xVMP Document v1.0

1. Introduction

We present XVMP, an LLVM-based code virtualization obfuscator fulfills these goals. It incorporates the obfuscation process of code virtualization into the compilation to mask the effects of different architectures and program languages. Specifically, XVMP identifies the to-be-protected functions according to developers' annotation and generates virtualized code based on LLVM intermediate representation (IR). After that, it embeds the interpreter of virtualized code into the IR. To evaluate its effectiveness, scalability, and compatibility, we applied XVMP on a microbenchmark and three real-world programs. Experiments show that XVMP's obfuscation is stronger than the state-of-the-art Tigress. XVMP can successfully protect C/C++ code and programs running on X86/64 and ARM32/64 architecture, The overhead of XVMP virtualization obfuscation is less than 2% when protecting critical functions in non-hot loops.

In the following documents, we introduce our prepared environment in Chapter 2; introduce how to reproduce our experiments in the paper in Chapter 3; we introduce how to use xVMP in Chapter 4; the protection effects of xVMP are covered in Chapter 5; finally, future improvements are presented in Chapter 6.

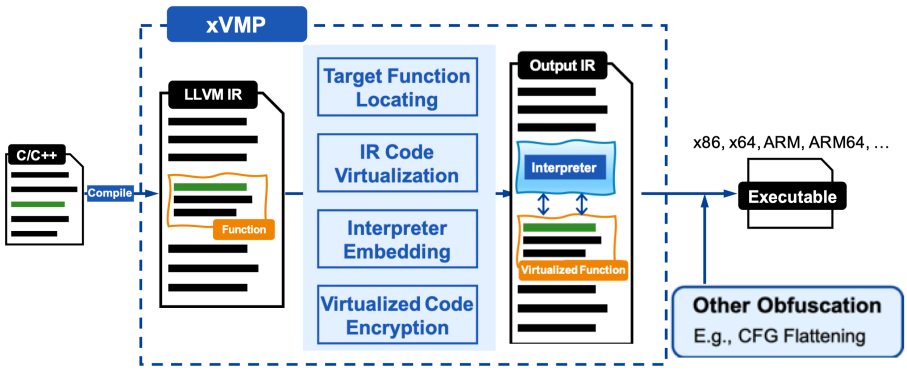


Fig. 1: The workflow of xVMP, which incorporates the obfuscation process into the compilation. Multiple obfuscation schemes are supported to add to the output IR for obfuscation strengthening.

2. Prebuilt Environment

We built a prototype system of xVMP in docker. You can directly download our pre-built docker image to run xVMP's prototype system. The source code will be released after acceptance.

The docker we provide includes two images: the xVMP running environment `xvmp-docker` and the de-virtualization tool environment `de-virtualization-docker` (for experiment 3.1).

Download the docker image of xVMP in the following way:

- Onedrive: https://1drv.ms/u/s!AkSbLEGzmY5MhmZM2uyENf0XPtBk?e=J4Aq2n
  - `xvmp-docker` : (sha256: bb7f6e8132675c4eadb773962099236a52a945e3547f433b05ffe25e4cbe2514)
  - `de-virtualization-docker` : (sha256: eaa8d8082c35705d0929bfde8605825ee88e2ed21e22ce243826d96eb93bab4e)

3. Evaluation Instructions

3.0 Setup Environment

1. Download the docker images (`xvmp-docker` and `de-virtualization-docker`)

```
wget http://xvmp.homes/xvmp_docker_v1.0.tar
wget http://xvmp.homes/xvmp_devirtualization_docker_latest.tar
```

2. Load the images

```
sudo docker load < xvmp_docker_v1.0.tar
sudo docker load < xvmp_devirtualization_docker_latest.tar
```

```
$ ls
xvmp_devirtualization_docker_latest.tar  xvmp_docker_v1.0.tar
$ sudo docker load < xvmp_docker_v1.0.tar
7789f1a3d4e9: Loading layer [==================================================>]  75.22MB/75.22MB
9e53fd489559: Loading layer [==================================================>]  1.011MB/1.011MB
2a19bd70fcd4: Loading layer [==================================================>]  15.36kB/15.36kB
8891751e0a17: Loading layer [==================================================>]  3.072kB/3.072kB
23367024ba6f: Loading layer [==================================================>]  1.178GB/1.178GB
c9051b5fac28: Loading layer [==================================================>]      4GB/4GB
Loaded image: xvmp:v1.0
$ sudo docker load < xvmp_devirtualization_docker_latest.tar
da2785b7bb16: Loading layer [==================================================>]  134.7MB/134.7MB
bbc674332e2e: Loading layer [==================================================>]  15.87kB/15.87kB
5b7dc8292d9b: Loading layer [==================================================>]  11.78kB/11.78kB
1a1a19626b20: Loading layer [==================================================>]  3.072kB/3.072kB
cd845b8013e3: Loading layer [==================================================>]  1.856GB/1.856GB
Loaded image: xvmp_devirtualization:latest
$ sudo docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
xvmp_devirtualization   latest    ba213fcc1c66   34 minutes ago   1.94GB
xvmp                    v1.0      7171405b8609   4 hours ago      5.23GB
$
```

3. Run the images with a mount directory to facilitate the file transfer.

   Please replace `/home/xgx/workspace/xvmp_mnt` with your path in the following commands.

```
mkdir /tmp/xvmp_mnt
sudo docker run -itd -v /tmp/xvmp_mnt:/mnt/xvmp_mnt --name xvmp xvmp:v1.0
sudo docker run -itd -v /tmp/xvmp_mnt:/mnt/xvmp_mnt --name xvmp_devirtualization xvmp_devirtualization:latest
```

```
$ ls
xvmp_devirtualization_docker_latest.tar  xvmp_docker_v1.0.tar
$ mkdir /tmp/xvmp_mnt
$ sudo docker run -itd -v /tmp/xvmp_mnt:/mnt/xvmp_mnt --name xvmp xvmp:v1.0
d81c703d204aa3314cb2ad724c16bada6e9009953c07b467f9cb7f89eaebef2f
$ sudo docker run -itd -v /tmp/xvmp_mnt:/mnt/xvmp_mnt --name xvmp_devirtualization xvmp_devirtualization:latest
449d34183eb23f43a2978c523912823832ef605128eca83b015f7fb93a6f6228
$
```

4. Enter the `xvmp-docker`

```
sudo docker exec -it xvmp /bin/bash
```

5. Enter the home directory of xVMP and set up the environment variables.

   The home directory is `/xvmp`. And you can use `clang -v` to check if the environment variables is set.

```
cd /xvmp/
source ./env.sh
clang -v
```

```
$ sudo docker run -itd -v /tmp/xvmp_mnt:/mnt/xvmp_mnt --name xvmp_devirtualization xvmp_devirtualization:latest
449d34183eb23f43a2978c523912823832ef605128eca83b015f7fb93a6f6228
$ sudo docker exec -it xvmp /bin/bash
root@d81c703d204a:/# cd /xvmp/
root@d81c703d204a:/xvmp# ls
README.md  env.sh  evaluation  llvm  test  tools
root@d81c703d204a:/xvmp# source ./env.sh
/xvmp/
root@d81c703d204a:/xvmp# clang -v
Obfuscator-LLVM clang version 8.0.0  (based on Obfuscator-LLVM 8.0.0)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /xvmp//llvm/obfuscator-llvm-8-install/bin
Found candidate GCC installation: /usr/lib/gcc-cross/i686-linux-gnu/9
Found candidate GCC installation: /usr/lib/gcc/x86_64-linux-gnu/9
Selected GCC installation: /usr/lib/gcc/x86_64-linux-gnu/9
Candidate multilib: .;@m64
Candidate multilib: 32;@m32
Candidate multilib: x32;@mx32
Selected multilib: .;@m64
root@d81c703d204a:/xvmp# cat README.md
# xVMP

## file structure
- `evaluation/` : three evaluations in the paper.
- `llvm/` : install path of xVMP, including xVMP based on original LLVM-8.0 and xVMP based on obfuscator-LLVM-8.0.
- `tools/` : the state-of-the-art virtualization obfuscator Tigress
- `test/` : three simple snippets for you to play
root@d81c703d204a:/xvmp#
```

3.1 [Evaluation 1] Strength of xVMP Obfuscation

> In this experiment, we use an automated de-virtualization tool to analyze the obfuscated programs of XVMP and Tigress to evaluate the strength and effectiveness of XVMP. Since most of the automatic de-virtualization work is outdated and unable to work, we chose Salwan's work and patched the sample to make it work normally. It is based on combining taint tracking, symbolic execution and code simplification.

Because the de-virtualization tool is highly dependent on the environment, we successfully configured it on an Ubuntu16.04-based system and packaged it as a docker image `de-virtualization-docker`.

1. In the `xvmp-docker` docker, we enter `/xvmp/evaluation` and run `python eva12_strength_compatibility.py eva1_strength` to generate the obfuscated binary.

   The obfuscated program will be generated in `/xvmp/evaluation/eva_strength/` .

```
cd /xvmp/evaluation/
python eva12_strength_compatibility.py eva1_strength
```

```
root@d81c703d204a:/xvmp# ls
README.md  env.sh  evaluation  llvm  test  tools
root@d81c703d204a:/xvmp# cd /xvmp/evaluation/
root@d81c703d204a:/xvmp/evaluation# ls
eva12_strength_compatibility.py  eva3_overhead_ccrypt.py  eva3_overhead_leveldb.py  eva3_overhead_openssl.py  eva_compatibility  eva_strength  microbenchmark  out  realworldsample
root@d81c703d204a:/xvmp/evaluation# python eva12_strength_compatibility.py
Please set argument 1
eva12_strength_compatibility.py eva1_strength
eva12_strength_compatibility.py eva2_compatibility_C
eva12_strength_compatibility.py eva2_compatibility_CPP
root@d81c703d204a:/xvmp/evaluation# python eva12_strength_compatibility.py eva1_strength
```

```
./eva_strength//tigress_c/test5_obf.c:204:12: note: 'snprintf' is a builtin with type 'int (char *, unsigned long, const char *, ...)'
11 warnings generated.
Done test5
Files generated in ./eva_strength/
root@d81c703d204a:/xvmp/evaluation# ls ./eva_strength/
original  tigress_c  tigress_out  xvmp_out
root@d81c703d204a:/xvmp/evaluation# ls -R ./eva_strength/
./eva_strength/:
original  tigress_c  tigress_out  xvmp_out

./eva_strength/original:
test1_ori  test2_ori  test3_ori  test4_ori  test5_ori  test6_ori

./eva_strength/tigress_c:
test1_obf.c  test2_obf.c  test3_obf.c  test4_obf.c  test5_obf.c  test6_obf.c

./eva_strength/tigress_out:
test1_tigress  test2_tigress  test3_tigress  test4_tigress  test5_tigress  test6_tigress

./eva_strength/xvmp_out:
test1_xvmp  test2_xvmp  test3_xvmp  test4_xvmp  test5_xvmp
root@d81c703d204a:/xvmp/evaluation#
```

2. Then, we copy the generated files to the shared directory.

```
cp -r ./eva_strength /mnt/xvmp_mnt/
```

```
root@d81c703d204a:/xvmp/evaluation# ls /mnt/xvmp_mnt/
root@d81c703d204a:/xvmp/evaluation# cp -r ./eva_strength /mnt/xvmp_mnt/
root@d81c703d204a:/xvmp/evaluation# ls /mnt/xvmp_mnt/
eva_strength
root@d81c703d204a:/xvmp/evaluation#
```

3. We open a new terminal and enter the `de-virtualization-docker`

```
sudo docker exec -it xvmp_devirtualization /bin/bash
```

```
$ sudo docker ps
[sudo] password for xgx:
CONTAINER ID   IMAGE                          COMMAND       CREATED          STATUS          PORTS     NAMES
449d34183eb2   xvmp_devirtualization:latest   "/bin/bash"   35 minutes ago   Up 35 minutes             xvmp_devirtualization
d81c703d204a   xvmp:v1.0                      "/bin/bash"   36 minutes ago   Up 36 minutes             xvmp
$ sudo docker exec -it xvmp_devirtualization /bin/bash
root@449d34183eb2:/#
```

4. Then, we enter the root directory of the de-virtualization tool.

```
cd /root/de-virtualization/
```

```
root@449d34183eb2:/# cd /root/de-virtualization/
root@449d34183eb2:~/de-virtualization# ls
CMakeLists.txt  deobfuscated_binaries  inscount.out      metrics  patch_endbr.py                    scripts               solve-vm.py          tigress-2.1
Dockerfile      dep                    inscount.py       misc     pin-2.14-71313-gcc.4.4.7-linux.tar.gz  solve-md5.py          solve-vms.sh         tigress-challenges
README.md       eva_strength.py        llvm_expressions  mytest   samples                           solve-vm-multiple-br.py  symbolic_expressions  virtualization
root@449d34183eb2:~/de-virtualization#
```

5. We de-virtualize the **Tigress** obfuscated binary in the shared directory.

```
python3 ./eva_strength.py TIGRESS
```

As shown in the following screenshot, `/mnt/xvmp_mnt/eva_strength/tigress_out/test1_tigress` has been de-virtualized in `deobfuscated_binaries/test1_tigress.deobfuscated` .

```
root@449d34183eb2:~/de-virtualization# python3 ./eva_strength.py
Please assign argv1 as an OPTION:
./eva_strength.py INSCNT
./eva_strength.py TIGRESS
./eva_strength.py XVMP
root@449d34183eb2:~/de-virtualization# python3 ./eva_strength.py TIGRESS
Handling /mnt/xvmp_mnt/eva_strength/tigress_out/test1_tigress
Done
Running command: timeout 36000 ./solve-vm.py /mnt/xvmp_mnt/eva_strength/tigress_out/test1_tigress
[+] Loading 0x400040 - 0x4002a8
[+] Loading 0x4002a8 - 0x4002c4
[+] Loading 0x400000 - 0x4004d8
[+] Loading 0x401000 - 0x401735
[+] Loading 0x402000 - 0x402250
[+] Loading 0x403e00 - 0x404070
[+] Loading 0x403e10 - 0x403ff0
[+] Loading 0x4002c4 - 0x4002e4
[+] Loading 0x4020d0 - 0x40211c
[+] Loading 0x000000 - 0x000000
[+] Loading 0x403e00 - 0x404000
[+] Hooking printf
[+] Hooking memcpy
[+] Hooking strtoul
[+] Hooking __libc_start_main
[+] Starting emulation.
[+] __libc_start_main hooked
[+] argv[0] = /mnt/xvmp_mnt/eva_strength/tigress_out/test1_tigress
[+] argv[1] = 1234
[+] strtoul hooked
[+] Symbolizing the strtoul return
[+] memcpy hooked
[+] printf hooked
3735927357
[+] Slicing end-point user expression
[-] Instruction not supported: 0x40108e: hlt
[+] Instruction executed: 1820
[+] Unique instruction executed: 276
[+] PC len: 1
[+] Emulation done.
[+] Asking for a new input...
[+] No model found! Opaque predicate?
[+] Generating symbolic_expressions/test1_tigress.py
[+] Converting symbolic expressions to an LLVM module...
[+] LLVM module wrote in llvm_expressions/test1_tigress.ll
[+] Recompiling deobfuscated binary...
[+] Deobfuscated binary recompiled: deobfuscated_binaries/test1_tigress.deobfuscated
------------------------------------------------------------------------
-> test1_tigress
        Instructions executed: 1820
        Unique Instructions executed: 276
        Functions simulated: 4
        Time of analysis: 0.138931 seconds
run_cmd:  0
============================================================
Handling /mnt/xvmp_mnt/eva_strength/tigress_out/test2_tigress
Done
Running command: timeout 36000 ./solve-vm.py /mnt/xvmp_mnt/eva_strength/tigress_out/test2_tigress
[+] Loading 0x400040 - 0x4002a8
[+] Loading 0x4002a8 - 0x4002c4
[+] Loading 0x400000 - 0x4004d8
[+] Loading 0x401000 - 0x401785
[+] Loading 0x402000 - 0x402260
[+] Loading 0x403e00 - 0x404078
```

`test5_tigress` may cause about 700~4000 seconds and `test6_tigress` is timeout for 10h.

If you want to **stop** the script, `Ctrl+C` may not work, you need to open a new terminal and kill the process.

```
$ sudo docker exec -it xvmp_devirtualization /bin/bash
root@449d34183eb2:/# cd
root@449d34183eb2:~# cd de-virtualization/
root@449d34183eb2:~/de-virtualization# ps -ef | grep python
root          33      16  0 13:25 pts/1    00:00:00 python3 ./eva_strength.py TIGRESS
root          84      83 99 13:25 pts/1    00:05:14 python2 ./solve-vm.py /mnt/xvmp_mnt/eva_strength/tigress_out/test5_tigress
root         105      86  0 13:30 pts/2    00:00:00 grep --color=auto python
root@449d34183eb2:~/de-virtualization# kill 33
root@449d34183eb2:~/de-virtualization# kill 84
```

6. We also can de-virtualize the Tigress obfuscated binary in the shared directory.

```
python3 ./eva_strength.py XVMP
```

`test1_xvmp` may cause about 360 seconds and other are timeout for 10h.

7. Finally, we count the executed instructions in the original program, **Tigress** obfuscated and xVMP obfuscated binary.

   You also could count the executed instructions of de-virtualized binary by running `./inscount.py ./deobfuscated_binaries/test1_tigress.deobfuscated` .

   It is worth noting that if you want to use `inscount.py` or `solve-vm.py` to process your program, you need to use `patch_endbr.py` to process instructions in the binary program that cannot be processed by the de-virtualization tool.

3.2 [Evaluation 2] Compatibility

> In this experiment, we evaluate XVMP's compatibility with different types of source code, different architectures, and traditional obfuscation.
>
> We set up three C++ programs in the microbenchmark, which were implemented using classes, inheritance and polymorphism, and function overloading in object-oriented programming. XVMP can effectively protect the C++ functions which annotated in the source code, and the virtualized functions have the same semantics.
>
> To test the compatibility of multiple architectures, we obfuscated all C and C++ files in microbenchmark with XVMP, compile them into executable files of X86, X64, ARM32, and ARM64 architectures, and use Qemu [9] to check the program behavior. The results show that the obfuscated executable files generated by all the test files in the microbenchmark can run normally under these four architectures.
>
> To prove that XVMP is compatible with the traditional obfuscation scheme based on LLVM IR, we set the effective order in the pass manager, so that the obfuscation option in Obfuscator-LLVM can be used to protect the XVMP interpreter. We obfuscated the program with options of control flow flattening and bogus control flow. The results show that the program protected by XVMP can still run normally after being protected by Obfuscator-LLVM.

1. First, we can test the compatibility of the C programs in microbenchmark with Obfuscator-LLVM, and the compatibility with x64, x86, arm32, arm64 architecture.

```
python ./eva12_strength_compatibility.py eva2_compatibility_C
```

```
root@d81c703d204a:/xvmp/evaluation# python ./eva12_strength_compatibility.py
Please set argument 1
./eva12_strength_compatibility.py eva1_strength
./eva12_strength_compatibility.py eva2_compatibility_C
./eva12_strength_compatibility.py eva2_compatibility_CPP
root@d81c703d204a:/xvmp/evaluation# python ./eva12_strength_compatibility.py eva2_compatibility_C
```

In the end, you can see that the five test files all get the same output in the original program, and the executable files are protected by Obfuscator-LLVM+xVMP in the x64, x86, arm32, arm64 architecture.

```
Checking microbenchmark test1
Test result: test1 OK
['3735837871', '3735837871', '3735837871', '3735837871', '3735837871']
Checking microbenchmark test2
Test result: test2 OK
['2576923054', '2576923054', '2576923054', '2576923054', '2576923054']
Checking microbenchmark test3
Test result: test3 OK
['647', '647', '647', '647', '647']
Checking microbenchmark test4
Test result: test4 OK
['61604132', '61604132', '61604132', '61604132', '61604132']
Checking microbenchmark test5
Test result: test5 OK
['1935567302', '1935567302', '1935567302', '1935567302', '1935567302']
root@d81c703d204a:/xvmp/evaluation#
```

2. We can also see the test results of the CPP file by following.

```
python ./eva12_strength_compatibility.py eva2_compatibility_CPP
```

```
Test result: cpp_test1 OK
['3735837871', '3735837871']
Test result: cpp_test2 OK
['864192', '864192']
Test result: cpp_test3 OK
['Int: 123456\nAddress: 0x401270\nString: Hello C++\n123456', 'Int: 123456\nAddress: 0x401270\nString: Hello C++\n123456']
root@d81c703d204a:/xvmp/evaluation#
```

3.3 [Evaluation 3] Overhead

> The test set contains 3 real-world software, including the command-line encryption program ccrypt-1.11, the widely used cryptography library OpenSSL-1.0.2u, and the database leveldb-1.23 written in C++, and we use them to evaluate the overhead of XVMP.
> We test the overhead of XVMP using three real-world pieces of software. For OpenSSL, we tested the average overhead of one million 1K-bytes block encryption by AESCBC-128 using

> OpenSSL speed. We use ccrypt to encrypt 1MB of random data and use c_test in the leveldb test suite.
>
> The results are shown in Table II, We select one function at a time for virtualization obfuscation, including two hotspot functions and three non-hotspot functions. We can see that when protecting the hotspot loop functions xKeyAddition and CRYPTO_cbc128_encrypt in ccrypt and OpenSSL, the overhead will be tens to hundreds of times. However, in real-world applications, virtualization obfuscation is more used to protect key generation and key logic, such as the function hashstring in ccrypt that calculates symmetric keys through passwords. Therefore, we protect three non-hotspot but critical functions, in which case the overhead is less than 2%. Because Tigress does not support C++, it cannot protect leveldb.
>
> We tested the code bloating effect brought by XVMP. Because the interpreter is embedded into the program, the program size will expand, but the actual code size increased by XVMP is less than 40K.

3.3.1 Realworld application: ccrypt

1. First, we need to decompress the gzip file.

```
cd /xvmp/evaluation/realworldsample
tar xzvf ./ccrypt-1.11.tar.gz
cd /xvmp/evaluation/
```

```
root@d81c703d204a:/xvmp/evaluation# ls realworldsample/
ccrypt-1.11.tar.gz  leveldb-1.23.tar.gz  openssl-1.0.2u.tar.gz
root@d81c703d204a:/xvmp/evaluation# cd realworldsample/
root@d81c703d204a:/xvmp/evaluation/realworldsample# tar xzvf ./ccrypt-1.11.tar.gz
```

2. We can run `eva3_overhead_ccrypt.py` to build and test ccrypt.
   We first protect the hotspot function `xKeyAddition` .

```
root@d81c703d204a:/xvmp/evaluation# python3 eva3_overhead_ccrypt.py xKeyAddition
```

```
root@d81c703d204a:/xvmp/evaluation# python3 eva3_overhead_ccrypt.py
Please set argument 1: to-be-protect function
eva3_overhead_ccrypt.py xKeyAddition
eva3_overhead_ccrypt.py hashstring
root@d81c703d204a:/xvmp/evaluation# python3 eva3_overhead_ccrypt.py xKeyAddition
```

```
 Running command:  out/ori/ccrypt -e ./test.txt --key 1234567890
ori 0.09668779373168945
 Running command:  out/ori/ccrypt -d ./test.txt.cpt --key 1234567890
 Running command:  out/vmp/ccrypt -e ./test.txt --key 1234567890
vmp 4.198898553848267
 Running command:  out/vmp/ccrypt -d ./test.txt.cpt --key 1234567890
 Running command:  out/tigress/ccrypt -e ./test.txt --key 1234567890
tigress 0.49912595748901367
 Running command:  out/tigress/ccrypt -d ./test.txt.cpt --key 1234567890
```

3. Then, we protect the non-hotspot function `hashstring` .

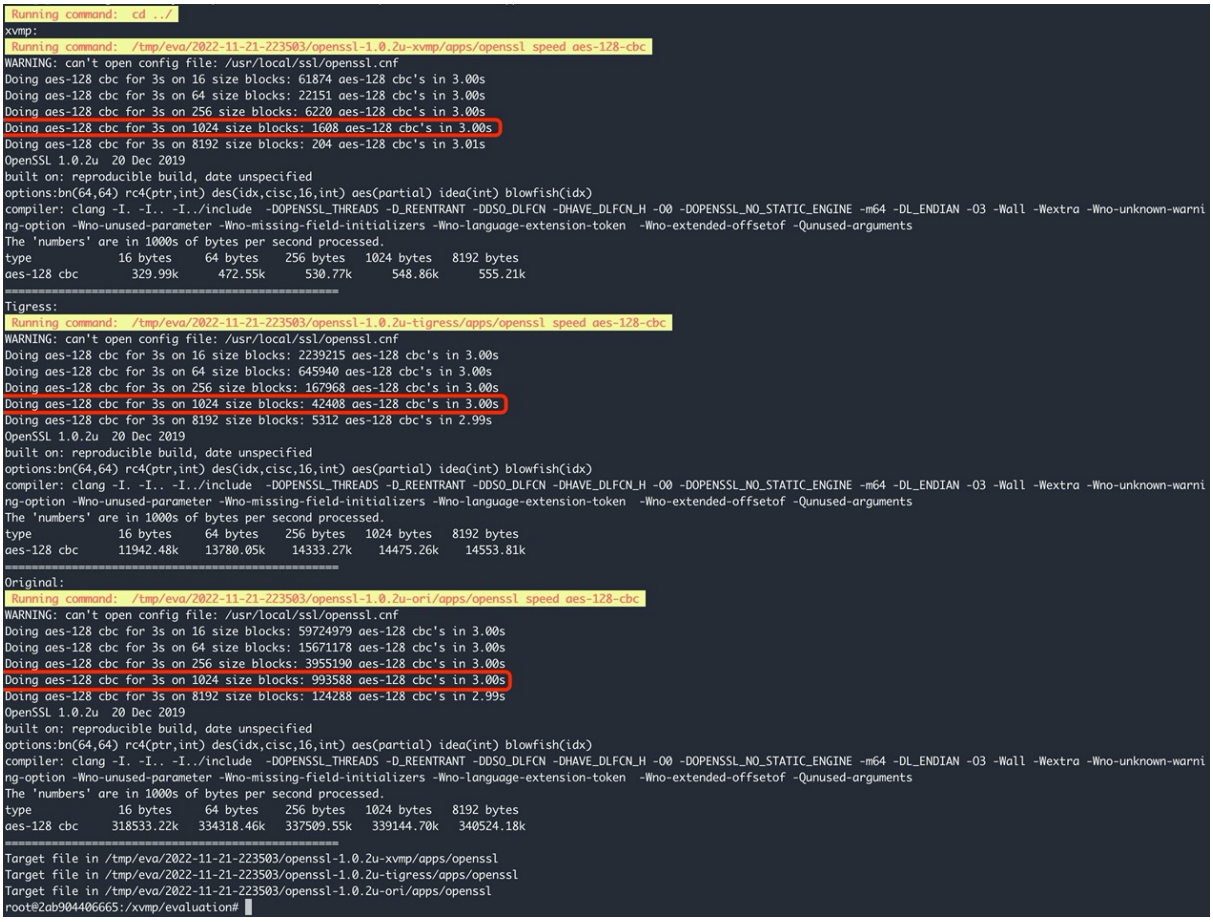
```
root@d81c703d204a:/xvmp/evaluation# python3 eva3_overhead_ccrypt.py hashstring
```

3.3.2 Realworld application: OpenSSL

1. We run `eva3_overhead_openssl.py` and select the hotspot function `CRYPTO_cbc128_encrypt`

```
root@d81c703d204a:/xvmp/evaluation# python eva3_overhead_openssl.py CRYPTO_cbc128_encrypt
```

```
root@d81c703d204a:/xvmp/evaluation# python eva3_overhead_openssl.py
Please set argument 1: to-be-protect function
eva3_overhead_openssl.py CRYPTO_cbc128_encrypt
eva3_overhead_openssl.py idea_set_encrypt_key
root@d81c703d204a:/xvmp/evaluation# python eva3_overhead_openssl.py CRYPTO_cbc128_encrypt
```



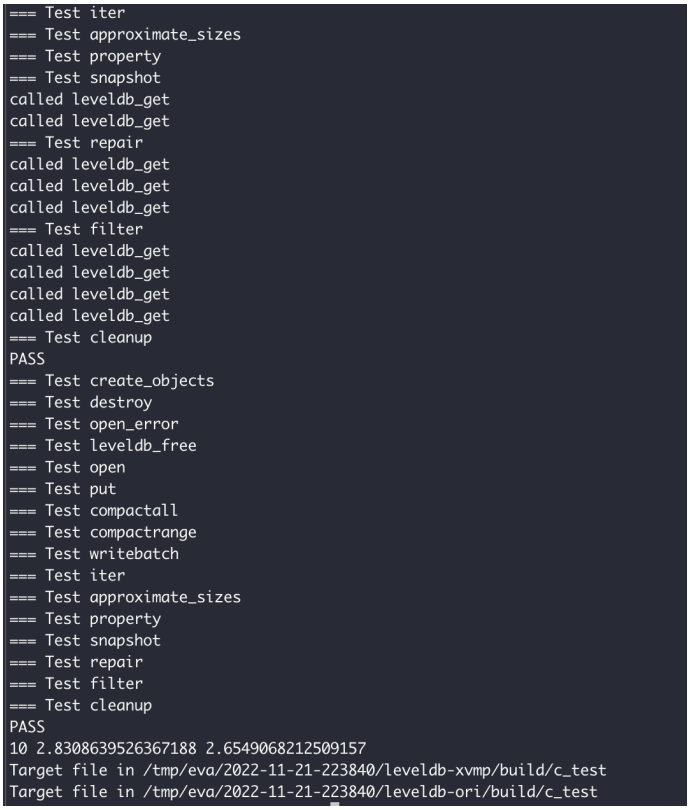
2. Then, we run `eva3_overhead_openssl.py` and select the non-hotspot function `idea_set_encrypt_key`

```
root@d81c703d204a:/xvmp/evaluation# python eva3_overhead_openssl.py CRYPTO_cbc128_encrypt
```

3.3.3 Realworld application: Level-DB

1. We run `eva3_overhead_leveldb.py` to test level-db.

```
root@d81c703d204a:/xvmp/evaluation# python eva3_overhead_leveldb.py
```



4. Tutorial for Playing with xVMP

Under the `/xvmp/test` directory, we prepared three simple programs for you to play with xVMP.

You can use the following command to build test cases.

For example, in `/xvmp/test/test3`

- Original:

  `clang test3_main.c test3_foo.c -o test3`

- xVMP
  - single xvmp:

    `clang -DENABLE_XVMP test3_main.c test3_foo.c -o test3_xvmp`

  - xvmp+ollvm:

    `clang -DENABLE_XVMP -DENABLE_OLLVM test3_main.c test3_foo.c -o test3_xvmp_ollvm`

- Tigress

  Because Tigress can only accept a whole program single C file as input, which means that the original compilation process needs to be broken and is very unfriendly to developers.
  And you need to merge all C files into one as follows.

  `tigress-merge test3_main.c test3_foo.c --out=test3_tigress.c`

```
tigress --Environment=x86_64:Linux:Gcc:4.6 --Seed=0  --Transform=Virtualize   --VirtualizeDispatch=interpolation --Functions=foo --VirtualizeEncodeByteArray=true test3_tigress.c --out=test3_tigress_obf.c
```

```
clang test3_tigress_obf.c -o test3_tigress_obf
```

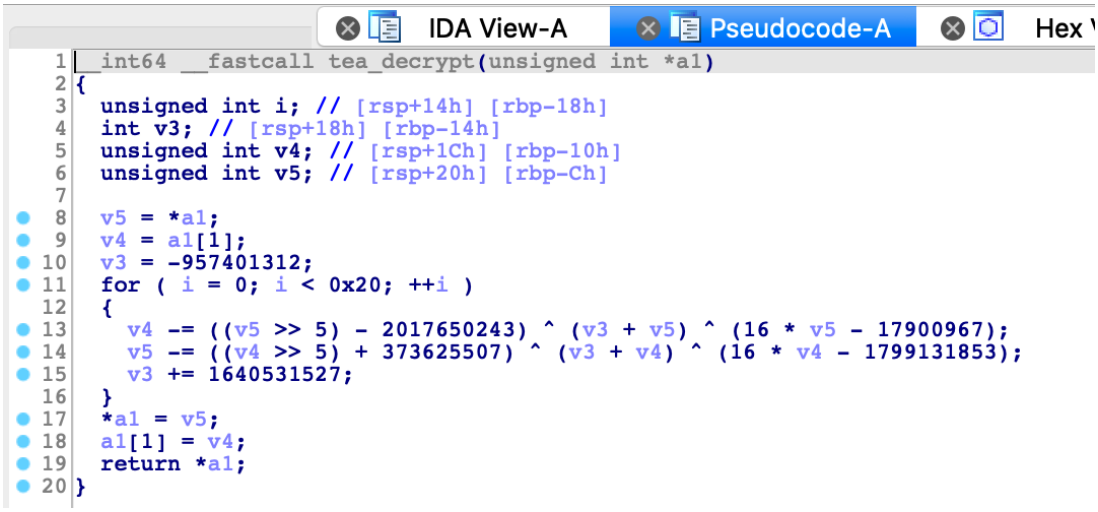
If you directly obfuscate `test3_foo.c` by Tigress, a new `main` function will be introduced in the output file of `test_foo.c`.

5. A Reverse Engineering Perspective

We use `/xvmp/test/test2` as an example.

Original

Unprotected programs can be easily analyzed for original logic.



Tigress

We can directly look at `test2_tigress.c` or use IDA for analysis.

In IDA, we can clearly see that the virtualized code only XORs a constant for protection, which is very easy to be broken.

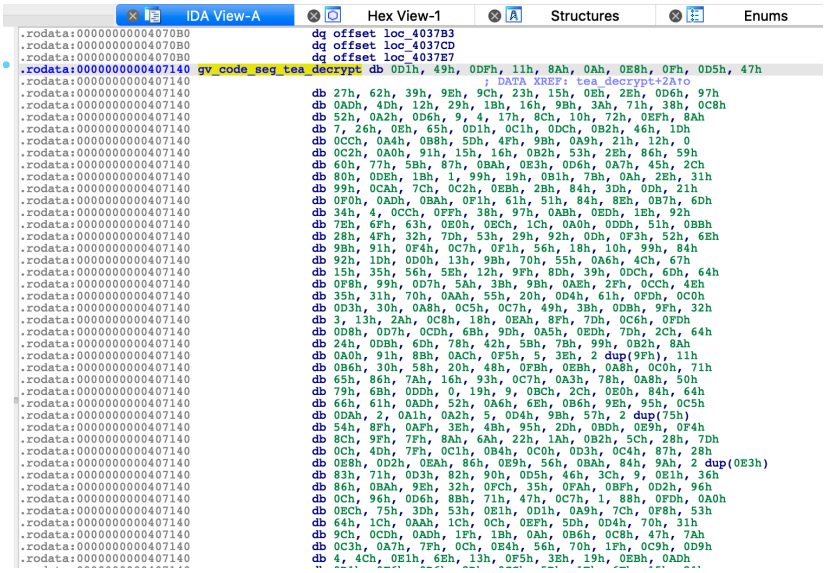
The length of virtualized function is 0x7CA.

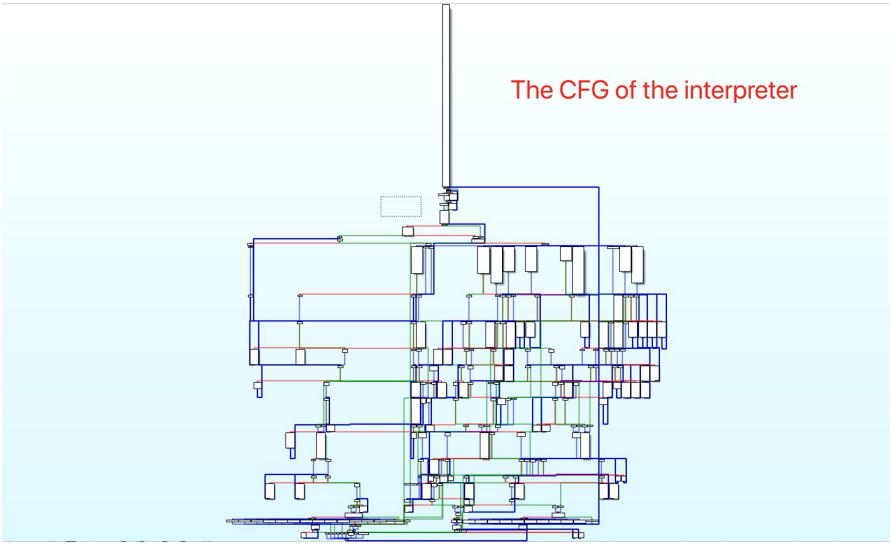
```
void __fastcall tea_decrypt(__int64 a1)
{
  int i; // [rsp+1Ch] [rbp-484h]
  int v2; // [rsp+20h] [rbp-480h]
  int v3; // [rsp+24h] [rbp-47Ch]
  int v4; // [rsp+28h] [rbp-478h]
  unsigned __int8 dest[336]; // [rsp+30h] [rbp-470h] BYREF
  unsigned __int8 v6[464]; // [rsp+180h] [rbp-320h] BYREF
  char *v7; // [rsp+350h] [rbp-150h]
  char *v8; // [rsp+358h] [rbp-148h]
  char v9; // [rsp+360h] [rbp-140h] BYREF
  __int64 v10; // [rsp+498h] [rbp-8h]

  v10 = a1;
  memcpy(dest, "\t", sizeof(dest));
  for ( i = 0; i < 452; ++i )
    v6[i] = l_tea_decrypt__array[i] ^ 0xA0;      // virtualized code just xor a constant
  v8 = &v9;
  v7 = (char *)v6;
  v4 = 0;
  v3 = 20;
  while ( v3 >= v4 )
  {
    v2 = (v3 - v4) * (v6[0] - dest[16 * v4]) / (dest[16 * v3] - dest[16 * v4]) + v4;
    if ( dest[16 * v2] >= (int)v6[0] )
    {
      if ( dest[16 * v2] <= (int)v6[0] )
        v3 = -1;
      else
        v3 = v2 - 1;
    }
    else
    {
      v4 = v2 + 1;
    }
  }
  __asm { jmp     rax }                          // vmcode dispatcher
}
```

xVMP

The virtualization code of program protected by xVMP is more hardened and protected, and the interpreter is more complicated.

The length of function `vm_interpreter_tea_decrypt` is 0x5A71.

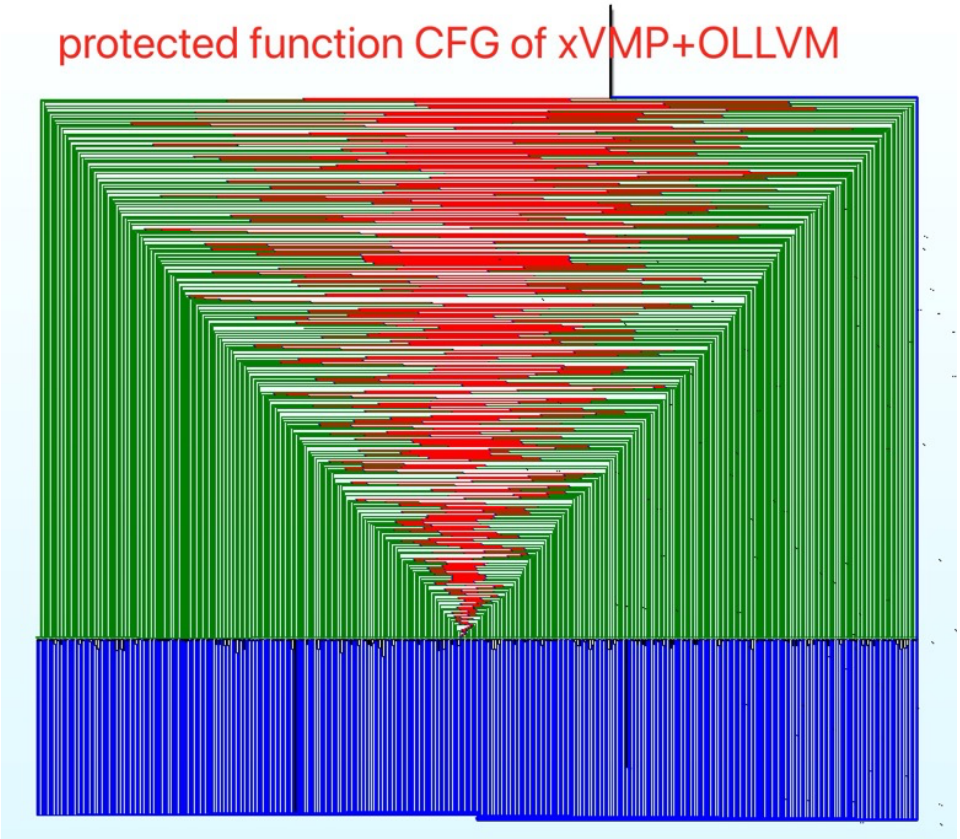




xVMP+Obfuscator-LLVM

The length of function `vm_interpreter_tea_decrypt` is 0x149F2.

The protection of xVMP+oLLVM makes it very difficult for attackers to be reversed.



6. To be Improved

xVMP currently supports most situations, but there are still features need to be improved.

- Currently we only support the most basic LLVM IR instructions, and some uncommon IR instructions such as `select` and `insertelement` are not yet supported.
- Please do not add the `-g` option, because it will introduce `llvm.dbg.declare`, `llvm.dbg.value` and other situations that are not currently handled.
- If you need to compile with `-O3`, please use `-O0 -Xclang -disable-O0-optnone` to compile to IR first, and then use `-O3` to compile to binary after obfuscating the IR.
- Since the translator involves some global variables, exceptions may occur when protecting functions with a dominance relationship. Currently, xVMP can support obfuscating multiple functions that do not have a dominance relationship.

We will continue to improve the work in the future.

If you have any questions, you can send an email to our anonymous mailbox. amy104249@gmail.com