Zero crypto attacks against preboot authentication passwords

Jonathan Brossard
CTO - P1 Code Security

jonathan@p1sec.com
endrazine@gmail.com

Telecomix Cyphernetics Assembly,
Goteborg, 17 of June 2010

Before we start...

- Thanks to the organizers :)
- Thank you for coming.
- I'm very happy to be here !

Agenda

Goals, contributions :

- Demonstrate the feasability of brute force attacks on preboot authentication passwords.
- Give a pessimist estimation of the cost of password cracking on full encryption software using a generic instrumentation methodology.
- Use this metric to adapt password length policy acording with the value of the protected assets.

Juridical environment

- Cryptographic software is mostly legalized in both North and South America and Europe.

- Wikipedia : « In China, a license is still required to use cryptography. Many countries have tight restrictions on the use of cryptography. Among the more restrictive are laws in Belarus, Kazakhstan, Mongolia, Pakistan, Russia, Singapore, Tunisia, and Vietnam. »

- Users of cryptographic software must give either a copy of their keys or plain text equivalent of any text asked by authorities in case of trial, or face prison sentences in most countries.

Crypto software
poor reviews
+ Governments interrests
+ global business
communications
+ terrorism blah blah

= high risk of (cryptographic ?)
backdoors
& **privacy threats**

Is such a thing credible?

- Quoting Wikipedia :

  « DES was designed to be resistant to differential cryptanalysis, a powerful and general cryptanalytic technique known to NSA and IBM, that became publicly known only when it was rediscovered in the late 1980s. According to Steven Levy, IBM rediscovered differential cryptanalysis, but kept the technique secret at NSA's request. The technique became publicly known only when Biham and Shamir re-rediscovered and announced it some years later. The entire affair illustrates the difficulty of determining what resources and knowledge an attacker might actually have. »

Technical motivations

- Even serious developpers don't test their crypto software enough, if at all (Debian SSL bug : ~32k keys).
- Vendors (in particular Truecypt) have adopted policies where they do not cover certain attacks (eg: Plain text password leakage as we presented at Defcon 0x16, or Joanna Rutowska's evilmaid attack) leaving the «official» attack surface left to : computer theft. Or simply put :

  - brute force,
  - brute force,

More globally

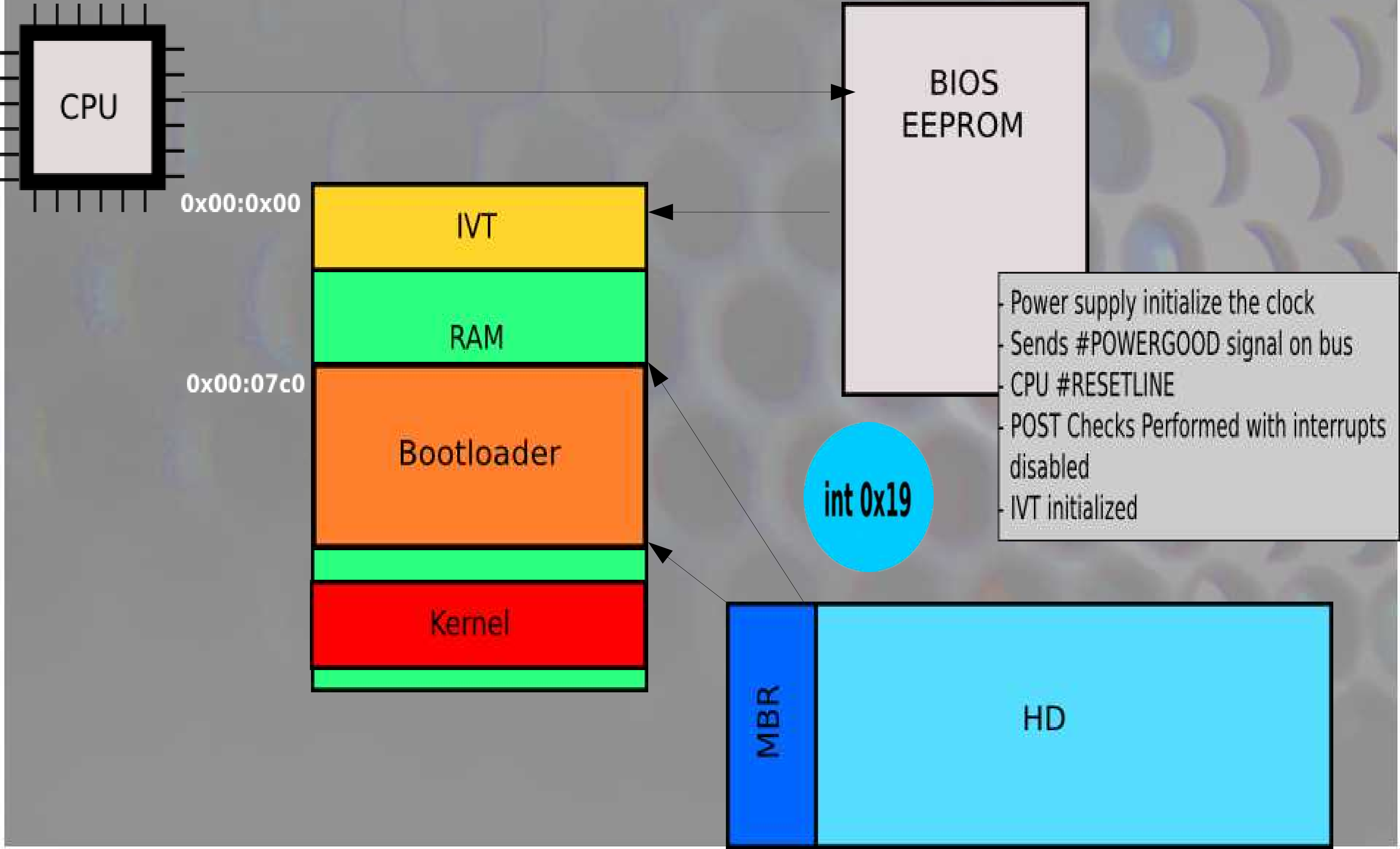
- Non tech people will say :

  « if it fails just go for bruteforce ».
- Sure.. but how do you do it ?

  I couldn't find a public tool myself. And then I started to wonder...

Keyboard internals

II-1) Boot sequence overview



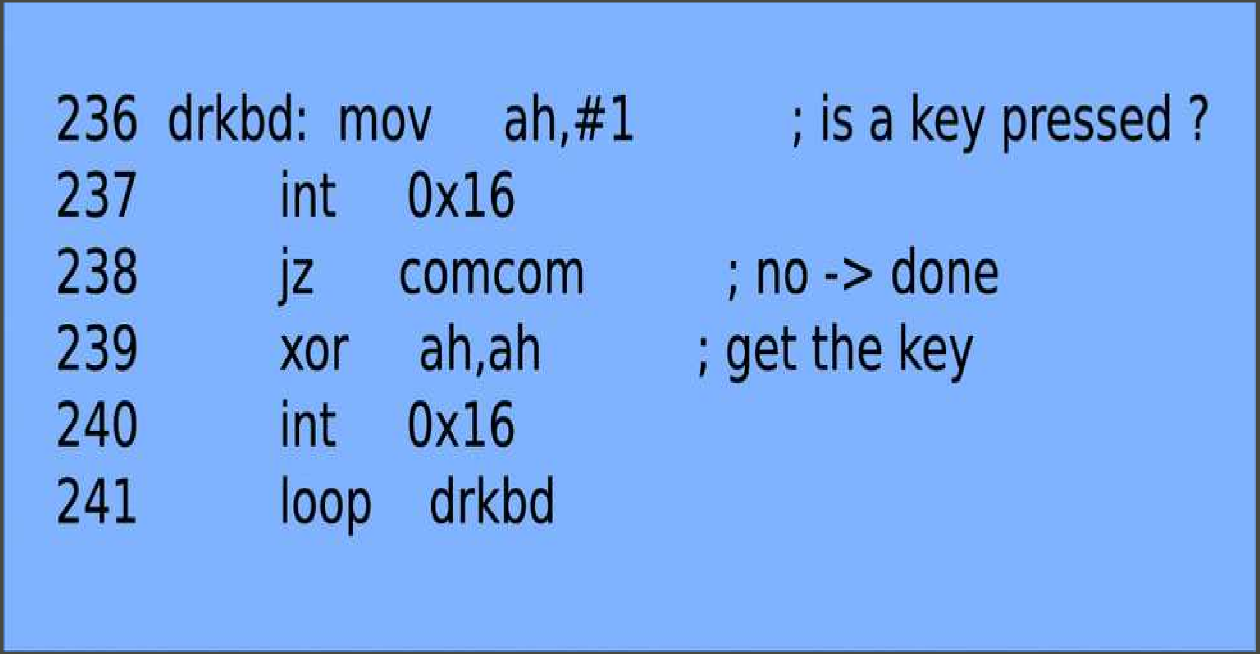
- Power supply initialize the clock
- Sends #POWERGOOD signal on bus
- CPU #RESETLINE
- POST Checks Performed with interrupts disabled
- IVT initialized

II-2) BIOS API for user inputs (1/2)

- Interruption 0x16 invoked via functions :
- ah=0x00 , “Get keystroke” : returns the keystroke scancode in AH and its ASCII code in AL.
- ah=0x01 , “Check for keystroke” : idem, but the Zero Flag is set if no keystroke is available in the Bios keyboard buffer.

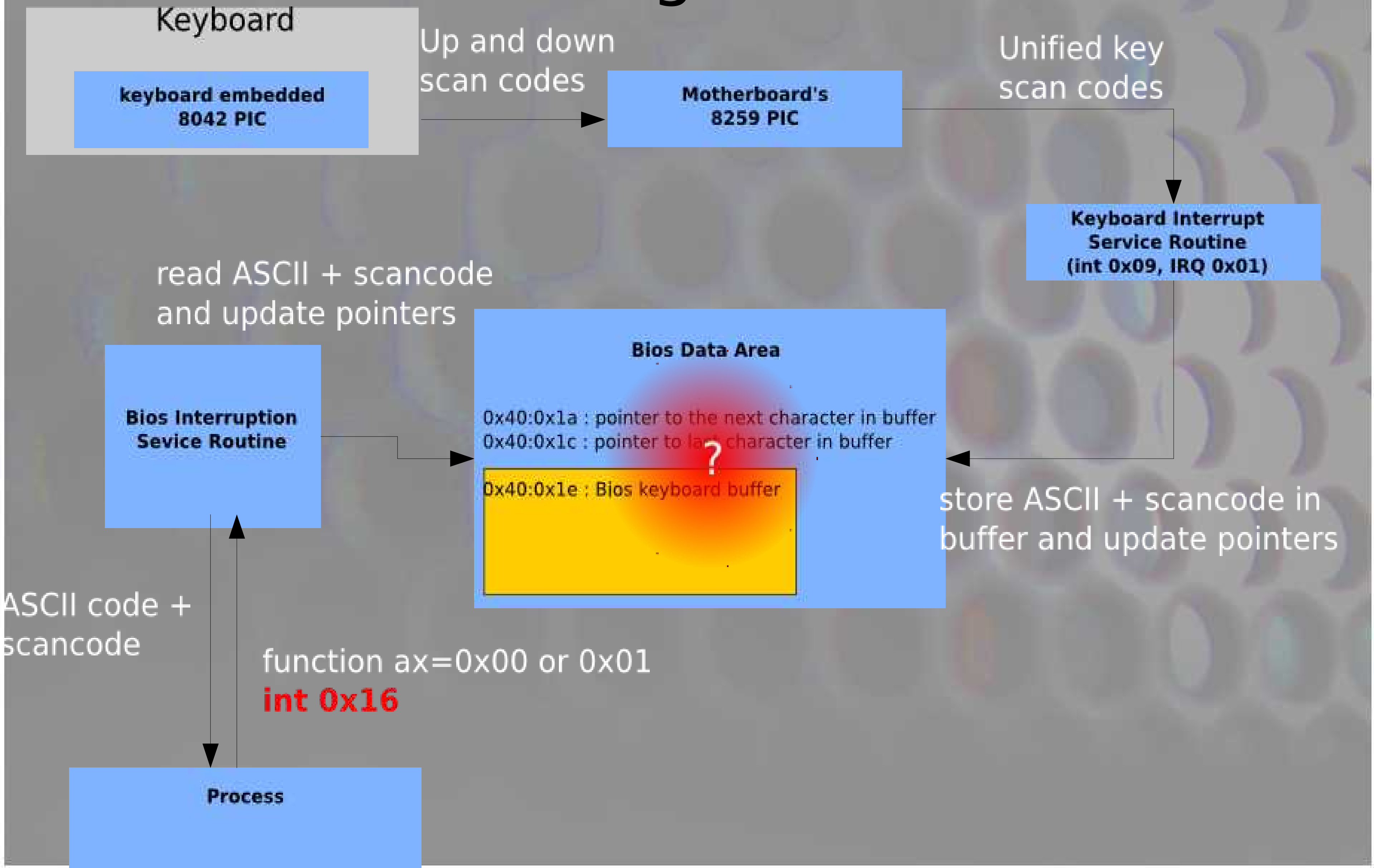
II-2) BIOS API for user inputs (2/2)

- eg : lilo password reading routine :



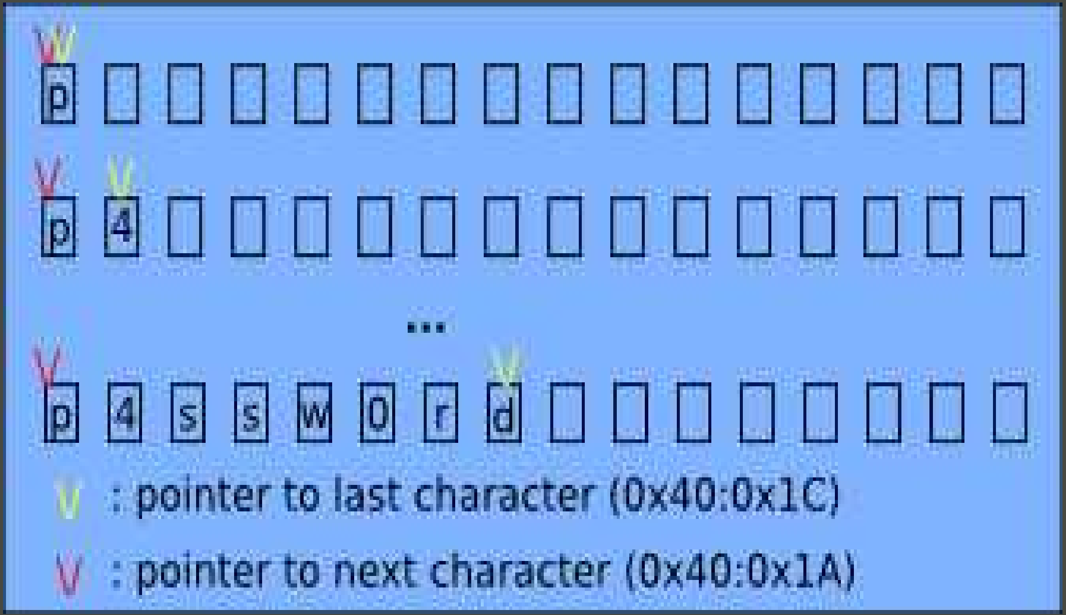
```
drkbd:  mov    ah,#1          ; is a key pressed ?
        int    0x16
        jz     comcom         ; no -> done
        xor    ah,ah          ; get the key
        int    0x16
        loop   drkbd
```

II-3) BIOS internals for keyboard management


Keyboard
keyboard embedded 8042 PIC
Up and down scan codes
Motherboard's 8259 PIC
Unified key scan codes
Keyboard Interrupt Service Routine (int 0x09, IRQ 0x01)
read ASCII + scancode and update pointers
Bios Data Area
0x40:0x1a : pointer to the next character in buffer
0x40:0x1c : pointer to last character in buffer
?
0x40:0x1e : Bios keyboard buffer
Bios Interruption Sevice Routine
store ASCII + scancode in buffer and update pointers
ASCII code + scancode
function ax=0x00 or 0x01
int 0x16
Process

II-4) BIOS keyboard buffer Remanance... (1/3)

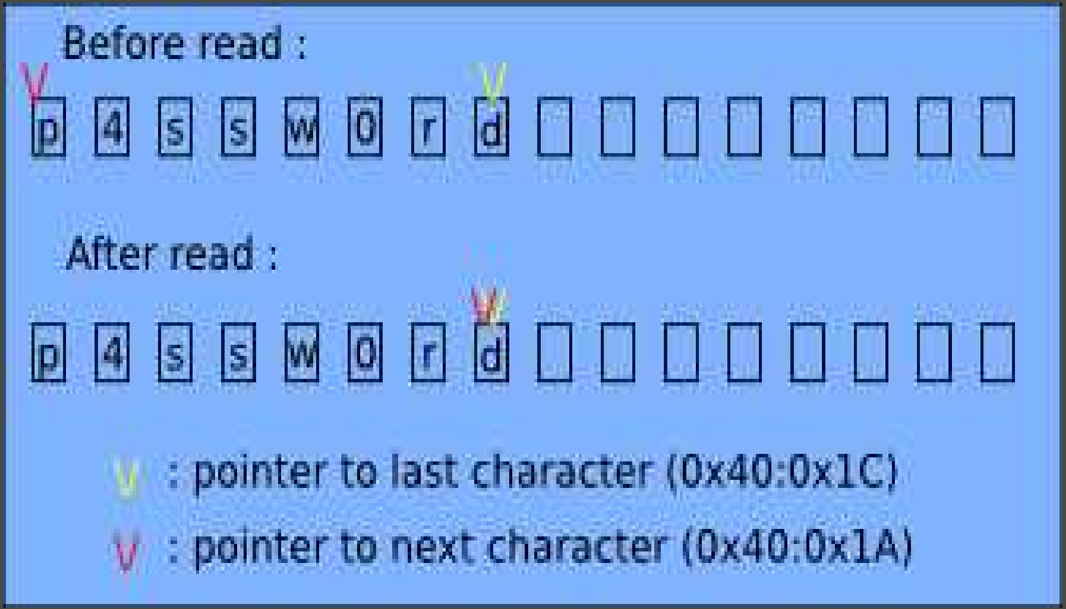
- Filling the BIOS keyboard buffer (with the keyboard) :



p
p 4
...
p 4 s s w 0 r d
V : pointer to last character (0x40:0x1C)
V : pointer to next character (0x40:0x1A)

II-4) BIOS keyboard buffer Remanence...

- Reading the BIOS keyboard buffer (using int 0x16, ah=0x00 or 0x01) :


Before read :
p 4 s s w 0 r d
After read :
p 4 s s w 0 r d
: pointer to last character (0x40:0x1C)
: pointer to next character (0x40:0x1A)

Demo

Simulating keystrokes by PIC programming (from real mode)

Demo

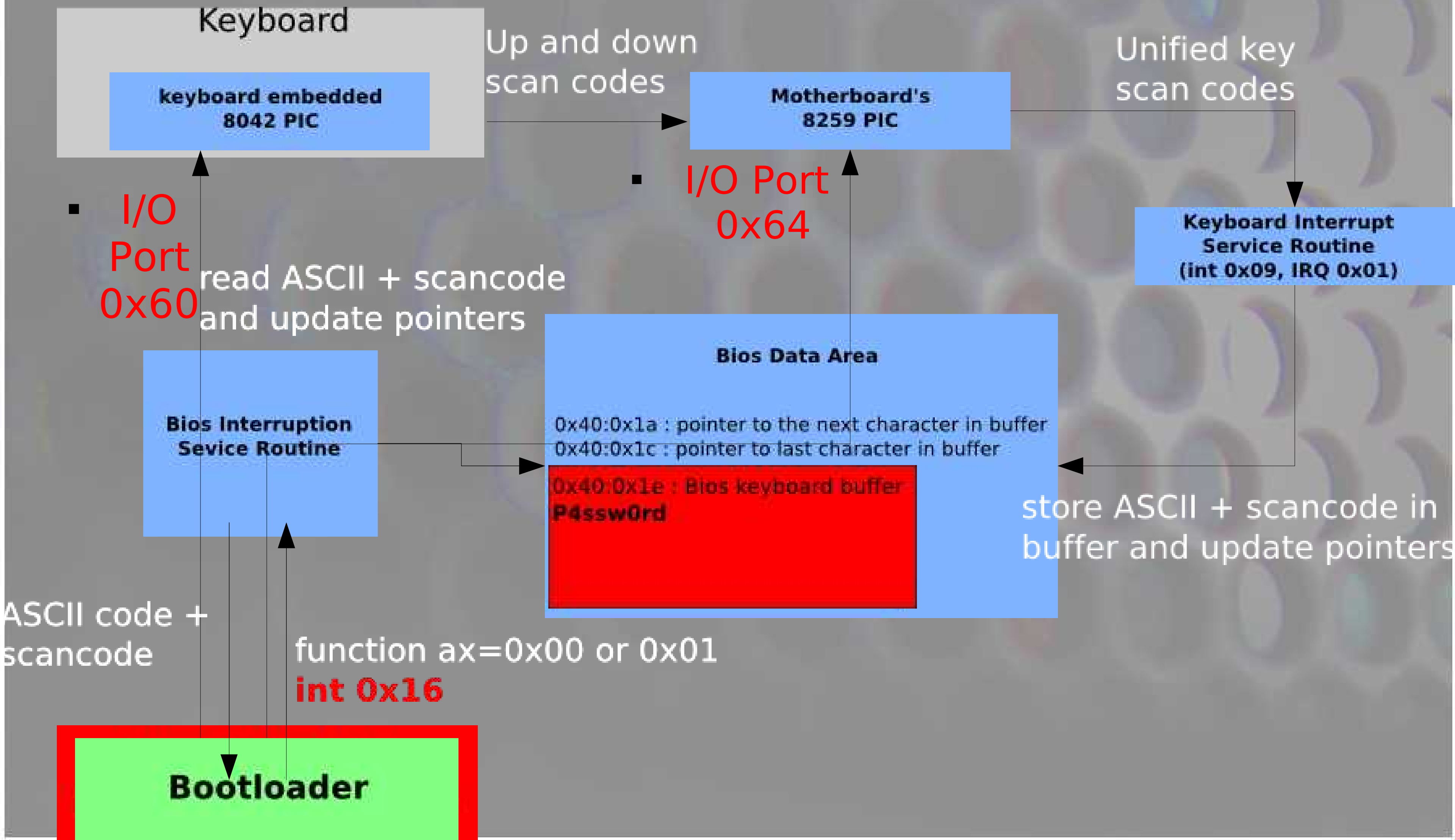
Simulating keystrokes by PIC programming (from protected mode under x86 GNU/Linux)

(aka: brute force any GUI)

Exemple of application :

Rebooting a computer protected with a password (assuming you know that password - for now ;), by simulating keystrokes at boot time...

Attack scenario :


Keyboard
keyboard embedded 8042 PIC
Up and down scan codes
Motherboard's 8259 PIC
Unified key scan codes
I/O Port 0x60
I/O Port 0x64
Keyboard Interrupt Service Routine (int 0x09, IRQ 0x01)
read ASCII + scancode and update pointers
Bios Data Area
0x40:0x1a : pointer to the next character in buffer
0x40:0x1c : pointer to last character in buffer
0x40:0x1e : Bios keyboard buffer
P4ssw0rd
Bios Interruption Sevice Routine
store ASCII + scancode in buffer and update pointers
ASCII code + scancode
function ax=0x00 or 0x01
int 0x16
Bootloader

Notes :

- You can get the code for this attack from the Defcon archive (the attack is called « Invisible Man »).

- For our cracking purpose, writing directly to 0x41e is way more efficient (but that was cool, right ? ;)

Demo
Retreiving passwords from physical memory from userland without privileges (up to Vista SP0)

Notes

- Bitlocker's fix in Vista SP1 (replacing any character by ' ') still leaks the password length.
- This plain text password leakage vulneability is still present on many software including Lilo and Grub if you can read from arbitrary physical memory locations (typically needs root privileges).

Brute forcer design

Challenges

- Installation & initial control flow modification (BIOS Firmware, other media, MBR replacing/patching)
- Maintaining control (BP, IVT hijack, runtime patching)

Design decisions

- We want something as generic as possible, so we will avoid application specific breakpoints etc.
- The media we boot from is irrelevant (usb/cdrom/floopy..)
- Keeping control over the control flow is a bit tricky.
- Very similar to MBR virus writting (old school !! ;)

Interrupts hijacking

- Int 0x13 : we need to proxy calls to the original int 0x13, changing disk number (dl). It also allows to detect successfull decryption
- Int 0x16 : simulate keystrokes
- Int 0x10 : for performance (we don't need display)

Full attack scenario

- Boot from our code (1 sector)
- Allocate BIOS memory
- Copy the rest of our code there
- Patch the IVT (int 0x16, int 0x10, 0x13)
- Emulate int 0x19 (copy code from original MBR to 0x00:0x7c00, jump there)

```
jonathan@blackbox:~/h2hc$
cat BF-OS.asm |grep -v "^;"|
grep [a-Z0-9]|wc -l
902
jonathan@blackbox:~/h2hc$
```

Demo
Bruteforcing Lilo

Demo

Bruteforcing Grub

with **MD5** hash

Demo
Bruteforcing full disk encryption with **TrueCrypt 6.3**

Experimental results

Result #1

It's doable :)

Result #2

The cost of hashing algorithms (MD5..) is negligible in the cracking process

Result #3 : performance

Hashing algorithms : we tried 705 passwords in 30s.

Truecrypt : 10s / password (whow !)

Metrics
(assuming a hashing algo is used)

Time taken to crack

Irrelevant
(cloud computing !)

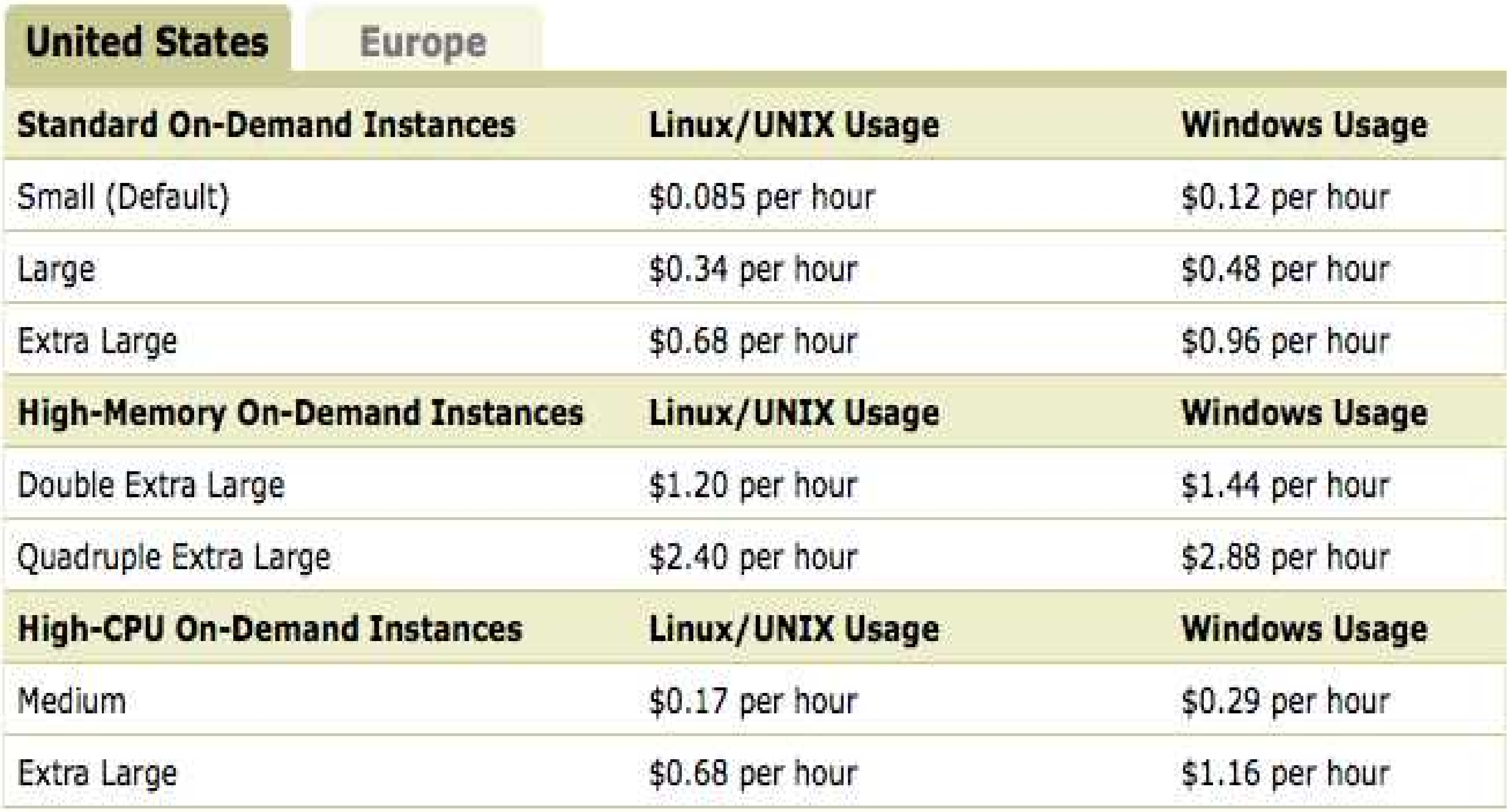
Search space

S = sigma (i=1,length) sizeof(charset)^i

Cost

C = O (S * 3/70 * cpu_freq/(1.6GHz) * cost_per_hour)

Amazon EC2



United States | Europe

Standard On-Demand Instances	Linux/UNIX Usage	Windows Usage
Small (Default)	$0.085 per hour	$0.12 per hour
Large	$0.34 per hour	$0.48 per hour
Extra Large	$0.68 per hour	$0.96 per hour
High-Memory On-Demand Instances	**Linux/UNIX Usage**	**Windows Usage**
Double Extra Large	$1.20 per hour	$1.44 per hour
Quadruple Extra Large	$2.40 per hour	$2.88 per hour
High-CPU On-Demand Instances	**Linux/UNIX Usage**	**Windows Usage**
Medium	$0.17 per hour	$0.29 per hour
Extra Large	$0.68 per hour	$1.16 per hour

Cost

C ~ 3/70 * 0.085 * sigma (i=1,length) (sizeof(charset)^length)

Cost

Exemple :
charset = [a-z]
Pass length = 5

Cost ~ $45 000

Cost

Exemple :
charset = [a-z]
Pass length = 8

Cost ~ $800 000 000

Cost

Exemple :
charset = [a-zA-Z0-9]
Pass length = 8

Cost ~ $800 000 000 000

Conclusions (1/2)

- Bruteforcing is physically doable for both hashing algorithms and complex symetric systems.

- Bruteforcing remains unpractical against Truecrypt so far (6 passwords / minute, recommended pass phrases of length 20).

- This methodology, while generic, is too costly to be practical against strong passwords (unless you're .gov ?).

Conclusions (2/2)

- Not using TPM like technologies allows attackers to take advantage of distributed computing, making the brute force time irrelevant.

A few more things on TrueCrypt 6.3

Truecrypt's policy and assumed attack surface

- No TPM support. Won't happen.
- No support against root or physical attacks (bootkits, trojaning ...)
- Regarding full disk encryption (the real thing why TC is great) : no keyfiles support as of version 6.3.

No TPM means

- No hardware sealing.
- We can modify the bootloader.
- We can scale on hardware/virtualisation.

Key/pass repudiation

- Setting a new key/passphrase pair is not enough : one needs to fully decrypt the drive, and then fully re encrypt it.
- Old key/pass pair would still be valid otherwise.

Forensics : HD dump vs. Rescue iso image

- They contain exactly the same crypto information (salt+keys : only password is missing).
- We can very well brute force from a Rescue cdrom image (easier to clone/steal than a whole HD).
- This is not intuitive : social engineering risk increased.

Demo
Reversing the Truecrypt Rescue disk

Thank you for coming :)

Questions ?