Code similarity analysis with r2diaphora

cybersecurity.att.com/blogs/labs-research/code-similarity-analysis-with-r2diaphora

1. AT&T Cybersecurity
2. Blog

October 27, 2021 | Fernando Dominguez

Executive summary

Binary diffing, a technique for comparing binaries, can be a powerful tool to facilitate malware analysis and perform malware family attribution. This blog post describes how AT&T Alien Labs is leveraging binary diffing and code analysis to reduce reverse-engineering time and generate threat intelligence.

Using binary diffing for analysis is particularly effective in the IoT malware world, as most malware threats are variants of open-source malware families produced by a wide range of threat actors. Generating and maintaining static signatures for variations on IoT malware is tedious, as the assembly code often changes across variants and architectures and text strings are subject to modification. For this reason, AT&T Alien Labs created a new open-source tool, r2diaphora, to port Diaphora as a plugin for Radare2, and included some use cases in this blog.

What is binary diffing?

Binary diffing (or program diffing) is a process where two files are compared at instruction level, looking for differences in code. Threat actors can easily transform the assembly code for a program without modifying its actual behaviour, so the typical “line-by-line” diffing is not good enough when looking at malware - a more advanced approach is needed.

There are several binary diffing tools publicly available, such as Diaphora, BinDiff, and DarunGrim. Alien Labs is using Diaphora, as we believe it is the most advanced of all the available options. Furthermore, Diaphora has the added benefit of being open source, allowing Alien Labs to modify it for our needs.

How can binary diffing be employed to identify malware?

Diaphora works by analyzing each function present in the binary and extracting a set of features from each analyzed function. These features are later used to compare functions across binaries and find matches. If instead of directly comparing features, we leverage them to build a database of malicious functions (indicators) for identification purposes, we can then begin analyzing incoming binaries and try to find matches amongst their functions when comparing to the indicator database.

If enough matches are found in the analyzed binaries, we can safely assume the analyzed sample is a malware sample. We can also note which malware family the functions belong to in the indicator database, thus obtaining family attribution for the analyzed samples.

Porting Diaphora to Radare2

Diaphora works as an IDA Pro plugin. In order to work, it needs a valid IDA license and, consequently, valid Hex-Rays licenses for each CPU architecture you may want to decompile. As this cost of these licenses is quite high, Alien Labs looked for a cheaper alternative, so the community could leverage it.

As such, we decided to port the existing Diaphora to the Radare2 disassembly framework. The ported version of Diaphora, named r2diaphora, is also open source and available here.

Radare2 (r2) is an open-source disassembly framework that supports a very wide range of CPU architectures. It also bundles a capable decompiler and supports the Ghidra decompiler as a plugin. As such, r2 is well suited for our objective of porting Diaphora to an open-source disassembler.

Additional changes made to the original Diaphora included swapping the SQLite3 databases for MySQL. This change was performed for the malware attribution process described previously, as more than one analyst would be writing to the indicator database. With multiple analysts writing to the database, the SQLite database would need to be shared

across team members and allow parallel write/read operations. SQLite databases are not made for this kind of usage, so the Alien Labs team swapped it for another database engine better designed for the task.

Installation

As r2diaphora uses Radare2 and MySQL they need to be set-up prior to its usage. Radare2 should be installed locally, while the MySQL server can be remote or local. Once the environment is set up you can install it with pip install r2diaphora. This pip package installs three command line utilities: r2diaphora, r2diaphora-db and r2diaphora-bulk.

- r2diaphora: The main command line utility, analyzes and compares files.
- r2diaphora-db: Performs database management and configuration.
- r2diaphora-bulk: Analyzes binaries in batches.

Further usage options can be obtained with the -h / --help command line option in each of them.

Once the pip package is successfully installed you can input your database credentials with r2diaphora-db config -u -p -hs . If you are using bash or a similar shell and do not want your database password to be saved in the shell history, precede the command with a space.

Finally, if you want to use the r2ghidra decompiler, install it with the r2pm -ci r2ghidra command, if it is not installed already.

Usage

As stated previously, r2ghidra lists all available options if executed with the -h flag. Currently, they are the following:

```
~  $ r2diaphora -h
usage: r2diaphora [-h] [-f] [-nbbs NBBS] [-o O] [-d {pdc,ghidra}] [-nd] [-a] file1 [file2]

positional arguments:
  file1                 File to analyze
  file2                 (Optional) File to diff against

optional arguments:
  -h, --help            show this help message and exit
  -f                    Force DB override
  -nbbs NBBS            Functions with a number of basic blocks below this number are excluded from analysis
  -o O                  Diff output file (HTML) - Default value: <db1name>_vs_<db2name>.html
  -d {pdc,ghidra}, --decompiler {pdc,ghidra}
                        Which decompiler to use
  -nd, --no-decompiler  Do not use the decompiler
  -a                    Analyze ALL functions (by default library functions are skipped)
```

As an example, we can execute r2diaphora on some test IoT samples. You can find file hashes in the Associated Indicators appendix.

First test - comparing to Sakura (a Gafgyt variant) samples with the same architecture:

r2diaphora 562b4c9a40f9c88ab84ac4ffd0deacd219595ab83ed23a458c5f492594a3a7ef 770363f9fd334c3f3c4ba0e05a2a0d4701f56a629b09365dfe874b2a277f4416

```
File 1: 562b4c9a40f9c88ab84ac4ffd0deacd219595ab83ed23a458c5f492594a3a7ef
Type: ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, with debug_info, not stripped
SHA256: 562b4c9a40f9c88ab84ac4ffd0deacd219595ab83ed23a458c5f492594a3a7ef
```

```
File 2: 770363f9fd334c3f3c4ba0e05a2a0d4701f56a629b09365dfe874b2a277f4416
Type: ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, with debug_info, not stripped
SHA256: 770363f9fd334c3f3c4ba0e05a2a0d4701f56a629b09365dfe874b2a277f4416
```

Found 40 matches across compared files

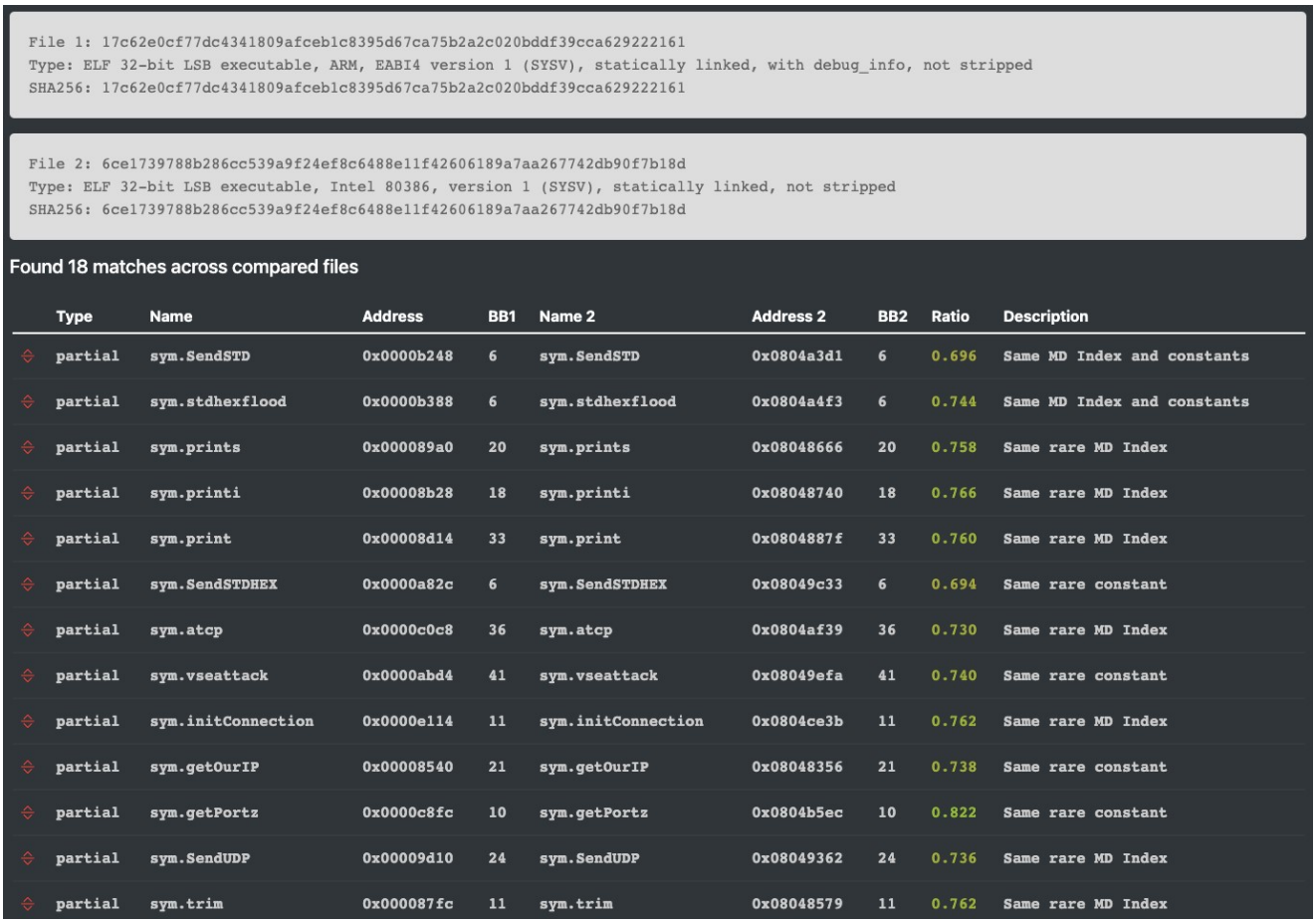
Type	Name	Address	BB1	Name 2	Address 2	BB2	Ratio	Description
best	sym._call_via_r6	0x000080e8	1	sym.call___do_global_dtors_aux	0x00008128	1	1.000	Same cleaned pseudo-code
best	sym._call_via_r5	0x000080e4	1	sym.call_frame_dummy	0x00008188	1	1.000	Same cleaned pseudo-code
best	sym._call_via_r3	0x000080dc	1	sym.call___do_global_ctors_aux	0x0001686c	1	1.000	Same cleaned pseudo-code
best	sym.prints	0x000089a0	20	sym.prints	0x00008988	20	1.000	Same cleaned pseudo-code
partial	sym.getArch	0x0000c8d8	1	sym.getArch	0x0000ca4c	1	0.770	Same constants
partial	sym.init_rand	0x000081ec	4	sym.init_rand	0x000081cc	4	0.854	Same constants
partial	sym.getOurIP	0x00008540	21	sym.getOurIP	0x00008508	21	0.900	Same constants
partial	sym.getPortz	0x0000c8fc	10	sym.getPortz	0x0000ca68	10	0.912	Same constants
partial	sym.SendSTD	0x0000b248	6	sym.SendSTD	0x0000b328	6	0.904	Same KOKA hash and constants
partial	sym.SendSTD_HEX	0x0000b4d8	6	sym.SendSTD_HEX	0x0000b5e0	6	0.902	Same KOKA hash and constants
partial	sym.SendSTDHEX	0x0000a82c	6	sym.SendSTDHEX	0x0000a83c	6	0.904	Same KOKA hash and constants
partial	sym.printi	0x00008b28	18	sym.printi	0x00008b0c	18	0.876	Same rare MD Index
partial	sym.print	0x00008d14	33	sym.print	0x00008cf4	33	0.848	Same rare MD Index

Figure 1. r2diaphora output for Sakura samples with the same architecture.

Observe how r2diaphora could identify the similarities between the two files. The system managed to find 40 matches out of 56 possible (71%). Furthermore, the similarity ratios for the matched functions are close to 1.0, indicating a very close resemblance in the matched functions. Additionally, the results point towards true positive matches since the matched functions have the same name and number of basic blocks.

Second test - comparing Sakura samples with different architectures:

r2diaphora 17c62e0cf77dc4341809afceb1c8395d67ca75b2a2c020bddf39cca6292221616ce1739788b286cc539a9f24ef8c6488e11f42606189a7aa267742db90f7b18d



Type	Name	Address	BB1	Name 2	Address 2	BB2	Ratio	Description
partial	sym.SendSTD	0x0000b248	6	sym.SendSTD	0x0804a3d1	6	0.696	Same MD Index and constants
partial	sym.stdhexflood	0x0000b388	6	sym.stdhexflood	0x0804a4f3	6	0.744	Same MD Index and constants
partial	sym.prints	0x000089a0	20	sym.prints	0x08048666	20	0.758	Same rare MD Index
partial	sym.printi	0x00008b28	18	sym.printi	0x08048740	18	0.766	Same rare MD Index
partial	sym.print	0x00008d14	33	sym.print	0x0804887f	33	0.760	Same rare MD Index
partial	sym.SendSTDHEX	0x0000a82c	6	sym.SendSTDHEX	0x08049c33	6	0.694	Same rare constant
partial	sym.atcp	0x0000c0c8	36	sym.atcp	0x0804af39	36	0.730	Same rare MD Index
partial	sym.vseattack	0x0000abd4	41	sym.vseattack	0x08049efa	41	0.740	Same rare constant
partial	sym.initConnection	0x0000e114	11	sym.initConnection	0x0804ce3b	11	0.762	Same rare MD Index
partial	sym.getOurIP	0x00008540	21	sym.getOurIP	0x08048356	21	0.738	Same rare constant
partial	sym.getPortz	0x0000c8fc	10	sym.getPortz	0x0804b5ec	10	0.822	Same rare constant
partial	sym.SendUDP	0x00009d10	24	sym.SendUDP	0x08049362	24	0.736	Same rare MD Index
partial	sym.trim	0x000087fc	11	sym.trim	0x08048579	11	0.762	Same rare MD Index

Figure 2. r2diaphora output for Sakura samples with different architecture.

In this case, we see how the number of matches has decreased from the previous test. This was expected as it is harder to match functions across different architectures. The similarity ratios have also decreased as the assembly code differs in all the compared functions. Still, r2diaphora recognized many similarities between both samples and identified correct matches across the compared files.

Third test - comparing a Sakura sample to a Yakuza (another Gafgyt variant) sample, both samples having different architectures:

$ r2diaphora
sakura/594a6b2c1e9beac3ad5f84458b71c1b7ec05ee0239808c9a63bc901040e413a3
yakuza/91392f5dbbfd4ad142956983208a484b91ac5e84c4f9a9fcb530a9b085644c93

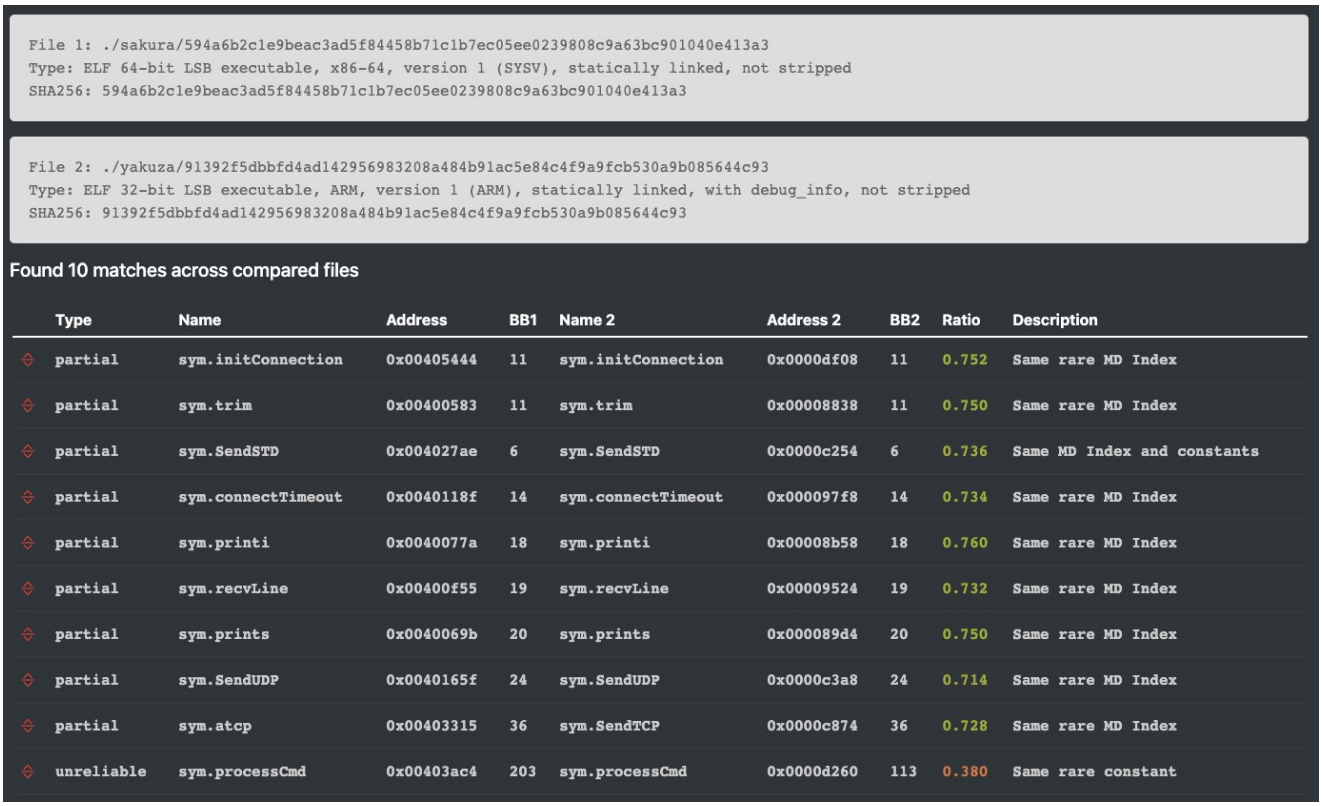


Figure 3. r2diaphora output for Sakura and Yakuza samples with different architecture.

In this case, observe how the number of matches have decreased even further while the ratios have been maintained mostly steady. This is due to the samples being different variants that perform different modifications over the base Gafgyt source code.

It is also notable that the processCmd function has been able to be matched with a low ratio. processCmd is the function that parses the received commands from the Command & Control server. The low ratio in this match is due to the variants being able to handle different commands, hence their implementation being different. However, the system was able to match it due to a common constant present in both functions.

Conclusion

Code similarity analysis is a powerful tool that can be leveraged to identify and attribute malware. While not flawless, program diffing can bypass many of the weaknesses of static signatures and thus could be used in conjunction with traditional detection methods to build a more robust detection pipeline.

Appendix

Associated Indicators (IOCs)

TYPE	INDICATOR	DESCRIPTION
SHA256	132948bef56cc5b4d0e435f33e26632264d27ce7d61eba85cf3830fdf7cb8056	Sakura sample, Arch: ARM, EABI4
SHA256	136dbd3cfa947f286b972af1e389b2a44138c0013aa8060d20c247b6bcfdd88c	Sakura sample, Arch: Intel 80386
SHA256	17c62e0cf77dc4341809afceb1c8395d67ca75b2a2c020bddf39cca629222161	Sakura sample, Arch: ARM, EABI4
SHA256	19e0f329b5d8689b14d901b9b65c8d4fb28016360f45b3dfcec17e8340e6411e	Sakura sample, Arch: Motorola m68k
SHA256	4cc11ffb3681ebced1f9d88e71b70a87e6d4498abca823245c118afead67b6a5	Sakura sample, Arch: MIPS, MIPS-I version 1
SHA256	562b4c9a40f9c88ab84ac4ffd0deacd219595ab83ed23a458c5f492594a3a7ef	Sakura sample, Arch: ARM, EABI4
SHA256	594a6b2c1e9beac3ad5f84458b71c1b7ec05ee0239808c9a63bc901040e413a3	Sakura sample, Arch: x86-64
SHA256	5fec87479a8d2fa7f0ed7c8f6ba76eeea9e86c45123173d2230149a55dcd760d	Sakura sample, Arch: MIPS, MIPS-I version 1
SHA256	603d14671f97d12db879cc1c7cd6abfa278bf46431ac73aeb6b3a4c4c2b16b9f	Sakura sample, Arch: x86-64

SHA256	6b128a64a497eb123f03b77ef45e99e856282dc9620dc26ab38998627a8f3216	Sakura sample, Arch: Renesas SH
SHA256	6ce1739788b286cc539a9f24ef8c6488e11f42606189a7aa267742db90f7b18d	Sakura sample, Arch: Intel 80386
SHA256	770363f9fd334c3f3c4ba0e05a2a0d4701f56a629b09365dfe874b2a277f4416	Sakura sample, Arch: ARM, version 1
SHA256	7c8ba5f88b1c4689a64652f0b8f5e3922e83f9f73c7e165f3213de27c5fb4d05	Sakura sample, Arch: PowerPC
SHA256	8090c3a1a930849df42f7f796d42e0211344e709a5ac15c2b4aca8ca41de2cd3	Sakura sample, Arch: Intel 80386
SHA256	94a279397b8c19ec7def169884a096d4f85ce0e21ff9df0be3ce264ef4565ea7	Sakura sample, Arch: x86-64
SHA256	96bb3e5209e083544ea6a78bc6fc4ebc456e135a786d747718d936af3b063298	Sakura sample, Arch: ARM, EABI4
SHA256	a079dfd60b55a7d74dd32d49a984bea43665b8b225beceae5b2729448892217f6	Sakura sample, Arch: MIPS, MIPS-I version 1
SHA256	b6c2f02b1bed62a6b845d5f13d9003f5aa3f6d0da3e62fa48d9822872453de10	Sakura sample, Arch: Renesas SH
SHA256	cef15aa60dc2c09fe117e37e07399f0ef89dca9f930ce13ac1e29f8cf63d9a31	Sakura sample, Arch: Motorola m68k

SHA256	e984334bbdd1179aadbde949f7c1b0fb02b6c18cb4a56d146150853b18adfa79	Sakura sample, Arch: MIPS, MIPS-I version 1
SHA256	2858982408bf1664b622e830ad83b871749608a7533e94672153ff90caa658a9	Yakuza sample, Arch: ARM, EABI4
SHA256	2b7262cae9e192fa7921f3ec02e0f924b32de3d418842fdad9a51603589a54c7	Yakuza sample, Arch: Intel 80386
SHA256	2faf7437c769abd92347d6f0a77f001523ec41c02d2bf12e3cebf5b950457ba3	Yakuza sample, Arch: Intel 80386
SHA256	4fc23e8409becb028997c2f0f2041e2dc853018b71e009e3d66f33876d5d4e99	Yakuza sample, Arch: Renesas SH
SHA256	6554d5edb401e2def2ef9fbb82b591351d3c8261ce0a20c431470f1c68fa3aea	Yakuza sample, Arch: ARM, version 1
SHA256	8005db9431013f094a2114046679ab971e62a8776639d6c2903fcc5d2fe8065c	Yakuza sample, Arch: x86-64
SHA256	91392f5dbbfd4ad142956983208a484b91ac5e84c4f9a9fcb530a9b085644c93	Yakuza sample, Arch: ARM, version 1
SHA256	b8aadb66183196868a9ff20bebd9c289fbfe2985fb409743bb0d0fea513e9caf	Yakuza sample, Arch: ARM, EABI4
SHA256	d4f223fc5944bc06e12c675f0664509eeab527abc03cdd8c2fbd43947cc6cbab	Yakuza sample, Arch: ARM, version 1

SHA256	`f64b5f6dd7f222b7568bba9e05caa52f9e4186f9ba4856c8bf1274f4c77c653c`	Yakuza sample, Arch: Intel 80386

Share this with others

Tags: alien labs, code analysis, r2diaphora