Remote Scoutware Threat Reconnaissance

v.1.3

by Vitaly Kamluk and Nicolas Collery

2021

https://t.me/learningnets

Quick Intro

If you are reading this, you must be interested in remote threat reconnaissance and incident response. Basic idea of this is to have a shell and a collection of tools at the remote location where a security incident occurred. This is the foundation of scoutware. There is a lot added on top of this, such as secure and low bandwidth comms, user access control, session sharing and monitoring, interactions with the physical owner of the remote system and more. You will learn about some of these features and can extend them to match your custom needs.

To avoid confusion, let's clarify what Bitscout is. Generally, it's a project name for a live OS constructor. Live OS is an operating system that can boot from removable media, such as a CD or a USB drive. The live OS constructor is simply a collection of shell scripts and resource files that produce a bootable system disk image file in ISO format. However, Bitscout name can also be used for the instance of a live running system booted from this disk. More to that, once Bitscout is fully running it splits into two systems: Bitscout host and Bitscout container.

Bitscout host system is natively running live OS. It has the main filesystem (rootfs) present in RAM (equivalent to RAM disk). This is why modifying any file on Bitscout doesn't affect any of the physical media. The media contains a compressed rootfs image which is decompressed when a file needs to be accessed. Bitscout host is used to configure the network, run a VPN link and forward connections to the Bitscout container, manage physical storage devices and attach them to Bitscout container for analysis by an expert.

Bitscout container is literally a containerized system (based on systemd-nspawn) which replicates Bitscout host. Why are there two systems? For access control, isolation, and additional protection against unintended system changes. Bitscout container is a disposable environment that runs as an unprivileged system user. This environment is provided to the expert using Bitscout remotely. Although the expert works as root and can install new software packages, change almost any system configuration files, it is an emulation of a superuser account. The 'root' user in the container cannot load/unload kernel modules, mount filesystems through kernel drivers, create device files, change firewall rules, etc.

Please treat Bitscout as a launchpad of your own remote scoutware. It comes with a set of default settings and tools, mainly including disk and file recovery and analysis tools, filesystem drivers, pattern recognition tools, and forensic frameworks. You may reduce the set of included tools and add your own, depending on your use case.

Chapter 1. Building Your Own Bitscout

Location: Bitscout build directory

To build your own scoutware ISO based on Bitscout, you need an Ubuntu system (can be a VM) connected to the Internet. There are multiple variants of Bitscout created from the packages that come with Ubuntu 16.04, 18.04, 20.04. Make sure that your build environment matches the Bitscout version you are using. Cross-building is possible but is likely prone to unexpected errors.

To start, open a terminal and navigate to a directory where you would like to keep the project. Before the building process, you need to fetch the latest version of Bitscout builder from Github with the following command:

```
$ git clone https://github.com/vitaly-kamluk/bitscout
```

You should be able to see the output similar to the following:

```
Cloning into 'bitscout'...
remote: Enumerating objects: 773, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 773 (delta 4), reused 7 (delta 2), pack-reused 760
Receiving objects: 100% (773/773), 621.36 KiB | 573.00 KiB/s, done.
Resolving deltas: 100% (460/460), done.
```

After this, enter the bitscout subdirectory

```
$ cd ./bitscout
```

And start the build

```
$ ./automake.sh
```

```
Welcome to Bitscout builder!
Host OS info: Linux 5.8.0-53-generic #60~20.04.1-Ubuntu SMP Thu May 6
09:52:46 UTC 2021 x86_64
Distributor ID:    Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:    20.04
Codename:    focal
Build Target:    Bitscout 20.04
Git commit:    5ab91ba373f53bf0f2711624be380a2a5bada11e
```

```
It seems that you are in a fresh build environment.
We need to populate the config with some essential data.
Please answer the following questions or put your existing build config to
config/bitscout-build.conf.
Proceed to interactive settings? [Y/n]:
```

Note that it is safe to stop the building process at any moment (by pressing Ctrl+C) and restart it by running ./autobuild.sh again. If you want to reset the constructor state, except removing configuration files and keys/certificates from previous build attempts, run "./clean.sh" script. A complete reset, including removing configuration files and generated keys/certificates requires running "./clean.sh" and removing ./config directory.

Note:
Bitscout autobuild.sh script should handle build interruption nicely in most cases, however, it's good to verify that there are no mounted filesystems remaining after an interruption occurred. You can do that with a " **mount | grep '/build'** " command.

Previous command starts the interactive building mode, which offers you to enter some most important settings. Type "Y" and press Enter to continue.

```
bitscout may be built in various sizes.
Please choose option number:
 1. compact - minimal size, less tools and drivers.
 2. normal - includes most common forensic tools, drivers, etc.
 3. maximal - includes maximum forensic tools and frameworks.
 Your choice (1|2|3):
```

At this point you need to select how many relevant packages will be included into the Bitscout ISO file. Building "normal" Bitscout is recommended to balance the time of build and include some essential forensic tools into the image. To proceed with normal mode, type "2" and press Enter.

```
To use bitscout over the internet you will likely need a VPN server.
Just enter your protocol/host/port and we generate an OpenVPN config
template for you. You can always change it later.
Examples:
 udp://127.0.0.1:2222
 tcp://myvpnserver:8080
Your input [connection string|<NONE>]:
```

Bitscout is normally accessed using OpenVPN by default and this is where you need to enter your OpenVPN server location and protocol (UDP or TCP). You may customize these settings

later, as well as place your own VPN certificates and other options. Also, as long as the training doesn't require connecting over VPN (we will enable direct connections from LAN later), we do not depend on this setting and can skip it by pressing Enter, which disables OpenVPN for the build.

Some off Bitscout project contributors used it together with a network syslog server to log all commands used within the environment. This feature requires additional setup on the server you are going to use. Feel free to skip this setting by pressing Enter with empty input:

```
You may configure a remote syslog server to log shell history. To continue
without a syslog server simply press Enter.
Your input [host|<NONE>]:
```

Probably the most important setting is the Bitscout native architecture, which is offered next:

```
What's the target system architecture?
1. 64-bit (amd64)
2. 32-bit (i386)
Please choose [1|2]
```

If you are planning to work with very old computers which run on 32-bit CPU, you may want to build a 32-bit version of Bitscout. However, most modern systems run on 64-bit CPUs nowadays, so using that is recommended. Note, that the analyzed OS architecture doesn't need to match Bitscout native architecture. In other words, you should be able to analyze both 32-bit and 64-bit Windows using 64-bit Bitscout.

```
Saving configuration..
Configuration saved. Continue? [Y/n]:
```

This is the last question before the building process starts. Pressing "Y" and Enter here continues to the automatic creation of the ISO file. Two requirements for successful build as follows:

- Internet availability for Ubuntu software packages and tools download
- Your user should have sudo privileges to install additional software and work with rootfs.

Generally, should you want to put your own SSH key, OpenVPN configuration with certificates or change any other settings, you can do that before continuing. However, for the training, no changes are required, so you can safely continue by pressing "Y" and Enter.

```
Updating submodules..
Checking base requirements..
```


https://t.me/learningnets

```
Downloading focal:amd64..
Building base root filesystem..
Fetching the list of essential packages..
I: Retrieving InRelease
...
```

If your building process breaks, or you stop it by pressing Ctrl+C, or you want to rebuild the system with some resource or script files changed, you may restart it by running ./autobuild.sh again. In this case, you won't be asked to enter all the settings again, but you may see the following options below.

```
Updating submodules..
Checking base requirements..
Found existing chroot directory. Please choose what to do:
 1. Remove existing chroot and re-download.
 2. Do not re-download, skip this step.
 3. Abort.
 You choice (1|2|3):
```

Using option 1 is the safest, as it rebuilds rootfs from scratch, however it takes some extra time to do so. The second option keeps rootfs. This is useful if you added some changes to rootfs manually.

Note:
Bitscout saves downloaded packages in ./build.amd64/cache (or ./build.i386/cache for 32-bit build respectively). This saves waiting time to download the software packages from the Ubuntu repository and reduces load on the internet servers. This cache directory can also be copied to other machines if you want to build it elsewhere.

Note 2:
Bitscout may ask for the root password of the current system, because it automatically installs software packages necessary to build the ISO and uses privileged system functions such as mount, chroot and more. Although everything it does is running plaintext script commands, which you can verify in the code from Github, if you are paranoid, it is recommended to run the building process on a VM.

Once the ISO build process is finished you should be able to find the ISO file in current directory:

```
$ ls -la ./*.iso
```


https://t.me/learningnets

```
-rw-r--r-- 1 root root 673509376 Jun  9 06:12 ./bitscout-20.04-amd64.iso
```

Running file tool on it shall report that this is bootable ISO file:
`$ file ./bitscout-20.04-amd64.iso`

```
./bitscout-20.04-amd64.iso: DOS/MBR boot sector; GRand Unified Bootloader, stage1 version 0x79, boot drive 0xbb, stage2 address 0x8e70, 1st sector stage2 0xb8db31c3, stage2 segment 0x201; partition 1 : ID=0xee, start-CHS (0x0,0,2), end-CHS (0x282,19,24), startsector 1, 1315447 sectors, extended partition table (last)
```

You may inspect the contents of the ISO file with isoinfo command:
`$ isoinfo -l -i ./bitscout-20.04-amd64.iso`

```
Directory listing of /
d---------   0    0    0   2048 Jun  9 2021 [     20 02]  .
d---------   0    0    0   2048 Jun  9 2021 [     20 02]  ..
d---------   0    0    0   2048 Jun  9 2021 [     22 02]  .disk
----------   0    0    0   193 Jun  9 2021 [   6877 00]  README.diskdefines;1
d---------   0    0    0   2048 Jun  9 2021 [     23 02]  System
d---------   0    0    0   2048 Jun  9 2021 [     26 02]  boot
----------   0    0    0   2048 Jun  9 2021 [   1717 00]  boot.catalog;1
d---------   0    0    0   2048 Jun  9 2021 [     86 02]  casper
----------   0    0    0   2949120 Jun  9 2021 [ 126 00]  efi.img;1
----------   0    0    0   0 Jun  9 2021 [   1716 00]  mach_kernel.;1
----------   0    0    0   1858 Jun  9 2021 [ 328678 00]  md5sum.txt;1
----------   0    0    0   0 Jun  9 2021 [   1716 00]  ubuntu.;1
...
```

You may also try a quick dynamic test of the ISO file to make sure that it boots and all critical services are available. This can be done by running "./autotest.sh" script, which starts a temporary VM and shows you the booting process in text mode. In the end it shall leave an output on terminal similar to the following:

```
autotest: Started new autotest on [date] [time] UTC
autotest: Started new autotest on Wed 09 Jun 2021 05:22:52 AM UTC
Creating a new local tmux session and starting qemu..
Using hardware virtualization support..
```


https://t.me/learningnets

```
Started qemu in tmux session.
Waiting for the qemu monitor socket..
Waiting for socket at ./bitscout.monitor.sock ..
Attaching to monitor socket..
Waiting for the serial port socket..
Waiting for socket at ./bitscout.serial.sock ..
Attaching to serial port socket..
Initiating boot process..
To view the VM console use command:
$ remote-viewer spice://localhost:2001
Attaching to the tmux session..
[exited]
Autotest complete. Quick summary:
 Started new autotest on [date] [time] UTC
 Bitscout serial port socket was opened. QEMU serial port: OK
 Login prompt found. System boot: OK
 Container is up. Container check: OK
 OpenVPN service is not installed. OpenVPN: ERROR
 PrivExec service is running. PrivExec: OK
 Historian service is running. Historian: OK
 Guest container shell is available. Container shell: OK
 Container's ssh service is running. Container SSH: OK
 Test graceful exit: OK
```

During the testing you may see a serial console attached to the VM temporarily. The script implements local login and validates that important components and services are present and running.

https://t.me/learningnets

Fig. An example of autotest.sh script booting a Bitscout instance.

Chapter 2. Starting Your Bitscout Instance

Location: Bitscout build directory and a VM booting from a Bitscout ISO

Once you have an ISO file, you may try writing it to a USB drive, burn a CD and use it with a physical computer. However, for a training purpose or just a quick try a local VM is enough.

You may find prepared VMs in the training environment. We are using libvirt based on QEMU with virt-manager (aka Virtual Machine Manager) for the GUI tool.

The VMs are configured to use Bitscout ISO from /home/ubuntu/bitscout/bitscout-20.04-amd64.iso path. Should you have placed the project directory elsewhere, you need to modify the path to the ISO file.

https://t.me/learningnets

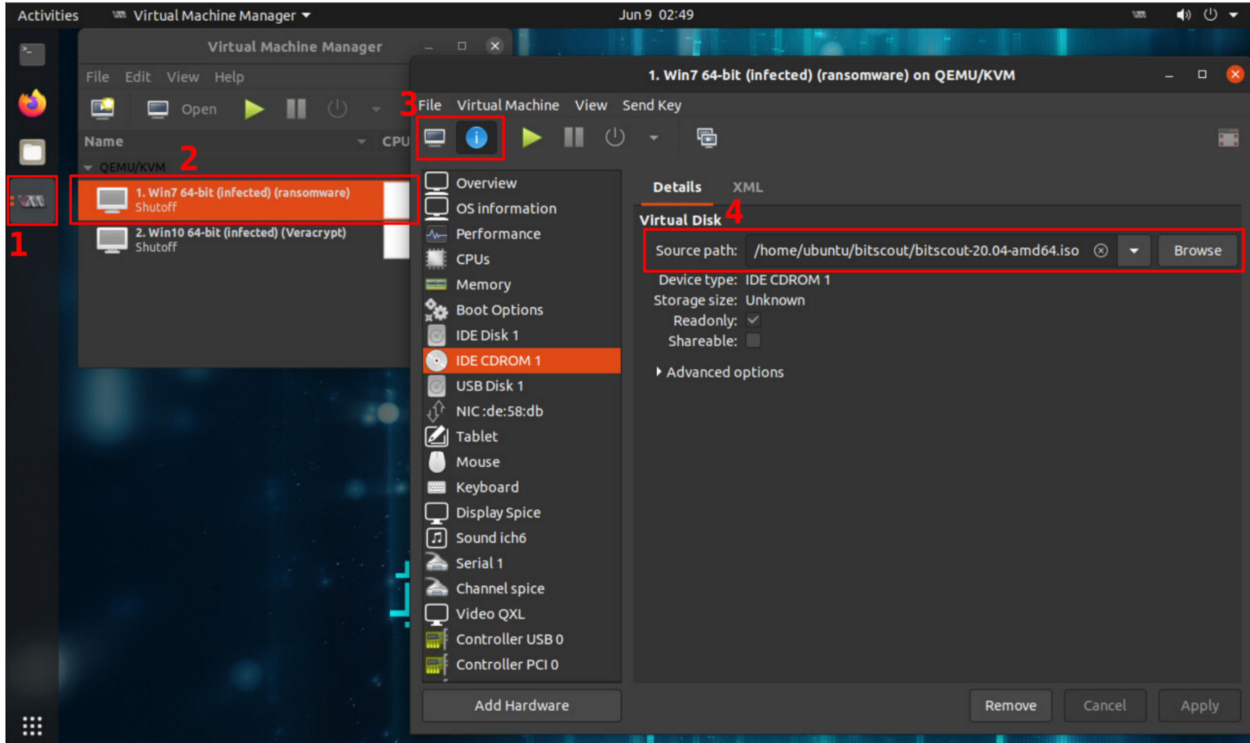


Fig. Quick visual instruction to modify path to the ISO file for a VM.

Once the path is set correctly you may boot the machine using the "play" button. Note that to see the VM display, you need to switch the view marked as element "3" in the visual instruction above.

Once your Bitscout boots successfully (may take a minute), you should be able to see the welcome screen with Bitscout management tool as follows.

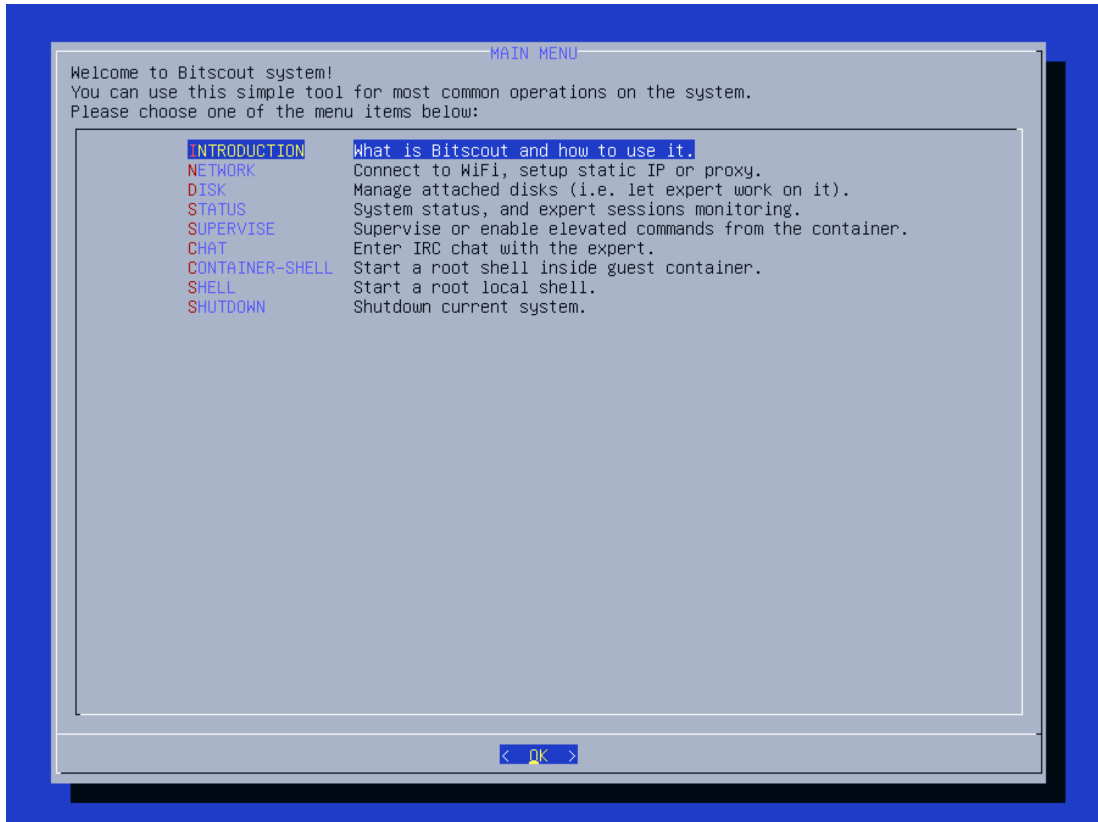


Fig. Bitscout management tool main menu.

The tool is implemented using TextUI (TUI). Use arrow keys, Enter, Spacebar and regular keyboard text input to navigate and interact with the tool.

https://t.me/learningnets

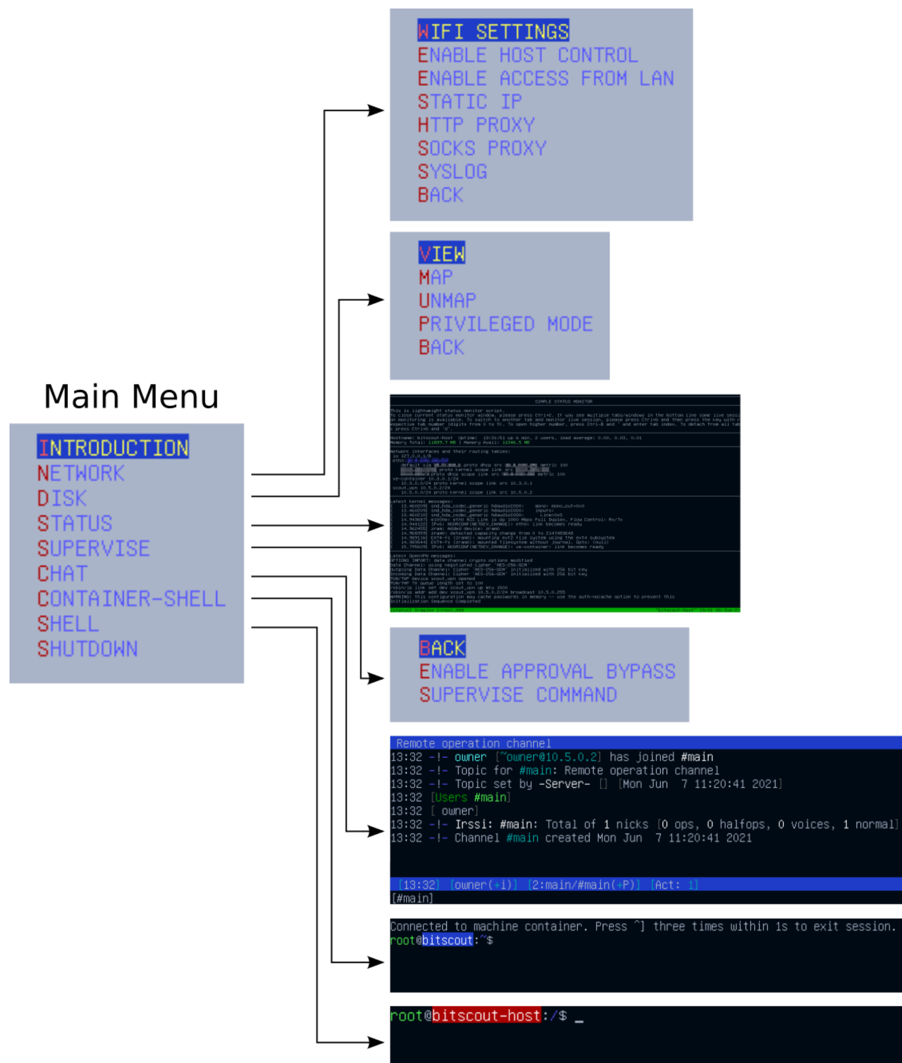


Fig. Map of Bitscout management tool submenus.

Some of the most commonly used functions are

- Network->Enable Access From LAN - enable direct connections to Bitscout instance.
- Network->Enable Host Control - enable connections to Bitscout host.
- Disk->Map - attach a block device (disk) to Bitscout container for analysis.

Chapter 3. Customise Your Bitscout

Location: Bitscout build directory

At some point you may want to include your own tools, or use some additional packages on Bitscout. This is very easy to do, you just need to find the right place to insert your changes. For that you need to understand stages of Bitscout building process, which include the following:

1. Interactive basic settings request from the user

https://t.me/learningnets

2. Preparation of the root filesystem (base kernel and software)
3. Container setup, configuration of other services (openvpn, ssh, etc)
4. Bitscout management tool installation
5. Preparation of the bootloader and other settings for hybrid CD/USB ISO image
6. Enhancing the bootloader script (checksum validation, optional kernel patching, etc)
7. Clean-up before compressing rootfs (removal of redundant software, cache files, etc)
8. Compressing rootfs, compiling the final ISO file
9. Exporting config files, keys, certificates to a dedicated directory (./exports)

In this chapter you need to extend Bitscout's default set of tools by adding more packages. There are two ways how this can be done:

1. Adding packages to the current root filesystem only (temporary solution).
2. Modifying build scripts to add them automatically to every build (persistent solution).

Method 1. Temporary rootfs modification

To add packages to the current rootfs only, you need to run inside a chroot-ed environment with rootfs of the Live OS. There is a script that enters chroot and mounts all necessary subdirectories (apt cache) and virtual filesystems (i.e. proc, sys):

```
$ ./scripts/chroot_enter.sh
```

This will provide you with an interactive shell to manipulate rootfs. You may run usual Linux commands to update packages (apt update) and modify system files.

> **Note:**
> These changes will not be preserved if you decide to fully rebuild Bitscout. Rootfs is removed either by running ./clean.sh script or by agreeing to redownload and rebuild rootfs after you run ./automake.sh.

Method 2. Permanent rootfs modification

To add packages to rootfs automatically with every Bitscout rebuild, you need to modify some of the building scripts. One such script for customization exists as a template in ./scripts/chroot_customize.sh. Note the following lines in the file:

```
…
#The following is executed as a script inside the chroot environment.
#Ignore the first line required for apt-fast (a faster alternative for apt)
chroot_exec build.$GLOBAL_BASEARCH/chroot \
'export DEBIAN_FRONTEND=noninteractive; aria2c(){ /usr/bin/aria2c
--console-log-level=warn "$@";}; export -f aria2c;

#apt-fast --yes install gdb chntpw libguestfs-tools
```


https://t.me/learningnets

```
#apt-fast --yes install yara samba python-pip nbd-client xnbd-client nbd-server
xnbd-server
#pip install --upgrade pip
#pip install artifacts bencode libscca-python
#systemctl disable smbd
'
...
```

"chroot_exec" is a function that executes the shell command provided in the second argument inside the given rootfs directory (the first argument). Example highlighted above shows a long shell command that occupies several lines for readability. The lines starting from '#' symbol are commented out lines of a template script. Feel free to uncomment some lines or add your own before the closing single quote sign.
After this permanent modification you may either start building ISO from scratch by running ./automake.sh or do it faster by running only selected scripts that are required to build the final ISO file:
$ ./scripts/chroot_customize.sh && ./scripts/image_prebuild_cleanup.sh && ./scripts/image_build.sh

Chapter 4. Disk Image Acquisition (Local)

Location: Mission 1 - WinServer2012 DC 64-bit (infected)

In this chapter we start working with a prepared VM that contains a Windows domain controller affected by an unknown malware. Some of the common incident response actions in such case include:

- Making a secure copy of the disk (acquiring disk image)
- Locating the malware
- Determining the origin of the infection
- Deactivating the malware
- Remediating/recovering the rest of the system
- Temporary putting it online (if critical) while building a replacement system

We will start from disk acquisition in this chapter. For the sake of saving time, we can copy and calculate checksum only for a part of the disk. Real disk image acquisition sometimes may take hours to complete.

We assume that the owner has provided expert read-only access to the target disk via evidence0 device (/dev/host/evidence0). This can be done via Bitscout management tool: DISK->MAP->...

https://t.me/learningnets

Bitscout by default includes standard tools for disk image acquisition, such as dd (disk duplicator) and enhanced version of dd called dcfldd (you may also check dc3dd[1]). The tools require input and output device/file path as parameters. In addition dcfldd/dc3dd can calculate various hashes as the data is copied. Here is some examples of using these commands on Bitscout:

```
$ dd if=/dev/host/evidence0 of=/mnt/usb/disk.dd
$ dcfldd if=/dev/host/evidence0 of=/mnt/usb/disk.dd hash=md5 hashlog=/mnt/usb/disk.dd.md5
```

However, to be able to save the output on a locally mounted external drive you will need to do some extra steps, depending on what filesystem you have and what type of drive you are using.

Below we show an example of mounting a USB drive with ExFAT filesystem.
Once a disk drive is attached to the computer running Bitscout, it shall appear as a block device. To see all block devices, you can use lsblk command:

```
$ lsblk
```

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
...
sda    8:0    0     36G  1 disk
├─sda1   8:1      0   350M  1 part
└─sda2   8:2      0  35.7G  1 part
sdb    8:16   0     16G  1 disk
└─sdb1   8:17   0 16G  0 part
...
```

The highlighted in bold is a partition on the USB drive that is of expected size: 16GB. At this point the owner needs to map the attached USB drive partition (in this case /dev/sdb1) to the read-write storage device of the container (i.e. storage0). That is done via Bitscout menu DISK->MAP->...

Once the block device sdb1 is attached to the container it is accessible via /dev/host/storage0. As far as filesystem mount operation is privileged (may cause disk data modification), it is not immediately available to the expert. The owner of the system needs to mount it for the expert or better authorise the expert to execute limited mount operation on the LiveOS host system. To simplify this process Bitscout contains a simple tool called **mount.priv**. Below is an example of the tool command line.

```
$ mkdir /mnt/usb
$ mount.priv -t ntfs -o rw /dev/storage0 /mnt/usb
```

[1] https://www.forensicswiki.org/wiki/Dc3dd

https://t.me/learningnets

Note that specifying the filesystem type (-t ntfs) is essential. Automatic filesystem detection is restricted for security. Also, privileged mount operation is disabled in Bitscout by default. So when running it for the first time, you should see the following message:

```
$ mount.priv -t exfat -o rw /dev/storage0 /mnt/usb
Privileged execution mode is not enabled.
```

This indicates that the owner hasn't enabled privileged execution mode yet. This mode can be enabled via Bitscout menu DISK->PRIVILEGED MODE->ENABLE

Note:
Privileged mode authorises to use only subset of filesystems from the following list: "ntfs" "ntfs-3g" "vfat" "exfat" "ext" "ext2" "ext3" "ext4" "iso9660" "udf" "xfs" "reiserfs" "hfs" "hfsplus" This list is part of ./resources/sbin/privexecd.sh script in the Bitscout build folder.

Other filesystems or mount operations that require more sophisticated procedure, can be mounted with more commands on the LiveOS. In this case we recommend using supervised-shell and shared directory between host and container systems. Directory /mnt/container on the Live OS is visible as /mnt/host on the container. Everything created there is shared between the container and the Live OS. This is achieved via LXC container configuration. To see full list of container settings and mapped devices, check the following config file on the Live OS host: /etc/systemd/nspawn/container.nspawn

Supervised Shell and Unrestricted Access.

When dealing with a computer system owner that lacks experience with Linux command line, you may want to use a feature of Bitscout called supervised shell. This is a way to execute commands outside of the container in the context of Live OS as root. This, however, requires supervision and approval of the system owner.

To request running privileged command on Live OS, enter the supervised shell with the following command:
`$ supervised-shell`
You will see red background prompt like this:
`supervised>`
After that you may type and run one command. Command execution will be delayed until the owner approves this via Bitscout menu Supervise->Supervise Command.

https://t.me/learningnets

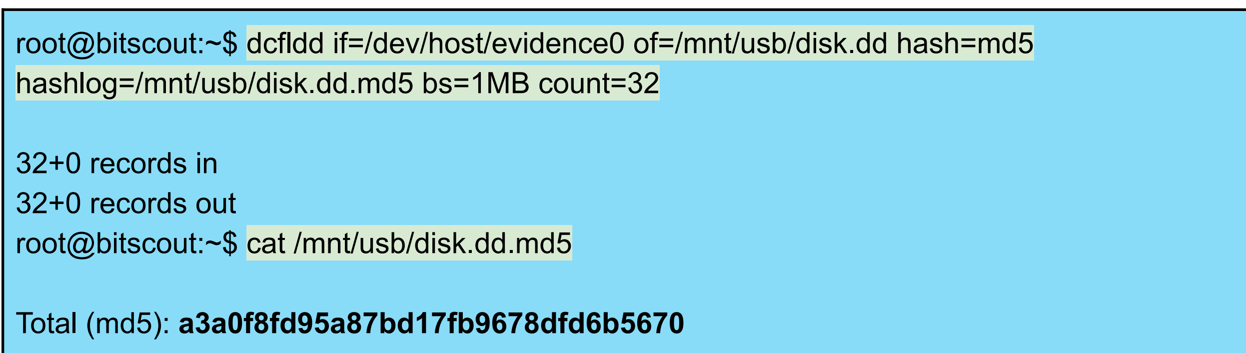
Once the user approves the command by pushing [ALLOW], it gets executed.

This feature shall be executed with caution to avoid mistakes and potential data corruption. The owner of the system may also take a screenshot of the command for the record.

If there is full trust to the expert it is possible to automatically approve all supervised commands by enabling approval bypass (see SUPERVISE->ENABLE APPROVAL BYPASS in Bitscout menu). Alternatively, you may simply allow SSH to the Live OS host to pass full control on the system to the expert. This is available via NETWORK->ENABLE HOST CONTROL. The expert may use the same SSH key to connect to the host system.

After all, once an external disk is attached and mounted, you can acquire disk image. Below is an example of acquiring partial image (32MB):

```
$ dd if=/dev/host/evidence0 of=/mnt/usb/disk.dd bs=1MB count=32
$ dcfldd if=/dev/host/evidence0 of=/mnt/usb/disk.dd hash=md5 hashlog=/mnt/usb/disk.dd.md5 bs=1MB count=32
```



```
root@bitscout:~$ dcfldd if=/dev/host/evidence0 of=/mnt/usb/disk.dd hash=md5
hashlog=/mnt/usb/disk.dd.md5 bs=1MB count=32

32+0 records in
32+0 records out
root@bitscout:~$ cat /mnt/usb/disk.dd.md5

Total (md5): a3a0f8fd95a87bd17fb9678dfd6b5670
```

As you can see the data was acquired and the hash was stored by the main data file.

Chapter 5. Analyzing Filesystem Contents

Location: Mission 1 - WinServer2012 DC 64-bit (infected)

https://t.me/learningnets

Let's analyze existing filesystems and collect files' metadata. First, you will need access to the whole analyzed disk, including the partition table. So, make sure that not only a partition but also the whole disk is accessible. Let's assume it was mapped as evidence0 device.

To list the partitions table you may use mmls command from Sleuthkit framework.

```
root@bitscout:~$ mmls /dev/host/evidence0
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

        Slot     Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0000718847   0000716800   NTFS / exFAT (0x07)
003:  000:001   0000718848   0075493375   0074774528   NTFS / exFAT (0x07)
004:  -------   0075493376   0075497471   0000004096   Unallocated
```

This is displayed in sectors (which are 512 bytes long). If you want to see partition sizes in bytes, use additional option (-B)

```
root@bitscout:~$ mmls -B /dev/host/evidence0
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

        Slot     Start        End          Length       Size    Description
000:  Meta      0000000000   0000000000   0000000001   0512B   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   1024K   Unallocated
002:  000:000   0000002048   0000718847   0000716800   0350M   NTFS / exFAT (0x07)
003:  000:001   0000718848   0075493375   0074774528   0035G   NTFS / exFAT (0x07)
004:  -------   0075493376   0075497471   0000004096   0002M   Unallocated
```

You may see that the largest partition here is 35GB long. It likely has the operating system files. We also learn that it is of type NTFS/exFAT, which gives us a hint which filesystem to use when mounting it later. Take note of the starting sector of the partition: **718848**. We will use it later.

For now, let's collect metadata of all files, including deleted and reallocated without mounting the whole disk. We can do that with another tool **fls** from the Sleuthkit.

```
root@bitscout:~$ fls -o 718848 -r -l -p -z UTC /dev/host/evidence0 | head
d/d 58-144-1:   PerfLogs        2013-08-22 15:52:33 (UTC)           2013-08-22 15:52:33
(UTC)           2015-12-11 23:05:37 (UTC) 2013-08-22 15:39:30 (UTC)     48       0         0
r/r 4-128-4:    $AttrDef        2015-12-11 23:04:09 (UTC)   2015-12-11 23:04:09 (UTC)
2015-12-11 23:04:09 (UTC) 2015-12-11 23:04:09 (UTC)     2560    0         48
r/r 8-128-2:    $BadClus        2015-12-11 23:04:09 (UTC)   2015-12-11 23:04:09 (UTC)
2015-12-11 23:04:09 (UTC) 2015-12-11 23:04:09 (UTC)     0        0         0
...
```

Note the -o argument and the usage of the offset value in sectors that we saw previously.
Additional recommended options for fls include
-r - process all directories recursively
-l - print output in long (detailed) format
-p - display the full path for each entry
-z UTC - use UTC timezone when printing the results. Otherwise Bitscout-local system timezone is used.

The output may be quite long, so we recommend first piping it to the head command and see only a few lines. After that you may just redirect the output to a file on external storage, store locally in RAM, or transfer over the network to another host.

```
fls -o 718848 -r -l -p -z UTC /dev/host/evidence0 > /mnt/usb/evidence0.fls
```

To interpret and understand fls tool output data format, we recommend checking the official Sleuthkit wiki[2].

We also recommend saving the listing in so-called mactime format:

```
fls -o 718848 -r -m C: -p -z UTC /dev/host/evidence0 > /mnt/usb/evidence0.fls.mactime
```

In this case we use **-m C:** instead of -l. The "C:" is used as a drive prefix and -m option forces the mactime output data format, which looks as following.

```
root@bitscout:~$ fls -o 718848 -r -m C: -p -z UTC /dev/host/evidence0 |head
0|C:/PerfLogs ($FILE_NAME)|58-48-2|d/drwxrwxrwx|0|0|82|1449875051|1449875051|1449875051|1449875051
0|C:/PerfLogs|58-144-1|d/drwxrwxrwx|0|0|48|1377186753|1377186753|1449875137|1377185970
0|C:/$AttrDef ($FILE_NAME)|4-48-2|r/rr-xr-xr-x|48|0|82|1449875049|1449875049|1449875049|1449875049
0|C:/$AttrDef|4-128-4|r/rr-xr-xr-x|48|0|2560|1449875049|1449875049|1449875049|1449875049
0|C:/$BadClus ($FILE_NAME)|8-48-3|r/rr-xr-xr-x|0|0|82|1449875049|1449875049|1449875049|1449875049
0|C:/$BadClus|8-128-2|r/rr-xr-xr-x|0|0|0|1449875049|1449875049|1449875049|1449875049
...
```

[2] https://wiki.sleuthkit.org/index.php?title=Fls

https://t.me/learningnets

This format is more compact and can be used with the mactime tool to print the timestamps in chronological order. More about it in the Sleuthkit wiki[3].

Chapter 6. Disk Image Acquisition (Remote)

Subject VM: win7_32bit_nofde_infected

As far as Bitscout is a remote shell-oriented solution, it may sometimes be useful to transfer the whole disk image or part of it to the expert or network attached storage. There may be many ways of doing that. Here are some simple solutions:

1. You may copy the acquired disk image over SCP from the attached external drive.
2. Alternatively you can stream data directly from the block device:

```
expert $ ssh -i <SSH_KEY> root@<BITSCOUT_IP> "dd if=/dev/host/evidence0" > ./disk.dd
```

> **Note:**
> You may use "pv" command to monitor the data transfer status. Simply add this to the command as follows.
> ```
> expert $ ssh -i <SSH_KEY> root@<BITSCOUT_IP> "dd if=/dev/host/evidence0" | pv > ./disk.dd
> ```
> To have ETA value of data transfer you shall pass disk size to pv tool using "-s" parameter. The size of the block device in bytes can be obtained with "blockdev" command such as follows.
> ```
> root@bitscout $ blockdev --getsize64 /dev/host/evidence0
> ```

Chapter 7. Memory Dump (Qemu)

Subject VM: win7_32bit_nofde_infected

When it comes to malware hunting it's important to have a memory dump confirming active malware presence in memory. Not only it may ease incident analysis but may be required as a proof of ongoing active attack.
While Bitscout focuses on offline system analysis, assuming that physical memory dump was made before the system was shutdown, it contains all the features to make a memory dump of a fully booted system. Of course, this is not the same as a memory dump made on a real system in the middle of an attack, but if the system had already been shutdown by the owner (unfortunately this is quite a common response to malware infection), it can be rather helpful. Opportunity to do such memory dump over the network or analyse remotely in place may significantly speed-up investigation.

[3] https://wiki.sleuthkit.org/index.php?title=Mactime

https://t.me/learningnets

One of the simplest methods to make a memory dump is to use built-in Qemu features to save guest memory.

Note:
During real life investigation, it is important that you do not get locked because Windows regular activity produced too many copy-on-write changes written to the RAM and consumed all the RAM, so that the host Bitscout Live OS couldn't continue normal execution and the whole process had to be restarted.
You should also know that Live OS is not designed for long term run and because of internal logging it will eventually consume all the RAM making the system locked.
It's good practice to sacrifice a temporarily spawned VM and not allow it to grow disk change files larger than a certain size. This can be achieved with running **ulimit** command in shell:

```
$ ulimit -f 1000000
```

The command above limits the maximum file size created by current shell and child processes to 1'000'000 blocks, which is equivalent to ~1GB (each block is equal to 1KB).

In this case you need to start the VM and let it boot until it reaches the desired state of system initialisation. Below is description of the process:

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. The expert creates copy-on-write protection layer via qcow2 file in RAM:
   ```
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
   ```
3. The expert starts VM and let's it boot:
   ```
   $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide -monitor stdio -vnc :0 -spice port=2001,disable-ticketing -vga cirrus
   ```

Note:
You may control the size of virtual RAM for the booted VM via **-m** option. Considering reducing it to the minimum required to boot OS.

To control the state of system boot, or if there is some keyboard input required, the expert may connect to the VM over VNC or SPICE protocol remotely

```
expert $ remote-viewer spice://%BITSCOUT_IP%:2001/
```

or

```
expert $ remote-viewer vnc://%BITSCOUT_IP%:5900/
```

or use any other VNC/Spice client tool, depending on the expert's base OS.

https://t.me/learningnets

Note:
Bitscout automatically forwards remote connections on certain ports from Live OS to the container. This makes direct connections to VM possible. In fact there are more forwarded ports reserved to run more VMs or other services. The forwarded ports include:
TCP: 2000-2009
TCP: 5900-5909
For more details or customisation of iptables rules, please see file **resources/sbin/host-iptables** in the Bitscout build directory.

4. Once the system is booted, switch to qemu monitor prompt and use dump-guest-memory command, such as shown below:
   (qemu) dump-guest-memory /root/memory.dmp

Note:
Use the help command to read about additional options of dump-guest-memory command.
(qemu) help dump-guest-memory

Chapter 8. Memory Dump (Winpmem)

Subject VM: win7_32bit_nofde_infected

Alternative and arguably better memory dump can be made within the VM itself by running tools such as Winpmem from Google. There is a family of memory dumpers for Windows, Linux, macOS known as *pmem. Winpmem is one of them and is used to dump Windows OS memory with some extra context information, such as CPU state, collection of swapped memory pages, etc. It relies on a digitally signed driver required to be loaded with administrative privileges on the OS.

The whole idea is to boot the original OS in a VM with copy-on-write protection, put winpmem on it or make it accessible via network share, dump the memory and copy it from the VM. Let's describe this process step by step in example below:

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. The expert creates copy-on-write protection layer via qcow2 file in RAM:
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
3. Next, the expert normally needs to boot the system $ qemu-system-x86_64 -name Local -enable-kvm -cpu host **-m 512** -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide -monitor stdio -vnc :0 -spice port=2001,disable-ticketing -vga cirrus

https://t.me/learningnets

In the case of the current exercise, you may notice that the system doesn't accept the password provided. This is the result of a malware operation, which locked out all system accounts and changed passwords.

Fig. User account is locked, no user picture is displayed: effect of LockerGoga.

This requires an extra step to unlock the system before you can log in and run any tools on it.

4. To unlock local Windows OS account, you need to edit Windows credentials database file (SAM file). This time we will try using liguestfs-tools package to modify Windows registry files.

> **Note:**
> Libguestfs-tools package is based on qemu virtualisation and relies on minimal virtual Linux appliance used to attach and automatically recognise partitions and filesystems. Of course this is helpful only when you are dealing with non-encrypted disk drives.
> Libguestfs-tools needs to build such an appliance live, and this takes some memory. Don't forget to remove temporary files after using the tools. Large temporary files are usually created in **/var/tmp/.guestfs-0/** or similar path.

In order to let libguestfs-tools create a virtual appliance you need to provide Linux kernel binary to the builder which expects it to be found in **/lib/modules/%KERNEL_VERSION%/**. This means that the owner of the system has to authorise privileged command execution or copy the file to the guest container.

```
supervised> cp /cdrom/casper/vmlinuz /opt/container/chroot/root/
```

Once the kernel is copied to the container it can be moved to **/lib/modules/%KERNEL_VERSION%/**

```
$ cp mv /root/vmlinuz /lib/modules/%KERNEL_VERSION%/
```


https://t.me/learningnets

5. Copy the SAM database out of the VM disk image:
   $ virt-copy-out -a ./evidence0.qcow2 /Windows/System32/config/SAM ./
6. Unlock user account, reset password and promote user to Administrators via chntpw command:
   $ chntpw -u user ./SAM
7. Place the modified SAM database back to the VM disk image:
   $ virt-copy-in -a ./evidence0.qcow2 ./SAM /Windows/System32/config/
8. Cleanup guestfs temp files:
   $ rm -rf /var/tmp/.guestfs-0/
9. Create a shared folder to exchange files with VM.
   $ mkdir /root/smb
   Although the directory is called "smb", it will be seen as "qemu" network share later.
10. Start the VM and connect via VNC/Spice:
    $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide -monitor stdio -s -vnc :0 -spice port=2001,disable-ticketing -device e1000,netdev=net0 -netdev user,net=10.0.2.3/24,**smb=/root/smb,smbserver=10.0.2.4**,id=net0,**restrict=y** -device qemu-xhci,id=xhci -device usb-tablet,bus=xhci.0 -vga cirrus

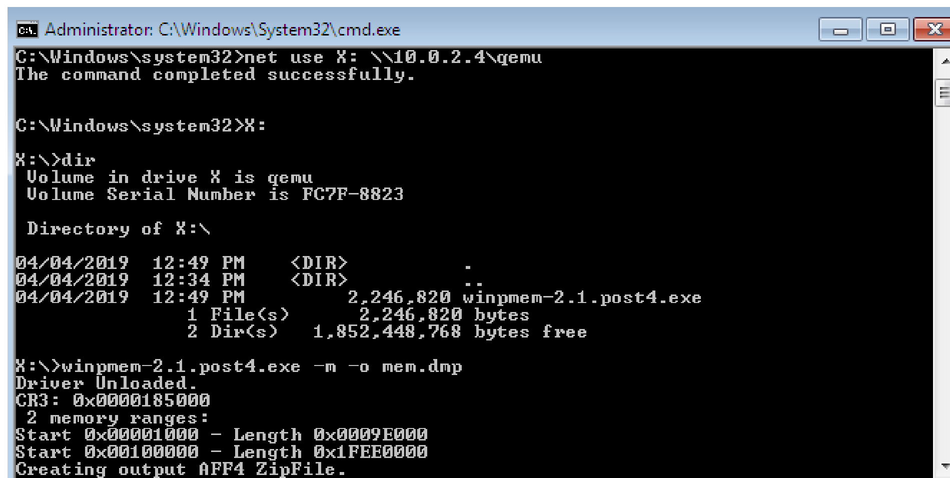
Note:
Qemu provides a feature called userspace network. This way it doesn't need root privileges to create a network adapter connected to the guest OS. It can also connect host OS services or forward TCP ports back and forth to the guest OS. The guest OS sees the host system as a network host in this virtualized host-only network. The option highlighted above is used to start samba daemon and make it visible as 10.0.2.4 host on this virtual net.
Note that qemu doesn't always start samba daemon automatically and you may need to run it manually after VM starts, using the following command:
$ smbd -s /tmp/qemu-smb.%RANDOM_STRING%/smb.conf

Option **restrict=y** is very important, because in case your Bitscout is connected to the internet, it isolates VM from connecting to other hosts on the network, while letting use local services such as samba.
More information about Qemu networking can be found here:
https://wiki.qemu.org/Documentation/Networking
https://www.qemu.org/2018/05/31/nic-parameter/

11. Once the machine is booted and samba daemon is running you should be able to access the shared directory.
    Run cmd.exe as Administrator and map network directory to a local virtual drive, i.e.:
    cmd> net use X: \\10.0.2.4\qemu
12. Copy winpmem tool using scp to the Bitscout shared folder as follows:
    expert $ scp -i %SSHKEY_PATH% %PATH_TO_WINPMEM% root@%BITSCOUT_IP%:smb/

https://t.me/learningnets

13. run winpmem tool saving the output to the same network directory.



The following minimal command is used to make basic memory dump with Winpmem-2.1 (for other versions see tool help output):

```
cmd> winpmem-2.1.post4.exe -m -o mem.dmp
```

Chapter 9. Memory Dump (remote NBD)

Subject VM: win7_32bit_nofde_infected

When size of data read from disk is significantly smaller than the allocated VM RAM, for example in ase of large database server, mail server or a domain controller, you may want to reduce time of transferring large memory dump file to expert's host by starting a VM not on Live OS but on the expert's host.
This is also useful when you need to run some custom tools or signatures that contain software licenses or other limitations that prevent you from transferring and using them on the remote VM. In this case using a network block device (NBD) may be a good solution.
NBD allows you to create a block device on the expert's system and map all read/write operations over the network to a remote block device. Running copy-on-write protection on top of it allows experts to boot Windows locally.

The following are the steps that are required to set up an NBD device and service.

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. The expert edits /etc/nbd-server/config file on Bitscout container to export /dev/host/evidence0 device:

```
[generic]
 user = nbd
 group = nbd
 includedir = /etc/nbd-server/conf.d
 port=2001
```


https://t.me/learningnets

```
allowlist = true

[export]
exportname = /dev/host/evidence0
```

3. Restart nbd daemon to reload the config:
   $ systemctl restart nbd-server.service
4. The expert uses nbd-client to map new exported device via TCP connection on port 2001 of Bitscout:
   expert $ sudo xnbd-client --exportname **export** --connect **/dev/nbd0** %BITSCOUT_IP% **2001**
5. Make the device accessible to all users:
   expert $ sudo chmod o+r **/dev/nbd0**
6. The expert creates copy-on-write protection layer via qcow2 file in RAM:
   $ qemu-img create -f qcow2 -o backing_file=**/dev/nbd0**,backing_fmt=raw ./evidence0.qcow2
7. The VM can be started locally on expert's host:
   expert $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=./evidence0.qcow2,format=qcow2,if=ide -monitor stdio -vnc :0 -spice port=2001,disable-ticketing -vga cirrus

After the last step you may proceed with memory dumping using Qemu guest memory saving feature or VM internal memory dump using winpmem or similar tools.

Chapter 10. Reconstruct Infection Sequence

Subject VM: win7_32bit_nofde_infected

In this chapter, you are given a malicious domain, which was detected by the network IDS and linked to the subject system. You need to identify how the domain was related to the cyberattack and possibly reconstruct the infection sequence.

In this chapter we analyse offline hard drive contents by mounting the C: filesystem and inspecting files' contents. The process is described further:

1. The owner of the system shall map target disk (i.e. /dev/sda) to evidence0 device and enable privileged mount command discussed in Disk Image Acquisition exercise (Disk->Privileged Mode->Enable).
2. The expert lists partitions on the evidence0 drive with mmls command:

```
root@bitscout:~$ mmls /dev/host/evidence0
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
```


https://t.me/learningnets

```
     Slot        Start        End          Length       Description
000:  Meta        0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------     0000000000   0000002047   0000002048   Unallocated
002:  000:000     0000002048   0000206847   0000204800   NTFS / exFAT (0x07)
003:  000:001     0000206848   0025163775   0024956928   NTFS / exFAT (0x07)
004:  -------     0025163776   0025165823   0000002048   Unallocated
```

Note that target C: partition starts from sector 206848 (each sector is 512 bytes long), and it contains 24956928 sectors of data. You need to identify such partition in your case and use these values in the following mount command.

3. Make new directory to mount drive C: and use mount.priv command to mount NTFS partition:
   $ mkdir /mnt/C
   $ mount.priv -o ro,show_sys_files,streams_interface=windows,norecover,loop,offset=$[512***206848**],sizelimit=$[512***24956928**] -t ntfs-3g **/dev/host/evidence0 /mnt/C**

   After the last command make sure /mnt/C directory contains Windows files and directory structure.
4. Create Yara rule file (i.e. in ./rule.yara) or use grep to find files containing the suspicious domain.

```
rule malware_domain {
  strings:
   $domain = "charlesprofile.website" wide ascii
  condition:
   $domain
}
```

5. Scan the subject system with your new Yara rule:
   $ yara -r ./rule.yara /mnt/C
6. Analyse files that contain the suspicious domain.

Note:
A tool called **reged** from chntpw package can be used to export contents of Windows registry hives into text form which can be analysed with simple tools such as grep or less.
However, Windows registry also contains timestamps which can be extracted using specialised tools. One such tool is RegRipper by H.Harvey (https://github.com/keydet89). A variant of it is provided in tools/regparser/regp.pl. It can convert binary registry hives into text files with timestamps.

https://t.me/learningnets

Chapter 11. Lazarus Backdoor & Persistence Chain

Subject VM: win7_32bit_nofde_infected

In this chapter you need to unlock the user account and promote the user to Administrators, boot the subject system in a VM and analyse all processes that created listening ports. Once a suspicious port is identified, you must recover the other components used to help malware start on system boot.

The stage of enabling local login is identical to the exercise of Memory Dump (Winpmem). After following those steps, you can use netstat command on Windows command line started as Administrator to list TCP network connections and owning processes as follows

```
cmd.exe> netstat -atnb
```

This may give you some hints about the owning process, but you still need to discover how the malware starts automatically and collect all related malware files.

In the end you should discover 4 Windows executable files related to passive backdoor activity.

Chapter 12. BitLocker & ShadowHammer Backdoor

Subject VM: win7_64bit_bitlocker.shadowhammer

In this exercise you will deal with an infected system running 64-bit Windows and Bitlocker encryption. Network IDS reported suspicious incoming HTTP requests with large data transfered (according to the headers). The host is running a regular Windows workstation and is not supposed to have a web server installed. Your objective is to find the webserver and identify potential backdoor. Sample HTTP URL detected by the network IDS:
http://%HOST_IP%/requested.html

BitLocker, used on current setup, is a full volume encryption feature included with Microsoft Windows versions starting with Windows Vista. It is designed to protect data by providing encryption for entire volumes. By default, it uses the AES encryption algorithm in cipher block chaining (CBC) or XTS mode with a 128-bit or 256-bit key.

BitLocker is an example of commonly used full-disk encryption (FDE). Just like other commonly used FDE software (i.e. VeraCrypt, etc) it has good support from the community and there are tools that let you mount encrypted drives directly on Linux. One such free and open source tool is called **dislocker**. You may need to use this tool to mount BitLocker-protected partitions. Of course, it assumes that you have the protected storage key or recovery code to mount encrypted volume.

https://t.me/learningnets

As far as dislocker is not installed in Bitscout this exercise highlights the feature of dynamic extension of Bitscout tools and packages by downloading and installing them from the repository right in the Live OS.

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. To use dislocker you need to install the tool and it's dependencies. First of all you need to make sure Internet connection works. Ping some Internet host such as 8.8.8.8 on Bitscout:
   $ ping 8.8.8.8
   If this works for you, try pinging a host by domain name, i.e.:
   $ ping google.com
   If ping by domain name doesn't work, you need to change system DNS settings in /etc/resolv.conf by editing the file and making sure the following line is present there (if you want to use Google public DNS):

```
nameserver 8.8.8.8
```

3. Update packages index from the repository:
   $ apt update
4. Install dislocker:
   $ apt install dislocker

Note:
Dislocker is based on a filesystem in userspace (FUSE). It creates a virtual file that contains a decrypted variant of the BitLocker-encrypted parition.
More about FUSE you can read here: https://en.wikipedia.org/wiki/Filesystem_in_Userspace

5. Next use dislocker to make a FUSE mount point with decrypted partition:
   $ mkdir /mnt/dislocker
   $ dislocker -f %PATH_TO_BEK_FILE% -O $[512***%PARTITION_OFFSET_IN_BLOCKS%**] -r -V /dev/host/evidence0 /mnt/dislocker
   Where **%PARTITION_OFFSET_IN_BLOCKS%** is something you need to find with mmls command. This was discussed previously in the Reconstruct Infection Sequence exercise.
   Also, prior to running this command you need to copy *.BEK file containing the encryption key for the BitLocker volume to Bitscout.
   Once this command completes, you should be able to see /mnt/dislocker/dislocker-file which size will be equal to the size of decrypted disk and it will contain decrypted volume data. You can use it to mount the filesystem as per the following command.
6. Mount the NTFS filesystem:
   $ mkdir /mnt/C

https://t.me/learningnets

```
$ mount -t ntfs-3g -o ro /mnt/dislocker/dislocker-file /mnt/C/
```

Make sure that after this command you can see the contents of Windows drive mounted to /mnt/C. If everything is OK, you can make a full disk copy without a layer of BitLocker by cloning the contents of /mnt/dislocker/dislocker-file.

7. Next, let's boot the BitLocker-protected Windows system in a VM and interact with it to locate the malware. Use usual copy-on-write protection:
   ```
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
   ```
8. When starting a VM you need to pass a new option that creates a virtual drive with FAT32 filesystem which contents is mapped to a directory in Bitscout. This is a convenient but somewhat limited way to exchange information between Bitscout and guest VM. However, this is essential, because Windows BitLocker searches for crypto-keys on attached drives and requires a valid key to boot the OS. The BEK file shall be placed inside this virtual disk directory.
   ```
   $ mkdir ./usbdisk
   ```
   Copy the BEK file to ./usbdisk/ directory.
   ```
   $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide -monitor stdio -vnc :0 -spice port=2001,disable-ticketing -vga cirrus -net user,net=10.0.2.3/24,id=usernet -net nic -s -hdb fat:rw:usbdisk
   ```
   Connect and see Windows automatically finds the decryption key and boots from the BitLocker-protected partition.
9. Once the machine is up you need to confirm the presence of a backdoor on TCP port 80 and locate the malware modules.

In the end, you should identify 2 Windows malicious PE files.

Chapter 13. Building a Honeypot

Subject VM: win7_64bit_bitlocker.shadowhammer

Silent and invisible analysis of compromised systems is critically important for collection of subject material. Sometimes malware discovered on the systems neither contains any functional payload, nor tells you anything about the attacker, who might still be in the network of the attacked party. Knowledge about passive backdoor such as ShadowHammer can be extended with information collected during attackers activity live. In this case you may need to build a system that looks and behaves on the network like the real one with certain differences: it doesn't contain any valuable information to steal anymore, it records all traffic coming to it, it automatically produces memory dumps once attackers are connected to it and start working. The system may also automatically shutdown or suspend execution after a certain time since attackers connected to it, simulating unstable systems and preventing attackers from revealing the clone.

https://t.me/learningnets

In this chapter we need to use hdd of the subject and make the system accessible on the network for the attackers using a specific port they used for the backdoor (TCP port 80).

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. Apply copy-on-write protection:
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
3. Similarly to exercise with Bitlocker, copy the BEK decryption key into /root/usbdisk.
4. Start the VM
   $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide **-monitor tcp:127.0.0.1:5555,server,nowait** -vnc :0 -spice port=2001,disable-ticketing -vga cirrus -device e1000,netdev=net0 -netdev user,net=10.0.2.3/24,id=net0,restrict=y,**hostfwd=::2080-:80** -s -hdb fat:rw:usbdisk

> **Note:**
> Qemu allows the export monitor console (the one used to control qemu with typed commands) via various modes of transport. Previously we used stdio, but this time an option to export monitor via TCP server socket is used. This is useful when you need to automate interaction with qemu.

5. Once a VM is running, you need to connect to it, let it fully boot, login as user and let Windows detect new network settings. After that, disable Windows Firewall to let incoming connections to the machine. At this stage you may delete all sensitive files or other information that the attackers shall not access if they connect to this host again. Note, that all files will be deleted only in copy-on-write cache, but not from the original hard drive. That means, if you recreate a qcow2 file, make sure you delete them again every time.
6. Next, we should build a chain of port-forwarding from the external network to the target malware service. This is done selectively per port, to prevent other potentially uncontrolled means of communication. So, we just need to let attackers connect to this host via TCP port 80.
   Change of firewall (iptables) rules on the host system of Bitscout is required and cannot be achieved without supervised command such as shown below (run "supervised-shell" to input supervised command) :
   supervised> iptables -t nat -I LXC_INCOMING -p tcp --dport 80 -j DNAT --to-destination 10.3.0.2:1080
   This command is going to forward all ingress TCP connections on port 80 to IP 10.3.0.2 (expert container IP) and TCP port 1080.
7. After this, you need to make a script that will be called upon every incoming connection to port 80 of the host. It may contain any logic, but the following should help you instruct

https://t.me/learningnets

Qemu to dump memory after 15 seconds after connection and at the same time link the incoming connection to Qemu to forward it to the malware service running inside the VM. Create a file in /root/on_connection.sh with the following contents:

```
#!/bin/bash
( sleep 15; echo "dump-guest-memory ./mem_$(date  +%H-%M-%S).dmp" |
nc -q0 127.0.0.1 5555  2>/dev/null >/dev/null ) &
socat stdio tcp-connect:127.0.0.1:2080
```

You can see the Qemu monitor command passed to the monitor via TCP (using netcat). The netcat connects to port 5555 where the Qemu monitor is listening, passes the command and quits immediately. The command to dump guest memory uses timestamp to differentiate memdumps. Here you can also make a loop to make a memory dump every N seconds. It's best to store to external storage or transfer over the network, but for testing purposes it's ok to save it into the same RAM.
Do not forget to make this script executable with
`$ chmod +x ./on_connection.sh`

8. Finally start the last link in the chain: data bouncer tool called socat, as follows.
`$ socat -v tcp-listen:1080,fork,reuseaddr exec:./on_connection.sh`
Socat options instruct it to listen on TCP port 1080 and forward data to the standard input of the on_connection.sh script. The script sets a timer to dump memory after 15 seconds and forward data further to Qemu, which relays it to the service running inside.

Overall the scheme of data and components interaction looks like shown below.

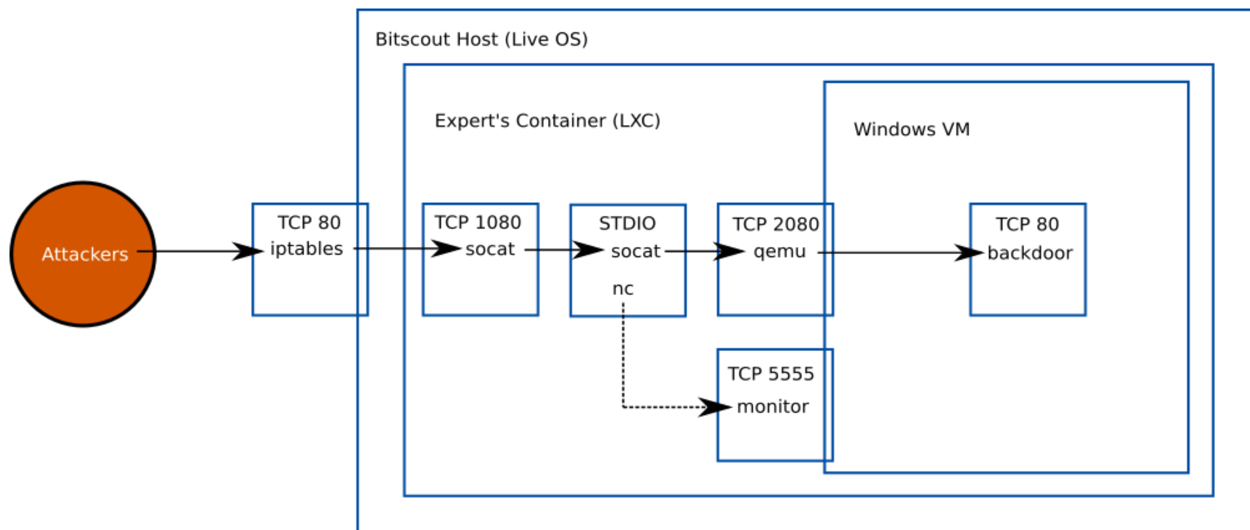


Fig. Data and control transfer.

If everything was done right, you should be able to connect to the Bitscout host on port 80 and send simple HTTP request such as

```
GET /requested.html HTTP/1.0
\n
\n
```

The malware shall respond accordingly and after 15 seconds new file /root/mem_*-*-*.dmp shall appear with a full memory dump.
If that works, try modifying the on_connection.sh script to poweroff or suspend the VM after 1 minute.

Chapter 14. Enter Custom FDE

Subject VM: win10_64bit_custom_mbrfde_infected

In this chapter we discover a malicious implant that beacons back to the hardcoded C2 server.

There are 2 places where FDE may be implemented:

- Based on Master Boot Record (MBR) interrupt handler
- Based on Extensible Firmware Interface (EFI) driver

The first case to be studied is using MBR.

Let's analyse the disk, and the MBR in particular - the first 512 bytes of the disk, with dd.

```
root@bitscout:~$ dd if=/dev/host/evidence0 count=1 bs=512 | xxd
00000000: 33c0 9090 eb10 1000 1e00 0000 0020 daff  3............ ..
00000010: 7f01 0000 0000 ea1b 7c00 00fa 31c0 8ed8  ........|...1...
00000020: 8ed0 bc00 40fb 8b1e 4c00 8e06 4e00 2666  ....@...L...N.&f
00000030: 817f 02bb 0147 6d74 0ae8 a800 7205 31db  .....Gmt....r.1.
00000040: 0653 cbbd 007c b8e0 1f8e c089 ee89 efb9  .S...|..........
00000050: 0002 fcf3 a406 685a 7ccb 8cc8 8ed8 bfbe  ......hZ|.......
00000060: 7df6 0580 7526 83c7 1081 fffe 7d72 f2e8  }...u&......}r..
00000070: f300 6e6f 2061 6374 6976 6520 7061 7274  ..no active part
00000080: 6974 696f 6e20 666f 756e 6400 668b 4508  ition found.f.E.
00000090: 66a3 0e7c 6631 db66 891e 127c 891e 0c7c  f..|f1.f...|...|
000000a0: 892e 0a7c 4389 1e08 7ce8 3800 7313 e8b4  ...|C...|.8.s...
000000b0: 0064 6973 6b20 7265 6164 2065 7272 6f72  .disk read error
000000c0: 0026 813e fe7d 55aa 7417 e898 0069 6e76  .&.>.}U.t....inv
000000d0: 616c 6964 2062 6f6f 7420 7365 6374 6f72  alid boot sector
000000e0: 0006 55cb 6089 d5ff 360c 7c07 6631 c066  ..U.`...6.|.f1.f
000000f0: 3906 127c 750b 6681 3e0e 7c00 c00f 0072  9..|u.f.>.|....r
```


https://t.me/learningnets

```
00000100: 1db4 41bb aa55 cd13 7214 81fb 55aa 750e  ..A..U..r...U.u.
00000110: f6c1 0174 09be 067c b442 89ea eb41 b408  ...t...|.B...A..
00000120: 89ea 06cd 1307 72ed 80e1 3ffe c666 0fb6  ......r...?..f..
00000130: c966 0fb6 f666 a10e 7c66 31d2 66f7 f142  .f...f..|f1.f..B
00000140: 88d1 31d2 66f7 f688 d688 c5c1 e802 24c0  ..1.f.........$.
00000150: 08c1 a108 7cb4 0289 eb88 da8b 1e0a 7c06  ....|.........|.
00000160: cd13 0761 c35e ac84 c074 feb4 0e31 dbcd  ...a.^...t...1..
00000170: 10eb f300 0000 0000 0000 0000 0000 0000  ................
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001b0: 656d 0000 0063 7b9a 3370 1ece 0000 8020  em...c{.3p.....
000001c0: 2100 07dd 1e3f 0008 0000 00a0 0f00 00dd  !....?..........  <-- [1]
000001d0: 1f3f 07fe ffff 00a8 0f00 0050 7001 0000  .?.........Pp...  <-- [2]
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

Notes:
Information on MBR can be found here:
https://thestarman.pcministry.com/asm/mbr/PartTables.htm
https://thestarman.pcministry.com/asm/mbr/PartTables3.htm

At 0x1be we have the first partition: ”**80**`20 2100` **07**`dd 1e3f` **0008 00**`00 00a0 0f00`” and the main information reads as follow:

Byte(s) Offset	Value in this Example	Description	Meaning
1BE	80	Bootable? (80h = Yes; 00 = No)	YES
1C2	07	Partition Type	NTFS
1C6 - 1C9	00 08 00 00	Relative Sectors (or Offset)	0x800 = 2048

At 0x1ce we have the second partition: ”**00**`dd 1f3f` **07**`fe ffff` **00a8 0f**`00 0050 7001`”

Byte(s) Offset	Value in this Example	Description	Meaning

1CE	00	Bootable? (80h = Yes; 00 = No)	NO
1D2	07	Partition Type	NTFS
1D6 - 1D9	00 A8 0F 00	Relative Sectors (or Offset)	0x0fa800 = 1026048

This is confirmed by the use of automated tools such as mmls (from The Sleuthkit project):

```
root@bitscout:~$ mmls /dev/host/evidence0
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0001026047   0001024000   NTFS / exFAT (0x07)
003:  000:001   0001026048   0025163775   0024137728   NTFS / exFAT (0x07)
004:  -------   0025163776   0025165823   0000002048   Unallocated
```

It is then possible to inspect the contents of each of the partitions by reading the values at the offsets it start, by using dd and xxd, or hexedit directly.

```
root@bitscout:~$ dd if=/dev/host/evidence0 count=1 bs=512 skip=2048 status=none|
xxd | head -5
00000000: eb52 904e 5446 5320 2020 2000 0208 0000  .R.NTFS    .....
00000010: 0000 0000 00f8 0000 3f00 ff00 0008 0000  ........?.......
00000020: 0000 0000 8000 8000 ff9f 0f00 0000 0000  ................
00000030: aaa6 0000 0000 0000 0200 0000 0000 0000  ................
00000040: f600 0000 0100 0000 f214 6ada 3b6a da18  ..........j.;j..

root@bitscout:~$ dd if=/dev/host/evidence0 count=1 bs=512 skip=1026048
status=none| xxd | head -5
00000000: 8ba3 53ca 36b1 0d1c f349 3809 5517 cb6b  ..S.6....I8.U..k
00000010: 6be2 2a9e b713 8e61 66f0 e8e6 e085 0efc  k.*....af.......
00000020: 7a64 8877 ab8a 87d3 2b1e 94c3 6b4e 08ca  zd.w....+...kN..
00000030: c4b1 f16c 485d 4278 682d 78c1 c407 ea57  ...lH]Bxh-x....W
00000040: bd22 0780 f121 8dcc 61ac 035d d6b6 c7dc  ."...!..a..]....
```

While we can immediately notice the NTFS Header for the first partition, the second partition looks encrypted. You may also verify this by calculating the information entropy of any part of data on the disk, i.e. first 100 sectors using tools or a short python script as demonstrated below:

https://t.me/learningnets

```
root@bitscout:~$ dd if=/dev/host/evidence0 count=100 bs=512 skip=1026048
status=none > /tmp/part2.bin
root@bitscout:~$ python
>>> import math
>>> from collections import Counter
>>>
>>> def entropy(s):
...     p, lns = Counter(s), float(len(s))
...     return -sum( count/lns * math.log(count/lns, 2) for count in
p.values())
...
>>> entropy("abcdefghijklmnopqrstuvwxyz")
4.700439718141092
>>> entropy("1223334444")
1.8464393446710154  ← not encrypted, low entropy
>>> f=open('/tmp/part2.bin', 'r')
>>> entropy(f.read())
7.996734466510677 ← not encrypted, high entropy (>=6.6)
```

Script available at https://rosettacode.org/wiki/Entropy#Python:_More_succinct_version

For now raw disk seems to have no recognisable header or magic string, so it is possibly using unknown algorithms or proprietary software which we have no tool for. It means that the filesystem of the infected OS cannot be mounted and checked in offline mode. However, as long as the user can boot the system using credentials, you should be able to virtualise and boot it too and inspect it by interacting with it.

Chapter 15. Locating Suspicious Process

Subject VM: win10_64bit_custom_mbrfde_infected

At this stage let's focus on confirming that the subject system is infected. We can boot and analyse decrypted the file system once the OS is fully started.

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. The expert creates copy-on-write protection layer via qcow2 file in RAM:
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
3. Next the expert starts VM from the newly create VM disk:
   $ mkdir /root/smb
   $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 768 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide -monitor stdio -s -vnc :0 -spice port=2001,disable-ticketing -device e1000,netdev=n0 -netdev

https://t.me/learningnets

```
user,net=10.0.2.3/24,smb=/root/smb,smbserver=10.0.2.4,id=n0,restrict=y -device
qemu-xhci,id=xhci -device usb-tablet,bus=xhci.0
```

Like it was discussed previously, qemu doesn't always start samba daemon automatically and you may need to run it manually after VM starts, using the following command:

```
$ smbd -s /tmp/qemu-smb.%RANDOM_STRING%/smb.conf
```

4. Connect to the screen of the VM using VNC or Spice and enter FDE password (provided by the owner, i.e. "12345" for the training image).

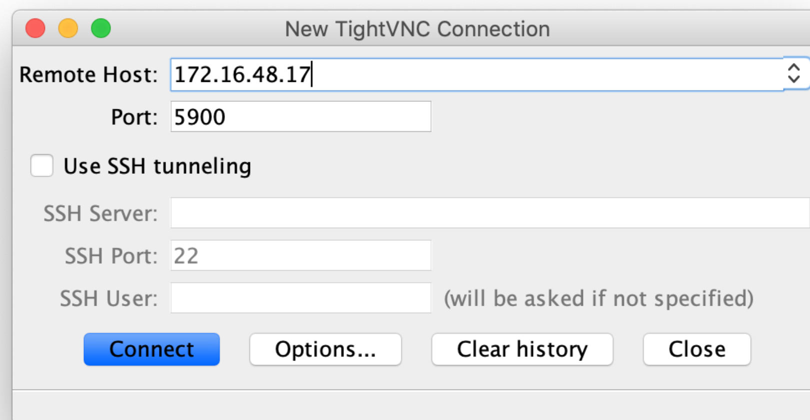


Fig. Connecting to VM console using VNC on macOS.

https://t.me/learningnets

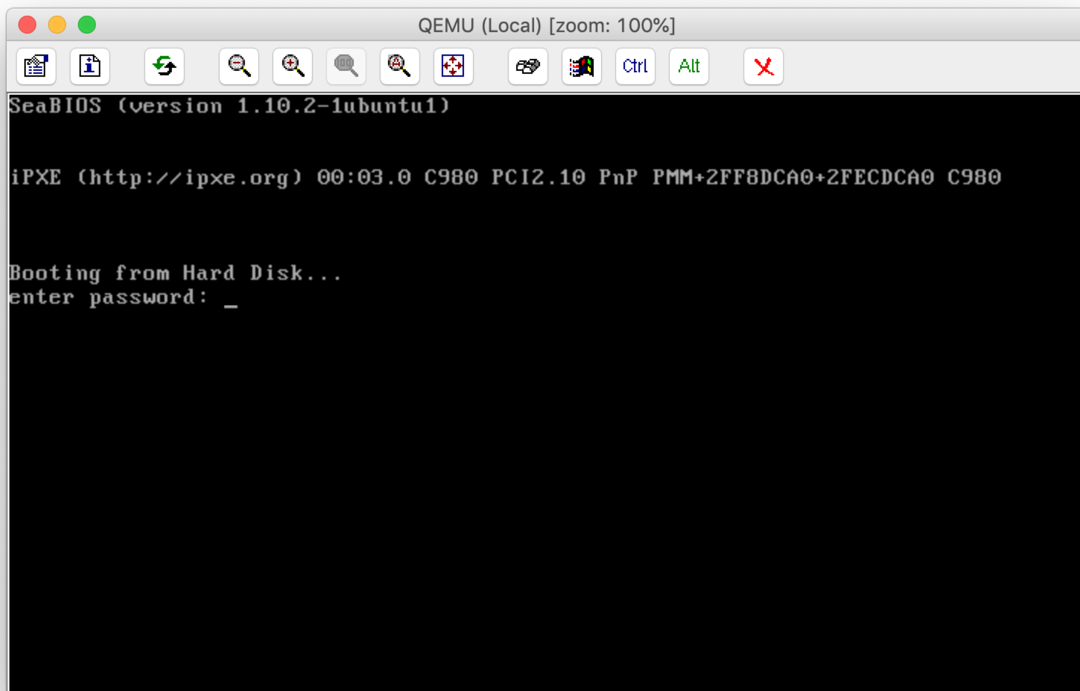


Fig. Disk encryption password prompt before Windows boot.

The first task is to identify the process responsible for the connection to suspicious IP. To analyse open connections you may use netstat Windows command or Sysinternals tools, such as TCPView.

> **Note:**
> Sysinternals tools can be downloaded from Microsoft website:
> https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite

You may pass Sysinternals tools suite to the expert's computer, they can then be copied over to the Bitscout container and accessed from the booted OS using Samba shared folder.

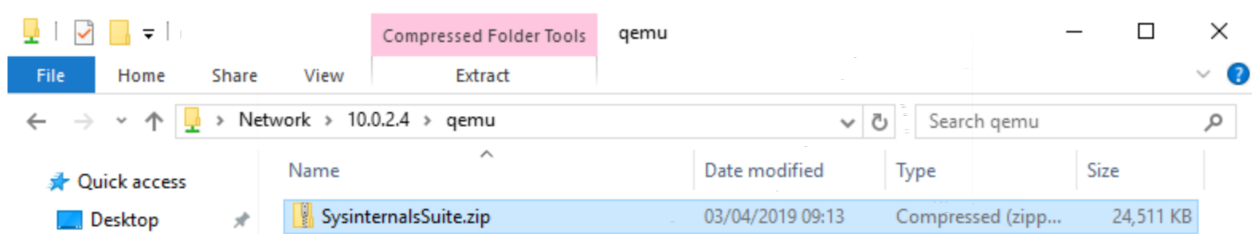


Fig. Samba-shared folder at 10.0.2.4 (as per qemu command line)

https://t.me/learningnets

TCP View, one of Sysinternals tools, should be able to spot the potential connection to the specified IP.

As the process was started with higher privileges and the user doesn't have any administrative privilege, the analysis might be limited.

Elevation of Privileges

One of the techniques to elevate the privileges of the currently logged on user is to start a process such as cmd.exe, unprivileged and allocate the security token of another process such as "System" (PID 4).

Let's attempt to elevate the privileges of **cmd.exe**, the command prompt of Windows, currently running as the unprivileged user.

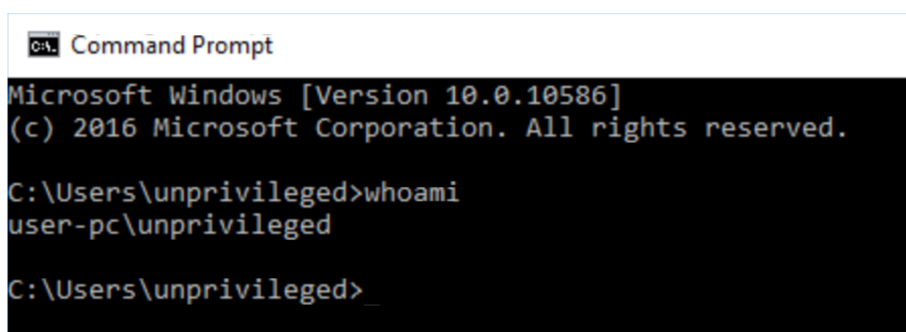


Fig. Example of an unprivileged command prompt

With Qemu, the memory can be dumped onto a file for analysis - to identify the offsets and tokens.

In order to preserve the state of the virtual machine, from the qemu console, we can "**stop**" the execution then "**dump-guest-memory**" to save the memory onto a designated file.

```
(qemu) stop
(qemu) dump-guest-memory /tmp/mem.dump
```

In another terminal, let's focus on the analysis of the memory dump with volatility.

While volatility sometimes gets the characteristics of the OS being analysed, it is safer to indicate the profile to use.

The volatility command "**imageinfo**" can help determine the right value.

```
root@bitscout:/tmp$ volatility -f /tmp/mem.dump imageinfo
Volatility Foundation Volatility Framework 2.6
```


https://t.me/learningnets

```
INFO: volatility.debug: Determining profile based on KDBG search...
Suggested Profile(s) : Win10x64_10586, Win10x64_14393, Win10x64,
Win2016x64_14393, Win10x64_15063 (Instantiated with Win10x64_15063)
                     AS Layer1 : SkipDuplicatesAMD64PagedMemory (Kernel AS)
                     AS Layer2 : QemuCoreDumpElf (Unnamed AS)
                     AS Layer3 : FileAddressSpace (/tmp/mem.dump)
                      PAE type : No PAE
                           DTB : 0x1aa000L
                          KDBG : 0xf801f86d6a60L
          Number of Processors : 1
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0xfffff801f872f000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2019-04-04 03:16:06 UTC+0000
     Image local date and time : 2019-04-04 04:16:06 +0100
```

Once the profile is identified, we can specify it in the subsequent commands. Let's start volatility in an interactive mode - "**volshell**" - to manipulate the memory structures, with the profile "**Win10x64_14393**", with the following command.

```
root@bitscout:~$ volatility --profile=Win10x64_14393 -f /tmp/mem.dump
volshell
```

Notes:
Help can be provided by using the hh command in volshell.

hh(cc)
Change current shell context.

This function changes the current shell context to to the process
specified. The process specification can be given as a virtual address
(option: offset), PID (option: pid), or process name (option: name).

If multiple processes match the given PID or name, you will be shown a
list of matching processes, and will have to specify by offset.

https://t.me/learningnets

Once the interactive shell has started, we can change the current shell context to the process with the Process ID = 4 (PID=4).

Notes:
PID 4 (System) is the first “process” created by the kernel and only runs in kernel mode with the highest level or privileges: NT\System

(Windows Internals 6, Part1, CHAPTER 2 System Architecture p69)

This can be achieved by the command “**cc**” such as:
volshell> cc(pid=4)

The offset of the security token and its value can be printed by using the “**dq**” command that prints the data at the given address, interpreted as a series of qwords (unsigned eight-byte integers) in hexadecimal.

volshell> dq(proc().Token.obj_offset,1)

Let’s take note of the value of the security token (the 2nd value).

Switching the context of the current process needs to be changed to now work on the targeted process - in our case cmd.exe. Volshell supports the attribute “**name**” to designate the process, as opposed to the “**pid**” used previously.

volshell> cc(name='cmd.exe')

The offset of the security token and its value can be printed by using the dq command again.

volshell> dq(proc().Token.obj_offset,1)

Let’s take note of the offset of the security token (the 1st value).

We now have everything necessary to associate the security token of the process with a pid=4 to cmd.exe and effectively elevate the privileges of the latter. It can be described by the simple equation:

$$security_token(process_name = cmd.exe) := security_token(pid = 4)$$

Process	**Security Token Offset**	**Security Token Value**
PID 4	0xffffe00181a54398	**0xffffc000b0218ab5**
cmd.exe (before)	**0xffffe0018372ab98**	0xffffc000b4759ab4

cmd.exe (after)	0xffffe0018372ab98	0xffffc000b0218ab5

An example of a volshell session is shown below:

```
root@bitscout:~$ volatility --profile=Win10x64_14393 -f ./mem.dump volshell
Volatility Foundation Volatility Framework 2.6
Current context: System @ 0xffffe00181a54040, pid=4, ppid=0 DTB=0x1aa000
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: cc(pid=4)
Current context: System @ 0xffffe00181a54040, pid=4, ppid=0 DTB=0x1aa000

In [2]: dq(proc().Token.obj_offset,1)
0xffffe00181a54398 0xffffc000b0218ab5

In [3]: cc(name='cmd.exe')
Current context: cmd.exe @ 0xffffe0018372a840, pid=3852, ppid=2868
DTB=0x2b8ed000

In [4]: dq(proc().Token.obj_offset,1)
0xffffe0018372ab98 0xffffc000b4759ab4
```

In another terminal, with GDB (GNU Debugger), you need to patch the VM memory to change the cmd.exe security token accordingly (patch with System process security token).

After security token is patched resume the VM with “**continue**” (or shortened "c") command.

```
root@bitscout:~$ gdb -ex 'target remote localhost:1234'
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
Remote debugging using localhost:1234
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0xfffff802a7219a2d in ?? ()

(gdb) set *0xffffe001aebffb98=0xffffc000b8618ab8
(gdb) c
Continuing.
```


https://t.me/learningnets

The Command prompt should now have maximum privileges and “**whoami**” can validate it to reflect “**nt authority\system**”.

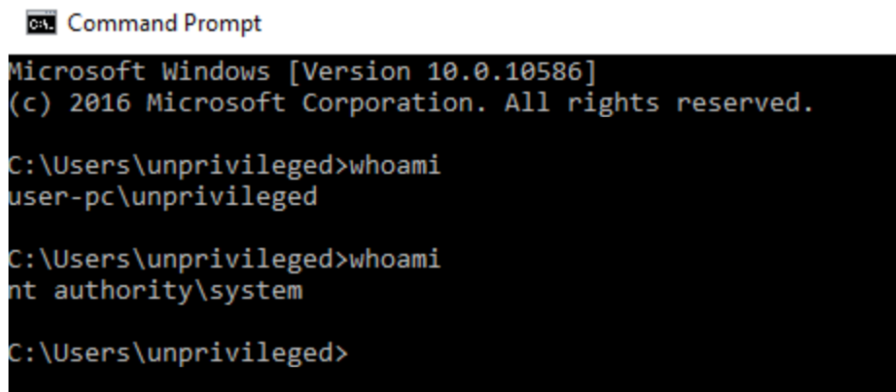


Fig. The Command Prompt is now running as a privileged user.

It is then possible to start other processes from Command Prompt that will inherit the same privileges.

Notes:
Some additional details can be found here:
https://downloads.volatilityfoundation.org/omfw/2012/OMFW2012_Gurkok.pdf (Tokens explanations)
https://diablohorn.com/2017/12/12/attacking-encrypted-systems-with-qemu-and-volatility/ ← trick we used

Complete Malware Analysis

From the Command Prompt with elevated privileges, you may
Next, you should locate the process that starts suspicious egress connection by a global IP. To confirm that this is malicious you need to find and analyse the code which is responsible for initiating the connection.

Process Explorer by Sysinternals, started with higher privileges, should now be able to reveal additional process properties that may help in understanding the root cause of the connection.

You now just have to decode potential hidden information to find out the malware connects to the command and control server.

https://t.me/learningnets

Notes:
It is also possible to suspend the entire virtual machine to prevent Windows OS from creating too many unnecessary disk writes. Use qemu console "stop" command to suspend VM execution, and "continue" to resume.

```
(qemu) stop
(qemu) continue
```

Chapter 16. Finding Malware by IOC

Subject VM: win10_64bit_custom_mbrfde_infected

You have received indicators suggesting that your network was breached by an advanced threat actor using malware signed by hijacked digital certificate. You have the certificate serial number (**68:21:19:09:85:a8:75:54:3e:49:38:d4:5c:ea:9d:cb**) from the researchers, who reported it to you and need to scan subject system which is suspected to be infected with this malware strain.

In this chapter we simulate the case where you got access to the subject system but missing any credentials to log in to the OS. This is similar to the LockerGoga attack described previously, where all users' passwords are changed and accounts are locked out. Last time we used chntpw to modify a SAM file, however this time this technique doesn't work, because the filesystem is encrypted and not accessible to you from the outside of the VM.

You will first have to find a way to log in, preferably with admin privileges, then discover malware in system directories using a given IOC.

Gaining Access Into The System

Winlogon.exe is one of the processes first started and handles logon process and user authentication. It is started with "**nt authority\system**" user account.

https://t.me/learningnets

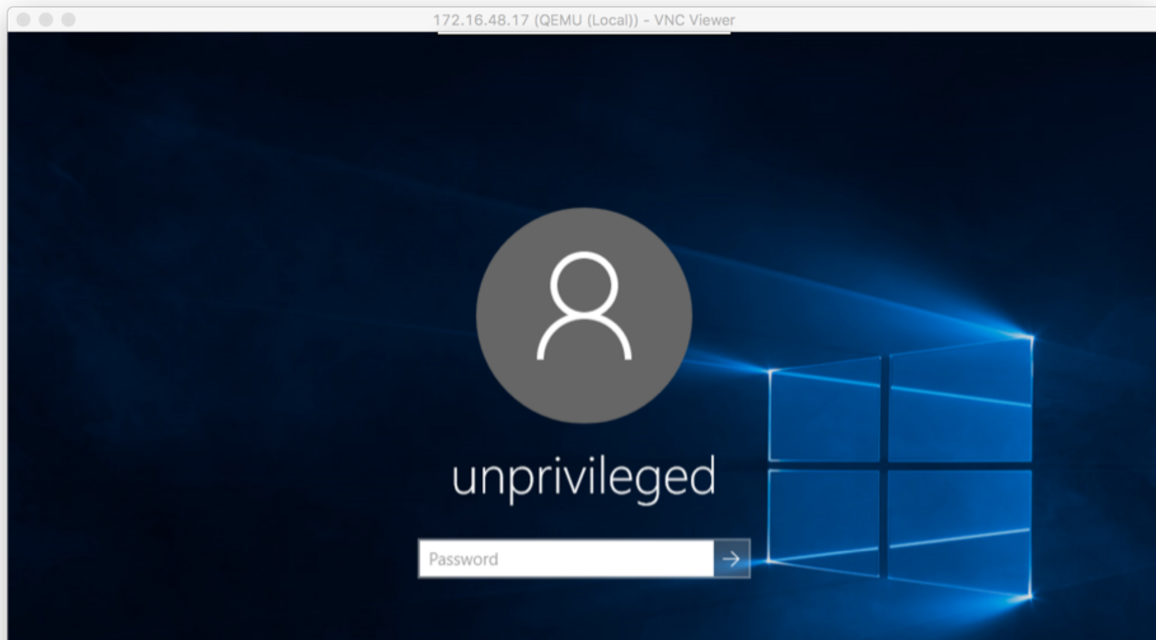


Fig. Windows login prompt by Winlogon.exe

The Winlogon process handles accessibility support such as high contrast UI theme or so-called "Sticky keys". The tool to handle accessibility features is activated with [Shift] key pressed 5 times. The hotkey makes Winlogon execute the system's “**sethc.exe**” executable (the name is hardcoded into Winlogon binary) to change the system settings.

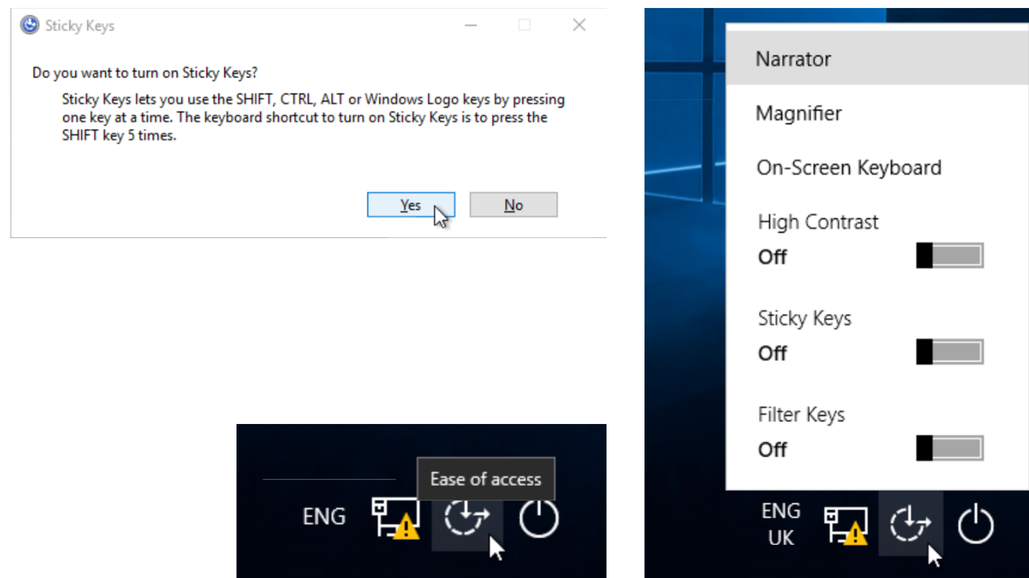


Fig. Windows accessibility features triggering run of sethc.exe

https://t.me/learningnets

Since you can access the memory of the Windows VM, you can replace the path of "**sethc.exe**" with the path of another executable to be started, i.e. "**cmd.exe**". This trick works in all actual Windows versions and has been widely known for several years.

There are many ways to access and modify memory of a VM, but the best is to edit memory of a suspended or stopped machine. One such simple method relies on "**savevm**" command in Qemu, which produces a snapshot containing full physical memory of the VM saved in qcow2 file. All you need to do is to modify the contents of the qcow2 file, and restore the snapshot with the "**loadvm**" command.

```
QEMU 2.11.1 monitor - type 'help' for more information
(qemu) stop
(qemu) savevm snapshot1
```

In a different terminal:

```
root@bitscout:~$ hexedit /root/evidence0.qcow2
// Press / to search
// Hex string to search ("sethc.exe" in Unicode):
730065007400680063002E00650078006500200025006C006400000000000000730065007400
680063002E006500780065
1A8C14A0  6E 00 69 00  6E 00 67 00  20 00 53 00  6F 00 75 00  n.i.n.g. .S.o.u.
1A8C14B0  6E 00 64 00  73 00 00 00  00 00 73 00  65 00 74 00  n.d.s.....s.e.t.
1A8C14C0  68 00 63 00  2E 00 65 00  78 00 65 00  20 00 25 00  h.c...e.x.e. .%.
1A8C14D0  6C 00 64 00  00 00 00 00  00 00 73 00  65 00 74 00  l.d.......s.e.t.
1A8C14E0  68 00 63 00  2E 00 65 00  78 00 65 00  00 00 00 00  h.c...e.x.e.....
1A8C14F0  00 00 C0 8C  B4 A8 BA 93  0B 5E 8B E0  4E DD B6 90  .........^..N...

// Copy & Paste the following string to replace the sethc.exe path:
63006D0064002E0065007800650000000000200025006C00640000000000000063006D00640
02E00650078006500000000
1A8C14A0   6E 00 69 00  6E 00 67 00  20 00 53 00  6F 00 75 00  n.i.n.g. .S.o.u.
1A8C14B0   6E 00 64 00  73 00 00 00  00 00 63 00  6D 00 64 00  n.d.s.....c.m.d.
1A8C14C0   2E 00 65 00  78 00 65 00  00 00 00 00  20 00 25 00  ..e.x.e..... .%.
1A8C14D0   6C 00 64 00  00 00 00 00  00 00 63 00  6D 00 64 00  l.d.......c.m.d.
1A8C14E0   2E 00 65 00  78 00 65 00  00 00 00 00  00 00 00 00  ..e.x.e.........
1A8C14F0   00 00 C0 8C  B4 A8 BA 93  0B 5E 8B E0  4E DD B6 90  .........^..N...

[ctrl]-x
Save changes (Yes/No/Cancel) ? Y
```

```
Notes:
Depending on the Windows version, the strings may be slightly different.
Here is a couple examples for Windows 10 32-bit and 64-bit:

Search for:
730065007400680063002E00650078006500200025006C00640000000000000073006500740
0680063002E006500780065 (win10_64)
730065007400680063002e00650078006500200025006c0064000007300650074006800630
02e006500780065 (win10_32)

Replace by:
63006D0064002E00650078006500000000002000025006C0064000000000000063006D00640
02E006500780065000000000(win10_64)
63006d0064002e00650078006500000000002000025006c00640000063006d0064002e00650
078006500000000 (win10_32)
```

Back to the qemu console:

```
(qemu) loadvm snapshot1
(qemu) cont
```

Now, pressing the [Shift] key 5 times should pop a Command Prompt window running as “**nt authority\system**”.

https://t.me/learningnets

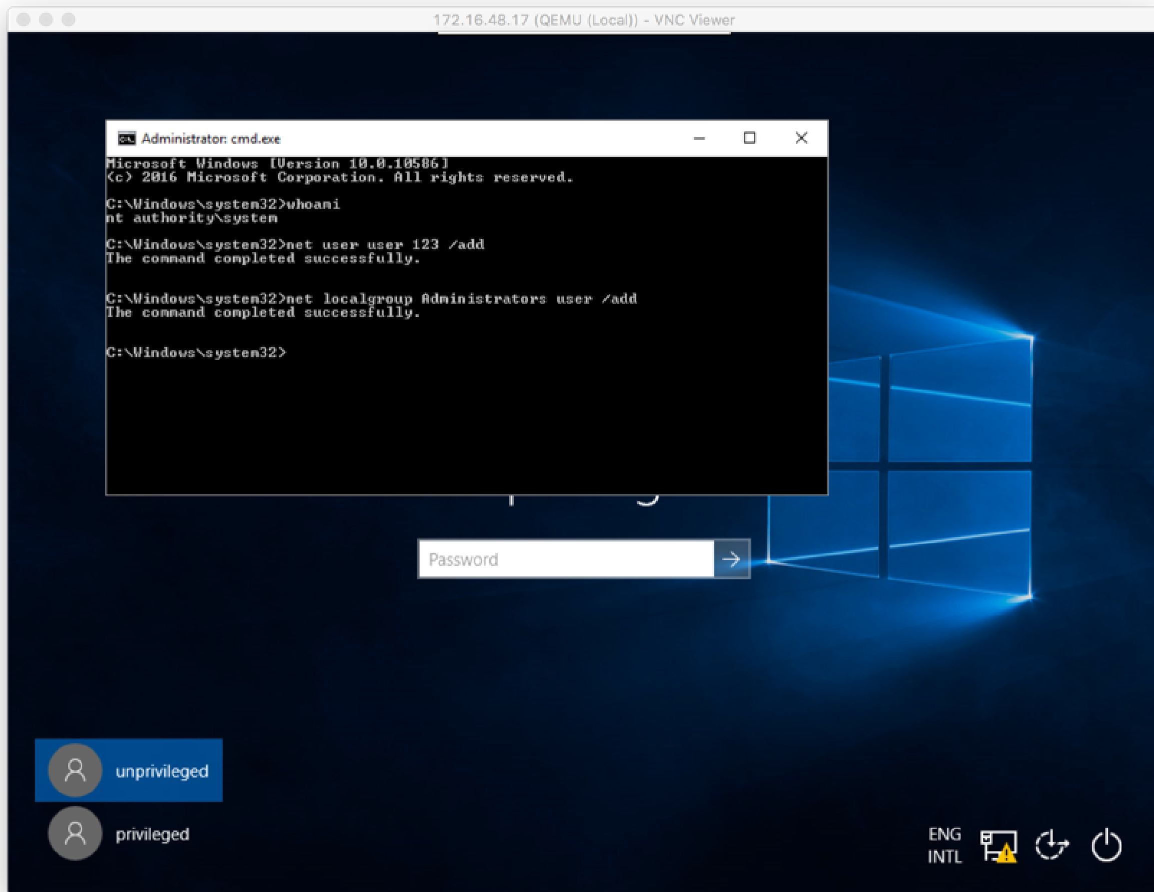


Fig. CMD.EXE started as “nt authority\system”

Notes:
More information about the Windows booting process can be found here:
https://securitybytes.io/blue-team-fundamentals-part-two-windows-processes-759fe15965e2
https://community.tribelab.com/mod/book/view.php?id=628&chapterid=217
http://carnal0wnage.attackresearch.com/2012/04/privilege-escalation-via-sticky-keys.html

Next, with system privileges, you may add users, change passwords of existing users and make users part of Administrators group.

Once you are able to login into the system, you can scan the filesystem with your tools to detect IOCs. One popular tool for quick pattern matching is Yara.

Writing Yara rule from IOC

https://t.me/learningnets

The IOC we have in this case is digital certificate serial number:
"**68:21:19:09:85:a8:75:54:3e:49:38:d4:5c:ea:9d:cb**".

1. Create a rule to match this condition and scan files on drive C: or at least system directory (“**C:\Windows\System32**”).

Notes:
To identify malware with digital signatures, the following yara rule can be crafted:

```
import "pe"
rule malware_signature {
meta:
  description = "Matches compromised digital certificate serial"
condition:
  pe.signatures[0].serial ==
"68:21:19:09:85:a8:75:54:3e:49:38:d4:5c:ea:9d:cb"
}
```

2. Use Yara to identify suspicious executable.
3. Analyse properties of the file and confirm certificate serial number.
4. Copy the file from the infected system to Bitscout for further analysis.

Hint: Try creating a new file, folder or registry key named after the suspicious executable. Can you explain the system's behavior?

Exporting Decrypted Partition

Considering discovery at the previous stage, access to the file system through the kernel of the infected system can’t be trusted. In such cases you need to use a clean system and drivers to analyse disk partitions. To do that, we can use one of the techniques discussed previously: network block device (NBD). There is a version of NBD server for Windows, which can help exposing the whole disk in raw over the network.

Copy the NBDServer tool to Windows, and run as follows.

```
> NBDServer.exe -p 2002 -c 10.3.0.1 -f \\.\C: -d -z
```

This makes the NBD server listen on TCP port 2002 and authorise connections coming from client 10.3.0.1 (our Qemu host/Bitscout container) and export device with system filepath "\\.\C:". A couple of extra options "-d -z" are used to enable verbose mode and use 0-bytes where the disk sectors cannot be read.

Notes:

https://t.me/learningnets

A customised version of NBDServer for windows
https://github.com/vitaly-kamluk/NBDServer
A copy was provided in the training toolkit.

NBDServer.exe v3.0

```
-c     Client IP address to accept connections from
-p     Port to listen on (60000 by default)
-f     File to serve ( \\.\PHYSICALDRIVE0 or \\.\pmem for example)
-n     Partition on disk to serve (0 if not specified)
-w     Enable writing (disabled by default)
-d     Enable debug messages
-q     Be Quiet..no messages
-h     This help text
-z     No read error (return zeroes instead)

-f option supports names like the following:
  \\.\PHYSICALDRIVE0    raw drive partition access along with -n option
  \\.\C:                volume access (no need for -n option)
  \\.\HarddiskVolume1   volume access (no need for -n option)
  afile.dd              serve up the contents of 'afile.dd' via nbd.
  \\.\pmem              access the pmem memory driver from volatility
```


https://t.me/learningnets

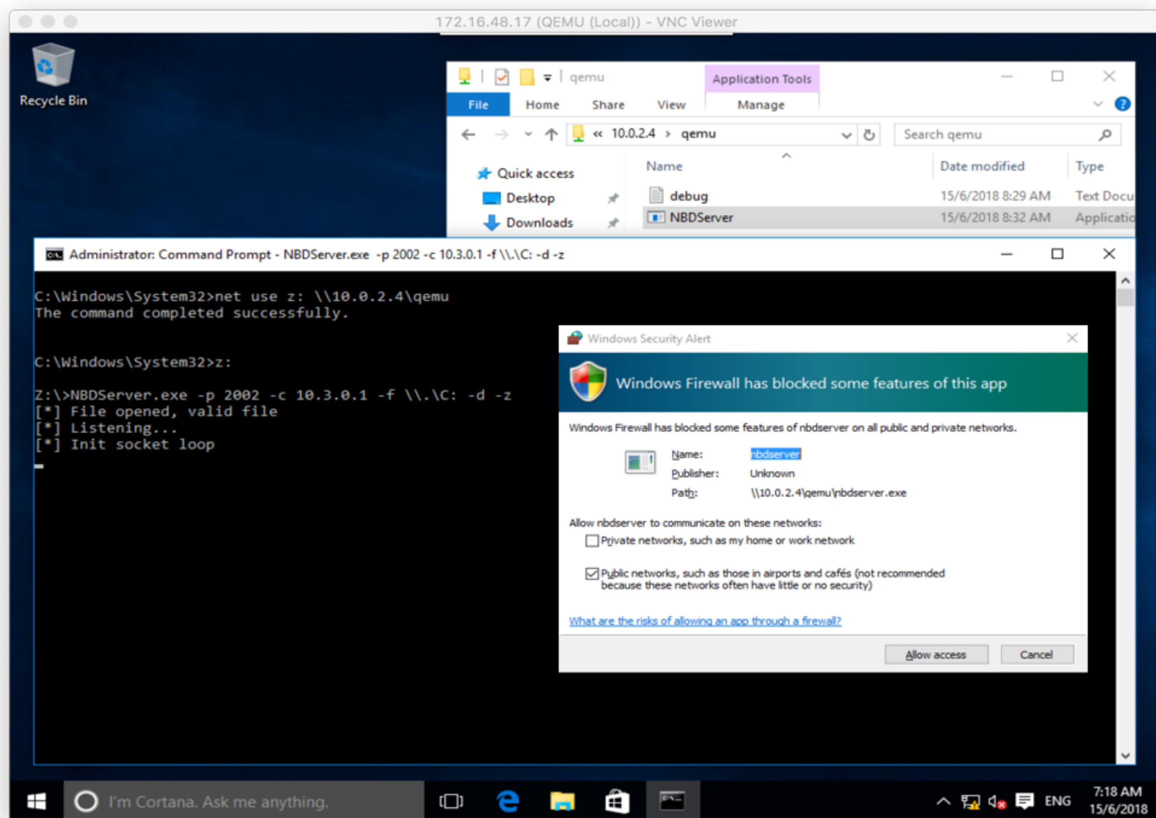


Fig. Starting NBDServer

You will need to use the xnbd-client tool from the expert container to connect to the NBDServer. However, current VM network settings restrict passing packets to/from external hosts (restrict=y option in qemu command line). This is an important option and is used to prevent infected systems from contacting any hosts on the network. To be able to exchange some packets with the infected system, we will forward just one port for NBDServer (port 2002). A simple way to do that in userspace is to use the socat tool, similarly to what was used in ShadowHammer backdoor exercise. Simply start socat that forwards external connections to a localhost interface to let Qemu treat such data flow as safe:

```
$ socat tcp-listen:2003,reuseaddr,fork tcp-connect:127.0.0.1:2002
```

This way socat will listen for incoming connections on TCP port 2003 and will forward them to localhost TCP port 2002, where Qemu forwards them further inside the virtual network and to the TCP port 2002 of Windows guest OS.

Note that operations with block devices, such as those maintained by NBD kernel modules in Bitscout are considered privileged and inaccessible to regular users.

https://t.me/learningnets

This is where you may need assistance from the owner by authorising execution of a couple of privileged commands via **supervised-shell**:

```
root@bitscout:~$ supervised-shell
supervised> xnbd-client --connect /dev/container/nbd0 10.3.0.2 2003 >/dev/null 2>/dev/null &
Your command is being reviewed. Once review is complete, you shall see output here.
[Ctrl+C] to exit
```

Note that we invoke xnbd-client with "&" in the end to put the process into background.

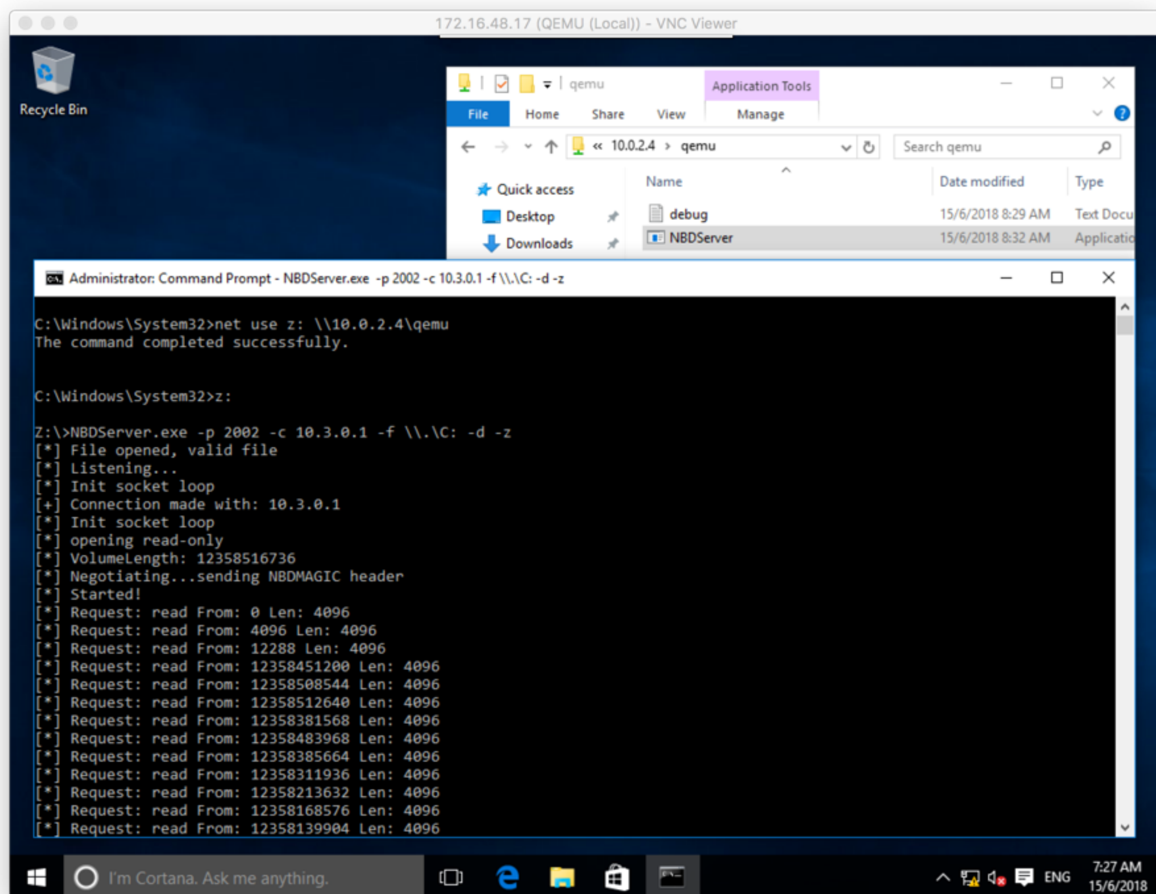


Fig. NBDServer receiving “read” instructions

If you did everything right, you should see a decrypted raw NTFS partition in **/dev/host/nbd0**.

8. Use the “**fls**” command on the device to confirm that it indeed contains the C: drive of the subject system.
9. Next, mount the remote disk on Bitscout. The encrypted disk appears decrypted as it is on Windows:

https://t.me/learningnets

```
$ mkdir /mnt/C
$ mount.ntfs-3g -o ro,show_sys_files,streams_interface=windows,norecover
/dev/host/nbd0 /mnt/C
```

10. Verify that the suspicious file identified earlier is visible and can be copied for further analysis.

It should look like the following session:

```
root@bitscout:~$ fls /dev/host/nbd0
r/r 84267-128-3:        $dcsys$
r/r 19313-128-3:        bootmgr
r/r 4-128-4:    $AttrDef
r/r 8-128-2:    $BadClus
r/r 8-128-1:    $BadClus:$Bad
r/r 6-128-4:    $Bitmap
r/r 7-128-1:    $Boot
r/- * 0:        $dcsys$
d/d 11-144-4:   $Extend
r/r 0-128-6:    $MFT
d/d 57-144-5:   $Recycle.Bin
r/r 19309-128-1:        BOOTNXT
r/r 85756-128-1:        hiberfil.sys
r/r 82752-128-1:        pagefile.sys
…

root@bitscout:~$ mkdir /mnt/C
root@bitscout:~$ mount -t ntfs-3g -o
ro,show_sys_files,streams_interface=windows,norecover /dev/host/nbd0 /mnt/C
```

After completing this exercise you should be able to acquire full drive C: disk image in decrypted form, removing proprietary disk encryption layer.

Chapter 17. Dealing With Broken System

Subject VM: win10_64bit_customfde_infected

Extracting data on the live running Windows system is a nice quick way for malware analysis, however strict forensic procedures may not accept such an approach. There could also be malware that detonates upon boot and destroys the system making it impossible even to see standard Windows logon. In this case, you shall find a way to access disk image in decrypted form before Windows even starts booting and introduces disk changes.

MBR FDE Boot Process

https://t.me/learningnets

We are reusing the same setup as in the previous exercise, but let's create a situation when malware detonates on boot and doesn't let you start Windows. For the purpose of the training, such malware was preloaded on the same Windows OS VM to start during system boot. You just need to activate it via registry settings. To do that, follow the instruction below:

1. Login as privileged user and navigate to the folder "**C:\Work**"
2. Install the registry key in "**perkiller2.reg**" by double-clicking the file.
3. Reboot the system.

If the installation works well, you should be presented with a "Blue Screen of Death" (BSOD) as seen in the image seen below.

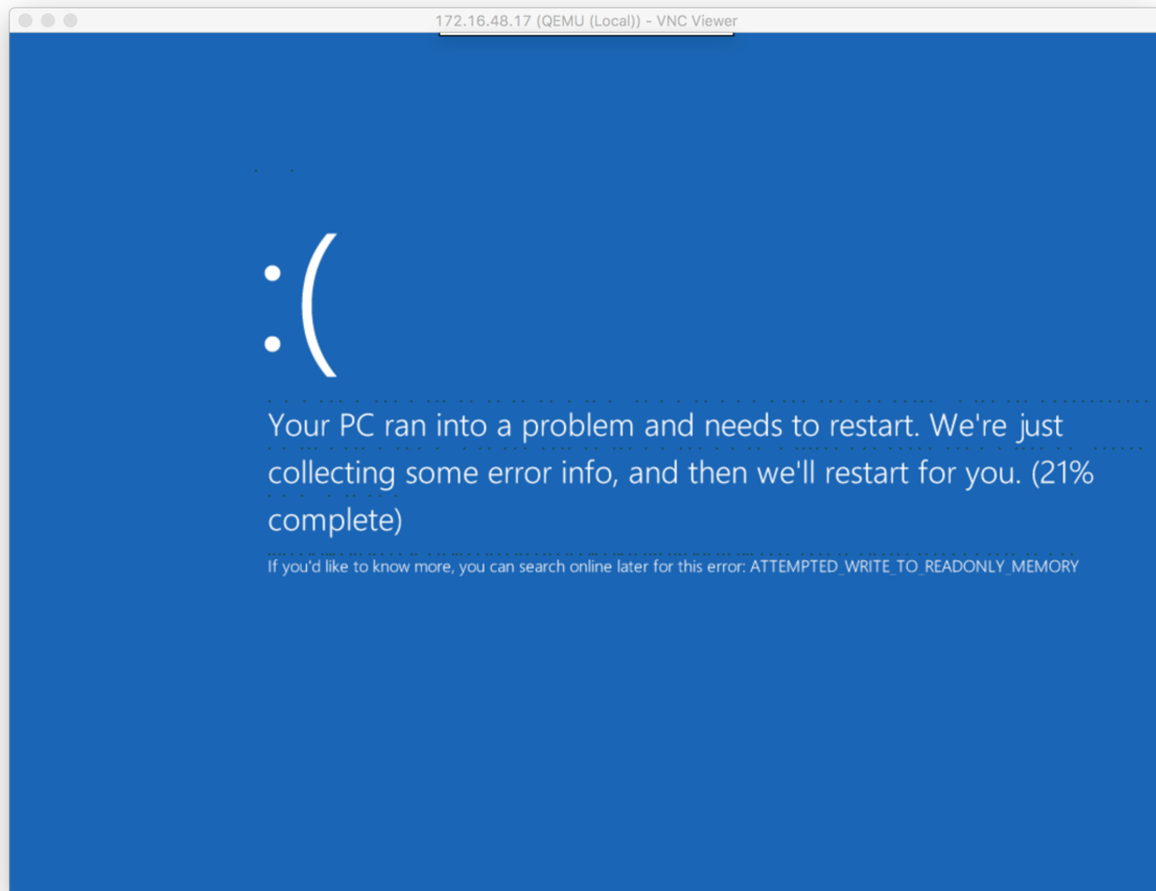


Fig. Windows BSOD during boot (malware effect).

Since the system is unusable, none of the tricks used before would work. The crash happens before Windows can even switch to graphical mode. Remember the initial assumptions about the OS relying on a disk that would appear decrypted? Next, you will have to work in the early stages of the boot, when the disk decryption driver is loaded but before the OS starts.

When the machine is powered on, the system performs POST (Power-On Self-Test) to check the hardware and initialize it. BIOS searches for the boot record in the first HDD sector and loads it into the RAM at a fixed location (0x7c00), then passes CPU execution there. In case of disk encryption for MBR-based setup, it will set an interrupt handler for int13 (disk i/o interrupt) to decrypt the disk on-demand once the password validation succeeds.

Notes:
More information about the early boot process can be found here:
https://neosmart.net/wiki/mbr-boot-process/
https://is.muni.cz/th/jih07/chromik-bootprocess.pdf

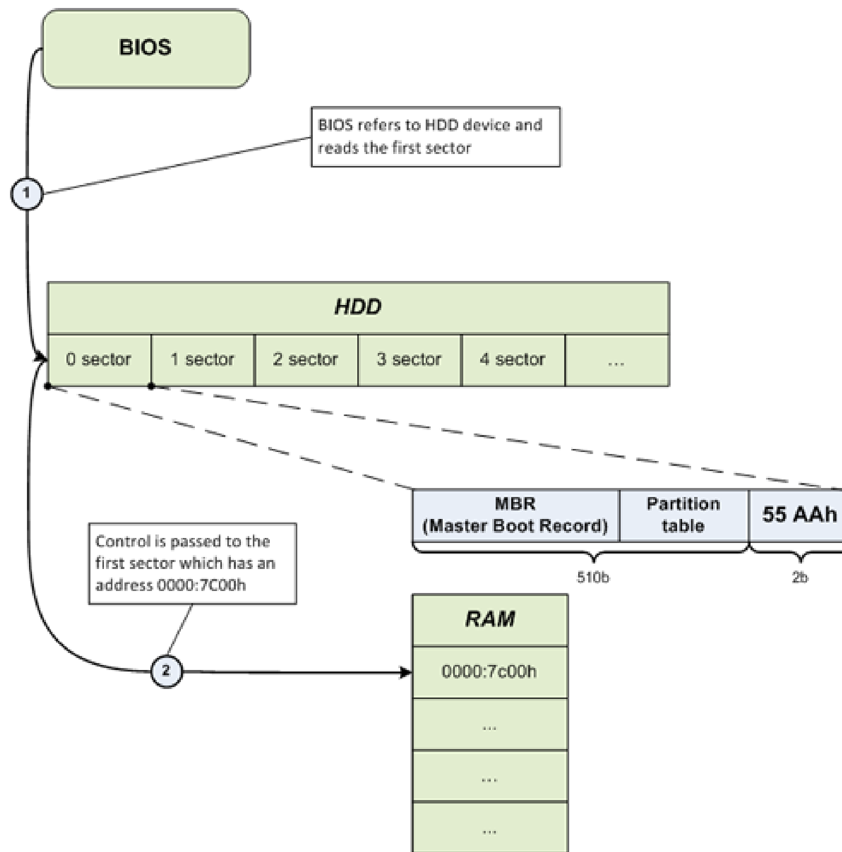


Fig. MBR boot process

https://t.me/learningnets

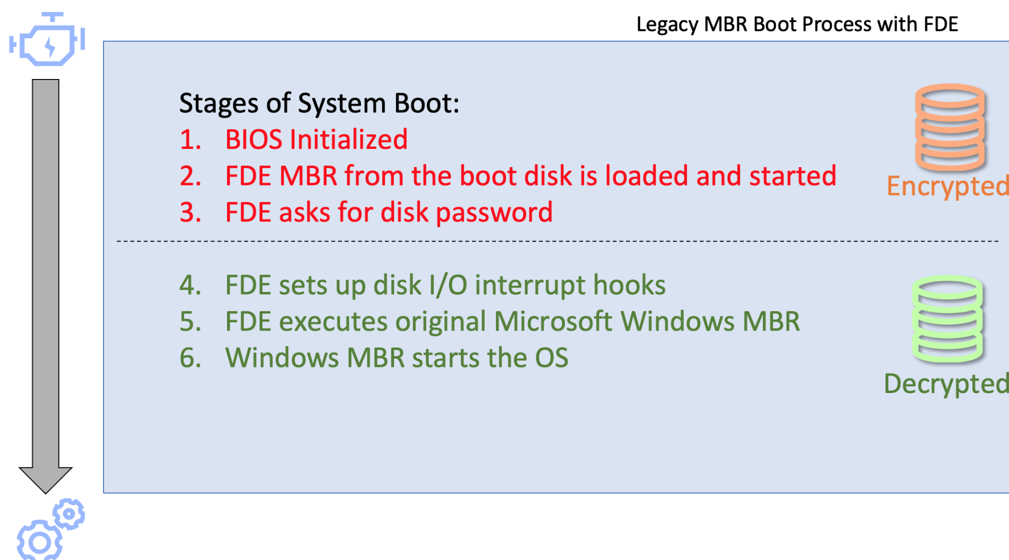


Fig. Generic MBR boot process with FDE

To intercept this sequence, you need to control every stage of the VM boot process starting from the poweron. Qemu provides option "-S" that suspends execution of the VM on start and waits for the user command to initiate the boot process.

The plan to access the drive is to inject some minimal code into the boot process. This code will communicate with the hypervisor via virtual serial interface (COM1) attached to one-directional pipes on the hypervisor. Serial port offers very slow connection speed and will be used only for synchronisation. The actual data transfer may be done via physical memory of the VM as it is readable and writable by both the hypervisor and the VM guest code.

You can start Qemu with the option to create a virtual serial port (COM1) that will be attached to the pipes on the host system. Two pipe files (guest.serial.in and guest.serial.out) need to be created beforehand with the "**mkfifo**" command as followed:

```
root@bitscout:~$ mkfifo guest.serial.{in,out}

root@bitscout:~$ ls -lh guest.serial.*
prw-r--r-- 1 root root 0 Apr  5 09:36 guest.serial.in
prw-r--r-- 1 root root 0 Apr  5 09:36 guest.serial.out
```

```
root@bitscout:~$ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m
512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide
-monitor stdio -s -vnc :0 -spice port=2001,disable-ticketing -chardev
pipe,path=guest.serial,id=com1 -serial chardev:com1 -S
```

```
QEMU 2.11.1 monitor - type 'help' for more information
(qemu)
```

The option “**-chardev**” connects the pipe files to the virtual COM1 serial interface.
To control execution stages, we will use a process debugger such as GDB. Qemu includes an embedded GDB server which can be started with “**-s**” option (Shorthand for -gdb tcp::1234 - start a gdbserver on TCP port 1234).
Once started, you can use the “**gbd**” command to start the debugger client and attach to server using option “**-ex 'target remote localhost:1234'**”.

To intercept MBR code a hardware breakpoint should be placed at **0x7c00** address of the VM. Once the breakpoint is set, resume execution of the VM by using the "c" (or "continue") command in the gdb prompt. You should see that the breakpoint is hit immediately.

Notes:
More information about breakpoints and MBR loading can be found here:
https://sourceware.org/gdb/wiki/Internals/Breakpoint%20Handling
https://en.wikibooks.org/wiki/X86_Assembly/Bootloaders
https://yangbolong.github.io/2017/02/12/lab1/
http://www.dewassoc.com/kbase/hard_drives/master_boot_record.htm
https://thestarman.pcministry.com/asm/mbr/index.html
https://thestarman.pcministry.com/asm/mbr/STDMBR.htm

At this time 0x7c00 contains the FDE code, which runs and asks the user to input the decryption password. Once the password is validated, the FDE code replaces the code at 0x7c00 with the original Microsoft Windows MBR code and restarts the boot process. This way Windows transparently accesses the decrypted drive through the FDE code running as a disk I/O interrupt handler.

Therefore when you continue execution by running "c" command in gdb prompt again, the second breakpoint is hit when the Windows MBR code is restored at 0x7c00.

In GDB, it would look like this:

```
root@bitscout:~$ gdb -ex 'target remote localhost:1234'
(gdb) hb *0x7c00
Hardware assisted breakpoint 1 at 0x7c00
(gdb) c  ← start running the VM until the MBR FDE code
Continuing.
Breakpoint 1, 0x00007c00 in ?? ()
(gdb) c  ← start the FDE module and verify the decryption password
Continuing.
```


https://t.me/learningnets

At that point, when accessing the VM screen over spice or VNC, you should see the FDE password prompt. Enter the password (i.e. "12345") and push [Enter].

In the terminal the breakpoint will be hit the second time.

```
Breakpoint 1, 0x00007c00 in ?? ()
(gbb) ← do not continue the execution for now!
```

The system is in the state when disk I/O operations are relayed through FDE code but Windows hasn't started booting yet (the FDE module transparently decrypts disk sectors to be read/written).

Accessing the disk with a code implant

The disk can be accessed through the FDE code but only from the inside of the VM. You now need to instruct the FDE code from the Bitscout console to read disk's sectors (in decrypted form) and export the data outside the VM.

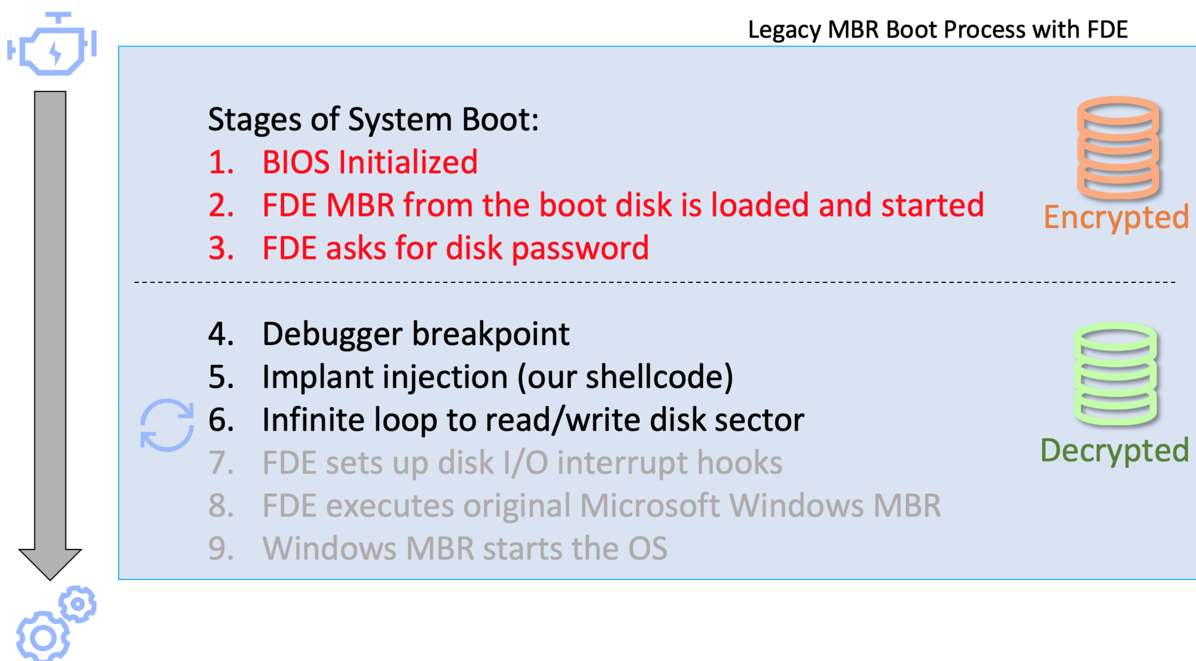


Fig. MBR boot process with FDE, interrupted to allow reading decrypted arbitrary sectors

https://t.me/learningnets

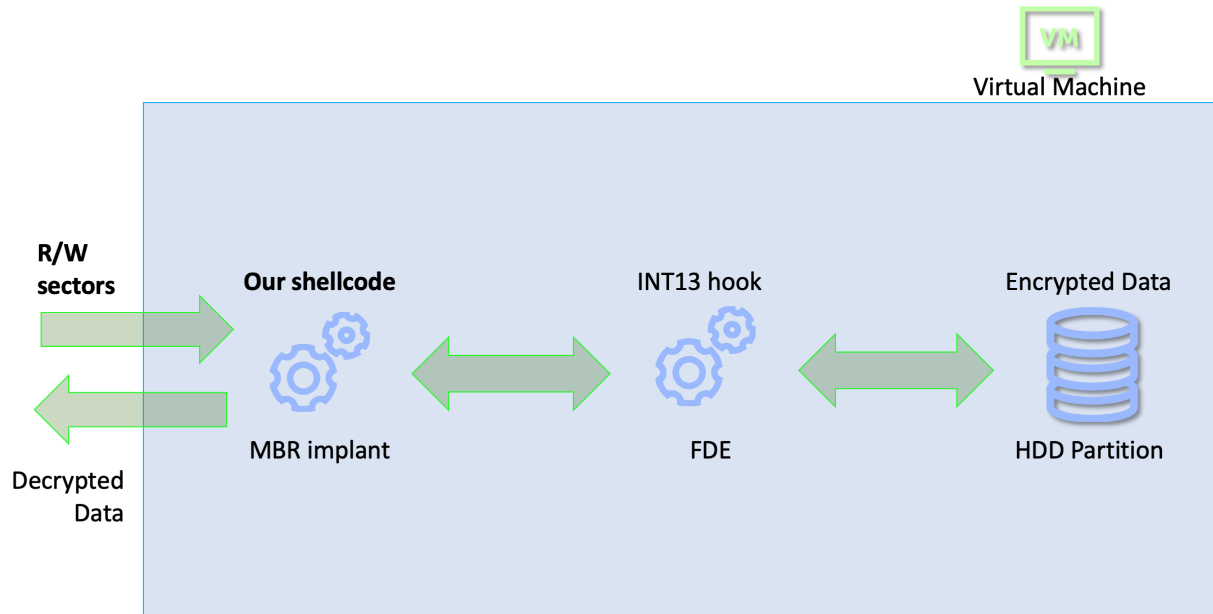


Fig. Visual representation of the query for decrypted sectors.

For sector data transfers we will use VM memory which is accessible in the hypervisor process (host system) and standard Linux file /proc/%PID%/mem. The mem file contains the whole memory of the process identified with %PID%. Somewhere inside this memory there is a large contiguous block of memory that is allocated by Qemu to be used as the physical RAM of the guest. All you need to do is to find a large block which is equal to the size of VM RAM (512MB). The mem file is accompanied by a **maps** file in the same directory. The maps file contains the description of the allocated contiguous memory blocks.

Example

```
//Find the qemu PID
root@bitscout:~$ pgrep qemu
703
//Show contents of maps file for the PID
root@bitscout:~$ cat /proc/703/maps
559d1bcb4000-559d1c586000 r-xp 00000000 00:3a 11712   /usr/bin/qemu-system-x86_64
559d1c786000-559d1c952000 r--p 008d2000 00:3a 11712   /usr/bin/qemu-system-x86_64
559d1c952000-559d1c9d1000 rw-p 00a9e000 00:3a 11712   /usr/bin/qemu-system-x86_64
559d1c9d1000-559d1ce34000 rw-p 00000000 00:00 0
559d1ddcb000-559d1f15e000 rw-p 00000000 00:00 0       [heap]
7f14fb8f9000-7f14fb8fa000 ---p 00000000 00:00 0
7f14fb8fa000-7f14fb9fa000 rw-p 00000000 00:00 0
7f14fb9fa000-7f14fb9fb000 ---p 00000000 00:00 0
7f14fb9fb000-7f14fbafb000 rw-p 00000000 00:00 0
```


https://t.me/learningnets

```
7f14fbafb000-7f14fbafc000 ---p 00000000 00:00 0
...
//Find the contiguous block of 512MB in size
root@bitscout:~$ cat /proc/703/maps | cut -d' ' -f1| awk -F -
'{if(strtonum("0x" $2)-strtonum("0x" $1)==0x20000000){printf("0x%x",
strtonum("0x" $1))}}'
0x7f1503e00000 ← the beginning of VM RAM
//0x7f1503e07c00 ← the location of 0x7c00 inside the VM
```

Next we will use a simple python script (called **set_procmem.py**), that writes binary data (input as hexadecimal string) to the specified file at the specified offset.

```
root@bitscout:~$ vi set_procmem.py
#! /usr/bin/env python
import re,sys
from binascii import unhexlify
if len(sys.argv) == 1:
  print("%s <pid> <start> <hex data>" % (sys.argv[0]))
  exit(0)
PID=int(sys.argv[1],10)
mem_file= open("/proc/%d/mem" % PID, 'w', 0)
start = int(sys.argv[2], 16)
mem_file.seek(start)  # seek to region start
mem_file.write(unhexlify(sys.argv[3]))
mem_file.close()

root@bitscout:~$ ./set_procmem.py 703 0x7f1503e07c00 0102030405060708090a0b
//You may verify that /proc/703/mem contains 01 02 03 04 05 06 07 08 09 0a
0b bytes at the offset 0x7f1503e07c00 now with the hexedit tool.

//Now, you can combine commands above into one-liner and paste the real
code of the implant.
root@bitscout:~$ ./set_procmem.py`pgrep qemu` $(cat /proc/`pgrep qemu`/maps
| cut -d' ' -f1| awk-F- '{if(strtonum("0x" $2)-strtonum("0x"
$1)==0x20000000){printf("0x%x", strtonum("0x" $1)+0x7c00)}}')
eb2d10000100007e0000002803000000000be007c81c60200b442b280cd13bafd03eca8207
4f8baf803b02deeeb3ebc007c31c08ec06a001fbaf903b000eebafb03b080eebaf803b003ee
baf903b000eebafb03b003eebafa03b0c7eebafc03b00beeb486b90000ba0010cd15bafd03e
ca80174eebaf803eca82e7595ebee
```


https://t.me/learningnets

set_procmem.py replaces the code at 0x7c00 by our 254 bytes implant. To find the right offset, we retrieve the offset of the memory region with the size of the RAM allocated tour VM (512mb = 0x20000000) and add 0x7c00.
/proc/%pid%/maps contains the offset and the size of all regions allocated,
/proc/%pid%/mem the memory content of that process

```
root@bitscout:~$ vi set_procmem.py
#! /usr/bin/env python
import re,sys
from binascii import unhexlify
if len(sys.argv) == 1:
  print("%s <pid> <start> <hex data>" % (sys.argv[0]))
  exit(0)
PID=int(sys.argv[1],10)
mem_file= open("/proc/%d/mem" % PID, 'w', 0)
start = int(sys.argv[2], 16)
mem_file.seek(start)  # seek to region start
mem_file.write(unhexlify(sys.argv[3]))
mem_file.close()

root@bitscout:~$ ./set_procmem.py `pgrep qemu` $(cat /proc/`pgrep
qemu`/maps | cut -d' ' -f1| awk -F- '{if(strtonum("0x" $2)-strtonum("0x"
$1)==0x20000000){printf("0x%x", strtonum("0x" $1)+0x7c00)}}')
eb2d10000100007e0000002803000000000be007c81c60200b442b280cd13bafd03eca8207
4f8baf803b02deeeb3ebc007c31c08ec06a001fbaf903b000eebafb03b080eebaf803b003ee
baf903b000eebafb03b003eebafa03b0c7eebafc03b00beeb486b90000ba0010cd15bafd03e
ca80174eebaf803eca82e7595ebee
```

The hexadecimal code of the implant that you see above was produced as follows.

The source code of the implant

Here below is the code that awaits for signals sent over the pipes (character "." for reading), reads the sector, copy the content of the sector in clear at the location 0x32800 in memory, then sends the signal back (character "-") to indicate completion, then loop again indefinitely.

```
USE16

  jmp     START
```


https://t.me/learningnets

```
DAP:
  db      0x10       ; Packet Size
  db      0          ; Always 0
blkcnt:
  dw      0x0001     ; Sectors Count
db_add:
  dw      0x0000     ; Transfer Offset
  dw      0x56E0     ; Transfer Segment
d_lba :
  dd      0x32800    ; Low offset value     | offset 0x0a
  dd      0          ; High offset value    | offset 0x0e

SECTORREAD:
  mov     si, 0x7c00   ; SI points to the base address
  add     si, DAP      ; load DAP structure offset to SI
  mov     ah, 0x42     ; function 42h (extended disk read)
  mov     dl, 0x80     ; drive ID (0x80 means the first drive)
  int     0x13         ; call the interrupt
  jmp     WRITESERIAL

SECTORWRITE:
  mov     si, 0x7c00   ; SI points to the base address
  add     si, DAP      ; load DAP structure offset to SI
  mov     ah, 0x43     ; function 42h (extended disk read)
  xor     al, al       ; set write verify to 'off'
  mov     dl, 0x80     ; drive ID (0x80 means the first drive)
  int     0x13         ; call the interrupt

WRITESERIAL:
  mov     dx, 0x3fd
  in      al, dx
  test    al,0x20
  jz      WRITESERIAL

  mov     dx,0x3f8
  mov     al, '-'
  out     dx, al

  jmp     RECVSERIAL    ; return to waiting for signal over serial

START:
  mov     sp, 0x7c00   ; Setup stack
  xor ax, ax
  mov es, ax
  push    0
  pop     ds
```


https://t.me/learningnets

```
INITSERIAL:
  mov    dx, 0x3f9 ; COM1 port is 0x3f8
  mov    al, 0x00
  out    dx, al ; Disable all interrupts

  mov    dx, 0x3fb
  mov    al, 0x80
  out    dx, al ; Enable DLAB (set baud rate divisor)

  mov    dx, 0x3f8
  mov    al, 0x03
  out    dx, al ; Set divisor to 3 (lo byte) 38400 baud

  mov    dx, 0x3f9
  mov    al, 0x00
  out    dx, al ; divisor high byte

  mov    dx, 0x3fb
  mov    al, 0x03
  out    dx, al ; 8 bits, no parity, one stop bit

  mov    dx, 0x3fa
  mov    al, 0xc7
  out    dx, al   ; Enable FIFO, clear them, with 14-byte threshold

  mov    dx, 0x3fc
  mov    al, 0x0b
  out    dx, al   ; IRQs enabled, RTS/DSR set

RECVDELAY:
  mov    ah, 86h  ; sleep function
  mov    cx,0h    ; high word of sleep interval (1,000,000ths of a second)
  mov    dx,1000h ; low word of sleep interval
  int    15h

RECVSERIAL:
  mov    dx, 0x3fd
  mov    al, 0
  in     al, dx
  test   al, 1
  jz     RECVDELAY
  mov    dx, 0x3f8
  in     al,dx     ; get 1 byte from COM1
  cmp    al, 0x2e  ; received '.' - read sector operation
  je     SECTORREAD
  cmp    al, 0x3a  ; received ':' - write sector operation
  je     SECTORWRITE
```


https://t.me/learningnets

```
jmp     RECVSERIAL
```

To generate the hexadecimal form of the code, first you need to assemble it into a binary using the "**nasm**" command.
$ nasm implant.asm

Next dump the hex values with xxd:

```
$ xxd -pos -c200 implant
eb2d10000100007e0000002803000000000be007c81c60200b442b280cd13bafd03eca8207
4f8baf803b02deeeb3ebc007c31c08ec06a001fbaf903b000eebafb03b080eebaf803b003ee
baf903b000eebafb03b003eebafa03b0c7eebafc03b00beeb486b90000ba0010cd15bafd03e
ca80174eebaf803eca82e7595ebee
```

Now you may resume the VM execution and it will enter the infinite loop of processing disk read/write requests from the hypervisor.

```
(qemu) del 1    ; remove the breakpoint
(qemu) c  ; continue
```

Now you may try the read_sector.py script that interacts with the implant in the given VM specified by process ID of Qemu, in/out pipe names (specified with prefix) and the offset of the disk sector to be read.

```
root@bitscout:~$ ./read_sector.py
./read_sector.py <pid> <control pipe> <disk hex offset>
root@bitscout:~$ ./read_sector.py 703 guest.serial 0xfa800 | xxd -r -pos |
hexdump -C
00000000  eb 52 90 4e 54 46 53 20  20 20 20 00 02 08 00 00  |.R.NTFS    .....|
00000010  00 00 00 00 00 f8 00 00  3f 00 ff 00 00 a8 0f 00  |........?.......|
00000020  00 00 00 00 80 00 80 00  ff 4f 70 01 00 00 00 00  |.........Op.....|
00000030  00 00 0c 00 00 00 00 00  02 00 00 00 00 00 00 00  |................|
00000040  f6 00 00 00 01 00 00 00  72 68 6a a8 a0 6a a8 2c  |........rhj..j.,|
00000050  00 00 00 00 fa 33 c0 8e  d0 bc 00 7c fb 68 c0 07  |.....3.....|.h..|
00000060  1f 1e 68 66 00 cb 88 16  0e 00 66 81 3e 03 00 4e  |..hf......f.>..N|
```

0xfa800 is the offset to the second (encrypted NTFS) partition of the system in sectors. To find this number you should use mmls tool and list partition properties of the disk drive and identify the start of the subject encrypted partition. Then convert it to hexadecimal.

https://t.me/learningnets

```
root@bitscout:~$ mmls /dev/host/evidence0
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0001026047   0001024000   NTFS / exFAT (0x07)
003:  000:001   0001026048   0025163775   0024137728   NTFS / exFAT (0x07)
004:  -------   0025163776   0025165823   0000002048   Unallocated
```

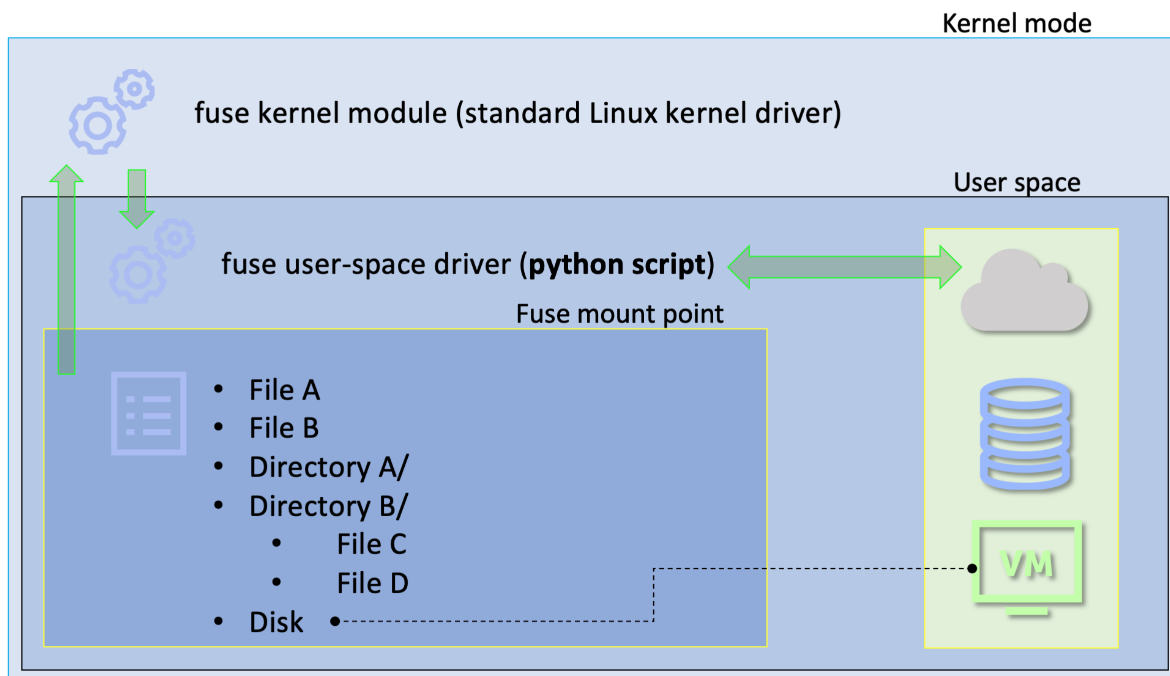
The offset 0xfa800 is the hexadecimal representation of 1026048.

If this works and you are able to read the disk sector by sector, you may combine this into a smooth read/write process done automatically on-demand by a FUSE driver.

Note:
More info about FUSE can be found here:
https://www.stavros.io/posts/python-fuse-filesystem/
https://github.com/skorokithakis/python-fuse-sample

The training uses a variant of such FUSE driver written in Python and provided as qemu2fuse_relay.py.

https://t.me/learningnets



The driver accepts parameters similar to the read_sector.py script, with a couple extra parameters for the size of the subject disk and the destination mount point (a directory where the virtual file will be created by FUSE). Start this script as follows (example).

```
root@bitscout:~$ mkdir /root/fuse
root@bitscout:~$ blockdev --getsize64 /dev/host/evidence0
12884901888
root@bitscout:~$ ./qemu2fuse_relay.py 703 guest.serial 12884901888
/root/fuse/

//In another terminal
root@bitscout:~$ ls -la ./fuse/
total 0
drwxr-xr-x 2 root root        60 Jan  1  1970 .
drwx------ 1 root root       260 Apr  6 12:33 ..
-rw-r--r-- 1 root root 12884901888 Jan  1  1970 disk
```

Note the "disk" file in the fuse directory. It should contain the whole HDD mapped from /dev/host/evidence0 in decrypted form. It can be used now to mount the filesystem on encrypted drive C:, acquire full disk image for further analysis and so on. This concludes the chapter and proves that the disk image can be acquired even on broken FDE setups where the OS cannot

https://t.me/learningnets

start. It can be treated as an untouched disk image, because the OS hasn't started booting and it introduces no additional changes coming from the process of system startup.

Chapter 18. Dealing with UEFI FDE

Subject VM: win10_64bit_customfde_uefi

Other than MBR based FDE, another popular setup relies on EFI-drivers for disk encryption. Considering that EFI has already become a standard for new laptops, desktops and servers, it's important to look at possible solutions for UEFI FDE.

UEFI FDE Boot Process

UEFI Boot:
1. SEC (Security Phase)
2. PEI (Pre EFI Initialisation Phase)
3. DXE (Driver Execution Env.)
4. EFI runtime UP
5. Boot Device Selection
6. EFI Shell ← You will have to interrupt the boot process to interact with the UEFI Shell and load a custom NBDServer driver and expose the file system (decrypted on-demand)
7. OS Bootloader

Qemu supports booting UEFI based systems, however it requires UEFI firmware file and NVRAM (variables storage) to be specified as arguments. The custom NbdSrv.efi DXE module was precompiled and provided in binary form together with the source code of the project.

UEFI NBD Driver

Bitscout supports booting on both UEFI and MBR systems and should start on UEFI systems as usual.

1. The owner of the system shall map the target disk (i.e. /dev/sda) to evidence0 device.
2. Apply copy-on-write protection:
   $ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2
3. Transfer OVMF_CODE.fd and VARS file to /root of Bitscout
4. Copy NbdSrv.efi to /root/efi directory
5. Start the VM using the command as follows
   $ qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive file=/root/evidence0.qcow2,format=qcow2,if=ide,id=drive-virtio-disk0 -monitor stdio -spice port=2001,disable-ticketing -vga cirrus -device e1000,netdev=net0 -netdev user,net=10.0.2.3/24,id=net0,restrict=y,hostfwd=::2001-:2001 -s **-drive**

https://t.me/learningnets

file=/root/OVMF_CODE.fd,if=pflash,format=raw,unit=0,readonly=on -drive file=/root/VARS,if=pflash,format=raw,unit=1 -hdb fat:rw:efi

6. When the VM boots you should press <F2> to enter the UEFI setup tool. If you missed the moment, reboot with Ctrl-Alt-Del and try again.
7. Use the menu to enter interactive EFI Shell.

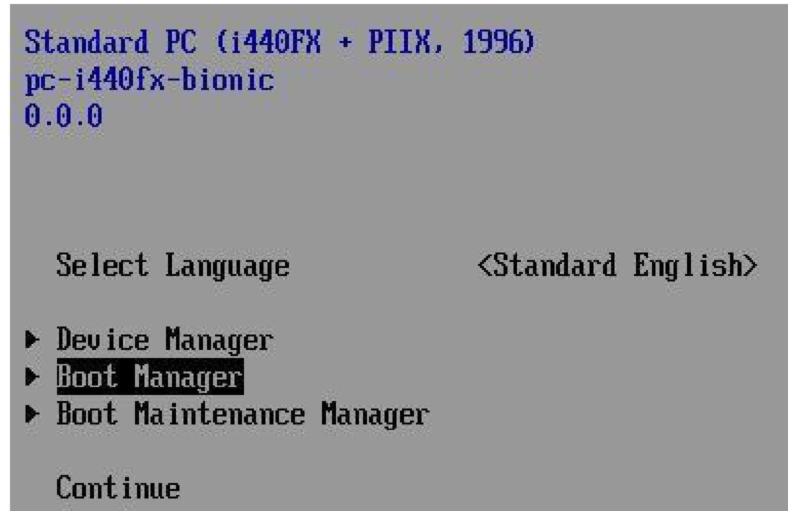


Fig. Tianocore EFI Setup Utility.

Once in the EFI shell ,you nede to change the drive to system EFI partition and load the FDE driver with "load" command.

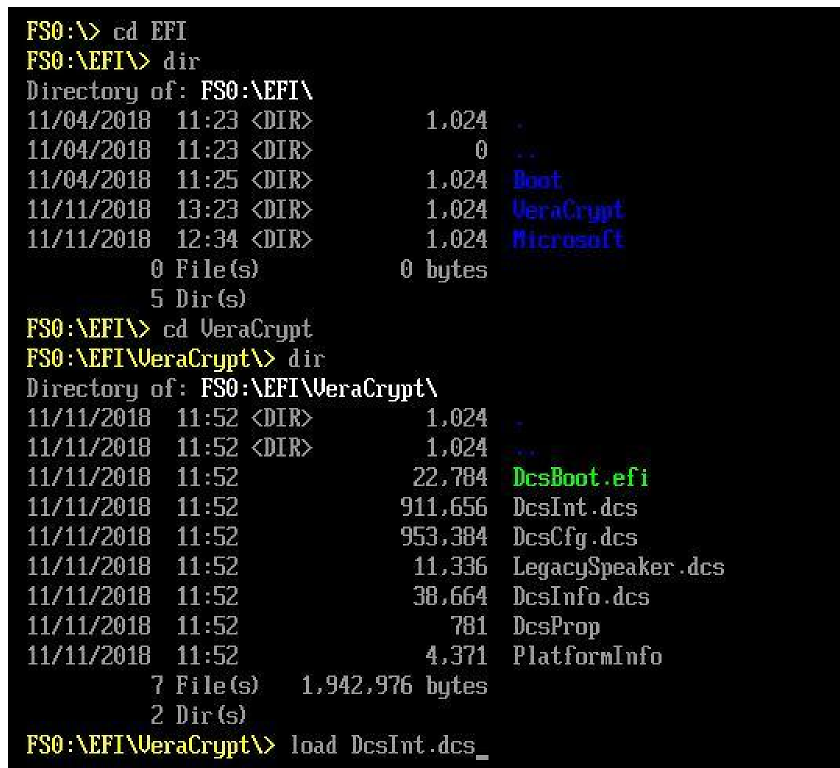


Fig. Loading the VeraCrypt FDE driver in EFI shell.

https://t.me/learningnets

The driver will interactively ask to enter the decryption password and will setup disk I/O access routed through proprietary DXE driver.

After that you need to start the NBD server in the same EFI shell and connect to it over the network from Bitscout.

Change drive to "FS1:" (virtual drive from the mapped directory) where you have NbdSrv.efi
EFI> FS1:

Start NBDServer.efi (if network hasn't loaded you may need to do it twice)
FS1:> NbdSrv.efi %TARGET_BLK_DEVICE%

Next start socat to relay connections from host to Qemu locally:
$ socat -v tcp-listen:2003,reuseaddr,fork tcp-connect:2002

Finally start xnbd-client via supervised shell:
supervised> xnbd-client --blocksize 512 --connect /dev/container/nbd0 10.3.0.2 2002 >/dev/null 2>/dev/null &

If all the commands were executed correctly you should find decrypted C: partition data on /dev/host/nbd0 device. You may mount and inspect device accordingly or acquire full disk image.

Chapter 19. Analysing macOS

It may happen that the compromised system is running macOS. Bitscout can help analyse such setups with static and even dynamic VM-based approaches.

macOS High Sierra (10.13) introduced a new proprietary file system, Apple File System (APFS). In addition, since macOS Panther (10.3), the disks might be protected by FileVault disk encryption.

> **Note:**
> More information about APFS can be found on this video:
> https://www.youtube.com/watch?v=OX5H-RsKexI

To be able to mount an APFS, you may need to download and compile additional software before or even during the analysis session on Bitscout.

> **Note:**

https://t.me/learningnets

Instructions to install APFS Fuse drivers (user-space) can be found on the github project page:
https://github.com/sgan81/apfs-fuse

The following commands will help building the apfs-fuse driver on Bitscout:

```
root@bitscout $ apt update
root@bitscout $ apt install libattr1-dev libicu-dev libbz2-dev libfuse-dev
git cmake
root@bitscout $ git clone https://github.com/sgan81/apfs-fuse
root@bitscout $ cd apfs-fuse
root@bitscout $ mkdir build
root@bitscout $ git submodule init
root@bitscout $ git submodule update
root@bitscout $ cd build
root@bitscout $ cmake ..
root@bitscout $ make
```

Once the drivers are compiled, create an empty folder “**/mnt/evidence0/**” then you can finally mount the Filevault encrypted disk:

```
root@bitscout $ mkdir /mnt/evidence0
root@bitscout $ apfs-fuse -o ro /dev/host/evidence0  /mnt/evidence0
```

Look for malicious activity and particularly some form of persistence.

Note:
More information about persistence on macOS can be found here:
https://www.gosecure.net/blog-archived/persistence-mac-os
https://null-byte.wonderhowto.com/how-to/hacking-macos-install-persistent-empire-backdoor-macbook-0184820/

In addition to mounting the disk to analyse the content, you can create a VM, similar to previous exercises and boot it.

macOS systems require a different set of options to cater for the boot loader and the drivers. The following command lines will create a new VM and start it:

```
root@bitscout $ qemu-img create -f qcow2 -o
backing_file=/dev/host/evidence0,backing_fmt=raw /root/evidence0.qcow2

root@bitscout$ qemu-system-x86_64 -enable-kvm -m 1024 -cpu
Penryn,kvm=on,vendor=GenuineIntel,+invtsc,vmware-cpuid-freq=on,+x2apic,+vme
,+hypervisor,+aes -machine pc-q35-2.9 -smp 4,cores=2 -usb -device usb-kbd
-device usb-tablet -device
isa-applesmc,osk="ourhardworkbythesewordsguardedpleasedontsteal(c)AppleComp
uterInc" -drive if=pflash,format=raw,readonly,file=OVMF_CODE.fd -drive
if=pflash,format=raw,file=OVMF_VARS.fd -smbios type=2 -device
ich9-intel-hda -device hda-duplex -device ide-drive,bus=ide.2,drive=Clover
-drive
id=Clover,if=none,snapshot=on,format=qcow2,file=./Clover-1280x960.qcow2
-device ide-drive,bus=ide.1,drive=MacHDD -drive
id=MacHDD,if=none,file=./evidence0.qcow2,format=qcow2 -netdev
tap,id=net0,script=no,downscript=no,ifname=tap0 -device
e1000-82545em,netdev=net0,id=net0,mac=52:54:00:c9:18:27 -monitor stdio -vga
std -spice port=2001,disable-ticketing -vnc :0
```

Note:
More information about macOS and VM creation can be found here:
https://gist.github.com/StefanoBelli/719105c97c7efe11907e3bfd1e1917ff
https://github.com/kholia/OSX-KVM

Unfortunately, a popular setup with drivers from the OSK-KVM project is not working correctly with the encrypted filesystem. You need to add extra drivers. At the boot screen with the various options, choose the Clover EFI shell and from the shell, load the drivers before resuming the boot process of the macOS. Instructions are as follows.

Boot on Clover EFI shell

```
FS5:
cd EFI\CLOVER\drivers64UEFI\
Load *
cd ..
CLOVERX64.efi
```

Now boot normally by choosing "**Boot Filevault Prebooter from Preboot**" and enter the password "**123**".

To perform further investigation, in addition to the filesystem analysis, you can use extra tools such as Monitor.app (from FireEye).

TLP:Red Sharing of this document without permission of the authors is prohibited.

https://t.me/learningnets

Note:
Monitor.app can be downloaded from:
https://www.fireeye.com/services/freeware/monitor.html

To exchange data between the Bitscout container and the macOS VM you may use Samba shared folders.

1. Copy the tools from your machine to the container such as over scp:

```
expert $ scp -r -i ~/.ssh/scout /tmp/sdl-monitor.zip root@192.168.1.153:
```

2. Configure the static IP on macOS and tap0 of Bitscout
3. Start SMB server on Bitscoutusing the following commands:

```
root@bitscout$ mkdir /root/smb
```

```
root@bitscout$ vi /tmp/smb.conf
[global]
security = user
map to guest = Bad User
[qemu]
path=/root/smb
read only=no
guest ok=yes
force user=root

root@bitscout$ smbd -F -S -s /tmp/smb.conf
```

With Monitor.app, you can now monitor the activity on the system similar to the way you did it earlier with sysmon from the Sysinternals tools.

Some further hints below to intercept outgoing TCP connection to malicious server:

1. Start dnsmasq to reply to DNS requests

```
root@bitscout$ vi /tmp/dnsmasq.conf
address=/#/192.168.95.10
address=/www.celasllc.com/192.168.95.1
log-queries
```


https://t.me/learningnets

```
log-facility=/tmp/dns.log

root@bitscout$ dnsmasq --keep-in-foreground -C /tmp/dnsmasq.conf
```

2. Start listening on port 443 to catch incoming connection:

```
root@bitscout$ nc -l -p 443
```

Notes:

Another way to exchange files from the macOS or any platform supporting python is to use the module SimpleHTTPServer (for download only by using $ python -m SimpleHTTPServer [%options]

Alternatively, you can implement the HTTP verbs GET or POST and the handling of the upload requests. 2 implementations can be found here:
https://gist.github.com/touilleMan/eb02ea40b93e52604938 (py3)
https://gist.github.com/UniIsland/3346170 (py27)

The modern Python3 equivalent of the SimpleHTTPServer is "**http.server**" module, started minimalistically as following:
$ python -m http.server

Chapter 20. Remote iPhone Data Acquisition

When it comes to digital forensics, the Apple iPhone as well as other mobile devices provide limited capabilities in what can be extracted and analysed. Due to strict isolation of certain system parts and components, it's not possible to have full disk image dump like in traditional computer forensics. Certain methods exist to obtain privileged access, such as jailbreaking or possible physical extraction of the memory chip, but they both void warranty for the device or are inaccessible in the remote way of operation.
Nevertheless, remote partial iOS inspection is possible and may be used to identify compromised devices without voiding their warranty.
A set of tools that comes with libimobiledevice library provides rich capabilities of collecting information from the iOS devices, including but not limited to

- iOS device UUID and name
- Full list of firmware version, hardware and related hardware identifiers
- State of the SIM card
- iOS application crash dump, kernel panic logs (if any) and sysdiagnose logs (created on demand)

https://t.me/learningnets

- Full iOS user files backup
- Screen contents

Note, that remote operation with the iOS device requires access to physical display. That is why the owner of the device and the remote expert shall work closely together.

To make use of the library and associated tools on Bitscout the following procedure is required.

Device owner:
1. Starts Bitscout from a USB/CD on a laptop or desktop computer.
2. Attaches an iOS device via USB cable.

Expert:
1. Connects via SSH to Bitscout-host instance. Note, that Bitscout-host access is required to work with USB devices. Future versions of Bitscout may provide such access via Bitscout-container too.
2. Lists available USB devices attached to a USB controller: `lsusb`. One of the devices should be the Apple iPhone.
3. Make sure usbmuxd.service is running by checking its state via `systemctl status usbmuxd`. The service should report a connected USB device.
4. Installs (if required) libimobiledevice-utils package: `apt update && apt -y install libimobiledevice-utils`
5. Uses idevice_id to list attached devices' UUIDs: `idevice_id -l`
6. Runs `idevicename` to get the attached device name
7. ideviceinfo tool outputs the device firmware version, SIM state, hardware version and identifiers, and more
8. idevicecrashdump can extract crash reports, sysdiagnose reports, kernel panic logs. Use it with the following commandline: `idevicecrashdump -e -k ./crashlogs`
9. To request full iOS user files backup use `idevicebackup2 backup ./backupdir`
10. The iOS screen contents can be requested, but it requires 2 steps: uploading an iOS Developer Disk (with `ideviceimagemounter DMG DMG.signature` command) onto the iPhone, requesting the screen contents with idevicescreenshot tool. Note that the DMG file has to be downloaded separately. It comes included with Apple Xcode and takes time finding the right DMG for your iOS version. We recommend using any public resource to fetch such DMG since Apple iPhone will accept only genuine Developer Disk images digitally signed by Apple. Once such source is currently on GitHub:
https://github.com/xushuduo/Xcode-iOS-Developer-Disk-Image/releases

If you have full user files backup (includes SMS and calls database, apps data, user security consents, etc) you may scan it with open-source mvt-ios tool. Here is an example:

```
apt update
apt install python3-pip
pip3 install mvt
mvt-ios -o ./scanlogs -i ./iocs.stix ./backupdir
```


https://t.me/learningnets

Note that to use mvt-ios you need to have IoCs in STIX format. An example of openly available IoCs for iOS can be downloaded here:
https://github.com/AmnestyTech/investigations/blob/master/2021-07-18_nso/pegasus.stix2?raw=true

Full analysis of the sysdiagnose contents, kernel panic logs, contents of user backups goes beyond the scope of this exercise and may be specific to iOS version.

Chapter 21. Intro Into Cloud Forensics

If you happen to be in position to investigate a partial compromise of a cloud infrastructure, you will need tools. Running forensic analysis on a cloud of servers is possible and well suited for Bitscout headless mode of operation. Some important things to note on the specifics of distributed usage of Bitscout for cloud forensics is provided below.

1. Make sure you build Bitscout raw disk image instead of ISO or other format. Otherwise, the cloud management software will not be able to unpack and modify your uploaded image. Such access and modification is required by the cloud to make sure network settings are set accordingly.
2. Enable LAN Access and Host Access if you are the sole user of the Bitscout instance and not planning to rely on a back connect VPN.
3. Enable automounting of existing partitions: this will save your time when you need to run some check on the fs contents.

The settings above can be applied by editing ./config/bitscout-build.conf file accordingly.

Once this is done, you can deploy Bitscout image in the cloud. The idea is similar to analyzing an isolated system, where you start Bitscout and have the subject system fully powered off and inspected in a forensically sound manner with read-only access. Deploying Bitscout consists of the following phases done by the cloud infrastructure administrator:

1. Take a snapshot of all server instances to be checked
2. Upload Bitscout raw disk and take a snapshot of that
3. Create a new instance from Bitscout snapshot and subject server instance merged into one. The instance boot process shall be bound to Bitscout, of course.
4. Start all the instances and pass the expert a list of instance IPs to conduct a forensic investigation on.

Once the instances are running, you can use multithreaded ssh/scp operations to run certain commands on the whole cloud. One of the tools to facilitate such operation is known as parallel-ssh aka pssh. It allows the user to run the same command on a merely unlimited number of SSH servers. Similarly, parallel-scp aka pscp can copy files from the expert host to a cloud of remote Bitscout instances with a single command.

https://t.me/learningnets

Below is an example exercise to locate malicious files on a private cloud of servers at Amazon Cloud.

Getting the cloud IPs

Some prerequisites installation is required before we proceed:
`sudo apt update && sudo apt install python3-boto3 pssh parallel`

After the Bitscout instances are running with the help of the cloud administrator (cloud owner in this context), the expert needs to request access keys to query the cloud properties and obtain the list of server IPs. This can be achieved with Python3 boto3 module and the following script:

```
#!/usr/bin/env python3
import boto3

KEY = '...'
SECRET_KEY = '...'

session = boto3.Session(aws_access_key_id=KEY, aws_secret_access_key=SECRET_KEY)
ec2 = session.client('ec2', region_name='...')

all_instances = []
all_ips = []
instance_ids = ec2.describe_instances()['Reservations']
for i in instance_ids:
        for j in i['Instances']:
        try:
        print(j['InstanceId'], j['State'],j['PublicIpAddress'])
        all_instances.append(j['InstanceId'])
        all_ips.append(j['PublicIpAddress'])
        except:
        print(j['InstanceId'], j['State'])
        all_instances.append(j['InstanceId'])

ipfile = 'ips.txt'
with open(ipfile, 'w') as f:
        f.write('\n'.join(all_ips))

print(f"Found {len(all_ips)} IP addresses, written in {ipfile}")
```

Note, that you need to fill the missing elements (by replacing the "…" with the actual values received from the cloud administrator).

https://t.me/learningnets

After executing the script, it shall produce ips.txt in the current directory. The file contains a simple list of the cloud instance IPs that are ready to be analysed.
Now you can work with an individual Bitscout instance in usual way:
`ssh -i ~/.ssh/scout_aws root@xxx.xxx.xxx.xxx`
Or run commands on all server simultaneously:
`parallel-ssh -h ./ips.txt -i -l root -x "-i ~/.ssh/scout_aws -o StrictHostKeyChecking=accept-new -o GSSAPIAuthentication=no -o PreferredAuthentications=publickey -o PubkeyAuthentication=yes" whoami`

Assigning aliases

To simplify the later, it is recommended to assign an alias to it, as follows:

```
alias pssh="parallel-ssh -h ips.txt -i -l root -x \"-i ~/.ssh/scout_aws -o
StrictHostKeyChecking=accept-new -o GSSAPIAuthentication=no -o
PreferredAuthentications=publickey -o PubkeyAuthentication=yes\""
```

This way you may just run `pssh hostname` and see the list of all hostnames from the cloud instances.

Similar alias will be convenient for parallel-scp tool:

```
alias pscp='parallel-scp -h ips.txt  -l root -x "-i scout_aws -o StrictHostKeyChecking=accept-new
-o GSSAPIAuthentication=no -o PreferredAuthentications=publickey -o
PubkeyAuthentication=yes"'
```

Running distributed yara scan

Installing missing software packages, i.e. yara, becomes simple too: `pssh 'apt update > /dev/null && apt -y install yara > /dev/null'`

If you would like to run a yara scan on some specific partition or a subdirectory inside the cloud instances you should first locate how the disk partitions were mounted and build up the path to scan:

```
pssh ls -1 /mnt/host/
```

Next, upload a yara rule file onto all Bitscouts with

```
pscp ./rule.yara /tmp/
```

And, lastly, run the scan with `pssh yara -r /tmp/rule.yara /mnt/host/…`

Acquiring filesystem metadata

Since we can run tools like fls in distributed fashion now, collecting the filesystem metadata is easy with a command like this: `pssh "fls -rlp -z GMT /dev/host/evidence0 | gzip -c > /tmp/fls.gz"`.

https://t.me/learningnets

However, the pscp tool cannot be used to collect the data out of the cloud. The pscp is used in one-directional transfer: host → cloud instance. In order to transfer files from all servers, we need to use regular scp in combination with another tool known as parallel:

```
parallel -j0 scp -i ./.ssh/scout_aws root@{}:/tmp/fls.gz {}_fls.gz < /tmp/ips.txt
```

As a result current directory will be populated with *_fls.gz files named after IPs of the servers they were downloaded from. Customise this command how you see fit.

TLP:Red Sharing of this document without permission of the authors is prohibited.

https://t.me/learningnets