microsurf

Nicolas Dutly

Mar 16, 2022

CONTENTS:

CHAPTER ONE

QUICKSTART

In the general case, testing a target binary for side channels requires a number of items which are highly dependent on the binary, these include:

- The arguments to be passed to the binary
- Which input is considered to be secret
- How to generate secret inputs

As a user of the `microsurf` library, you have to specify these items. Fortunately, doing so is straightforward. The `SCDetector` is used to specify these items:

`class microsurf.`**`SCDetector`**`(`*binPath*, *args*, *randGen*, *deterministic*, *asFile*, *jail*, *leakageModel*, *sharedObjects=[]*`)`

The SCDetector class can be used to detect secret dependent memory accesses in generic applications

Parameters

- **`binPath`** (`str`) – Path to the target binary
- **`args`** (`list[str]`) – List of arguments to pass to the binary. For a secret argument, substitute the value of the argument with @ (for example, ['–encrypt', '<privkeyfile>'] would become ['–encrypt', '@']). Only one argument can be defined as secret.
- **`randGen`** (`Callable[[], str]`) – A function that generates random bytes in the format expected by the target binary. The SCDetector class will save these bytes to a temporary file and substitute the secret placeholder ('@') with the path to the file
- **`deterministic`** (`bool`) – Force deterministic execution by hooking relevant syscalls
- **`asFile`** (`bool`) – Specifies whether the target binary excepts the secret to be read from a file. If false, the secret will be passed directly as an argument
- **`jail`** (`str`) – Specifies the a directory to which the binary will be jailed during emulation. For dynamic binaries, the user must ensure that the appropriate shared objects are present.
- **`leakageModel`** (`Callable[[str], Any]`) – (Callable[[str], Any]): Function which applies a leakage model to the secret. Example under microsurf.pipeline.LeakageModels
- **`sharedObjects`** (`list[str]`) – List of shared libraries to trace. For example ['libssl.so.1.1', 'libcrypto.so.1.1']. Defaults to None, tracing only the target binary. Only applicable to dynamic binaries.

`exec``(`*report=False*`)`

Run the side channel analysis

Parameters **`report`** – Generate a markdown report.

1.1 Example: OpenSSL Camellia-128

Let us consider a practical example: Imagine that we want to test for side channels in openssl's Camellia-128 encryption algorithm on `x86-32`. Furthermore, let us assume that the binary is dynamically compiled.

What follows is a closer explanation of the non-trivial arguments that are needed by the `SCDetector` class.

A full working example can be found here.

1.1.1 Emulation root directory (`jail` argument)

For dynamic binaries you will have to provide a root directory (the `jail` argument), in which the binary will be emulated. The structure of the directory might look as follows:

```
jail-openssl-x8632/
lib/
  ld-linux.so.2
  libc.so.6
  libcrypto.so.1.1
  libssl.so.1.1
input.bin
openssl
```

As a user of the framework, you have to ensure that all required shared objects are present in the correct location.

```
jailroot = '/path/jail-openssl-x8632/'
scd = SCDetector(
        ...
        jail=jailroot,
        ...
    )
```

1.1.2 Specifying arguments (`args` argument)

If we want to encrypt a file `input.bin` using Camellia-128, we would run

```
openssl camellia-128-ecb -e -in input.bin -out output.bin -nosalt -K hexdata
```

where `hexdata` would be a 128bit hexadecimal key (for example: `96d496ea1378bf4f6e1f377606013e25`).

In the command above, the data we pass to the `-K` argument is secret. We can signal this to the microsurf framework by replacing the key with an '@' character:

```
opensslArgs = [
        "camellia-128-ecb",
        "-e",
        "-in",
        "input.bin",
        "-out",
        "output.bin",
        "-nosalt",
        "-K",
        "@",
```

(continues on next page)

(continued from previous page)

```
    ]
scd = SCDetector(
        ...
        args=opensslArgs
        ...
    )
```

The '@' will be replaced with the data produced by the `randGen` function. If `isFile` is true, then the framework will assume that the target binary loads the secret from a file. In that case it will replace the '@' with the path to a temporary file, whose content is generated by the `randGen` function.

1.1.3 Producing secrets (`randGen` argument)

A secret often has to adhere to some specific format in order to be processed by the target binary. Since microsurf cannot guess that, it is the end user's job to specify such a function. In our example, the `-K` flag expects a 128bit key specified as a hex string:

```
def genRandom() -> str:
    KEYLEN = 128
    kbytes = KEYLEN // 8
    rbytes = os.urandom(kbytes)
    return f"{int.from_bytes(rbytes, byteorder='big'):x}"
```

The `SCDetector` class will validate that the function generates different secrets at each call.

```
scd = SCDetector(
        ...
        randGen=genRandom,
        ...
    )
```

1.1.4 Secret leakage model (`leakagemodel` argument)

To estimate how *much* information is leaked, the `SCDetector` class needs a leakage model.

A leakage model is a transformation which is applied to the secret. An example model is the hamming weight of the secret key:

`class microsurf.pipeline.LeakageModels.`**`hamming`**`(`*secret*`)`

Computes the hamming distance of the secret

Parameters **`secret`** (`str`) – A base 10 or base 16 string representation of the secret

Return type `ndarray`

Returns a numpy array containing the hamming distance of the secret.

Custom leakage models can be passed to the `SCDetector` object.

1.1.5 Selective tracing (sharedObjects argument)

To selectively trace shared objects, a list of names can be passed to the sharedObjects argument. Note that this only works for dynamic libraries. For example:

```
sharedObjects = ['libssl', 'libcrypto']
```

will ignore every other shared library (libc etc).

1.2 Execution and results

Once an SCDetector object has been initialized with all the arguments needed, analysis can be started by calling the .exec() function:

The results will look be split in several columns:

```
0003018c - [MI = 0.22]   at set_hex                       openssl
0013f5cf - [MI = 0.21]   at OPENSSL_hexchar2int           libcrypto.so.1.1
00095610 - [MI = 0.12]   at _x86_Camellia_encrypt         libcrypto.so.1.1
00095ef0 - [MI = 0.13]   at Camellia_Ekeygen              libcrypto.so.1.1
000955f5 - [MI = 0.11]   at _x86_Camellia_encrypt         libcrypto.so.1.1
```

The first column is the relative offset within the shared object at which the leak was identified. The second column gives the estimated mutual information score optained by applying the specified leakage model. The next column provides the function name (if the symbols are available). The last column gives the name of the (shared) object.

The exec function takes an optional argument report. If set to True, the results will be saved in markdown table format, as exemplified by the next section.

1.3 Sample results

Run at: 03/15/2022, 11:19:26

Elapsed time (analysis): 00:01:31

Elapsed time (single run emulation): 0:00:00.427573

Binary

/home/nicolas/Documents/msc-thesis-work/doc/examples/rootfs/jail-openssl-arm64/openssl

> ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, with debug_info, not stripped

Args

['camellia-128-ecb', '-e', '-in', 'input.bin', '-out', 'output.bin', '-nosalt', '-K', '47356fc8ca111593e80e47e4667fb4c3']

Deterministic

False

Emulation root

/home/nicolas/Documents/msc-thesis-work/doc/examples/rootfs/jail-openssl-arm64/

Sample secret

```
e02216de8a7fc5c0f31aa99d84eacf76
```

Leakage model

```
identity
```

Results

Top 5, sorted by MI

offset	MI score	Function	Object
0x0df0ec	1.07667	Camellia_Ekeygen	libcrypto.so.1.1
0x0e05e8	0.279915	Camellia_EncryptBlock_Rounds	libcrypto.so.1.1
0x0e07f8	0.27345	Camellia_EncryptBlock_Rounds	libcrypto.so.1.1
0x0df44c	0.239999	Camellia_Ekeygen	libcrypto.so.1.1
0x0e0778	0.150515	Camellia_EncryptBlock_Rounds	libcrypto.so.1.1

Grouped by function name

Function	Leak Count
Camellia_Ekeygen	30
Camellia_EncryptBlock_Rounds	14

INDEX