How to expose your customer's data within your mobile app

Stephan Breitrainer / confidr.me
Version 0.1

stephan@confidr.me
September, 17th 2014

Executive Summary

Getting data in a secure and convenient way from customers to your servers has always been a challenge. SSL, in this special case it's implementation in the HTTP protocol, is the first thing you may think of when it comes to “secure data transfer”. But only HTTPS is not what you want if you're running apps for payment, social media or cloud storage. It's just one limb in a chain of security mechanisms you should use.
This whitepaper will demonstrate, how your HTTPS-only app can be bypassed and how to mitigate this issue.

Based on random samples, I discovered many vulnerable apps. As a result of my findings, I contacted two affected companies, one big player when it comes to

social media and one service provider for some banks in Austria. Guess who's concerned about their customers data.

Here is a list of tested and vulnerable apps[1]:

- Amazon
- PayPal
- eBay
- Facebook
- Facebook Messenger
- Microsoft OneDrive

You may want to add some more, but as I discovered PayPal will leak my credentials, I was not amused. How can a PCI DSS certified payment provider expose my login data? And how did this get through QA?

But how did I get the app to tell me, what's inside their SSL packets? It's as simple as embarrassing: good old man-in-the-middle.

NOTE: the list above is not a full list of vulnerable apps. It's just an excerpt of how vulnerable even big-players are.
This whitepaper can only take care of Andorid based apps, iOS apps may or may not be affected too.

1 as per Sept. 24, 2014, tested on latest Android 4.4.4 on a HTC One m7

Table of contents

Interceptable data

In fact, using a simple mitm-attack can expose all data transfered from A to B. It doesn't matter if its your passport image, your driving license or your username and password. Every single request can be seen as plaintext.
So even if an app suggest it's using secured connections to transfer your data, there may be an attacker between recording your session.

Possible attack vendor

When it comes to mitm attacks, there's at least a little help required: the users device has to accept any SSL connection to any service without throwing an exception or exposing the “untrusted certificate” dialog.
But in times where certificate based authentication and VPN on mobile devices are getting spread and common, it's easy to abuse this certificates to tie another certificate to it. Many users will simply accept and install the certificate on their devices (“I *need* free WiFi, so I better install this..”, “when I am abroad, I do not have any mobile data package, so I have to use free WiFi”).

So once the certificate is in place, the owner of the WiFi AP may abuse the users' faith and route their traffic through a proxy. Proxy servers are very common and a legitim way, when any service provider want's to ensure, their users don't do illegal stuff while using their network.

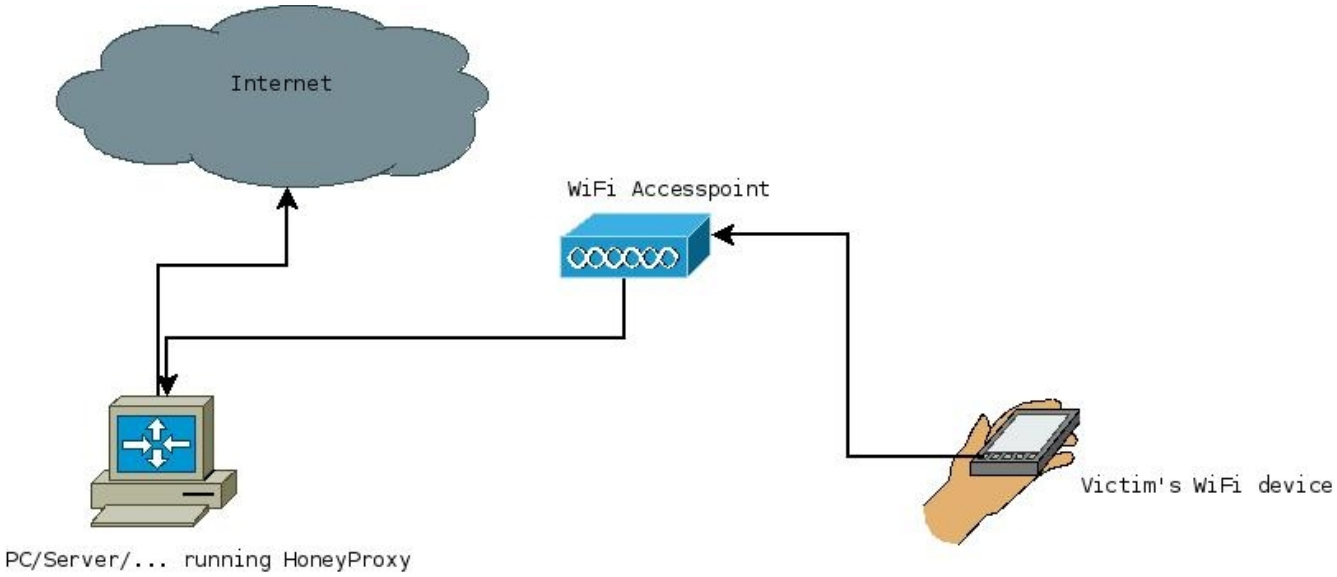
PoC

In this PoC we'll attack a famous payment provider where you can pay your eBay auctions or send money from A to B.

What we need

- WiFi AP (any device should be fine as long as you're able to install iptable rules)
- HTTP/S Proxy[2] (if you want to geht both, HTTP and HTTPS)
- some certificates and tools[3]
- a victim (preferred using an android device)

The big picture



As you may figured out, my drawing skills are not that good, but I hope you'll get the point.

2 http://honeyproxy.org/
3 http://www.tcpdump.org/ and/or https://www.wireshark.org/

So, what we want to achieve is:

- get the victim to install our certificate
- route the traffic through our proxy

To get the first point done, you may use any social engineering tactics or even sideload your certificate to the “victims” device.

The second part are our iptable rules:

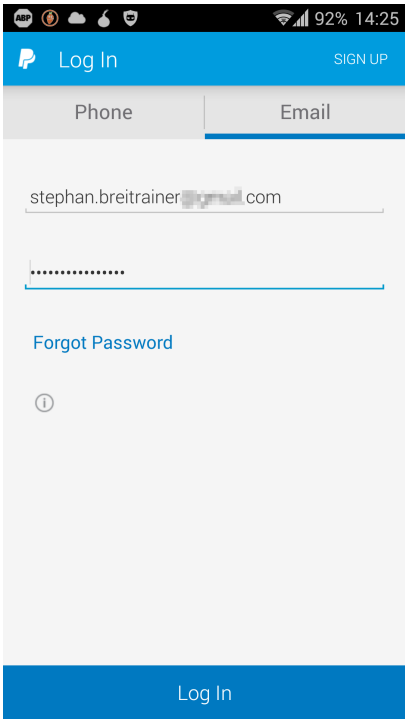
```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -j DNAT
--to-destination <PROXY IP>:8080
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to-
destination <PROXY IP>:8080
```

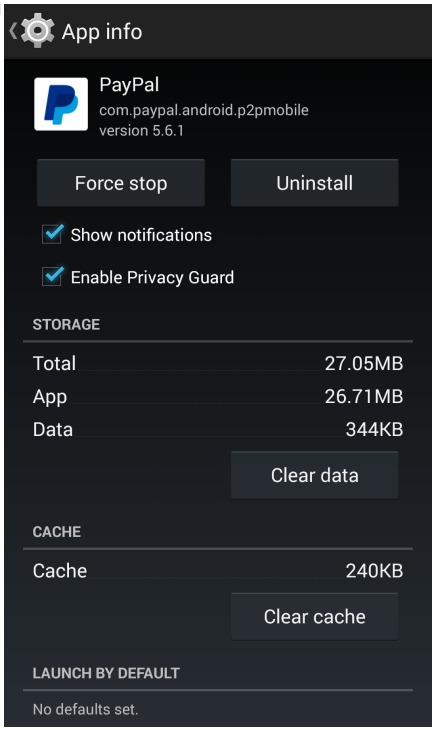
You may want adopt those rules and/or add others.

If you don't want your whole device's traffic to be routed over the proxy, you can use some local proxy app to route only specified apps (they mostly require a rooted device).

Go and get your data

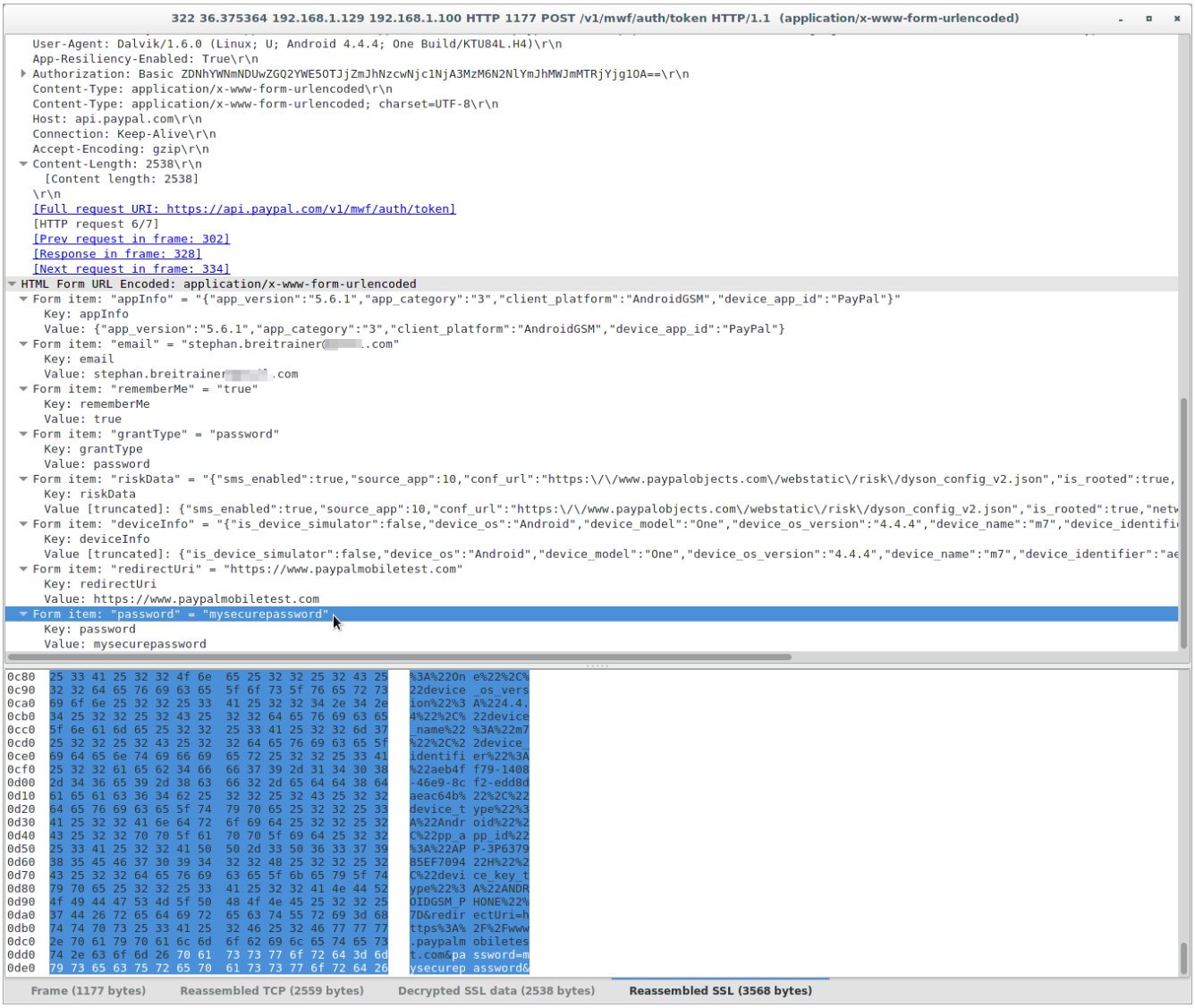
Once we have our stuff in place (WiFi AP, proxy and sniffer) and up and running, we can start our first attempt to gather the login data.
Let's start here:





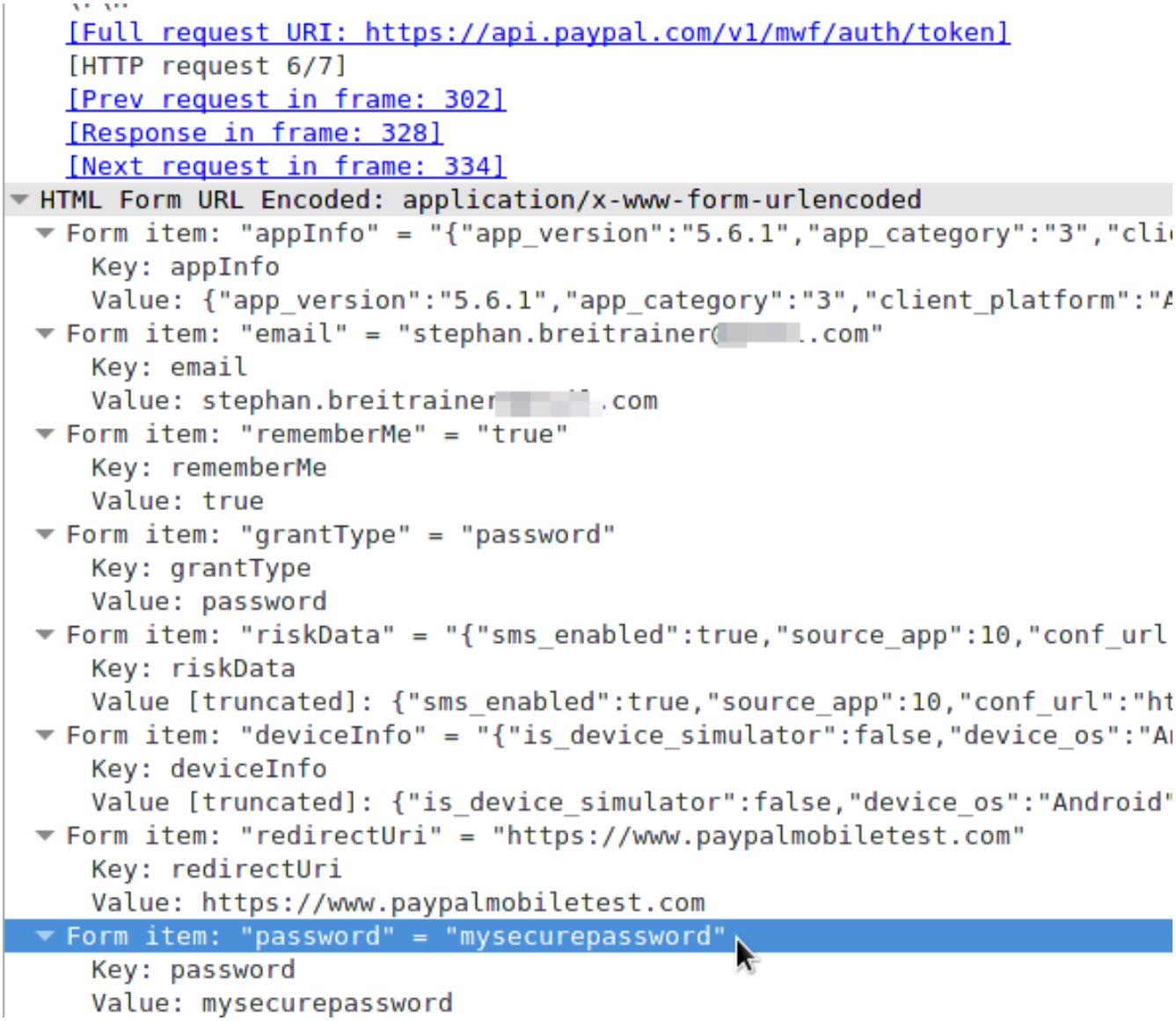
As you can see, it's the newest version of PayPal's Android app and I've used my email and password to login.
If you are using HoneyProxy as your proxy, you can use the “Report” tab to generate a POST-report via the “post_extract.js” JavaScript. If not, you have to tell wireshark how to decode your SSL traffic[4].

Now it's time to dig in the packages we've captured dring our login. If you've done it right, you can see a POST request like this:



The relevant data can be found here:

4 http://support.citrix.com/article/CTX116557



And it's reproduceable. But that's not all, you also can intercept and manipulate money transfers and grab new added credit card numbers and so on.

How to NOT expose your data

The developer's way

Just use this common technology called “Certificate-pinning”. Even if you're not

familiar with this stuff: it's nothing bad or tricky. Pinning is a simple way to ensure, your app will only talk to the servers, you allow. If the server's certificate does not match your hardcoded hash, it will deny any connections and won't expose any data to third parties.

Certificate pinning was a major part at the “Black Hat 12”[5] in the U.S. and the OWASP (Open Web Application Security Project) also provides great examples[6].

Regular Updates vs. self-signed certificates

If you have to choose between a regular update cycle to renew your certificates or using a self-signed certificate, it's up to you and/or your organisation's policy. But keep in mind, even using a big CA does not protect your certificates[7] [8].

So you may want to use a long-term self-signed certificate which you can exchange every now and then for your mobile endpoints. Keep in mind to restrict access to your mobile endpoints too.

If it's impossible to implement pinning for all participating servers, pin at least those servers, where you send your sensitive data to.

The users way

Don't do any risky tasks while using open WiFi networks. It's wise to avoid exposing sensitive data (such as banking or other personal data) if you are not sure, nobody is listening.

If you have to use open WiFi networks, please ask or check your app's service provider website, if they implemented certificate-pinning in the app you're about to use.

Possible signs your privacy is being invaded

- some apps refuse to connect or log you in
- your mobile device may show a warning (“Network monitoring. A thrid party is capable of monitoring your network activity [...]”)

5 https://media.blackhat.com/bh-us-12/Turbo/Diquet/BH_US_12_Diqut_Osborne_Mobile_Certificate_Pinning_Slides.pdf
6 https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning
7 https://www.schneier.com/blog/archives/2012/02/verisign_hacked.html
8 http://arstechnica.com/security/2011/09/comodo-hacker-i-hacked-diginotar-too-other-cas-breached/

- transactions on your banking account you are not aware of
- read e-mails you are sure you did not read before

Always double check your banking account “the good old way” using bank statements on plain paper and change all your passwords on a (ir)regular basis.

Summary

Mitm Attacks are nothing new and there are still many vulnerable apps. And when it comes to app development, it's the developers, QAs and Chief Security Officers job to eliminate all possible threats. Of course, pinning itself is not the Holy Grail, but it's a simple and easy step to implement and make an attacker's life harder.

Threat management

If you're running apps or services, it's a good idea to implement something like a “Bug Bounty” program to avoid damage to your comany's image and motivate bughunters to report issues. A clear and open communication with reporters should also be part of the game. So act quick, respond clear and don't try to hide your mistakes.

Responsible Disclosure

17.09.2014: discovered vulnerability
19.09.2014: reported vulnerability to a service provider for eBanking apps in Austria
21.09.2014: reported vulnerability to facebook (amazon does not take bug reports for enduser apps)
22.09.2014: response from the service provider
22.09.2014: response from facebook (facebook does not classify this vulnerability as major; no fix planned)
23.09.2014: telco with service provider's Chief Information Security Office and Chief Developer (quick and very professional response; vulnerability will be fixed within the next release)
24.09.2014: reported vulnerability to PayPal/eBay
24.09.2014: reported vulnerability to Microsoft
03.10.2014: PayPal/eBay ranked the bugreport as “out of scope” (no fix planned)
26.11.2014: Microsoft ranked issue as “non security issue”
14.12.2014: Published document.