Bazaar Wiki

Bazaar is our focused effort to make the F-Droid app store the most private and secure available, while embedding the best known methods for guaranteeing access no matter the conditions of the internet:

- share apps on your phone with people nearby using WiFi, Bluetooth, NFC, SDCard, etc
- audit installed apps by comparing them to the versions that other people have installed to make sure they are not malware
- use decentralized app stores from all sorts of organizations
- securely build and distribute app releases
- curate your own collections of media and apps

Overview

- [[Concept Note]]
- [[User Stories]]
- [[Bazaar Phase 2 OTF Proposal]]
- [[Questions + Answers]]

Activities and Research

- research published on our blog
- [[FDroid Audit]]
- [[Auditing Existing APKs]]
- [[Bootstrapping Trust]]
- [[Local Data Transfer]]
- [[OTRDATA Integration Plan]]
- [[trustedintents:Wiki|Trusted Intent Interaction]]
- [[Chained TLS Cert Verification]]
- [[Signing the Local APK Index]]
- [[Improving the APK Signing Procedure]]
- [["Swap" apps]]
- [[Swap over bluetooth (in development)]]
- [[Ideas for the Next Phase]]

Related Discussions

- posts on our blog: https://guardianproject.info/tag/bazaar/
- posts on the FDroid forum: https://f-droid.org/forums/tag/bazaar/
- [[Oct 23rd IRC Scrum log]]
- [[Nov 21st IRC log about identifying repos]]
- F-Droid and decentralized trust convo on twitter
- [[OpenITP UX Hackathon - Cydia/Community Notes]]
- [[March 26th IRC Scrum log]]

Code Repositories

- FDroid Android client - the Android app store
- FDroid server tools - the tools for managing app repos
- androidobservatory - website to present information about APKs

Relevant F-Droid Issues

Whenever possible we should try to frame our work in terms of the F-Droid development process. If we can fix issues in F-Droid by submitting the functionality that we need for Bazaar, then its a win-win.

- Resumeable downloads? - p2p and tor will mean lots of flaky connections
- Repo as virtual category in client - we will need a way to represent what is on the device on the other side of a p2p sync
- backgrounding apk download - downloading via Tor and OTRDATA could be slow
- Method for suggesting users uninstall an apk - if an APK proves to be compromised, it should be able to be revoked and the client should recognize that

Design Assets

Starting Point
Wifi-QR/IP screen: s3_wifi_QR.ai (attached)

These documents cover the UI for all of the design we've discussed as of May 30th, 2014. I've labeled them as p01 for 'phase 1'.
Illustrator file: swapUI_p01.ai
Images of each screen (22 total): swapUI_p01.zip
Diagram of the swap workflow and an outline of the other UI components: workflow_diag_p01.pdf

*Phase 2 Design
SVG files are attached below as 'swap_p2_v1.zip'

Files

s3_wifi_QR.ai	317 KB	05/26/2014	carriestiens
swapUI_p01.zip	1.28 MB	05/30/2014	carriestiens
swapUI_p01.ai	2.46 MB	05/30/2014	carriestiens
workflow_diag_p01.pdf	1.76 MB	05/30/2014	carriestiens
swap_p2_v1.zip	12.3 MB	06/16/2015	carriestiens
swap_p2_v2.zip	23.4 MB	09/15/2015	carriestiens

"Swap" apps

This is an extension of the technology implemented for [[Local Data Transfer]]. This wiki page will track less of the technical details, and more of the UI/UX related stuff.

Progress

The following screenshots are from the actual layout files being worked on in the Android client. The views were rendered by Android Studio's rendering stuff, whcich as far as I am aware, is essentially compiling and rendering as they would be on the real device (so they are quite accurate).

start_swap.png

I struggled to find a way to make the swap icon go to the left of the image. It seems that the only solution people have online for aligning an image next to the text, and then centering the text + image combined is to not do it at all. Alternatives seem to be: put the image above the text, and place the image on the left with padding (which is what I've done here). This uses the "drawableLeft" attribute of the button to place the image on the left.

select_apps_to_swap.png

It will use the exact same list item view as the main list of apps. My concern is that we don't actually have infomration such as "Summary" and "License" available for installed apps, only for those in a repo we know about. The current [[Local Data Transfer]] implementation does away with the extra info, showing instead the app icon + name + package. In the long run, we should probably try a combination of:

- Icon + name + summary + license (etc) for apps which we can get that info from a repo we know about
- Icon + name + package for installed apps not in a repo

join_wifi.png

The first view I'm implementing the logic for. It is using the same WifiStateService stuff that the Local Repo stuff uses, so that the UI is refreshed automatically when the WiFi network changes. Still need to have some feedback for the "Connecting..." state. Also, the buttons down the bottom of the screen need to have an additional style for when they are pressed.

I must admit, the orange/green still feels a little like a button, but other than that, looks great. Also, on smaller screens, the wifi icon tends to get so small that it may as well not be there.

nfc.png

Not a particularly complex view.

wifi_qr.png

Again, buttons need to have relevant styles for when they are pressed.

Distribution Ecosystem

Here is a diagram of the complete app swapping ecosystem:

(NOTE: It would be cool if this diagram also included mirrors of the servers - both those hosted by the same owners (e.g. GP repo on S3) and those hosted by 3rd parties (e.g. how debian repos are mirrored around the world).

distribution-ecosystem.png

Files

join_wifi.png	51.5 KB	06/11/2014	pserwylo
nfc.png	46.4 KB	06/11/2014	pserwylo
select_apps_to_swap.png	48.2 KB	06/11/2014	pserwylo
start_swap.png	40.1 KB	06/11/2014	pserwylo
wifi_qr.png	48.8 KB	06/11/2014	pserwylo
distribution-ecosystem.png	182 KB	12/05/2014	hans

Auditing Existing APKs

One key feature of Bazaar is the ability to audit the existing APKs already installed on the phone.

- do zero-knowledge comparison so as not to leak the entire app list to the other person
- flag when two APKs are claiming to be the same thing, but have differences.
  - use package name as the canonical ID for an app
  - what differences should be flagged? signing key, checksum, size, version name/code, etc.
- compare all APKs both system and user-installed

Alert Conditions

- Are the two apps signed by the same set of public keys?
- Are the two apps the same version code?
  - Are the two apps requesting the same permissions?
  - Do the two apps have identical classes.dex?

Next Step Actions

The app should prompt the user when it finds that APKs don't match, and should provide simple ways to trigger actions that could help the situation.

- Uninstall
- "Freeze" the app, i.e. uninstall it but keep a copy around just in case
- check APK on external site
  - https://androidobservatory.org/
  - https://openintegrity.org/

Secure Two-party Computation

- One Java framework: http://www.mightbeevil.com/framework/
- Example app to find common contacts between Android devices: http://mightbeevil.com/contacts/
  - Description: http://mightbeevil.com/contacts/description.pdf

Bootstrapping Trust

In order to have a trusted platform for sharing files, the Bazaar app needs to be installed

- how to represent the choice
  - as much auto-detection as possible
  - wizard?

User-preferred File Transfer Method

There are so many apps and techniques that people are using for sharing files on phones. We should make it straightforward for people to share the Bazaar app via their own preferred method

Implementation

- make the Bazaar APK easily available via standard Android methods
  - in Bazaar, a *Share APK* option, which triggers a SENDIntent
  - make Bazaar copy its own APK to the SDCard, and regularly verify its the same as the installed APK.

Issues

- should not encourage insecure methods, like emailing APKs
- far too broad to document
- unknown security issues

Bluetooth File Transfer

Most phones have a built-in method of transferring files via Bluetooth. Since its very local, and there is some security in the pairing process, there is some level of trust in the process.

Implementation

- test if APK can be sent, if so send it
- rename APK to apk.zip, then transfer and instruct the user to rename

Issues

- stock Android blocks transferring .APK files
  - rename file to .ZIP to transfer, then remove .ZIP on receiving device (there is no default method of renaming files)
  - zip up APK then unzip on receiving device
  - tiny unzipper app could be NFC'ed or maybe QRcoded over.
- Bluetooth pairing can be confusing

local HTTPS

The Bazaar app can run a local HTTP server that shares out the bootstrapping app, and the rest of the process, including the [[index.jar]].

Implementation

- app generates cert for signing [[index.jar]] and HTTPS
- transports
  - detect whether on wifi, and present current IP
  - create own wifi AP (WiFi Direct)
  - adhoc wifi
  - mesh

Issues

- dealing with a self-signed cert is often painful
- perhaps just HTTP only and enforce local-only traffic

NFC

NFC provides an easy way to locally swap a bit of data, which can be used to easily setup another higher bandwidth connection, like Bluetooth.

Implementation

- set up a Bluetooth connection via NFC: https://github.com/Mobisocial/EasyNFC
- install an unzipper app via NFC, then Bluetooth a zipped APK

Issues

- limited device support
- APKs not allowed to be transferred on many devices
- unproven security

ChatSecure/OTR

If ChatSecure is installed, then we have a trusted channel to communicate over. This would provide an easy way to bootstrap the Bazaar app.

Implementation

- use [[gibberbot:Sharing]]

Issues

- ChatSecure must already be installed

Kerplapp with Pinned F-Droid

We can make a version of Kerplapp for the Google Play Store that includes the ability to download an F-Droid.apk and then check the hash against an included sha256. Then it could download it from multiple sources, and still be able to verify it. CiaranG said they can give us a static URL to a specific version of F-Droid for this. We can probably also rely on the APK being in the archive repo, i.e. https://f-droid.org/archive/org.fdroid.fdroid_45.apk

Bootstrap Decision Tree

https://dev.guardianproject.info/attachments/download/1148/BootstrapDecisionTree.png

index.jar

This is a signed index of all of the apps a given F-Droid repo contains.

Chained TLS Cert Verification

Problem

The central challenge is that as we have more application repos appearing on both servers and peer devices on local networks, we need to handle the fact that the majority of these will not have certificates signed by a Root CA and cannot be pinned. As an example, the Kerplapp app mentioned runs a tiny HTTPS server on your device, and we need a way to verify that cert in a dynamic way.

TLS secret key pinning is great when you have a finite amount of known, centralized servers. We aren't doing that, so we need something more flexible. We've taken the first step with ChatSecure (which incorporates AndroidPinning and Memorizing Trust Manager, without a Root CA store), but we want to standardize this a bit more for Bazaar / F-Droid.

Definitions

- **SPKI** - "Subject Public Key Identifier", the public key, key size, and key type in a single X509 record
- **fingerprint** - the hash over a standard chunk of a key
- **pin** - a hostname, SPKI, CA-signed boolean, and optional expire date to compare all connections to
- **TOFU** - the process of prompting the user whether to add a pin or not

Flowchart

(source)

https://raw.githubusercontent.com/guardianproject/GuardianProjectPublic/master/Chained%20TLS%20Verification/Chained%20TLS%20Verification%0flowchart.png

Discussion and Resources

- Rethinking SSL Development in an Appified World
- https://www.imperialviolet.org/2011/05/04/pinning.html
- http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/
- http://tack.io/
- https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning
- [[2013-12-13 IRC conversation]]
- [[2014-08-01 IRC conversation]]

Certificate vs. Secret Key

One question is whether to use the specific certificate or just the site's key that signed the certificate in the verification. Another option is having both included in the verification chain. First, cert pins would checked, then private key pins.

If the certificate is what is checked, it is easier to implement:

- byte-by-byte comparison of the locally stored certificate versus the presented remote certificate
- Android checks APK signatures this way
- F-Droid checks index.jar signatures this way

If the secret key is what is checked, then it is more flexible:

- transitions to new certificates is easier when once expires or is revoked
- Google Chrome pins the secret key
- this is much more vulnerable: if the secret key is compromised, then the attacker could issue certs that would be trusted by the pin/tofu

Implementation

Merge existing code into one project.

- AndroidPinning: https://github.com/moxie0/AndroidPinning
- MemorizingTrustManager: https://github.com/ge0rg/MemorizingTrustManager
  - still fails miserably at hostname validation (can't be used to accept a cert for the wrong hostname)
  - MTM needs to respect expiration dates
- a version of this idea was implemented in F-Droid: https://gitorious.org/f-droid/fdroidclient/merge_requests/56

2013-12-13 IRC conversation

_hc any thoughts on the validation order? pin --> tofu vs. tofu --> pin

_hc sounds like you got the CN=IP stuff figured out too :-D

pd0x Yup! Without having to do anything scary like implement my own hostname validation

pd0x I think that pin's should override tofu

_hc what do you use as the CN then?

pd0x the CN is still the IP (to make browsers happy). I had to add a SubjectAltName extension to the cert that also has the IP

_hc right, the grand plan we discussed pin --> tofu --> CA --> prompt

pd0x You can specify an IP type for the SAN and that was enough to make things happy

_hc does MTM/tofu check the cert or the key?

pd0xhc I'm looking at the code now. It checks whether the Certificate is in the MTM keystore to decide if it's seen it before.

pd0x So it's based on the whole cert (which matches what I see with Kerplapp)

_hc but the pinning is based on the key?

pd0x if your IP changes the Kerplapp keypair will be the same but the cert will be regenerated with the new CN & SAN

pd0x and MTM prompts twice

pd0x Yes, pinning is based on the key

pd0x So if you pinned the Kerplapp keypair it wouldn't prompt for any of the certs, despite IP changes

_hc even though it checks MTM then pinning?

pd0x Yah, MTM will say "Do I have this cert in my keystore?" the answer will be no if the IP changed, then it will go to Pinning trust manager which will say "Ah, I have a pin for the SPKI on this cert" and it will work transparently

_hc I see

pd0x hmm, so it's really tofu -> pin -> CA -> prompt I suppose

pd0x since if tofu hsa a memory of the cert it'll accept it before the pin

pd0x I can't decide if that's a problem or not

pd0x the cert memory is more specific than the pin (in that it trusts an exact certificate memory and not any cert with the right SPKI)

pd0x and in theory if there was a pin appropriate for the memorized cert it would have used that on first decision and not the memorizing trust manager

pd0x so there should never BE a stored TOFU'd memory for a cert that we knew a pin for

pd0x because the Pinning manager will OK the cert before prompt to memorize

_hc I think the grand plan would be* pin-cert --> tofu-cert --> pin-key --> tofu-key --> CA --> prompt

_hc so what you added to fdroid is a version of that with pin-cert and tofu-key removed

_hc or maybe the tofu-key step is pointless

pd0x I'm not sure it buys much and will be a complicated thing to explain in a prompt to a user

pd0x "Trust the key or trust the certificate?"

_hc tofu-key might make it too complicated

pd0x Do you think we need pin-cert either? That's basically the same as tofu-cert except it allows for preloading the trust

pd0x and we can already preload the trust by key instead of cert

_hc I think pin-cert would be useful

pd0x using AndroidPinning

_hc but lots of people are lazy about https keys, because they don't really matter that much in the CA model

pd0x You think there are cases where we want to pin to a very precise cert and not the pub key?

_hc in the CA model, the signed cert is what's important

pd0x I see pins as things you mostly ship statically with a build and so you want them to be flexible enough to live through standard SSL lifecycle stuff like renewing a cert with the same keypair but a new expiry

_hc I think that people should only pin the key if they know that means they have to treat the key in a full paranoid way

_hc yes, that's true

pd0x I think we're on the right path anyway :-) maybe we should try to get this workflow merged before we add more complication to justify

_hc pin-cert means you can be lazier about the signing key, but have to be good about client/cert/pin updates

pd0x yeah, it's a trade-off

_hc pin-key means you can be lazy about cert/client updates, but have to good about protecting the ky

pd0x but you need to be good about protecting the key if you want to be able to trust the cert pin too

_hc but since pins are done by devs, and affect the dev mostly, I think we can have both in the framework, and have the tradeoffs documened

pd0x if I can steal the private key I can use the pinned cert just as easily as I can generate a new cert with the same key

_hc if someone gets the key, they can't easily get the cert they generate also signed by the CA

pd0x they don't need to though

_hc but if there is no CA in the picture, and everything that comes from that key is trusted, then the key becomes everthing

pd0x with the private key they can offer the certificate you already got signed & prove knowledge of the corresponding priv key

pd0x there isn't a need to get another cert generated/signed

pd0x if the priv key is compromised it doesn't matter if you pinned the cert or the pub key

pd0x You're equally hosed

_hc but that cert would only work for the pre-registreed domains

pd0x that's true

_hc so you need to either pwn the key and server, which are the same thing, or pwn the key and CA

pd0x hmm

pd0x or pwn the key & MITM?

_hc if you get the server, the key is on that server, and you could get the pw from RAM

pd0x you just need to get in the path of traffic from alice to bob and can use your stolen private key & the public cert signed by the CA to transparently MITM

_hc unless the server has the key on a smartcard

_hc then tey can get the pin but not the private key

pd0x hmmmmmmmmmmm

_hc hmm

_hc so if you have the private key, you can MITM? how would that work? the cert validation would fail

pd0x Why would it?

_hc as far as I understand it, HTTPS ensures you're only talking to the host that's specified in the cert

pd0x Alice attempts to connect to https://guardianproject.org/fdroid/repo and I MITM a response that offers the valid CA signed cert that the real repo would use. I have the private key that corrsponds to the public key in the certificate so I can perform the standard handshakes and everything will proceed identically to the real host

_hc if you have the private key, I suppose you could decrypt the traffic, but not inject

pd0x it ensures the hostname you connect to matches the one in the cert

pd0x Maybe I'm wrong and missing something fundamental? Haven't had much coffee today lol

*13:20

_hc so if you have control of the client's DNS and you have the server's private key, then you can MITM

pd0x I don't think you need control of the client DNS. That would let you change things so Alice connects directly to Mallory instead of Bob. I think you could also do this by being able to manipulate traffic between Alice and Bob

_hc if you have access to the traffic, you could modify the traffic I suppose

pd0x Alice detects that they are talking directly to Bob and there isn't a Mallory in between on the basis of the HTTPS certificate

pd0x with the private key corresponding to that certificate's public key you can provide Alice all the same assurances they are talking to Bob

pd0x You effectively ARE Bob with possession of Bob's private key

_hc except for the CA sig

pd0x which is on the certificate that you provide

pd0x You can take the byte for byte identical certificate chain offered by the real Bob and provide it as your own

_hc a new cert, or the original one

pd0x What prevents you from doing that normally is that you need to also have possession of the private key to use it in SSL negotiation

_hc to do that, you'd have to change the clients mapping of hostname to IP

pd0x not if you're injecting traffic in the conversation between Alice and Bob's IP

_hc yes, if you have the server private key, and router-level access to their traffic, you could modify the traffic

_hc without changing the IP

pd0x Right, that's what I'm saying

pd0x and pinning to the whole cert, or the public key doesn't change the outcome

_hc I don't know if I'd call that a MITM since there is no redirection

pd0x fair

pd0x have I convinced you that adding cert-pinning isn't a meaningful addition?

_hc yeah, pinning certs would give only a little bit of extra protection at a higher risk of usability penalty

_hc pinning certs would help with the situation where someone got the private key, but didn't pwn the server

_hc so I guess its something like tofu-cert --> pin-key --> CA --> prompt

_hc but then the question is* do you tofu the cert after its been verified by pin-key?

_hc probably not

pd0x Nope

_hc then it shouldn't it be pin-key --> tofu-cert --> CA --> prompt

_hc or maybe even* pin-key --> tofu-key --> tofu-cert --> CA --> prompt, with tofu-key being an "advanced option"

_hc for kerplapp, we'll need tofu-key

_hc but my guess is that most apps will want to tofu-cert

pd0x that's going to take some patches to the Memorizing Trust Manager

_hc Ge0rG was open to that

_hc I think the idea was to start with AndroidPinning and MTM, and make a new lib

pd0x that makes sense

_hc I guess the last question is whether to tofu-key, tofu-cert, or both

2014-08-01 IRC conversation

***hc**: hey Ge0rG*
***_hc**: saw your email*
***Ge0rG**: chatsecure is fixed already, https://github.com/n8fr8/Gibberbot/commit/b2d0b2a71f6317e055e63b73540974807cfbe956*
***_hc**: the change in chatsecure is simple since it is just removing a null. but in fdroid is not so simple since a default trust manager is being passed.*
***_hc**: does MemorizingTrustManager then connect to the system trust manager using the new constructor?*
***Ge0rG**: MemorizingTrustManager will fall back to whatever you supply, and to whatever that will fall back.*
***Ge0rG**: the system trust manager will only be used if the pinning trust manager uses it*
***Ge0rG**: which IIRC it does.*
***Ge0rG**: technically, MemorizingTrustManager always only had a fall back to one trust manager. the first parameter was a wrongly exposed local variable.*
***_hc**: this first parameter? do you mean the third parameter, the one that was removed?*
***Ge0rG**: from the three parameters, the second was removed. both chatsecure and fdroid incorrectly used it, instead of the third one.*
***Ge0rG**: but of course, the error was in exposing it at all.*
***Ge0rG**: what MemorizingTrustManager exposed as the second parameter was its own trustmanager based on its local keyfile. overwriting that essentially broke MemorizingTrustManager*
***Ge0rG**: the former third (now second) parameter is the default trustmanager that MemorizingTrustManager should use if it does not have a server cert stored.*
***Ge0rG**: so you supply AndroidPinning as the new second parameter, and MemorizingTrustManager will do the follwing: a) check local keystore, b) check AndroidPinning, c) ask user*
***Ge0rG**: the first succeeding one will authenticate the connection*
***_hc**: what would be useful is if it would check AndriodPinning before MemorizingTrustManager*
***Ge0rG**: could you please explain why?*
***_hc**: if there is a successful pin, why check anythign else?*
***_hc**: the pinning form is the simplest to get right, so it seems it should be the most trusted*
***_hc**: what happens if something adds a MITM cert for a hostname to the local keystore?*
***_hc**: then the pinning will be ignored*
***Ge0rG**: what should happen if there is no pinning?*
***_hc**: I don't understand*
***Ge0rG**: as long as either MemorizingTrustManager or AndroidPinning will accept the certificate, the order does not matter.*
***_hc**: you mean if AndroidPinning is installed but without pins?*
***_hc**: the order does matter*
***_hc**: say you have a pin for google.com*
***Ge0rG**: AndroidPinning does not allow pinning hostnames. it only has a whitelist of certificates.*
***_hc**: then someone adds a MITM cert for google.com to the local keystore*
***_hc**: ah, ok, I didn't realize that*
***Ge0rG**: so AndroidPinning will only tell you "I don't know this cert, I fail now". and MemorizingTrustManager will happily ask the user*
***Ge0rG**: there is no "this cert does not match what I know about google.com, OMGMITM"*
***Ge0rG**: it would make a useful feature for AndroidPinning though. Or for MemorizingTrustManager*
***Ge0rG**: My goal with MemorizingTrustManager is to integrate all the interesting things you discussed for chained cert trust*
***Ge0rG**: and maybe also DANE*
***_hc**: yeah, the chaining only works with pinning with the hostname*
***Ge0rG**: I'm going to add an option to MemorizingTrustManager to tell the user that a certificate has _changed.*
Ge0rG: but that will only affect certificates stored in MemorizingTrustManager, not the system-approved ones.
Ge0rG: unless you supply null as the defaultTrustManager, which makes MemorizingTrustManager distrust everyone
Ge0rG: so back to your question: with the current design of AndroidPinning, it does not make any sense to check pinning before local trust store.
Ge0rG: if you change AndroidPinning to pin hostnames, you might add a defaultTrustmanager parameter in there and pass MemorizingTrustManager
Ge0rG: that way, if AndroidPinning succeeds, MemorizingTrustManager is queried, and if AndroidPinning fails, the connection fails.
Ge0rG: however, with X509TrustManager you can not easily pin hostnames.
Ge0rG: which is a completely separate issue.
_hc: god this stuff is a mess
Ge0rG: read the link from my mail.... :>
_hc: I skimmed it
Ge0rG: that must suffice then
_hc: for fdroid I think it'll make sense to use only AndroidPinning and MemorizingTrustManager, and distrust everything else
_hc: there is already a tofu dialog for the repo signing key, so that dialog can also function for the HTTPS certs
Ge0rG: AndroidPinning trusts the system by default. or better. AndroidPinning perfoms (pinned AND system_trusted)
Ge0rG: with MemorizingTrustManager it is then (pinned AND system_trusted) OR user_accepted
_hc: ah, that provides the hostname check then
_hc: system_trusted
Ge0rG: you wish.
Ge0rG: that is completely orthogonal. but if you are using the default HTTP AndroidPinningIs, they do check hostnames
Ge0rG: I assume what you want to achieve is (pinned OR user_accepted)
_hc: so does MemorizingTrustManager check hostname-cert?
_hc: well, it should continue to check the system store whenever possible since that is insurance from the lib making security worse than default
_hc: I think it needs to be more complicated than (pinned OR user_accepted)
Ge0rG: but you see how (user_accepted AND system_trusted) is worthless for most private cloud installations?
Ge0rG: you are probably right.
_hc: I think it should be like:(pinned OR user_accepted)
_hc: oops
Ge0rG: "(pinned AND system_trusted) OR user_accepted" is actually pretty reasonable, provided you have hostname-pins.
Ge0rG: which we don't

Ge0rG: actually, it all boils down to "do we trust the user to do the right thing"?
_hc: (pinned AND system_trusted) OR (user_accepted-ca_signed AND system_trusted) OR (user_accepted-self_signed)
Ge0rG: what about signed_by_user_created_ca?
_hc: then user_accepted can be automatically set on the first time if it is system_trusted
Ge0rG: ah, wait.
Ge0rG: you are adding state now.
_hc: you mean like cacert?
Ge0rG: right.
_hc: meh
_hc: does not seem worth supporting, but I suppose it would be possible
Ge0rG: I define "user_accepted" as "there was an MemorizingTrustManager popup and the user clicked 'always'"
_hc: if there is a user-ca, they should import it manually into the system store
_hc: that's the intended model
Ge0rG: so if there is such a popup both with self_signed and (ca_signed AND system_trusted), it does not make sense to differentiate these
Ge0rG: on android 4.4, you get really scary warning popups if you have a CA added
_hc: c'est la vie
Ge0rG: that's exactly what they said when I started developing MemorizingTrustManager.
_hc: that is an issue of android 4.4
_hc: you can't fix all the issues in this library
Ge0rG: that's an issue that is not going to go away magically.
_hc: that is a recipe for security disaster
Ge0rG: like SSLSocket mis-design being an issue of Java.
_hc: that is separate
_hc: the CA system is not really designed to handle user configured CAs
Ge0rG: we can't force google to do "the right thing". especially not if we don't know what the right thing is.
_hc: so trying to force that case in the MemorizingTrustManager library is a mistake
Ge0rG: I don't see the principal difference between self-signed and private-ca-signed.
_hc: the CA?
Ge0rG: from the users perspective, that is.
_hc: that's a huge difference in the eyes of the CA verification system
Ge0rG: the user just wants his private cloud deployment to work, instead of throwing nasty error messages onto him
_hc: sounds like they should get a proper certficiate then
Ge0rG: I see how trusting a CA is more problematic than trusting a single host cert. However, trusting a CA-signed host cert might be acceptable behavior.
Ge0rG: not everybody can get a valid certificate as defined by the CA extortion racket.
_hc: I'm not denying it is a problem, I just don't think it can be solved here without weakening the security for the rest of the cases
_hc: that does not seem worth it
_hc: that sounds like it should be a separate library then
Ge0rG: I don't see how this is a different problem.
_hc: with support for the private-CA, the library uses the CA intact and unchanged
_hc: the CA system
Ge0rG: currently, if you tell MemorizingTrustManager to always accept a CACert-signed certificate, MemorizingTrustManager will from now on accept all CACert-signed certs.
Ge0rG: I suppose this is a security problem.
_hc: yes, that's not good
Ge0rG: so I'll change it to only trust the given cert.
_hc: I think here's the solution: categories certs by trusted signing key (CA) and untrusted (private CA and self-signed) and
Ge0rG: this will mean that self-signed and CACert-signed will be handled the same way.
_hc: MemorizingTrustManager will TOFU prompt the untrusted on first use
Ge0rG: this is how it is done already.
_hc: for trusted ones, the library will check the CA sig and mark as trusted if the CA sig is good, otherwise prompt the user
Ge0rG: the system trust manager performs the CA check already.
_hc: so MemorizingTrustManager trusts the private key?
Ge0rG: I only ask the user if thsat fails.
Ge0rG: MemorizingTrustManager trusts the certificate, not the key.
_hc: the point for us is to make TOFU work like AndroidPinning, so that it is always checking the TOFU and the CA
Ge0rG: so you want an error if the server's cert changed?
_hc: MemorizingTrustManager trusts the signing certificate? or the TLS cert?
Ge0rG: currently, if you tap "always", MemorizingTrustManager will trust the whole chain of the presented cert, including CA and intermediate.
Ge0rG: I can see how this is worse than only trusting the given server cert.
_hc: I think MemorizingTrustManager should trust the SPKI of the certificate itself
_hc: like AndroidPinning
_hc: not the whole chain
_hc: trust the server's private key
_hc: not the CAs
_hc: or signer
_hc: then the server can freely generate new certs as long as the private key is the same
_hc: that is the pinning model
Ge0rG: I need to check how I can store the public key info of a server
Ge0rG: and let a default TrustManager verify that.
Ge0rG: currently, MemorizingTrustManager will re-ask if the certificate changed.
Ge0rG: unless it was signed by the same CA.
_hc: I am pretty sure that AndroidPinning uses the SPKI of the cert, so check there
_hc: https://www.imperialviolet.org/2011/05/04/pinning.html
Ge0rG: yeah, the question is if I can store it in my KeyStore.

Ge0rG: I don't really want to rewrite most of MemorizingTrustManager's backend
_hc: " the SubjectPublicKeyInfo is not just the public key bit string. The SPKI includes the type of the public key and some parameters along with the public key itself."
_hc: you'll just need to implement a check of algorithm and keysize, I beleive
_hc: the public key can be stored in the keystore
Ge0rG: I use the cert's DN as the keystore key.
Ge0rG: so a changed cert will impose some challenges.
_hc: in your implementation?
Ge0rG: right
_hc: it should be straightforward: when there is a new cert, check that the DN and SPKI match, and if so, swap in new for old
Ge0rG: or just ask the user?
_hc: mind if I post this conversation to our public wiki?
Ge0rG: what if the private key was leaked, and Mallory created a new self-signed cert with it?
_hc: asking the user would defeat the purpose of pinning based on SPKI
Ge0rG: do it please.
Ge0rG: silently replacing a cert is the opposite of pinning.
_hc: the pin is based on the key, not the cert
_hc: so the cert can change, i.e.when one expires, but the key must remain the same
Ge0rG: the PKI model implies that a key can be leaked once its certificate is expired
_hc: that's not Google's assumption
_hc: or moxies
Ge0rG: we can't marry PKI and SPKI without creating frankenstein's baby.
_hc: since both of their pinning is based on key
Ge0rG: neither google nor moxie are your typical users.
_hc: millions of people use chrome, which includes pins to google's keys
_hc: who are the "typical users" that you mention here?
Ge0rG: people who can't afford a verisign cert for their owncloud instance hosted on a raspberry pi
_hc: when you get a certificate from a CA, they assume you are reusing the private key
_hc: if people don't know that you should keep private keys secret, they have much bigger problems than a bad pin
Ge0rG: you might be reusing the private key when you are requesting a new certificate, but what do you do with the key once the certificate has expired?
Ge0rG: I'm just saying that a change of the server certificate, even with the same private key, is a noteworthy event
_hc: for some people
_hc: sure
_hc: the vast majority of people using services based on TLS have no idea that there are even certificates, let alone that they expire
Ge0rG: so, what can we do?
_hc: I personally don't see a need to notice when a cert has changed as long as the DN and SPKI are the same,
_hc: and its pinned
_hc: I need to get to writing a proposal due very soon, but good conversation, looking forward to seeing what you come up with
Ge0rG: sure.
Ge0rG: another thing that bothers me is: if I add a cert for a mismatching hostname to my keystore, my local trustmanager will happily accept the certificate itself for any hostname.
Ge0rG: you might want to pull MemorizingTrustManager cd9bbf8
Ge0rG: https://github.com/ge0rg/MemorizingTrustManager/commit/cd9bbf8f7cc3cffa1abe1a7a2c775f345e7c489f
Ge0rG: the same applies for chatsecure.

Trust On First Use/Persistence Of Pseudonym

TOFU/POP is a trust model for verifying the identity of the other side of a secure connection. It is used in SSH and other protocols. The idea is that you blindly trust the identity of the other side the first time that you see it. That trust is then stored to be checked in the future. So in a sense a pseudonym for the other side has been created and persisted since there is no other verification that the other side is who they say they are.

Core Concept

This is the concept for the first phase of Bazaar. For the second phase, see [[Concept Note]]

A worrying trend that is a major impediment to current mobile users' basic freedom is that increasingly users are moving away from the open (or semi-open) web, and towards download applications. Apps themselves are not necessarily bad, though if not open-source, they do inherently lack the power of "view source" that made the HTML-based web spread in the first place. It is the fact that access to apps are controlled by stores or markets, curated and controlled by the device manufacturer or mobile operator. These stores provide little transparency to why certain apps are allowed in or not, or removed, and even worse, often outright censor apps deemed to be incompatible with some sort of unclear rule of law or ethical guidelines.

It is in fact true that the fears of SOPA/PIPA style censorship, not to mention Great Firewall style censorship, are already well underway in many of the mobile stores and markets. While Google has provided some amount of openness with Google Play, at least at point of publishing, they still log every install, active use and can retroactively remove or update apps installed on any device. In this model, the store or market is not so much a filter, as a complete backdoor.

There are however some benefits to the app store model:

- Centralized, semi-trustworthy distribution point backed by a known entity
- User feedback and reputation system
- Malware testing, scanning and filtering
- Binary signature verification on download
- Tracking of downloads and active use by region, device, language for app publisher

Here is a list of problems again we see with the centralized distribution model:

- App Markets are centralized, closed ecosystems, or worse authoritarian censor-states
- Some stores are incompatible or hostile to open-source / free software
- Most require registration of developers and users
- App Market back-end tracks all downloads and usage
- App Markets can remove applications retroactively from devices
- App Markets have unclear security or privacy requirements, like HTTPS or support for circumvention or anonymity proxies
- App Markets contain content that is protected by DRM, or otherwise not easy to reshare
- App Markets are driven by a premium for-fee model

With that, we propose to engage in creation of a new model for secure and social app distribution, because we recognize the benefit of apps, and some of the benefits of the current store and market model, but believe we can build a better, more social, decentralized model, that can better serve users in both free and heavily surveilled regions of the world.

Instead of a vending machine model, employ a swap, barder, trade design.. think more of a Moroccan Souk or Flea Market (or perhaps just a jam session or free library). The emphasis is not solely on freely licensed, like with the F-Droid project, but more about freedom to curate and re-share through a non-centralized model. We know that ringtones and mobile 3gp videos spread rapidly through Bluetooth in some parts of the world, and we think that same phenomenon can be taken advantage of here.
Will be based on existing projects focused on new modes for app distribution:
- Guardian Power Up bundle installer (https://github.com/guardianproject/powerup)
- F-Droid open source repo app (http://f-droid.org/)
- Market itself is secure, privacy by design
- provides very secure, hash verfied downloads of all installer files can download apps via Android Market (if available) or via direct HTTPS, including over Tor or other proxy no logging, no analytics, no user backdoors, or tracking
- if root enabled, can modify core device (update CACerts, change fonts, remove bloatware)
  Person-to-person/device-to-device sharing
- can also deploy with a built-in bundle of compressed apps on SDcard for quick install
- every item can be easily shared via Bluetooth, NFC or WIFI lan

Allow users to define their own collections or bundles of apps and content to be shared (an "album")
- Provide a default set of curated collections
- apps, books, videos, podcasts, etc.
- must be freely licensed content and code
- Also provide specialized categories for relevant interests
- privacy & security apps: guardian apps, whisper apps, droidwall, beem, etc
- hacker, open-source culture/dev
- free culture: doctorow books, CC licensed, wikipedia
- media content can include epubs, videos focused on tech training, human rights, advocacy, etc

FDroid Audit

- [[Initial FDroid Audit by pd0x]]

Relevant Code repositories

- Android client: https://gitlab.com/fdroid/fdroidclient
- build and repo tools: https://gitlab.com/fdroid/fdroidserver
- collection of apps built by fdroidserver: https://gitlab.com/fdroid/fdroiddata

Architecture

- User Stories
- [[Bootstrapping_Trust]]
- [["Swap"_apps]]

Useful Documentation

- Installing_the_Server/Repo_Tools
- Setup an FDroid App Repo

Proposed Extensions

These are important parts that are prototyped but not deployed.

- Include links to mirrors in repo metadata
- Reproducible Builds
- Verification Server

Files

skype customer service.jpg	47.1 KB	07/22/2016	Anonymous

Initial FDroid Audit by pd0x

(this is the notes from the Guardian Project's initial audit of FDroid as part of the Bazaar project)

h2.

Rough notes being transformed from a research journal to a more finely honed state.

Kerplapp P.O.C

- **Kerplapp** - dropping apps onto droids.
- In-progress code dumped to a Github branch

F-droid Notes

- Results of looking at the F-Droid model for app delivery, specifically from a security audit angle.

Overview

- How does F-Droid Work?

Server

- Simple design that requires only a static HTTP server.
- Repos can be either so called "binary repos" or built from source
  - Binary repos host pre-built APKs (the Guardian Project repo is one such binary repo)
  - Source repos build APKs from source, signing all apps with a unique-per-pkg randomly generated signing key
- Quickly stand up an F-droid repo dir with:

```
cd /repo/dir; python -m http.server
```

- Metadata contained in index.xml
  - Signed repos also have a index.jar (detail later)
  - Hierarchy of repo -> app -> apk
  - Create a signer repo out of an unsigned repo with:

```
jar cf index.jar index.xml
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore mytest.keystore index.jar mykeyalias
```

  - If an index.jar is present the signed index.xml located within the JAR should be considered authoritative
  - The index.xml is loaded on first-add of the repository, if it defines a public key attribute for the repository subsequent repository requests will use index.jar pinned to the public key (TOFU).
- APK Signing for from-source repos has issues:
  - When a new pkg is seen and there isn't a key in the keystore for the pkg (more on that follows) a new key must be generated and put in the keystore
  - Keygen defaults are good, RSA2048
  - Key alias is auto generated by taking the MD5 of the pkg string and truncating it
    - Decreases resilience of the hash
    - Thankfully relies on second preimage resistance of MD5 to find a collision with another pkg string
      - Should still be redesigned, MD5 has been demonstrated broken for realistic first pre-image attacks in the SSL/X509 space.
    - If you could find a colliding pkg string you could have the source code for your app built and signed with the private key of another app in the same repo, allowing you to utilize the shareduserid feature of Android to access the data of the other app.
- Server relies on HTTPS for transport security
  - Should pin certificates, does not
- Due to the way the index.xml metadata is signed, replay and freeze attacks are possible.
  - An adversary can archive the index.jar and matching APKs from the signed repo across several updates/releases.
  - On a subsequent MITM they can replace the current signed index.jar with a previous version. The signature on the metadata will verify and the APKs can be replaced with the version that matched the old index.jar.
  - This attack allows an adversary to perform a 'freeze' attack (i.e. keep all clients at a fixed repository state, regardless of updates)
  - This attack also allows an adversary to pick an exact moment in history to freeze clients at. This gives the adversary the flexibility to decide which version of apps 'first install' clients get (choosing an old version of an apk only works on first install clients as we assume anyone running a version newer than the frozen app at install time can't be forced to downgrade their installation due to Android update semantics).

Client

- Clients add repositories by URL
  - Index.xml is fetched first, used to TOFU a pubkey for the index.jar
    - No technical reason why the index.xml must exist. Perhaps facilitates in-browser viewing, but an attacker can replace index.xml without knowing the signing key, meaning it shouldn't be used to make decisions about the repo in the presence of a index.jar

- APKs are subject to normal Android code signing policy
  - **MUST** be signed
  - TOFU, first install the pkg is associated with the certs used to sign it
  - Subsequent updates to the pkg must be signed with exact same set of certs
- APK hashes and the first signature on the APK are included in the index.xml metadata for the app. Client app verifies that the hash of the binary matches, and that the hash of the first signature on the APK matches. Note: F-Droid only processes single certificates for signed APKs. This seems to be an assumption based on the F-droid build process signing all pkgs with one unique randomly generated signing certificate. A so called "binary" repo could have apps that are signed with multiple certificates, only the first is included in the index.xml and authenticated by clients
  - This allows a (rather pointless) attack in which an adversary can add their signature to applications from the repo in a MITM attack. If they MITM the first install, the APK will install as the first certificate matches the one in the index.xml and the second signature is ignored. Subsequent updates from the un-MITM'd repo are unable to be installed as they lack the 2nd signature and break the Android code signing policy. If the client had previously installed the single-signed app, the adversaries MITM'd double signed copy won't install. An adversary in a persistent MITM role could repeat this process for every APK on the wire, preventing timely updates.

Ideas for the Next Phase

Why FDroid?

There are two main approaches to keeping apps updated: an "app store" model and a "self-update" model. The "app store" model is generally the approach taken by package management systems that are integrated into the core OS, so for example the Debian package management system, Apple App Store, Google Play, etc. Many apps instead include a way that they can update themselves, for example Firefox, Chrome, etc. The self-update model puts more control in the hands of the app's developer, while the app store model puts more control in the hands of the user.

- technically, it is possible to do all the same things with both models, so to understand the strengths and weaknesses, we need to focus on the user experience.

- F-Droid is the best developed alternative app store for Android that is free software.
- it already is translated into many languages, including Arabic, Chinese, Farsi, Uigher, and Spanish.

With the app store model, it is much easier for the user to decide when and what to update since all updates are presented in a single UI. This UI is also the same UI that users use for getting new apps and removing installed apps.

Users operating in high risk areas generally have more awareness of what their current risks are than the developers of the software in question, for example which networks are safer and which are more heavily monitored, or which networks can be traced to the user versus ones which are more anonymous.

For example, if a user wants to ensure that all app downloads and updates go through Tor, with F-Droid, they can just set the proxy in the F-Droid settings, root access is not required.
The app store model also lends itself to much more flexibility of which transports to use because there is a central place to manage connectivity.
- HTTPS
- tor hidden service
- amazon S3 and other cloud services
- local wifi to on-device repos
- bluetooth to on-device repos
- OTRDATA
- SDCard

The way that the current f-droid.org app repository is structured has a major weakness: all apps there are signed by a specific f-droid.org key rather than the developer's official release key. This means that there are two valid APKs for each released version of an app, but they have conflicting keys. This is not a design feature of f-droid.org, but rather a side effect of the goal of verifying that the apps are built entirely from freely available source code. The f-droid.org team is working on turning this weakness into a major strength: with fdroid verify, the f-droid.org build infrastructure will build APKs from source as it always has done, but instead of signing the APK with an f-droid.org key, it will instead verify that the developer's APK matches exactly the APK built by f-droid.org. And since it matches, the developer's original signature can be freely swapped into the APK built by f-droid.org. This turns f-droid.org into a build verification service, reproducing builds made by the original developer while eliminating the duplicate f-droid.org key.

- like Tor Browser Bundle's reproducible build process, except it will be also deterministic most of the time
- for pure Java Android apps, this is not difficult to setup
- for apps with native components, it is also possible, we have done it with out Lil' Debi app

Modules of work

Build and Distribution Integration

- Deteriminstic builds sound like some pedantic technical process that only the most low level hackers would love. And indeed, getting software to builds exactly the same each time is such a process. But once that process is in place, it can drastically simplify the process of delivering secure software. Projects like Tor and Guardian Project go to great lengths to build releases on machines that have never touched the network, are stripped down to the bare minimum, and are generally difficult to use. If the build process was deterministic, then a release build can be made on any machine, and then verified by various other machines also building that same release.

The FDroid central build server is already set up to be build verification service, so if it is easy for developers to integrate their releases with the FDroid tools, then the FDroid central build server will act as a automatic build verification service. That central build server is all free software, so anyone can set up their own build verification service.

- full stack deterministic build process
  - deterministic builds for relevant Debian packages
  - contributes to TAILS

- following gitian's model, make automated method for creating VMs for the build system
- standardize on Debian/jessie to ease deterministic build process

Hardening All The Way Down

The final piece of the puzzle of free, open source software providing open and auditable software is the process of verifying that given source code produces a given app, every time. This is generally known as "Reproducible Builds" or "Deterministic Builds". When anyone build an app themselves and then see it is exactly the same app as the developer has released, they know that all they only need to look at the source code to prove that the app is safe and secure. This is unfortunately a difficult and very manual process.

Luckily, it is a process that is very ripe for automation. For Android apps, the first step is to verify that the Java code generates the same APK as the official release. The problem is that everyone builds their Android apps using the software that builds and releases, so they are all using the same binaries which could all contain and implant the same code on every machine. So then the Android tools should be built in a reproducible way, and the system that the Android tools relies on should also be built in a reproducible way. Debian has started work on supporting reproducible builds, and indeed that is a feature that the TAILS and Tor Browser Bundle authors have requested.

This module of work is focused on providing reproducible builds all the way down the toolchain from apps to Debian itself. Debian already has the process prototyped, key pieces of the Android SDK are already included in Debian, and the FDroid tools already provide fdroid verify for comparing a locally built Android app to the developer's release. The remaining work is finishing the prototypes and making sure everything is integrated and working.

- https://summit.debconf.org/debconf14/meeting/78/reproducible-builds-for-debian/

- Backend Developer:
- Release Engineer:
- Tutorial Writer:

Peer-to-peer Malware Tools

Android tools like Lookout, BlueBox Security's scanner, and Google's APK scanner have shown that local scanners can catch lots of malware without requiring network access. For Android, the most serious exploits like Master Key and FakeID, can be detected by scanning the APK alone. There is also free demo code for doing this. This code can be incorporated into FDroid to catch most malware the first time it hits FDroid on a device.

There are also generic techniques for spotting malware based on comparing APKs that claim to be the same version (versionCode) of the same app (packageName). When two devices swap app information, each device can compare the APKs installed to the information received from the other device. If there are two APKs that have the same packageName and versionCode but have different hashes, then one of them is likely to be either corrupt or malware. FDroid can take this information and prompt the user with actions to take, like uploading to a malware scanning service, uninstalling, or temporarily disabling the app until it can be verified.

Polishing the Peer-to-peer Swap

With the first round of Bazaar, we added a wide range of app swapping capabilities to the FDroid app store. Users can create an app store on their device, then use Wifi, NFC, and/or Bluetooth to swap local app stores with whoever they encounter. The core functionality is mapped out and implemented already. The next step is to survey users and conduct user studies to figure out how the whole swapping experience can be make more understandable with fewer gotchas.

- UI Developer:
- UX Designer:
- Tutorial Writer:
- Backend Developer:

Trusted Outside App Channel for iOS where iTunes is blocked/filtered

- Tie into Cydia App Store/Hockeyapp/etc
- Apple iTunes is blocked in Iran, other places
- Apple iTunes is censored in China, including OTF's app
- p2p distribution support when possible

Cydia is a long-running, self sustained alternative app store for rooted iOS devices (an alternative app store is only possible iOS if the device is rooted, unlike Android). It has a very active developer community with about 1000 individual app repos. It does not include any of the common circumvention techniques that we recently added to FDroid like support for Tor and peer-to-peer app swapping.

While the software in each store would only work in the specific platforms, the core concepts are the same. Including work on Cydia would make sense in terms of cross-pollenation of ideas and shared project management resources.

All sorts of notes

There is a wide range of great software being developed that allow activists, journalists, etc. to use their mobile devices for private, secure communications and operations. And Google Play provides a secure and established way for distributing apps all over the world. But in many high risk areas like China or Iran, Google Play is entirely unavailable. Google Play is also built to track it's users, and that data is all stored by Google, which means that a number of governments have access to that data, via legal orders or by breaking into Google's systems.

People around the world have found all sorts of creative way of distributing apps and files when they cannot access Google Play, but these are fraught with issues: alternative app stores are a awash in malware, most ways of sending apps and files via the internet can be easily spoofed and monitored, and not many people know how to operate when the internet is completely unavailable.

- deterministic builds are an arcane technical process that makes it much easier to deliver secure software

- add support for making it the default app store in ROMs like Replicant, Omni, etc
- moving f-droid.org/repo from centralized trust to deterministic build verifying infrastructure
  - fdroid verify already passes many apps
  - fdroid.org/repo uses official dev's
- developer support for setting up their own (cydia is an example of what it could become: it has ~1000 developer repos)
- make FDroid have a modern, easy, attractive client UI
- fdroid is already sustainable, we will speed development, and contribute to the maintenance
- fdroid's main reason for the central repo is to have a repo of software that is verified to be free software, they don't want to have their own signing keys
- fdroid is a IaaS, any free software can be hosted on f-droid.org
- fdroid is also free software for building your own app store with build/verify service
- tie into user-generated data like openintegrity.org
- fdroid as app store kit
- efforts to tap into
  - making FDroid app store for Replicant, etc
  - fdroid verify
  - androidobs
  - openintegrity.org
- automatically trigger release builds based on signed tag
- replace hockeyapp, Play Beta, etc
  - include crash submissions
  - automatic nightly build repo
  - in-app bug reports
- include basic repo server that uses modern crypto instead of TLS-CA (PAKE or whatever)

- make your own app store kit
  - app stores for high risk threat models
  - all free aptoide
  - approach various existing app stores about using fdroid
- deterministic full stack
  - `fdroid verify`

- tools for people to curate repos
  - `fdroid remote`
  - `remotes = {}` in config.py

- why not update framework embedded in an app?
  - app store update model gives users control over updates
  - updates in APK gives users less control
  - high risk users will know best what will put them at risk and what will work
  - single experience for many different ways of transferring APKs

- tie into user generated content to verify APKs
  - crypto-link APK on local device to public data repo
  - combine user-generated data with data mining and observation
  - build upon existing research projects:
    - https://openintegrity.org/
    - https://androidobservatory.org/
    - https://guardianproject.info/code/weatherrepo/

- integrated translation management
  - FDroid/Google Play description and changelog in metadata
  - in standard, easily translatable format
  - if possible, automatic updating to Google Play

- use cases
  - humanitarian teams, when infrastructure is depleted due to disasters

- threats

  As our experience with helping Tibetan activists has taught us, and the leaks from FinFisher and other sources have shown, it is usually quite easy for an organization with network level or country-wide network access to exploit browser bugs to install software on target computers. A developer of circumvention software is a high-value target for a state-level actor since one exploit can infect that developer's software, which will then be distributed with an included backdoor implanted by the initial exploit of the developer. Best practice is to use dedicated machines for building and signing apps, but that is not practical for small scale developers, and can take a lot of time to setup. When using a deterministic build process, the software can be built on an exploited machine, then verified by trusted sources. The Fdroid tools are available for people to set up their own verification server, and the f-droid.org repository is also available to provide this service as a third party. This means that people can get automatic verification without having to setup their own infrastructure.
  - Pwn-To-Own competition is always won by browser exploits

- use hardware security modules as part of workflow
  - Linux kernel commits require one

http://www.linux.com/news/featured-blogs/203-konstantin-ryabitsev/784544-linux-kernel-git-repositories-add-2-factor-authentication

- why focus on android?
  - android is the biggest platform

    - android is the cheapest smart phone platform
    - android allows for anybody to run an app store
    - almost all app stores only address a single platform (iTunes, Google Play, etc)
    - Cydia is a well developed alternative app store for iOS
    - the technical problems line up cleanly along platform lines

- concept note was light on use cases, add lots
- research use cases of developers in challenged environments
    - want needs do they have that are different than developers in places with internet freedom
    - add in note about Iran-focused dev outreach project
- discuss how focusing on GP/Psiphon/Amnesty developer use cases will directly benefit users in target locations
- send email with possible extensions
    - full paranoid build stack
    - media handling in app store (parallel to Courier/Paik effort)
    - Blackberry or Windows Mobile running Android apps interesting at all?
    - ties to Cydia?

Bazaar Phase 2 OTF Proposal

- https://www.opentechfund.org/submit/guide

Proposal Summary

(*400 words recommended, 1000 word max*)

A great number of mobile apps have been developed to assist users in high-risk scenarios, but little has been done to address the issues facing distribution of the apps themselves. Google Play is blocked in many countries, and app stores like iTunes often censor to comply with regional law, whether just or not. Regional app stores are often cesspools of malware. In many countries, people exchange apps through web forums, email, bluetooth, SD Cards, or any other method they can figure out, whether safe or not. Effective techniques for circumventing censorship and internet outages exist, and work in many places, but none work in all, and most organizations are not able to keep track of them all. This current state requires users, trainers, developers, and organizations to be fluent in and aware of many technical details in order to effectively distribute mobile apps and media.

In Cuba, only 5% of the population have internet access, but many have smartphones and computers and share files using DIY networks and thumb drives. In Vietnam, swapping apps and media with Bluetooth is widespread. In Burundi, people get apps via SD cards using "APK installer" apps. Each of these workarounds can also be useful in many other parts of the world. In China, the internet is ubiquitous but heavily filtered and monitored; but "collateral freedom" techniques have proven effective. Beyond the Guardian Project's use of these methods (https://guardianproject.info/fdroid), this matches real world needs expressed by our colleagues. Psiphon already contributed support for Amazon S3 to automate their own process, Benetech needs highly targeted app collections to deploy their Martus system, and these distribution channels must be secret and secure. StoryMaker is pushing for distribution of their app, along with supporting third-party apps, to highly censored contexts. They all need a well-defined and audited process for ensuring that apps and media safely reach users and are kept up-to-date, regardless of the pitfalls and roadblocks along the way.

The past five years have produced big developments in security and privacy on the internet. Tor is ever expanding while getting quite easy to use; HTTPS is the widespread default for sites that manage private information; and apps like ChatSecure, CryptoCat, and TextSecure make secure messaging easy. Even the big players like WhatsApp and WeChat are also improving. The interest in tracking journalists and targeting activists has only grown, and the tools for mass surveillance are getting only cheaper. As the old channels of surveillance get shutdown, new paths for feeding that massive desire is the app developers themselves. More and more software developers are being targeted. If a backdoor can be placed in an app, or even more effective, in a common developer tool, then the attacker who controls that backdoor will gain access to data from the masses despite recent progress.

Basically all of these distribution methods are ripe candidates for the kind of automation that software does so well. We do not have to to convince developers to pay detailed attention to security first, we can entice them to improve their security by providing secure tools that reduce their workload. Google Play and iTunes demonstrate that "app stores" work well for distributing media as well as apps. We have a collection of working prototypes for a wide variety of techniques from the first phase of the Bazaar project, FDroid and Guardian Project have been honing those in the meantime. The next step is encapsulating all of them into a single system that provides smooth interactions for developers, organizations and end users.

- The developer will be in control of a simple set of commands that automate the entire distribution workflow for making highly secure, reproducible builds then getting them out through all possible channels.
- Organizations and trainers can use these tools to make curated collections of apps and media use, without getting caught up in the technical details of the whole process.
- The end user gets a familiar app store experience, regardless of the complexity behind their successful connectivity.

This proposed second phase of the Bazaar project, aka Bazaar2, will implement the entire system and user experience for Android, the most popular computing platform in the world. FDroid is the perfect community to build upon to spread into repressive environments because it is made up of activists and hackers who had privacy as a goal from the beginning. Projects and organizations focused on internet freedom can then pool their resources for managing circumvention techniques in this common platform. Additionally, since this model has already proven effective, we will also prototype extending this system to other major mobile platforms like iOS as well as to the desktop.

Proposal Narrative

(*800 words recommended, 3000 word max*)

In the past few years, there has been a lot of attention to improving the base level of security on the internet, and glaring issues that allow mass surveillance are being fixed. Even Chinese companies like WeChat utilize encrypted network connections and local storage. This means that repressive governments are looking for new channels to exploit. The standard mobile developer work process is a ripe target, but we can nip this in the bud for internet freedom developers by building upon the work of groups like Tor Project, FDroid, Debian, and more.

Targeted attacks are getting easier through the use of software like Finfisher. Key internet freedom tools are proven to provide privacy and security, and more developers and organizations are developing tools for privacy and circumvention. Users of these internet freedom tools are targeted: Occupy Central in Hong Kong was recently targeted) via download links in WhatsApp that delivered tailored malware. It is only a matter of time before there are regular attacks directed at the people creating the tools. Standard software development practices simply cannot survive targeted attacks of today's scope and scale, and the internet freedom community must improve distribution methods to stay ahead of this curve. From the software user's point of view, internet freedom tools and media need to be as easy as possible to distribute, otherwise people will find easier ways, regardless of the risks. We know how to build privacy and security into apps, and there are many good circumvention techniques in use already. It is time for the next big leap in internet freedom tools: a complete distribution ecosystem that provides secure, streamlined tools for developers and organizations, while providing an easy "app store" experience with built-in circumvention.

our users and developers need to stay safe in shark-infested waters

Our focus is mainly on Android, since it is the largest smartphone platform, and in many countries it is more popular than any other computing platform, including Windows. This project will therefore create a complete experience on Android. Since building a thriving media ecosystem requires spreading beyond a single platform, we will also address iOS, desktop and others.

Effective techniques for circumvention of internet filtering and monitoring exist. They include accessing services via proxies, mirroring content on cloud services, and direct peer-to-peer connections (like direct Bluetooth connections and local WiFi links). Device-to-device transfers are especially valuable when internet access is prohibitively expensive (Burma, North Korea, Cuba), or unreliable or disabled by states (Syria, Iran, Zimbabwe, etc). By combining a centralized app store with federated app stores as well as direct peer-to-peer distribution, the Bazaar2 experience will provide an easy to use central system with the flexibility of all three methods. Google Play also provides a secure way to get apps and media on Android, but is not available in many parts of the world (Iran, North Korea, Cuba) and is frequently blocked in many others (China, Syria, Vietnam, Ethiopia, etc). The Bazaar2 system will work in conjunction with Google Play for app distribution, when available.

This idea is already implemented in the FDroid app store for Android. The central f-droid.org app repository allows FDroid to deliver over 1300 apps without any configuration by the user. The `fdroidserver` developer tools allow anyone to set up their own repository of apps, so users can easily add that repository to FDroid. This also provides a channel for users to get apps via “collateral freedom” techniques, using Amazon S3, Akamai, etc. to distribute files where major hosting services like those are unlikely to be blocked. The FDroid app itself can act as an app repository; devices can connect to each other using local WiFi, mesh, Bluetooth, and removable media. The remaining challenge is combining them all into a usable experience. This has been tested, discussed, sketched out, and our first implementation of an integrated user experience is available for testing.

centralized.png"

federated.png

centralized and federated systems

peer-to-peer.png

mixed system of centralized, federated, and peer-to-peer

Improving developers' workflow and security

Developers rarely have time to implement strong security, and are a very ripe target for targeted attacks. For example, managing signing keys on a fully offline system is the best practice, but few do this. Even worse, Google provides tools) that encourage developers to keep their signing key on their regular development machine. Using common browser exploits, it would be easy to get the signing key and password in that setup. Hardware security modules (HSMs boost key security and are cheap, but difficult to use. Both offline key management and HSMs are problems that are well automated. The combination of reproducible builds and HSMs means developers can achieve high security processes using standard practices for setting up and running their computers. Specialized setups like offline builds and signing will then only be required for the most extreme risks. Once setup, this process will simple to use, there will be only three commands for regular use:

- `fdroid publish` makes new releases of apps
- `fdroid update` updates the app repository
- `fdroid server update` pushes everything to every configured app store

Developers focus on making sure that their apps are private and secure, but the tools for improving the process, like bug/crash reporters, and beta

channels etc. are usually not designed with strong privacy in mind. For example, if an Orbot user posts a standard crash report, information contained in that report can deanonymize the user as well as their Tor traffic. In order to deliver secure software, the entire toolchain must be designed with privacy in mind. Some core parts have already been developed by free software projects, like Redmine for issue tracking and ACRA for crash reporting. There are far too many options, so we will integrate these tools and techniques into a single tool suite. To get buy-in from developers, we will provide a smooth, simple workflow that serves as management tool for software distribution.

There must also be a seamless user experience for organizations that distribute software and media, and one tailored for it's non-technical users. Bazaar2 will provide tools that save time and effort to trainers and trainees who rarely have extra time to figure out new software.

The last key piece is translation. No matter how good software is, if it is not in a language the target audience understands, it will be useless to them. We will build integrated translation management, drawing from our own experience of managing translations in many languages for the software, related text, descriptions, and tutorials.

Weaknesses and Challenges

Systems built upon free association can be poisoned by bad actors when the community is still small. This concern is even greater if the system might be targeted by large state actors, like China, who have the human and technical resources in place to work against the construction of a flourishing ecosystem. Two aspects of this proposal are vulnerable: anyone can start an app store, and apps can be freely swapped between devices without being verified on centralized resources. We have included a couple of approaches to prevent this. First, we will rely on well known methods of building up social software and hire people to work specifically on this issue. Second, we will tap into our existing channels to app developers around the world to get them involved as early as possible.

This project is well positioned since there is already a dedicated, flourishing FDroid community with an excellent security track record, and over 50 million downloads. The FDroid community includes many highly technical people who have a strong interest in privacy and security. Also, there are many active and passionate users who care about spreading Free Software. While our mission is to spread internet freedom, there is so much overlap that FDroid's activist point of view encourages the existing community to support internet freedom activism around the world

FDroid is already the premier hacker app store, popular on Hacker News, referenced by CyanogenMod as a supported option, even long time tech journalist Dan Gillmor uses FDroid. More technical people involved means more people looking out for funny business.

To build out the Bazaar Internet Freedom community on top of FDroid, we will work directly with trusted partners and organizations to lay the core building blocks of the ecosystem. Getting the initial community members in place will be labor intensive, and will require a dedicated community manager as well as developer resources to directly address the needs of the initial organizations. Also, FDroid community is already a willing testbed for circumvention features, and it has not yet gathered attention from known adversaries.

This then highlights the biggest social challenge: getting buy-in from users and developers. App store users often stick with what they know, and insecure app stores are often familiar and perhaps easier to use. In Iran, there were basically no good alternatives to Google Play, until Cafe Bazaar. Cafe Bazaar is now rapidly gaining market share because they focused on providing an easy experience for getting apps, including pirated software from outside of Iran. They have also tied into national pride by highlighting apps from Persian developers. China has many alternative app stores that are based on vast collections of pirated software but they are often corrupt and mislabeled. Xiaomi and Baidu are focusing lots of effort in developing their own app stores, and are both required to collaborate with the Chinese government. Their market share and resources could be very difficult to compete with on a grand scale.

Promotion and outreach must be multi-pronged in order for a project to take off. We have been talking with commercial ROM makers and device manufacturers about including FDroid as a default app store. We are working on getting contributions from developers around the world, and that can also serve as fodder for national pride. Google Play automation is now possible, and there are no notable projects providing solid features there, so FDroid will soon be the first tool to automate the whole release process. The more developers that ship popular apps on fdroid, the more users will use it. For example, AdAway was popular in Google Play before Google banned it. The developer said to get it from FDroid, then FDroid saw 2 million downloads in a short period of time. Another successful promotion technique is providing access to banned content, like Free Weibo does. We plan to market FDroid as a store for blocked apps.

Developers rarely have time to learn new tools, so the developer experience must provide compelling features that make the developer more efficient. If there is a clear benefit to using a new tool, developers will invest the time to switch to it. Therefore, this proposal also addresses tasks that Android developers are already doing, like providing alpha and beta channels, managing releases in Google Play, working with translators, and uploading apps to VirusTotal.

The biggest implementation challenge will be putting together an understandable user experience that guides non-technical users through an array of options for getting apps. We will provide a functional distribution experience because we are building upon existing circumvention techniques known to work in the real world and gathering them together into a common user experience. We'll be taking the best practices from the field, smoothing them out and automating them, not creating new techniques.

The promise of Bazaar2 makes taking this on worthwhile. Once the ecosystem is seeded and there are many users who can swap apps, it's nearly impossible to block the flow of these apps and media. When internet-based central services are available, they will be utilized transparently, but it is only needed here for initial seeding and then to check that malware has not infiltrated. Users can always fall back to entirely local, peer-to-peer methods of data transmission (bluetooth, local wifi, etc), or use local, decentralized networks. This changes the dynamic of app distribution: users can still have open distribution channels even during a full internet outage.

Technical Approach

The technical approach will combine new development efforts with integration work. Free software projects, including FDroid, ACRA, and gitian, can solve large pieces of the problems presented in this proposal. The largest chunks of development work relate to building unified and intuitive user experiences for the app store, publishing mechanisms, and developer tools. The core of this work, focused on Android, will happen as part of the FDroid project. FDroid is the best developed free software app store for Android, so our improvements will be included in the community-maintained software. The app is already translated into many languages, including Arabic, Chinese, Farsi, Uigher, and Spanish.

There is an alternative to the app store model for updating software: the "self-update" model. The app store model is generally the approach taken by package management systems that are integrated into the core OS, like the Apple App Store, Google Play, etc. The "self-update" model means the apps update themselves, on the desktop like Firefox, Chrome etc. The self-update model gives more control to the app's developer, while the app store model gives more control to the user, letting the user decide when and what to update. Putting the user in charge allows them to adjust the behavior and timing of updates based on their risks and situation while still providing timely and transparent updating.

The current f-droid.org app store has a major weakness: all apps are signed by a specific f-droid.org key rather than by the developer's official release key. This creates confusion because two valid builds of an app could be signed by conflicting keys. This is not by design, but rather a side effect of their core goal: verifying that the apps are built entirely from freely available source code. The Bazaar2 project will turn this weakness into a major strength: the f-droid.org build infrastructure will build APKs from source as before, but it will then verify that the developer's APK is exactly the same as the APK built by f-droid.org. Since a reproducible build will match exactly, the developer's original signature can be copied into the APK built by f-droid.org. f-droid.org becomes a build verification service, reproducing builds made by the original developer while eliminating the duplicate f-droid.org key.

Malware detection is an essential part of any software distribution ecosystem, more so on Android devices since the core software isn't often updated by the manufacturer. Google provides its malware scanning services via Google Play, so places without access to it lack that malware service. A number of the worst Android exploits, like "Master Key", are relatively straightforward to detect with free and available working code. Bazaar2 will include work to integrate this code into the FDroid app store and developer tools providing baseline malware protection everywhere in the world. This protection will work in conjunction with others, like malware scanner apps from Lookout or BlueBox.

A free software tool, ACRA is shaping up to be the preferred way to include crash reporting in apps. ACRA will be integrated into the Bazaar app store and developer tools to provide full featured crash reporting in a privacy-preserving way. FDroid already provides beta channel functionality, so it just needs to be fine-tuned. Since Google Play provides an integrated experience with solid security, it is reasonable to assume that other app stores can achieve this. Alternative app stores exist in many countries,but with terrible security and frequent filtering and/or monitoring. They lack basic security practices: aptoide.com does not have HTTPS by default, and CafeBazaar.ir does not have functional HTTPS available, even to developers uploading their apps.

User Stories

A developer in Iran has created some apps and wants to distribute them to as wide an audience as possible. He knows that some apps have been blocked from Cafe Bazaar, so he wants to make sure that there are as ways as possible for people to get this apps. He sets up the FDroid tools to manage publishing his apps to Cafe Bazaar, Google Play, f-droid.org, and a couple "collateral freedom" services like github and Amazon S3. He then runs two simple commands to update his app repository and publish it to all of the app stores. He makes his releases using the FDroid reproducible build and harded signing process. Even though other local developers have found Finfisher on their computers, he feels confident that his release process has not been infiltrated.

StoryMaker is making a targeted campaign that delivers everything that a user needs to make a video, including tutorials and guidelines for that specific campaign, and a channel to publish videos to others. They need everything delivered to users' devices with a single download and single install process. It must also automatically stay updated. They use a custom FDroid installer bundle that includes everything needed in a single download link. This same link will also direct users who already have StoryMaker to the campaign without making them download everything again. For new users, FDroid is first installed, and it then downloads a standard StoryMaker release with a trigger for it to get the campaign materials. Since FDroid is now installed, it will automatically get StoryMaker updates while also providing a full app store to users. Users publish their videos from StoryMaker, which also adds them to this campaign's FDroid media channel, so they can easily get the videos that others are making.

A human rights organization produces videos and ebooks of training materials and important information, they also have their own mobile app. In their trainings, they use apps from Tor and Guardian Project. A trainer sets up an app store that includes their app and all their videos and publications, and set it up to automatically include the most recent updates from Tor and Guardian Project. At trainings, students get an FDroid bundle from the local app store on the trainer's phone. The installation process lets them click through to install Orbot and ChatSecure, and FDroid is set up with direct access to download and share all of the videos and ebooks.

Project Objectives

(*500 words recommended, 2000 word max*)

Simple developer experience for distributing apps in a multi-pronged approach

Getting developers to put their apps into this eco-system is a central part of the mission. Therefore, the developer experience must be easier to use than existing ways to make it worth the work of switching to a new workflow. It must also provide unique benefits that are difficult to achieve. Automating the whole build and release workflow will save developers time. Providing a deterministic, verifiable build process will give developers the unique benefit of removing the requirement for hardened build machine for making secure builds.

- integrated, highly secure build and distribution process
- single tool to run whole release build process
- reproducible, deterministic build process
- simplified process for anyone to verify an app against its source code
- same tool also uploads to Google Play, Amazon S3, etc.
- toolset for developers/projects to pool circumvention techniques
- tools to simplify offline signing key management

Organizations and trainers securely curate and distribute collections

Organzations like Amnesty and trainers like Tactical Tech need to distribute media and apps from multiple sources to users who face a wide array of challenges to getting unfettered access.

- set up their own repositories of apps and media
- curation tools allow mixing in from other collections
- orgs that need Orbot, etc. can include the Guardian Project repo in theirs
- trainers can easily setup custom repos for any given training
- simple, user-friendly tools for managing local apps and media collections

Modern app store experience with a full suite of circumvention techniques

The key to this whole system working is presenting it all to the end user in a familiar and intuitive interface. There are many design patterns that large app stores follow, so they are well tested and familar to many users. Most of this system fits neatly into a standard app store experience, so the app store itself can handle figuring out which network circumvention technique to use. Secure device-to-device sharing requires the user to manually opt-in, so the "swap" interaction is then added in to the standard app store experience at the points where they are intuitive.

- transparent support for multiple distribution channels: centralized, decentralized, peer-to-peer ** enable distribution despite internet outages and blockages ** easy enough that everybody wants to use it all the time, not just during outages
- add media distribution to the existing distribution eco-system
- get people off of malware-ridden unofficial app stores
- make it easier than emailing, direct downloads, etc.
- implement complete experience for Android
- implement web interface to collections of apps and media
- research bringing this distribution experience to iOS and desktop
- include a wide range of apps and media by default, including privacy and circumvention tools
- not just an activist tool, a widespread, general purpose reputation
- scans for common exploits (e.g. "Master Key")
- give users clear actions when suspect apps are found (e.g. uninstall or upload to malware scanner)

partner deployments in Iran, Tibet, Cuba

It is not possible to develop effective tools without having regular and frequent communications with people who are directly affected by the issues that we aim to address. Therefore, we work in partnership with organizations on the ground. This means that they can bring their direct experience to us, and we can have rapid and regular feedback on the software we develop. For this project, we have three partners covering three different sets of challenges.

user research on in-country developers in Iran and Cuba

There has been very little user research done on developers who are working countries where the internet is heavily monitored and filtered, so the issues that they face are not well known outside of the circles of these developers. A core part of this project is partnering with developers who operate in Iran and Cuba. After we have worked with them to address real distribution issues, we will publish user stories and information based on our experience. Also, we will seek feedback about more general problems that they face. The goal is to produce user stories and information that can be shared with other organizations working on internet freedom.

Project activities

(*500 words recommended, 2000 word max*)

Below are the major groups of activities that we are seeking funding for. These represent a range of work from supporting the first phase of the Bazaar project to tackling a large new area with an even larger potential impact. This work must be multi-faceted in order to most effectively develop and deliver real solutions that address core problems that affect many people and organizations work on internet freedom.

The majority of our work will be software development, testing and deployment. The bulk of the work will be done by Guardian Project and FDroid core team members. To encourage community contributions, we will also work in the open and make time to be available to answer any questions. All of the software and tutorial material that we produce is Free Software, and freely usable and modifiable by anyone. We also aim to freely publish all of the written materials that we produce, as long as there are not privacy or security concerns. We push our development discussions to public email lists and IRC chat rooms. All source code, including tutorials, will be published on public, open git hosting services like Guardian Project's github and FDriod's gitlab.

Other work includes user studies, specific development activities, and interfacing with developers, users, trainers, and organizations around the work. We will tap our own network of highly qualified consultants to help us deliver solid and transformative tools. With a project of this size, we also need to broaden our horizons, so we will also seek out new voices to help us shape these core activities.

simple developer experience
- integrate tools for verifying reproducible builds (fdroid verify)
- develop build image based on Debian/jessie as a VirtualBox VM that runs on all platforms

regular software releases
- release early, release often
- push beta releases to community of testers

ongoing developer support
- one live 3 hour workshop with online participants, recorded for posting on the web
- five online "office hours" sessions open for all to ask topical questions
- support via email, voice/video/text chat, IRC, etc.

generate online walkthrough tutorials

- write click-through illustrated tutorials
- translate into Spanish, Farsi, Chinese, and Tibetan

support for organizations and trainers plugging into the Bazaar distribution system

- user walkthrough tutorials on app swapping

- study existing app "swap" user experience and integrate with app store user experience info

Budget details

(*500 words recommended, 2000 word max*)

In rough order of portion of total budget:

- Backend Developer
- product and community manager
- UI Developer
- tutorials and outreach
- UX Designer
- F-Droid support contract

Sustainability

(*300 words recommended, 1000 word max*)

In terms of the core software, the FDroid app and tools, it has been developed and maintained by the FDroid community for a couple of years before we based our internet freedom work on it. The community is represented by a small non-for-profit organization that receives direct donations to cover the maintenance of its infrastructure. The FDroid team makes all of the releases and runs the infrastructure, those are not dependent on Guardian Project or its funding. We actively discuss this proposal with the core FDroid team and they are completely support our efforts so far, so our contributions will maintained as part of the whole package.

Additionally, the core mission of this proposal is to get developers to use the FDroid tools as their primary method of making releases of their software. FDroid already receives contributions from many developers, and the more developers that we can get setup using these tools means more contributions from those developers who have vested interest in maintaining the tools they use in their own processes.

- small partnership consulting projects to aid larger orgs to get setup in the FDroid eco-system

Other support information

(*300 words recommended, 1000 word max*)

- Guardian Project has just kicked off a project to build a developer support network for people working in Iran and the Iranian diaspora. The goal of that project is outreach, education, and community building. This proposal ties directly into that effort by adding in a user study of that developer community and then using that knowledge to improve the FDroid developer tools to address the issues that affect the Iranian developers.

Organization and/or individual background

(*400 words recommended, 1000 word max*)

References

(*200 words recommended, 1000 word max*)

Similar/Complementary efforts

(*400 words recommended, 1000 word max*)

There are a couple of software companies like Apperian and Aptoide that provide a service for creating and running custom app stores. The ones that we have looked did not have solid security. Most are also not free software, so we can not customize them ourselves. Apperian does have the advantage of supporting Android, Blackberry, iOS, and Windows Mobile. Aptoide is also a company, but it's client app is a free software (the FDroid client app started out as a fork of Aptoide's). At this point, Aptoide does not provide even close to the same level of security as FDroid does, and the FDroid developers were specifically interested in user privacy as a goal, while Aptoide is not. Aptoide has recently greatly improved its app store user experience, giving us the possibility to learn from their experience while we are designing ours.

The Tor Project's work on reproducible builds has been very informative to our research on reproducible builds for Android. Also, Debian has a very

active effort to provide reproducible builds for all packages. We are already planning how to best collaborate on this work with The Tor Project and Debian. Guardian Project core team member Hans-Christoph Steiner is an official member of the Debian project as a "Debian Developer", and has been participating in the Debian effort.

For the specific issue of secure software updates, The Update Framework (\url{http://theupdateframework.com/)} addresses the need for a secure updating system. It is a research project that has already produced a number of interesting ideas, but it is not directly applicable because: a) it is written in Python, which is difficult or impossible to run on mobile; and b) it is intended to embedded into each app to update itself, which major mobile platforms do not allow.

Cafe Bazaar (\url{http://cafebazaar.ir/?l=en)} is an Iranian company that provides an Android app store focused on the Iranian market. It highlights apps from Persian contributors, but also includes pirated versions of many popular apps and games from outside Iran, like Google Chrome, Twitter, and as of September 24th, 2014, a real version of our ChatSecure app. Apparently, they receive funding from the Iranian government intended to support Iranian businesses that compete against foreign companies, so it seems safe to assume that they must be working with the government.

Community Interaction

(*400 words recommended, 1000 word max*)

Project Evaluation

(*400 words recommended, 1000 word max*)

Evaluation and assessment is built into all Guardian Projects and project partnerships. We use both formal and informal methods as well as qualitative and quantitative measures, however exactly how depends on the particular situation. We will monitor data collected through the project to ensure we are on track to meeting proposed targets and to ensure we adapt to changing circumstances or unforeseen developments as the project progresses. A comprehensive list of indicators will be tracked throughout the project. It is extremely important to self-evaluate and assess, but we must always consider the safety of our users and target audience. Metric information can often be deanonymized and can put users at risk, running counter to our mission. We will assemble overarching reports at the midpoint and end of the project. Our evaluation methods will include:

- tracking builds and app downloads
- tracking users and usage information from sources that require tracking
- tracking users and usage anonymously from trusted partners and organizations
- measuring how many languages the apps have been translated
- tallying new code contributions
- hands-on training including feedback from users
- interviews with in-country developers
- written and oral surveys

While it is fully possible to implement systems that track users and provide detailed usage information, we avoid tracking our users since that data can be mistakenly leaked, or deliberately stolen. We do use data from distribution channels that require user tracking, like Google Play and Amazon S3, so that can provide a rule of thumb based on situational knowledge (for example, Google Play is blocked in Iran and China). We will set certain development targets and quantitatively measure how much we achieved, for example, how many languages an app has been translated to. For these kinds of things, we will track them and report them quantitatively. For more detailed stories about the progress, qualitative metrics must be the focus in order to avoid publishing too detailed information about specific users.

We do gather information on people and organizations that we have a relationship with, handling it with all due caution. From this, we can also generate some concrete metrics, albeit in a state largely disconnected from individual people. We aim to work in public as much as possible, including when evaluating our progress. Since we are dealing with a lot of high risk users and it is easy to mistakenly leak identity information when providing dataset to the public, we feel we do not have the resources available to publish all of these metrics publicly.

Running hands-on trainings is a very valuable form of informal, qualitative feedback since we can directly experience how people learn and use our software. For example, we ran a very successful "practocalypse" workshop in collaboration with OTI/Commotion and Eyebeam where the Bazaar prototype version of FDroid was used to distribute all the Android apps at that workshop. This experience provided the most valuable feedback confirming our overall design was effective while pointing out the usability issues we still needed to address.

Some key goals to measure:

- FDroid app store and website fully localized
- available 80% of test group rates the app store experience as good and helpful
- confirmed app store accessibility in China/Iran via "collateral freedom" and/or proxies like Tor
- confirmed app swapping ability without internet

Some quantitative metrics that we aim to meet over the project time span:

- new organizations running their own app/media repositories (Target: 3)
- apps building using the fdroid reproducible process (Target: 10)
- total downloads of apps built using the reproducible process (Target: 1,000)
- FDroid downloads (Target: 100,000)
- code contributions from new developers (Target: 3)
- in-country developers interviewed for user research (Target: 10)
- apps with all related content fully translated into Spanish, Farsi, and Tibetan (Target: 10)
- 90% of trainees in hands-on training are able to swap apps and/or media
- 95% of all text content able to be translated

And qualitative metrics:

- needs assessment from interviews of trainers, organizations, and developers
- cataloging the identified needs which have been met
- data collected from test groups
- test reports collected from test groups
- written surveys at midpoint and end of project
- results of security audits by OTF Red Team at start and end of project

Files

centralized.png	11.3 KB	01/14/2016	hans
fdroid.png	8.4 KB	01/14/2016	hans
federated.png	27.7 KB	01/14/2016	hans
finfisher-arabic-brochure.png	64 KB	01/14/2016	hans
peer-to-peer.png	31.1 KB	01/14/2016	hans
sm-martus.png	7.37 KB	01/14/2016	hans
sm-storymaker.png	16.3 KB	01/14/2016	hans
storymaker.png	49.4 KB	01/14/2016	hans

Concept Note

A great number of mobile apps have been developed to assist users in high-risk scenarios, but little has been done to address the issues facing distribution of the apps themselves. Google Play is blocked in many countries, and app stores like iTunes often censor to comply with regional law, whether just or not. Regional app stores are often cesspools of malware. In many countries, people exchange apps through web forums, email, bluetooth, SD Card, or any other method they can figure out, whether safe or not. Effective techniques for circumventing censorship exist, such as accessing services via proxies or mirroring content on cloud services. These techniques work in many places, none work in all, and most organizations are not able to take advantage of them all. This current state requires users, trainers, developers, and organizations to be aware of many pieces in order to effectively distribute mobile apps.

In the first round of Bazaar, we focused on empowering the user by creating decentralized and peer-to-peer techniques for sharing apps between devices. The next step is to focus on the app developers and organizations by providing tools to secure their entire app release process and to publish their apps into this decentralized system. The developer experience will be a simple set of commands that automate the entire distribution workflow: making highly secure, reproducible builds then getting them out through all channels possible. Organizations that make curated collections of apps will have tools to verify that those apps match the original source. Once configuration is complete, there are only three commands for regular use: `fdroid publish` makes new releases, `fdroid update` updates the app repository, and `fdroid server update` pushes everything to the public. The user gets a familiar app store experience, regardless of the complexity behind their successful connectivity.

Beyond the Guardian Project's use of these methods (https://guardianproject.info/fdroid), this matches real world needs expressed by our colleagues. Psiphon already contributed support for Amazon S3 to automate their own process. Benetech needs highly targeted app collections to deploy their Martus system, and these distribution channels must be secret and secure. StoryMaker is pushing for distribution of their app, along with supporting third-party apps, to a highly censored context. They all need a well-defined, audited process, toolkit and user experience for ensuring their users can safely access and update these apps, regardless of the pitfalls and roadblocks along the way.

The funding we seek is to focus our existing team on this end-to-end problem, with the work spanning further development of the fdroid toolchain, release workflow design and auditing, usability improvements, and training content development. The work will be done in partnership with F-Droid Limited, an English non-profit organization funded by direct donations, and we include support for their efforts through this funding. Over the past 18 months, we have built a solid working relationship with that team, and they have taken on these goals of security and decentralization as part of their mission.

https://twitter.com/guardianproject
For technical details: https://guardianproject.info/tag/bazaar

This is the concept for the second phase of Bazaar. For the first phase, see [[Core Concept]]

Improving the APK Signing Procedure

Using Hardware Security Modules for APK Signing

Ideally the OpenPGP card could be used for both jarsigner and OpenPGP signing. That means a card that can do both PKCS#11 and GnuPG. Otherwise there would have to be a PKCS#11 card for Java and an OpenPGP card for GnuPG. OpenSC now seems to provide PKCS#11 access to the OpenPGP card. Via OpenSC, keytool can see the key/certificate on the OpenPGP card via opensc 0.13, but jarsigner does not cooperate.

Relevant Discussions and Documentation

- PSA: Think About Stronger App Signing Keys
- |guardian-dev| APK signing keys are vulnerable
- Guide to GPG with a Smart Card
- |gnupg-users| using an OpenPGP card with Java
- Debian Smartcards Wiki
- YubiKey NEO and OpenPGP
- How to use a Yubikey NEO (or any OpenPGP card or GnuPG in general) to sign X.509 CSRs?
- Working with Aventra MyEID
- [[Notes on working with HSMs]]
- Java PKCS#11 Reference Guide
- Mozilla Blog: Using CryptoStick as an HSM

Related Software

- OpenSC
- ykneo-openpgp (openpgp app for the YubiKey NEO)
- [[psst:CleanRoom]]
- libengine-pkcs11-openssl

Hardware We're Looking At

- http://g10code.com/p-card.html
- https://www.crypto-stick.com/
- http://aventra.fi/English/products_MyEID_E.html
- https://www.yubico.com/products/yubikey-hardware/yubikey-neo/
- Yubikey PIV should allow both opensc and OpenPGP support:

Getting Info about the Signatures

- Display PKCS7 info
  - openssl pkcs7 -inform DER -in $SOMECERTFILE -print_certs -noout -text
- Extract a PEM encoded X509 cert from the PKCS7 structure
  - openssl pkcs7 -inform DER -in $SOMECERTFILE -print_certs -out ExtractedX509Cert.PEM
- Display the X509 fingerprint of the extracted cert
  - openssl x509 -inform DER -in ExtractedX509Cert.PEM -noout -fingerprint

Here is the https://androidobservatory.org import code for displaying cert info. It should run in isolation of the Observatory code, but it's largely untested & undocumented.

Generating a New Signing Key

The default way to generate APK signing keys is with Java's keytool. It is widely used and not too hard to use. We recommend using at least OpenJDK 7, since it includes much improved versions of keytool and jarsigner. If you are willing to get your hands dirty, it is probably better to generate your key with openssl then import it with keytool (see below).

- keytool -genkey -v -keystore test.keystore -alias testkey -keyalg RSA -keysize 4096 -sigalg SHA1withRSA -dname "cn=Test,ou=Test,c=CA" -validity 10000
  - do **not** specify passwords on the command line (i.e. do not use -keypass or -storepass)
  - -keysize 2048 is the minimum, but -keysize 4096 is better
  - -keysize 8192 is overkill and might not work on older Android versions
  - SHA256withRSA and other better hashes supported on **Android 4.3 and above only**! 1
  - SHA1withDSA*should* work, but we haven't tested it

On Debian/Ubuntu/Mint/etc., you can keep OpenJDK 6 as the default JRE and JDK for compatibility while using keytool and maybe jarsigner from OpenJDK 7. Use sudo update-alternatives --config FOO, where FOO is: java javac jar jarsigner keytool

Using openssl to Generate the Key

An alternate way to generate the key is to use openssl. When it comes to generating, openssl has a better security track record than Java. Also, it is a less common combination so that exploits that might work with Java/keytool/jarsigner might not work with a key generated with openssl then imported using keytool. The downside is that there might also be weaknesses exposed by this trick, but that seems less likely than Java/keytool having

problems. Additionally, we recommend generating your key using /dev/random because this is a long-lived key and therefore more sensitive. This does make generating the key take a lot longer.

```
openssl genrsa -out secretkey.pem -aes128 -rand /dev/random 4096
openssl req -new -key secretkey.pem -out request.pem
openssl x509 -req -days 9999 -in request.pem -signkey secretkey.pem -out certificate.pem
openssl pkcs12 -export -out certificate.p12 -in certificate.pem -inkey secretkey.pem

keytool -importkeystore \
   -srckeystore certificate.p12 -srcstoretype PKCS12 \
   -destkeystore certificate.jkr -deststoretype JKS
```

This should be possible with using GnuPG's gpgsm instead of openssl, but that requires GnuPG 2.1, which is not easily installable, and still in beta. So we'll save that HOWTO for later. You can follow the progress of all this work in our git repo: https://github.com/guardianproject/smartcard-apk-signing

Files

cert_info.py	8.67 KB	02/07/2014	hans

Notes on working with HSMs

Aventra MyEID

- use options files with pkcs15-init! The command line is weird and flaky
- ACS ACR38T card reader requires an extra lib to work with pcscd: apt-get install libacr38u
- requires Debian package opensc 0.13.0-4 or newer to work, or opensc from master in git
- full details: https://guardianproject.info/2014/03/28/security-in-a-thumb-drive-the-promise-and-pain-of-hardware-security-modules-take-one/

crypto-stick

- OpenPGP Card like GnuPG
- requires libccid 1.4.16 to be fully recognized
- RSA 4096 max

Yubikey Neo PIV

- PIV is non-upgradable beta
- has both OpenPGP and PIV applets on the card
- has OTP and password features
- very nice USB form factor
- RSA 2048 max

Feitian ePass2003

- seems to have been developed with free software projects in mind
- proprietary firmware
- has specific support in the OpenSC Ubuntu packages
- nice USB form factor
- cheap! 5 for US$70
- RSA 2048 max
- http://www.uselessbrain.org/wiki/index.php/Token.ePass_2003

ACS ACOS5-64

- ACOS5-64 Cryptographic Smart Card
- CryptoMate64 USB Cryptographic Token
- cheap card that supports RSA 4096
- partially complete OpenSC support, stalled 3 years ago
- available in USB thumb form, called CryptoMate64

maybe useful info here:

- https://github.com/hiviah/gnupg-pkcs11-scd
- http://majic.rs/book/export/html/81

Local Data Transfer

This is a survey of p2p data transfer methods that exist on mobile devices, and can be setup by an app.

The mechanisms have been divided into two groups, to avoid confusion. The first group is protocols which are suitable for transferring files. The second is means by which to initiate connections, using any of the protocols from the first section.

Protocols/Technologies for transferring files

Bluetooth

Wifi

Locating the website could be handled in a few ways:

- QR URL
- bonjour .local domain names
- captive portal splash page
- manually typing simplified IP, i.e. http://10.3.2.1:8080

Using existing wifi

- check if device is already associated with a wifi AP
- use that IP, and include name of wifi AP for other users

Setting Up An Access Point on the Device

- Paik style

Desktop Wifi Sharing

- make a desktop app that can download, verify APKs
  - HTTPS pin
  - GPG keys included
- it then makes a simple webserver to share them
- generates QR code of URL for scanning

Adhoc Wifi

Mesh

Wifi Direct

- Hans tried this and failed to get anything working, so this is punted as too difficult to be useful

NFC data transfer

Technically it is possible to transfer data via NFC, but it is not recommended. I presume it is due to the low bandwidth, but perhaps it is also due to lack of error correction or other requirements that make file transfer suitable. Most NFC software implementations (e.g. AndroidBeam, S-Beam) use NFC to bootsrap a more suitable connection such as bluetooth or wifi (see below).

Methods of initiating/bootstrapping p2p connections

NFC and QRCode's are both means to the same end - encode a small amount of information, transfer it between two devices in the easiest way possible, and use that information to initiate a connection between two devices. Whether the resulting connection is via Bluetooth, WiFi, WiFi-Direct, should be somewhat inconsequential to the users. In fact, most of us had a bit of trouble wrapping our head around the actual mechanism Android Beam uses, because it does such a good job of abstracting the underlying protocol. This is in stark contrast to, e.g. Bluetooth, where users are made accutely aware that they are initiating a two way communication channel with another device, which then sends data to them using the Bluetooth protocol.

NFC

- Android Beam (Uses NFC to initiate a Bluetooth connection)
- S-Beam (Uses NFC to initiate a WiFi-Direct connection)

QRCode

QRCodes are usually used to encode a URL. Although the URL doesn't have to be a http:// URL (it could equally be an fdroid:// or some other custom scheme), it is probably good practice to do so. The reason is that for an fdroid:// URL to work, the device scanning the barcode requires F-Droid to be installed.

However, if you use a URL such as http://10.0.0.10:8888/fdroid/repo, then if F-Droid is installed, it will be able to handle that URL, and if not, the user

will be able to be shown a webpage served from that location, showing further information on how to connect.

Simulating p2p connection over HTTP

When using WiFi to transfer data between devices, we opt for HTTP, because the client already knows how to understand this, and it is a ubiquitos standard. One downside of HTTP is that it is a stateless, client-server protocol. That is, once you ask to download a file, the server generally doesn't then have the capability to request information back from the client (Yes, websockets, long-polling, etc exist to achieve this goal, but I'm referring to the general usage of HTTP).
One suggestion we had during a Bazaar scrum was the following:

- Device A: starts a swap
- Device A: shows a QRCode with it's IP address (along with a queyr parameter that indicates it is willing to accept repositories from clients - "http://10.0.0.10:8888?connect_back"
- Device B: scans the QRCode
- Device B: uses info from code to setup repository of Device A
- Device B: also POSTs to the URL of Device A telling it Device B's willingness to swap: "http://10.0.0.10:8888" with the POST params [swap_available : "http://10.0.1.20:7777"].
- Device A: uses info from POST request to setup repository of Device B

March 26th IRC Scrum log

@_hc pd0x: n8fr8 ready when you are
pd0x _hc n8fr8: here as well
@n8fr8 okey doke
@_hc anyone want to start? or shall we do jitsi?
pd0x I can start
pd0x I mentioned the app audit ticket is coming along. I have it almost all pieced together in fdroid client to be an activity that can be offered after a repo sync or on demand
* n8fr8 puts up a sign saying "the bazaar scrum is happening now. learn more here: https://dev.guardianproject.info/projects/bazaar/wiki"
pd0x I have it working for signature & hash right now. Not doing version code comparisons or looking at hashes of resources inside the APK (dex, resources, strings, AM)
@_hc :)
pd0x I'm doing some heavy work in the CursorAdpater that I need to move out to something cleaner/off the UI thread
pd0x on the observatory side I have the start of an API for doing REST and getting back JSON
pd0x which should let me hook into the observatory history for apks on the audit screen
pd0x I've been thinking more about the META-INF/ settings file stuff. I think it would be reasonably easy to have Kerplapp insert a properties file into the META-INF dir of a copy of the FDroid APK it makes to boostrap clients
pd0x then we can write a patch for FDroid that on app init looks for that file in its own APK
pd0x _hc: do you know much about the FDroid preferences as it exists now?
@_hc enough
@_hc as in Settings?
pd0x I don't think it's worth going through the trouble of deleting the inserted settings file since we know it won't fix the FDroid hash
@_hc its pretty plain Android settings
pd0x the Kerplapp repo can tell the to-be-bootstrapped client what the hash should be (with the settings bundle inserted) anyway
pd0x _hc: since the repo details are in the DB that'll require some thinking...
pd0x We could put a file with repo details & have the patch load those details & do the corresponding DB inserts through the add repo mechanism I guess
@n8fr8 right
@_hc yeah, just thru the add repo mechanism
@_hc send an Intent to the right Activitiy
@n8fr8 i wonder how the default repos asre stored in the code tho
@n8fr8 could we make this meta-inf behavior the default behavior
@_hc they are hard coded
@n8fr8 META-INF/repo.props
@_hc I think its good to leave the default repos hard-coded, less room for mistakes and exploits
pd0x Yeah
@_hc do you think that can hook the androidobs network stuff into the Downloader infrastructure in FDroid? Then we'll have a single place to implement Tor, etc
@_hc its still in the works
pd0x _hc: I don't think so. I think the requirements are too different
pd0x well, maybe not.
pd0x I think it could work actually
@_hc you need more than "GET http://ome.rul/?query=foo
pd0x I'm not POSTing anything
pd0x It should be fine if the Downloader can give me the response body from the GET's
@_hc ok
@_hc its a moving target currently, but I'll keep that in mind
pd0x that's where I am :-)
@_hc so that's all work in your fdroid fork/
@_hc ?
pd0x yes. I haven't pushed to gitorious, I have a rebase pending
@_hc ok
pd0x the observatory stuff is in a github repo I need to flip public
@_hc excellent :)
@_hc so I've been working on the HSM stuff, I can go into more detail if anyone has questions, but not much else to report, besides that I nuked my OpenPGP card in the process :-D
@_hc at least, they are cheap, and I have others
@_hc the yubikey-neo-piv and cryptostick are both looking promising as more "all in one" HSMs
@_hc but I haven't dived in very deeply into them
@n8fr8 yr going to blog about this? or just wiki it?
@_hc both
@_hc there is already stuff in the wiki
@_hc my goal is to ultimately come up with the one HOWTO to rule them all
@_hc but since its vastly more horrible landscape than I thought, that'll be a longer term goal
@n8fr8 gr8. ew can make it the standard for all infosec/humanrights app releasese
@_hc so I'm going to keep wikiing and blog on particular card setups
* n8fr8 is typing one handed with a baby in the other
@_hc I think we can also make it something for high risk people using gpg, ssh, etc.
pd0x _hc: which card were you able to get playing nicely with jarsigner?
@_hc Aventra MyEID
pd0x did you end up having to use a GPG->PKCS11 bridge?
@_hc no, its not a OpenPGP card at all
pd0x _hc: is that one a typical smartcard form factor, need a reader?

pd0x Ahhh. It's just PKCS11?
@_hc PKCS15!
pd0x I don't even know what that one is haha.
@_hc and opensc provides PKCS#11 from that card, and many others
pd0x Hate these obtuse standards names
@_hc oh, its soooo horrid
@_hc PKCS15 is the file format on the card
@_hc 11 is the way to talk to cards
pd0x ok
@_hc opensc is basically a giant translation layer
@_hc so you can talk to lots of cards via PKCS#11
pd0x Interesting
@n8fr8 so i was thinking about chatsecure sharing and was wondering if we could simplify things for now
pd0x Do you use a USB reader with the Aventra MyEID?
@n8fr8 and just use the kerlpapp UI for supporting single app sharing
@n8fr8 or something less than full repo sync
@_hc ha, if only if were that simple. there are many smart card readers, and they require varioius drivers, and they are not all compatible with all cards
@_hc so you need to have a matched pair of smartcard and reader
@_hc yubikey and cryptostick are both in one package
pd0x n8fr8: _hc and carrie and I talked about that a bit. I definitely want to see single app sharing supported easily but there's a tradeoff in not getting all the repo data (for the audit usecase)
@_hc and updates
@_hc full repo sharing means automatic updates
pd0x _hc: yeah, that's why I asked. Which reader did you get to play nicely with both jarsigner and the Aventra card?
pd0x _hc: good point, also updates.
@_hc ACS ACR38T, which requires a driver specific to it
@_hc the good ones are the ones supported by libciid without extra drivers
pd0x is the driver an os specific blob>?
@n8fr8 yeah... but i think the user story is at leadt "hey let me send this app to you" via otrdata
@_hc this is a good overview resource https://wiki.debian.org/Smartcards
@n8fr8 but somehow doing that in a non horrible way
pd0x _hc: thanks
@_hc the driver seems to be free software, its in Debian/Ubuntu
@_hc n8fr8: "let me send this app to you" can also be achieved by swapping repo data
@_hc and it gets those extra benefits
@n8fr8 ok.
@_hc we just need to make it clear to the user that their repo is shared upon every user interaction
@_hc we were thinking that the repo sharing always happens, but it should be possible to manually send an APK that is not in your repo
@_hc pd0x: match your recollection?
@n8fr8 i can add "share app" in chatsecure pretty easily
@n8fr8 w/o any bazaar stuff
@n8fr8 i suupose
@_hc true
@_hc we want to do it in a generic way in FDroid tho
@n8fr8 but i will think more again about the otrdata/hhtp proxy
@_hc so it works over bluetooth, local wifi, etc.
@n8fr8 otrdata/http
@_hc adding "share app" to chatsecure seems like an easy place to start to test the idea
pd0x maybe there's a way we can do a repo add/update & then launch directly into the details screen for a specific app listed in the repo metadata?
pd0x I'm not sure how at this point
@n8fr8 yeah, lets not confuse things
@n8fr8 i can explore the simple use case direclty
@_hc it would be easy to add, if its not already possible
pd0x but if it was already on the app detail screen with a big 'intsall' button that might be the same net effect
pd0x as sending an apk outside of the repo data
@n8fr8 okay so my other question is, is the priority to merge kerplapp into fdroid, or should we focus on a separate release to get something out sooner?
pd0x I think the separate app makes the whole thing confusing to talk about
@n8fr8 pdox: there is an easy "share to chatsecure" option where you can select a contact etc, via our intent API
@_hc let me look at the merge process
@_hc then make that call
@n8fr8 ok
@_hc it might not be that much work
pd0x n8fr8: cool
@_hc otherwise, we can release what we have now
@_hc it worked pretty well at the UX sprint
@_hc if you know how to do it ;)
pd0x Yeah. Could probably get better at communicating the two app process if we went that way
pd0x start using more of the terms we came up with Carrie (swap,etc)
pd0x less 'repo' & 'metadata'
@n8fr8 "start swappin'"
@n8fr8 how about "Swap Meet"!
@n8fr8 (is that an american thing?)

@n8fr8 Start Swappin'!
pd0x I get the reference but Canada is USA-lite...
pd0x Might be overly north american
@n8fr8 last thing: i think the META-INF/myrepos.prop can be used for static repos as well
@n8fr8 like i mentinoed in our other scrum
pd0x Agreed.
@_hc sure
@n8fr8 so i would like to have a HTML landing page in our gp fdroid repo that promotes an fdroid download with our repos in it etc
pd0x you could make a project specific fdroid by writing a .prop file and running 'zip -u META-INF/branding.prop FDroid.apk'
@n8fr8 i think that is better than forking fdroid to rebrand it
@_hc I think when bootstrapping, the META-INFO prop file shoudl include all static repos as well, but just not enable them by defalt
@_hc or maybe enable them by default
@n8fr8 right, that is what i was thinking before _hc
@_hc so its like "let me give you my fdroid setup"
pd0x all static repos from the senders fdroid?
@_hc yeah
pd0x hmm yeah.
@n8fr8 yes viral sharing of lots of repos is good
@n8fr8 over sharing always!
pd0x Sharing the fingerprint is good too. Removes a TOFU decision
@_hc hmm, I had one up here https://guardianproject.info/fdroid/repo/ but it seems to ahve disappeared
@n8fr8 but we do want our repo (or the shares) list of apps to be shown first, and not the full list of say fdroid repos
pd0x yeah. That'll bump into the FDroid client work that needs to be done for repo priority/filtering
@_hc newly added repos have precedence over older ons
@_hc ones
@n8fr8 ok
pd0x yeah. Priority for app duplicates
pd0x but the main screen would show apps from all active repos merged
@n8fr8 yeah but we just just want to see the apps we are telling you to install
@_hc that could be set in the META-INF/prop: which category to show by default
pd0x yeah. That's the ideal case I think. I sent you my FDroid, it has a META-INF prop bundle for my Kerplapp repo, and it opens to display just those apps
pd0x _hc: good idea
@n8fr8 ok
@n8fr8 coo... yeah ultimately the goal is to ahve a single URL (and perhaps a mirror on Amazon S3) that we can say is our bootstrap URL for our stuff
@n8fr8 and then all the other benefits of fdroid come along with it
@n8fr8 nifty
@_hc does anyone have an S3 account? I could push our repo to it. I'm supposed to be adding that feature to the fdroid tools, based on the psyphon script
@n8fr8 i have one that i am using for our stuff
@n8fr8 i can provide keys when you need them
pd0x That's neat. It sounds like a META-INF bootstrap would be useful for a lot of things
@n8fr8 https://s3.amazonaws.com/guardianproject/orbot-latest.apk
@n8fr8 etc is up there
pd0x synergy
@n8fr8 https://s3.amazonaws.com/guardianproject/fdroid/repo
@_hc n8fr8: cam you gpgemail it to me?
@n8fr8 and then https://s3.amazonaws.com/guardianproject/index.html can be our bootstrap page
@n8fr8 (you have to specify index.html)
@n8fr8 btw, the Psiphon3 guys also have a nice landing page for Iranians using their network (a few million of them)
@n8fr8 and we can get our bootstrap page/fdroid download there likely
@_hc ah, here, I have a scetch page up: https://guardianproject.info/fdroid/
@n8fr8 some other people have similar things already, so this one click bootstrap flavor of fdroid is useful
@n8fr8 great
@_hc I'll add a qrcode too
@_hc just because ;)
@n8fr8 yeah and link to get fdroid
@n8fr8 this would be neat too for the librarybox "app box" work
@n8fr8 since it can serve a static fdroid.apk tied to its http://librarybox.lan/content/fdroid/repo url
pd0x I'll add a ticket to the tracker for the META-INF bootstrap
pd0x and I can put the IRC log on the wiki if we're more or less wrapped up?
@_hc yes please :)
pd0x n8fr8: yeah. I really like the idea of having a preconfigured FDroid APK & a URL with the fdroidrepo(s):// url
@_hc I'm adding a ticket about the repo setup (s3, index page, etc)
pd0x If you have FDroid it's a one click add. If you don't it's a one click download

Nov 21st IRC log about identifying repos

_hc: mvdan what do you think about including the fingerprint in the DB?

_hc that would make it much easier to add repos that include the fingerprint in the URL

_hc fingerprint == fingerprint of DB.pubkey

CiaranG: _hc How would it have the pubkey already in the database?

mvdan Sure, though remember to bump the dbvercode and update the repo database

CiaranG Or am I misunderstanding?

mvdan (I meant table)

_hc: CiaranG it wouldn't yet

_hc so when it first makes the socket connection and gets the pubkey, then it would have to compare the store fingerprint

_hc but it would have the pubkey and fingerprint if the repo already existed in FDroid

_hc client

CiaranG Oh, ok, that makes sense

_hc then when we have an incoming repo that includes a fingerprint, it can easily compare both the URL and the fingerprint to what's stored in teh DB

CiaranG So it's just a temporary storage of it until it gets the actual public key

_hc I suppose

_hc it wouldn't really be temporary since it would be used for comparisons whenever there was an incoming repo

CiaranG It's not tied to a particular repo?

_hc otherwise the Add Repo UI will have to make a socket connection to get the pubkey

_hc it would definitely be tied to a certain repo

CiaranG Right

CiaranG So what I'm saying is that you have your incoming url+fingerprint

_hc it would be one of the standard bits of data stored about a repo

CiaranG But once you've actually connected to that repo during an update, you get the real pubkey

_hc exactly

CiaranG At which point, the fingerprint is never needed again

_hc it would be needed the next time someone adds a repo with a fingerprint, since that's how AddRepo checks to see if the repo already exists

CiaranG Why does it check the fingerprint, and not the url?

_hc it would check both

_hc but actually, we're planning on implementing bluetooth and OTRDATA methods for contacting repos, in which case, the pubkey becomes more of the identifier of a given repo

CiaranG aha

CiaranG So that's why

_hc now that I think about it, the fingerprint could be removed from the database once the pubkey is downloaded since the fingerprint can be caluclated from the pubkey

CiaranG But that means a) you can't have an unsigned repo? and b) you can't have two repos signed with the same key?

_hc hopefully we can still support both

_hc I think we cna

_hc but yeah, that's something to think about

CiaranG I was about to say I don't think b) is a problem - I was only raising it because it's a restriction that's never existed before

CiaranG But actually it is a problem

CiaranG f-droid.org/repo - f-droid.org/archive

CiaranG Two repos, same pubkey

_hc for /repo and /archive?

_hc hehe

_hc yes, something to think about

_hc in the meantime, I'm proposing just adding the fingerprint to work with HTTPS URLs

_hc we're still working out how to handle bluetooth and OTRDATA

_hc we have a working prototype of the p2p HTTPS repos

CiaranG Seems to me they could still have a unique 'url'

CiaranG Even if it's not a real url

_hc yeah, that's one possibilitiy

_hc I think that works with bluetooth since there is some kind of unique id that represents the device

_hc the tricky one is local wifi

_hc and with OTRDATA we should be able to use the name of the XMPP account of the other device in the URL

_hc with local wifi, it'll share the same URL and code as the static HTTPS servers, but will be changing a lot

_hc URL-style

_hc I suppose we could make up a fake hostname based on a unique id of the phone

_hc something like a hash of the devicename and Android ID

Oct 23rd IRC Scrum log

n8fr8 okay so devrandom _hc pd0x we are now having an irc scrum

n8fr8 about the Bazaar project

n8fr8 for anyone else listening in

n8fr8 you can learn more about this project at: https://dev.guardianproject.info/projects/bazaar/wiki/

n8fr8 in short though, we are trying to support p2p or decentralized app sharing, syncing, etc

n8fr8 building on some of what f-droid has done, and also using chatsecure's new otrdata share feature

n8fr8 so....

n8fr8 i haven't had a chance to build pd0x's kerplapp repo code yet

n8fr8 but i am hoping to soon

devrandom OK

_hc my notable output of the week is this wiki page: https://dev.guardianproject.info/projects/bazaar/wiki/Trusted_Intent_Interaction

_hc thinking about ways to verify the sender or receiver of an explicit Intent

_hc to be bundled up into an easy to use library

_hc this is contingent to the core idea of chaining validation methods not being invalidated ;)

devrandom "domain name"? is that the package name?

_hc I should reword that

_hc it could be APK hash, signing key, or package id

_hc i'm feeling a bit sick, so I was mixed this up with the idea for chaining validation of HTTPS and TLS

devrandom sorry to hear you're sick

n8fr8 well, i am sick AND tired! :)

devrandom uh oh

n8fr8 not really

n8fr8 i just wanted to complain

n8fr8 feeling quite healthy

devrandom :)

devrandom kvetch away, we'll suspend disbelief

devrandom so this is not really like chaining certs

_hc hmm, so I guess I haven't thought that much about how to chain this here

_hc right

devrandom in chaining certs, A signs B and B signs C

devrandom in this case, these are fallbacks

_hc no

_hc ok

n8fr8 let's not get into certs right now

_hc I was thinking of chaining methods of validation, but if that's confusing, it can be worded differently

n8fr8 in this context, we are tlaking about how to build trust between apps, beyond just the Intent permission model

_hc yeah

n8fr8 so right now in ChatSecure

_hc skip certs for now, that's a separate discussion

n8fr8 we prompt the user when an app is asking to creaet an account, or create a group chat

n8fr8 etc

devrandom I like the idea of trusting at different levels (at the APK and signing key levels for example)

devrandom but ok

_hc yeah, same idea but for Intents

_hc not HTTPS/TLS certs

_hc what would be the token? that's where I 'm spacing

devrandom I do mean for intents... so if an intent comes in, and the APK was not ToFU'ed but the signing key was, then it's all good

devrandom token?

_hc with HTTPS, the token that you're matching is the domain name

_hc in pinning and tofu

devrandom the token would be the APK hash or the signing key hash (aka fingerprint)

_hc is there a pin for this domain name?

pd0x hi, sorry here now.

pd0x looking over the Trusted Intent page

pd0x don't judge me too harshly if you dig into the Kerplapp repo. It's pretty unpolished <cringe>

_hc you gotta start somewhere :)

pd0x I'm still fighting with the fdroid client to get it to install apps from a Kerplapp repo. Almost there, the metadata all parses letting you browse the repo/apps doesn't want to install an app fresh or as an update (even when sigs match)

_hc so the thing I haven't figured out yet with IntentPinningTofu is what's the token/key that the process starts with

n8fr8 pd0x: maybe you can setp back from the f-droid debugging

n8fr8 and just have it spit out an HTML page with all the apps

n8fr8 and links to download?

n8fr8 by "it" i mean your nano server

pd0x n8fr8: at that point already :-)

n8fr8 just to debug that aspect

n8fr8 ah ok

pd0x nano hosts the index in a way that you can HTTP get the APKs

n8fr8 so is that working? the issues i somewhere in f-droid client?

pd0x I think the issue is with the metadata for each APK in the index.xml file Kerplapp builds

pd0x F-droid parses it OK but it gets into an inconsistent state in the fdroid client app

pd0x so in there when you click to install an app that isn't present on-device it tries to uninstall it first

pd0x and that fails because it isn't there

pd0x I'm trying to sort out where/why it gets confused into thinking apps not present on-device need to be uninstalled before the APK can install

pd0x If you directly navigate to the kerplapp repo with a browser and download the APK it will install

pd0x _hc: the idea with the trusted intent interaction - what's is the pin? Is it the subject public key identifier from the cert that signed the APK?

pd0x _hc: or a hash of the APK (both? neither?)

_hc I'm thinking perhaps the package ID is the token, and the signing key and APK hash are the things that get looked up.

pd0x _hc: Ok. Involving the APK hash means you're pinning to a specific release/version

_hc so packageID is the key, and the signing key or hash is the value, depending on the prefereence

pd0x _hc: I don't think that's a bad thing, just want to make it explicit because it will be potentially confusing to a user to TOFU the same pkg when it updates

_hc yes, using the hash would be the more extreme case, like to work around the master key bug

pd0x Gotcha

_hc TOFU is probably more useful with the hash rather than pinning

pd0x agreed

_hc but maybe not...

_hc well, yeah, guess so

_hc since the master key exploit can be done by malware

pd0x signature pinning is basically what Google did for the Authenticator -> Authenticator2 export

_hc some time after the install

_hc signing key or apk hash?

pd0x signing key

_hc hash pnning could be the key to upgrading our signing key on existing apps

_hc new package-id/signing key

_hc both installed at the same time

pd0x it's probably the safest way to do data migration to a new pkg/key

_hc old one grants the new one perms to read all data based on hash pin

pd0x I like the idea of building it as a library ala moxie's pinning lib

_hc yeah, that's the idea

devrandom how is Bazaar going to use this?

_hc for communicating with ChatSecure for OTRDATA

_hc maybe also for talking to GPG

pd0x to sign the index.jar with the OTR DSA key

pd0x as another potential idea

devrandom interesting

devrandom oh, maybe ChatSecure should be able to bootstrap Bazaar through OTRDATA

_hc pd0x: I forget if we covered this: if there are multiple sigs in the index.jar, do they all need to be present to validate? or is that just Android?

_hc devrandom: definitely

_hc but more likely, bazaar is going to bootstrap chatsecure

_hc i.e. starting from a blank, new phone, first install Bazaar, then the rest follows

_hc but there are lots of chatsecure users that can go the other way

devrandom yes

pd0x _hc: Not sure I understand the question. Are you talking about multiple signatures on the Fdroid repo index.jar or multiple signatures on an APK (one listed in the repo or otherwise?)?

_hc multiple sigs in the repo index.jar

devrandom in or on? ;)

_hc I believe its in

_hc doesn't the jar format include the sig?

pd0x the fdroid client (presently) only checks one signature on the index.jar. When you first configure the repo if it finds a pubkey attribute on the index.xml's repo XML then it will subsequently compare that cached attribute value making sure it's the Signature on the index.jar

pd0x It's a hash of the PKCS7 bytes IIRC

pd0x it doesn't care how many signatures there are as long as the one specified in the index.xml from repo install is there

pd0x there isn't any support for more than one pubkey attribute in the XML (as best I can tell)

_hchttps://dev.guardianproject.info/projects/bazaar/wiki/Signing_the_Local_APK_Index :-D

pd0x and yes, jar format includes the signature in META-INF/ as a *.DSA, *.RSA, *.EC file next to the .SF

pd0x that file is a PKCS7 structure with the signature & a self signed certificate with the pubkey info for the signature

pd0x _hc: thanks for transcribing :-)

devrandom are we thinking yet of multiple signing keys with thresholds? (like Gitian)

pd0x devrandom: you could probably do that for the index.jar, you can't for APKs

pd0x at least not without OS support

pd0x I was futzing around with libgfshare + openssl for splitting an Android app signing key using shamir's secret sharing to do thresholded releases

_hc I added more to that page too, feel free to edit, I'm out of it

devrandom well, Bazaar could do the extra checking before handing over to the OS for install

pd0x but it's not as ideal as true threshold signatures because you have to recombine the n shares on the same machine revealing the private key for signing ops :-(

devrandom (I didn't mean real threshold sigs - just a quorum of signers)

pd0x devrandom: If Bazaar did the checks and handed things over you'd still have to get all updates signed by the same threshold unique signatures for the OS to install it

pd0x exact same

devrandom sorry, I think I was unclear

devrandom I mean the following flow:

pd0x listens.

devrandom * multiple builders build the package

devrandom * each builder signs the package

devrandom * Bazaar checks that m out of n known good builders signed the package

devrandom * Bazaar then hands off to the OS for install

devrandom .

pd0x and the m out of n signatures, are they applied to the APK via standard Android code signing?

pd0x using jarsigner that is

devrandom the extra sigs can be attach using a similar (but not identical) mechanism

_hc that could be useful, but it seems a later step

_hc right now, its just about getting a trusted index from one phone to another

pd0x if they're applied with jarsigner then Bazaar can do the m out of n check but if m ever changes it breaks updates

devrandom no reason to confuse the OS with multiple sigs that only Bazaar wants to look at

pd0x bazaar could strip m down to 1 once it was happy

pd0x but then it's a question of which 1 of m

devrandom oh I see

pd0x otherwise the scenario is Bob goes from NoApp -> Bazaar Vetted APK signed by A & B (of ABC)

pd0x Then trusted builders change, APK arrives signed BC

devrandom just declare one builder as the "main" one for keeping the OS happy

pd0x Bazaar is happy, OS isn't

devrandom and that one is mandatory

pd0x because the OS mandates that all updates to AB app are signed by exactly AB and only AB

pd0x devrandom: could do that

devrandom alternatively, Bazaar could fake-sign the APK for installation purposes

_hc then we have APKs that can only be installed by Bazaar

devrandom so that only Bazaar can upgrade the APKs

pd0x _hc: indeed. not idea

pd0x I

_hc since it would have to strip out the extra sigs

devrandom using a randomly generated cert

pd0x devrandom: hmmm. yeah.

_hc so might as well use detached sigs for gitian sigs

_hc since they would be stripped out anyhow

n8fr8 okayback now

pd0x devrandom: you can 'unsign' an APK by deleting the META-INF sig entries

devrandom right

devrandom it's just files in META-INF

pd0x devrandom: so you're imagining after Bazaar verified the quorum on APK it'd create APK' by striping the sigs and applying one with its own priv key

devrandom right

devrandom or leave them in the META-INF, since they would use filenames that are not seen by the OS

n8fr8 woah that is an interesting idea

n8fr8 resigning the apk after we have verified it

pd0x it's a little bit divergent from what bazaar is now

devrandom SteeleNivenson: see ^^ re ostel.me broken links

pd0x because it means you have to get updates to the bazaar installed apps throguh bazaar forever

pd0x or go back to a clean slate

n8fr8 right...

devrandom yes, that's a tradeoff

pd0x I guess I imagined sharing an app through bazaar and the other side being able to update it via Google Play/etc later

n8fr8 well if you don't ever have access to googl eplay

n8fr8 then bazaar could just intercept downloaded APKs

n8fr8 liek lookout does

devrandom but I'm pushing the gitian project these days (Debian seems to be into it ... ;))

n8fr8 well, we should consider the "no google play" option as one type of important user

pd0x devrandom: don't get me wrong. I <3 <3 gitian

pd0x devrandom: android code signing is poor :-(

devrandom thanks :)

pd0x spent a lot of time thinking about alternatives and ended up at needing OS support for much

devrandom in any case, we could keep the external upgrade path alive by keeping the "main builder's signature" on the APK

pd0x n8fr8: agreed. I think we'd need to spend more time thinking about it. Without the Gitian type setup to produce the quorum builds we don't have the need yet either

n8fr8 i also like _hc's detached signature as an option

n8fr8 with ChatSecure and otrdata/dataplug support

n8fr8 we can transfer a variety of data types more seamelessly

pd0x yeah, that seems an easier path to start with

n8fr8 than just limiting ourselves to only fdroid metadata

n8fr8 so perhaps that is a feature once you level up to chatsecure+bazaar

n8fr8 I.e. keeping/store multiple deatached sigs per apk

devrandom pd0x: curious if you inject stuff into META-INF, does that invalidate the APK sig at all?

pd0x devrandom: it doesn't. That's how you can add a signature to an already signed APK without invalidating the first one

pd0x running keytool on an already signed app will (assuming no key alias collisions) drop another .RSA/.DSA/etc file in the META-INF

devrandom OK... so in that case, we should keep in mind that we can just inject whatever additional metadata through that route and it can stay there

pd0x it will change the APK hash though, a detach sig wouldn't

devrandom true

pd0x not sure what that tradeoff costs us, likely nothing if we aren't pinning to it. I doubt anything else does

devrandom one option is to use the algorithm from the Android verifier to get the hash of the actual data in the APK (minus the META-INF)

_hc pd0x: n8fr8 devrandom I have to run now, can someone post the log of this meeting on https://dev.guardianproject.info/projects/bazaar/wiki/Wiki ?

devrandom does fdroid hash the APK or the internals?

_hc also it would be good to have a wiki page about it there

pd0x devrandom: the APK

devrandom I see

- Blogpost about Bazaar: https://guardianproject.info/2013/11/18/turn-your-device-into-an-app-store/
- Core concept: https://dev.guardianproject.info/projects/bazaar/wiki/Core_Concept
    - This page should note that Cydia is a widely-used decentralized software installer for mobile (based on APT) that has tons of people running their own repositories to distribute their own software, including some experiments running on-device repositories: http://www.idownloadblog.com/2011/09/03/how-to-create-your-own-cydia-repo/ / http://www.redmondpie.com/host-your-very-own-repository-on-cydia-with-irepo-for-iphone-and-ipad-jailbreak-tweak/
- User stories: https://dev.guardianproject.info/projects/bazaar/wiki/User_Stories
    - (semi unrelated to this team, Twine game system for prototyping interactions - http://www.auntiepixelante.com/twine/)
- "bazaar" has some inconvenient name collisions

if this were a blog post, what would the title be? what's the theme?

here are our guesses for why this should exist. here are some obstacles (that we can think of) to getting it exist.

more notes about cydia stuff

so an idea for bazaar-like results on jailbroken ios: set up a package like "iRepo" (linked above) that includes a lighttpd configuration file and depends on lighttpd (http://cydia.saurik.com/info/lighttpd/), ask people to turn on "personal hotspot" (http://support.apple.com/kb/ts2756) to share over bluetooth, and then ask people to add the device as a repository. if they have a phone that doesn't already have personal hotspot (some carriers don't support it), they can install a package that enables personal hotspot, like tetherme (which is $5). also possibly vaguely interesting for prototyping: https://code.google.com/p/cydia-ios-lighttpd-php-mysql-web-stack/

Could be interesting to document how to do this and share the link in the jailbreaking community (such as by linking it on http://www.reddit.com/r/jailbreak/), to see if people are interested in it and have ideas for how to use it - cheap customer validation.

fitting what people want

How do we encourage the use of Bazaar?

one friend is interested in easy developer-to-phone app distribution (along with easy friend-to-friend app distribution)

Who can we find to talk to who would actually want to use this, to find out what they would find useful? (informal customer validation)

Seems hard to find people to interview who have limited internet access, since they have limited internet access - argh. Or people who only have access to sketchy app stores, since they probably don't speak English - argh.

User stories: https://dev.guardianproject.info/projects/bazaar/wiki/User_Stories

Promote it as a safe alternative to sanctioned/blocked app stores (contextual) and the sketchy sites, which most users already suspect of being filled with malicious apps (or maybe they dont). A few examples of malicious apps dowloanded from random webistes might promote users to use Bazaar. Connecting with trusted groups in the diaspora with community outreach to promote and vet Bazaar. Use cureent trust networks in the diaspora.

- [There aren't all that many examples of traditionally malicious apps (information stealing, etc.) even in very popular sketchy pirate app stores though. But government intervention in communication apps is a plausible/worrying thing.]
    - there are examples of backdoored circumvention tools distributed through alternative sources (There has been a few Psiphon clients, both Windows and Android. And the most known one is: https://citizenlab.org/2012/05/iranian-anti-censorship-software-simurgh-circulated-with-malicious-backdoor-2/ - although it was a windows only software, I think the example is still relavant.
        - [Nice example, thanks!!]

Bazaar can also be used to distribute apps that are no longer on the app store but are still installed on some user's devices. The FlappyBird use-case?

- Talking about this would get into copyright infringement stuff that isn't so good for reputation...probably. More legit-sounding to focus on communication tool distribution for revolutions
    - Are there cases of apps being removed for reasons related to platform policy but not piracy? e.g. Google removing apps that compete with their core services? Something that wouldn't cause the same reputational concerns? Maybe someone knows of examples....
        - Oh yeah, Apple has removed or rejected lots of politically controversial apps, but there are already alternative distribution paths for them (websites that distribute pirated apps also distribute them) - http://www.wired.com/dangerroom/2012/08/drone-app/ http://bits.blogs.nytimes.com/2011/09/13/game-that-critiques-apple-vanishes-from-app-store/ etc.
        - ((Adblock Plus is a good example of an app that Google censored that we would feel comfortable people Bazaar'ing to each other))

Encouraging the uptake of "run your own" F-Droid repositories:

Cydia has a distributed community with 1000+ individual repos. Repositories as a first-class feature of the cydia app. F-Droid repository support is largely vestigial, and as a result few individual repositories have been created.

A lot of the large independent Cydia repositories are language-specific; the default/core Cydia repositories are fairly English-specific, so jailbreakers who speak other languages self-organize and run their own repositories and forums.

explanation for why we care about the below social/technical trust questions

Decentralized peer-to-peer app distribution introduces lots of room for people to modify and distribute apps with malware added. We want people to be able to check, with some reasonable amount of confidence, that they're using an app that hasn't been tampered with to add malware or evil government stuff. Bazaar requires users to allow "installation from third party sources", exposing them to threats not otherwise present if the user only installs applications from the Google Play store.

Assuming that some of the current apps are malicious (downloaded from random websites do to sanctions and censorship), this could enhance promotion and trust in Bazaar as an Alternative.

social forms of trust

What are the existing social structures + social media sources that people will be using alongside Bazaar? (How do we build on existing trust networks?).

How can we exploit the "trusted techie" angle. If one technically sophisticated individual is bootstrapping friends it is likely those friends defer to the technical user for questions of trust/safety.

Using trusted networks and organizations in the diaspora. Students and individuals travlleing back to home countries.
What does the "information" page for an app look like?
- Something along the lines of: https://androidobservatory.org/app/6270E8DD74F67A2C117335B5278AF91EDF440C28 ? What other information would be useful? How can it be made more "accessible"?

Examples of already established social structure and practices of peer to peer sharing. Especially offline methods that are more accessible and widely used to exchange other forms of information.
Where does this information come from?
- Organzaiton in contact with targeted endusers. Localization, translation, community management adn outreach.

Will Bazaar run some kind of centralized web site to help provide information? (languages)

Maybe a guidline that can also be shared among users, emailed?

technical forms of trust

How do users verify that an app is one they can trust? (What kind of signature checking/verifying is readable/usable for people?) How do we deal with people having different legitimate versions of the same app, causing different signatures?
How to solicit information about app signatures/sources?
- When a user uploads an APK sample to the Android Observatory we want them to be able to include contextual information about where they found the app, suspicions they may have about the integrity/quality, etc. Current upload page offers no chance to provide this information (https://androidobservatory.org/upload). Perhaps the most important question is "Where did you get this app sample? Your phone? An app store? A forum? A friend?" * Similarly, it would be advantageous to allow users/uploaders to "tag" signatures when they know more about the certificate than we do. I.e. there are a number of "public" private keys used by developers:
https://github.com/CyanogenMod/android_build/tree/gingerbread/target/product/security
- Example of an app signed with the CM platform key:
  - https://androidobservatory.org/app/07D66B55D7BB7318003C38E40FF6DF2B8F68CEB1

- How to present errors/problems with apps?
  - Types: of errors/problems we can detect:
    - Signature mismatches of Applications - two package names using a different set of signing certificates
    - Two apps claiming to be the same version code with different hashes - eg two "Version 5's" of Facebook should be byte for byte identical.

An issue from many other projects: signature mismatches are usually innocent technical errors with setup for repositories and packages, so users learn that these errors should be ignored, even if they have scary language. (On Android it isn't possible to install apps without a signature, but debug keys will often be used accidentally)

- E.g. of sig mismatches:
  - SwiftKey 3:
    - https://androidobservatory.org/app/3A0BEA753B955560165CC961F7EF64C0A80EFA07
    - https://androidobservatory.org/app/1E53BD6A8B80C5F39E9ED9094578B27EE0FCB4D7
  - TweetLanes:
    - https://androidobservatory.org/app/6270E8DD74F67A2C117335B5278AF91EDF440C28
    - https://androidobservatory.org/app/C10BB6A335F603E589F3DA15EB71AA170E690C46

OTRDATA Integration Plan

OTRDATA / Dataplug

Bazaar plans to use OTRDATA as one of the transports. In particular, it will use the ChatSecure Dataplug API.

The Dataplug quickstart guide

Schedule

- ChatSecure V14, alpha - 13 Nov 2013 - merge the dataplug branch into ChatSecure master. Begin writing a Bazaar transport using the Dataplug API.
- ChatSecure V14, general release - 20 Nov 2013 - release with dataplug merged in but not enabled in the general release. Complete first version of Bazaar Dataplug transport. Make any necessary improvements to Dataplug API and OTRDATA.
- ChatSecure V15, general release - 4 Dec 2013 - release with dataplug enabled. Bazaar is able to use this version in production.

Questions and Answers

How is this platform an attractive proposition to non-blocked users?

FDroid was built from the ground up to avoid tracking and respect privacy. FDroid has already attracted a lot of attention for being the most private alternative to Google Play, (which gathers a lot of data from the users' devices). For that reason, it is included in Mike Perry's Mission Impossible project, which aims to be the most private Android setup.

Also, FDroid already provides a lot of flexibility for how apps can be distributed. This gives organizations many options that are not available via Google Play, including: distribution channels without the internet; and, very private channels for a small set of high risk users.

h3.How aware are developers of the existing distribution deficiencies? Please provide more information on the deficiencies themselves as well.

Developers are generally aware of major, widespread deficiencies, like when things do not use HTTPS, but overall there is not a lot of scrutiny of the security of developer tools. Tech media also spends much less time on security issues that only affect specific developers since it applies only to a much smaller audience. For those reasons, we will entice developers with secure tools that improve their workflow rather than first trying to convince them that they should be concerned about exploits in their tools.

The most common type of deficiency is developer tools that download and execute code without a decent method for verifying they are unchanged, like Maven and Gradle. Those both also will not warn people if the download happens over plain HTTP, yet with either, a per-project config file can force HTTP. Another common practice is to blindly trust downloaded `.jar` files when adding them to projects.
Here are some examples that are relevant:

- https://guardianproject.info/2015/02/24/phishing-for-developers
- http://www.forbes.com/sites/andygreenberg/2013/02/20/developer-site-used-to-hack-facebook-and-apple-issues-mea-culpa
- http://arstechnica.com/security/2014/10/hp-accidentally-signed-malware-will-revoke-certificate
- http://blog.ontoillogical.com/blog/2014/07/28/how-to-take-over-any-java-developer
- http://arstechnica.com/security/2015/05/researchers-uncover-self-sustaining-botnets-of-poorly-secured-routers
- https://github.com/gradle/gradle/pull/448

h3.How will this new system be pitched as a value add to developers?

There are many tasks in the process of managing Android app releases that can recently be automated. These are tasks that no developer enjoys, and act as a barrier to every release. This system will include all these things into a single tool set. Then these tools will be an easy sell as part of Guardian Project's and FDroid's regular outreach to developers because the time savings are easy to demonstrate. Here are tasks that will be covered:

- releasing new builds to Google Play
- managing including translations in app stores
- seeding of releases in malware scanners like VirusTotal
- deploying beta/alpha/nightly releases
- deploying to multiple channels (Play, FDroid, Amazon S3, etc)
- a system of deploying test builds that allows for upgrading and downgrading

h3.What new feature or characteristic will factor in most prominently to this promotional effort?

In promoting to users, we are planning on two new approaches, on top of our standard outreach methods:

1. market FDroid as "blocked apps repo"
2. get official FDroid distribution of some key apps

We are taking inspiration from Great Fire with their Free Weibo project that catalogs censored messages from Weibo. That got a lot of attention in China, and drove lots of users to their offerings. We will get list of blocked apps from Baidu, Tencent, Xiaomi, etc. and built up an app store of blocked apps for use in China.

To expand the offerings easily available in FDroid, we will work with our contacts at Lookout, Firefox, Twitter, and others to set up official distribution channels for their apps within the FDroid ecosystem. Lookout already provides direct downloads for devices without Google Play. Also, we succesfully lobbied Twitter to add proxy settings in order to support Orbot/Tor, so we think they could be receptive.

h3.What will be F-droid's involvement in this project? Will they be participating in pushing adoption and/or deployment?

The FDroid team is focused on their core mission of providing the premier Free Software app store for Android. That overlaps a lot with our mission, so that work directly aids this Bazaar2 project. For example, FDroid wants to prove that the app builds only using available source code, which is the same functionality needed for reproducible builds. They will continue to promote FDroid, and independently working to get FDroid included into more custom Android distributions like OmniROM and CyanogenMod. FDroid team members regularly speak at various developer and free software conferences in Europe and Australia.

h3.Are the budget numbers safe approximations or hard numbers?

Since this kind of funding is based on fixed amounts, our budget numbers are the amount needed to guarantee minimum useful functionality. Our track record shows that we usually deliver a lot more than the minimum. That said, reducing the funding of a given chunk would not guarantee failure, but it would increase the chances of ending up with incomplete code.

h3.Why is the scope of Bazaar 2 so much larger than Bazaar 1?

The first phase of Bazaar was an experiment to see how we could open up app distribution. It was limited in scope since it was not clear whether it would be successful. Now that we have that core functionality built and it is being used, this approach has proven both clearly viable and quite valuable. So for this proposal, we instead outlined the complete platform as we imagine it to order to greatly expand the coverage of these solutions.

Have the vulnerabilities identified by the Cure53 audit been addressed?

All vulnerabilities identified in the server-side were fixed in January 2015. The high risk vulnerabilities in the Android client were fixed by early February 2015. As of version 0.89 of the FDroid app, all vulnerabilities from the audit have been addressed.

h3.Do other malware protection tools exist? How effective are they?

There are some good malware scanner apps available in Google Play, like Lookout. Starting with Android 4.2, the operating itself provides some malware scanning. Google also runs regular malware scans on the apps in the Play Store. These are reasonably effective, and we recommend people install Lookout on their devices. The malware scanning in FDroid addresses devices without Google Play (which includes most devices sold in China), as well as the very low end devices that are still being sold with Android 2.3.3.

h3.Could the top apps receive more rigorous security testing than the testing proposed?

Top apps will definitely receive more attention. One key goal of this project is to have the top apps built using a reproducible process, guaranteeing that the versions included in f-droid.org only include what is actually in the source code, and nothing else. This provides a trustworthy distribution chain all the way from the developer's source code to the end user. The most common malware pattern that we see is to take the binary APK file of a popular app, then modify it to add exploits like Master Key, like the various fake Angry Birds.

Work is already underway to get apps from Firefox, Guardian Project, and LEAP building reproducibly. Mobile Firefox (aka Fennec) is also the base for the Tor Browser for Android that we are in the process of finalizing. We are happy to assist any Android developer in making their build process reproducible, and regularly propose to key developers that they adopt a reproducible process.

Providing wholesale malware testing or auditing is beyond the scope of this proposal, since there are existing products like Lookout available without payment and it would be quite expensive for us to build and run such a system.

h3.What other avenues do you expect attackers to take?

As the encryption of network traffic becomes the default, attackers and agencies looking to track, monitor, and exploit devices via the network will have to look to new methods. Commercial malware like Finfisher mostly exploit users when they visit websites using HTTP. But as recent leaks have shown, those companies are working hard to diversify their exploit methods, which includes more emphasis on Android malware.

h3.Why is budget included for translation? Could this be replaced by the OTF Localization Lab?

We are big believers and users of the OTF Localization Lab. We have found that it works very well for many languages, like French, German, and Spanish, but other languages that are more important to this project, like Arabic, Tibetan, Chinese, Persian, and Burmese get very few contributions from volunteers. Therefore, we must hire translators in order to ensure that we get translators for those languages.

h3.Why was the budget reduced by $40,000 after the first round of feedback?

We are finalizing contracts with Benetech to implement features in FDroid as part of their Secure App Generator project. They expanded the scope beyond what we originally thought when writing this proposal, so $40,000 of that work is directly applicable to the work outlined here. Also, the FDroid team has implemented more of the peer-to-peer swap process than we thought they would take on.

Response to the Second Round of Feedback

At its core, this project is entirely about usability and that is why we believe that OTF is the best funder. We appreciate the push back that we get from OTF, especially from non-technical points of view, because that keeps us focused on what matters most in real world situations while we are mired in the swamps of annoying technical details. Based on feedback, we have reduced the scope of the project to keep the focus on making proven technology usable by the target audiences that most overlap with OTF's and Guardian Project's missions. We eliminated the more experimental aspects, like extending the F-Droid platform to also work on desktop and iOS. We also reduced the focus on the less urgent security issues, like reproducible builds of the tools used to build Android apps. This project serves three audiences, listed by priority:

1. users: people installing, sharing apps, videos, e-books, etc
2. curators: people and organizations wanting to promote a specific set of apps, videos, e-books, etc
3. developers: people and organizations making apps for/in high risk contexts who need to distribute them safely, securely, reliably

This project will make proven techniques usable in the contexts important to internet freedom, specifically in places where there are strong, active forces working to thwart the free flow of information. The tools required to circumvent censorship and prevent tracking are currently not easy to use. Starting with modern mobile user experience design and a core of well proven ideas like Tor support, signed metadata for secure distribution, and app stores that handle media, we will integrate the proven techniques into a system that is as transparent and familiar as possible. Every technology in the project is already functional, whether widely used or a prototype. Certain arcane technical approaches can unlock possibilities for a more usable ecosystem by eliminating certain hard problems from the system. Reproducible builds are in this project to enable key usability improvements for developers, which in turn makes it easier for them to distribute apps in a flexible yet secure way.

h1.Questions and Answers

h3.Why these particular budget reductions?

We have reduced the overall scope and budget of this project in direct response to the feedback that we have received. Here is our rationale behind the reductions:

- We reduced costs related to partner deployments by removing the development of new features. This shifts the entire focus of the work with parners to usability testing and research, based on their contexts.

- While still a pressing issue, reproducible builds are less urgent than providing access to the secure distribution channels of the FDroid ecosystem

- Over the past year since this proposal was first submitted, there has been substantial work contributed to Guardian Project and the F-Droid project with overlaps with the goals of this project, for example: * https://fossdroid.com - a modern web interface to the app store * gplayweb - tools to curate custom F-Droid app stores * ACRA integration * Debian Reproducible Builds - over 80% of Debian can be built reproducibly now

- We removed work on other platforms to focus on delivering on Android. We predict users will be most interested in distributing apps, and apps in F-Droid are only Android.

- We received substantial translations from community, but we still funding to finish those community translations. We are big users of the OTF Translation Hub, and believe it is a very valuable and cost-effective resource. Organized community contributions work very well for building up baseline translations and keeping polished translations updated when the software changes. For certain target languages like Tibetan or Belarusian, community contributions are rare. Also, in order to get completed, polished translations, it is necessary to hire translators It is for this that we included funds for translations.

h3.Is this effort more focused on giving users a means to access censored apps via the play store or to combat malicious app distribution? Is it about safely distributing apps or everything including media?

This project is focused on guaranteeing access in the face of a variety of forms of restriction and tracking, with apps at the center and media as part of the package. An essential part of that problem is combating malware, as we can we see with Google's efforts in their own app store. This proposal includes only essential malware prevention, e.g. things that can be statically detected and blocked. It does not include to develop new anti-malware approaches, since those are available from third parties like Lookout, VirusTotal, etc. Instead it integrates existing source code, and where possible, like with https://virustotal.com and https://andrototal.org.

We included media in the project due to feedback from organizations like Amnesty, Internews, StoryMaker, etc. For example, this would allow a trainer to use a local swap to distribute the apps with the associated training materials in eBook and video form, with no internet required. Combining apps with media is a proven model, as seen with iTunes, Google Play, Amazon, etc. F-Droid already provides reliable infrastructure for moving large blobs of data around, so it will not require much work, and the technical work will have a very low risk of going over budget. Therefore, we believe that including media is an important part of this project.

h3.The proposal did not adequately explore the means and impact of malicious actors blocking access to F-Droid.

We know of no other app that does more to get around blocking than the plan we've laid out here:

- access app stores via Tor/Psiphon/Lantern/etc
- anyone can easily setup an app repository anywhere on any network, even off the internet
- "collateral freedom" mirrors on Amazon S3, GitHub, and others
- automatic fail-over mirrors when one is blocked
- entirely local app "swapping" via Bluetooth, local WiFi, mesh, etc.
- FDroid itself can be installed via all of the above channels

FDroid already uses NetCipher to provide solid support for secure networking, as well as integrated support for Tor. It uses Moxie Marlinspike's AndroidPinning library to limit HTTPS certificates to a trusted set of Certificate Authorities. We are currently working with Psiphon, Lantern, and Tor obfuscated transports to get support integrated into NetCipher, which FDroid will then inherit. The FDroid ecosystem is fully decentralized, anyone can provide an app store via any webserver on any network, including most cloud services.

We are happy to answer specific concerns as well.

h3.Will the people most at risk actually use this or will only technically advanced users adopt the platform?

We are confident that this project will give FDroid a comparable user experience to the popular app stores, and we will include apps that are popular in the target markets. So the barriers to entry will be as low as possible. We partner with other organizations in order to reach key networks of users, as well as get usability feedback from them. Trainers will see the biggest improvements since it will provide a reliable app and media distribution method that works without internet on the phone they are already using. We work with many trainers and will continue to address the usability issues they encounter. Trainers are working directly with the people most at risk.

h3.Will malicious apps or implementations of the platform undermine trust within the ecosystem of FDroid and limit the effectiveness/impact of the approach?

The short answer is: yes, and we are doing everything possible to combat that. As with any app store including Xiaomi and Google Play, if it has too much malware in it, then users will stop trusting it. Like Baidu, Cafe Bazaar, and Amazon: users have to install the F-Droid app store themselves. That is inherint issue for all third party app stores. Combating this problem is mostly a question of user behavior: people should only install software from trusted sources. Therefore, we will make it easiest for users to avoid the dangerous behaviors and use trustworth paths to installs. Next, there must be methods to verify what someone has is from the original source, and that there isn't malware. This project also provides multiple methods for verifying the source of files and integrates with malware databases like VirusTotal and AndroTotal.

We chose to base this work off of F-Droid because it is built by a strong community with a good track record for security and privacy. F-Droid provides the strongest security of any Android app store, including Google Play. Including the f-droid.org central app repository is also important to provide a central source of trust information for things distributed via other app stores and/or peer-to-peer via app swapping. We will also make F-Droid use the

developer's original APK files whenever possible, verified via a reproducible build process, so that the APKs distributed via F-Droid match exactly the APKs distributed in Google Play. There are already a few apps in F-Droid that are included via this process.

Users can choose additional app stores based on their trust of the organization, so we build upon existing user knowledge of how to verify trusted. For example, we host our FDroid app store on our regular website, so people can easily see that the app store comes from the same organization. So anyone can see that the HTTPS is green in the browser and the URL is the same as what is listed in Google Play before adding our app store to F-Droid on their phone.

Lastly, there is a large, active technical community around F-Droid that takes pride in its trustworthiness. They are already serving to monitor and police the apps that are included. Part of the standard process for including any new app in f-droid.org includes an audit for tracking and non-free software.

For more on this topic, see the "Weaknesses and Challenges" section of the Proposal Narrative.

h3.How exactly will the reproducible builds work?

An app's build process is described in a metadata file, then that build process is then run in a virtual machine (VM) which is setup based on that build metadata. That provides a reproducible setup for the build. After the build is complete, it is compared to the developer's original APK signature. The mechanism is simple: copy the developer's APK signature into the FDroid-built APK and then run standard JAR/APK signature verification process.

It turns out that running reproducible builds has a lot in common with setting up an app for "Continuous Integration" (CI), i.e. automated test builds provided by services like Jenkins, Travis CI, etc. Continuous Integration is already considered a best practice, and it widely used in app development. To set up a build for Continuous Integration, the whole setup and build process is written into a script. That same script can then be used to run a build on multiple machines and/or virtual machines. To support reproducible builds, the tools just compare the output of those repeated builds and provide feedback on what is different between builds.

h3.What confidence should we have that these f-droid systems can resist attack?

Guardian Project is quite confident that the F-Droid systems are very resistant to attack, so much so that we have based our distribution on F-Droid when Google Play is not available. F-Droid has been running for almost 5 years, and there have not been any successful attacks against it's infrastructure. The metadata signing is modeled after Debian, a well proven example security-wise. And F-Droid has been publicly audited. There were no egregious issues found, and all issues found in that audit have been fixed. We plan a full penetration test of the core infrastructure to further prove attack resistance.

h3.What security mechanisms will exist for creating own app repositories?

What exists currently:

- all fdroid repositories use signed metadata
- the client only accepts metadata that is signed
- a repository signing key can be provided or automatically generated
- the repository is assembled on a private computer, then pushed to any number of public servers
- the tools are designed to automate a fully offline signing process
- the signing key can be hosted on a hardware security module, aka smart card
- each app can also be automatically GPG-signed

What we will add:

- app store curating tools will verify the content based on the signed metadata of other app repositories
- apps will be checked for malware using services like VirusTotal, AndroTotal, Lookout, etc.

h3.Doesn’t app signing already prevent the TLS MITM attack?

Android app signing does not protect against a MITM attack when the app is being installed for the first time. The first time an Android app is installed, there is no check of whether the app signer is trusted, it is entirely "trust on first use". So if an attacker can reliably recognize first installs from the network traffic, then the attacker can automatically insert malware when the requested app is not already installed. Also, Google Play and other app stores will accept commands from the server to install apps, giving a MITM attacker a method of installing custom malware that is always a first install.

Basically all app stores except FDroid require user data when downloading apps, giving a MITM attacker a means of tracking who has which apps installed. That means that the security of installing apps for the first time is entirely reliant on the TLS connection. Google has done a good job with TLS, but even still, there has been vulnerabilities. Other app stores have a much worse track record, including many like Baidu and CafeBazaar that lack even HTTPS/TLS connections.

Additionally, there are a number of well known exploits in the app signing, such as Master Key. These are fixed in the very latest Android releases, but most Android devices are running older releases. For example, the most affordable Android devices being sold now are still running Android 2.3, which was released in 2010 and last patched in 2011.

h3.Is it a good idea to centralize trust into a single authority which keeps signing keys online rather than have them distributed?

This question reads like a rhetorical question since the answer is: of course not. Let me unpack it so I can give a more sensible answer.

We will make it easy for internet freedom developers to use a secure release process without having to devote significant resources. The FDroid eco-system already provides an example of best practices here: fully offline signing, reproducible builds, signed metadata, hardware security modules for signing keys, HTTPS pinning, support for developer's own signature on release APKs, etc. The app repositories at https://f-droid.org and https://guardianproject.info/fdroid both use fully offline signing for APKs and repository metadata. This security model is available in the FDroid tools now, what needs to happen is to make this easy for any organization or app developer to use.

Decentralized APK signing is a big feature for sure, and FDroid supports APKs signed by the developers' keys via the reproducible build process. FDroid's central signing key management is still a valuable service, since it is a fully offline process. Many developers use this service even for apps that are uploaded to Google Play because they do not have the time or skill to setup and manage a fully offline signing process.

h3.Does F-Droid pull code from GitHub and build it?

Yes, F-Droid pulls the source code from a git, mercurial, or subversion repository. All apps included in the f-droid.org repository have been built from a source repository, not a source tarball or other package.

h3.What makes it so hard for an attacker to install their own fake FDroid, via sharing, or spear-phishing, or manufacturer compromise?

The risk of a fake FDroid used for spearphishing are real, as with any software that is installed via direct link. Since manufacturers do not yet include FDroid, all users will have to get it installed. That does then provide an avenue for spearphishing attacks. Since our target audience is already installing app stores, we are not encouraging this as a new behavior. In China, there are many popular app stores that people add. Xiaomi does have a built-in app store, but Xiaomi is still a small minority of devices sold and used in China. In Iran, Cafe Bazaar is a popular app store and it still must also be installed by end users. Amazon and Aptoide are other popular app stores that also must be installed by the users.

Spearphishing attacks rely on human behavior to work, and human behavior is difficult to change with software alone. We are certainly not encouraging users to learn new, risky behaviors, like installing software from random links. Instead, this project aims to replace that widespread practice for the hundreds of millions of Android users who do not have access to Google Play. By moving users to FDroid secure distribution channels, this project makes the safer behaviors easier than the risky behaviors, and thereby greatly reduces the opportunities for spearphishing.

The more users understand a process, the less likely that process can be exploited. This project includes a lot of work to keep the install as simple as possible:

- simple official download link: https://f-droid.org/FDroid.apk
- includes alternate domains like http://fdroid.org
- only HTTPS downloads allowed from f-droid.org
- installation via direct, peer-to-peer, local connection (e.g. Bluetooth, local WiFi, etc) that are difficult to mass monitor and exploit
- get FDroid included in ROMs by default (currently Replicant and Mike Perry's/Patrick Connolly's Mission Impossible).

Also, by pushing users to get apps via FDroid "swapping", that works only on nearby connections (Bluetooth, WiFi hotspot, etc), we can steer users away from ever directly downloading APKs at all. As for the attacker sharing directly a fake version, this would be an extremely limited attack requiring lots of chutzpah because of the way that the app swapping is structured. It is only ever something that people do face-to-face, since both "swappers" must opt in, and it uses Bluetooth and the local WiFi network to transmit the data.

For technical users and organizations that setup their own Android devices, FDroid has an update.zip that is flashed onto an Android device using the same mechanism that people add Google Play to ROMs like CyanogenMod. This allows a trusted partner to do bulk device setup with FDroid as a system app, leaving "Unknown Sources" disabled.

Verification is also important, so FDroid provides mechanisms for technical people to verify that they have gotten the original version. For example, all FDroid APKs are also GPG signed: (e.g. https://f-droid.org/FDroid.apk and https://f-droid.org/FDroid.apk.asc). The APK signing key information is also published so the APK signature can be verified, for example, using our utility app Checkey.

If an attacker compromised a manufacturer, then why would that attacker bother with making a fake FDroid? This attacker would have transparent, root access to the device, including FDroid, by inserting their own malware wherever it can be best hidden (e.g. in Android itself). This attacker could also install a compromised version of Google Play.

h3.Does F-Droid verify that the developer’s signature matches their own?

With the reproducible process, FDroid builds an APK from source, then downloads the developer's signed release APK. It copies the signature from the developer's APK into the FDroid-built APK then runs a standard jarsigner verification. So the developer's signature is the only one that is used. We are working towards creating reliable methods for reproducing the exact same file, bit-for-bit (i.e. APKs with the exact same hash). We are able to do this manually, so with time to work on it, it is possible to automate.

h3.Could an attacker compromise the F-Droid build systems?

Any system can be compromised, so we plan accordingly. The current state of the f-droid.org infrastructure is solid:

- separate release build machine with limited internet access
- builds are run in a disposable VM instance
- dedicated hardware for entirely offline release signing process
- core signing keys stored on Hardware Security Modules

Verification is also essential for strong security. We are currently almost to alpha for a "verification server" that serves as pre-packaged, automated auditor by running all of the builds that f-droid.org does, then comparing the results to make sure they match. We would love the opportunity for a full penetration test of this infrastructure as part of this proposal.

Signing the Local APK Index

Threat model

Protect against:

- MITM between Alice and Bob
- Package modified by Mallory and supplied to Bob who is gullible. How does Alice detect this?

Overview

The fdroid client (presently) only checks one signature on the index.jar. When you first configure the repo if it finds a pubkey attribute on the index.xml's repo XML then it will subsequently compare that cached attribute value making sure it's the Signature on the [[index.jar]]. it doesn't care how many signatures there are as long as the one specified in the index.xml from repo install is there. there isn't any support for more than one pubkey attribute in the XML

- the sig is a hash of the PKCS7 bytes

Signing using the local HTTPS key

Bazaar will generate a key for use in local HTTPS connections and for signing the [[index.jar]] that is generated locally.

Implied signature over OTRDATA

When data is transmitted over a secure / authenticated channel, such as OTRDATA the sender implicitly confirms that the repository as a whole and the files are as they are stored on the source (Bob's)system. This doesn't mean that the APKs were not modified before they reached Bob.

Signing using the user's OTR key

Another idea is to have ChatSecure somehow sign the index.jar using the user's existing and trusted OTR private key. This would be useful because it would then have existing trust relationships, but it would be tricky to get Bazaar talking to ChatSecure in the right way.

There are a few ideas on how to do this:

- make ChatSecure be a Certificate Authority and sign Certificate Signing Requests
- make ChatSecure sign arbitrary files from other apps
- make ChatSecure generate secret key, then sign them and pass them to an app

Swap over bluetooth (in development)

Overview

Most of the bluetooth *connection* stuff is stock standard stuff from the Android SDK. However, once we move beyond connecting, and to *requesting* and *serving* information over Bluetooth, it becomes more specific. Most bluetooth stuff (communication related, not UI related) should be in org.fdroid.fdroid.net.bluetooth.* and the .httpish package below that.

Communication protocol

I figured there was going to be a bit of arbitrary information exchange, so I went ahead and made something which works essentially like HTTP (hence the .httpish package name). Each request will begin with a request type (GET or HEAD) and then HTTP headers. The server responses have headers and optionally a body.

The BluetoothServerSocket is encapsulated in a BluetoothServer, which continually listens for incoming connections, then handles them in a BluetoothServer.Connection object.

The BluetoothServer.Connection object will continually monitor its socket for incoming info, and parse them as if they were HTTP requests (i.e. request method, headers and body).

Bluetooth swap workflow

(Until UX stuff comes through from carrie, I've made it pretty generic, purely so that I can test the protocol stuff, so this is subject to change)

1. User starts a swap
2. Selects apps to swap
3. Presented with the wifi connection screen (and a button to use Bluetooth)
4. When pressing the use bluetooth button, F-Droid:
   - Asks to enable bluetooth (if not already enabled)
   - Asks to make device discoverable (if not already discoverable <- this is probably doesn't need to happen every time, maybe should be initiated manually)
   - Shows the name and address of your Bluetooth device
   - Shows a list of Bonded bluetooth devices (not related to whether they are in range, just whether they have previously been Bonded with the device or not)
   - Offers a scan button in the menu, which will populate the list of devices as more are found

Getting the code and contributing

The code currently lives in a branch called bluetooth-swap in the main F-Droid repo on GitLab.

I'll try to ensure that I (pserwylo) don't push directly to that, but rather create MR's so that others can review code if necessary. In the near future, I'm also going to create a Milestone in the issue tracker on gitlab to deal with bluetooth swap, so that we can create issues and assign them to that milestone, and see progress towards a stable implementation.

Outstanding tasks

Currently, you are able to start a Bluetooth swap server, which is able to listen for incoming client connections, parse any incoming data, forward it to a local proxy webserver which is the same thing that handles WiFi swapping, and serves relevant files from the webroot in response.

The client side still needs much more work though, in that the infrastructure which downloads icons, repo indexes, and .apk files does not yet know about how to do this over Bluetooth.

Connect to a bluetooth swap

This is distinct from creating a swap (which is already implemented). We need to be able to create a new repo on the swap client, so that the UpdateService is able to pull down an index from that repo and populate the database with apps from that repo.
There are two ways in which this happens with the WiFi swap right now.

- Scan the QR code which is displayed
- Over NFC
- Visit the displayed URL, and then click the link in the resulting web page

All three of them have the same effect, which is to send an intent to F-Droid which tells it to add a repo (e.g. "http://10.0.0.1:8888/fdroid/repo?swap=1&fingerprint=ABC"). Thus, they all end up in the same location, but use a different method to get there.
For Bluetooth, it could happen in a similar way, but this time the intent would look like, e.g. "bluetooth://aa:bb:cc:00:11:22/?fingerprint=ABC" (probably doesn't even need "swap=1", at least not until someone builds an fdroidserver repo which works over Bluetooth somehow):

- Scanning a QR code
- Over NFC

We can't get people to visit a URL over the browser, because that will depend on the devices being on the same TCP/IP network. But if they are using Bluetooth, than this is not the case. That is, even though I've mimicked HTTP with my bluetooth protocol (see above), it is not for the purpose of

interoperating with existing HTTP clients.

The end goal is to have the clients F-Droid end up here, which will require modification of the NewRepoConfig class so that it doesn't assume HTTP, and also have the manifest register interest in the relevant bluetooth URLs.

Decide more on the UX (not exactly UI at this point) for Bluetooth swap

The server is relatively straightforward.

- Person X thinks "I want to share my apps with Person Y".
- Person X initiates swap through Main Menu -> Start Swap.
- They select apps they want to swap.
- Proceed to the "Select WiFi network" screen.
- Indicate somehow that they don't want WiFi, but want to use Bluetooth (e.g. through a button on that screen)
- Ensures they have bluetooth running.
- Starts a bluetooth swap server.

I'm presuming from here, that we do the same as with WiFi, which is to offer NFC, and then display a QR Code with the relevant "bluetooth://.../" URL encoded in it.
The things which are different from wifi though, are:

- If they don't have F-Droid installed, but use a barcode scanner to scan the QR Code, how do you bootstrap the process?
- If they don't have a barcode scanner, how can we get them to connect? The WiFi swap asks them to enter a URL in the browser.

Make BluetoothDownloader work for client

Currently, you can instantiate a BluetoothDownloader, and actually send http(ish) requests to a F-Droid app with a running BluetoothServer running. An example of this can be seen in some test code here.

However, notice that that code requires a BluetoothConnection to be passed to it?

The typical way in which F-Droid connects to a remote repository to download either an icon, an index, or indeed a .apk is to use the DownloaderFactory to create the relevant downloader for a URL. The API for this class does not know about BluetoothConnections, nor should it need to (because the vast majority of downloads will be from a HTTP repo).
To solve this, we need one of two things:

- A way for a BluetoothConnection to be stored globally somewhere
- The ability to construct a BluetoothConnection from a url such as "bluetooth://aa:bb:cc:00:11:22/fdroid/repo/index.jar" using the MAC address in the URL.

The second would be more resiliant to Bluetooth turning off and on again, because between requests for files, it wont try and maintain an open socket, whereas the global connection would. To make the global connection (option 1) resiliant in this way, it will essentially have to work like 2. anyway, and be able to reconnect when required. However, I'm not sure about the other implications of option 2, such as having to prompt the user to connect to bluetooth because F-Droid needs to download an icon, or the performance overhead of continually creating new connections, e.g. for downloading several icons to display in the swap app list.

this page moved to here: [[trustedintents:Wiki|Trusted Intents Wiki]]

User Stories

José has an Android phone, but there is no 3G Internet in his region. He can, however, get on WiFi at his school, or when he walks by a local hotel or café. He uses these occasions to download new apps and upgrade the ones he has. Once he gets to school, he wants to share these with all of his friends.

Jane is traveling to an event to help train journalists in using smartphones. She has to buy the smartphones in the local area, as she could not bring them all with her; then she needs to quickly install a set of apps on each phone. She doesn't have much time and needs to make sure it's done securely.

Juniper lives in a country where Google Play is blocked. The only way to get apps is through a few local sketchy app stores, or by downloading APK files off of message boards where malware is also sometimes posted. She knows some of her friends use VPNs, and wishes she could swap apps with them.

Linda is technically sophisticated and is able to access unfiltered Internet through [Tor | Psiphon | Lantern | etc.]. She keeps the apps on her device up-to-date using her unfiltered Internet connection. José lacks the technical ability to work around the filters himself and wishes he could update his apps from Linda's device.

A developer in Iran has created some apps and wants to distribute them to as wide an audience as possible. He knows that some apps have been blocked from Cafe Bazaar, so he wants to make sure that there are as ways as possible for people to get this apps. He sets up the FDroid tools to manage publishing his apps to Cafe Bazaar, Google Play, f-droid.org, and a couple "collateral freedom" services like github and Amazon S3. He then runs two simple commands to update his app repository and publish it to all of the app stores. He makes his releases using the FDroid reproducible build and hardened signing process. Even though other local developers have found Finfisher on their computers, he feels confident that his release process has not been infiltrated.

StoryMaker is making a targeted campaign that delivers everything that a user needs to make a video, including tutorials and guidelines for that specific campaign, and a channel to publish videos to others. They need everything delivered to users' devices with a single download and single install process. It must also automatically stay updated. They use a custom FDroid installer bundle that includes everything needed in a single download link. This same link will also direct users who already have StoryMaker to the campaign without making them download everything again. For new users, FDroid is first installed, and it then downloads a standard StoryMaker release with a trigger for it to get the campaign materials. Since FDroid is now installed, it will automatically get StoryMaker updates while also providing a full app store to users. Users publish their videos from StoryMaker, which also adds them to this campaign's FDroid media channel, so they can easily get the videos that others are making.

A human rights organization produces videos and ebooks of training materials and important information, they also have their own mobile app. In their trainings, they use apps from Tor and Guardian Project. A trainer sets up an app store that includes their app and all their videos and publications, and set it up to automatically include the most recent updates from Tor and Guardian Project. At trainings, students get an FDroid bundle from the local app store on the trainer's phone. The installation process lets them click through to install Orbot and ChatSecure, and FDroid is set up with direct access to download and share all of the videos and ebooks.
A developer of circumvention and anonymity tools produces Android apps for journalists and human rights defenders who are targeted by multiple state actors. Having access to their app signing key means that an attacker could deliver targeted malware via peer-to-peer and internet app repositories. Getting access to their app repo signing key means malware can be injected into the official app repo.

- http://threatpost.com/new-version-of-destover-malware-signed-by-stolen-sony-certificate/109777

State Security Agencies target developers to install backdoors in all users:

- https://firstlook.org/theintercept/document/2015/03/10/strawhorse-attacking-macos-ios-software-development-kit/
- https://firstlook.org/theintercept/2015/03/10/ispy-cia-campaign-steal-apples-secrets/

Files

BootstrapDecisionTree.png	144 KB	02/20/2014	pd0x