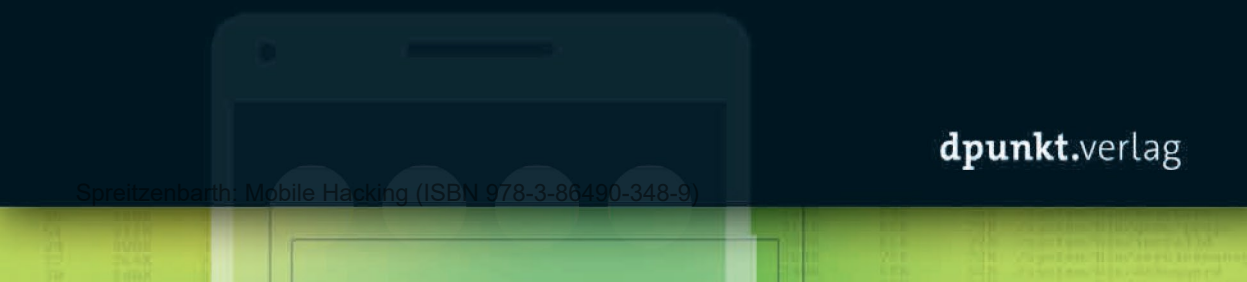
Michael Spreitzenbarth

Mobile Hacking

Ein kompakter Einstieg ins Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile



dpunkt.verlag

Spreitzenbarth: Mobile Hacking (ISBN 978-3-86490-348-9)

Michael Spreitzenbarth

Lektorat: René Schönfeldt
Lektoratsassistenz: Stefanie Weidner
Projektkoordination: Miriam Metsch
Copy-Editing: Ursula Zimpfer
Satz: Da-TeX, www.da-tex.com
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Print 978-3-86490-348-9
PDF 978-3-96088-124-7
ePub 978-3-96088-125-4
mobi 978-3-96088-126-1

1. Auflage 2017

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Inhaltsverzeichnis

Spreitzenbarth: Mobile Hacking (ISBN 978-3-86490-348-9)

Spreitzenbarth: Mobile Hacking (ISBN 978-3-86490-348-9)

9 Angriffe auf die Datenübertragung

In der heutigen Zeit, in der es den Nutzern immer mehr auf die Sicherheit der privaten Daten ankommt und die Bevölkerung sehr stark den Bereich des Datenschutzes und der Privatsphäre schätzt, sind auch mobile Applikationen von diesem wichtigen Trend betroffen.

Im Wesentlichen werden wir in diesem Kapitel auf die drei häufigsten der möglichen Angriffe auf eine mit SSL gesicherte Verbindung zwischen Applikation und Backend eingehen: klassische *Man-in-the-Middle-Angriffe* mit (nicht-) vertrauenswürdigen Zertifikaten, die Umgehung des *Certificate Pinning* sowie *SSL-Strip*.

9.1 Schutz der übermittelten Daten auf dem Transportweg

Immer mehr Webseiten und -services bieten ihren Dienst nur noch per verschlüsselter Verbindung an, um damit die Datenübermittlung auf dem Transportweg vor unberechtigten Dritten zu schützen. Im Volksmund wird diese Verschlüsselung oft *Secure Sockets Layer (SSL)* genannt – auch wenn dies für heutige Verbindungen meist nicht mehr zutrifft.

SSL war der Grundstein, auf dem in den vergangenen Jahren viele neue Verschlüsselungsstandards aufgebaut wurden. Jedoch sind vor allem die älteren SSL-Protokollversionen in den vergangenen Jahren Ziel von standardisierten Angriffen geworden oder besitzen gravierende Schwächen in der Implementierung, die eine Sicherheit der Übertragung nicht mehr gewährleisten. Heutzutage verwenden die meisten Webseiten daher die neueren Protokollversionen, die unter dem Namen *Transport Layer Security (TLS)* bekannt sind. Der aktuelle Standard ist *TLS1.2*. Im Rahmen dieses Buches halten auch wir uns an die gebräuchliche Deklaration und verwenden SSL als Synonym für alle Protokollversionen.

Auf welche Protokollversion sich die Applikation und der Backend-Server einigen, ist für die im Folgenden gezeigten Angriffe jedoch nicht von Interesse: Wir konzentrieren uns hierbei nicht auf Schwachstellen in der Implementierung des Protokolls selbst, sondern nur auf Schwachstellen oder Designfehler in der Anbindung dieser Protokolle innerhalb der mobilen Applikationen. Denn dies

Spreitzenbarth: Mobile Hacking (ISBN 978-3-86490-348-9)

sind auch die Fehler, die ein Entwickler schnell beseitigen kann, um die Sicherheit und das Vertrauen in die Applikation wieder herzustellen.

Exkurs: SSL-Verbindung

Prinzipiell handelt es sich bei einer SSL-Verbindung um ein hybrides Verschlüsselungsprotokoll zur sicheren Datenübertragung, d. h., es kommt sowohl eine symmetrische Verschlüsselung (beim Session-Key) als auch eine asymmetrische Verschlüsselung (beim Schlüsselaustausch) zum Einsatz. Dabei baut der Client (hier die mobile Applikation) eine Verbindung zum Backend-Server auf. Der Backend-Server authentisiert sich gegenüber dem Client mit einem X.509-Zertifikat, das vom Client speziell in Bezug auf die Gültigkeit und die Übereinstimmung des Servernamens geprüft wird.

Im Anschluss an eine erfolgreiche Prüfung schickt entweder der Client dem Server eine mit dem öffentlichen Schlüssel des Servers verschlüsselte geheime Zufallszahl oder die beiden Parteien berechnen mithilfe des *Diffie-Hellman-Schlüsselaustauschs* ein gemeinsames Geheimnis. Aus dem Geheimnis wird dann ein kryptografischer Schlüssel abgeleitet, der in der Folge genutzt wird, um alle Nachrichten der Verbindung mit einem symmetrischen Verschlüsselungsverfahren zu verschlüsseln und somit vor unberechtigten Dritten zu schützen.

9.2 Man-in-the-Middle-Angriffe

Ziel des *Man-in-the-Middle-Angriffs (MitM)* ist es, sich Schwächen in der Implementierung einer Applikation zunutze zu machen, um sich in diese Verbindung einzuklinken und den übermittelten Datenstrom zu entschlüsseln.

Bei Angriffen dieser Art setzt sich der Angreifer, oder in unserem Fall der Sicherheitsanalyst, zwischen das mobile Endgerät und den Backend-Server. Dies geschieht entweder dadurch, dass der Rechner des Analysten als Proxy für HTTP- und HTTPS-Verbindungen konfiguriert wird, oder durch sogenannte *ARP-Spoofing-Angriffe*, bei der der Rechner des Analysten vorgibt, der Router zu sein. In beiden Fällen erhält der Analyst alle Datenpakete und kann selbst bestimmen, wohin sie weitergeleitet werden.

Der obere Bereich von Abbildung 9–1 zeigt eine schematische Darstellung einer normalen HTTPS-Verbindung zwischen einer mobilen Applikation und einem Backend-Server. Der untere Bereich hingegen stellt die Verbindung bei einem erfolgreich durchgeführten *MitM-Angriff* dar.

Da für diese Angriffe die Zertifikate ausgetauscht werden, die zur Authentifizierung des Backend-Servers und zur Verschlüsselung der Verbindung bestimmt sind, unterscheiden wir zwei Fälle: nicht vertrauenswürdige und vertrauenswürdige Zertifikate.

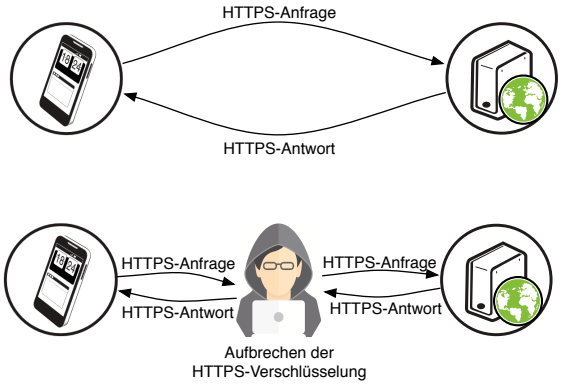


Abb. 9–1: Schematische Darstellung eines Man-in-the-Middle-Angriffs

9.2.1 Man-in-the-Middle-Angriff mit nicht vertrauenswürdigem Zertifikat

Um den Netzwerkverkehr einer Applikation zu überwachen, benötigen wir nun noch einen Proxy, der sich in die Verbindung einklinkt und so alles mitschneiden kann. Hierfür verwenden wir *Burp Suite*[1]. Tools wie *mitmproxy*[2] oder *Charles Proxy*[3] können hierfür ebenfalls verwendet werden und sind – im Fall von mitmproxy – sogar als kostenlose Open-Source-Varianten erhältlich. Die in den folgenden Abschnitten verwendete Burp Suite ist in Java geschrieben, daher benötigen wir keine Installation, ein einfacher Download und das nachfolgende Ausführen der JAR-Datei reicht völlig aus.

Nach dem erfolgreichen Start der Burp Suite müssen wir das Programm nun so konfigurieren, dass es als Proxy fungiert und für das mobile Endgerät mit der zu testenden Applikation erreichbar ist. Dies gelingt uns am einfachsten über `Proxy` → `Options` → `Proxy Listeners` → `Edit` → `Binding`. Dort setzen wir den Haken bei `All Interfaces` (siehe Abbildung 9–2). Wenn bekannt ist, auf welchem Interface das Testgerät lauscht, kann hier auch die entsprechende IP-Range eingetragen werden, um nicht eventuell Datenverkehr von anderen Programmen oder Geräten mitzuschneiden.

Als nächsten Schritt müssen wir nun das mobile Endgerät dazu bringen, dass es diesen Proxy auch verwendet. Dies erreichen wir, indem wir die IP und den Port des Proxy in die Verbindungseinstellungen des Endgerätes eintragen.

- **Android:** `Einstellungen` → `WLAN` → lange auf das `Lab-Wi-Fi` klicken, bis ein Auswahlfenster erscheint → `Netzwerk ändern` → Haken setzen bei `Erweiterte`

[1] Burp Suite: https://portswigger.net/burp/

[2] mitmproxy: https://mitmproxy.org

[3] Charles Proxy: http://www.charlesproxy.com/

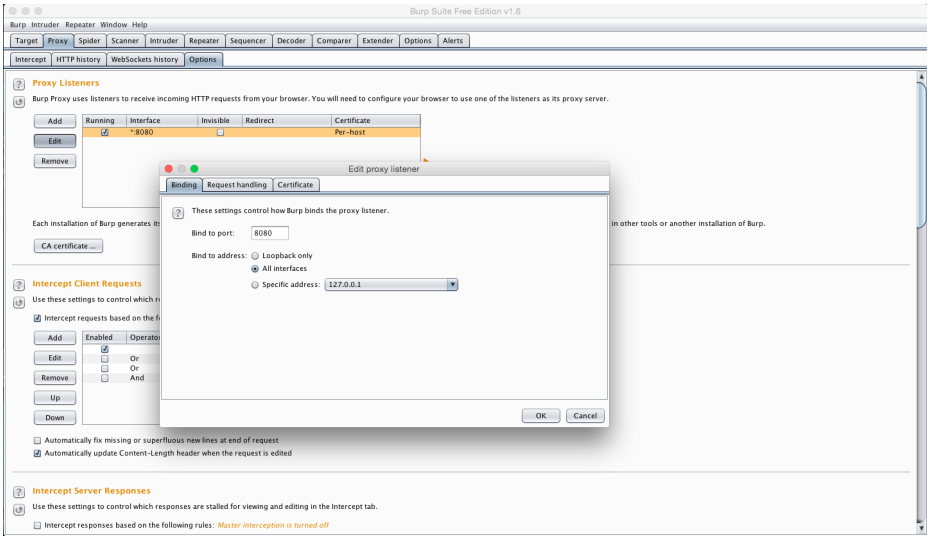


Abb. 9–2: Einrichten des Proxy Listeners in der Burp Suite

`Optionen` → den Proxy auf `Manuell` setzen und Port sowie IP des Analyserechners eintragen.

- **iOS:** `Einstellungen` → `WLAN` → auf das ⓘ in der Zeile des `Lab-Wi-Fi` klicken → den Proxy auf `Manuell` setzen und Port sowie IP des Analyserechners eintragen.
- **Windows 10 Mobile:** `Einstellungen` → `WLAN` → `Verwalten` → `Lab-Wi-Fi` → den Schieber bei `Proxy` auf `Ein` schalten → Port sowie IP des Analyserechners eintragen.

Um nun zu testen, ob wir alles richtig eingerichtet haben, reicht es, im Browser des mobilen Endgerätes auf die Google-Startseite zu wechseln. In Abbildung 9–3 ist dies exemplarisch mit einem Android-Emulator dargestellt.

Sollte beim Versuch, die Google-Startseite aufzurufen, eine Warnung eingeblendet werden, die im Wesentlichen aussagt, dass dem Zertifikat von Google nicht vertraut wird und die Verbindung eventuell unsicher ist, so liegt das daran, dass Burp standardmäßig sogenannte *Self-signed*-Zertifikate verwendet, die keine Vertrauensbasis zu den auf dem Gerät installierten Root-Zertifikaten besitzt. Diese Meldung kann oft mit einem einfachen Klick auf `Weiter` übersprungen werden, würde bei einem realen Angriff aber vermutlich für erstes Misstrauen sorgen.

Für uns als Sicherheitsanalysten sollte allerdings die Tatsache, dass eine zu testende Applikation keine Warnmeldung anzeigt, viel mehr Misstrauen hervorrufen, da dies ein Anzeichen dafür ist, dass die Applikation die Serverzertifikate nicht korrekt prüft. Dies ist im Bereich der Sicherheitsüberprüfung von mobilen Applikationen eine der gravierendsten Schwachstellen, da hierüber alle Nutzer

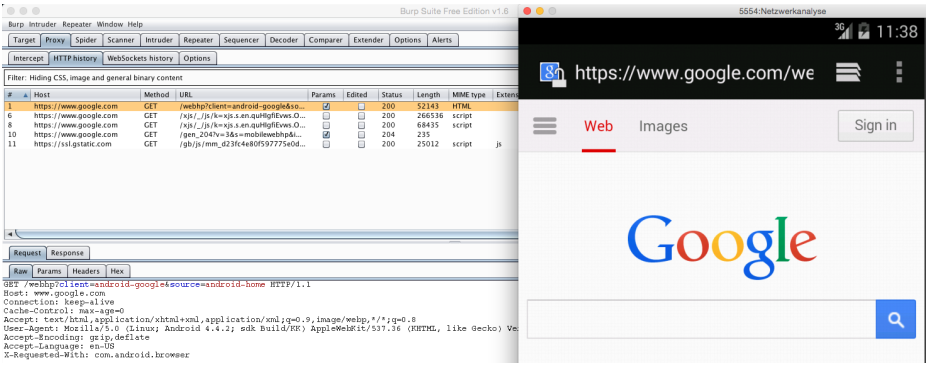


Abb. 9–3: Testen der Proxy-Einrichtung am Beispiel der Google-Startseite und eines Android-Emulators

der Applikation betroffen sind und eine sichere Übertragung der Daten zwischen mobilem Endgerät und Backend-Server nicht mehr gewährleistet ist. Klassische Szenarien für solche Angriffe in der freien Wildbahn sind manipulierte WLAN-Netze, die unter Kontrolle eines Angreifers stehen (meist an öffentlichen Orten wie Cafés oder Flughäfen).

Hat der Entwickler der Applikation alles richtig gemacht, so verweigert die Applikation die Kommunikation mit einem nicht vertrauenswürdigen Zertifikat, und unsere Aufzeichnung in der Burp Suite bleibt ohne sensible Daten aus der Applikation.

9.2.2 Man-in-the-Middle-Angriff mit vertrauenswürdigem Zertifikat

Um etwaige Fehlermeldungen bzw. Warnungen oder sogar das Verweigern einer Verbindung zu umgehen, ist es nötig, dass das Endgerät davon ausgeht, dass unser Burp-Suite-Zertifikat (das für die Verschlüsselung des Netzwerkverkehrs zwischen Endgerät und Proxy verwendet wird) ein gültiges und vertrauenswürdiges Zertifikat ist. Dies erreichen wir, indem wir das Burp-Suite-Root-Zertifikat in unseren *Trusted Certificate Store* importieren und so eine gültige Vertrauensbasis schaffen. Dazu müssen wir wie folgt vorgehen:

1. Extrahieren des Root-Zertifikates aus der Burp Suite (`Proxy` → `Options` → `Proxy Listeners` → `CA Certificate...` → `Export Certificate in DER format` → `Next` → `burp.cer`), wie in Abbildung 9–4 gezeigt.
2. Importieren des Root-Zertifikates in das mobile Endgerät.
   - **Android:** `adb push burp.cer /sdcard/Downloads/` und danach auf dem Endgerät selbst `Einstellungen` → `Sicherheit` → `Von Speicher installieren` → `Downloads` → `burp.cer` auswählen und für Wi-Fi importieren.

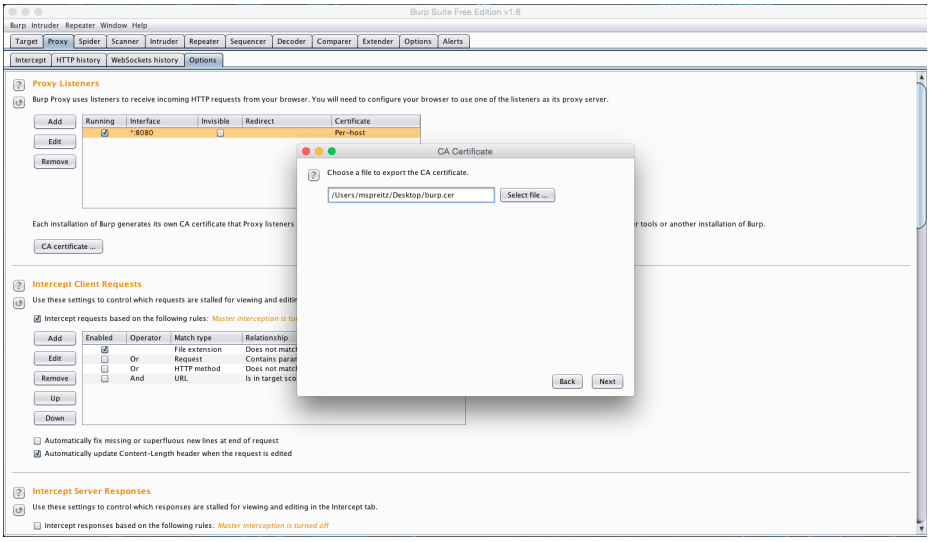


Abb. 9–4: Export des SSL-Root-Zertifikats aus Burp

- **iOS:** Am einfachsten ist es, wenn wir uns das `burp.cer`-Zertifikat per E-Mail an das Testgerät senden. Dort können wir per Klick auf den E-Mail-Anhang das Zertifikat direkt installieren. Als Alternative können wir mittels *Safari* auf die URL http://burp/cert surfen und von dort das Zertifikat importieren.
- **Windows 10 Mobile:** Am einfachsten ist es auch hier, wenn wir wie bei iOS beschrieben vorgehen.

3. Erneuter Test der Verbindung – diesmal sollten keine Warnungen mehr erscheinen.

Für Entwickler von sensiblen Applikationen gibt es nur eine Möglichkeit, sich gegen einen solchen Angriff zu schützen: *Certificate Pinning*. Bei diesem Verfahren werden bestimmte Punkte in der Vertrauenskette des Zertifikats fest verankert, sodass selbst bei gültiger Kette zu einem vertrauenswürdigen Root-Zertifikat das verwendete Zertifikat abgelehnt wird. Wie man diese Schutzmechanismen umgeht, erfahren Sie im kommenden Abschnitt.

9.2.3 Angriffe bei aktiviertem Certificate Pinning

Die Königsdisziplin ist das Mitschneiden der verschlüsselten Kommunikation bei aktiviertem *Certificate Pinning*. Dies ist auf aktuellen Geräten nur möglich, wenn der Analyst oder Angreifer Root-Zugriff auf das mobile Endgerät erhält und in der Lage ist, weitere Werkzeuge zu installieren oder die Applikation zur Laufzeit zu manipulieren. Prinzipiell gibt es zwei Arten des Vorgehens, um das Certificate Pinning zu umgehen:

- Die erste Variante ist sehr aufwendig und erfordert tiefes Verständnis des Betriebssystems sowie der verwendeten Programmiersprache bzw. eingebetteten Bibliotheken. Hier versucht der Analyst, sämtliche Stellen im Code der Applikation zu finden, in denen die SSL-Zertifikate geprüft werden. An diesen Stellen ändert er die Prüfung nun so ab, dass das manipulierte Zertifikat akzeptiert wird. Je nach Grad der Obfuskierung einer Applikation ist dies enorm zeitaufwendig.
- Der zweite Ansatz ist toolgestützt und wird im weiteren Verlauf des Abschnitts dargestellt. Hier übernimmt ein Tool die zuvor beschriebenen Aufgaben automatisiert durch sogenanntes *API-Hooking*. Problematisch ist hierbei jedoch, dass die Tools nur die API-Zugriffe überwachen und manipulieren können, die sie auch kennen. Verwendet eine Applikation also eigene Bibliotheken oder ändert sich etwas an den Systemschnittstellen, so kann es sein, dass diese Tools nicht mehr ordnungsgemäß funktionieren.

Da dieses Vorgehen auf den verschiedenen mobilen Plattformen deutliche Unterschiede besitzt, werden wir die einzelnen Plattformen nachfolgend separat betrachten.

Android

Für unser Vorgehen benötigen wir hier erneut unser bereits in den vorherigen Kapiteln beschriebenes Werkzeug zur Laufzeitmanipulation von Applikationen: *Cydia Substrate*. In diesem Fall brauchen wir die Erweiterung *Android-SSL-TrustKiller*[4] von iSEC Partners.

Diese Erweiterung sorgt dafür, dass alle Methodenaufrufe einer Applikation, die für die Verifikation eines SSL-Zertifikates verantwortlich sind, den Wert `TRUE` zurückliefern, egal ob die Prüfung erfolgreich war oder nicht. Hierdurch ist es nun möglich, das Certificate Pinning zu umgehen und wieder, wie in Abschnitt 9.2.2 gezeigt, die Verschlüsselung der Verbindung aufzubrechen. Auf unserem Testgerät gehen wir dazu wie folgt vor:

1. Installation der Android-SSL-TrustKiller-Applikation per
   `adb install Android-SSL-TrustKiller.apk`
2. Öffnen der Substrate-Applikation
   a. `Link Substrate Files`
   b. `Restart System (Soft)`
3. Starten der Applikation, die wir analysieren möchten

Nun sollten alle SSL-Verbindungen in der Burp Suite auftauchen und die Applikation ohne Fehlermeldungen oder Warnhinweise funktionieren. Wenn wir uns zeitgleich die Ausgaben von *logcat* anschauen, so sollten wir Hinweise wie die

[4] Android-SSL-TrustKiller: https://github.com/iSECPartners/Android-SSL-TrustKiller

nachfolgend dargestellten sehen. Diese deuten auf ein erfolgreiches Hooking des SSL TrustKiller hin:

```
I/CydiaSubstrate( 9999): MS:Notice: Loading:
    /data/app/com.android.SSLTrustKiller.apk
I/SSLTrustKiller(11594): getTrustManagers() override
I/SSLTrustKiller(11594): Hooking init in javax.net.ssl.SSLContext
I/SSLTrustKiller(11594): init() override in javax.net.ssl.SSLContext
I/SSLTrustKiller(11594): isSecure()
    called(org.apache.http.conn.ssl.SSLSocketFactory)
I/SSLTrustKiller(11594): getTrustManagers() override
```

Codebeispiel 9–1: Ausgabe von logcat mit Hinweisen auf aktive Android-SSL-TrustKiller-Erweiterung für Cydia Substrate

Sollte der zuvor erwähnte Android-SSL-TrustKiller nicht zum erhofften Erfolg führen, so kann ein weiteres Werkzeug vom Team der iSEC Partners verwendet werden: *Android-SSL-Bypass*[5].

iOS

Für Apples iOS haben wir mehrere Tools zur Auswahl, die uns bei dieser Aufgabe unterstützen können. Beginnen wollen wir mit *Snoop-it*, das bereits in Abschnitt 6.1.2 ausführlich beschrieben wurde.

Snoop-it funktioniert im Wesentlichen identisch zu dem unter Android beschriebenen Modul SSL-TrustKiller. Es sucht nach API-Aufrufen, deren Zweck das Überprüfen von SSL-Zertifikaten ist. Findet es zur Laufzeit einen solchen Aufruf, modifiziert es die Ergebnisse des Aufrufs so, dass jedes Zertifikat als gültig anerkannt wird und die Applikation die Kommunikation fortsetzt.

Das Vorgehen für den Analysten ist hierbei sehr einfach gehalten: Er muss lediglich den entsprechenden Schieber in den Einstellungen von Snoop-it aktivieren (siehe Abbildung 9–5).

Eine weitere Möglichkeit, das Certificate Pinning zu unterbinden, bietet die Lösung *iOS-SSL-Kill-Switch*[6] von iSEC Partners. SSL-Kill-Switch modifiziert die Low-Level-SSL-Funktionen innerhalb der *Secure Transport API*. Darunter befinden sich z. B. die Aufrufe von `SSLSetSessionOption()` und `SSLHandshake()`. Wie auch bei Snoop-it schafft es die Lösung, die Prüfung der SSL-Zertifikate so zu verändern, dass alle Zertifikate als gültig zurückgegeben werden. Um dieses Tool auf unserem Testgerät zu installieren, gehen wir wie folgt vor:

[5] Android-SSL-Bypass: https://github.com/iSECPartners/android-ssl-bypass

[6] iOS-SSL-Kill-Switch: https://github.com/iSECPartners/ios-ssl-kill-switch

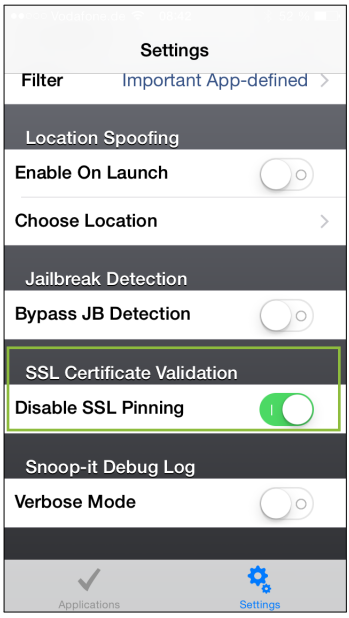


Abb. 9–5: Deaktivieren von SSL Certificate Pinning via Snoop-it

1. Übertragen der iOS-SSL-Kill-Switch-Applikation per
   `scp com.isecpartners.nabla.sslkillswitch.deb root@<IP>:/`
2. Nun verbinden wir uns per SSH auf das Testgerät.
3. Dort installieren wir die iOS-SSL-Kill-Switch-Applikation per
   `dpkg -i com.isecpartners.nabla.sslkillswitch.deb`
4. Nun müssen wir noch das SpringBoard neu starten, damit die Änderungen auch sichtbar werden:
   `killall -HUP SpringBoard`
5. Im Anschluss deaktivieren wir das SSL Certificate Pinning, wie in Abbildung 9–6 gezeigt.

Seit einiger Zeit gibt es auch eine Version 2[7] des hier gezeigten iOS-SSL-Kill-Switch, die in manchen Fällen bessere Ergebnisse liefert.

Die dritte Möglichkeit, das Certificate Pinning zu unterbinden, bietet die Lösung *TrustMe*[8] von der Intrepidus Group. Diese Lösung deaktiviert die *SecTrustEvaluate*-Methode, wodurch alle Zertifikate als gültig anerkannt werden.

[7] SSL-Kill-Switch 2: https://github.com/nabla-c0d3/ssl-kill-switch2

[8] TrustMe: https://github.com/intrepidusgroup/trustme

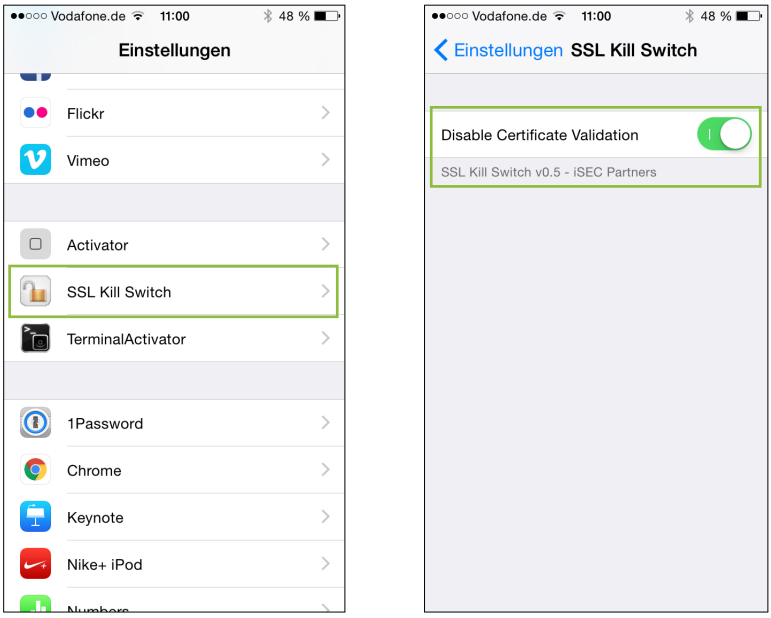


Abb. 9–6: Deaktivieren von SSL Certificate Pinning via iOS-SSL-Kill-Switch

Problematik: 64-Bit-Systeme

Zum aktuellen Zeitpunkt haben beide der vorgestellten Lösungen ein Problem mit Betriebssystemen, die auf 64 Bit beruhen (z. B. iOS 8.x und 9.x). Daher bleibt bei neueren iOS-Versionen oft nur das manuelle Suchen und Manipulieren der API-Aufrufe übrig – zumindest so lange, bis Updates der Tools bereitgestellt werden. Ein Versuch ist es meist aber trotzdem wert.

Windows 10 Mobile

Für Windows 10 Mobile sind zum aktuellen Zeitpunkt leider keine Tools und Wege bekannt, das Certificate Pinning einer Applikation zu umgehen, außer der Verwendung eines Debuggers und dem manuellen Anpassen der entsprechenden Abfragen zur Laufzeit der Applikation.

9.3 SSL-Strip-Angriffe

Der *SSL-Strip-Angriff* – oder auch *SSL-Downgrade-Angriff* genannt – ist im Wesentlichen auch ein klassischer *Man-in-the-Middle-Angriff*. Das Spezielle an dem *SSL-Strip-Angriff* ist, dass der Analyst mithilfe von speziellen Tools die mobile Applikation dazu bringt, nicht die gewünschte verschlüsselte Verbindung aufzubauen, sondern auf eine unverschlüsselte Verbindung zurückzugreifen.

Dazu entwickelte der Softwareexperte Moxie Marlinspike einen Proxy namens *sslstrip*[9]. Dieser Proxy durchsucht automatisiert die angefragten Webseiten nach eingebetteten `https://`-Links und ersetzt diese durch gleichlautende `http://`-Links. Sollte die Applikation direkt versuchen, eine HTTPS-Webseite aufzurufen, so biegt auch hier das Tool die Anfrage zu einer unverschlüsselten HTTP-Webseite um (siehe Abbildung 9–7).

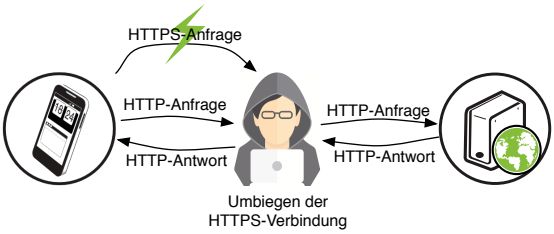


Abb. 9–7: Schematische Darstellung eines SSL-Strip-Angriffs

Gelingt dieser Angriff, so erhält der Analyst Zugriff auf alle Informationen aus der eigentlich verschlüsselten Verbindung. Darunter befinden sich sehr häufig Passwörter oder andere sensible Nutzer- bzw. Applikationsdaten. Im Unterschied zu den in den vorherigen Abschnitten dieses Kapitels beschriebenen Angriffen tauchen hier beim Benutzer meist keine Warnhinweise auf.

Diese Art des Angriffs war in den Jahren 2012 bis 2014 sehr verbreitet und gerade auch bei mobilen Applikationen sehr oft erfolgreich. In den vergangenen Jahren hat man jedoch eine Veränderung des Bewusstseins der Entwickler festgestellt und so sind heutzutage nur noch sehr wenige Applikationen für diesen Angriff anfällig. Trotz dieser Tatsache sollte der *SSL-Strip-Angriff* bei jedem Assessment einer mobilen Lösung berücksichtigt und getestet werden. Um diesen Angriff durchzuführen, gehen wir wie folgt vor:

1. Als ersten Schritt müssen wir die IP-Weiterleitung aktivieren:
   - `sudo sysctl -w net.inet.ip.forwarding=1`
   - `sudo sysctl -w net.inet.ip.fw.enable=1`
   - `sudo sysctl -w net.inet.ip.fw.verbose=1`

[9] sslstrip: https://github.com/moxie0/sslstrip

2. Im nächsten Schritt konfigurieren und aktivieren wir den Paketfilter wie folgt:
   - Wir erstellen eine Datei namens `/etc/pfsslstrip.conf`.
   - In diese Datei fügen wir folgenden Inhalt ein:
     ```
     rdr pass on en1 proto tcp from any to any port 80
     -> 127.0.0.1 port 8080.
     ```
   - Nun aktivieren wir den Filter mithilfe des Befehls:
     `sudo pfctl -e -f pfsslstrip.conf`.
3. Im Anschluss daran starten wir sslstrip per Eingabe von:
   `python sslstrip.py -l 8080 -w ./sslstrip.log`.
4. Als weiteren Schritt müssen wir nun noch den Proxy auf dem mobilen Endgerät auf die IP-Adresse des Analyserechners und Port 8080 setzen.
5. Zu guter Letzt starten wir die Applikation, die wir untersuchen wollen, und füttern sie mit Daten bzw. interagieren wie gewohnt mit ihr.

Die Datei `sslstrip.log` enthält nun alle aufgezeichneten Pakete und somit auch sämtliche sensible Nutzerdaten aus der mobilen Applikation.

Nachdem wir nun wissen, wie man die Verschlüsselung der Datenübertragung aufbrechen kann, werden wir in Abschnitt 9.4 uns anschauen, wie man die so mitgeschnittenen Daten auswertet und welche Informationen man darin finden kann.

9.4 Anwendungsbeispiele und Auswertung

Nachdem wir in den ersten Abschnitten gesehen haben, wie man sich in die verschlüsselte Verbindung von mobilen Applikationen einklinken kann, werden wir uns nun an ein paar Beispielen anschauen, wie man die aufgezeichneten Daten analysieren kann. In Abschnitt 9.4.2 lernen wir noch eine sehr nützliche Funktion von vielen MitM-Proxies kennen – die Möglichkeit, Daten zu manipulieren, bevor sie an das Backend bzw. das mobile Endgerät gesendet werden. In diesem Schritt lassen sich oft schon wichtige Änderungen vornehmen, die das weitere Analysieren der Applikation deutlich erleichtern.

9.4.1 Parameter in Paketen decodieren

Bei fast allen gängigen Applikationen ist es so, dass die Daten nicht im Klartext an den Backend-Server gesendet werden, sondern in einer codierten Form. Dies liegt meist daran, dass die APIs im Hintergrund sensibel auf Änderungen in dem empfangenen Zeichensatz oder auf bestimmte Zeichen reagieren. In vielen Fällen werden die Nutzerdaten in *Base64* umgewandelt oder URL-codiert. Ein klassisches Beispiel hierfür sind E-Mail-Applikationen, wie wir sie im Folgenden ebenfalls betrachten werden.

In Abbildung 9–8 sehen wir den Verlauf unserer Proxy-Aufzeichnung. Speziell von Interesse sind für uns hier Aufrufe vom Typ *POST* und die URL `Microsoft-Server-ActiveSync`. POST ist immer ein guter Hinweis auf Daten, die das Mobiltelefon verlassen, und somit natürlich von hohem Interesse, da sich hierunter sehr häufig Nutzernamen und Passwörter befinden, aber auch andere persönliche Nutzerdaten. In diesem Beispiel macht es uns die URL sehr leicht, da wir an ihr schon erkennen, dass es sich um eine Kommunikation mit einem vermeintlichen ActiveSync-Gateway handelt.

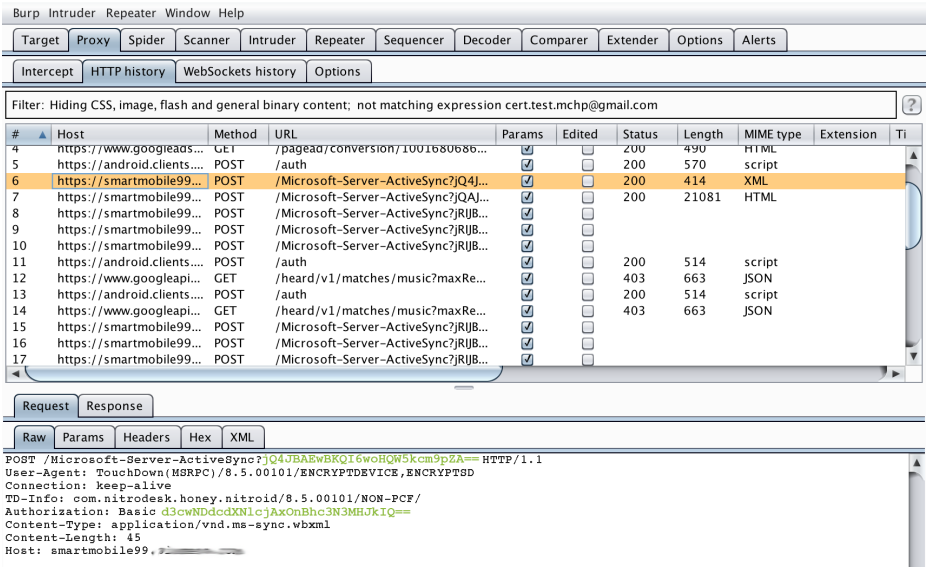


Abb. 9–8: Proxy-Aufzeichnung der Burp Suite

Betrachten wir den unteren Teil der Ausgabe, so sehen wir in der ersten Zeile die URL, die von der Applikation kontaktiert wird, zusammen mit einem codierten POST-Parameter. Dies könnte bereits der erste interessante Wert sein, der uns im Laufe späterer Analysen weiterhelfen kann. Hierzu verwenden wir die Funktion von Burp Suite, diesen Parameter an den `Decoder` zu senden und dort auszuwerten (wie in Abbildung 9–9 gezeigt).

Wie wir an diesem Ergebnis erkennen, war der codierte Parameter nur wenig hilfreich. Wir haben aber noch eine weitere Chance: *Authorization*. Dieser Wert verspricht schon dem Namen nach interessante Inhalte. Auch hier senden wir den codierten Parameter an den Decoder. Das Ergebnis zeigt die Nutzerdaten zum Abrufen der E-Mails: Domäne, Nutzername und Passwort. In unserem Fall: `ww047\user01:passw0rd!`

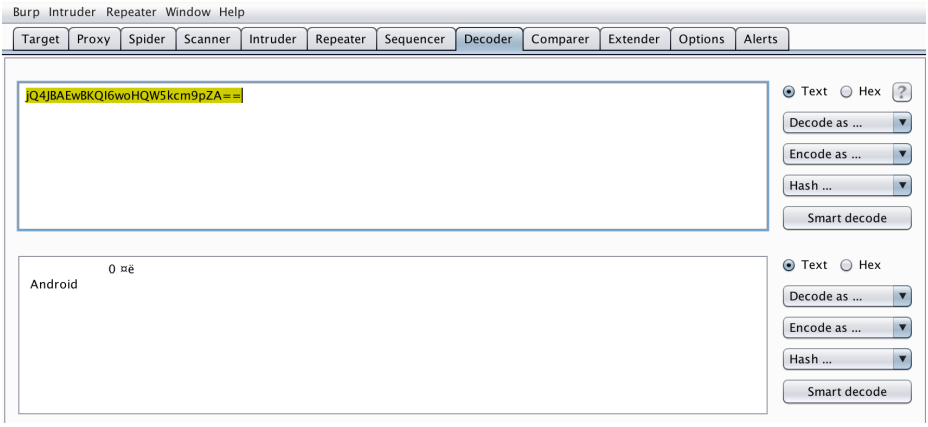


Abb. 9–9: Burp-Suite-Decoder-Funktionalität

9.4.2 Pakete abfangen und manipulieren

Nachdem wir nun an einem einfachen Beispiel gesehen haben, wie hilfreich das Decodieren von einzelnen Parametern aus dem Netzwerkverkehr sein kann, wird in diesem Abschnitt gezeigt, dass es sehr oft auch von Vorteil für unser Assessment sein kann, wenn wir bestimmte Pakete blockieren oder manipulieren, bevor sie weiterverarbeitet werden.

Das erste Beispiel zeigt, wieso es hilfreich sein kann, Daten zu manipulieren. In diesem Fall verwenden wir eine Android-Applikation, die zur PIM-Synchronisation im Unternehmen eingesetzt werden kann. Beim Einrichten der Applikation, aber auch nach gewissen Zeitabständen, versucht die Applikation, die Sicherheitsrichtlinien vom Exchange-Server der Firma zu aktualisieren. Je nach gesetzten Einstellungen können diese ein Assessment erschweren. Daher ist es praktisch, wenn wir ein solches Paket abfangen und manipulieren können. Abbildung 9–10 zeigt einen Ausschnitt eines solchen Paketes.

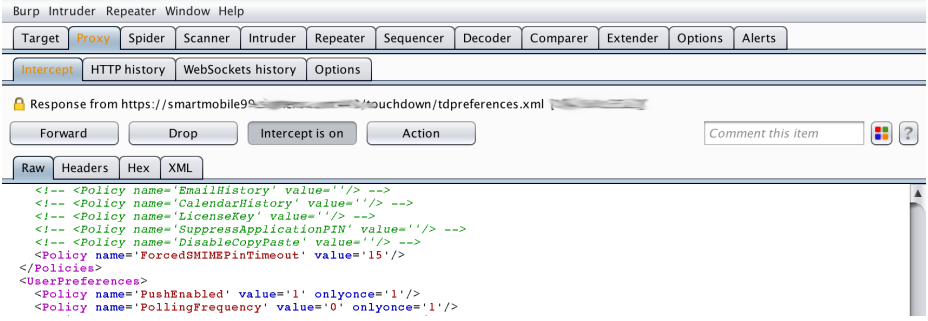


Abb. 9–10: Burp-Suite-Interception-Funktionalität

Hier können wir z. B. den Timeout der S/MIME-PIN verändern, sodass es diesen Timeout nicht mehr gibt und somit die PIN-Abfrage für das eingefügte S/MIME-Zertifikat hinfällig wird und wir an alle verschlüsselten E-Mails kommen, ohne die PIN des Nutzers zu kennen. Des Weiteren können wir in diesen Richtlinien auch noch die Möglichkeit einer Fernlöschung deaktivieren, was einem Angreifer deutlich mehr Zeit verschaffen würde. Was man bei solchen Richtlinien ebenfalls sehr häufig findet, sind Einstellungen bezüglich einer *Jailbreak-Erkennung*, also ob die Applikation auf potenziell gerootete Geräte reagiert oder nicht. Gerade solche Einstellungen können bei einem Assessment oder einem Angriff enorm hilfreich sein, da man sich als Analyst die Arbeit spart, diese Detektionstechniken manuell zu finden und zu umgehen.

Genauso wichtig wie das Verändern von Anfragen kann aber auch das Abfangen und Blockieren von solchen Nachrichten sein. Gerade Applikationen, die sich mit ActiveSync-Servern verbinden – aber auch z. B. *Mobile Device Management (MDM)*-Applikationen –, empfangen häufig per HTTPS die Aufforderung, den Container oder das ganze Endgerät zu löschen. Können wir eine solche Nachricht in unserem Proxy im Intercept-Modus abfangen, so haben wir auch die Möglichkeit, dieses Paket per `Drop` einfach wegzuwerfen, sodass das Endgerät die Nachricht nie erhält und somit auch keine Daten löschen kann. Aus diesem Grund ist es sehr wichtig, dass man während einer Analyse einer Applikation das Endgerät konstant an den Proxy angeschlossen hat und die Pakete überwacht.

9.5 Zusammenfassung

In diesem Kapitel haben wir gesehen, warum es hilfreich sein kann, die Verbindung zwischen einem mobilen Endgerät bzw. der Applikation und den Backend-Servern zu überwachen. Dies haben wir durch Man-in-the-Middle-Angriffe auf die verschlüsselte Verbindung realisiert. Dazu haben wir das Gerät an einen Proxy angeschlossen und gezeigt, wie man Zertifikatsbeschränkungen umgehen und sogar Certificate Pinning auf den einzelnen Plattformen deaktivieren kann.

Im letzten Abschnitt sind wir noch auf hilfreiche Erweiterungen der Burp Suite eingegangen – Decoder und Intercept – und haben sie uns anhand realer Applikationen angeschaut.

So konnten wir mithilfe der im Rahmen dieses Kapitels gezeigten Techniken Sicherheitsrichtlinien für Applikationen ändern oder wichtige Befehle des Backend-Servers abfangen und eine Ausführung verhindern.