Bypassing pre-boot authentication passwords

by instrumenting the BIOS keyboard buffer
(practical low level attacks against x86 pre-boot authentication software)

Jonathan Brossard - iViZ Technosolutions Research Team

jonathan@ivizindia.com
endrazine@gmail.com

DEFCON 16

Who am I ?

Scope of this presentation

- We present a new class of vulnerabilities,
- Affecting multiple pre-boot authentication software under x86 and x64 architectures,
- Exploitable without physical access.

Limitations : we will focus on password based authentication solely.

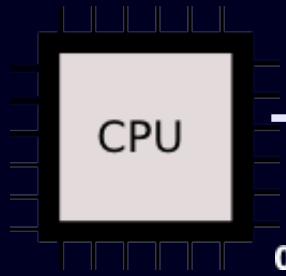
Contents

- (Technically) defining pre-boot authentication
- Password leakage under Windows
- Password leakage under *nix
- Rebooting in spite of a pre-boot authentication
- Examples of vulnerable software
- Mitigating those vulnerabilities

I - (Technically) defining pre-boot authentication

- Boot sequence overview
- Taxonomy of pre-boot authentication software
- BIOS API for user inputs
- BIOS internals for keyboard management
- BIOS keyboard buffer Remanence...
- Verifying this bug exists “in real life”
- Password chaining

I-1) Boot sequence overview



BIOS
EEPROM

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

0x00:0x00

0x00:07c0

IVT

RAM

Bootloader

Kernel

int 0x19

MBR

HD

I-2) Taxonomy of pre-boot authentication softwares

- Bios Passwords
- Bootloader Passwords (Vista's Bitlocker, Grub or Lilo, and most others pre-boot authentication software : Truecrypt, Diskcryptor...)
- Early kernel stage passwords – typically before decompression (eg: suspend2 hibernation patch for GNU/Linux)

I-3) 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.

I-3) BIOS API for user inputs (2/2)

- eg : lilo password reading routine :

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

I-4) 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 Interruption Sevice Routine

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

store ASCII + scancode in buffer and update pointers

ASCII code + scancode

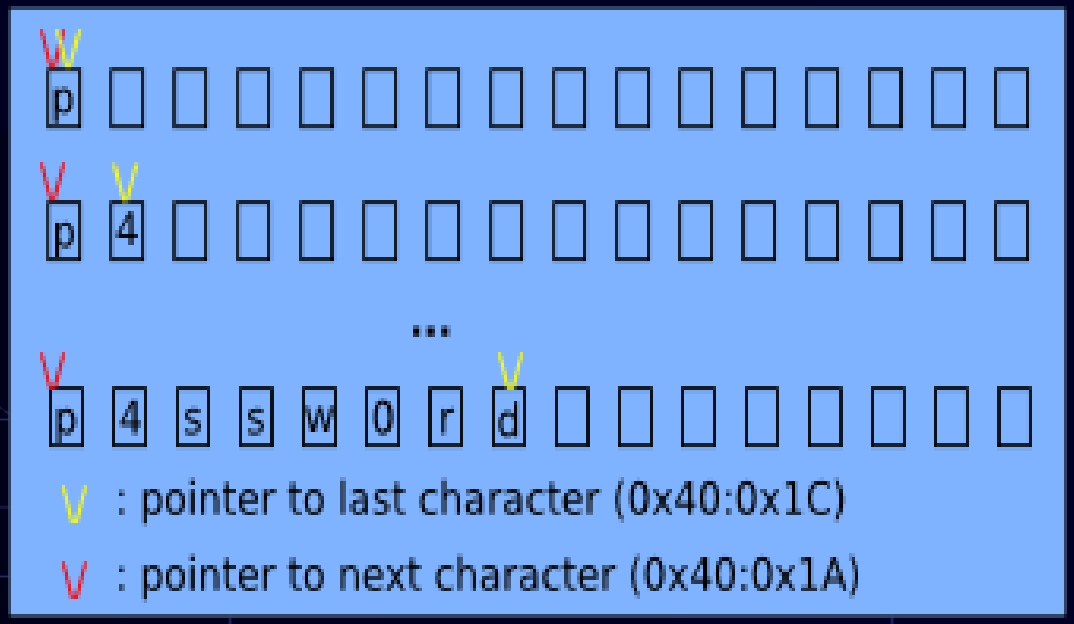
function ax=0x00 or 0x01

int 0x16

Process

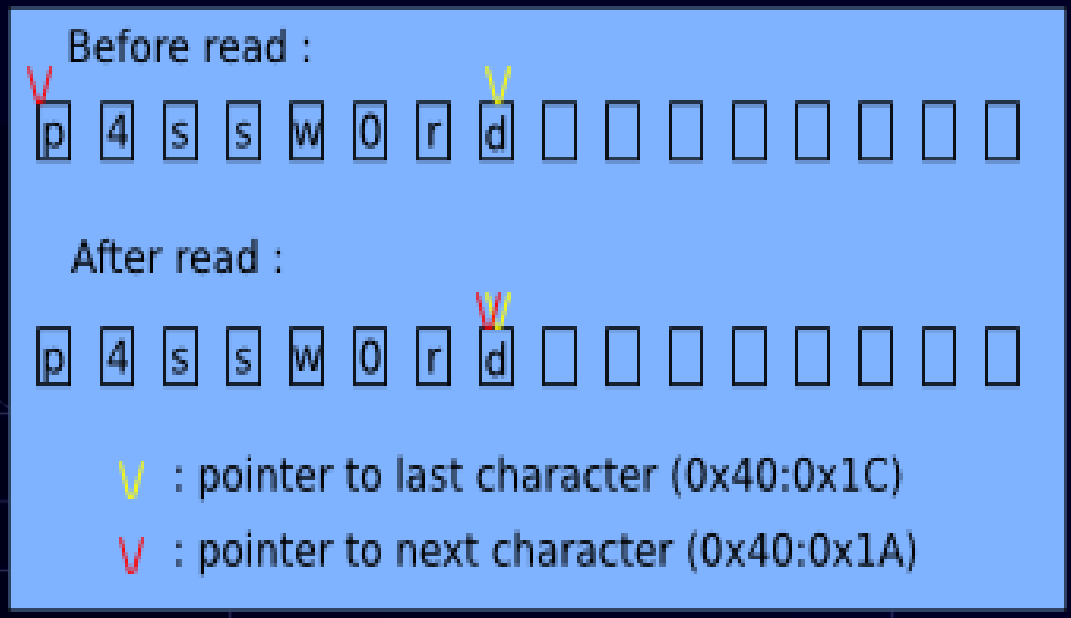
I-5) BIOS keyboard buffer Remanance... (1/3)

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

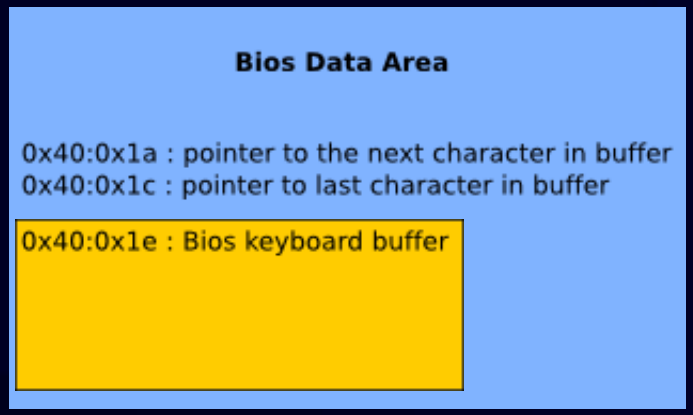


I-5) BIOS keyboard buffer Remanence... (2/3)

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



I-5) BIOS keyboard buffer Remanence... (3/3)



- Who is supposed to clear the keyboard buffer ?

I-6) Verifying this bug exists “in real life” (1/2) :

- We want to check the authentication routines in the BIOS themselves (aka: BIOS Passwords)

- We will write a small USB-bootable OS in 16b asm to read the content of the BIOS keyboard buffer in Real Mode (sploitOS.S)

I-6) Verifying this bug exists “in real life” (2/2) :

- **Results :**
- Most BIOS Passwords are vulnerable (more on this later).
- ... if the BIOS Programmers themselves do not clear the BIOS keyboard buffer... just imagine third party programmers...

I-7) Password chaining :

- Let's now imagine we have two authentication devices in a raw (asking for pass1 and pass2 respectively)....
- What happens in the BIOS keyboard buffer ?
- The passwords are concatenated ! So we can retrieve both ;)
  - [p][a][s][s][1][Enter][p][a][s][s][2][Enter]

SCOPE :

In the following two sections, we assume the OS has fully booted and the attacker is given a local shell, but no physical access.

II - Password leakage under Windows

- The Challenge
- Possible attack scenarii
- Reading the password from a guest account

II-1) The Challenge :

How to read the password at 0x40:0x1e ?
(once in protected mode...)

II-2) Possible attack scenarii :

- Get back to real-mode
- Switch to SMM
- Get it from kernel land

- All those scenari require very high privileges :(

II-3) Reading the password from a guest account :

- **The MS-DOS emulation mode :**
- built on top of x86 Vmode to emulate 16b execution
- Windows “feature” : maps physical memory ranges 0-FFF and C0000-FFFFF into userland !!! (http://readlist.com/lists/securityfocus.com/bugtraq/1/9422.html)


CPU 0, ASSAULT STATE
CPU 1, ASSAULT STATE
CPU 2, ASSAULT STATE
DEFCON 5 4 3 2 1 • 00:53.52
launch detected
launch detected
launch
san francisco hit, 1.4m dead
los angeles hit, 3.2m dead
launch
launches detected
launches detected
launch detected
launch detected
launch detected
Demo
units
alliances
comms
scores
radar
population
orders

III – Password leakage under *nix

- Challenge
- Getting the password from user land
- Getting the password from kernel land
- Conclusion

III-1) Challenge :

- Unfortunatly, no goodie like the RAM leakage under Windows... We will try to retrieve the password from a privileged (typically root) account...

III-2) Getting the password from user land (1/4):

- We know the address of the BIOS keyboard buffer in Physical Memory.
- under most flavors of Unix, /dev/mem contains a mapping of the Physical memory...

```
root@blackbox:~# xxd -l 32 -s 0x041e  /dev/mem
000041e: 7019 3405 731f 731f 7711 300b 7213 6420  p.4.s.s.w.0.r.d
000042e: 0d1c 0d1c 0000 0000 0000 0000 0000 0000  ................
root@blackbox:~#
```

III-2) Getting the password from user land (2/4):

- /dev/kmem contains a mapping of kernel memory :
- /dev/kcore contains the same information in the f

```
root@blackbox:~# dd if=/dev/kmem ibs=1 skip=3221226526 count=32 2>/dev/null|xxd
0000000: 7019 3405 731f 731f 7711 300b 7213 6420  p.4.s.s.w.0.r.d
0000010: 0d1c 0d1c 0000 0000 0000 0000 0000 0000  ................

root@blackbox:~# xxd -l 32 -s 0x141e  /proc/kcore
000141e: 7019 3405 731f 731f 7711 300b 7213 6420  p.4.s.s.w.0.r.d
000142e: 0d1c 0d1c 0000 0000 0000 0000 0000 0000  ................
root@blackbox:~#
```

III-2) Getting the password from user land (3/4):

- We have coded a simple tool that will work under virtually any x86 based *nix (tested under OpenSolaris, FreeBSD, OpenBSD and GNU/Linux) to read the possible passwords from /dev/mem, but also /dev/kmem, /dev/kcore etc if available...

III-2) Getting the password from user land (4/4):

```
root@blackbox:/home/jonathan/userland-unix# ./generic.unix.sploit -m

 [ Bios keyboard buffer hysteresis generic userland exploit for *nix. ]
 // Jonathan Brossard - jonathan@ivizindia.com - endrazine@gmail.com

 Tested under several flavours of GNU/Linux, *BSD and Solaris.

--[ Password (to the latest pre boot authentication software) : p4ssw0rd

root@blackbox:/home/jonathan/userland-unix#
```

III-3) Getting the password from kernel land (1/3):

- The BIOS Data Area is copied to a “safe” zone during kernel early booting (the infamous “Zero Page”, cf: Setup.S in the Linux kernel).
- If you assume a 3Gb/1Gb kernel split, the address of the BIOS Keyboard buffer is : 0xC000041e

III-3) Getting the password from kernel land (2/3):

- Verifying that the password is located at 0xC000041e (using remote kernel debugging...)

```
root@blackbox:/home/jonathan# cd /usr/src/linux-2.6.19/
root@blackbox:/usr/src/linux-2.6.19# gdb ./vmlinux
GNU gdb 6.6-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
(...)
gdb $ target remote 127.0.0.1:8832
[New Thread 1]
(...)
0xc0103db0 in apic_timer_interrupt ()
gdb $ x /1s 0xC000041E
0xc000041e:     "p\0314\005s\037s\037w\0210\vr\023d"
gdb $
```

III-3) Getting the password from kernel land (3/3):

- We have coded a simple LKM to automate the work and display the possible passwords in a new entry under the /proc pseudo-filesystem :

```
root@blackbox:/home/jonathan/ksploit-proc/src# insmod ./ksploit.ko
root@blackbox:/home/jonathan/ksploit-proc/src# cat /proc/prebootpassword
Password to the latest pre boot authentication software) : p4ssw0rd
root@blackbox:/home/jonathan/ksploit-proc/src#
```

III-4) Conclusion :

- This bug has been there since the very beginning of BIOS passwords (25+ years).
- Retrieving the password is as simple as reading a file at a given location... Open your eyes ;)


CPU 0, ASSAULT STATE
CPU 1, ASSAULT STATE
CPU 2, ASSAULT STATE
DEFCON 5 4 3 2 1 • 00:53.52
launch detected
launch detected
launch
san francisco hit, 1.4m dead
los angeles hit, 3.2m dead
launch
launches detected
launches detected
launch detected
launch detected
launch detected
Demo
units
alliances
comms
scores
radar
population
orders

IV – Rebooting in spite of a pre-boot authentication password

- In some cases, it is handy for an attacker to reboot the computer (to boot a weaker kernel for instance). But if a pre-boot authentication device is on the way, this is a non trivial taks...
- In the next section, we assume the attacker can write to the MBR (ie: typically root access) and is willing to reboot the computer.

IV – Rebooting in spite of a pre-boot authentication password

- **Agenda :**
- The password is not used to decrypt anything
- The password is used to decipher part of the disk or the whole disk.

IV-1) Rebooting in spite of a preboot authentication password without disk encryption (1/2):

- Since the password checking routine doesn't perform any useful task (from an attacker point of view), he can simply patch it.
- See phrack article “Hacking deeper in the system” by Scythale for a deeper analysis of Grub hacking).

IV-1) Rebooting in spite of a preboot authentication password without disk encryption (2/2):

```
                    [ LILOL ]

UP  Gentoo-2.6.24
UP  Gentoo-2.6.19

 enhanced lilo for extra fun

 Kiss your pre-boot password goodbye...

boot:
Loading Gentoo-2.6.24.....................................................
.........................................._
```

IV-2) Rebooting with a password used for disk decryption :

- The BIOS keyboard buffer “feature” reloaded
- Attack scenario
- Methodology to install the rogue bootloader
- “Invisible Man” roadmap

IV-2-a) The BIOS keyboard buffer “feature” reloaded :

- **The Problem :**
- What happens if the BIOS keyboard buffer is not initialized ?
- If the attacker can somehow enter the password before the genuine bootloader prompts for a password, the authentication routine will decrypt the disk nicely ;)

IV-2-b) 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 Interruption Service Routine

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

store ASCII + scancode in buffer and update pointers

ASCII code + scancode

function ax=0x00 or 0x01
int 0x16

Bootloader

IV-2-c) Methodology to install the rogue bootloader :

1) Open the device in read/write mode.
2) Search for a 512b buffer to store a backup of the MBR.
3) Copy the first sector of disk to the backup buffer.
4) Find the initial jump to MBR's code.
5) Write our own payload to that address, preserving the partition table and the final 0xaa signature marking the disk as bootable.

IV-2-d) “Invisible Man” roadmap :

1) Use a delta offset trick to find our own location in memory.
2) Fill the Bios keyboard buffer using PIC 8048 and PIC 8259 programming.
3) Allocate a 10Ko buffer in the free RAM reserved to the BIOS.
4) Find the first bootable disk by checking if it is marked as bootable.
5) Read the first 20 sectors of disk in reserved free RAM.
6) Patch the disk with the backed up MBR.
7) Jump to our own code copied in RAM.
8) Load the old MBR in Ram at address 0x0000:0x7c00 .
9) Unallocate the reserved Bios memory if possible.
10) Jump to original bootloader's entry point at 0x0000:0x7c00 .


CPU 0, ASSAULT STATE
CPU 1, ASSAULT STATE
CPU 2, ASSAULT STATE
DEFCON 5 4 3 2 1 • 00:53.52
launch detected
launch detected
launch
san francisco hit, 1.4m dead
los angeles hit, 3.2m dead
launch
launches detected
launches detected
launch detected
launch detected
launch detected
Demo
units
alliances
comms
scores
radar
population
orders

V – Examples of vulnerable softwares...

V-1) Vulnerable Softwares (1/3):

- **BIOS passwords :**
- Award BIOS Modular 4.50pg
- Insyde BIOS V190
- Intel Corp PE94510M.86A.0050.2007.0710.1559
- Hewlett-Packard 68DTT Ver. F.0D (11/22/2005)
- IBM Lenovo 7CETB5WW v2.05 (10/13/2006)

V-1) Vulnerable Softwares (2/3):

- **Full disk encryption with pre-boot authentication capabilities :**
- Bitlocker with TPM chip under Microsoft Vista Ultimate Edition SP0.
- Truecrypt 5.0 for Windows (open source)
- DiskCryptor 0.2.6 for Windows (open source)
- Secu Star DriveCrypt Plus Pack v3.9

V-1) Vulnerable Softwares (3/3):

- **Boot loader passwords :**
- grub (GNU GRUB 0.97) (latest CVS)
- lilo version 22.6.1 (current under Mandriva 2006)

V-2) Non vulnerable Softwares (1/2):

- **BIOS Passwords :**
- Hewlett-Packard F.20 (04/15/2005)
- Hewlett-Packard F.05 (08/14/2006)
- Pheonix BIOS Version F.0B, 7/3/2006
- Phoenix Technologies LTD R0220Q0 (25-05-2007)

V-2) Non vulnerable Softwares (2/2):

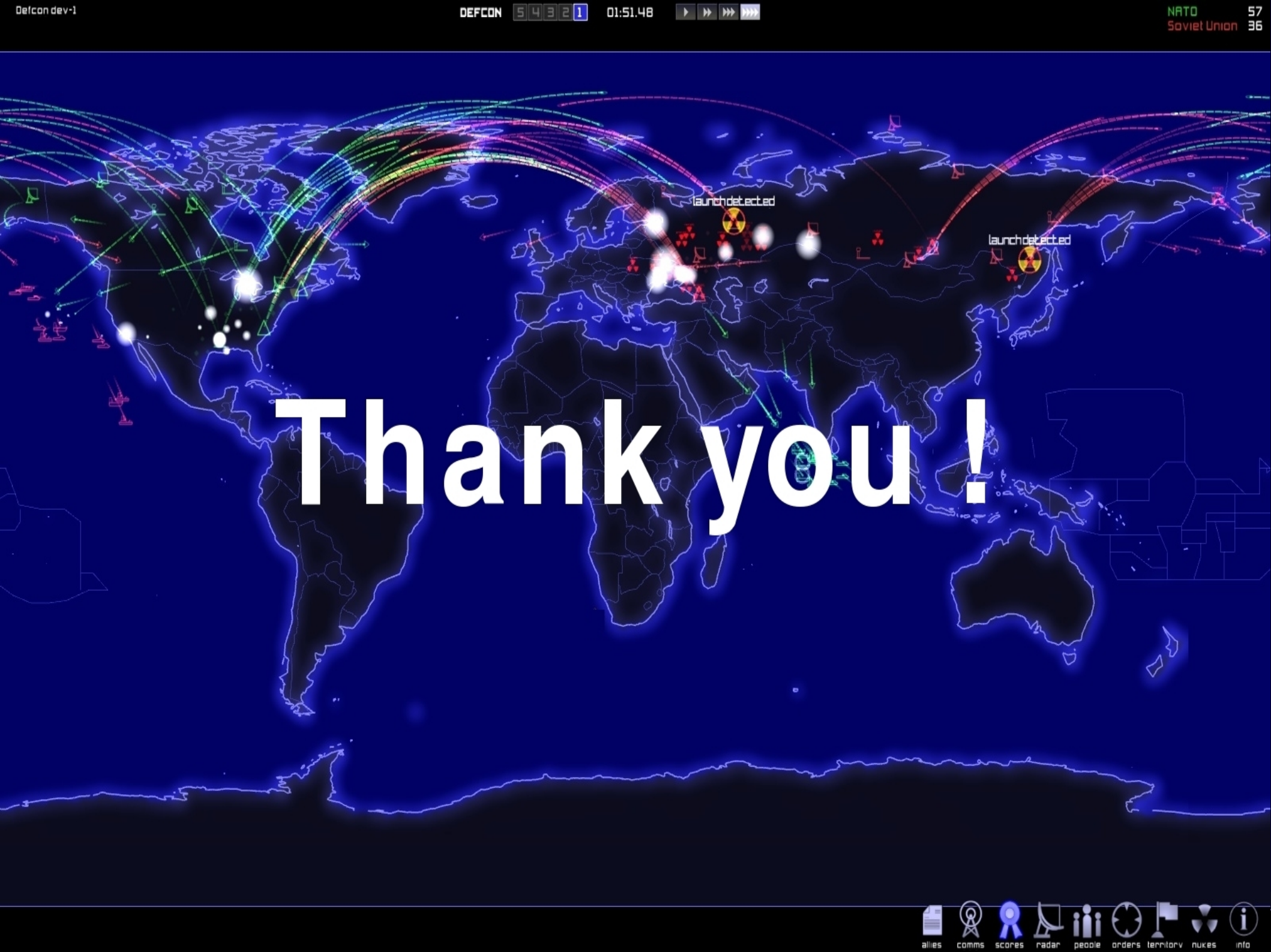
- **Full disk encryption with pre-boot authentication capabilities :**
- SafeGuard 4.40 for Windows
- PGP Desktop Professional 9.8 for Windows (Trial Version)

VI) Mitigating those vulnerabilities :

- Write correct software : sanitize the BIOS keyboard buffer (and more generally any password buffer) before and after use...
- We keep a list of patches on our website : http://www.ivizindia.com/BIOS-patches/ (contributions are most welcome).
- For GNU/Linux users, the latest version of Grsecurity (http://www.openwall.net) sanitizes the BDA at boot time (thanks to Brad for this).

Greetings :

- My uber elite reviewers (you know who you are) : many thanks guys :)
- The iViZ Technical Team for your support and the time spent on testing software.
- http://www.everybody-dies.com/ web site for letting me use the screenshots of their game “Defcon : everybody dies !” in my slides ;)
- irc.pulltheplug.org and irc.blacksecurity.org...
- All of you for coming to this presentation.
- The Defcon Staff for the awesome event and parties...


Defcon dev-1
DEFCON 5 4 3 2 1
01:51.48
NATO 57
Soviet Union 36
launch detected
launch detected
Thank you !
allies
comms
scores
radar
people
orders
territory
nukes
info


DEFCON 5 4 3 2 1
00:23.44
ScoreMode Default
[CPU]Player 2 0
[CPU]Player 1 0
[CPU]Player 0 0
Spectators
Chris
Questions ?
comms scores radar people orders territory nukes info