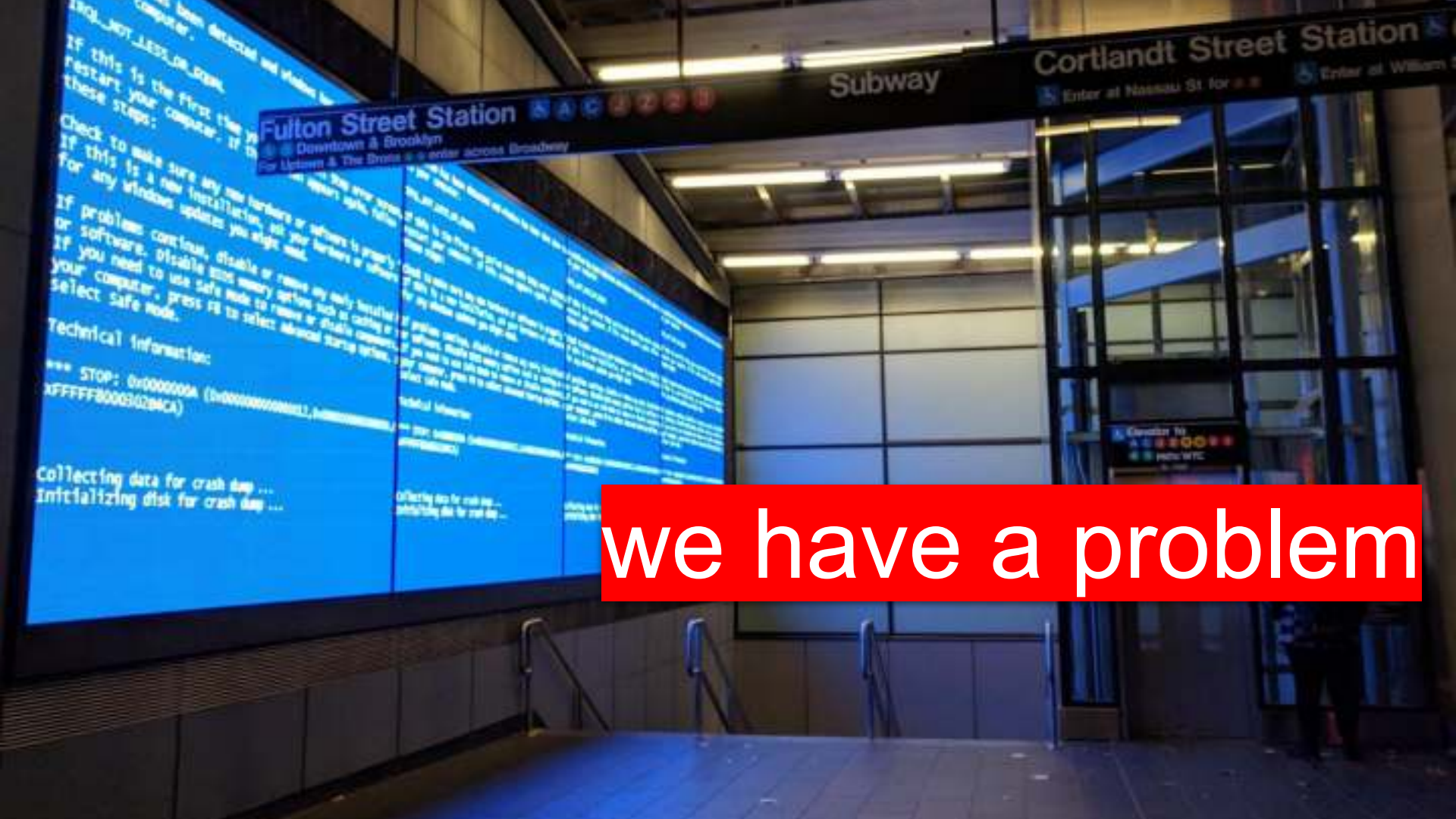
Safely Booting Windows with Linux

Trammell Hudson

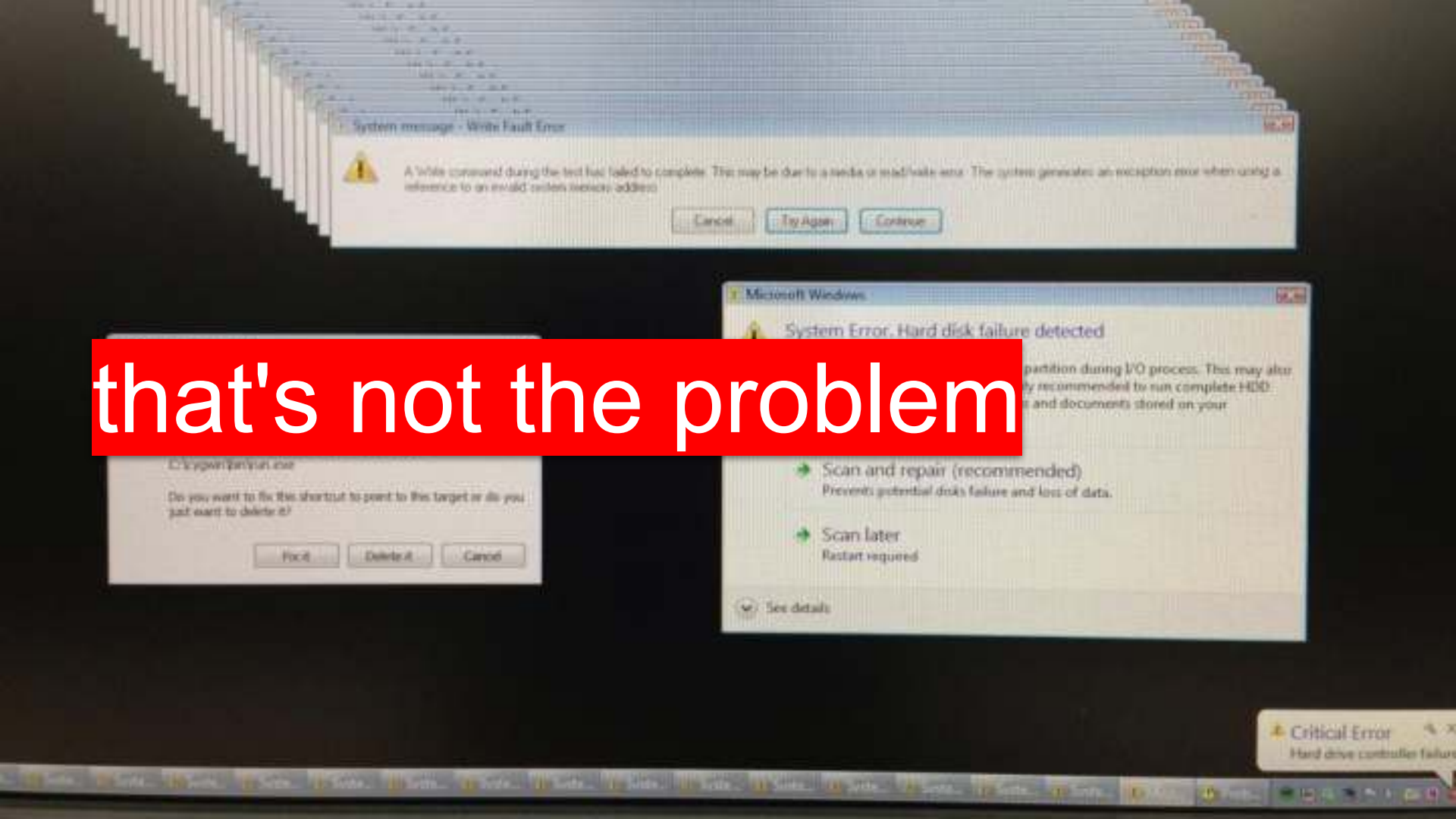
https://safeboot.dev/
https://github.com/osresearch/safeboot-loader/


Fulton Street Station
Downtown & Brooklyn
Subway
Cortlandt Street Station
Technical information:
Collecting data for crash dump ...
Initializing disk for crash dump ...
we have a problem


digital pdp11
digital pdp11
we have servers

with local disks


that run Windows


System message - Write Fault Error
Cancel
Try Again
Continue
Microsoft Windows
System Error. Hard disk failure detected
Scan and repair (recommended)
Prevents potential disks failure and loss of data.
Scan later
Restart required
See details
Do you want to fix the shortcut to point to the target or do you just want to delete it?
Fix it
Delete it
Cancel
Critical Error
that's not the problem


where do those secrets go
during decomissioning?

{* SECURITY *}

https://www.theregister.com/2016/06/28/ebay_hard_drives_still_contain_sensitive_data_study/

You know how that data breach happened? Three words: eBay, hard drives

Social Security Numbers, financial data, CVs and more

John Leyden

Tue 28 Jun 2016 // 16:20 UTC

84

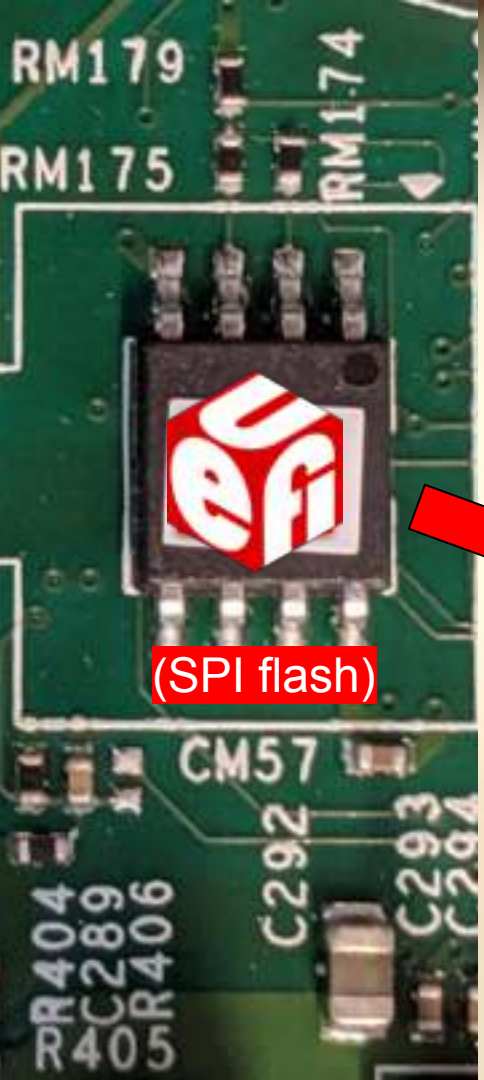
Two-thirds contained personally identifiable information and 11% contained sensitive company information

forensic analysis to find out what data was recoverable. Two-thirds (67 per


"this sounds like a job for encryption!"
ПУСК

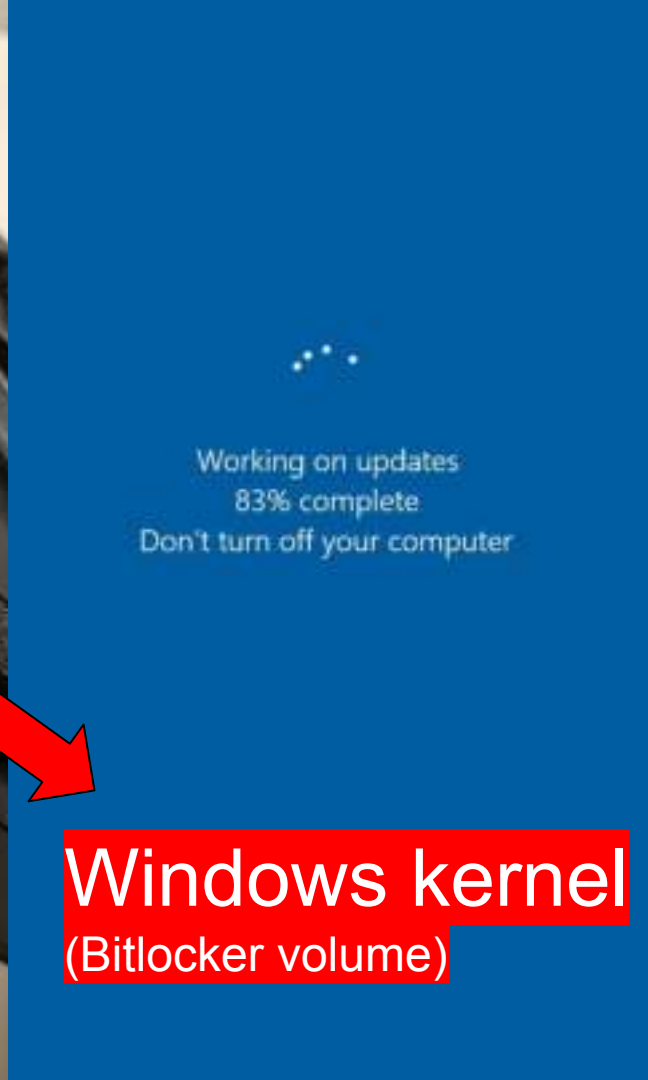

now we have a
key management problem


the boot process
Boot Flow
Embedded Recipes
SHILE

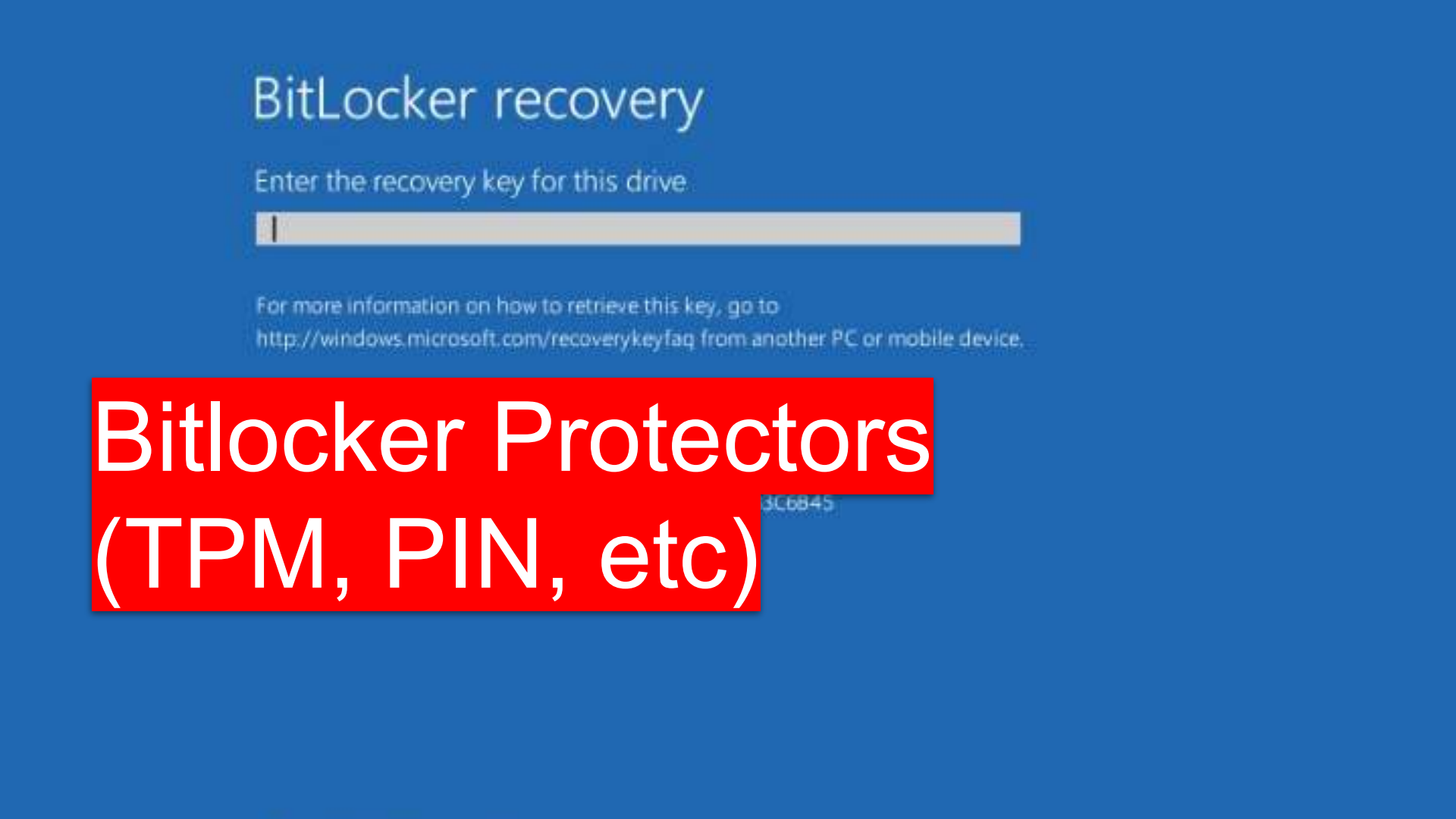


(SPI flash)

bootmgfw.efi
(EFI System Partition)



Windows kernel
(Bitlocker volume)


BitLocker recovery
Enter the recovery key for this drive
For more information on how to retrieve this key, go to
http://windows.microsoft.com/recoverykeyfaq from another PC or mobile device.
Bitlocker Protectors
(TPM, PIN, etc)

we want
unattended booting…

DOLOS GROUP

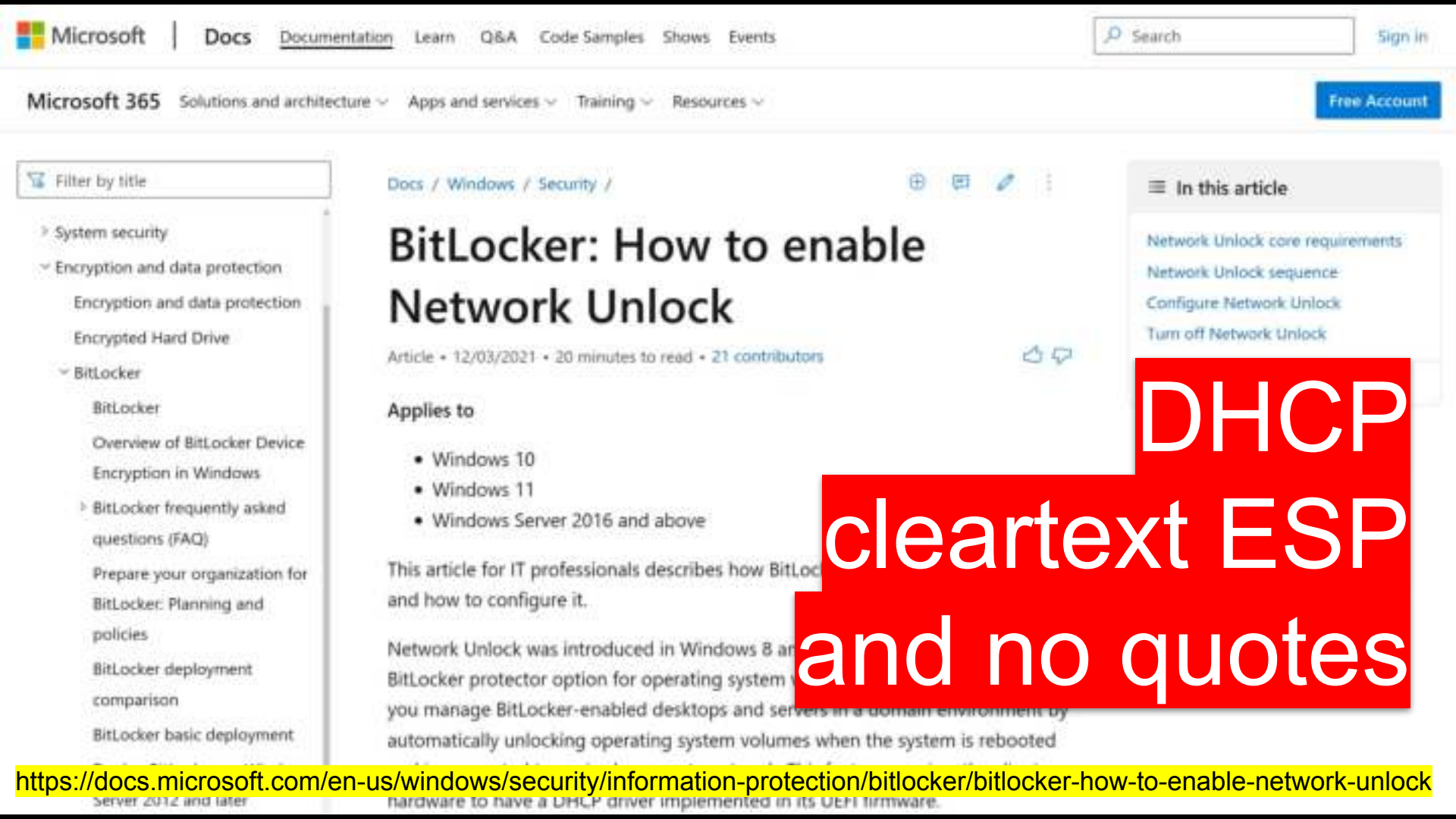
https://dolosgroup.io/blog/2021/7/9/from-stolen-laptop-to-inside-the-company-network

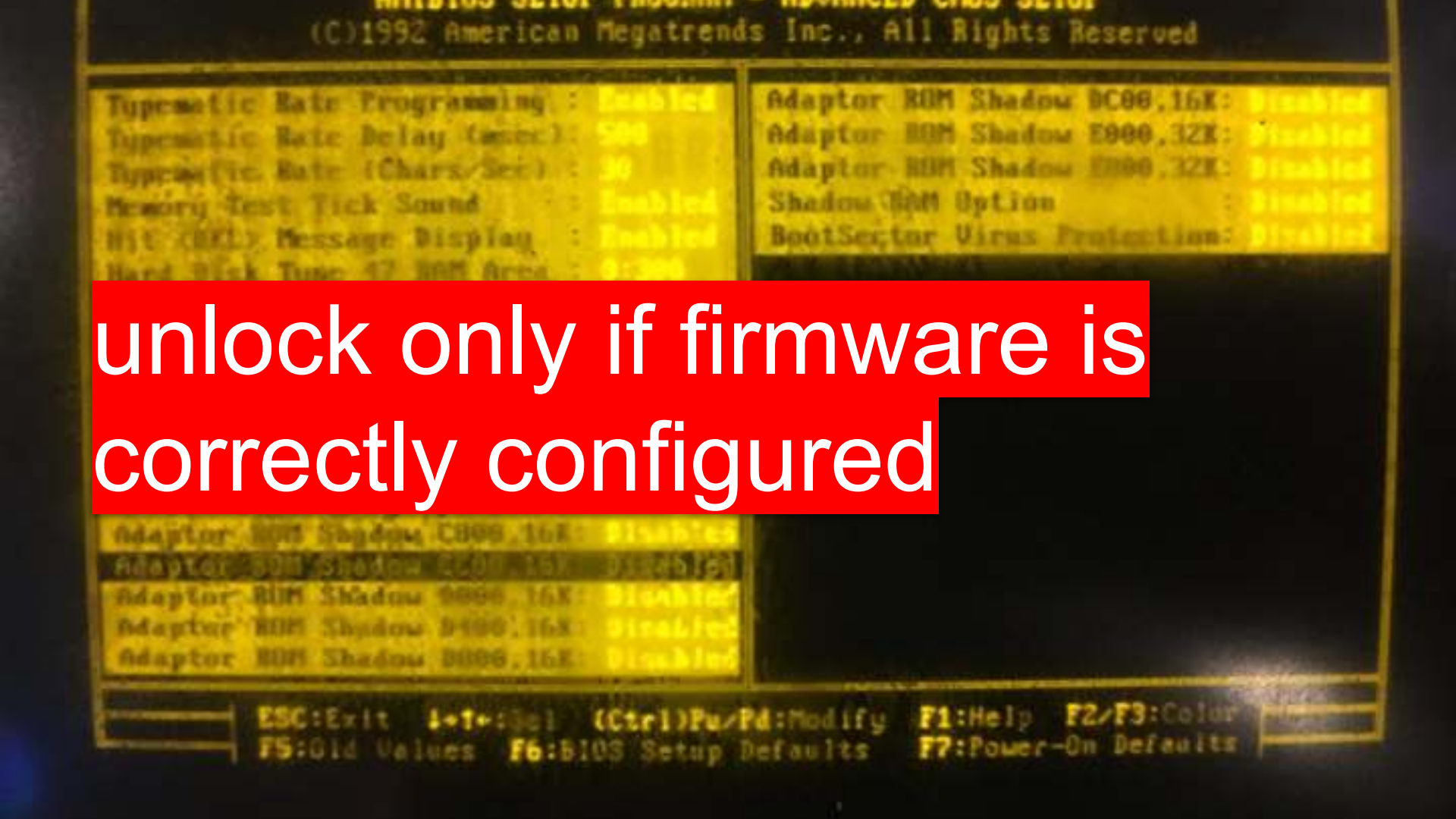
JULY 28, 2021

FROM STOLEN LAPTOP TO INSIDE THE COMPANY NETWORK

we took a locked down FDE laptop & sniffed the BitLocker decryption key coming out of the TPM


… only on our network


Microsoft | Docs Documentation Learn Q&A Code Samples Shows Events
Search
Sign in
Microsoft 365 Solutions and architecture Apps and services Training Resources
Free Account
Filter by title
System security
Encryption and data protection
Encryption and data protection
Encrypted Hard Drive
BitLocker
BitLocker
Overview of BitLocker Device Encryption in Windows
BitLocker frequently asked questions (FAQ)
Prepare your organization for BitLocker: Planning and policies
BitLocker deployment comparison
BitLocker basic deployment
Docs / Windows / Security /
BitLocker: How to enable Network Unlock
Article • 12/03/2021 • 20 minutes to read • 21 contributors
Applies to
Windows 10
Windows 11
Windows Server 2016 and above
This article for IT professionals describes how BitLoc
and how to configure it.
Network Unlock was introduced in Windows 8 ar
BitLocker protector option for operating system
you manage BitLocker-enabled desktops and servers in a domain environment by
automatically unlocking operating system volumes when the system is rebooted
hardware to have a DHCP driver implemented in its UEFI firmware.
In this article
Network Unlock core requirements
Network Unlock sequence
Configure Network Unlock
Turn off Network Unlock
DHCP
cleartext ESP
and no quotes
https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-how-to-enable-network-unlock

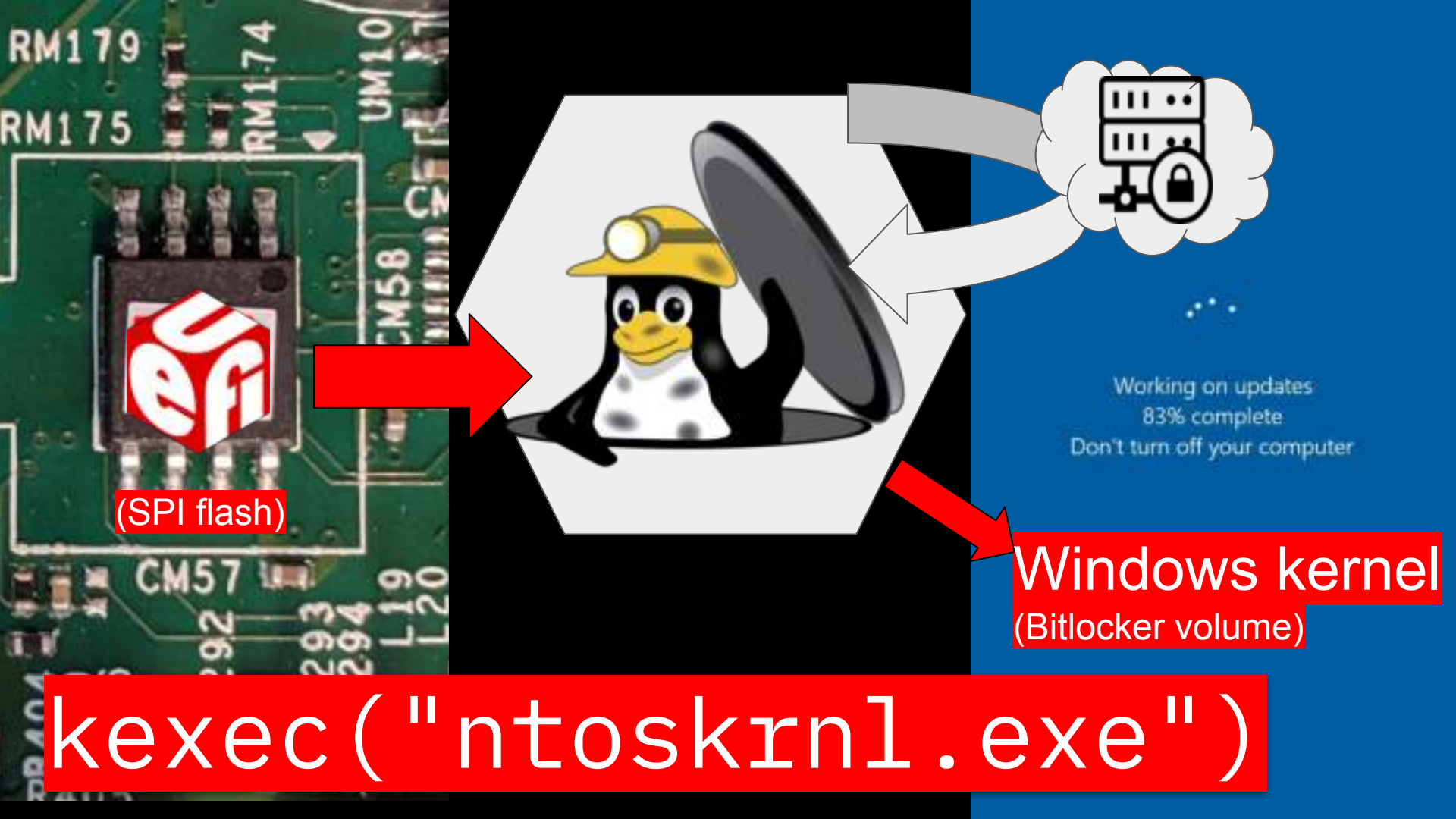

(C)1992 American Megatrends Inc., All Rights Reserved
Typematic Rate Programming : Enabled
Typematic Rate Delay (msec) : 500
Typematic Rate (Chars/Sec) : 30
Memory Test Tick Sound : Enabled
Hit (DEL) Message Display : Enabled
Adaptor ROM Shadow DC00,16K: Disabled
Adaptor ROM Shadow E000,32K: Disabled
Adaptor ROM Shadow E800,32K: Disabled
Shadow ROM Option : Disabled
BootSector Virus Protection: Disabled
unlock only if firmware is correctly configured
Adaptor ROM Shadow C800,16K: Disabled
Adaptor ROM Shadow D000,16K: Disabled
Adaptor ROM Shadow D400,16K: Disabled
Adaptor ROM Shadow D800,16K: Disabled
ESC:Exit ↓→↑←:Sel (Ctrl)Pu/Pd:Modify F1:Help F2/F3:Color
F5:Old Values F6:BIOS Setup Defaults F7:Power-On Defaults


to detect local attackers


"this sounds like a job for remote attestation!"

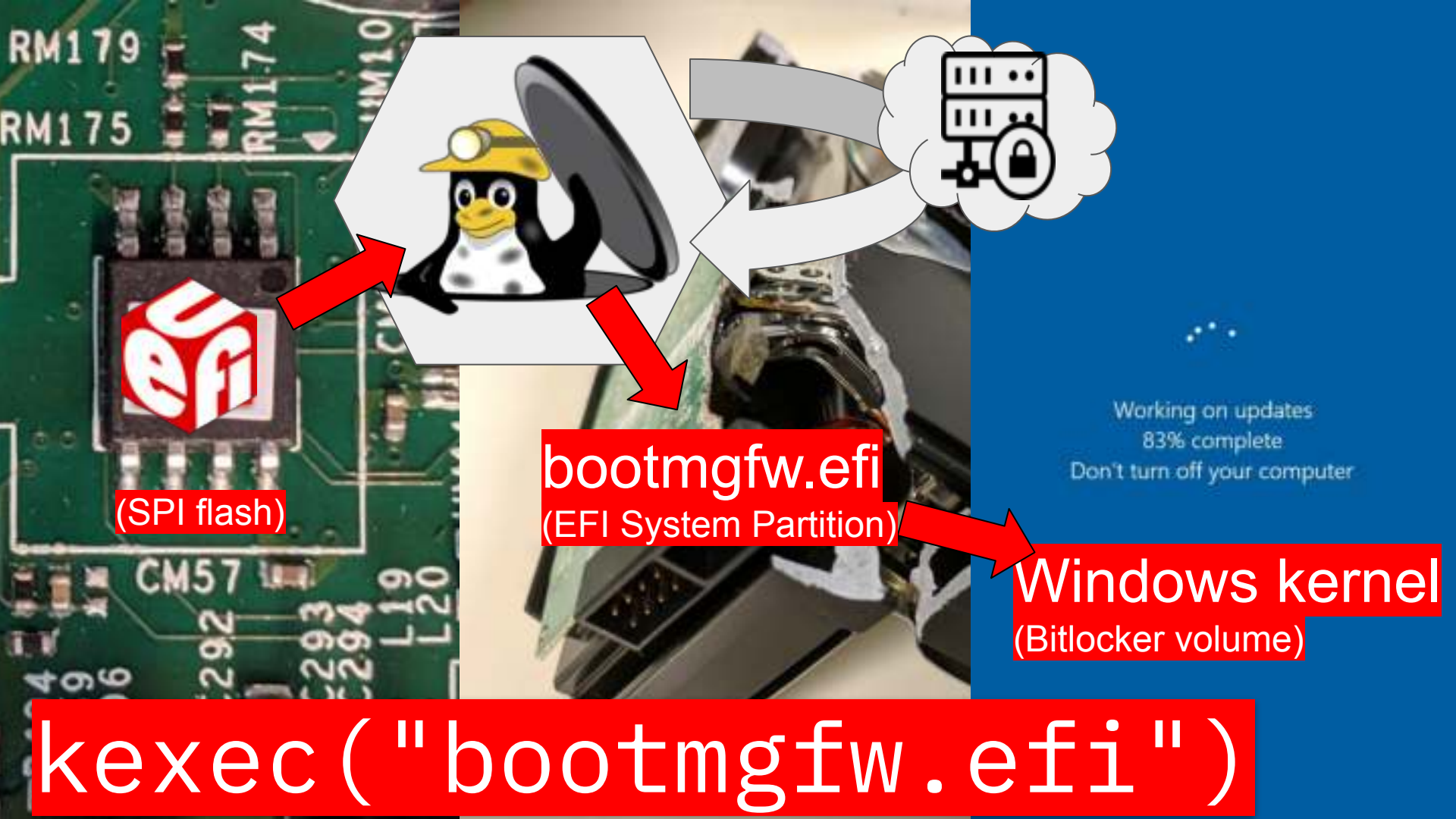

which works for Linux
safeboot @ OSFC 2020


what about legacy systems?

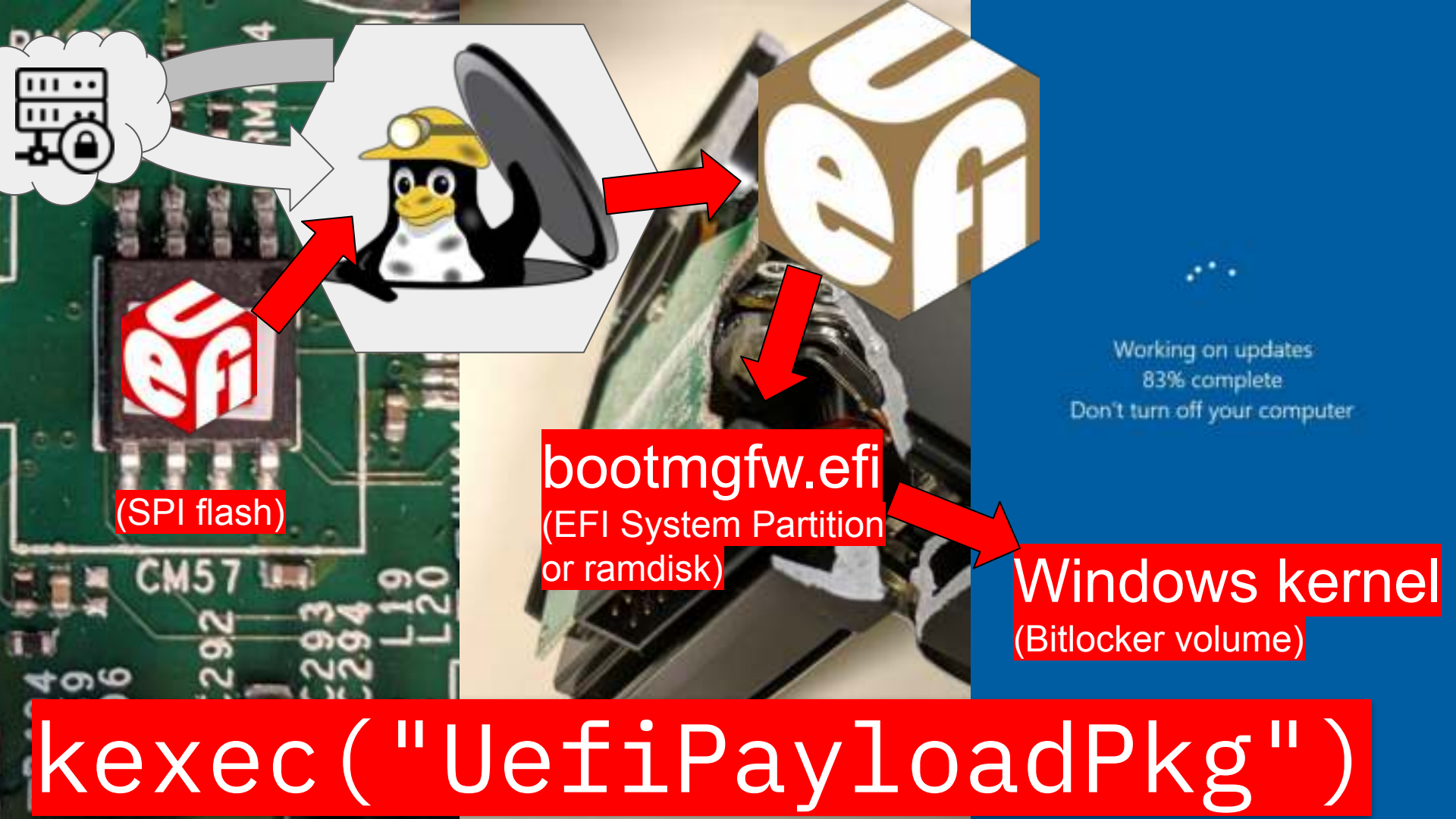

RM179
RM175
RM174
UM10
CM58
CM57
(SPI flash)
Working on updates
83% complete
Don't turn off your computer
Windows kernel
(Bitlocker volume)
kexec("ntoskrnl.exe")

undocumented APIs
fragile to Windows versions

https://github.com/maharmstone/quibble

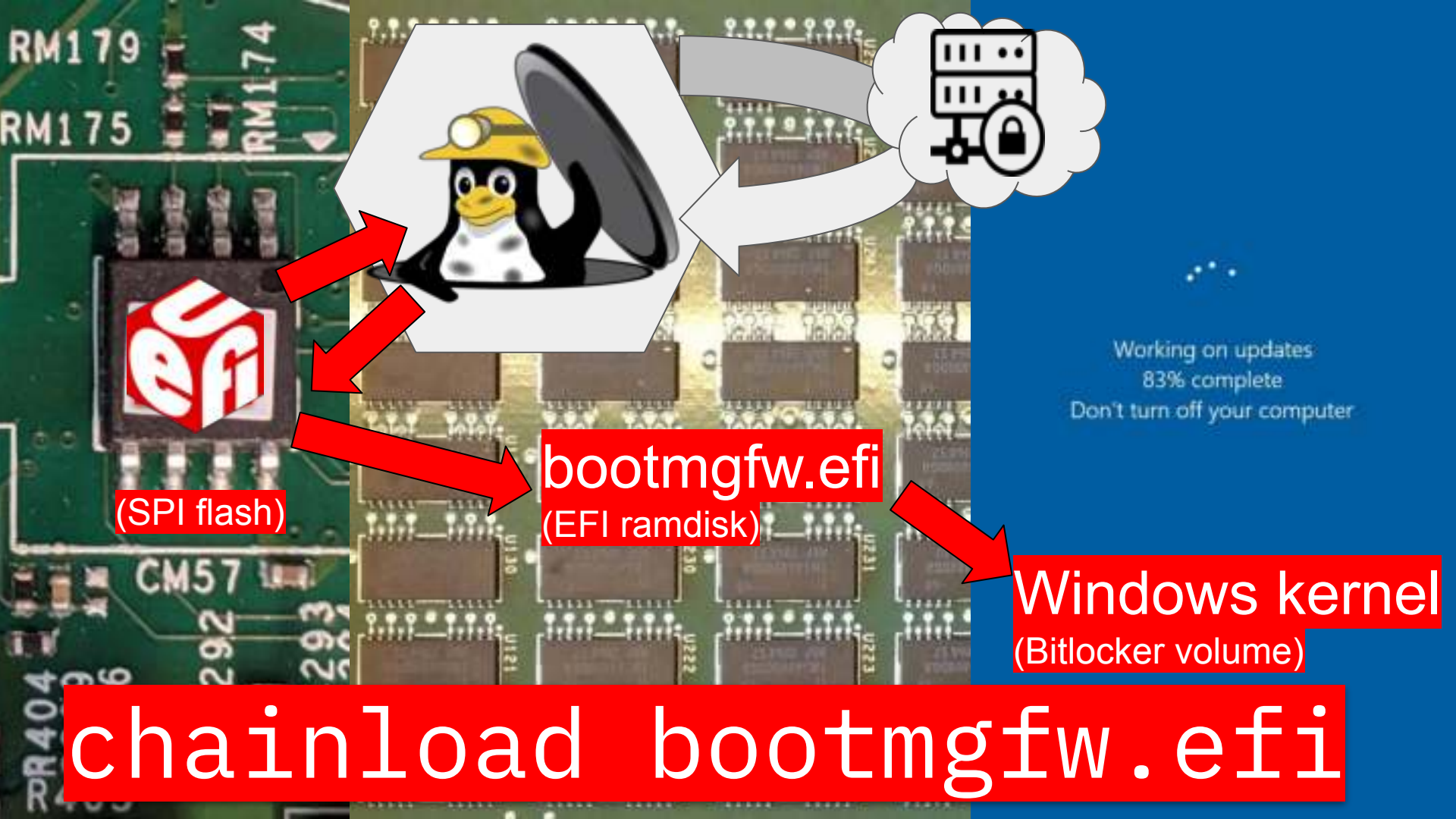

RM179
RM175
(SPI flash)
bootmgfw.efi
(EFI System Partition)
Windows kernel
(Bitlocker volume)
Working on updates
83% complete
Don't turn off your computer
kexec("bootmgfw.efi")

kexec only supports
bzImage & multiboot
No EFI Boot services


(SPI flash)
bootmgfw.efi
(EFI System Partition
or ramdisk)
Windows kernel
(Bitlocker volume)
Working on updates
83% complete
Don't turn off your computer
kexec("UefiPayloadPkg")

Not the real Boot Services
Limited device drivers
No NVRAM
No SMM

https://platformsecuritysummit.com/2019/speaker/koch/PSEC2019-LinuxBoot-Chris-Koch.pdf


RM179
RM175
RM174
(SPI flash)
bootmgfw.efi
(EFI ramdisk)
Working on updates
83% complete
Don't turn off your computer
Windows kernel
(Bitlocker volume)
chainload bootmgfw.efi

All OEM drivers
Real NVRAM
Real SMM
Must not disturb anything

https://github.com/osresearch/safeboot-loader

how does it work?

PXE boot safeboot-loader

(signed wrapper+kernel+initrd+commandline)

```
>>Start PXE over IPv4.
  Station IP address is 10.0.2.15

  Server IP address is 10.0.2.2
  NBP filename is ./build/bootx64.efi
  NBP filesize is 16505012 Bytes
 Downloading NBP file...

  NBP file downloaded successfully.
BdsDxe: loading Boot0003 "UEFI PXEv4 (MAC:525400123456)" from PciRoot(0x0)
x0)/MAC(525400123456,0x1)/IPv4(0.0.0.0,0x0,DHCP,0.0.0.0,0.0.0.0,0.0.0.0)
BdsDxe: starting Boot0003 "UEFI PXEv4 (MAC:525400123456)" from PciRoot(0x0
0x0)/MAC(525400123456,0x1)/IPv4(0.0.0.0,0x0,DHCP,0.0.0.0,0.0.0.0,0.0.0.0)
```

safeboot-loader UEFI wrapper

```
SysTab:  000000007FBEE018
BootSvc: 000000007FF2FEC0
Reserved 0000000040000000 + 20000000 for Linux
kernel:  000000007C3F9000 + 0026F2A0
initrd:  000000007C669000 + 00D42000
cmdline: 000000007D3B9000 + 0000006B "memmap=exactmap,128K@0G,512M@1G"
CR3=000000007FC01000
GDT=7FBEE698 + 0047
IDT=7F2D0018 + 0FFF
LDT=00000000 + 0000
Not exiting boot services this time...
[    0.000000] Linux version 5.4.117 (builder@safeboot) (gcc version 9
.4.0-1ubuntu1~20.04.1)) #loader-uefidev 514f6769f70d7985
```

Linux does not call

`gBS->ExitBootServices()`

`noefi acpi=off pci=off`

No Linux device drivers

Linux device drivers wrap UEFI protocols

Unified Extensible Firmware Interface (UEFI) Specification

Version 2.8

March 2019

EFI_SIMPLE_NETWORK_PROTOCOL

EFI_SIMPLE_NETWORK_PROTOCOL

Summary

The EFI_SIMPLE_NETWORK_
packets, receive packets, and close a network interface.

GUID

```
#define EFI_SIMPLE_NETWORK_PROTOCOL_GUID \
 {0xA1983289,0xAC25,0x11D3,\
  {0x9A,0x2D,0x00,0x90,0x27,0x3f,0xc1,0x4d}}
```

Revision Number

```
#define EFI_SIMPLE_NETWORK_P
```

Protocol Interface Structure

```
typedef struct _EFI_SIMPLE_N
 UINT64          Revision;
 EFI_SIMPLE_NETWORK_START
 EFI_SIMPLE_NETWORK_STOP
 EFI_SIMPLE_NETWORK_INITIALI
 EFI_SIMPLE_NETWORK_RESET
 EFI_SIMPLE_NETWORK_SHUTDOW
 EFI_SIMPLE_NETWORK_RECEIVE_
 EFI_SIMPLE_NETWORK_STATION_
 EFI_SIMPLE_NETWORK_STATIST
 EFI_SIMPLE_NETWORK_MCAST_IP
 EFI_SIMPLE_NETWORK_NVDATA
 EFI_SIMPLE_NETWORK_GET_STA
 EFI_SIMPLE_NETWORK_TRANSMIT
 EFI_SIMPLE_NETWORK_RECEIVE      Receive;
 EFI_EVENT                                WaitForPacket;
 EFI_SIMPLE_NETWORK_MODE                  *Mode;
} EFI_SIMPLE_NETWORK_PROTOCOL;
```

```
static netdev_tx_t uefi_net_xmit(struct sk_buff * skb, struct net_device *
{
        status = nic->uefi_nic->Transmit(
                nic->uefi_nic,
                0,                      // HeaderSize 0 == packet is fully formed
                skb->len,               // BufferSize
                skb->data,              // Buffer,
                NULL,                   // SrcAddr, unused since HeaderSize == 0
                NULL,                   // DstAddr, unused
                NULL                    // Protocol, unused
        );

        dev_consume_skb_any(skb);
```

EFI_BLOCK_IO_PROTOCOL

GUID

```
#define EFI_BLOCK_IO_PROTOCOL_GUID \
 {0x964e5b21,0x6459,0x11d2,\
  {0x8e,0x39,0x00,0xa0,0xc9,0
```

Revision Number

```
#define EFI_BLOCK_IO_PROTOCOL_
#define EFI_BLOCK_IO_PROTOCOL_
```

Protocol Interface Structure

```
typedef struct _EFI_BLOCK_IO_
  UINT64                Revisi
  EFI_BLOCK_IO_MEDIA    *Media
  EFI_BLOCK_RESET       Reset;
  EFI_BLOCK_READ        ReadBl
  EFI_BLOCK_WRITE       WriteB
  EFI_BLOCK_FLUSH       FlushB
} EFI_BLOCK_IO_PROTOCOL;
```

Parameters

Revision The revision to which the block IO interface ad

```
static blk_status_t uefi_blockdev_request(
        struct blk_mq_hw_ctx * hctx, const struct blk_mq_queue_data
{
        struct request * rq = bd->rq;
        uefi_blockdev_t * dev = rq->rq_disk->private_data;

        rq_for_each_segment(bvec, rq, iter) {
                char * buffer = kmap_atomic(bvec.bv_page);
                EFI_BLOCK_READ handler = is_write
                        ? dev->uefi_bio->WriteBlocks
                        : dev->uefi_bio->ReadBlocks;
                status |= handler(
                        dev->uefi_bio,
                        dev->uefi_bio->Media->MediaId,
                        disk_sector,
                        full_block_len,
                        dest
                );
                kunmap_atomic(buffer);
        }
```

EFI_RAM_DISK_PROTOCOL

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_RAM_DISK_REGISTER_RAMDISK) (
 IN UINT64                  RamDiskBase,
 IN UINT64                  RamDiskSize,
 IN EFI_GUID
 IN EFI_DEVICE_PATH
 OUT EFI_DEVICE_PATH_PROTOCO
);
```

Parameters

RamDiskBase The base

RamDiskSize The size

RamDiskType The type of defined

ParentDevicePath Pointer t then *Par*

DevicePath On retur device. I *Device* device p

DevicePath is a RAM disk device path without appending. This function is responsible for allocating the buffer *DevicePath* with the boot service `AllocatePool()`.

```
+ fdisk /tmp/ramdisk.121.img < /tmp/gpt-script
+ echo /tmp/ramdisk.121.img > /sys/efi/firmware/ramdisk
[    8.420000] uefi_read_file: /tmp/ramdisk.121.img => 1048576
[    8.420000] uefi_ramdisk: /tmp/ramdisk.121.img
[    8.420000]  1048576 bytes
[    8.420000] uefi9: RamDisk(0x7E21A000,0x7E319FFF,0,AB38A0DF
[    8.440000] uefi9: rev=10000 id=0 removable=0 present=1 log
[    8.450000] uefi_blockdev: created 1 block devices
+ mkfs.vfat /dev/uefi9
mkfs.fat 4.1 (2017-01-24)
+ mount /dev/uefi9 /ramdisk
+ tar -C /ramdisk --no-same-owner -xvf /tmp/assets.tgz
```

EFI_BOOT_SERVICE.StartImage()

Summary

Transfers control to a loaded image's entry point.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_IMAGE_START) (
 IN EFI_HANDLE        ImageHandle,
 OUT UINTN            *ExitDataSize,
 OUT CHAR16           **ExitData OPTIONAL
 );
```

```
/# echo /bin/RamDiskDxe.efi > /sys/firmware/efi/loader
[   32.720000] uefi_read_file: /bin/RamDiskDxe.efi => 28672
[   32.730000] uefi_loader: starting /bin/RamDiskDxe.efi
[   32.730000]  (28672 bytes)
[   32.750000] uefi_loader: status=20 exit_data=0
```

Parameters

ImageHandle

ExitDataSize

parameter is ignored and the contents of *ExitDataSize* are not modified.

ExitData

Pointer to a pointer to a data buffer that includes a Null-terminated string, optionally followed by additional binary data. The string is a description that the caller may use to further indicate the reason for the image's exit.

EFI_TCG2_PROTOCOL

The EFI TCG2 protocol SHALL use the following GUID.

GUID –

```
#define EFI_TCG2_PROTOCOL_GUID \
  {0x607f766c, 0x7455, 0x42be, 0x93, \
    0x0b, 0xe4, 0xd7, 0x6d, 0xb2, 0x72, 0x0f}
```

6.1 Protocol Version

A user of this protocol should call the EFI_TCG2_PROTOCOL.GetCapability operation (Section 6.4) to determine the functionality implemented by this interface. There are earlier implementations of this protocol that implement a subset of the functions and capabilities defined here.

6.2 Protocol Interface Struct

```
typedef struct tdEFI_TCG2_PROTO
  EFI_TCG2_GET_CAPABILITY
  EFI_TCG2_GET_EVENT_LOG
  EFI_TCG2_HASH_LOG_EXTEND_EVEN
  EFI_TCG2_SUBMIT_COMMAND
  EFI_TCG2_GET_ACTIVE_PCR_BANKS
  EFI_TCG2_SET_ACTIVE_PCR_BANKS
  EFI_TCG2_GET_RESULT_OF_SET_AC

} EFI_TCG2_PROTOCOL;
```

```
static int uefi_tpm_send(struct tpm_chip * chip, u8 *buf, s
{
        uefi_tpm_t * const priv = dev_get_drvdata(&chip->de

        status = priv->uefi_tpm->SubmitCommand(
                priv->uefi_tpm,
                len,
                buf,
                sizeof(priv->recv_buf),
                priv->recv_buf
        );
```

TPM eventlog

```
/# cp /sys/kernel/security/tpm0/binary_bios_measurements /tmp/eventlog
/# tpm2 eventlog /tmp/eventlog | tail
pcrs:
  sha256:
    0  : 0x727ab8eb3a36dd771f029de555abb99fa7116b0a5b39c2c7b4ba28088a58f8fd
    1  : 0x5155deddd51610992c847113112bb53ddaacdbdaa80d29ee49eea428e8efdef8
    2  : 0x3f803541645e4d32a3f8a3a3d4850177c290a77c18c57e036568340ea7ab3f8d
    3  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    4  : 0xe3bee823383be5604bb6206445c7f65fb88ffe2f7f84b190d84035ac695b5133
    5  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    6  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    7  : 0x65caf8dd1e0ea7a6347b635d2b379c93b9a1351edc2afc3ecda700e534eb3068
/# tpm2 pcrread sha256:7
  sha256:
    7 : 0x65CAF8DD1E0EA7A6347B635D2B379C93B9A1351EDC2AFC3ECDA700E534EB3068
```

*"don't UEFI Boot Services use a **linear memory map**?"*

// hack!

```
// get the current CR3, masking out the ASID, to get the
// pointer to this Linux kernel thread's page table
__asm__ __volatile__("mov %%cr3, %0" : "=r"(cr3_phys));

// read the UEFI CR3 from the low memory cache
// and map it into this Linux kernel threads memory
uefi_cr3 = *(uint64_t*) phys_to_virt(0x100); // hack!
uefi_pagetable = ioremap(uefi_cr3 & ~0xFFF, 0x1000);

// Map UEFI's 1GB linear map of low memory into Linux's 1GB
linux_pagetable = phys_to_virt(cr3_phys & ~0xFFF);
linux_pagetable[0] = uefi_pagetable[0];

// Reload this thread's CR3
__asm__ __volatile__("mov %0, %%cr3" : : "r"(cr3_phys) : "memory");

// we can use the UEFI address space pointers now
gBS = gST->boottime;
```

safeboot-attest

```
+ ifconfig eth0 10.0.2.15
[    2.200000] uefi0: started nic
+ safeboot-attest http://10.0.2.2:5000/
tpm2: reading endorsement key
tpm2: creating ephemeral attestation key
tpm2: generating quote 628f56c9
attest: sending quote to http://10.0.2.2:5000/
attest: unsealing response
```

```
fe8e99bebff8ccd9fafa81c0e0556bf853f684e09fcaa93d2cc27129d6
valid quote and eventlog
./
./403B03D1-2D06-4E0A-B071-8D9434172633.BEK
fe8e99bebff8ccd9fafa81c0e0556bf853f684e09fcaa93d2cc27129d6
attest_verify passed, 317 bytes to send
fe8e99bebff8ccd9fafa81c0e0556bf853f684e09fcaa93d2cc27129d6
sealed into /tmp/tmp9on15bmq/credential.blob
127.0.0.1 - - [26/May/2022 10:30:35] "POST / HTTP/1.1" 200
```

Pass Bitlocker BEK through GPT partitioned ramdisk

```
+ fdisk /tmp/ramdisk.121.img < /tmp/gpt-script
+ echo /tmp/ramdisk.121.img > /sys/efi/firmware/ramdisk
[    8.420000] uefi_read_file: /tmp/ramdisk.121.img => 1048576
[    8.420000] uefi_ramdisk: /tmp/ramdisk.121.img
[    8.420000]  1048576 bytes
[    8.420000] uefi9: RamDisk(0x7E21A000,0x7E319FFF,0,AB38A0DF-6873-4
[    8.440000] uefi9: rev=10000 id=0 removable=0 present=1 logical=0
[    8.450000] uefi_blockdev: created 1 block devices
+ mkfs.vfat /dev/uefi9
mkfs.fat 4.1 (2017-01-24)
+ mount /dev/uefi9 /ramdisk
+ tar -C /ramdisk --no-same-owner -xvf /tmp/assets.tgz
```

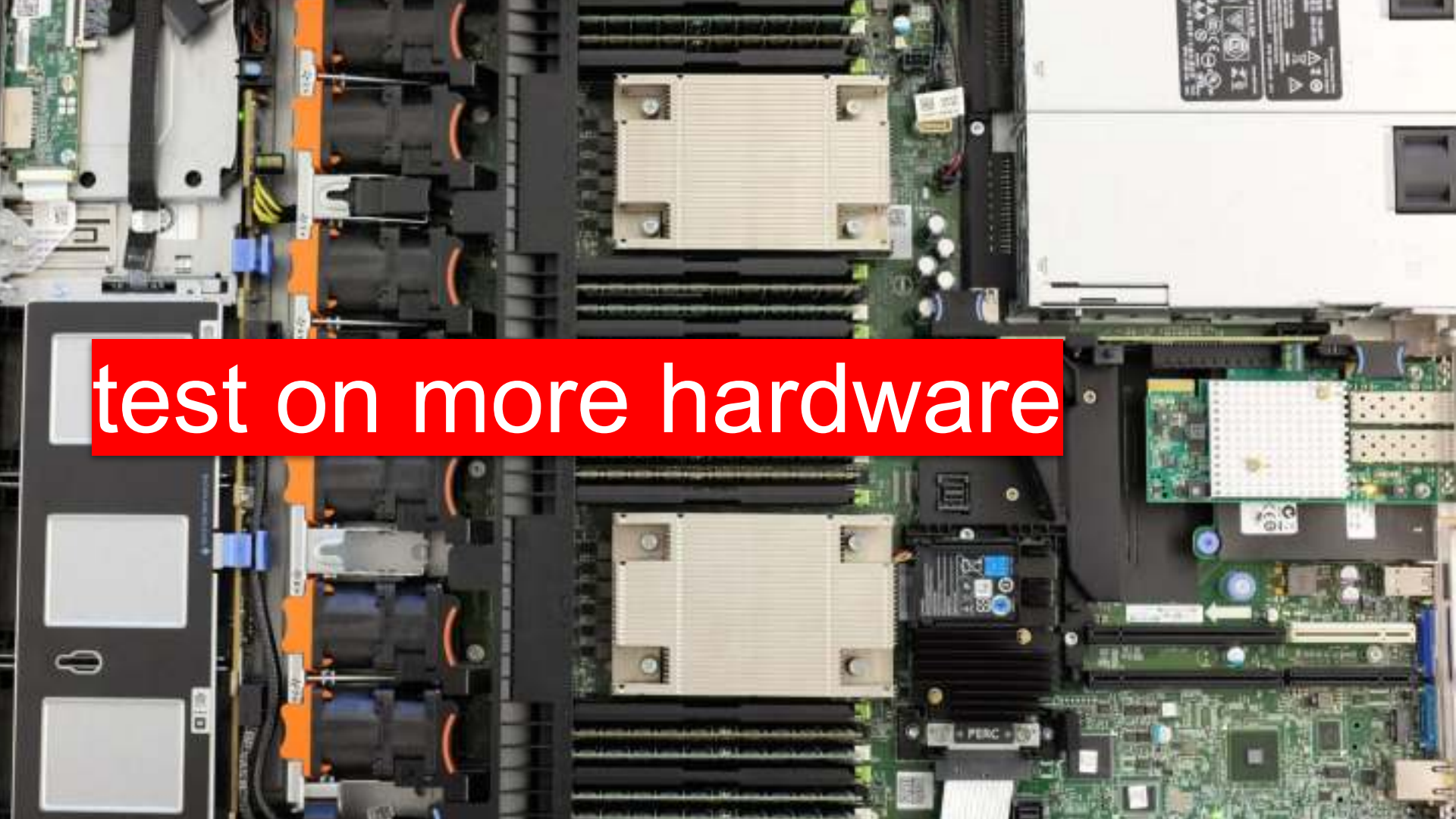
```
+ mount -o ro /dev/uefi6 /boot
+ chainload -v --boot-device uefi6 /boot/EFI/Microsoft/Boot/bootmgfw.efi
/boot/EFI/Microsoft/Boot/bootmgfw.efi: 1469752 bytes
context: CR3=0x7fc01000 gST=0x7fbee018
0: 0x55ff1b2f72a0 + 4096 => 0x0 + 4096
1: 0x55ff1a25a960 + 4336 => 0x40000000 + 8192
2: 0x7f41699ed010 + 1469752 => 0x40020000 + 1470464
3: 0x7ffdbec06510 + 32 => 0x40010000 + 4096
[   20.590000] kexec_core: Starting new kernel
chainload says hello
Boot device PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x0,0xFFFF,0x0)/
HD(2,GPT,D240F225-7427-4B55-AB3C-15B87C251E2D,0xFA000,0x31800)
```

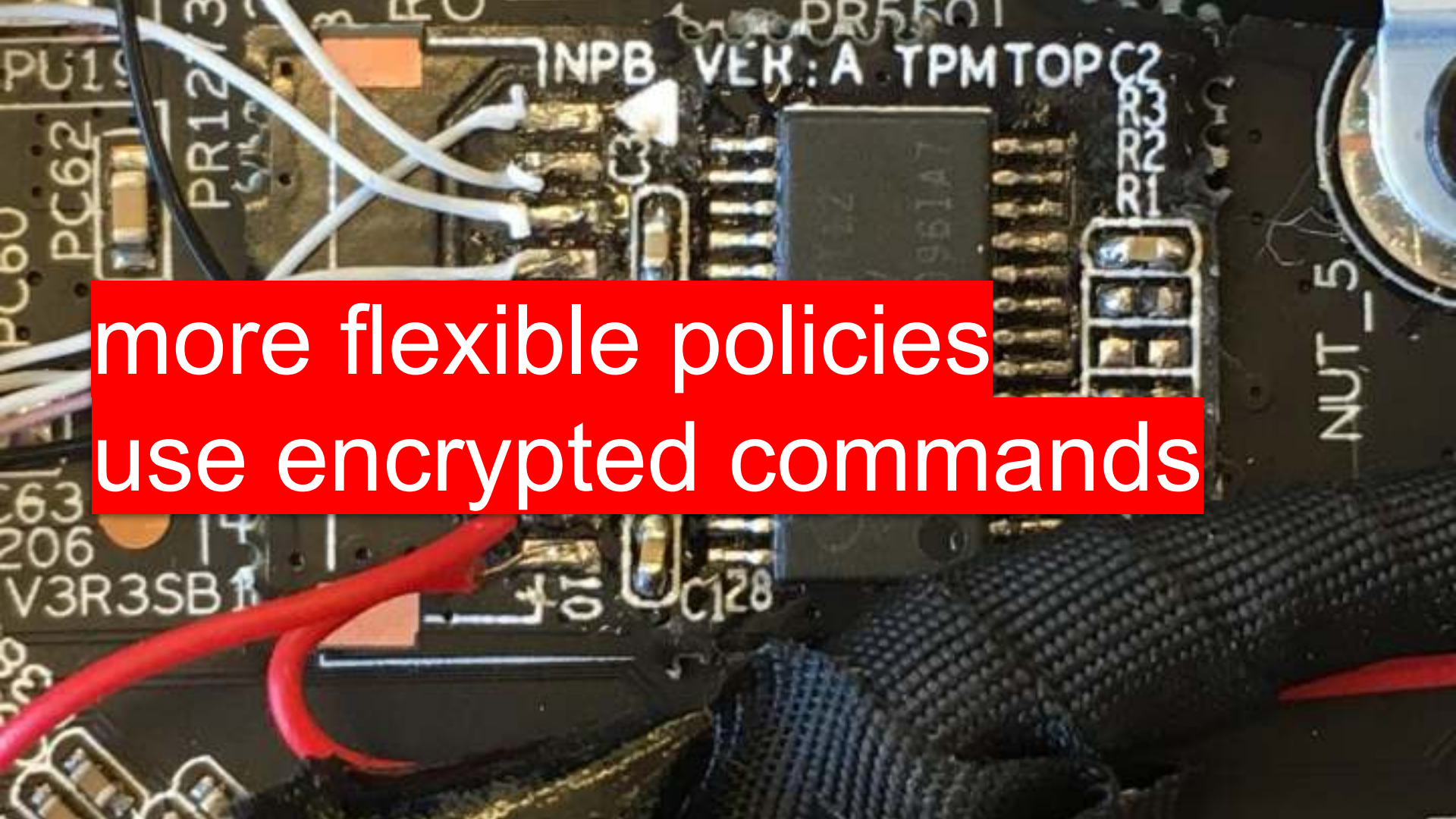
chainload Windows boot loader

```
drwxr-xr-x  2 0 0 16384 Jan  1  1970 ./
drwx------ 14 0 0   320 May 26 10:30 ../
-rwxr-xr-x  1 0 0   156 Mar 10 14:41 403B03D1-2D06-4E0A-B071-8D9434172633
.BEK*
+ umount /ramdisk
+ mount -o ro /dev/uefi6 /boot
+ ifconfig eth0 down
[    9.940000] uefi0: stopped nic
+ chainload -v -d uefi6 /boot/EFI/Boot/bootx64.efi
/boot
/dev/
/sys/                                                              18
context: CR3=0x7fc01000 gST=0x7f0ee018
0: 0x55e5bc18d2a0 + 4096 => (nil) + 4096
1: 0x55e5ba9ac960 + 4336 => 0x40000000 + 8192
2: 0x7f7285b4c010 + 1469752 => 0x40020000 + 1470464
3: 0x7ffcec49c910 + 32 => 0x40010000 + 4096
kexec-load: rebooting
[   10.460000] kexec_core: Starting new kernel
chainload says hello
Boot device PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x0,0xFFFF,0x0)/HD(2,GPT,D240
F225-7427-4B55-AB3C-15B87C251E2D,0xFA000,0x31800)
```

Hello, Windows 10!

what next?


test on more hardware


more flexible policies
use encrypted commands


installation environment


noexitbootservices and
UEFI devices into mainline

Safely Booting Windows with Linux

Trammell Hudson

Lower Layer Labs

https://safeboot.dev/
https://github.com/osresearch/safeboot-loader/