Binary Analysis Tool

Armijn Hemel, MSc
Tjaldur Software Governance Solutions

5 februari, 2016

Over Armijn

- gebruiker van Open Source software sinds 1994
- drs informatica Universiteit Utrecht
- kernteam `gpl-violations.org` van 2005 - mei 2012
- eigenaar Tjaldur Software Governance Solutions

Binary Analysis Tool

Binary Analysis Tool (kortweg: BAT) is een lichtgewicht programma om het analyseren van binaire bestanden te automatiseren.

- inzichtelijker maken van "compliance engineering" door kennis te coderen
- zorgen voor reproduceerbare resultaten
- "lingua franca" voor het analyseren van binaire bestanden

BAT is een generiek raamwerk, maar wordt meestal gebruikt voor "open source compliance". Huidige versie is BAT 24, uitgebracht 27 januari 2016. BAT 25 komt (hopelijk) eind maart.

Binaire bestanden analyseren: waarom? (1)

Software wordt bijna altijd geleverd in binaire vorm (op een apparaat/CD/DVD/flash chip/download/app). Soms is er broncode beschikbaar, die (als je geluk hebt) ook nog overeenkomt

Het verschepen van software als broncode is de *uitzondering*.

Als je binaire bestanden (voorbeeld: een router) verder distribueert/verkoopt dan moet je weten wat er in zit! Je moet daarvoor de binaire bestanden analyseren.

Binaire bestanden analyseren: waarom? (2)

Steeds meer apparaten worden “slim”, maar de fabrikanten zijn dat zelf niet altijd. Er is veel rotzooi op de markt (met al dan niet embedded software) die op allerlei netwerken worden aangesloten en die lek zijn en waar geen beveiligingsupdates voor komen.

Enkele recente voorbeelden:

- HEMA USB-stick
- CyberQ
- Fisher Price Smart Toy Bear

Ouder voorbeeld: `http://www.upnp-hacks.org/`

Nieuwe wetten met betrekking tot het melden van datalekken (van kracht in Nederland, binnenkort in de hele EU) maken het niet-veilig zijn een dure grap.

Wat is dit?

```
00000000  50 4b 03 04 14 00 00 00  08 00 29 52 57 3c fa c0  |PK........)RW<..|
00000010  03 a7 26 9e 16 01 f4 ae  19 01 15 00 00 00 76 31  |..&...........v1|
00000020  2e 31 2e 31 2e 31 37 5f  53 4d 43 5f 61 6c 6c 2e  |.1.1.17_SMC_all.|
00000030  65 78 65 ec 3a 6d 78 53  55 9a f7 26 69 9a 42 ca  |exe.:mxSU..&i.B.|
00000040  0d d0 38 65 69 30 60 50  94 96 56 43 91 98 06 03  |..8ei0‘P..VC....|
00000050  92 18 9f e1 e3 d6 c8 4d  03 f4 03 69 6b b8 a3 88  |.......M...ik...|
00000060  78 2f 83 da 76 c3 a6 d9  6d 7a 37 0f 38 8b 33 ae  |x/..v...mz7.8.3.|
00000070  33 ce d0 89 ee 8a f8 38  ae 3a 88 1f 30 61 c2 52  |3......8.:..0a.R|
00000080  3a ea 33 ac e3 02 0e 3c  b3 38 ea ee e9 a4 ce d4  |:.3....<.8......|
00000090  85 2d 01 0b 77 df f7 dc  f4 03 1c 67 66 9f fd db  |.-..w......gf...|
000000a0  ab 37 f7 9c f7 bc e7 fd  38 e7 bc 5f a7 ac 5c bb  |.7......8.._..\.|
000000b0  8b d1 33 0c 63 80 57 55  19 e6 00 a3 3d 5e e6 cf  |..3.c.WU....=^..|
000000c0  3f 67 e1 9d 72 fd 5b 53  98 d7 8b de 9f 7d 80 5d  |?g..r.[S.....}.]|
000000d0  f1 fe ec fb 22 9b 1e b5  6f d9 fa f0 03 5b 37 3c  |....".o....[7<|
000000e0  64 df b8 61 f3 e6 87 25  fb fd 2d f6 ad f2 66 fb  |d..a...%..-...f.|
000000f0  a6 cd f6 e5 ab 83 f6 87  1e 6e 6e 59 50 5c 3c c9  |.........nnYP\<.|
00000100  91 a7 d1 fc c1 99 4b f6  d7 5e dd 3b f2 5e da f5  |......K..^.;.^..|
00000110  f2 de 6a f8 ae 7e e9 cd  bd f3 e0 9b fa c9 3b 7b  |..j..~........;{|
00000120  17 d2 fe 81 bd 9b e0 fb  eb 5d fb f6 56 52 dc d7  |.........]..VR..|
00000130  f6 7e 1f be 37 ee 7a 73  ef 2d f0 fd af 9f be be  |.~..7.zs.-......|
00000140  77 36 7c ef dd b4 31 82  74 46 64 e4 7d 0c b3 82  |w6|...1.tFd.}...|
00000150  35 30 43 1b fd 9e 31 b9  39 76 32 6b 64 98 2a 96  |50C...1.9v2kd.*.|
00000160  61 9a f4 14 76 a1 1b 7e  2c a8 38 ab 69 6f d1 fa  |a...v..~,.8.io..|
00000170  86 fc 9c 91 2f b3 c7 a0  8d c1 a3 a3 bf 96 7c df  |..../.........|.|
00000180  32 0a b7 8c 5b a3 c8 3d  2c b3 07 1b c7 59 e6 85  |2...[..=,....Y..|
```

Binary analysis

Een binair bestand lijkt op ruis, maar er zit vaak een structuur in, zoals bestandsystemen, gecomprimeerde bestanden, enzovoort, die met enige moeite herkend kunnen worden.

Analysestappen

Stappen die genomen moeten worden om te bepalen of een binair bestand gemaakt is van bepaalde (open source) broncode:

1. extraheer binaire bestanden uit blobs (firmwares, installers, etc.), recursief (indien nodig)
2. extrahereer “identifiers” (strings, functienamen, variabelename, enzovoort) en vergelijk deze met (publiek beschikbare) broncode
3. gebruik andere informatie zoals bestandsnamen, packagedatabases, enzovoort voor extra bewijs

“Ducktyping”

“If it looks like a duck and quacks like a duck, it is probably a duck”

Als je veel strings, functienamen, variabelenamen, enzovoort, uit een binair bestand kan correleren met broncode dan is de kans heel groot dat deze broncode ook is gebruikt.

Vaak is het mogelijk om honderden, zo niet duizenden, strings, functienamen, enzovoort, direct te koppelen aan broncode.

Interne werking van BAT

1. opzoeken van offsets van bekende bestandsystemen, gecomprimeerde bestanden, enzovoort en deze recursief uitpakken en verifiëren
2. elk uitgepakt bestand analyseren
3. rapportage, genereren van plaatjes, ...

Modules

BAT is erg modulair opgebouwd en komt standaard met:

- uitpakken van meer dan 30 bestandsystemen, gecomprimeerde bestanden en mediabestanden
- geavanceerd opzoeken met identifiers
- verificatie van dynamisch gelinkt ELF-bestanden
- Linux kernelmoduleanalyse
- beveiligingsaspecten
- en nog meer

Geavanceerd opzoeken met identifiers

Meest geavanceerde scan in BAT extraheert string literals, functienamen, variablenamen, enzovoort uit *binaire bestanden* en vergelijkt deze met een grote database met string literals, functionamen, enzovoort, geëxtraheerd uit *broncode*.

Database is geen onderdeel van BAT. Een voorgegenereerde database met informatie uit meer dan 200.000 pakketten uit GNU, GNOME, F-Droid, Debian, Samba, Fedora, Linux kernel, enzovoort, is te koop. Deze database is een slordige 200 GiB (SQLite) of 250 GiB (PostgreSQL) groot.

Algoritme is gepubliceerd op MSR 2011 (Mining Software Repositories). Scripts om een database zelf op te bouwen zijn open.

Interne werking

BAT gebruikt een database met data geëxtraheerd uit *broncode*:

- string literals (geëxtraheerd met `xgettext`)
- functienamen, methodenamen, enz. (geëxtraheerd met `ctags`)
- variablenamen (geëxtraheerd met `ctags`)
- licenties (met behulp van Ninka en FOSSology)
- SHA256/SHA1/MD5/CRC32/TLSH checksums, andere metainformatie

Extraheren van informatie uit binaries

Uit elk binair bestand dat nog niet genegeerd wordt (plaatjes, video, audio, enzovoort zijn allemaal niet interessant) worden de string literals gehaald.

Deze string literals worden gebruikt omdat een compiler deze niet weggooit, zelfs niet als de binary gestript wordt.

In veel gevallen kan met wat extra steppen de kwaliteit van de string literals verbeterd worden (minder “false positives”) en kunnen beter resultaten behaald worden. Voorbeeld: voor ELF-bestanden worden alleen bepaalde ELF-secties bekeken.

Score (1)

Elk bestand wordt toegewezen aan een “familie”:

- C (C/C++/QML/etc. + onbekende binaries)
- Java (JDK/Dalvik/Scala/etc.)
- C#
- ActionScript

De reden is dat sommige strings niet belangrijk zijn in één familie, maar heel significant in een andere. Dit werkt meestal erg goed, hoewel in sommige gevallen zoals “language embedding” (denk .NET) dit soms lastig is.

Score (2)

Elke string literal wordt opgezocht in de database. Als een string literal gevonden kan worden dan wordt een score toegewezen aan deze string.

De score voor een unieke string (kan maar in één pakket gevonden worden) is de lengte van de string.

Als de string niet uniek is dan is deze score afhankelijk van in hoeveel pakketten het gevonden kan worden. De string wordt dan toegewezen aan het meest waarschijnlijke pakket.

Uitdagingen

- namen van packages en bestanden is erg belangrijk in BAT. Nieuwe workflow van softwareontwikkeling (DVCS zoals Git) maakt het niet makkelijker.
- “cloning” tussen softwarepakketten: code wordt steeds meer gekopiëerd en het is niet altijd duidelijk wat het origineel is en wat de kopie.

Elk analysepakket (niet alleen BAT) heeft hier last van.

Decompilatie vs vingerafdrukken

BAT decompileert binaire code niet, maar maakt gebruik van vingerafdrukken. Dit werkt goed, mits de broncode beschikbaar is zodat deze geanalyseerd kan worden. Aangezien steeds meer software gebruikt maakt van componenten onder een opensourcelicentie is dit vaak geen probleem.

Het maken van vingerafdrukken is makkelijker dan decompileren en dan broncode vergelijken:

- architectuuronafhankelijk
- snel

maar ook makkelijk te verslaan.

Verslaan van BAT

BAT verslaan is niet moeilijk:

- herschrijven van identifiers
- versleutelen van bestandsystemen

Maar om economische en contractuele redenen is dit vaak niet gewenst:

- hogere kosten (testen, integratie)
- controle door klanten (afgedwongen door contracten)
- hogere boetes in sommige jurisdicties indien het ontdekt wordt

Koppelen met extra informatie

Nadat is uitgevonden welke broncode mogelijk gebruikt is om de binary te maken kan er extra informatie toegevoegd worden.

- licentieinformatie
- beveiligingsinformatie

Beveiligingsinformatie: CVE

Informatie over beveiligingsbugs wordt meestal verspreid via CVEs. Deze zijn vaak incompleet, vermelden niet alle versies van pakketten waar problemen inzitten (of de verkeerde versies) en bevatten ook weinig informatie over o.a. checksums waarmee broncodebestanden geïdentificeerd kunnen worden.

Uitdaging:

- cloning: broncode wordt gekopieerd en niet vermeld in de CVE
- incompleet: niet alle pakketten worden genoemd

Door informatie uit de CVE te koppelen met de database is het mogelijk om uit te vinden waar CVEs incompleet zijn en ook om te zien of een binary een mogelijk bekende bug heeft, beschreven in een CVE. Scripts worden onderdeel van BAT 25.

Beveiligingsinformatie: nieuwe bugs opsporen (1)

Veel softwarebugs worden nooit gerapporteerd in een CVE en blijven dus onontdekt, hoewel ze misschien wel opgelost worden in nieuwere versies van software. Veel bedrijven in de (consumenten)electronica pakken een versie en laten dat vervolgens jaren ongemoeid.

Pro-actief zoeken naar beveiligingsbugs is bijna noodzakelijk.

Beveiligingsinformatie: nieuwe bugs opsporen (2)

Opsporen van beveiligingsfouten in code kan door middel van verschillende scanners die broncode analyseren. Dit is (voor mij) niet altijd een goede oplossing:

- false positives
- soms niet toegestaan om informatie verder te verspreiden

Maar er is hoop! Binnenkort zullen in de BAT-database analyses gedaan met Joern opgenomen worden:
`http://mlsec.org/joern/`

Shellcommando's opsporen in binaries

Veel ontwikkelaars bij ODMs nemen nog wel eens een loopje met veiligheid en voeren direct shellcommando's uit vanuit een binary, met `system()`, waarbij ongecontroleerde invoer als parameter meegegeven wordt. Deze binaries draaien bijna altijd met volledige systeemprivileges op een apparaat.

Het is makkelijk om te zoeken naar aanroepen:

- bekende namen van programma's (zoals `iptables`)
- paden
- wildcards

Andere beveiligingschecks

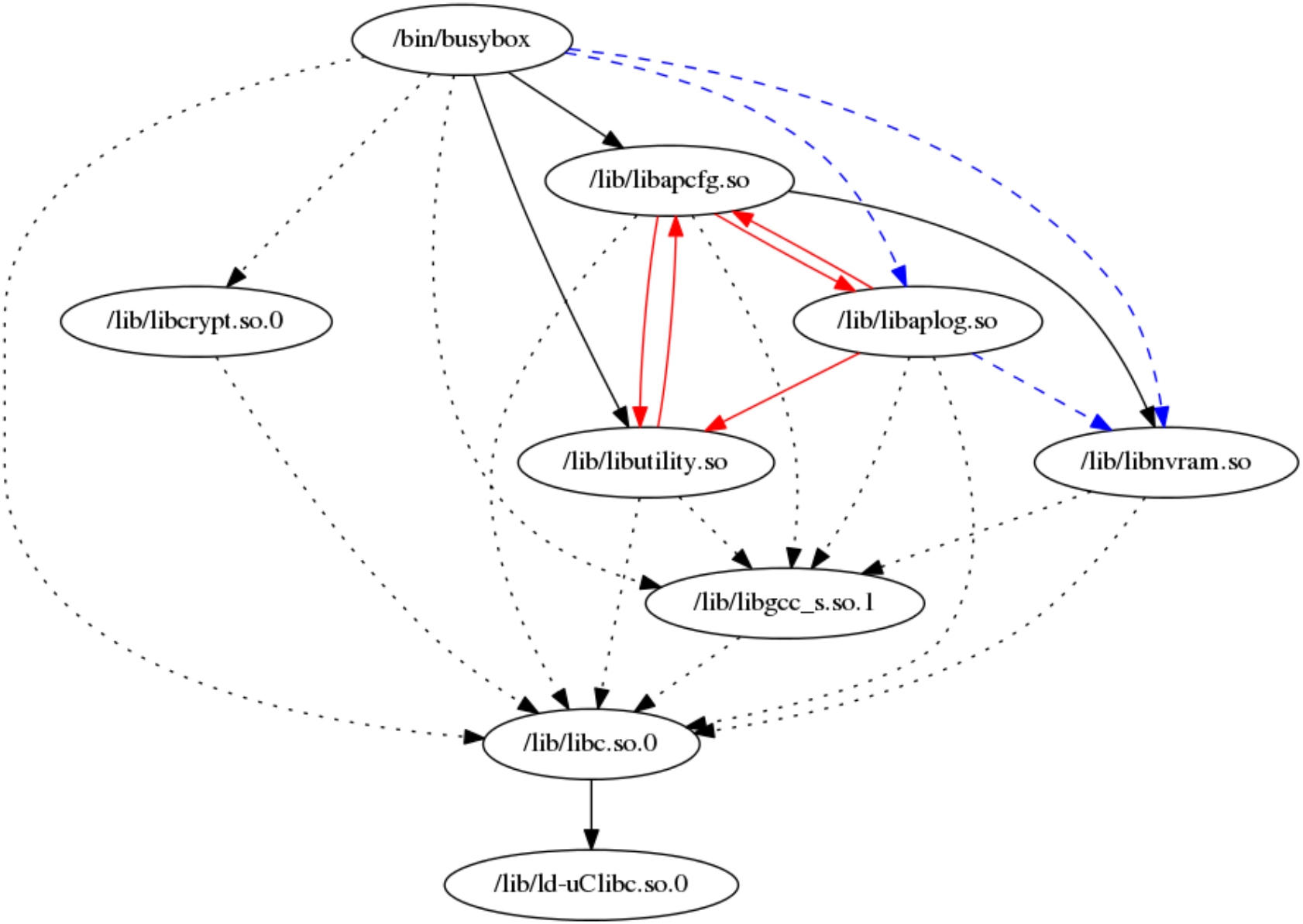
Reeds aanwezig in BAT:

- virusscanner (met ClamAV)
- wachtwoorden kraken (John)
- informatie vinden voor KPA op beveiligde ZIP-bestanden
- opsporen van private keys (hele bestanden)

Toekomstig:

- vergelijken van firmwares (checksums, identifiers, fuzzy hashing)
- opsporen van o.a. private keys binnen bestanden

Overig: ELF dynamic linking



Toekomst

- webservice/opbouwen van firmwarebibliotheek
- meer bestandsystemen (F2FS, FAT, enz.)
- betere ondersteuning voor Windows-binaries
- verbeteren van snelheid

Vragen?

Contact

- armijn@tjaldur.nl
- http://www.tjaldur.nl/
- Binary Analysis Tool: http://www.binaryanalysis.org/