Bitcoin P2P

Breaking Bitcoin

Principaux enseignements

- Depuis la fin de la guerre des blocs, le principal sujet de désaccord entre bitcoiners semble être les t-shirts
- En dépit des progrès réalisés sur la confidentialité en particulier avec la démocratisation de CoinJoin, les outils de désanonymisation restent extrêmement puissants (est-ce que Schnorr/Taproot nous donnera définitivement l'avantage ?)
- Dans Bitcoin les dangers viennent de partout, sauf de Bitcoin (technologie web, dépendances linux, l'humain bien sûr)
- Voler la seed dans un Trezor en quelques secondes (https://youtu.be/DqhxPWsJFZE?t=10718)
- Amsterdam n'est pas un très bon choix pour une conférence technique (trop toxique)

Pour tout savoir

https://youtu.be/DKOG0BQMmmg

https://docs.google.com/document/d/1xiMDShdkaNHC16icEU5zlu0jBZ8SNn3BCHTB2d8gzHk/edit

Don’t trust, verify(?)

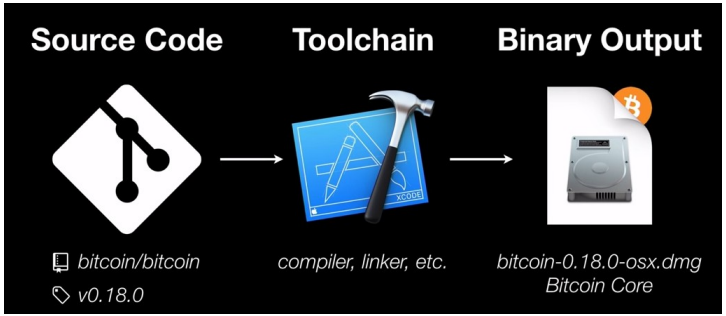
- C’est à la fois la pierre angulaire de Bitcoin, et quelque chose de plutôt encombrant dont on se passerait bien
- Qui n’a jamais entendu l’argument “mais pourtant tu fais confiance aux développeurs de Bitcoin Core/à ton compileur/à ton hardware” ?
- C’est pourquoi j’ai décidé de m’attarder un peu sur la présentation de Carl Dong, qui concernait le travail en cours sur Bitcoin Core pour diminuer la confiance nécessaire à différents niveaux logiciels, et permet de revenir sur les différentes couches de confiance qu’on ne soupçonne pas forcément

Carl Dong, Bitcoin Build System Security

https://youtu.be/DKOG0BQMmmg?t=19836

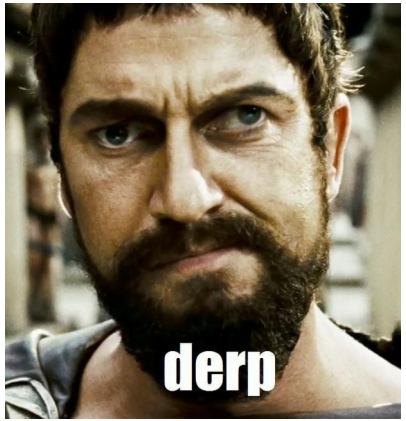
- Le programme (Bitcoin Core, ou autre) que l'utilisateur télécharge et utilise sur son ordinateur est un *fichier binaire*
- Ce fichier doit être généré à partir du *code source*, un processus appelé *build*
- Cette opération nécessite d'autres logiciels regroupés dans un *toolchain*

L'utilisateur peut être victime de différentes attaques qui visent in fine à injecter un code malicieux dans le fichier binaire qu'il va exécuter.



#1 Je fais confiance à tout ce que je télécharge

- Un attaquant peut tout simplement dupliquer le code source de Bitcoin Core, le modifier en introduisant des vulnérabilités, créer un nouveau fichier exécutable “empoisonné” et tenter de le faire installer à l’utilisateur
- Cette attaque repose donc principalement sur l’ingénierie sociale (phishing…)
- Contre-mesure: vérifier la signature du fichier grâce à la clé GPG fournie par le développeur




```
gpg --recv-keys 01EA5486DE18A882D4C2684590C8019E36C2E964

gpg --verify SHA256SUMS.asc

sha256sum --ignore-missing --check SHA256SUMS.asc
```

#2 Je fais confiance au développeur

- Personne ne peut vérifier que le fichier sur bitcoincore.org et signé par Wladimir Van der Laan est identique au code source disponible sur Github
- Vérifier la signature n’est alors plus suffisant, car l’attaquant est légitime !
- Une solution serait de permettre à l’utilisateur de compiler le code source lui-même, et de comparer le fichier obtenu avec celui compilé par d’autres, mais ce n’est pas si simple car compiler le même code source sur des machines différentes produit des fichiers différents !
- Des contributeurs de Bitcoin Core ont inventé gitian, qui permet de réaliser des builds *reproductibles* en standardisant l’environnement de compilation (machine virtuelle) et en rendant le processus totalement déterministe (les dépendances sont compilées et signées)

Gitian

- Procédure permettant de générer des *reproducible builds* inventée par des contributeurs de Bitcoin Core vers 2013.
- D’autres projets s’y sont rapidement intéressés, comme Tor
- Grâce à cette initiative, >90% des packages Debian sont aujourd’hui reproductibles

https://blog.torproject.org/deterministic-builds-part-one-cyberwar-and-global-compromise

#3 Je fais confiance à Ubuntu

- Gitian doit encore faire confiance aux packages suivants :

Le pire cauchemar d’un développeur

- Mick Stute était engagé pour auditer un logiciel en apparence très simple, mais qui présentait un comportement étrange (affichage subliminal de messages racistes)
- Même en corrigeant le code source, lors de la compilation ce dernier était régénéré sous sa forme malicieuse, et le fichier binaire compilé sur cette base !
- Après 15 jours de recherche, Mick a fini par comprendre que le problème venait de la toolchain, qui avait été modifiée pour empoisonner le programme lors du build.
- Pire: si on essayait de compiler la toolchain à nouveau pour la nettoyer, elle était capable de le détecter et d’empoisonner la nouvelle toolchain !

https://www.quora.com/What-is-a-coders-worst-nightmare

#4 Je ne fais confiance à (presque) personne

- Carl Dong a proposé d’utiliser Guix afin de :
  - minimiser le nombre de dépendances auxquels faire confiance
  - rendre auditable le processus de build
- Guix est un package manager utilisable sur toutes distributions de Linux, et la proposition de Carl est de s’en servir comme d’un environnement de build “contenairisé” dans lequel le toolchain est compilé à nouveau