Binbloom v2

Ceci est une (r)évolution

Damien Cauquil

Quarkslab

/me

- Security researcher @Quarkslab
- Sécurité des systèmes embarqués

/me

- Security researcher @Quarkslab
- Sécurité des systèmes embarqués
- Non, ceci n'est pas un talk sur BLE.

Plan

Les adresses de base des firmwares et comment les trouver

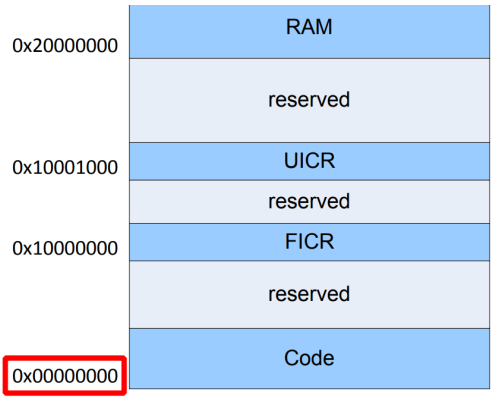
Comment ça fonctionne
- Les points d'intérêts
- Les pointeurs

Testez binbloom v2
- Télécharger, tester, contribuer

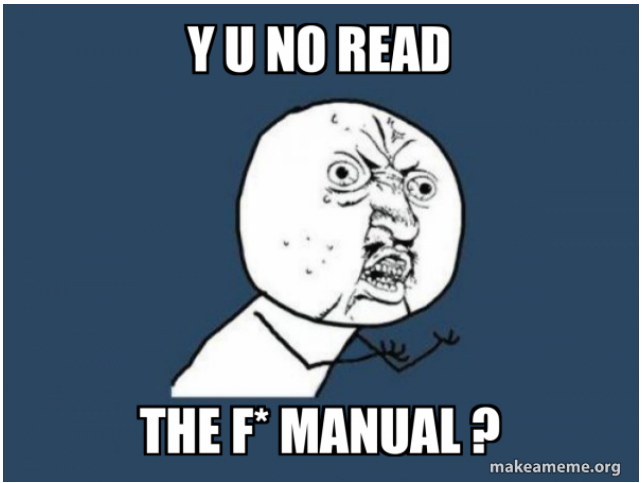
Les adresses de base des firmwares et comment les trouver

Quarkslab

C’est quoi une adresse de base ?



RTFM



Pourquoi la chercher ?

- **Absence** de documentation technique ;

Pourquoi la chercher ?

- **Absence** de documentation technique ;
- **Portion de code** d'un firmware.

Usual Suspects

basefind.cpp & basefind.py

- **Bruteforce** de l'adresse de base ;
- **32-bit** seulement;
- ne se basent que sur les **chaînes de caractères**.

`https://github.com/mncoppola/ws30/blob/master/basefind.cpp`
`https://github.com/mncoppola/ws30/blob/master/basefind.py`

rbasefind

- Portage en Rust de *basefind.cpp*;
- **32-bit** seulement (encore);
- ne se base que sur les **chaînes de caractères**;
- **Multi-threadé** et très performant !

`https://github.com/sgayou/rbasefind`

basefind2

- Pas de bruteforce, tente de **déduire les candidats des différences d'offsets** des chaînes de caractères;
- **32-bit** seulement (encore);
- ne se base que sur les **chaînes de caractères**;

`https://github.com/soyersoyer/basefind2`

Binbloom v1

- Première version développée par **Guillaume ”PapaZours” Heilles**;
- Implémentation en **C** relativement performante;
- **Dépendante d’une liste de fonctions** identifiées par un outil externe (IDA);
- Implémente la **détection d’endianness** et la **recherche de structures UDS**

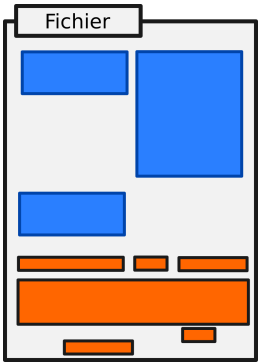
binbloom reloaded (version 2)

Comment ça fonctionne

Comment ça fonctionne

- On identifie des **points d'intérêts** ;
- On cherche des **pointeurs** qui pointent sur **un maximum** de points d'intérêt.

Les points d'intérêts

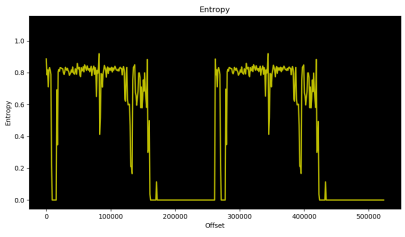


- **Chaînes de caractères** remarquables ;
- **Tableaux d'entiers** relativement proches ;
- Une portion de code clairement identifée (**fonction**) ;
- Un point d'intérêt a grande chance d'être **référencé** dans le code, via un ou plusieurs **pointeurs**.

Les pointeurs

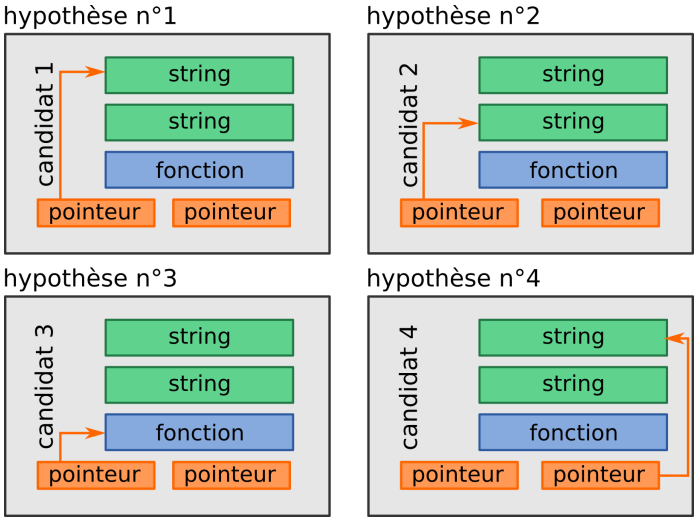
- Généralement une adresse pointant sur un point d'intérêt ;
- **Impossible** à identifier sans connaître l'adresse de base !
- Quelquefois **présents dans des tableaux**, assimilés à des tableaux d'entiers.

Phase 1: mesure d'entropie et classification



- Classification: **code** ou **données** ;
- **Seuils génériques** déterminés par expérimentation ;
- Loin d'être idéal !
- Permet de **cibler la recherche de points d'intérêts** (dans les données).

Phase 2: génération des candidats



Phase 3: Evaluation des candidats

Pour chaque adresse candidate:

- **On compte le nombre de pointeurs** qui mènent à des points d'intérêts ;
- **On pondère** si nécessaire (tableau de pointeurs valides);

On retourne enfin les **adresses ayant les meilleurs scores**.

Fonctionnalités additionnelles

- Détection de l'***endianness*** ;
- Détection de **base de données *UDS*** ;
- Possibilité de renseigner une **liste d'adresses de fonctions connues**.

Améliorations possibles

- **Détection automatique de l'architecture**
  - amélioration de la classification code/données
  - détection de prologues de fonctions
- **Annotation automatique** des structures dans les désassembleurs habituels

En résumé

- Supporte les architectures **32 et 64 bits** ;
- Pas de bruteforce de l'adresse de base mais une **déduction** ;
- Permet de trouver une **liste d'adresses de base probables** ;
- Peut **détecter l'endianness**

Testez binbloom v2 !

Quarkslab

Télécharger, tester, contribuer

`github.com/quarkslab/binbloom`

`https://blog.quarkslab.com/binbloom-blooms-introducing-v2.html`

Thank you !

quarkslab
virtualabs
virtualabs

quarkslab.com