Ransomware Testing Framework

Overview of Ransomware Pattern

Report

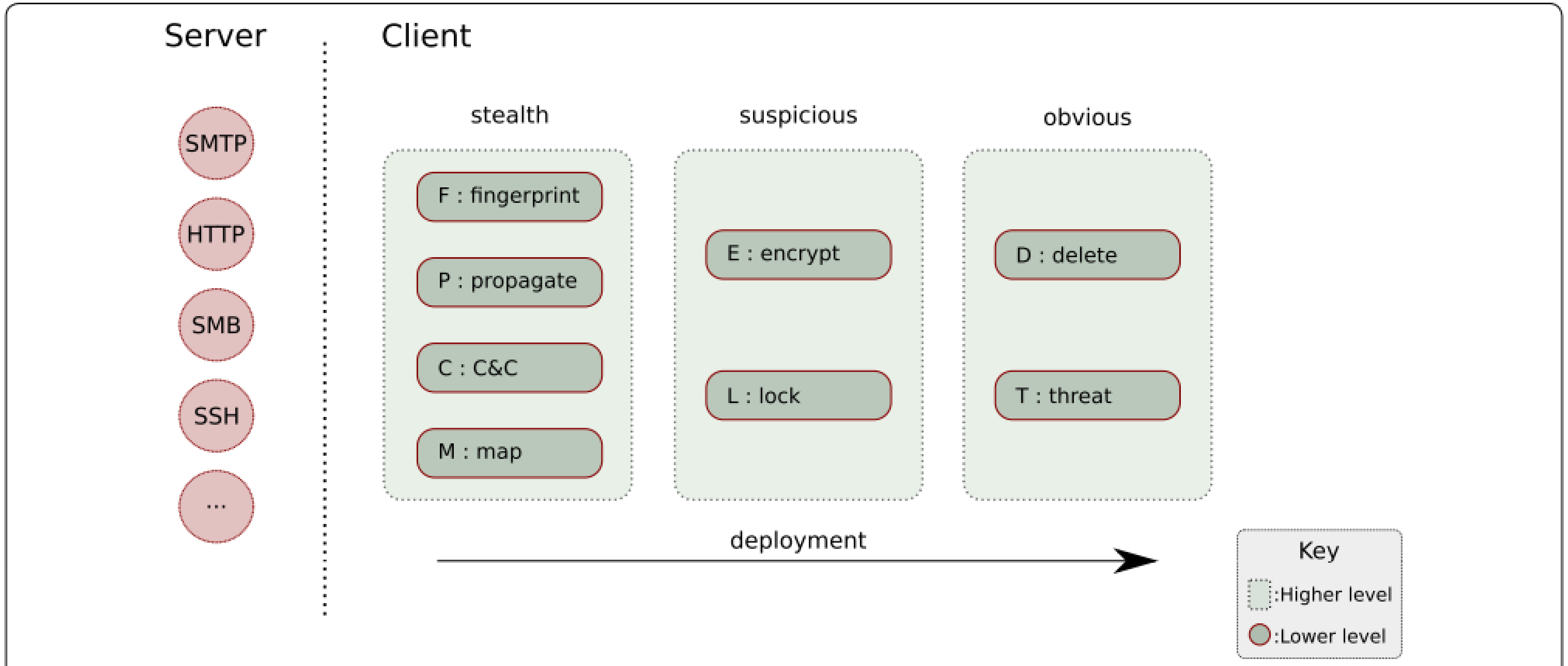


Fig. 2 Predictive model of ransomware deployment methods

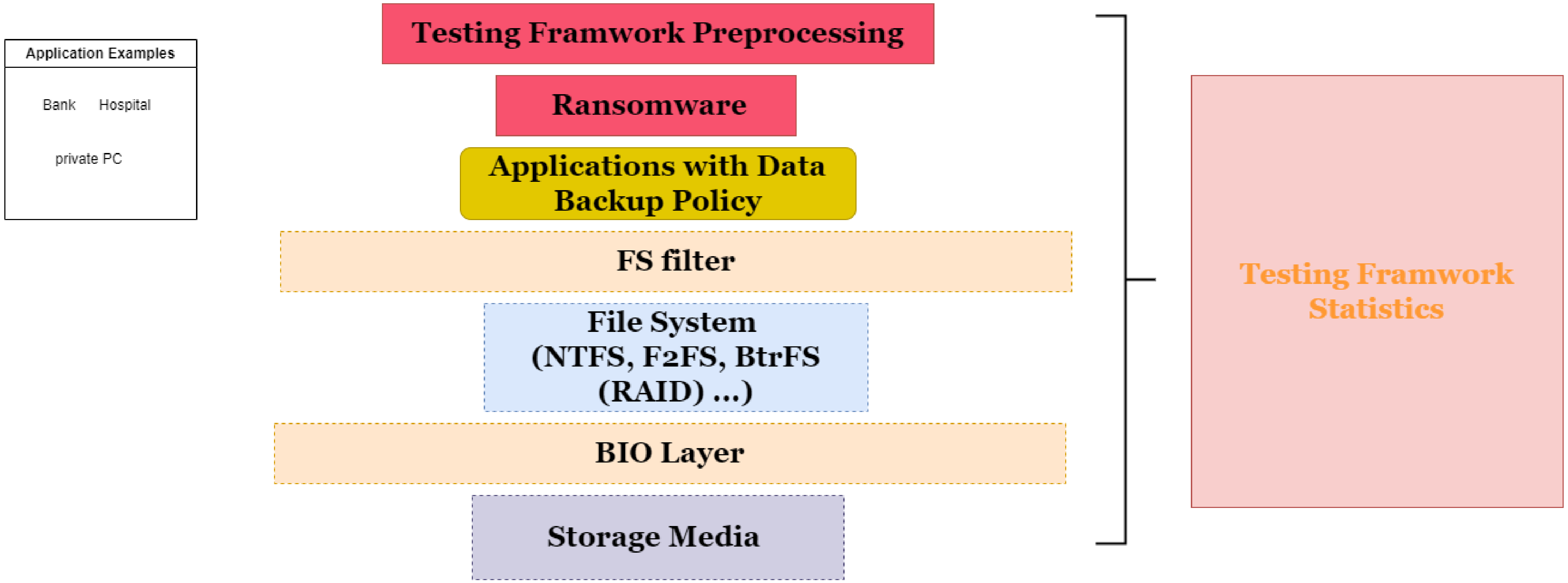
Testing Framework Structure

For banks, hospitals, private PC, etc. they store files in their system (our target system).

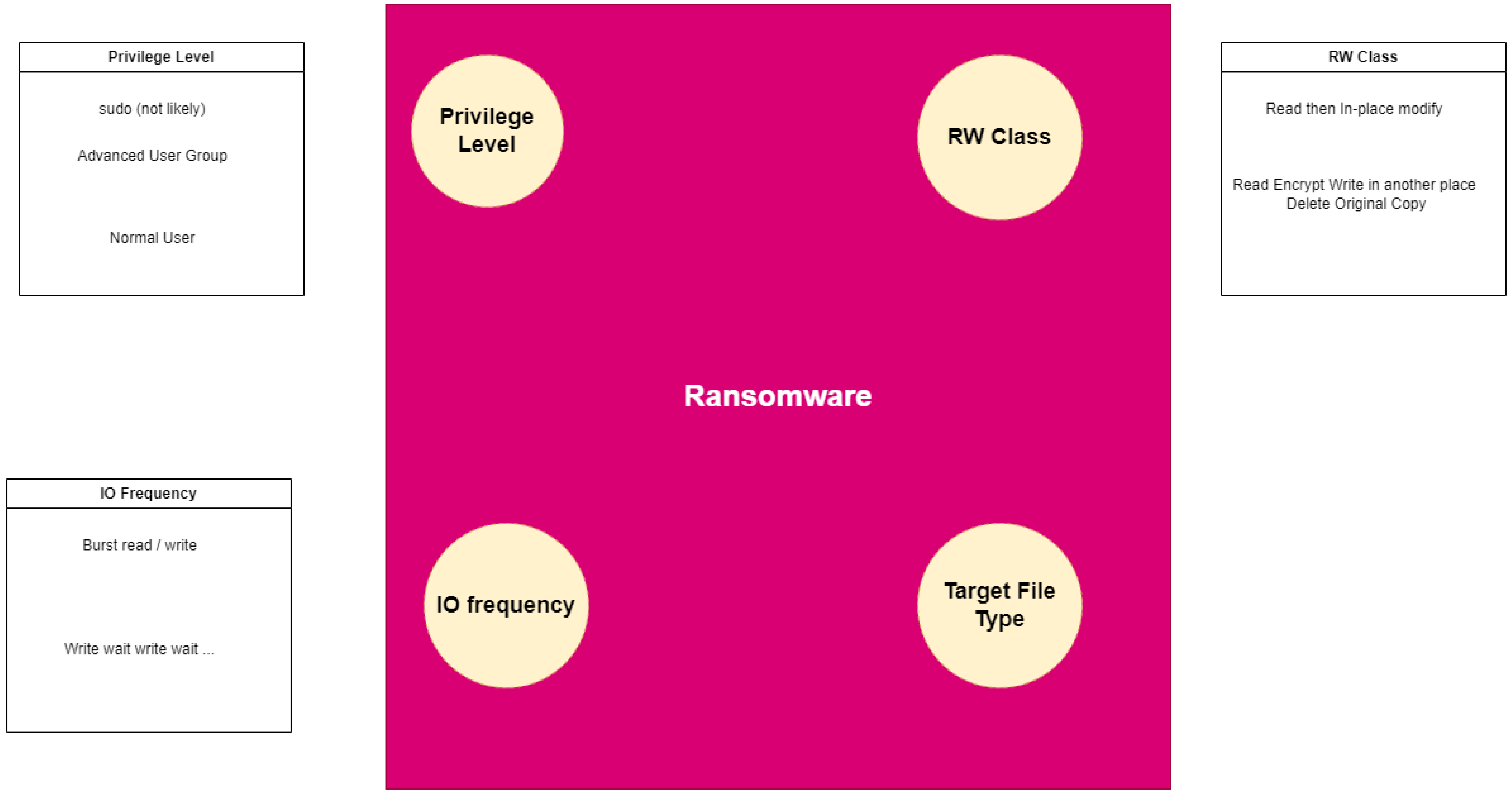
Ransomware reads files in our target system, encrypt it, then overwrite them(in-place or delte then create new copies).

The testing framework detects how susceptible the target system is to ransomware.

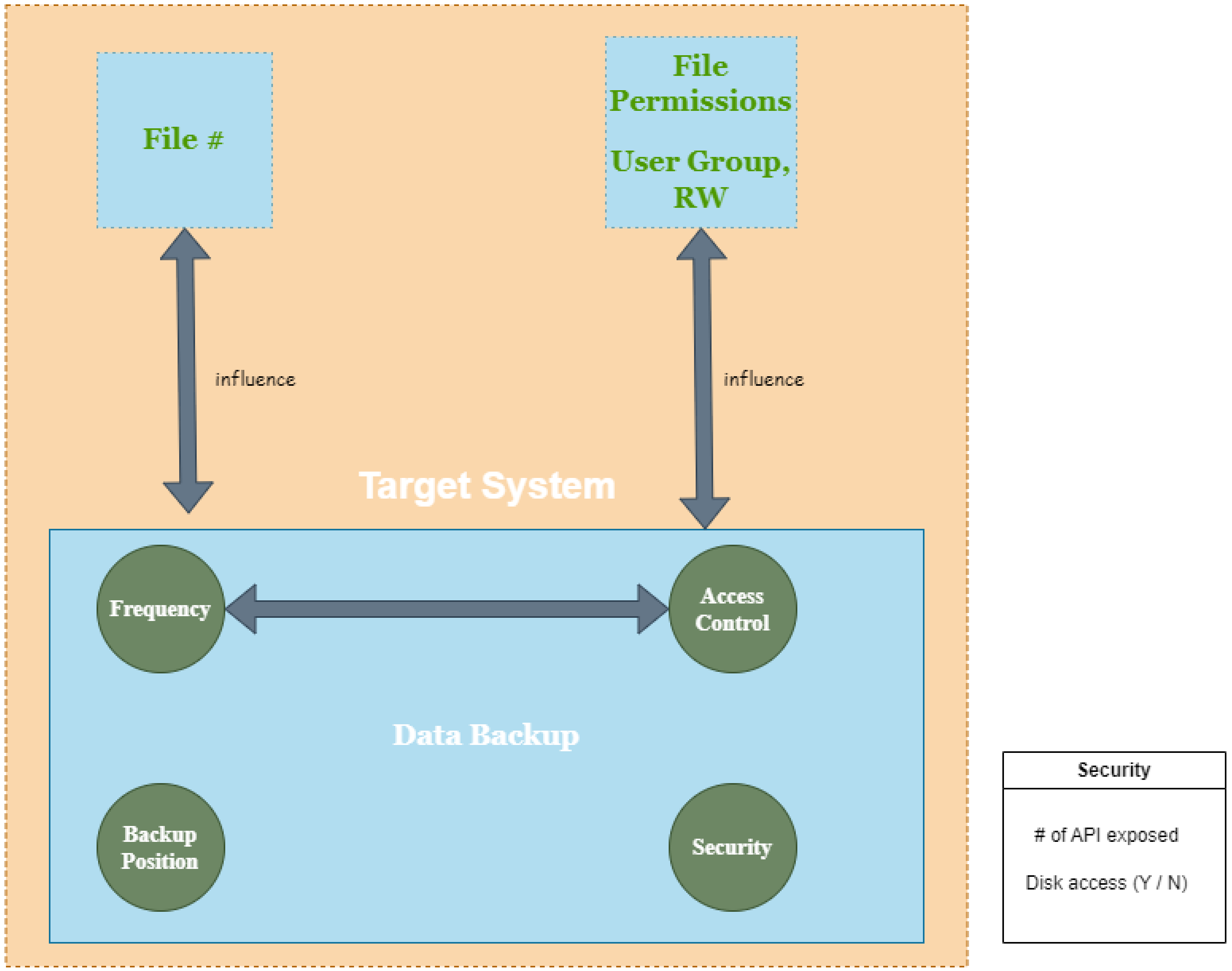
It collects data in **target system** (preprocessing), **FS filter** (VFS in Linux) layer as well as **BIO layer**. It also optionally collects data with **standardized ransomware** to illustrate the pattern of attack and verify the sanity of other satistics.


Application Examples
Bank
Hospital
private PC
Testing Framwork Preprocessing
Ransomware
Applications with Data Backup Policy
FS filter
File System (NTFS, F2FS, BtrFS (RAID) ...)
BIO Layer
Storage Media
Testing Framwork Statistics

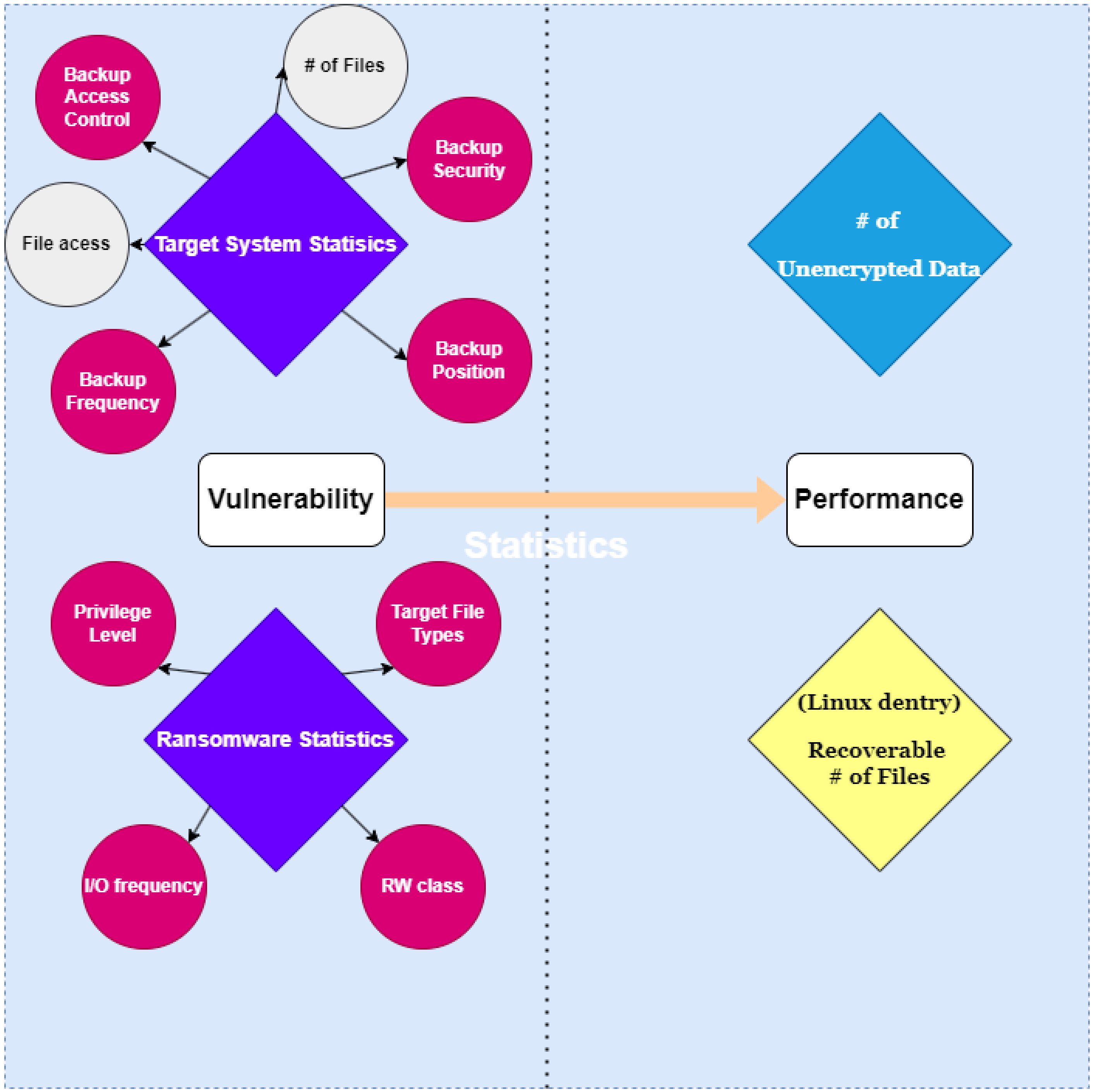
Standardized Ransomware (encryption and deletion)


Privilege Level
sudo (not likely)
Advanced User Group
Normal User
IO Frequency
Burst read / write
Write wait write wait ...
Privilege Level
RW Class
Ransomware
IO frequency
Target File Type
RW Class
Read then In-place modify
Read Encrypt Write in another place Delete Original Copy

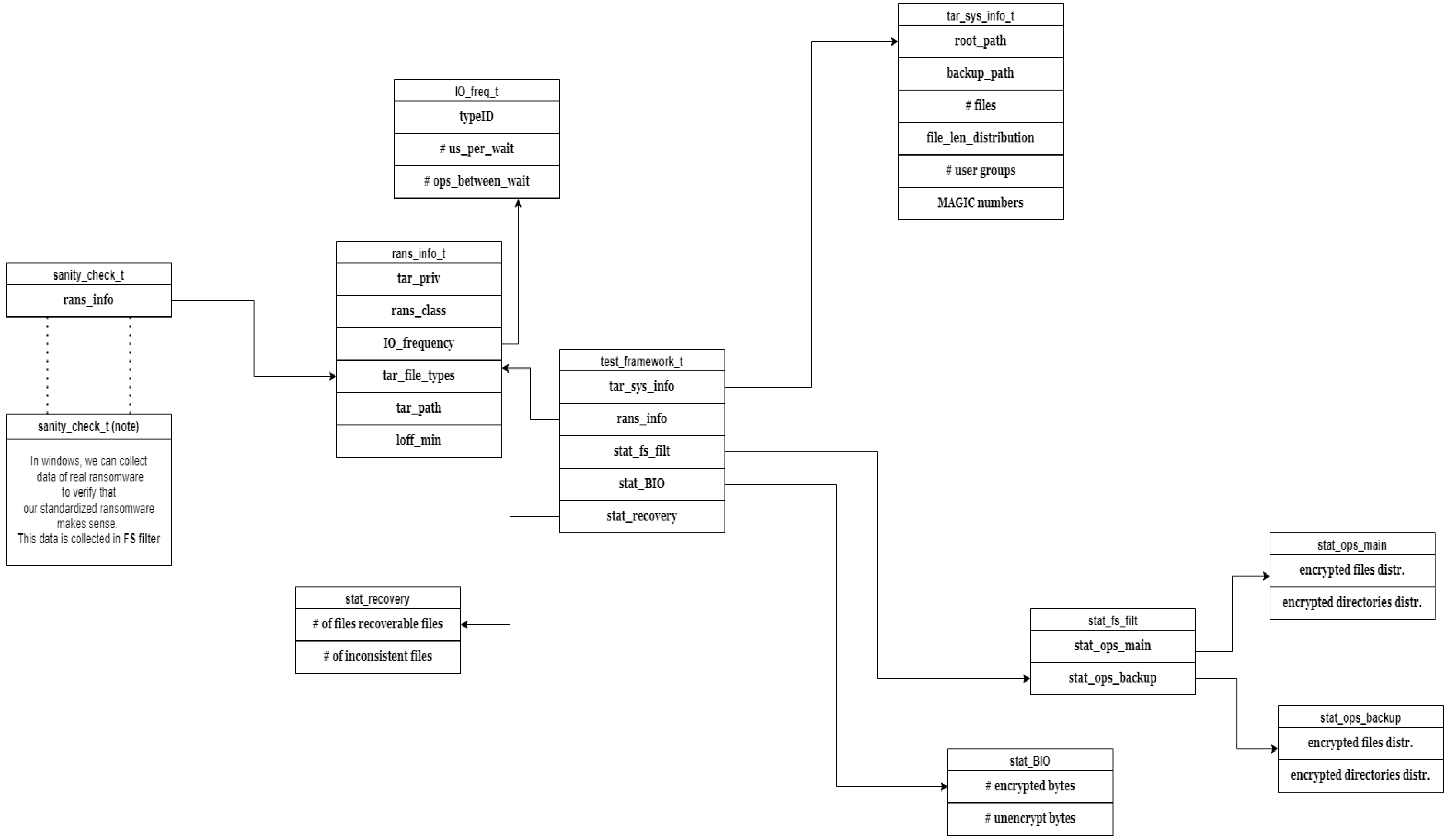
Target System (fingerprinting)


File #
File Permissions
User Group, RW
influence
influence
Target System
Frequency
Access Control
Data Backup
Backup Position
Security
Security
# of API exposed
Disk access (Y / N)

Statistics


Backup Access Control
# of Files
Backup Security
File acess
Target System Statisics
Backup Frequency
Backup Position
# of Unencrypted Data
Vulnerability
Performance
Statistics
Privilege Level
Target File Types
Ransomware Statistics
I/O frequency
RW class
(Linux dentry) Recoverable # of Files

Data Structure

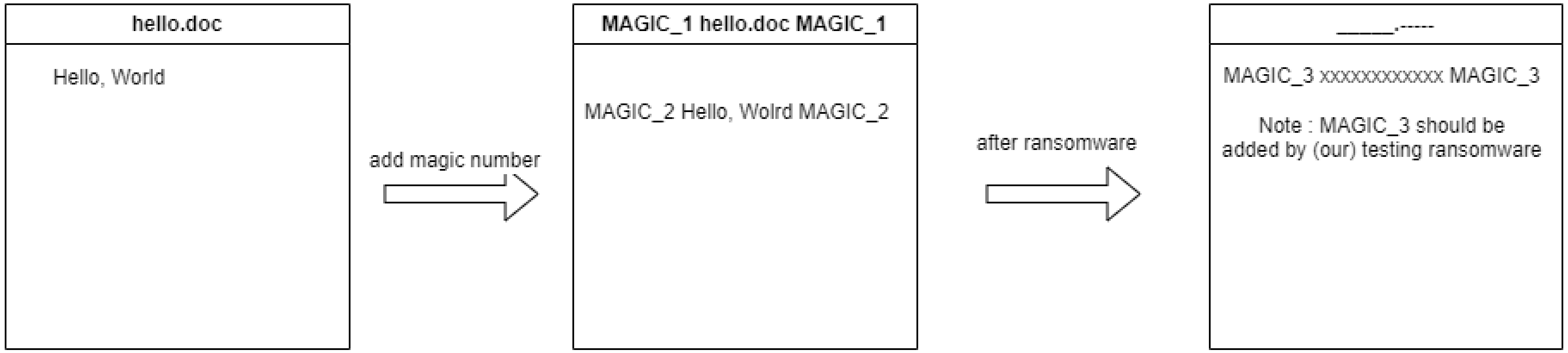

tar_sys_info_t
root_path
backup_path
# files
file_len_distribution
# user groups
MAGIC numbers
IO_freq_t
typeID
# us_per_wait
# ops_between_wait
sanity_check_t
rans_info
rans_info_t
tar_priv
rans_class
IO_frequency
tar_file_types
tar_path
loff_min
test_framework_t
tar_sys_info
rans_info
stat_fs_filt
stat_BIO
stat_recovery
sanity_check_t (note)
In windows, we can collect data of real ransomware to verify that our standardized ransomware makes sense. This data is collected in FS filter
stat_recovery
# of files recoverable files
# of inconsistent files
stat_ops_main
encrypted files distr.
encrypted directories distr.
stat_fs_filt
stat_ops_main
stat_ops_backup
stat_ops_backup
encrypted files distr.
encrypted directories distr.
stat_BIO
# encrypted bytes
# unencrypt bytes

Basic Implementation

Clone target system, and backup to a safe place

Migrate / Prepare Target System & Preprocess `tar_sys_info`

Add magic numbers to files in target file system


hello.doc
Hello, World
add magic number
MAGIC_1 hello.doc MAGIC_1
MAGIC_2 Hello, Wolrd MAGIC_2
after ransomware
______.-----
MAGIC_3 xxxxxxxxxxxx MAGIC_3
Note : MAGIC_3 should be added by (our) testing ransomware

MAGIC number should be 8 bytes (to avoid collision) to help BIO layer gather more information more easily.

Launch standardized ransomware, with `rans_info` prepared

When running ransomware

- In standardized ransomware, fill in `stat_fs_filt`
- In BIO, fill in `stat_BIO`.

BIO tracing in Linux

Currently implemented

- Target System & Databackup Generation
- Fine-grained Access Control (via ACL)
- Fingerprinting Report
- Ransomware Encryption

TO DO

- BIO dump
- Data backup
  - consistency report
  - security report (To discuss)
- Propagation