Windows Disk Image

- Assumes basic setup from setup
- All command below should be
  - copied to a “IR log” text/md file
  - edited as needed
  - executed with output copied back to “IR log”
    - recommendation: move commands/output from “useless commands” to second “IR dump” text/md file
- ## is a placeholder for a number

setup

```
echo 'source $HOME/.case.env' >> ~/.bashrc
```

Env variables

```
rm ~/.case.env
pwd
WD=/case/working/folder && echo "WD=$WD" >> ~/.case.env
E=/case/working/folder/evidence/evidence.E01 && echo "E=$E" >> ~/.case.env
```

Disk Image basics

```
mmls -r $E
sudo -E ~/.local/bin/imount -p -o ## -md /mnt/imount $E
# -p has 'pretty' & predictable folder name, fails if in use
# new tab or crtl-z
DISK_C=<imount_dir> && echo "DISK_C=$DISK_C" >> ~/.case.env
```

Hayabusa

```
target-fs "$E" cp 'c:/Windows/System32/winevt/Logs' --output ./logs
docker run -it --rm -v "$PWD":/work --entrypoint /opt/hayabusa/hayabusa
tabledevil/hayabusa:3.8.1 csv-timeline -d /work/logs -o
/work/output/hayabusa.csv -p super-verbose
docker run -it --rm -v "$PWD":/work --entrypoint /opt/hayabusa/hayabusa
tabledevil/hayabusa:3.8.1 json-timeline -d /work/logs -o
/work/output/hayabusa.jsonl -p super-verbose
```

Plaso

```
docker run -it --rm --entrypoint=/bin/bash -v ./:/work log2timeline/plaso
# docker run -it --rm --user :$(id -g) --entrypoint=/bin/bash -v ./:/work
log2timeline/plaso

# one step
psteal.py --source /work/evidence/<filename> -o dynamic,json_line -w
/work/data/plaso_#.json

# two step
log2timeline.py --storage-file /work/data/timeline.plaso
/work/evidence/<filename>
psort.py -o json_line -w /work/output/plaso_out.json
/work/data/timeline.plaso
#psort.py -o json_line -w /work/output/plaso_out.json
/work/data/timeline.plaso


cp output/plaso_out.json tools/splunk/import/
cd tools/splunk
docker compose up -d
# -> localhost:8000, admin:password,
settings->add->monitor->files->index_once->source_type=plaso
```

Splunk

```
mkdir -p etc/system/local/
vim props.conf
cp props.conf etc/system/local
```

Browser

```
Users\...\AppData\Local\Google\Chrome\User Data\Default\History
Users\...\AppData\Local\Microsoft\Edge\User Data\Default\History
Users\...\AppData\Roaming\Mozilla\Firefox\Profiles\<ProfileName>.default\pla
ces.sqlite


C_DR=
EUSER=
mkdir -p evidence/chrome
mkdir evidence/edge
mkdir evidence/firefox
cp -r $C_DR/Users/$EUSER/AppData/Local/Google/Chrome/User\
Data/Default/History evidence/chrome
cp -r $C_DR/Users/$EUSER/AppData/Local/Microsoft/Edge/User\
Data/Default/History evidence/edge
cp -r $C_DR/Users/$EUSER/AppData/Roaming/Mozilla/Firefox/Profiles/*-
release/places.sqlite evidence/firefox/
```

```
sqlitebrowser evidence/firefox/places.sqlite
sqlitebrowser evidence/chrome/History
```

PE

```
https://github.com/sethenoka/Install_EZTools.git
```

Prefetch on Linux

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.cargo/env
rustup toolchain install 1.85.0
rustup default 1.85.0
```

```
use chrono::{DateTime, Utc};
use std::{env, fs};

fn filetime_to_datetime(ft: i64) -> DateTime<Utc> {
    // FILETIME epoch (1601-01-01) -> Unix epoch (1970-01-01)
    const EPOCH_DIFF_100NS: i64 = 116_444_736_000_000_000;

    let unix_100ns = ft - EPOCH_DIFF_100NS;
    let secs = unix_100ns / 10_000_000;
    let nanos = ((unix_100ns % 10_000_000) * 100) as u32;

    DateTime::from_timestamp(secs, nanos)
        .expect("invalid timestamp")
}

fn main() {
    let path = env::args()
        .nth(1)
        .expect("usage: pf_info <file.pf>");

    let raw = fs::read(&path).expect("read failed");

    let info = prefetch_core::parse(&raw)
        .unwrap_or_else(|e| panic!("parse error: {:?}", e));

    println!("Executable : {}", info.executable);
    println!("Run count  : {}", info.run_count);

    println!("\nLast run times:");
    for ft in &info.last_run_times {
        println!(
            "  {}  ({})",
            filetime_to_datetime(*ft).to_rfc3339(),
```

```
            ft
        );
    }

    println!("\nVolumes:");
    for v in &info.volumes {
        println!("  Device: {}", v.device_path);

        let created = filetime_to_datetime(v.creation_time);
        println!("  Created: {}", created.to_rfc3339());

        println!("  Serial : {:08X}", v.serial);
        println!();
    }
}
```

```
#![allow(clippy::unwrap_used, clippy::expect_used)]
use std::path::PathBuf;
use std::fs;
use std::path::Path;
use chrono::{DateTime, Utc};

/// based on
https://docs.rs/crate/prefetch-core/0.1.0/source/examples/pf_dump.rs


fn filetime_to_datetime(ft: i64) -> DateTime<Utc> {
    // FILETIME epoch (1601-01-01) -> Unix epoch (1970-01-01)
    const EPOCH_DIFF_100NS: i64 = 116_444_736_000_000_000;

    let unix_100ns = ft - EPOCH_DIFF_100NS;
    let secs = unix_100ns / 10_000_000;
    let nanos = ((unix_100ns % 10_000_000) * 100) as u32;

    DateTime::from_timestamp(secs, nanos)
        .expect("invalid timestamp")
}

fn csv_escape(s: &str) -> String {
    let mut out = String::new();
    out.push('"');
    for c in s.chars() {
        match c {
            '"' => out.push_str("\"\""),
            _ => out.push(c),
        }
    }
    out.push('"');
    out
}
```

```
fn main() {
    let mut root = PathBuf::from(env!("CARGO_MANIFEST_DIR"));
    root.pop();
    let out_dir = std::env::args()
        .nth(1)
        .unwrap_or_else(|| "/tmp/pf_scca".to_string());
    std::fs::create_dir_all(&out_dir).expect("mkdir out_dir");

    let dir = Path::new("/media/data/IR/case260618/prefetch");

     let files: Vec<String> = fs::read_dir(dir)
        .unwrap()
        .filter_map(Result::ok)
        .map(|e| e.path())
        .filter(|p| {
            p.extension()
                .and_then(|ext| ext.to_str())
                .map(|ext| ext == "pf")
                .unwrap_or(false)
        })
        .map(|p| p.to_string_lossy().to_string())
        .collect();

//.filter(|p| p.extension().and_then(|s| s.to_str()) == Some("pf"))
    println!(
"file,scca_len,version,executable,run_count,last_run_times,filenames_count,filenames,volumes"
    );
    for name in files {
        let p = PathBuf::from(&name);

        let raw = std::fs::read(&p).expect("read fixture");
        let scca = prefetch_core::decompress(&raw).expect("decompress");

        std::fs::write(PathBuf::from(&out_dir).join(format!("{name}.scca")),
&scca)
            .expect("write scca");

        let info = prefetch_core::parse(&raw).expect("parse");

        let filenames = info
            .filenames
            .iter()
            .map(|f| escape(f))
            .collect::<Vec<_>>()
            .join(";");

        let volumes = info
            .volumes
            .iter()
            .map(|v| {
```

```
                format!(
                    "{{serial:{},device_path:{},creation_time:{}}}",
                    v.serial,
                    escape(&v.device_path),
                    filetime_to_datetime(v.creation_time)
                )
            })
            .collect::<Vec<_>>()
            .join(";");

        let last_runs = info
            .last_run_times
            .iter()
            .map(|t| filetime_to_datetime(*t).to_string())
            .collect::<Vec<_>>()
            .join(";");

        println!(
            "{},{},{},{},{},{},{},{},{}",
            csv_escape(&name),
            scca.len(),
            info.version,
            csv_escape(&info.executable),
            info.run_count,
            csv_escape(&last_runs),
            info.filenames.len(),
            csv_escape(&filenames),
            csv_escape(&volumes)
        );
    }
}

fn escape(s: &str) -> String {
    s.replace('\\', "\\\\")
}
```

Extended disk image anaylsis

```
fsstat -o <offset> $E
istat -o <offset> $E 5 # root node
istat -o <offset> $E <inode from fsstat>

fls -o <offset> -m C: -r $E > data/bodyfile
mactime -b data/bodyfile -d -z UTC yyyy-mm-ddThh:mm:ss >
output/disk_timeline.csv

mactime -b data/bodyfile -d -z UTC yyyy-mm-ddThh:mm:ss..yyyy-mm-dd >
output/disk_timeline.csv
```

```
# dissect

target-query -f hostname,domain,version,ips,install_date,timezone $E
# much more useful for queries on multiple disks at once

target-query -j -f services $E | jq -r '.name'
# JSON output → jq
target-query --list | grep -iE
'userassist|shimcache|amcache|services|powershell_history|browser.history'
```