Introducing the Binary Analysis Tool

Armijn Hemel, MSc
Tjaldur Software Governance Solutions

May 27, 2013

About Armijn

- using Open Source software since 1994
- MSc Computer Science from Utrecht University (The Netherlands)
- core team `gpl-violations.org` from 2005 - May 2012
- owner Tjaldur Software Governance Solutions

Binary Analysis Tool

Binary Analysis Tool (or: BAT) is a lightweight tool under an open source license that automates binary analysis.

- demystify compliance engineering by codifying knowledge
- make it easier to have reproducable results
- common language for binary analysis
- only analyses binaries, but draws no legal conclusions

Although BAT is a generic framework for binary analysis my focus is on software license compliance.

Important: a license violation is not a technical issue, but a legal issue. Technical measures are only used to obtain evidence.

Why analyse binaries?

- software is often supplied in binary form by vendors (on a device/CD/DVD/flash chip/download/app). Sometimes source code is supplied and if you're lucky it matches the binary.
- binaries are shipped to customers

Shipping software as source code is the *exception*.

If you pass on the binary software (for example in a product) you have to know what you ship! This means you have to analyse the binaries.

Place of BAT in an open source compliance process

BAT is not meant as a replacement of a source code scanner: if you have all source code there is more than enough information to work with and you don't necessarily need BAT.

BAT is useful when:

- you get binaries, but no source code and want to know what could be in there
- you get binaries and source code, but don't know if binaries and sources match

What’s in this blob?

```
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..|
00000190  5f e8 fe 82 55 fd 0b 1f  90 d3 a0 ff fa e1 05 52  |....U.........R|
```

Binary analysis

A binary usually looks like a blob with random data. Often there is a structure, with embedded file systems or compressed files that can “easily” be recognized.

Analysis steps

Steps to determine if a binary contains a particular source code:

1. extract binary files from blobs (firmwares, installers, etc.) recursively (if needed)
2. extract identifiers (strings, function names, variable names, etcetera) from binary files and compare these to (publicly available) source code
3. use other information like file names, presence of other files, package databases, etcetera, for circumstantial evidence

“Ducktyping”

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

If you can relate many strings, function names, variable names, and so on from a binary file to source code it becomes statistically hard to deny (re)use of a certain software.

Often it is possible to match hundreds or even thousands of strings, function/method names or variable names.

Drawbacks of manual inspection

Checking can be done by hand using standard Linux tools, but there are drawbacks:

- limited by the knowledge of the engineer
- time consuming (so expensive)
- easy to overlook things

So you really want to automate this! BAT can do this for you.

Inner workings of BAT

1. discovery of offsets of known file systems and compressed files and unpacking of found file systems and compressed files (recursively)
2. check each unpacked file, like identifier search
3. reporting, generating pictures, etcetera

BAT modules

BAT is extremely modular and it comes with several modules:

- unpacking over 30 file systems and compressed files
- report on common properties (file type, size, etcetera)
- search for license markers and identifiers
- advanced string identifier search
- dynamic ELF linking verification
- kernel module analysis
- many more

Advanced identifier search/ranking

Most advanced check in BAT extracts string constants, function/method names, variable names, etcetera, from binaries and compares them with a large database of strings and function names extracted from source code:

Currently over 170,000 packages from GNU, GNOME, KDE, Samba, Debian, Savannah, FedoraHosted, Linux kernel, Maven, . . .

Database is not part of BAT, but only available as a subscription, or you can "roll your own".

Algorithm has been published at the Mining Software Repositories 2011 conference and scripts to create the database are open.

Demo with BAT

Running a test would take too long for this presentation, so I will only browse pregenerated results.

Inner workings of BAT ranking

BAT ranking algorithm uses a database where data is extracted from *source code*:

- string constants (using `xgettext`)
- function names (C) and method names (Java) (using `ctags`)
- variable names and Linux kernel symbols (C), field names and class names (Java) (using `ctags`)
- licenses (if enabled) (using Ninka and FOSSology)
- various characteristics of the file (SHA256 checksum, etc.)

Core of algorithm uses string constants, rest of information is used to verify/strengthen the findings.

String extraction from binaries

From each binary that has not yet been discarded (graphics, video, audio, resources files and text files are not interesting) string constants are extracted.

String constants are used because they are not discarded by the compiler.

Some preprocessing steps can be used to increase quality of the strings extracted (to avoid false positives and get better scan results).

Scoring (1)

Each binary file is sorted into a family of languages:

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

Reason is that strings that are very insignficant in one family could be very significant in another and vice versa.

Drawback: language embedding (specifically .NET) is at odds with this. For most systems (Java, embedded Linux) this is much less of an issue.

Scoring (2)

Each string constant is compared to the database. If a string can be matched a score is assigned to that string.

The score for a unique string (single package) is the length of the string.

If it is not unique the score very rapidly drops depending on in how many different packages it can be found.

If there is *cloning* the string is assigned to a package using an algorithm that picks the most promising package.

Database challenges

There are challenges in creating a database:

- package names and file names are very important in BAT. DVCS like Git make software development more fluid.
- cloning of software between packages.

These challenges are not exclusive to BAT.

Cloning

Sometimes the wrong package will be detected, but this reflects how open source works!

- software reuse: code is “cloned” between packages. Software reuse is actually good! Some packages are forked and incoporated into others (in Java more so than in C code).
- packages are renamed for various reasons. For example in Debian: `httpd` is renamed to `apache2`, Firefox is Iceweasel, and so on.

BAT tries to take alternatives and “cloning” into account.

Database quality

Results of scanning are dependent on quality of the database: if only BusyBox is included everything will look like BusyBox.
Making a good database is not easy:

- What to include?
- What to exclude?
- When is a package a new package?

Other functionality in BAT

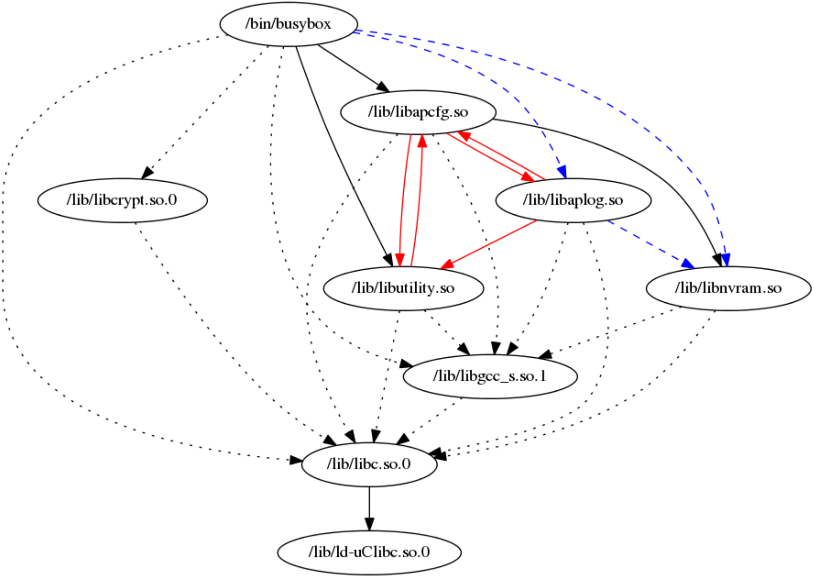
- finding duplicate files in firmwares
- finding leftover kernel modules (mismatching kernel version numbers)
- ELF dynamic linking dependency inspection
- and more

ELF dynamic linking

Actual license of a binary is only determined at *run time*. This is a largely unresearched area.

BAT can give more information about how ELF binary files interact.

ELF dynamic linking dependency graph



Upcoming functionality in BAT

- better GUI
- deployment as a webservice
- more file systems: vendors often use non-standard versions of file systems. Right now BAT supports 14 different flavours of squashfs and I *know* there are more.

Questions?

Contact

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