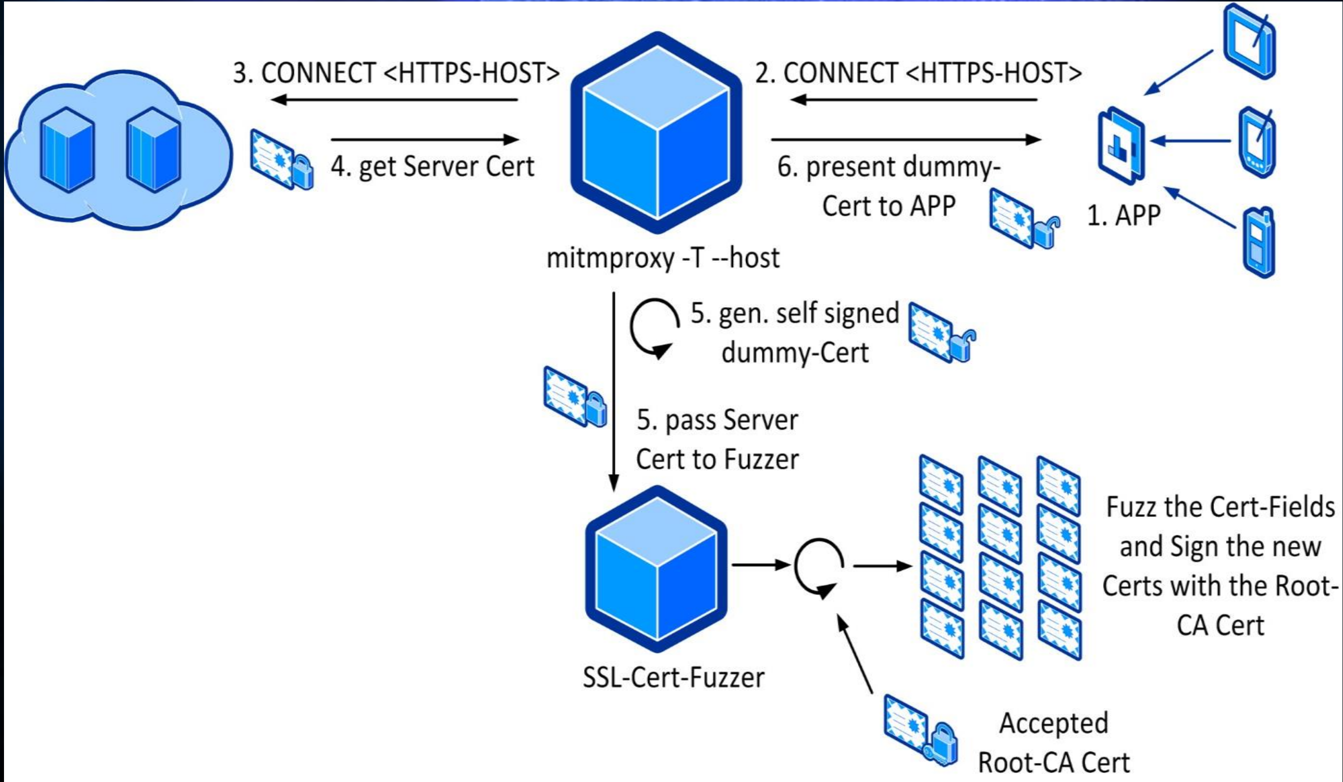
Main Assessment Components



Architecture & Setup of Assessment Environment



Testing Approach: Target-oriented SSL validation fuzzing/checking



Test cases! But which ones do make sense? How twisted can a developer's mind be?

LIMES SECURITY

fh/// st.pölten

- Case 1: arbitrary certificate
- Case 2: valid certificate
- Case 3: invalid notAfter
- Case 4: invalid notBefore
- Case 5: invalid Hostname, original serial no
- Case 6: invalid signature, modified serial no
- Case 7: invalid signature, original serial no
- Case 8: not signed with key of CA
- Case 9: issuer field of certificate does not match subject of CA
- Case 10: hostname in subject field modified
- Case 11: no hostname in subject field, subjectAltNameExtension changed
- Case 12: version 2 certificate with wrong hostnme in subject field, correct one in subjectAltName-Extension
- Case 13: certificate chain is extended with intermediate certificate
- Case 14: incorrect intermediate-certificate (basicConstraints = CA:FALSE)
- Case 15: tbd

List Initial test cases based on x509 standard certificate fields,
In addition:
- SSL stripping
- Certificate pinning

RESULTS

90 mobile applications tested as of August 8th, 2014)

3Kundenzone	Airbnb	Alinma Bank - مصرف	Amazon	American Bank – Mobile Banking	Anson Bank & Trust e-zMobile Banking	Apothekerbank	Ärztebank	Bank Austria MobileBanking	BAWAG PSK	BDSwiss - Die Trading App
BKS Bank Österreich	BNY Mellon Business Banking	BNY Mellon Private Banking	Börse Frankfurt	Börse, Aktien, Aktienkurse - finanzen.net	Brokerjet	BTV Banking	bwin Sports	bwin.com Poker	cfd Banking Services	Commerzbank
DenizBank AG – Österreich	easybank	E-Central mobile Banking	E-POST KontoPilot - Banking App	Erste Bank / Sparkasse Österreich - netbanking	Fidor Bank	First Bancorp Mobile Banking	FX on J.P. Morgan Markets	Gärtnerbank	German American Mobile Banking	GLS
Hampden Bank Mobile Banking	HDFC Bank MobileBanking	HYPO Landesbank	Hypo Mobile	HYPO NOE Mobile Banking	Hypo Vorarlberg	Immobiliensuche - Wohnnet.at	ING-DiBa Austria Banking App	Interwetten – Sportwetten	J.P. Morgan adr.com	Kotak Bank
LLB Mobile Banking	LOTTERIEN SHAKER	Lufthansa	Mein A1	mein bob	Meine Bank	My T-Mobile	ÖAMTC	ÖBB Scotty	Oberbank	Openbank
Paypal	paysafecard	Personal Banking	Pizza Mann Austria	Plus500	Post	Postbank Finanzassistent	Prime on J.P. Morgan Markets	Quick Mac	Raiffeisen Meine Bank.	Santander Accionistas
Santander Bank	Santander Río	Skrill	Southern Michigan Bank & Trust	Sparda-Bank	SPARDA-BANK Linz	Suncorp Bank	TeleTrader	timr	Tipico Sports	Trader's Box
	Tyndall e-Banking	UBS Mobile Banking	VeroPay	Volksbank Mobile Banking	VP Bank e-banking mobile App	Wells Fargo CEO Mobile	Wells Fargo Mobile	WKO Mobile Services	yesss!	

Results 1 / 2: The bad news

- Even in the world of mobile banking apps: In 2014 there are still several apps of European / international banks (regardless of company size) that do not apply ANY validation checking and are susceptible to MITM attacks => Total fail ☹
- Several lower degrees of failed validations found
- Some apps are susceptible to SSL stripping, allowing for undetected malicious redirects e.g. “good” way of supporting phishing purposes
- Some payment apps transmit quite a bunch of (device) data possibly for fraud detection, maybe raising privacy concerns
- Some use out-of-band tcp connections for whatever reasons

Interesting to see what data is being sent by an app, e.g. Paypal, probably for risk/fraud estimation:

device_info:
{"device_identifier":"c5eeca5e-56ef-4878-af58-09b1e6a0e056","device_os":"Android","device_name":"GT-I9100","device_model":"GT-I9100","pp_app_id":"APP-3P637985EF709422H","device_os_version":"4.1.2","device_type":"Android","device_key_type":"ANDROIDGSM_PHONE","is_device_simulator":"false"}

app_info:
{"device_app_id":"APP-3P637985EF709422H","client_platform":"AndroidGSM","app_version":"5.4.3","app_category":"3"}

risk_data:
{"sms_enabled":true,"conf_url":"https:\/\/www.paypalobjects.com\/webstatic\/risk\/dyson_config_v2.json","is_rooted":false,"network_operator":"23210","payload_type":"full","ip_addrs":"192.168.45.100","app_version":"5.4.3","is_emulator":false,"conn_type":"WIFI","comp_version":"2.1.3","os_type":"Android","timestamp":1401226027532,"risk_comp_session_id":"396c4bd0-5a1e-4395-b3ad-eb67cecdb88b","device_model":"GT-I9100","device_name":"GT-I9100","sim_serial_number":“XXXXX3102000793002460","ssid":"GBT-Party","roaming":false,"device_uptime":284285979,"cell_id":7441899,"phone_type":"gsm","mac_addrs":"04:46:65:4A:CA:59","subscriber_id":“XXXXX922600356","ip_addresses":["fe80::646:65ff:fe4a:ca59%wlan0","192.168.45.100"],"device_id":“XXXXX0044348101","app_guid":"c5eeca5e-56ef-4878-af58-09b1e6a0e056","locale_lang":"de","os_version":"4.1.2","locale_country":"AT","bssid":"64:66:b3:c7:0b:bd","linker_id":"b1d3074f-9ec1-45dc-9550-9723cb5388f8","location_area_code":2031,"app_id":"com.paypal.android.p2pmobile","total_storage_space":12353372160,"tz_name":"Mitteleuropäische Zeit"}

Results 2 / 2: The good news

- Several banking/payment apps do exist which apply all SSL validation checks – homework properly done ☺
- Certificate pinning is being done some cases(platform-dependent) but not totally widespread
- If platform-provided validation functions are used instead of home-grown code, results look more decent (as long as there's no other go-to fail of course…)

Summary & Take-Aways

- Assessing SSL validation checks of a 3rd party app(lication) is possible to a good degree even without source code
- Even in 2014 in the banking sector, SSL validation checking is not done properly in all cases – bad guys have probably figured out where(locally) it's worthwhile
- More education of developers creating apps with secure channels seem to be necessary to prevent the next go-to fail for widely-used apps

Thanks to:

Christian Stoiber/FH St. Pölten

for first working concept

Peter Panholzer/Stefan Keil of Limes Security

for ideas and refinement

More information available at

www.limessecurity.com/sslvalidation

Contact info:

thomas@limessecurity.com