WHITEPAPER

FEBRUARY 2018

Abstract

This document will discuss the interplay between Man in The Middle (MiTM/ MITM) attacks and the security technologies that are deployed to prevent them. The discussion will follow a mostly chronological exposition, starting from no protections at all and ending with robust certificate pinning backed-up by local attestation solutions to prevent tampering pinned certificate data. The reader will complete with the knowledge of what the mitigations against MITM attacks are, and how these common mitigations can be defeated. In most cases, concrete examples of mitigation implementations 'in the wild' will be given from Android application disassemblies pulled from a survey of automotive mobile apps completed by the authors.

Man in the Middle Attacks and Secured Communications

This document will discuss Man in The Middle (MiTM/MITM) attacks. It will describe the increasing levels of protection in HTTP/HTTPS communications and how these mitigations are defeated. At each level, defeat of a mitigation will give rise to another mitigation of the next order – as is often the case in security systems design. The reader will be brought up through the rationale for the design of secure communications controls, as they are borne from attacks. We will also share the results of a survey of multiple automotive mobile apps, focusing on Android.

We define MiTM as the act of intercepting and possibly modifying communications between two parties where:

- neither party detects the interception
- it is assumed that encryption and authentication are also bypassed -- when encrypted and/or authenticated communications are involved

This attack is applicable to nearly all transport-layer and application-layer protocols in some form or another. In the following, we will discuss HTTP/HTTPS.

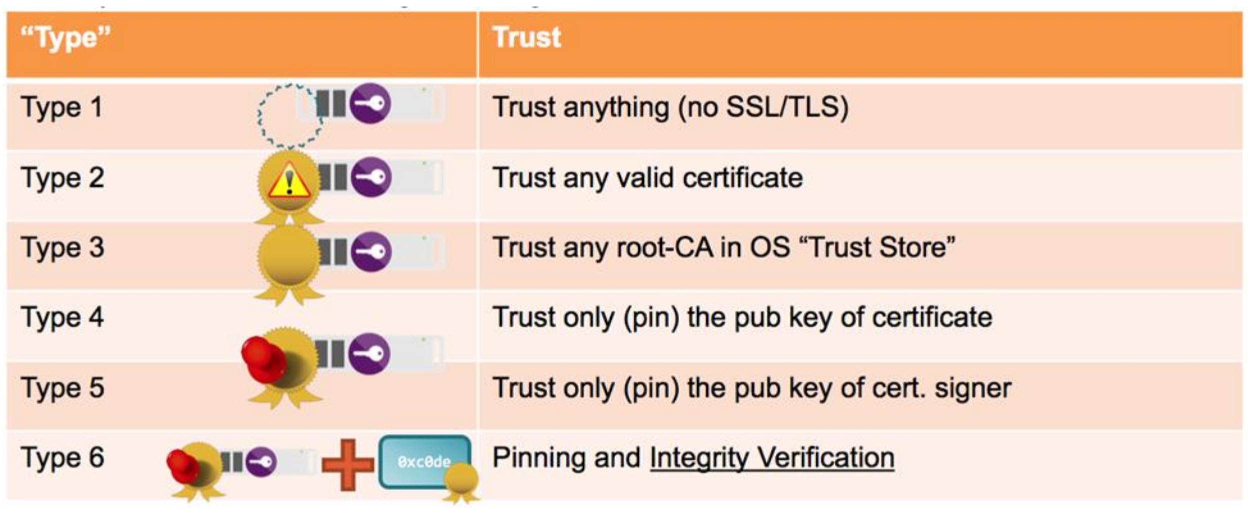
There are two *flavors* of MiTM attack worth mentioning:

- Siphon (no data modification)
- Proxy (allows full control to siphon and/or modify)

An attacker can still achieve useful goals with the lower privileges (and sometimes simpler to realize) *siphoning*.

Overview of the Types

From the least-protected to the most-protected, we will discuss the following ‘types’ of communications which are secured against MiTM attacks.

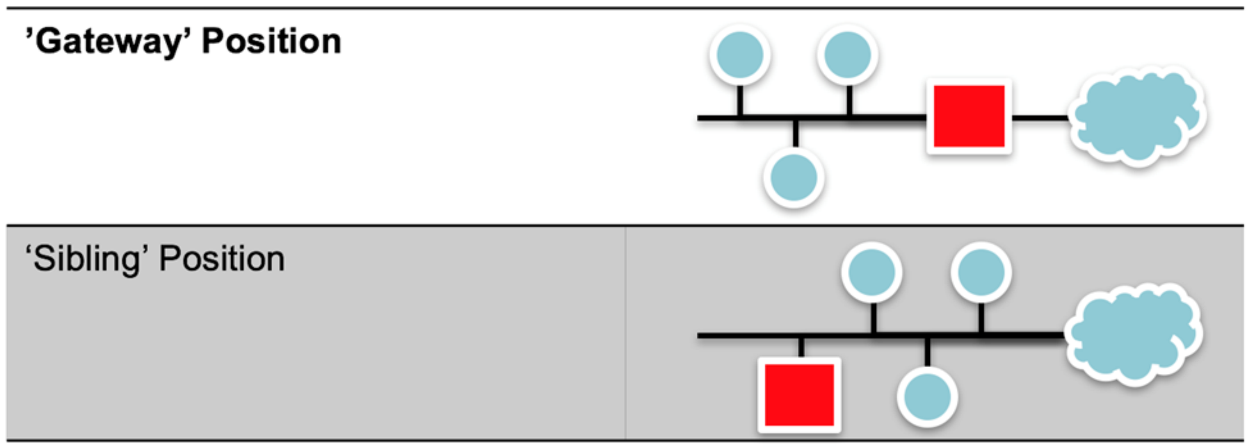


“Type”	Trust
Type 1	Trust anything (no SSL/TLS)
Type 2	Trust any valid certificate
Type 3	Trust any root-CA in OS “Trust Store”
Type 4	Trust only (pin) the pub key of certificate
Type 5	Trust only (pin) the pub key of cert. signer
Type 6	Pinning and Integrity Verification

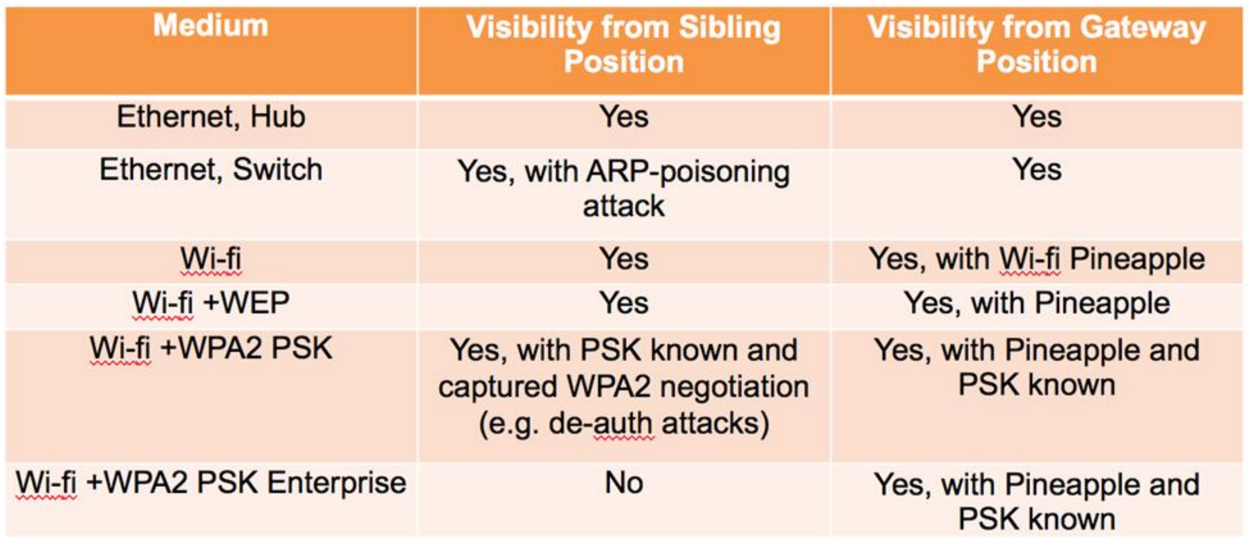
Type 1: Just HTTP (no S: SSL/TLS)

There is no encryption, integrity or authentication in place. This type of connection is “in the clear”. To attack this type of connection, only a valid ‘position’ in the network is required since, to view the unencrypted traffic, attackers need only to receive the packets.

The difficulty of achieving such a position depends on the network in question and on whether the attacker is limited to a position where their host is a network peer (sibling) of the target of the MiTM attack, or if they can be a ‘chokepoint’ of the communications – i.e. a ‘gateway’.



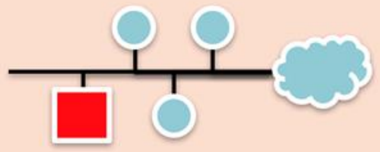
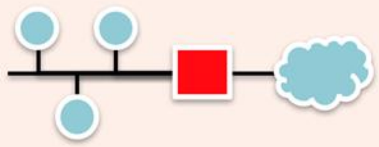
Each position yields a visibility into traffic by the attacker, depending on the medium in question. In the following, we highlight these visibilities for some common mediums and both positions.



Medium	Visibility from Sibling Position	Visibility from Gateway Position
Ethernet, Hub	Yes	Yes
Ethernet, Switch	Yes, with ARP-poisoning attack	Yes
Wi-fi	Yes	Yes, with Wi-fi Pineapple
Wi-fi +WEP	Yes	Yes, with Pineapple
Wi-fi +WPA2 PSK	Yes, with PSK known and captured WPA2 negotiation (e.g. de-auth attacks)	Yes, with Pineapple and PSK known
Wi-fi +WPA2 PSK Enterprise	No	Yes, with Pineapple and PSK known

A key takeaway is that the gateway position will yield visibility in all cases. This is a position available to attackers with physical access, that is, attackers that can take devices and stage them in their own attacker-friendly environment. This is common in early or research stages of attacker's efforts where reverse engineering plays a central role.

Throughout the document, we will offer our own estimates of difficulty - both in-field and at-home. The at-home difficulty is important to keep in mind because attackers seeking to stage remote campaigns can start with physical access to a device or mobile application and, using reverse engineering, discover vulnerabilities or create an understanding sufficient to further their own goals. In this case, difficulty in-field ranges from unfeasible to trivial; whereas, the at-home difficulty is always trivial.

		Difficulty in-field	… at-home
'Sibling' Position		Trivial (Ethernet, WEP) to Moderate (WPA2 PSK)	Trivial
'Gateway' Position		Easy (Wi-fi) to Unfeasible (WPA2 PSK Enterprise)	Trivial

Survey Results

Of all the Android applications we surveyed, none were using *type 1* secured communications. There was a minor exception. Some libraries used by the apps included code that was performing type 1 communications; however, there was no evidence that these code paths were being executed. The takeaway here is that it is important to audit and/or test the behavior of your 3rd party components.

Type 2: Trust 'any' Endpoint with a Certificate

In this type of communication, the app has the necessary implementation to set up an SSL/TLS connection; however, the implementation contains no further checks regarding the endpoint of the SSL/TLS connection. In the case of Android and iOS, the lack of checks actually requires code changes on the part of the developer, as we will show in the survey results.

Thus, the app is setting up a secured connection, but there is no verification or enforcement of validity of the certificate presented by the endpoint. In this case, self-signed certificates would be accepted – these certificates contain public keys that can be used to set up secured communications but not a chain of trust to any 'authority.' If this notion is foreign to you, please consult the Wikipedia entry on *Chain of Trust*.

Type 2 Defeats: MITM Proxies

In addition to the correct network position required to defeat type 1 communications, attackers also require an *MITM Proxy*. These tools are able to substitute certificates (self-signed or sometimes signed by an authority as-configured) for proxied requests to domains on-the-fly. Many options exist in this class of tools. E.g., burp suite, OWASP Zap, mitmproxy; also SSLStrip qualifies in this category as well.

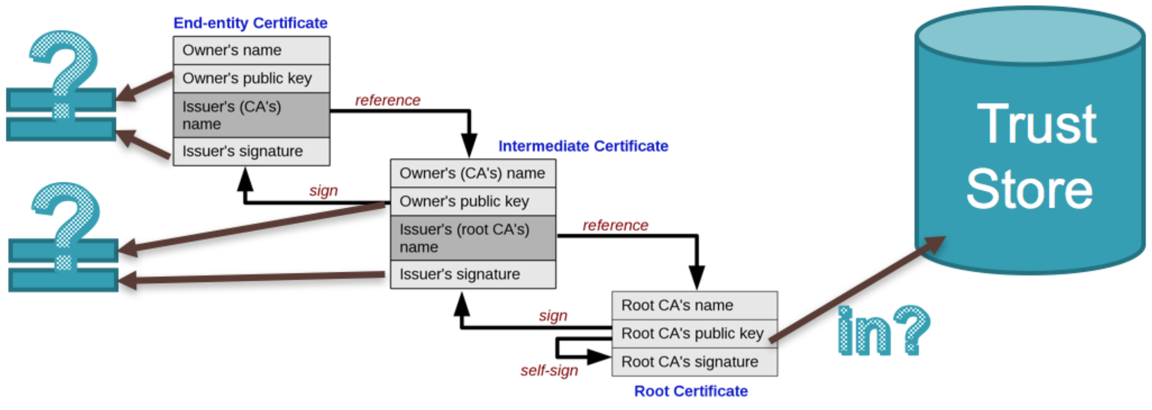
Attack	Difficulty in-field	… at-home
Proxy Https Traffic	Moderate	Trivial

Type 2 Survey Results:

Type 2 communications were found in several apps, but in non-critical areas. Type 2 was also found in library code. Reminder: audit your third-party libraries. For example, Java code that is overriding any TrustManager verify() methods is very likely disabling the default verification of certificates presented by the endpoint (especially if they are stubbed-out to return 'true'!).

Type 3: Authentication Against TrustStore

This is the OS default for Android and iOS – your mileage may vary on Linux depending on the frameworks used. In type 3 communications, the app implements the necessary calls to set up an SSL/TLS connection. The implementation includes verification that the certificates presented by the endpoint are signed by something, which is in turn signed by something, and so on, ending at being signed by something that is implicitly trusted by the OS or framework. The collection of implicitly trusted *somethings* (in this case, certificates or Root CAs) is commonly referred to as the TrustStore. Both Android and iOS (and your web browser) come pre-loaded with Root CAs in their TrustStores.



https://commons.wikimedia.org/wiki/File:Chain_of_trust.svg

Type 3 Defeats

In addition to the network position required for type 1 defeats, attackers will require a way to spoof or otherwise falsify the chain of trust being verified by the platform (OS or framework). Some means available to them to accomplish this include:

- Using a compromised Root CA – sometimes the private keys of RootCAs are compromised, or the entity controlling access to that RootCA can be paid to make use of the secret.
- Abusing improperly signed certificates – certificates as presented by endpoints and, as signed by Root CAs, have a great deal of metadata about validity. We will not go into detail here, but the metadata can include making the certificate valid for too many domains or for too much time.
- Cryptographic attacks – certificates show their public half of the public/private key pair. Guessing the private half through cryptographic attacks is possible. These attacks are usually quite difficult; however, there have been practical attacks in the past so it shouldn't be ruled-out.
- Most importantly: addition of new Root CAs to the trust store. On Android and iOS, this can happen as part of user interaction, and there are many ways to convince the users to perform the actions. In some cases, users are tempted to do so to gain access to a Wi-Fi that promises 'more speed' following the installation of a certificate. Some captive portal systems force users to install a new cert to proceed. Furthermore, there are both corporate environments and some public regions of the globe where the installation of additional Root CAs is normal and/or required.
    - NB: the mitmproxy tool includes a convenience webserver to make installation of the necessary additional Root CA semi-automatic, making intercepting type 3 communications even simpler for at-home attackers.

Attack	Difficulty in-field	… at-home
Improperly signed Certificates	Difficult	Difficult
Private key leaks	Difficult	Difficult
Custom certificate installation	Moderate	Easy

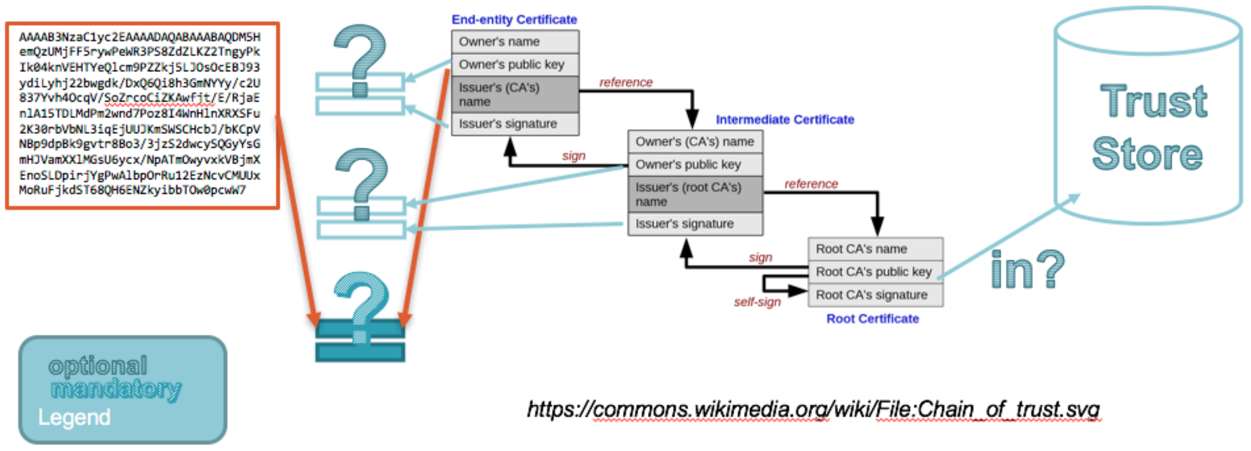
These attacks on Type 3 can be mitigated by reducing the TrustStore to a minimum set no deployment. This principle can also be followed to the trivial case where only a single Root CA is trusted; which relates to our next two types: certificate pinning.

Type 3 Survey Results

The vast majority of apps surveyed relied on the system trust store. This is to be expected since it is the OS default behavior.

Type 4: Trust the Public-Key of the Endpoint

In type 4 communications, the app has implemented the necessary calls to set up an SSL/TLS connection and, furthermore, has added its own endpoint certificate verification and enforcement code, requiring that the endpoint certificate have an expected public key value (or hash). In this case, there is no need to check whether or not the certificate provided by the endpoint is signed by any other authority, nor has a full root of trust. This is because the endpoint must possess the private half of the public key presented by the endpoint for it to decrypt the communication the app will send to it.


AAAAB3NzaC1yc2EAAAADAQABAAABAQDM5H
emQzUMjFF5rywPeWR3PS8ZdZLKZ2TngyPk
Ik04knVEHTYeQlcm9PZZkj5LJOsOcEBJ93
ydiLyhj22bwgdk/DxQ6Qi8h3GmNYYy/c2U
837Yvh4OcqV/SoZrcoCiZKAwfjt/E/RjaE
nlA15TDLMdPm2wnd7Poz8I4WnHlnXRXSFu
2K30rbVbNL3iqEjUUJKmSWSCHcbJ/bKCpV
NBp9dpBk9gvtr8Bo3/3jzS2dwcySQGyYsG
mHJVamXXlMGsU6ycx/NpATmOwyvxkVBjmX
EnoSLDpirjYgPwAlbpOrRu12EzNcvCMUUx
MoRuFjkdST68QH6ENZkyibbTOw0pcwW7

https://commons.wikimedia.org/wiki/File:Chain_of_trust.svg

Type 4 Defeats

Type 4 defeats also include the cryptographic attacks mentioned for type 3. There are no further in-field attacks. At this point, we re-emphasize the importance of the at-home attacks. These can give attackers a 'foothold' into network systems, or enable the development of exploits at larger scales. At-home attacks against Type 4 include:

- Patching the app to disable the public key check
- Patching the app to change the public key value checked
- Patching SSL Libraries to disable SSL or siphon clear communications at the boundaries of the library

  NB: all of the above patching attacks include both on-disk and in-memory variants. Some are fully-automated. C.f. the FRIDA pcipolloni/universal-android-ssl-pinning-bypass-with-Frida script.

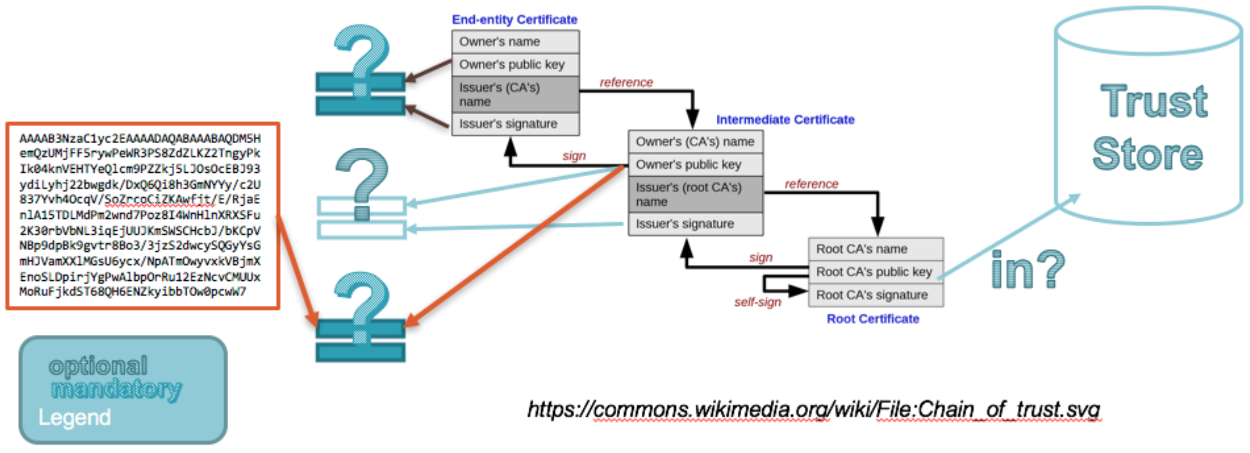
Attack	Difficulty in-field	… at-home
Public key Collision	Infeasible	Infeasible
Patch Software	Infeasible	Moderate.
Disable SSL	Infeasible	Easy

Type 4 Survey Results

None of the apps surveyed included type 4 communications; we believe this is because, as compared to the other certificate pinning methods, this one is fragile. Specifying a single valid public key value (or hash) in an app means that deployments across multiple servers or geographic areas are impossible (where sharing the private keys is infeasible). It also means that rolling a new private key on a schedule, or in response to an issue, requires an update of the app before any further communications. For more information on certificate pinning, consult the OWASP page: owasp.org/index.php/Certificate_and_Public_Key_Pinning

Type 5: Trust the Public-Key of a Signer-of the Endpoint

In type 5 communications, we have the same situation as with type 4, but with a variation. Rather than verifying and enforcing that the endpoint certificate has an expected public key value (or hash), in this case the app asserts that the signer of the endpoint-provided certificate has an expected public key value (or hash). It is necessary for the app to also assert that the signer has, in fact, signed the endpoint-provided cert.



```
AAAAB3NzaC1yc2EAAAADAQABAAABAQDM5H
emQzUMjFF5rywPeWR3PS8ZdZLKZ2TngyPk
Ik04knVEHTYeQlcm9PZZkj5LJOsOcEBJ93
ydiLyhj22bwgdk/DxQ6Qi8h3GmNYYy/c2U
837Yvh4OcqV/SoZrcoCiZKAwfjt/E/RjaE
nlA15TDLMdPm2wnd7Poz8I4WnHlnXRXSFu
2K30rbVbNL3iqEjUUJKmSWSCHcbJ/bKCpV
NBp9dpBk9gvtr8Bo3/3jzS2dwcySQGyYsG
mHJVamXXlMGsU6ycx/NpATmOwyvxkVBjmX
EnoSLDpirjYgPwAlbpOrRu12EzNcvCMUUx
MoRuFjkdST68QH6ENZkyibbTOw0pcwW7
```

Type 5 Defeats

Same as type 4, above.

Type 5 Survey Results

There was one instance that we found where this method was employed (in Android).

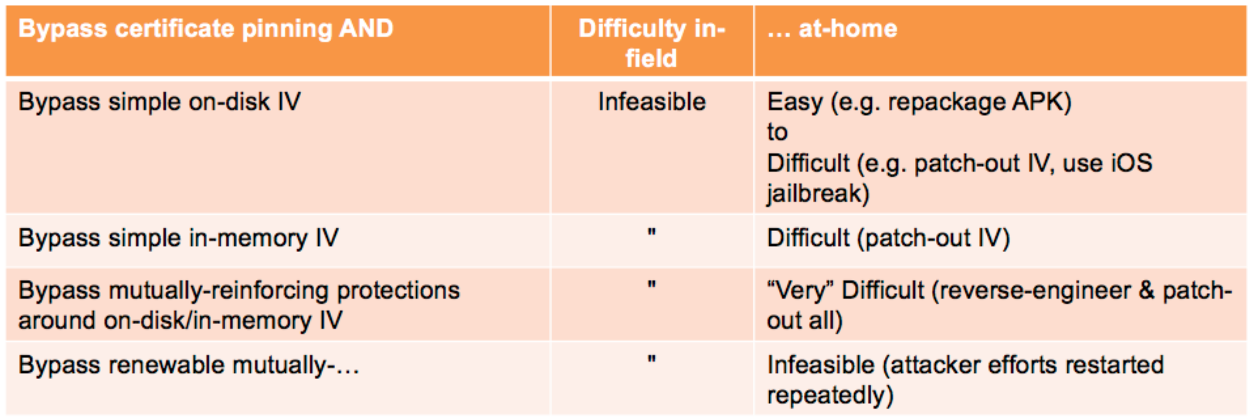
Type 6: Integrity-Verification of Public Key Pinning Data

In type 6 communications, additional protections are applied on-top-of either type 5 or type 4 communications discussed above. Here, the app is developed to include integrity verification of the asserted expected value (or hash) of the endpoint certificate.

A simple way to implement such checks is to include a ‘checksum’ assertion of the application’s on-disk file somewhere during the early phases of execution of the app. More difficult would be asserting the in-memory contents of the app are as-expected. Both of these checks (without additional mitigations) are easily defeated by at-home attackers.

Type 6 Defeats

None of the attacks to this category are (in the opinion of the authors) feasible as in-field attacks. As at-home attacks, they are very feasible. In this case, we will consider a *progression* of attacks, each borne out of the need to defeat progressively more robust anti-tamper mitigations. The details of how to create robust anti-tamper mitigations are beyond the scope of this paper; suffice it to say, that to create such a mitigation that cannot be easily disabled by at-home attackers requires a strong anti-reverse engineering component, including mitigations against both static and dynamic analysis.



Bypass certificate pinning AND	Difficulty in-field	… at-home
Bypass simple on-disk IV	Infeasible	Easy (e.g. repackage APK) to Difficult (e.g. patch-out IV, use iOS jailbreak)
Bypass simple in-memory IV	"	Difficult (patch-out IV)
Bypass mutually-reinforcing protections around on-disk/in-memory IV	"	“Very” Difficult (reverse-engineer & patch-out all)
Bypass renewable mutually-…	"	Infeasible (attacker efforts restarted repeatedly)

Conclusions

Secure communications require both encryption and authentication. Endpoints of secure communications channels should be blindly trusted, and assertion of expected conditions of the endpoints is required. Attackers can exploit the lack of these checks in-field, but also, they can tamper with applications at-home to bypass these checks. Further complementary mitigations are required to stop attackers from inspecting an application’s *secured communications* at-home.