Lab 5 - Software Security

> Igor Mpore
>
> BS19-CS01
>
> i.mpore@innopolis.university

1. Preparation

a. Debugger Choice

In this lab, I used **Cutter** as my debugger for the these binaries. Cutter is a QT based GUI for reverse engineering binaries, which makes use of **radare2** framework. It's by far the best tool used for reverse engineering. Ref

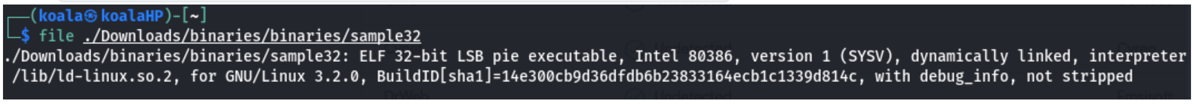
b. Checking if the given binaries are safe

As we already did in the previous lab of this course, we have to perform Malware analysis before we do reverse engineering of these binaries to know what they do. For these binaries to be analyzed, we'll use an online tool called **virustotal** as ANY.RUN doesn't support linux binaries. This will be a basic Static malware analysis.

sample32

1. *Checking the hash of a file to make sure it maches what Virustotal get:*

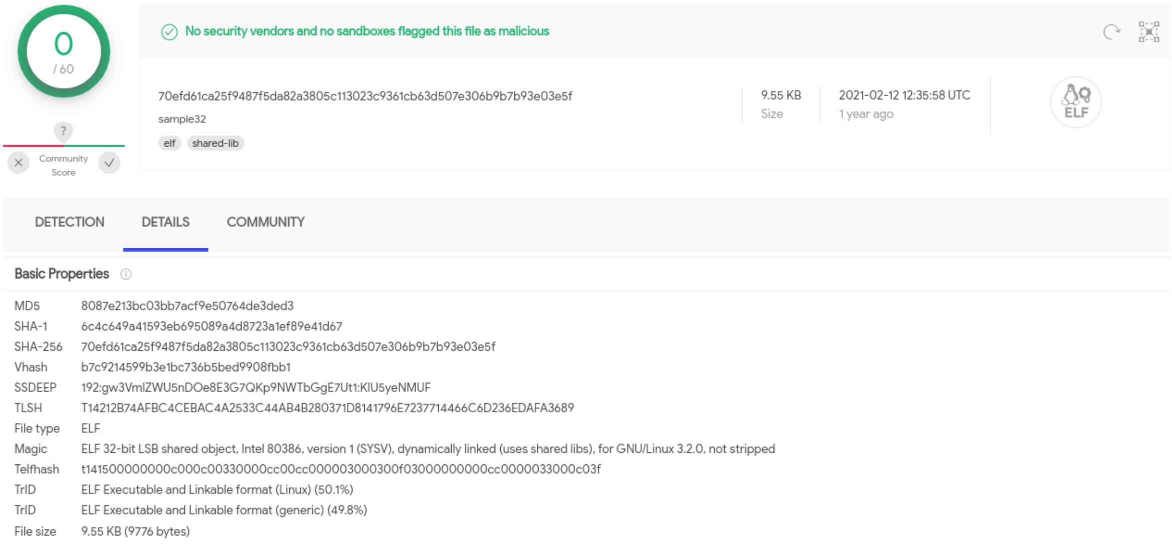
   `hashdeep ./Downloads/binaries/binaries/sample32`



```
┌──(koala㉿koalaHP)-[~]
└─$ file ./Downloads/binaries/binaries/sample32
./Downloads/binaries/binaries/sample32: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter
 /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=14e300cb9d36dfdb6b23833164ecb1c1339d814c, with debug_info, not stripped
```

- md5: 8087e213bc03bb7acf9e50764de3ded3
- sha256: 70efd61ca25f9487f5da82a3805c113023c9361cb63d507e306b9b7b93e03e5f

2. Report from virus total (Full report can be found here)



No security vendors and no sandboxes flagged this file as malicious

70efd61ca25f9487f5da82a3805c113023c9361cb63d507e306b9b7b93e03e5f

sample32

elf shared-lib

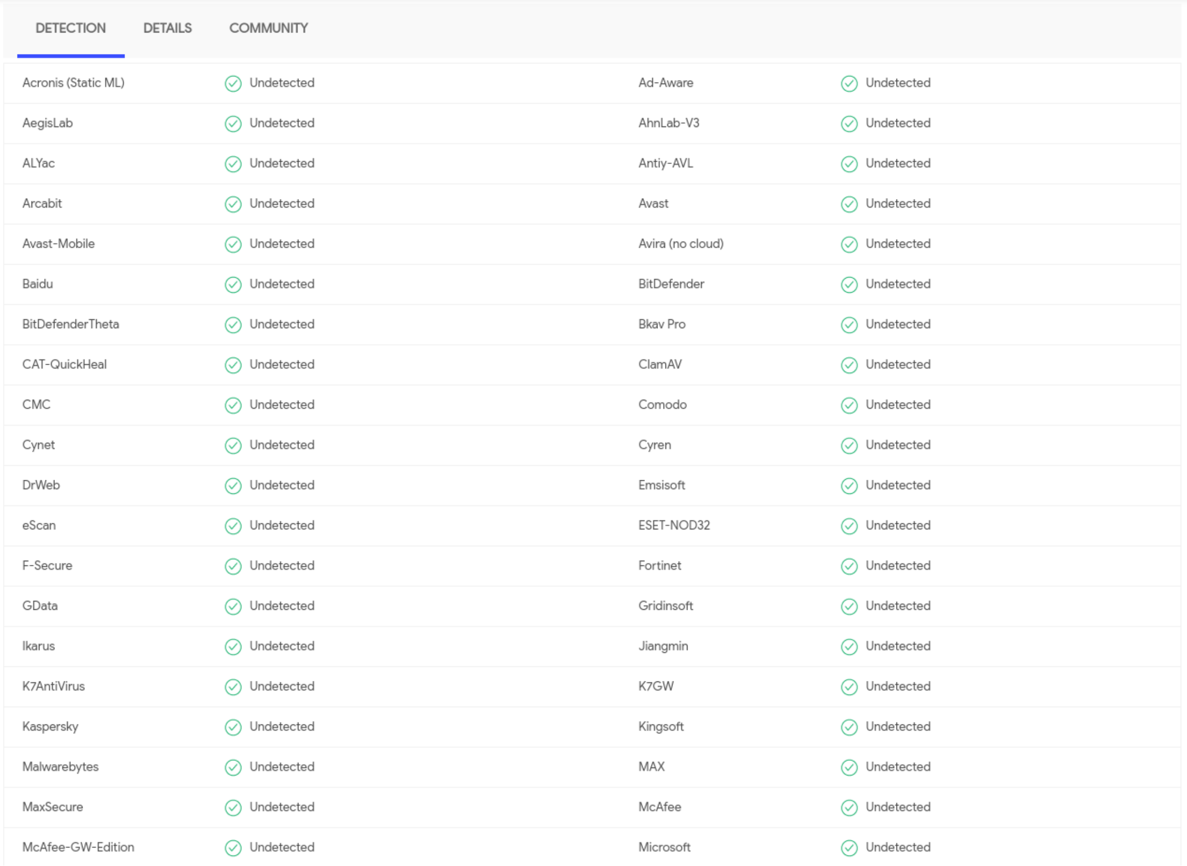
9.55 KB
Size

2021-02-12 12:35:58 UTC
1 year ago

Basic Properties

MD5	8087e213bc03bb7acf9e50764de3ded3
SHA-1	6c4c649a41593eb695089a4d8723a1ef89e41d67
SHA-256	70efd61ca25f9487f5da82a3805c113023c9361cb63d507e306b9b7b93e03e5f
Vhash	b7c9214599b3e1bc736b5bed9908fbb1
SSDEEP	192:gw3VmlZWU5nDOe8E3G7QKp9NWTbGgE7Ut1:KlU5yeNMUF
TLSH	T14212B74AFBC4CEBAC4A2533C44AB4B280371D8141796E7237714466C6D236EDAFA3689
File type	ELF
Magic	ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.2.0, not stripped
Telfhash	t141500000000c000c00330000cc00cc000003000300f03000000000cc0000033000c03f
TrID	ELF Executable and Linkable format (Linux) (50.1%)
TrID	ELF Executable and Linkable format (generic) (49.8%)
File size	9.55 KB (9776 bytes)

As you can see, we can confirm that the file's **MD5 and SHA256** is the same as what virus total got.



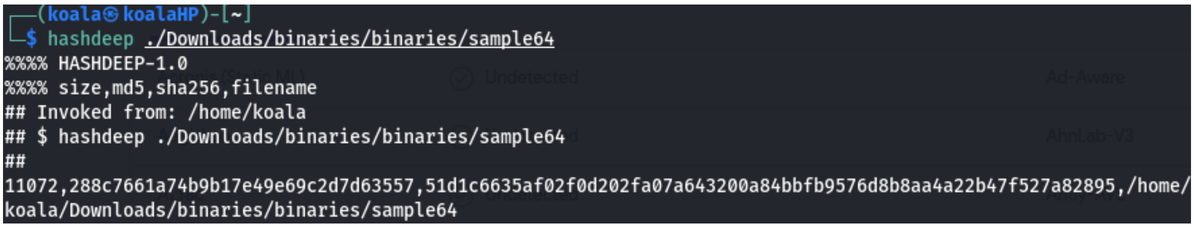
DETECTION DETAILS COMMUNITY

Acronis (Static ML)	✓ Undetected	Ad-Aware	✓ Undetected
AegisLab	✓ Undetected	AhnLab-V3	✓ Undetected
ALYac	✓ Undetected	Antiy-AVL	✓ Undetected
Arcabit	✓ Undetected	Avast	✓ Undetected
Avast-Mobile	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
CAT-QuickHeal	✓ Undetected	ClamAV	✓ Undetected
CMC	✓ Undetected	Comodo	✓ Undetected
Cynet	✓ Undetected	Cyren	✓ Undetected
DrWeb	✓ Undetected	Emsisoft	✓ Undetected
eScan	✓ Undetected	ESET-NOD32	✓ Undetected
F-Secure	✓ Undetected	Fortinet	✓ Undetected
GData	✓ Undetected	Gridinsoft	✓ Undetected
Ikarus	✓ Undetected	Jiangmin	✓ Undetected
K7AntiVirus	✓ Undetected	K7GW	✓ Undetected
Kaspersky	✓ Undetected	Kingsoft	✓ Undetected
Malwarebytes	✓ Undetected	MAX	✓ Undetected
MaxSecure	✓ Undetected	McAfee	✓ Undetected
McAfee-GW-Edition	✓ Undetected	Microsoft	✓ Undetected

sample64

1. *Checking the hash of a file to make sure it maches what Virustotal get:*

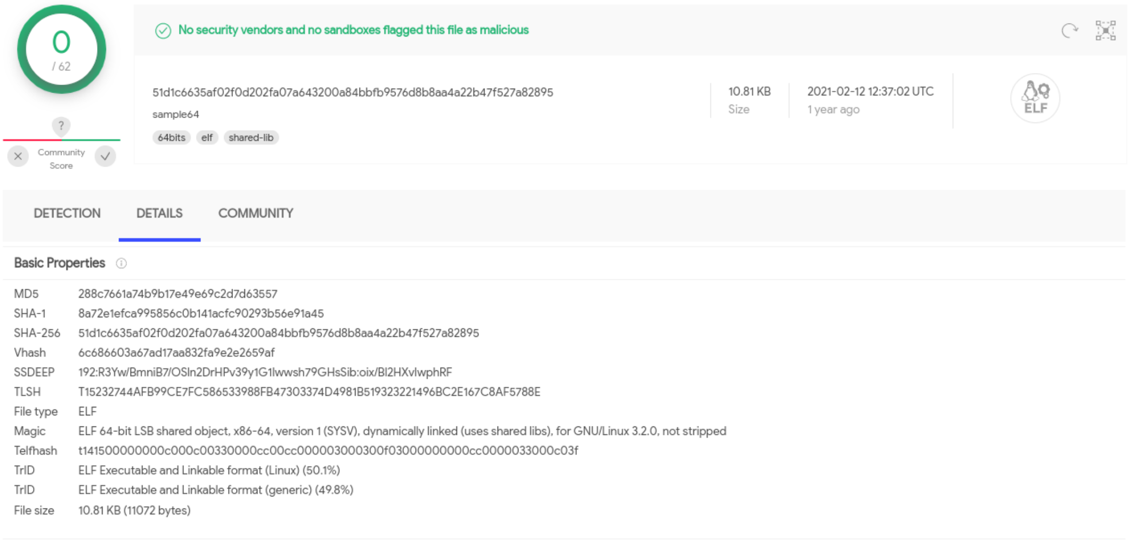
   `hashdeep ./Downloads/binaries/binaries/sample64`



```
┌──(koala㉿koalaHP)-[~]
└─$ hashdeep ./Downloads/binaries/binaries/sample64
%%%% HASHDEEP-1.0
%%%% size,md5,sha256,filename
## Invoked from: /home/koala
## $ hashdeep ./Downloads/binaries/binaries/sample64
##
11072,288c7661a74b9b17e49e69c2d7d63557,51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22b47f527a82895,/home/koala/Downloads/binaries/binaries/sample64
```

- md5: 288c7661a74b9b17e49e69c2d7d63557
- sha256: 51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22b47f527a82895

2. Report from virus total (Full report can be found here)



No security vendors and no sandboxes flagged this file as malicious

51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22b47f527a82895
sample64
64bits elf shared-lib

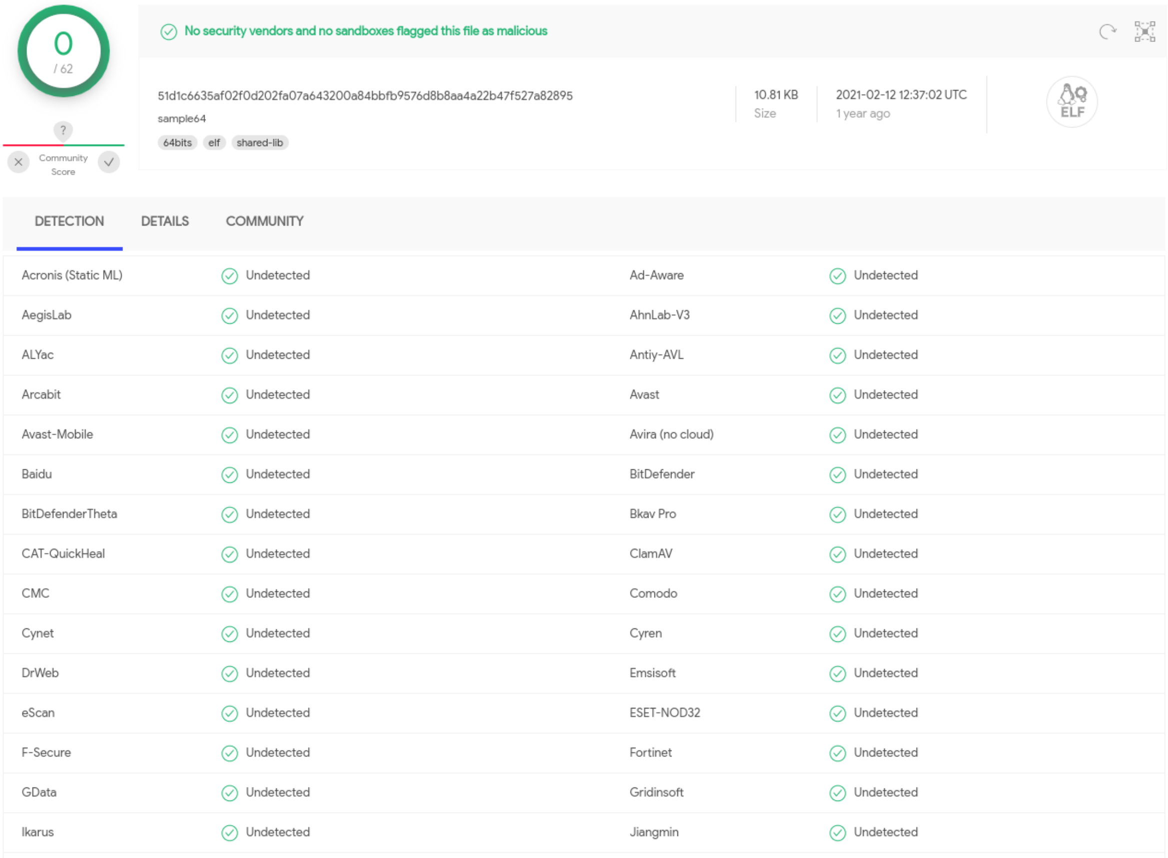
10.81 KB
Size

2021-02-12 12:37:02 UTC
1 year ago

Basic Properties

MD5	288c7661a74b9b17e49e69c2d7d63557
SHA-1	8a72e1efca995856c0b141acfc90293b56e91a45
SHA-256	51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22b47f527a82895
Vhash	6c686603a67ad17aa832fa9e2e2659af
SSDEEP	192:R3Yw/BmniB7/OSln2DrHPv39y1G1lwwsh79GHsSib:oix/Bl2HXvlwphRF
TLSH	T15232744AFB99CE7FC586533988FB47303374D4981B51923221496BC2E167C8AF5788E
File type	ELF
Magic	ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.2.0, not stripped
Telfhash	t141500000000c000c00330000cc00cc000003000300f03000000000cc0000033000c03f
TrID	ELF Executable and Linkable format (Linux) (50.1%)
TrID	ELF Executable and Linkable format (generic) (49.8%)
File size	10.81 KB (11072 bytes)

As you can see, we can confirm that the file's **MD5 and SHA256** is the same as what virus total got.



No security vendors and no sandboxes flagged this file as malicious

51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22b47f527a82895
sample64
64bits elf shared-lib

10.81 KB
Size

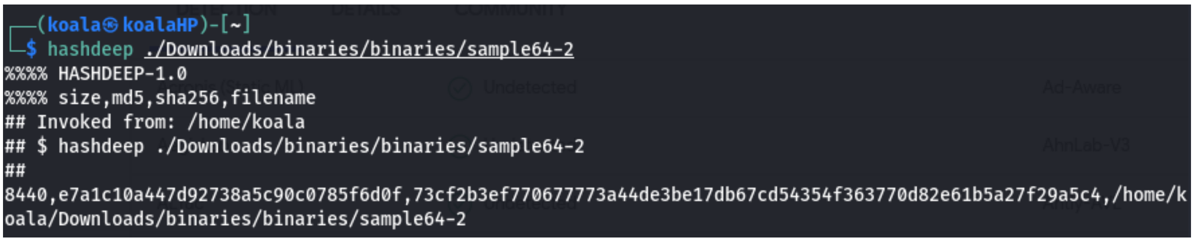
2021-02-12 12:37:02 UTC
1 year ago

Acronis (Static ML)	✓ Undetected	Ad-Aware	✓ Undetected
AegisLab	✓ Undetected	AhnLab-V3	✓ Undetected
ALYac	✓ Undetected	Antiy-AVL	✓ Undetected
Arcabit	✓ Undetected	Avast	✓ Undetected
Avast-Mobile	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
CAT-QuickHeal	✓ Undetected	ClamAV	✓ Undetected
CMC	✓ Undetected	Comodo	✓ Undetected
Cynet	✓ Undetected	Cyren	✓ Undetected
DrWeb	✓ Undetected	Emsisoft	✓ Undetected
eScan	✓ Undetected	ESET-NOD32	✓ Undetected
F-Secure	✓ Undetected	Fortinet	✓ Undetected
GData	✓ Undetected	Gridinsoft	✓ Undetected
Ikarus	✓ Undetected	Jiangmin	✓ Undetected

sample64-2

1. *Checking the hash of a file to make sure it maches what Virustotal get:*

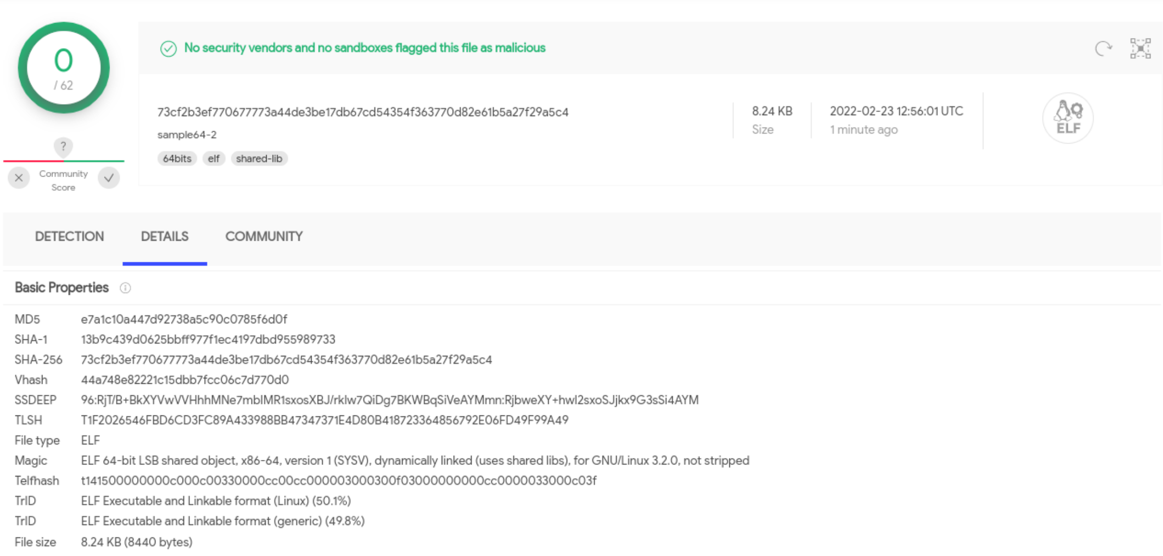
```
hashdeep ./Downloads/binaries/binaries/sample64-2
```



```
┌──(koala㉿koalaHP)-[~]
└─$ hashdeep ./Downloads/binaries/binaries/sample64-2
%%%% HASHDEEP-1.0
%%%% size,md5,sha256,filename
## Invoked from: /home/koala
## $ hashdeep ./Downloads/binaries/binaries/sample64-2
##
8440,e7a1c10a447d92738a5c90c0785f6d0f,73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61b5a27f29a5c4,/home/k
oala/Downloads/binaries/binaries/sample64-2
```

- md5: e7a1c10a447d92738a5c90c0785f6d0f
- sha256:73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61b5a27f29a5c4

2. Report from virus total (Full report can be found here)



0 / 62

?

No security vendors and no sandboxes flagged this file as malicious

73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61b5a27f29a5c4
sample64-2
64bits elf shared-lib

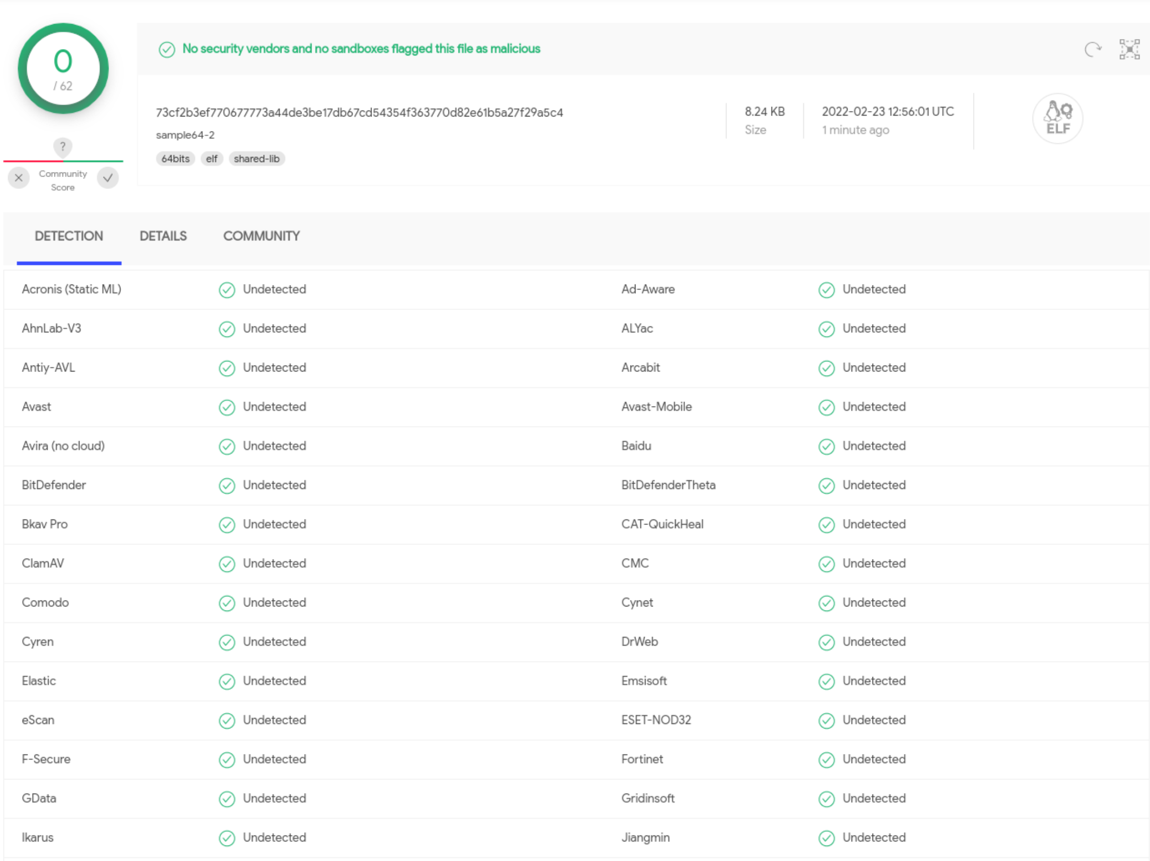
8.24 KB
Size

2022-02-23 12:56:01 UTC
1 minute ago

Basic Properties

MD5	e7a1c10a447d92738a5c90c0785f6d0f
SHA-1	13b9c439d0625bbff977f1ec4197dbd955989733
SHA-256	73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61b5a27f29a5c4
Vhash	44a748e82221c15dbb7fcc06c7d770d0
SSDEEP	96:RjT/B+BkXYVwVVHhhMNe7mblMR1sxosXBJ/rklw7QiDg7BKWBqSiVeAYMmn:RjbweXY+hwl2sxoSJjkx9G3sSi4AYM
TLSH	T1F2026546FBD6CD3FC89A433988BB47347371E4D80B418723364856792E06FD49F99A49
File type	ELF
Magic	ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.2.0, not stripped
Telfhash	t141500000000c000c00330000cc00cc000003000300f03000000000cc0000033000c03f
TrID	ELF Executable and Linkable format (Linux) (50.1%)
TrID	ELF Executable and Linkable format (generic) (49.8%)
File size	8.24 KB (8440 bytes)

As you can see, we can confirm that the file **MD5 and SHA256** has we have is the same as what virus total got.



0 / 62

?

No security vendors and no sandboxes flagged this file as malicious

73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61b5a27f29a5c4
sample64-2
64bits elf shared-lib

8.24 KB
Size

2022-02-23 12:56:01 UTC
1 minute ago

Acronis (Static ML)	✓ Undetected	Ad-Aware	✓ Undetected
AhnLab-V3	✓ Undetected	ALYac	✓ Undetected
Antiy-AVL	✓ Undetected	Arcabit	✓ Undetected
Avast	✓ Undetected	Avast-Mobile	✓ Undetected
Avira (no cloud)	✓ Undetected	Baidu	✓ Undetected
BitDefender	✓ Undetected	BitDefenderTheta	✓ Undetected
Bkav Pro	✓ Undetected	CAT-QuickHeal	✓ Undetected
ClamAV	✓ Undetected	CMC	✓ Undetected
Comodo	✓ Undetected	Cynet	✓ Undetected
Cyren	✓ Undetected	DrWeb	✓ Undetected
Elastic	✓ Undetected	Emsisoft	✓ Undetected
eScan	✓ Undetected	ESET-NOD32	✓ Undetected
F-Secure	✓ Undetected	Fortinet	✓ Undetected
GData	✓ Undetected	Gridinsoft	✓ Undetected
Ikarus	✓ Undetected	Jiangmin	✓ Undetected

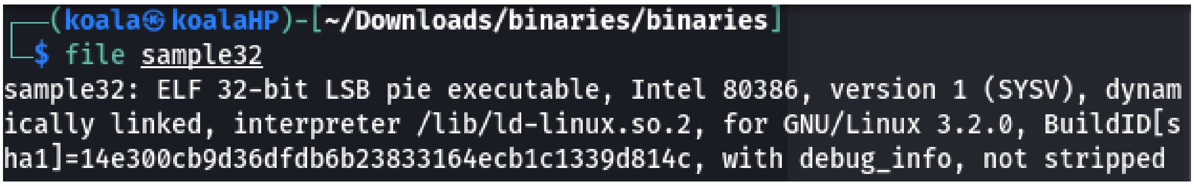
2. Theory

a. What kind of binaries I received

used the command `file binary_name` to know which kind of binary it is

- sample32

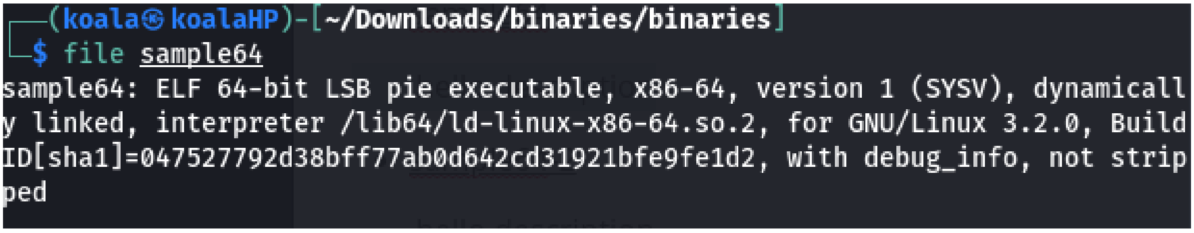
This is a 32bit binary



```
┌──(koala㉿koalaHP)-[~/Downloads/binaries/binaries]
└─$ file sample32
sample32: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynam
ically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[s
ha1]=14e300cb9d36dfdb6b23833164ecb1c1339d814c, with debug_info, not stripped
```

- sample64

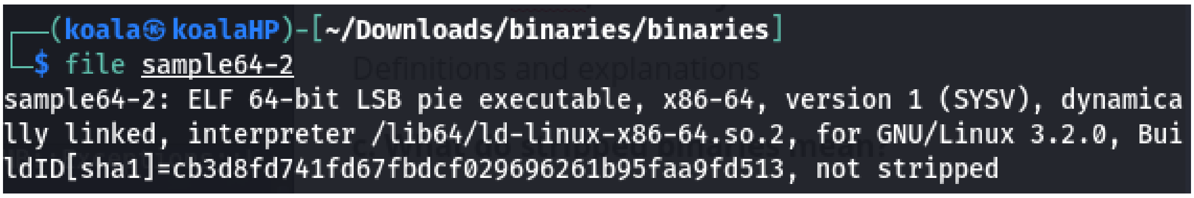
  This is a 64bit binary



```
┌──(koala㉿koalaHP)-[~/Downloads/binaries/binaries]
└─$ file sample64
sample64: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamicall
y linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, Build
ID[sha1]=047527792d38bff77ab0d642cd31921bfe9fe1d2, with debug_info, not strip
ped
```

- sample64-2

  This is a 64bit binary

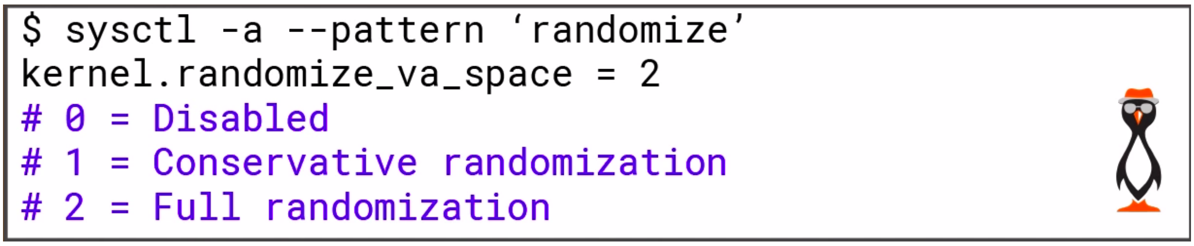


```
┌──(koala㉿koalaHP)-[~/Downloads/binaries/binaries]
└─$ file sample64-2
sample64-2: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, Bui
ldID[sha1]=cb3d8fd741fd67fbdcf029696261b95faa9fd513, not stripped
```

b. What is ASLR, and why do we need it?

Definition: Reference

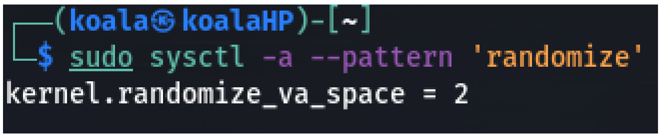
Address Space Layout Randomization (ASLR) is a security technique which involves positioning randomly the base address of an executable an the position of libraries heap, and stack, in a process's address space.



```
$ sysctl -a --pattern ‘randomize’
kernel.randomize_va_space = 2
# 0 = Disabled
# 1 = Conservative randomization
# 2 = Full randomization
```

we can check the if it's enabled/disabled using:

`sudo sysctl -a --pattern 'randomize'`



```
┌──(koala㉿koalaHP)-[~]
└─$ sudo sysctl -a --pattern 'randomize'
kernel.randomize_va_space = 2
```

Why do we need it

It helps secure a system by guarding it from attacks through **buffer-overflow** and finding address space of application. ASLR is able to put address space targets in unpredictable locations. If an attacker attempts to exploit an incorrect address space location, the target application will crash, stopping the attack and alerting the system.

c. What do stripped binaries mean?

To explain better the meaning of stripped binaries, let's check the difference between non-stripped and stripped binaries. **Non-stripped binaries** have debugging information build into them. When they are being compiled, they have a `gcc's -g` flag activated.

Where as **Stripped binaries**, the debugging information are removed from the exe (which isn't necessary for execution) to reduce the size of the exe.

d. What are GOT and PLT?

PLT stands for Procedure Linkage Table which is, put simply, used to call external procedures/functions whose address isn't known in the time of linking, and is left to be resolved by the dynamic linker at run time whereas

GOT stands for Global Offsets Table and is similarly used to resolve addresses in memory.

e. How can the debugger insert a breakpoint in the debugged binary/application?

To understand how a debugger inserts a breakpoint, let's first understand what a **breakpoint** is.

In software development, a breakpoint is an intentional stopping or pausing place in a program, put in place for debugging purposes. Ref

They are two forms of breakpoints: Ref

- Software breakpoint

  Software breakpoints replace an instruction opcode with a special “breakpoint opcode" by modifying the original program text.

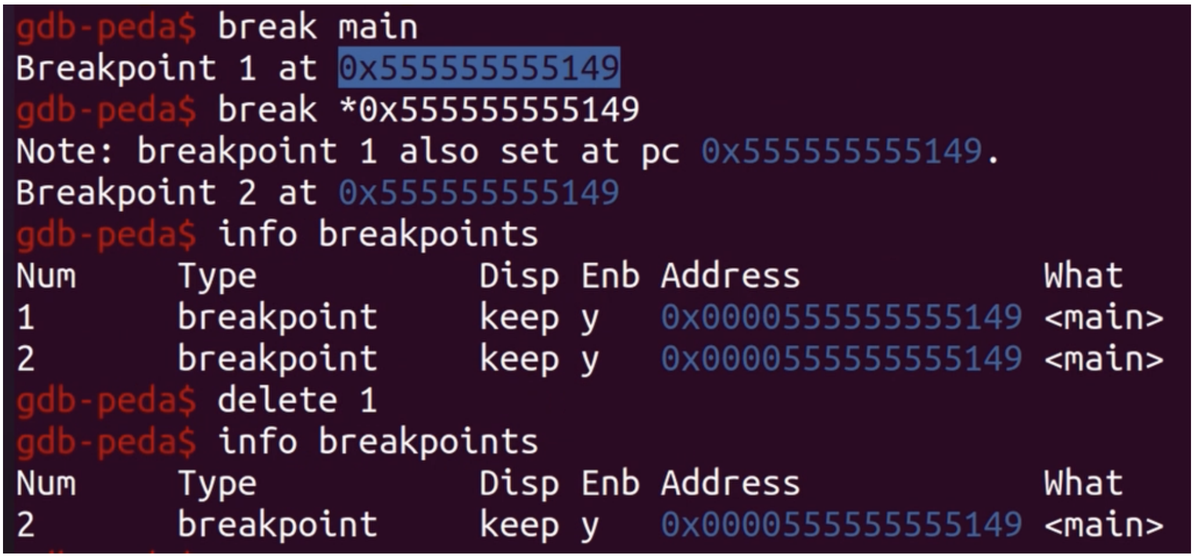
- Hardware breakpoint

Hardware breakpoints use dedicated hardware to examine the program counter and halt the machine when it reaches the specified address.

How the debugger inserts a breakpoint in the debugged binary/application:

From the above definitions, we can insert a breakpoint at a certain memory address.

For example, in GRB debugger; here's how it can be done after opening a file you want to debug:

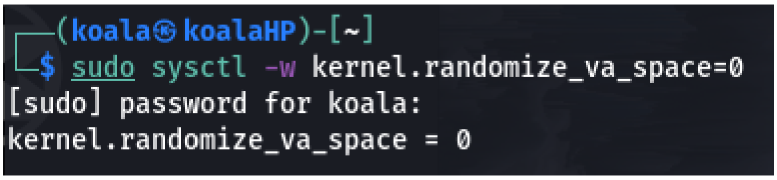


```
gdb-peda$ break main
Breakpoint 1 at 0x555555555149
gdb-peda$ break *0x555555555149
Note: breakpoint 1 also set at pc 0x555555555149.
Breakpoint 2 at 0x555555555149
gdb-peda$ info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555149 <main>
2       breakpoint     keep y   0x0000555555555149 <main>
gdb-peda$ delete 1
gdb-peda$ info breakpoints
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x0000555555555149 <main>
```

3. Reversing

a. Disable ASLR using the command below

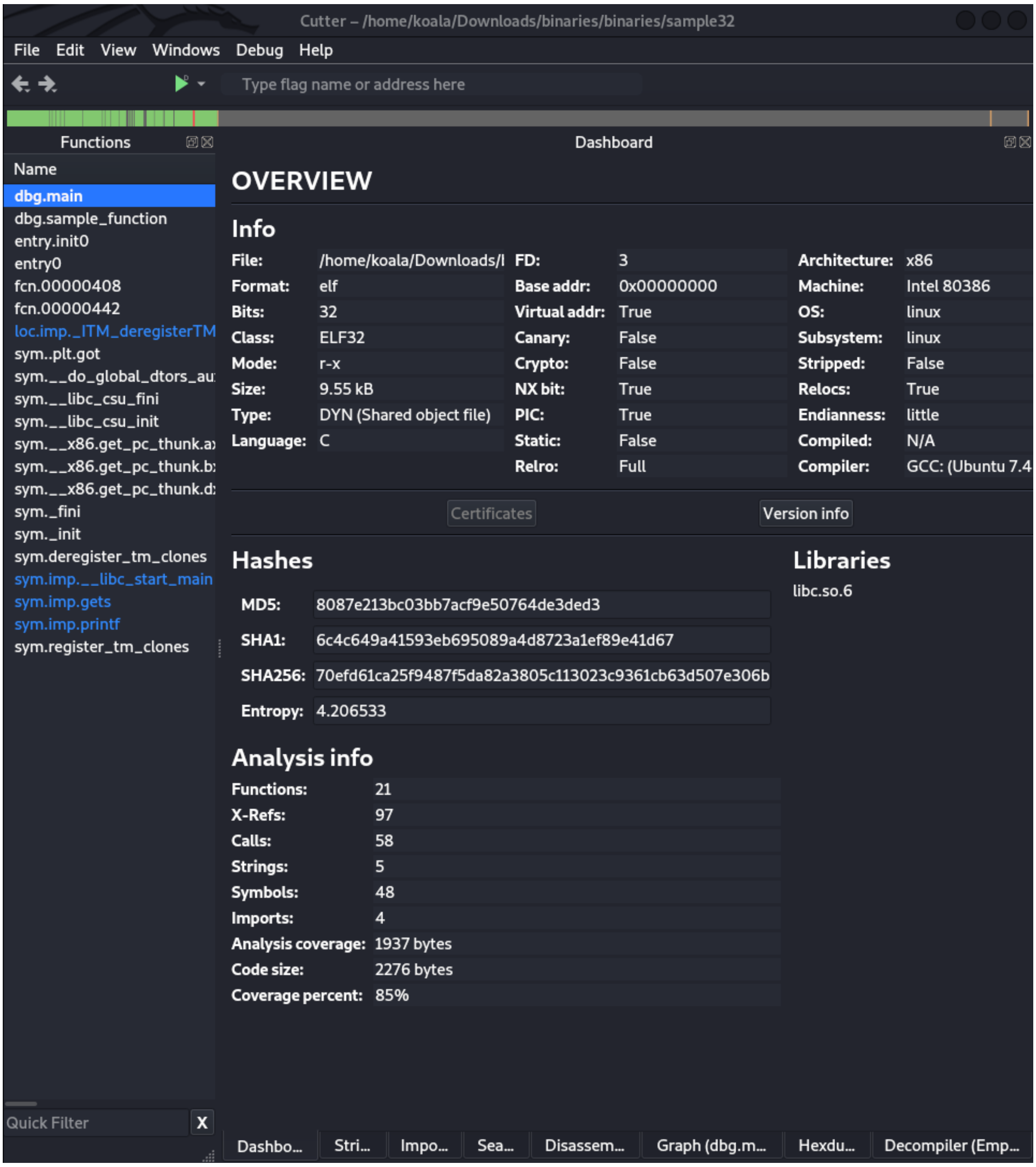
`sudo sysctl -w kernel.randomize_va_space=0`



```
┌──(koala㉿koalaHP)-[~]
└─$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for koala:
kernel.randomize_va_space = 0
```

b. Load the binaries into a disassembler/debugger

- sample32



Name

- dbg.main
- dbg.sample_function
- entry.init0
- entry0
- fcn.00000408
- fcn.00000442
- loc.imp._ITM_deregisterTM
- sym..plt.got
- sym.__do_global_dtors_au
- sym.__libc_csu_fini
- sym.__libc_csu_init
- sym.__x86.get_pc_thunk.a
- sym.__x86.get_pc_thunk.b
- sym.__x86.get_pc_thunk.d
- sym._fini
- sym._init
- sym.deregister_tm_clones
- sym.imp.__libc_start_main
- sym.imp.gets
- sym.imp.printf
- sym.register_tm_clones

OVERVIEW

Info

File:	/home/koala/Downloads/l	**FD:**	3	**Architecture:**	x86
Format:	elf	**Base addr:**	0x00000000	**Machine:**	Intel 80386
Bits:	32	**Virtual addr:**	True	**OS:**	linux
Class:	ELF32	**Canary:**	False	**Subsystem:**	linux
Mode:	r-x	**Crypto:**	False	**Stripped:**	False
Size:	9.55 kB	**NX bit:**	True	**Relocs:**	True
Type:	DYN (Shared object file)	**PIC:**	True	**Endianness:**	little
Language:	C	**Static:**	False	**Compiled:**	N/A
		Relro:	Full	**Compiler:**	GCC: (Ubuntu 7.4

Hashes

MD5:	8087e213bc03bb7acf9e50764de3ded3
SHA1:	6c4c649a41593eb695089a4d8723a1ef89e41d67
SHA256:	70efd61ca25f9487f5da82a3805c113023c9361cb63d507e306b
Entropy:	4.206533

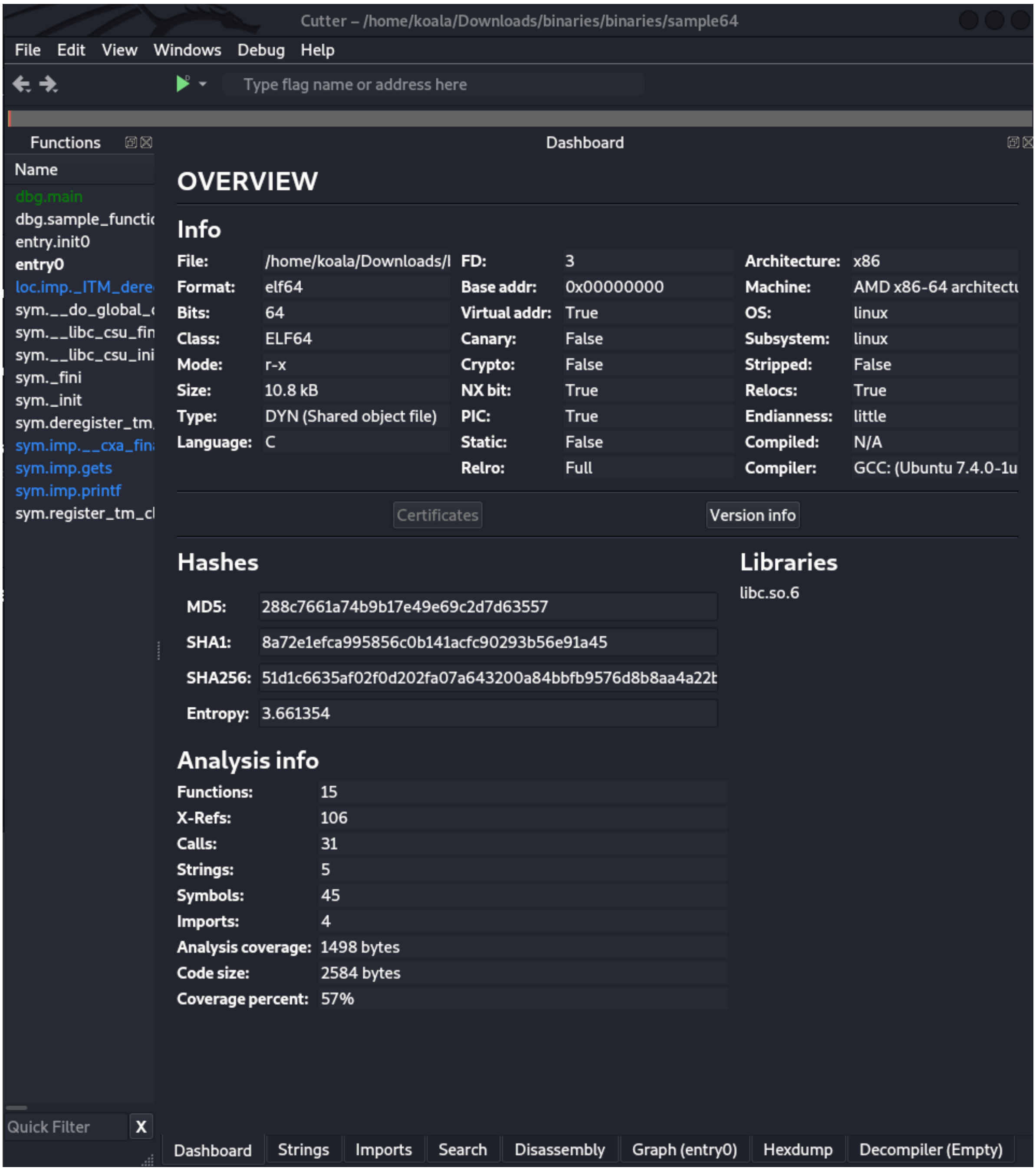
Libraries

libc.so.6

Analysis info

Functions:	21
X-Refs:	97
Calls:	58
Strings:	5
Symbols:	48
Imports:	4
Analysis coverage:	1937 bytes
Code size:	2276 bytes
Coverage percent:	85%

- sample64



dbg.main
dbg.sample_functic
entry.init0
entry0
loc.imp._ITM_dere
sym.__do_global_c
sym.__libc_csu_fin
sym.__libc_csu_ini
sym._fini
sym._init
sym.deregister_tm
sym.imp.__cxa_fin
sym.imp.gets
sym.imp.printf
sym.register_tm_cl

OVERVIEW

Info

File:	/home/koala/Downloads/l	**FD:**	3	**Architecture:**	x86
Format:	elf64	**Base addr:**	0x00000000	**Machine:**	AMD x86-64 architectu
Bits:	64	**Virtual addr:**	True	**OS:**	linux
Class:	ELF64	**Canary:**	False	**Subsystem:**	linux
Mode:	r-x	**Crypto:**	False	**Stripped:**	False
Size:	10.8 kB	**NX bit:**	True	**Relocs:**	True
Type:	DYN (Shared object file)	**PIC:**	True	**Endianness:**	little
Language:	C	**Static:**	False	**Compiled:**	N/A
		Relro:	Full	**Compiler:**	GCC: (Ubuntu 7.4.0-1u

Hashes

MD5:	288c7661a74b9b17e49e69c2d7d63557
SHA1:	8a72e1efca995856c0b141acfc90293b56e91a45
SHA256:	51d1c6635af02f0d202fa07a643200a84bbfb9576d8b8aa4a22t
Entropy:	3.661354

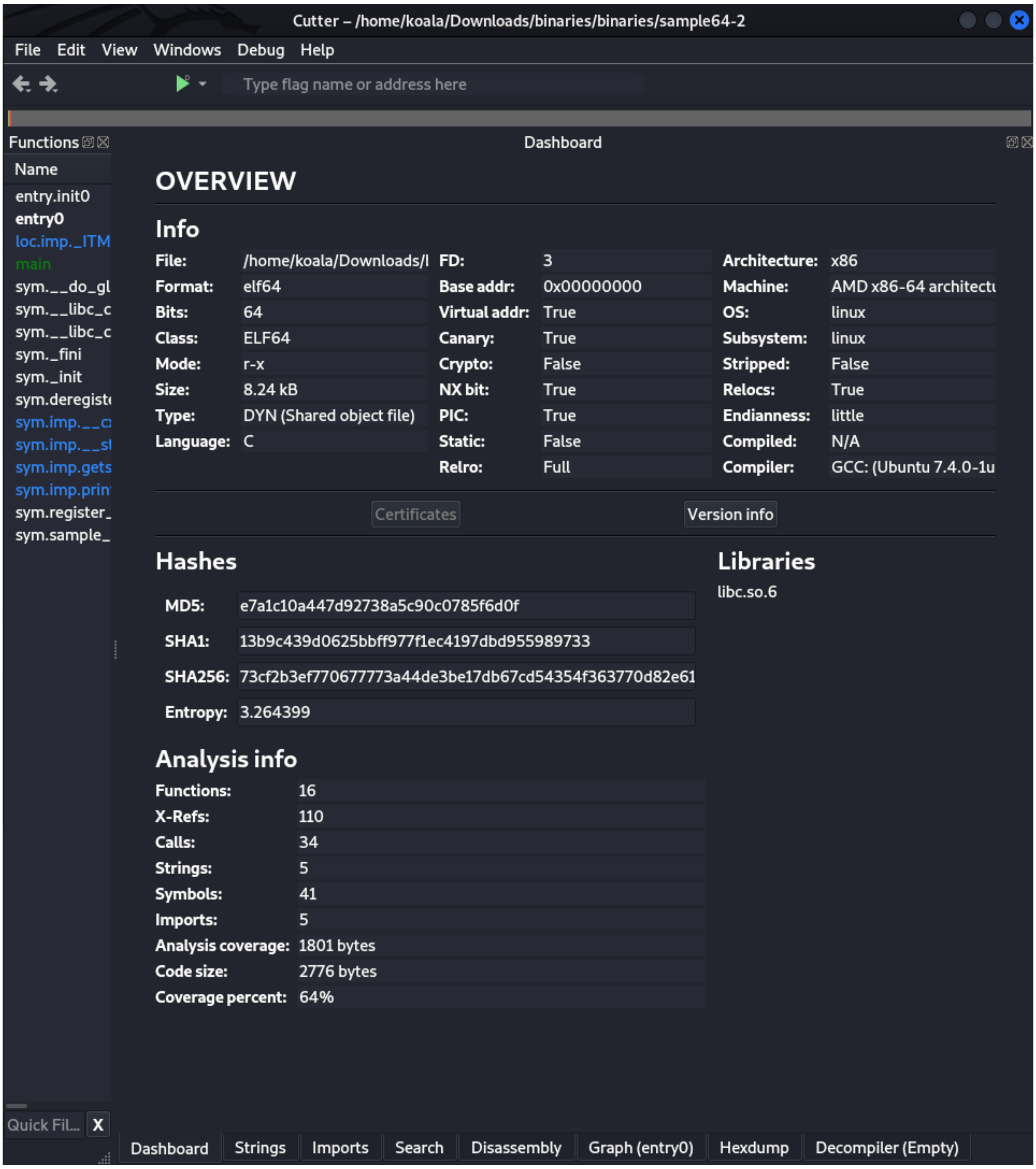
Libraries

libc.so.6

Analysis info

Functions:	15
X-Refs:	106
Calls:	31
Strings:	5
Symbols:	45
Imports:	4
Analysis coverage:	1498 bytes
Code size:	2584 bytes
Coverage percent:	57%

- sample64-2



OVERVIEW

Info

File:	/home/koala/Downloads/l	FD:	3	Architecture:	x86
Format:	elf64	Base addr:	0x00000000	Machine:	AMD x86-64 architectu
Bits:	64	Virtual addr:	True	OS:	linux
Class:	ELF64	Canary:	True	Subsystem:	linux
Mode:	r-x	Crypto:	False	Stripped:	False
Size:	8.24 kB	NX bit:	True	Relocs:	True
Type:	DYN (Shared object file)	PIC:	True	Endianness:	little
Language:	C	Static:	False	Compiled:	N/A
		Relro:	Full	Compiler:	GCC: (Ubuntu 7.4.0-1u

Hashes

MD5:	e7a1c10a447d92738a5c90c0785f6d0f
SHA1:	13b9c439d0625bbff977f1ec4197dbd955989733
SHA256:	73cf2b3ef770677773a44de3be17db67cd54354f363770d82e61
Entropy:	3.264399

Libraries

libc.so.6

Analysis info

Functions:	16
X-Refs:	110
Calls:	34
Strings:	5
Symbols:	41
Imports:	5
Analysis coverage:	1801 bytes
Code size:	2776 bytes
Coverage percent:	64%

c. Does the function prologue and epilogue differ in 32bit and 64bit? How do they operate?

Let's first define **Function Prologue** and **Function Epilogue**

- **Function prologue** is a few lines of code at the beginning of a function, which prepare the stack and registers for use within the function.
- **Function epilogue** appears at the end of the function, and restores the stack and registers to the state they were in before the function was called.

How do they differ in 32bit and 64bit?

The function prologue and epilogue differ in these two architectures since their job is prepare stack and registers of different sizes (32 bits of register size and 64 bits of register size respectively)

d. Does function calls differ in 32bit and 64bit? What about argument passing?

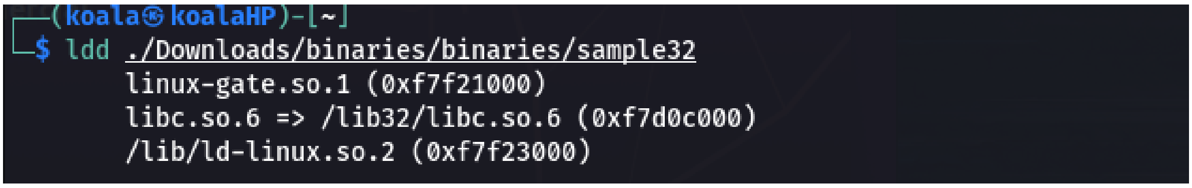
Yes, the function calls differ in 32bit and 64bit because the register sizes are different so, you need to always have address translation constantly between these two architectures. This also applies to argument passing.

e. What does the command ldd do?

`ldd BINARY-NAME`

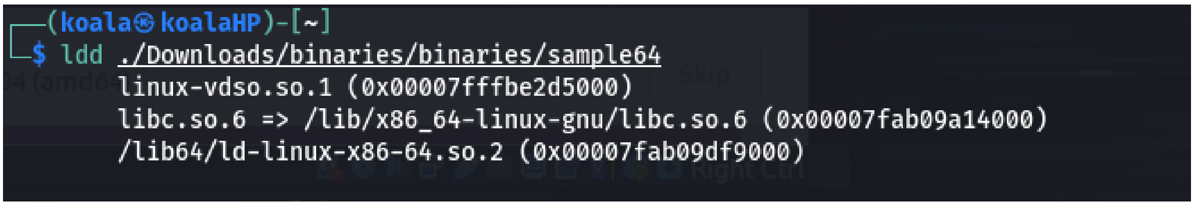
ldd command is used to display shared library dependencies of an executable or even for a shared library.

`ldd ./Downloads/binaries/binaries/sample32`



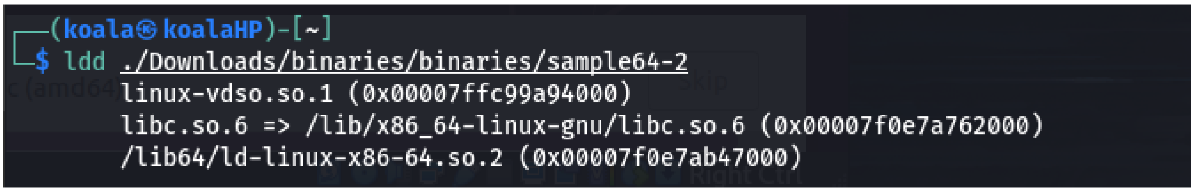
```
┌──(koala㉿koalaHP)-[~]
└─$ ldd ./Downloads/binaries/binaries/sample32
        linux-gate.so.1 (0xf7f21000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d0c000)
        /lib/ld-linux.so.2 (0xf7f23000)
```

`ldd ./Downloads/binaries/binaries/sample64`



```
┌──(koala㉿koalaHP)-[~]
└─$ ldd ./Downloads/binaries/binaries/sample64
        linux-vdso.so.1 (0x00007fffbe2d5000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fab09a14000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fab09df9000)
```

`ldd ./Downloads/binaries/binaries/sample64-2`



```
┌──(koala㉿koalaHP)-[~]
└─$ ldd ./Downloads/binaries/binaries/sample64-2
        linux-vdso.so.1 (0x00007ffc99a94000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f0e7a762000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f0e7ab47000)
```

f. Why in the “sample64-2“ binary, the value of i don’t change even if our input is very long?

> Hint: Stack Protection and Canary Protection

Stack protection refers to inserting a guard variable (referred to as canary) onto the stack frame for each vulnerable function or for all functions. For protecting **stack buffer overflows** that can be caused by returning larger values compared to what the stack can hold, Stack protection plays a bigger role.

In our case, the value of i returned is significantly large. To prevent stack buffer overflow, the value of i is kept the same after the function call **gets()**

Reference 1 , Reference 2