Structure:

/src

1. /attack
   1. /basic
   2. /frequency
2. /defense

/util

1. /fs-hasher
2. /leveldb
3. /synthetic-trace

Environment:

This toolkit is built on Ubuntu 14.04 LTS with g++ version 4.8.1.

1. OpenSSL (<https://www.openssl.org/source/openssl-1.0.2a.tar.gz>)
2. LevelDB (<https://github.com/google/leveldb/archive/master.zip>)
3. Fs-hasher (http://tracer.filesystems.org/fs-hasher-0.9.4.tar.gz)

Preparation:

1. Change the $training\_trace in each script to where your fsl trace are.
2. Copy the target trace to the $target\_trace path.
3. Copy the fs-hasher and leveldb folder from /util into the working directory
4. Compile fs-hasher and leveldb inside their folders.

/util/synthetic-trace

Tools for generating the synthetic VM trace

Usage

Necessary tools: qemu-kvm, kpartx

1. Create a qcow2 virtual machine image in directory $vmdir

2. Create a mount point directory 'root' in $vmdir

3. Untar the package file

4. Go into the untarred directory: $cd synthetic\_trace/

5. Create a soft link to the mount point directory: $ln -s $vmdir/root synthetic\_fs/

6. Modify the variable $vmdir and $curdir in the 'backup\_routine.sh' file with absolute path of the directory where your qcow2 image file resides and the untarred directory, respectively

7. To start the generation, using command './backup\_routine.sh'

Parameters

You can configure following parameters in backup\_routine.sh:

a: percentage of files in target file system

b: percentage of file content to modify

M: total size of newly created files (KB). Make sure the size is not too large. Generally, the value should be smaller than ${available space of target file system}/($vers\*5\*1024)

vers: number of full backup versions that you want to generate. (96 indicates weekly full backups covering 2 year, which is the same as Lillibridge's work)

/src/attack/basic

Basic Attack (Sec. IV.A)

Basic frequency analysis program for given hash file

./FSL-ANALYSIS [hashfile]

Output the chunk frequency according to the appearance frequency in the descending order, see algorithm 1 in Sec. IV.A

/src/attack/frequency

Frequency Attack

PARSE: (Sec. IV.B)

First, it collects four sets of chunks for (C,M): (i) LC that includes the left ciphertext chunks of C; (ii) LM that includes the left plaintext chunks of M; (iii) RC that includes the right ciphertext chunks of C and (iv) RM that includes the right plaintext chunks of M. Thus, there exists an inherent relationship between LC and LM (resp. RC and RM ), in that the plaintext chunks in LM (resp. RM ) are potentially mapped to the ciphertext chunks in LC (resp. RC).

[Normal parse] for inference rate analysis:

./fsl-count-local [hashfile] [frequencyDB] [leftDB] [rightDB]

Parse the input hash file C into F\_C and P\_C (both left and right),

which can be used on both training set or target set.

/src/defense

[Defense parse] for mitigation analysis:

./mh-count [encryptedHash] [frequencyDB] [leftDB] [rightDB] [originalHash] [relateDB]

Parse the encrypted Hashfile into F\_C and P\_C (both left and right),

also store the mapping between encrypted hash and original hash in relate DB.

========================================

FREQUENCY-ANALYSIS: (Sec. IV.B)

Then, the attack employs a special variant of frequency analysis on the chunks in LC and LM (resp. RC and RM ), based on their co-occurrences with C and M. Specifically, we augment frequency analysis with an adjustable parameter t, meaning that it just maps top-t of the ciphertext chunks that have the most co-occurrences with C to the corresponding plaintext chunks that have the most co-occurrences with M. The rationale is the frequencies of top-frequent chunks are often distinct and dramatically high (see Figure 1 for an example), so that the frequency ranks of these top-frequent chunks are expected to be stable throughout multiple backups. We parse the output mapping as a set of new ciphertext- plaintext chunk pairs and add them into the inferred set for future iterations. The iteration terminates when all pairs in the inferred set have been iterated.

[Normal analysis] for inference rate:

./db-stat-local [param1] [param2] [param3] [param4]

[F\_C] [P\_C\_left] [P\_C\_right]

[F\_M] [P\_M\_left] [P\_M\_right]

param1: initial truth set size.

param2: top frequency set size.

param3: the inference queue size.

param4: the leak rate

/src/defense

[Defense analysis] for inference rate:

./mh-stat [param1]

[F\_C] [P\_C\_left] [P\_C\_right]

[F\_M] [P\_M\_left] [P\_M\_right] [relateDB]

work with mh-count which creates the relateDB that connects F\_C and F\_M

param1: the leak rate

set other parameters in program.

1: initial truth set size.

2: top frequency set size.

3: the inference queue size.

========================================

fsl-count-local.cc

(fsl-count-local excutable)

analysis module in the paper. parse trace into three dbs.

@input argv[1]: hash file created by "hf-stat -h [HASHFILE]"

@output argv[2]: leveldb path for main chunk frequency db.

@output agrv[3]: leveldb path for left adjacent db.

@output argv[4]: leveldb path for right adjacent db.

========================================

db-dedup.cc

(db-dedup excutable)

test deduplication ratio of a particular hashfile input.

@input argv[1]: target hashfile to be tested.

@input argv[2]: original leveldb path, the dedup ratio is based on the db.

========================================

db-print-local.cc

(db-stat-local excutable)

inference module in the paper. infer chunks from 6 dbs.

@input argv[1]: t', the initial truth set size

@input argv[2]: t, the top frequence size in each step

@input argv[3]: q, the inference queue size

@input argv[4]: leakage rate, the percentage of leaked chunk

@input argv[5]: main frequency db path for training set

@input argv[6]: left db path for training set

@input argv[7]: right db path for training set

@input argv[8]: main frequency db path for target set

@input argv[9]: left db path for target set

@input argv[10]: right db path for target set

see example in attack\_param.sh

========================================

/src/defense

kminhash.cc

(k-minhash excutable)

defence encryption simulation program. transfer hashfile into encrypted hash

set K for kminhash in program K\_MINHASH

set segment size in program SEG\_SIZE, SEG\_MIN, SEG\_MAX

@input argv[1]: input hashfile

@output: a hashfile contains encrypted chunk hashes

========================================

/src/defense

mh-count.cc

(mh-count excutable)

defence version of analysis module. record plain chunk and cipher chunk information in an extra db call "db-relate"

@input argv[1]: hash file created by "k-minhash"

@output argv[2]: leveldb path for main chunk frequency db.

@output agrv[3]: leveldb path for left adjacent db.

@output argv[4]: leveldb path for right adjacent db.

@input argv[5]: original hash file created by "hf-stat"

@output argv[6]: leveldb path for recording plain and cipher chunk relation

========================================

/src/defense

mh-print.cc

(mh-stat excutable)

defence version of inference module. the verify procedure contains lookup in the "db-relate" to see if a specific cipher chunk is derived from a plain chunk.

@set: t, TH\_K in program. the top frequence size in each step

@set: q, QUEUE\_LIMIT in program. the inference queue size

@input argv[1]: leakage rate, the percentage of leaked chunk

@input argv[2]: main frequency db path for training set

@input argv[3]: left db path for training set

@input argv[4]: right db path for training set

@input argv[5]: main frequency db path for target set

@input argv[6]: left db path for target set

@input argv[7]: right db path for target set

@input argv[8]: relate db for plain cipher information

see example in rerun.sh