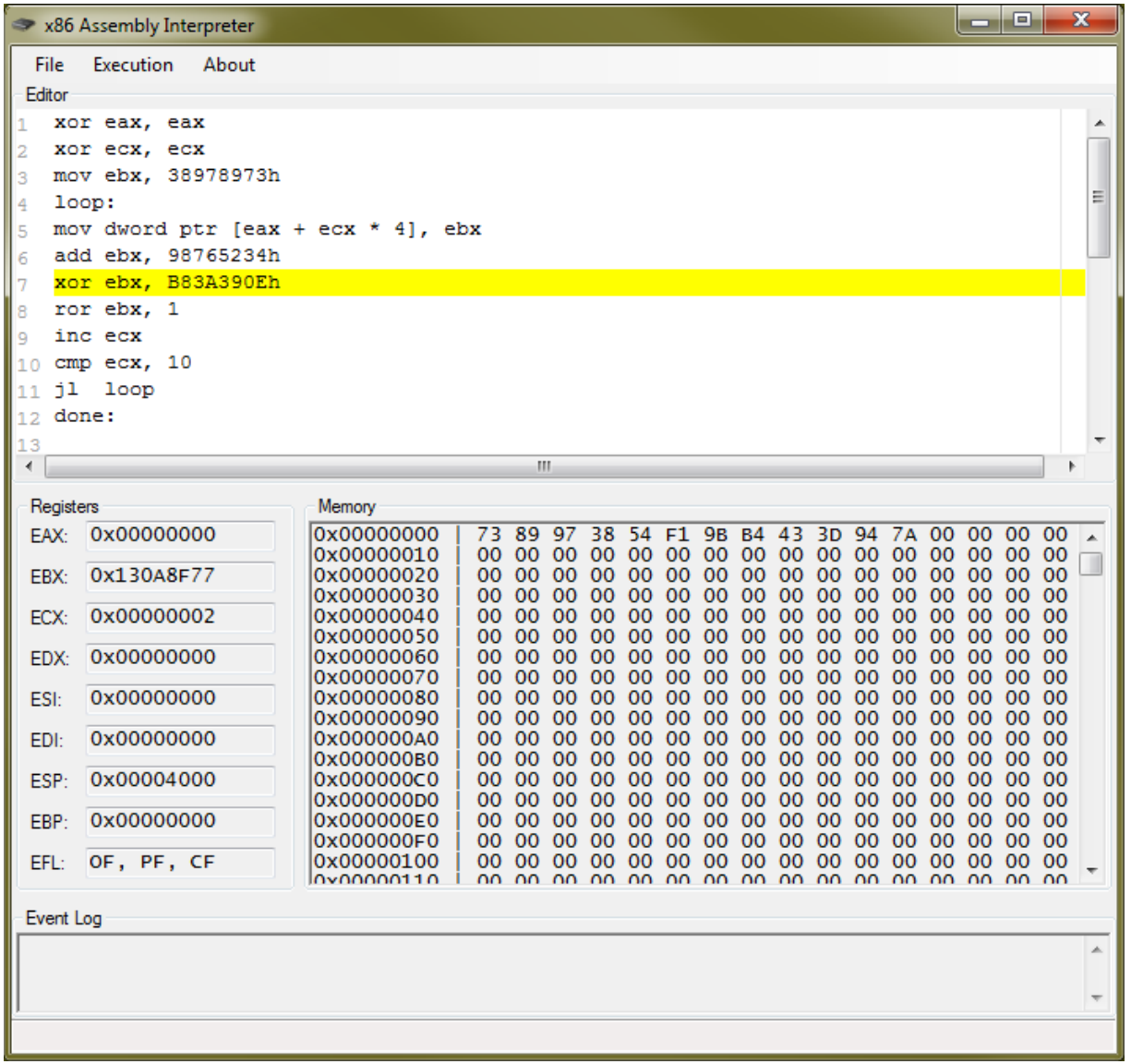
Introduction à la rétro-ingénérie (re)



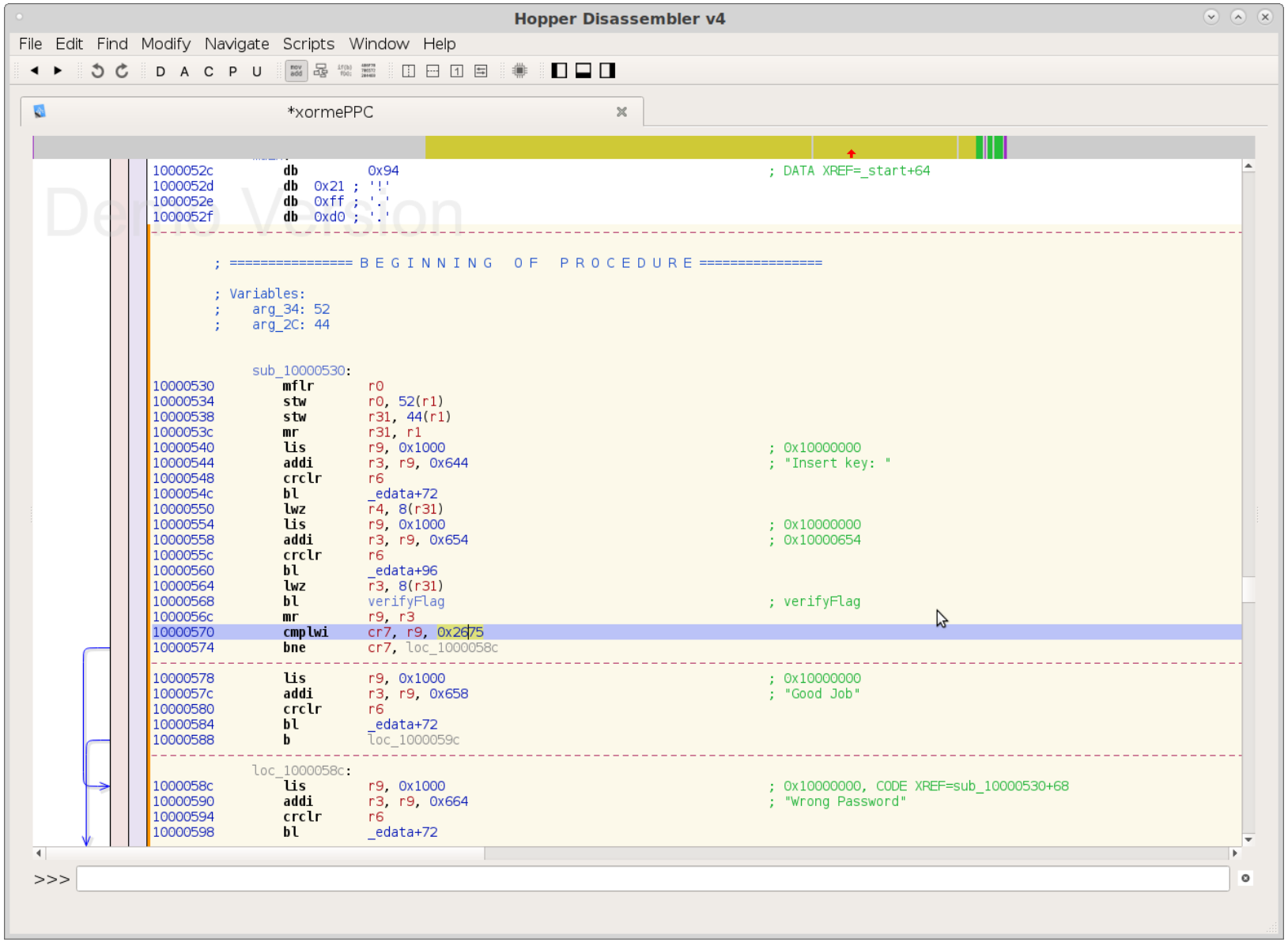
Ian-Kyle Wagner

Table de contenu

- Décompilation “high level” (Java,.Net,Android)
- Décompilation “Middle level” (C,C++)
- Intro à l’Assembleur x86 et ARM
- Décompilation “Low level” (ASM)
- Outils et techniques pour chaque niveau

Qu’est-ce que la rétro-ingénérie

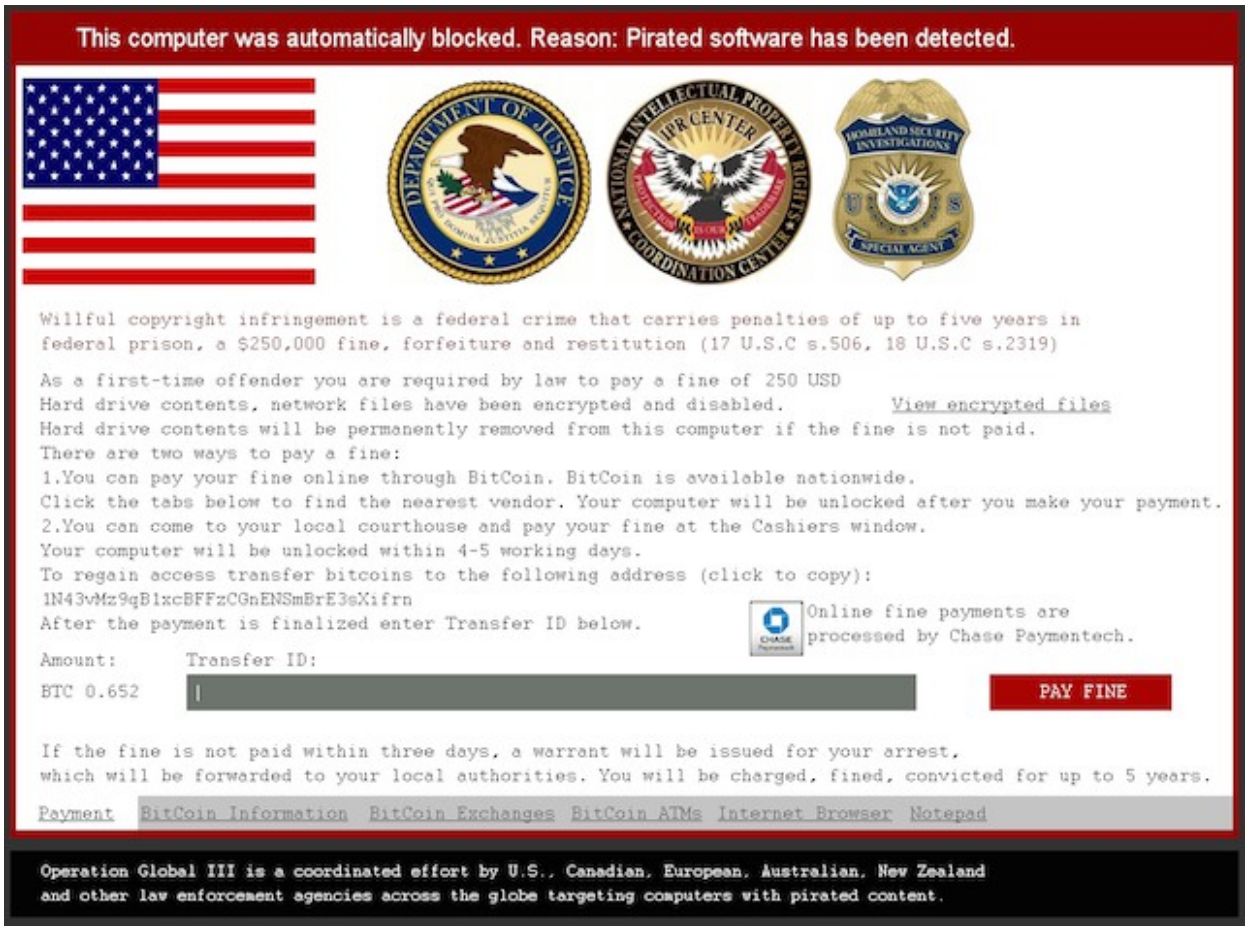
- Analyser le code source à partir d’un binaire



Pourquoi Re

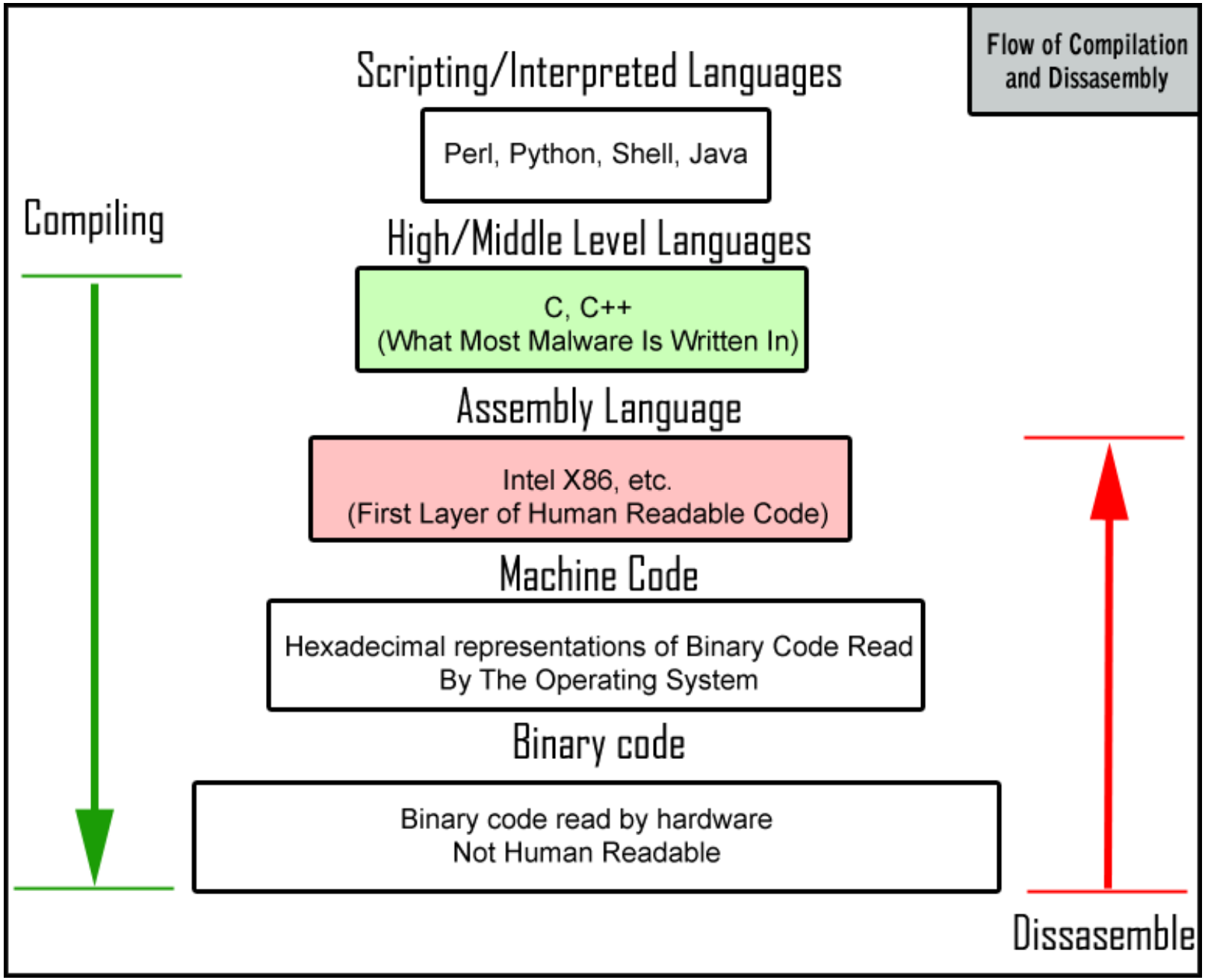
https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fwww.framboise314.fr%2Fwp-content%2Fuploads%2F2016%2F02%2Fsamba_linux_windoq.jpg&f=1

https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fwww.redeszone.net%2Fapp%2Fuploads%2F2014%2F12%2Fvirlock_foto.png&f=1



https://vignette.wikia.nocookie.net/logopedia/images/e/e1/Intel_Itanium_Logo_alt.svg-1--0.png/revision/latest?cb=20170131230417

Les différents niveaux



https://images.duckduckgo.com/iu/?u=https%3A%2F%2Fblog.malwarebytes.org%2Fwp-content%2Fuploads%2F2012%2F09%2FFlowDiagram2.png&f=1

Décompilation haut niveau

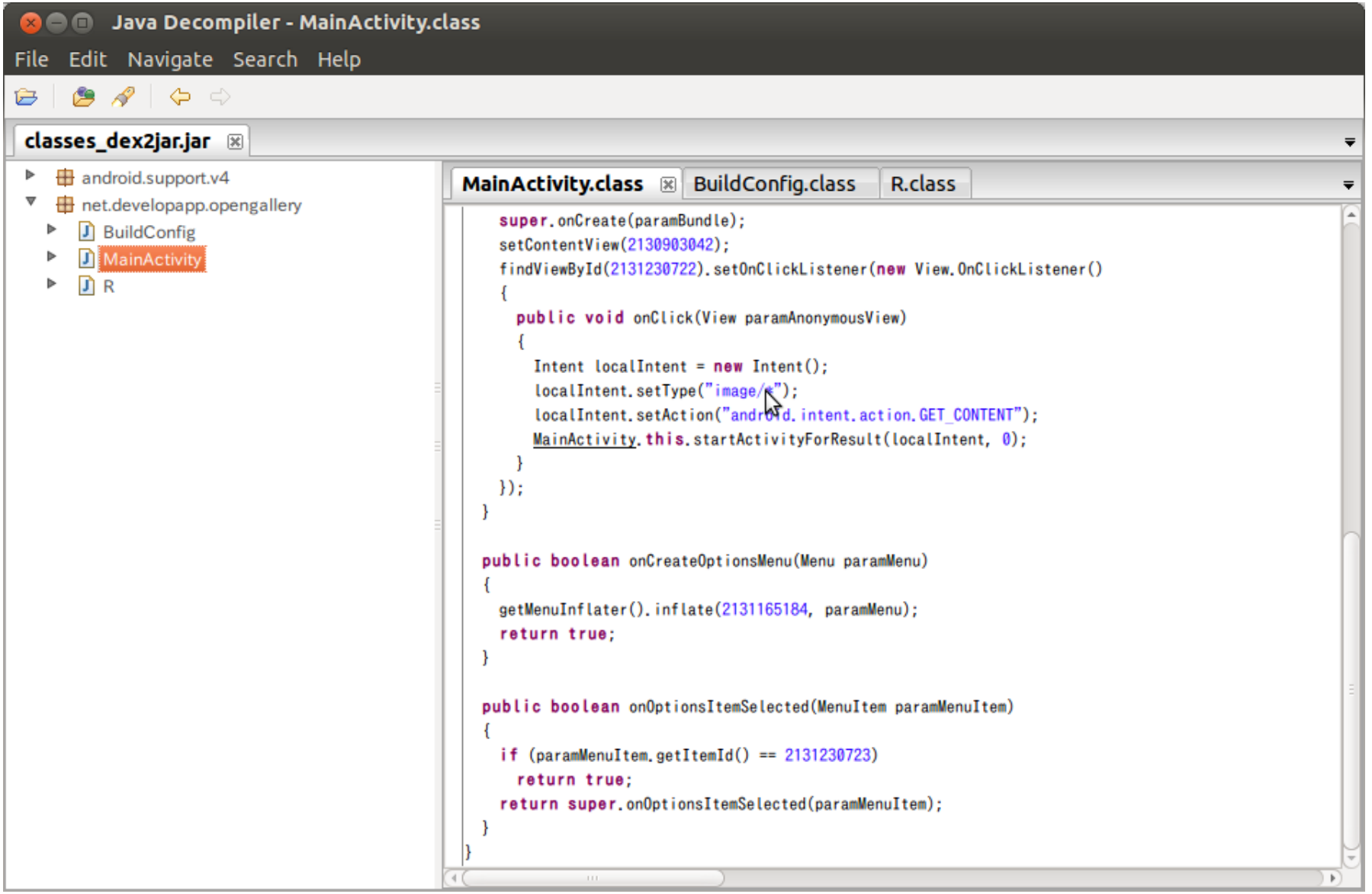
- Utile pour le java, C#
- Évite les problèmes de désassemblage de plusieurs niveaux .net -> gcc/llvm -> arch
- Plus facile à analyser

https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fresources.jetbrains.com%2Fstorage%2Fproducts%2Fdotpeek%2Fimg%2Fmeta%2Fdotpeek_250x250.png&f=1

https://images.duckduckgo.com/iu/?u=https%3A%2F%2Favatars1.githubusercontent.com%2Fu%2F11619605%3Fv%3D3%26s%3D400&f=1

Décompilation en java

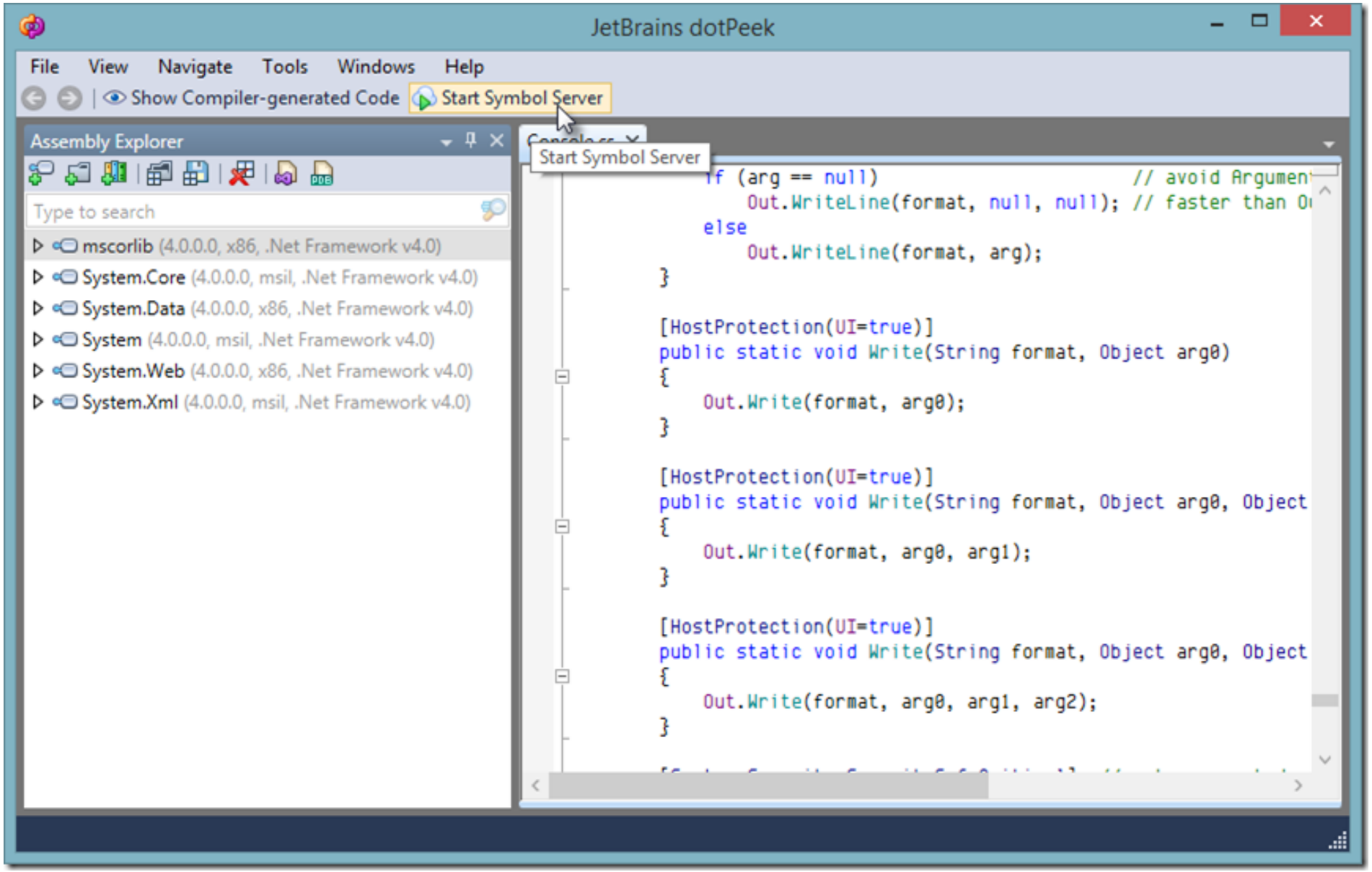
Site http://www.javadecompilers.com/



https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fwp.developapp.net%2Fwp-content%2Fuploads%2F2015%2F01%2Fjd.png&f=1

Décompilation en .Net

- Désassembleur syscalls (PE, C# -> C inconnu)



http://hmemcpy.com/2014/07/how-to-debug-anything-with-visual-studio-and-jetbrains-dotpeek-v1-2/image1.png

Démonstration (Haut Niveau)

Haut Niveau exercices

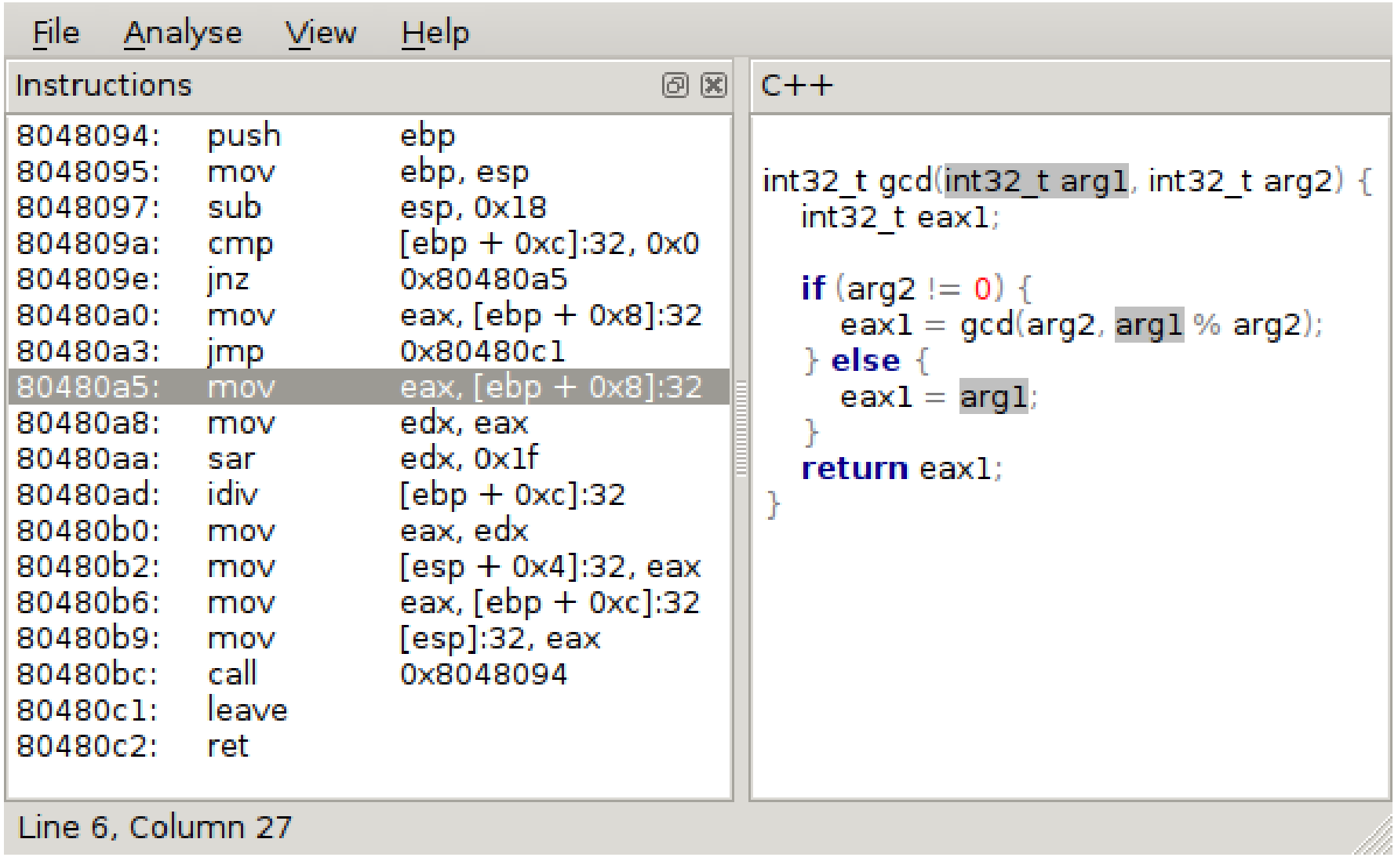
- Décompiler les fichiers .jar sur le site pour les challenges (github)
- Analyser le fichier InsecureBankingv2.apk et trouver l’encryption vulnérable (optionnel)
- Décompiler les fichier PE de type .net (.exe) et trouver le mot de passe ou la clé du programme

Moyen Niveau C/C++

- Possible d’éviter l’apprentissage de multiples assembleurs (NASM, GAS (ARM,PPC),etc.)
- Portable, facile à comprendre et à analyser
- C++ meilleure compréhension OOP, pas de transition directe dans l’assembleur
- Permet d’avoir une meilleure compréhension du code qui est traduit de façon ambiguë en ASM

Analyser le C++

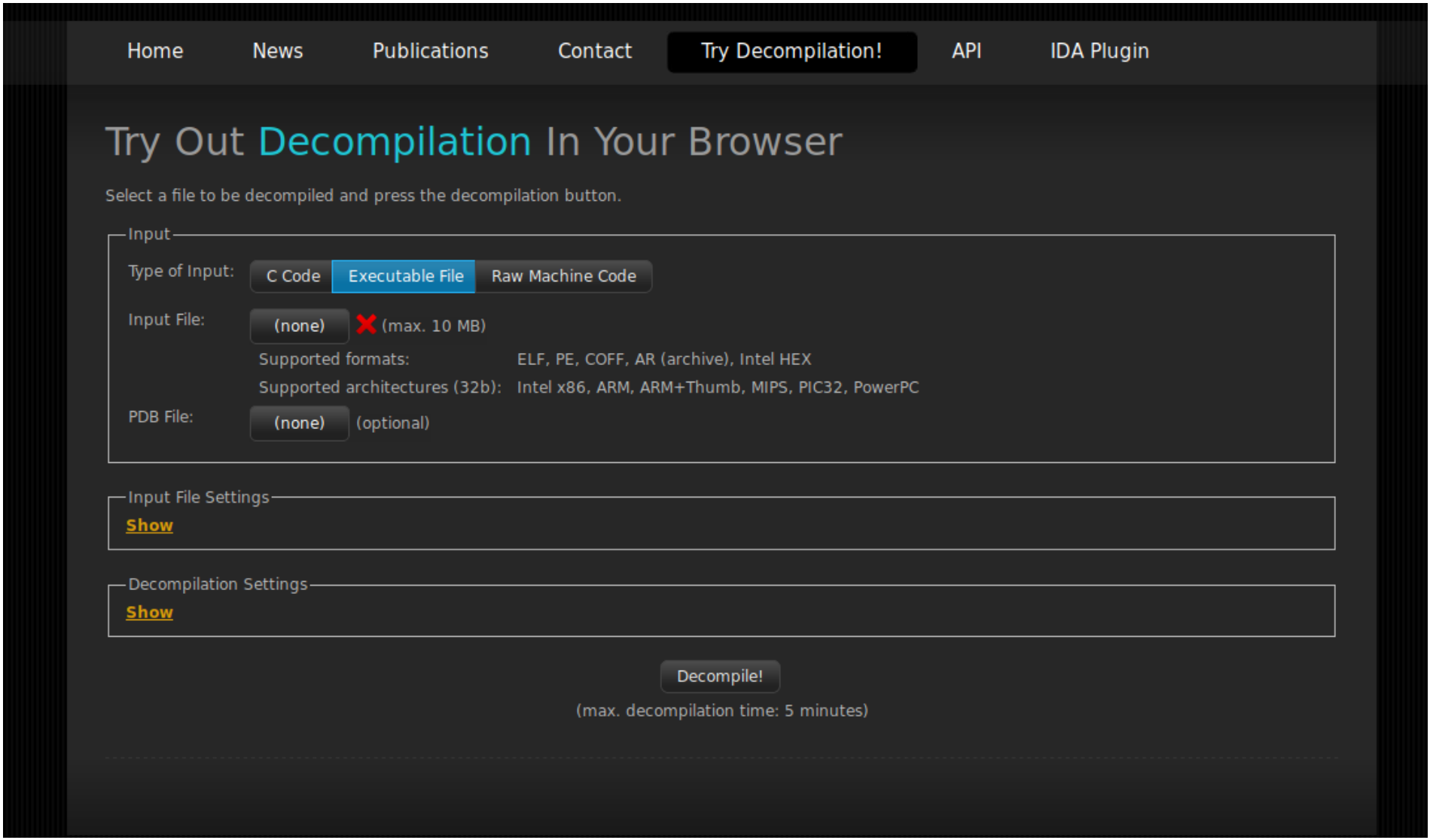
- Utiliser Ida Pro ARM et x86 seulement
- Plugin snowman (F3)



https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fi.stack.imgur.com%2FubvGP.png&f=1

Analyser le C

- Utiliser le site internet: https://retdec.com/decompilation/



Assembleur point clés

- Encodage (little/big endian)
- RISC vs CISC
- Les instructions les plus communes et utiles

```
.section    ".text"
        .global         _start
_start:
        mov 4,%g1                   ! 4 is SYS_write
        mov 1,%o0                   ! 1 is stdout
        set .msg,%o1                ! pointer to buffer
        mov (.msgend-.msg),%o2      ! length
        ta  8

        mov 1,%g1                   ! 1 is SYS_exit
        clr %o0                 ! return status is 0
        ta  8

.msg:
        .ascii      "Hello world!\n"
.msgend:
```

Endianness

- Big Endian (PowerPc,SPARC)

  le bit le plus significatif est à l'adresse la moins significative. Ex: 0x7370617263 -> "sparc"

- Little Endian(x86,(arch)el)

  le bit le moins significatif est à l'adresse la moins significative. Ex: 0x6372617073 -> "craps"

- En python mot[::-1] pour changer l'endianness d'un mot

CISC

- CISC (Complex Instructions Set)
- Peu de registres (généraux et spéciaux)
- Exemple x86: eax, ebx, ecx, edx, esp, edp
- Instrusctions complexes par exemple
- Inc edx-> i++ in C/C++

RISC

- RISC (reduced instruction set)
- Beaucoup de registres
- Example ARMv7: r0-r12 (généraux), r13 (stack), r14 (link) pour fonctions, r15 (horloge)
- Instructions simples (logiques, arith, etc.)
- Ex ARM :
- MOV R2, #0

  ADD R2, R2, #1

  i++ in C/C++

CISC VS RISC exemples

- CISC

  x86 (amd64,i386)

  6502 (nes)

  68k (m68k)

- RISC

  ARM (armhf, armel)

  SPARC (sparc(64))

  PowerPc (ppc(64(el)))

https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fimg3.wikia.nocookie.net%2F__cb20140207125257%2Flogopedia%2Fimages%2Fd%2Fd6%2FIntel_Xeon.png&f=1

https://images.duckduckgo.com/iu/?u=http%3A%2F%2Fwww.tug.ca%2Feblast%2FV24%2Flibrary%2Flogo_IBM_POWER6_BoP.gif&f=1

Instructions en ASM (ARM & x86)

- Arithmétiques
- Logiques
- Accès données et mémoire
- Les boucles et conditions

  Pour les architectures RISC (ARM,powerpc,sparc,etc.) {opt}.

  Pour l’x86 (opt).

Instructions arithmétiques

ASM C/C++

- ADD dest, src/val, {val2/src2} -> z= x + y;
- SUB dest, src/val, {val2/src2} -> z= x – y;
- (i)MUL dest(ecx), src/val, val2/src2 -> z= x * y;
- (i)div {dest}(ecx),{src/val},{src2/val2} -> z= x / y;
- DIV pas dans ARM seulement x86, ppc, sparc, etc.
- imul et idiv pour opérations signés en x86

Instructions Logiques

ASM C/C++

- AND dest, src/value,{src2/value2}-> z= x & y;
- OR dest, src/value,{src2/value2} -> z = x | y;
- XOR dest,src/value,{src2/value2} -> z = x ^ y;

Accès Mémoire et Donnés

- MOV dest,src/val ;move data to register (all)
- lea dest,src/val ;load calculated address x86
- LDR dest, [src]/=label ;loads at label/src address ARM
- STR dest, [src] ;stores at src address ARM
- PUSH {{reg}}/(val/reg) ;puts data on stack
- POP{{reg}}/(reg) ;restores registers(ARM)/removes data from stack into register (x86)

Les boucles et conditions

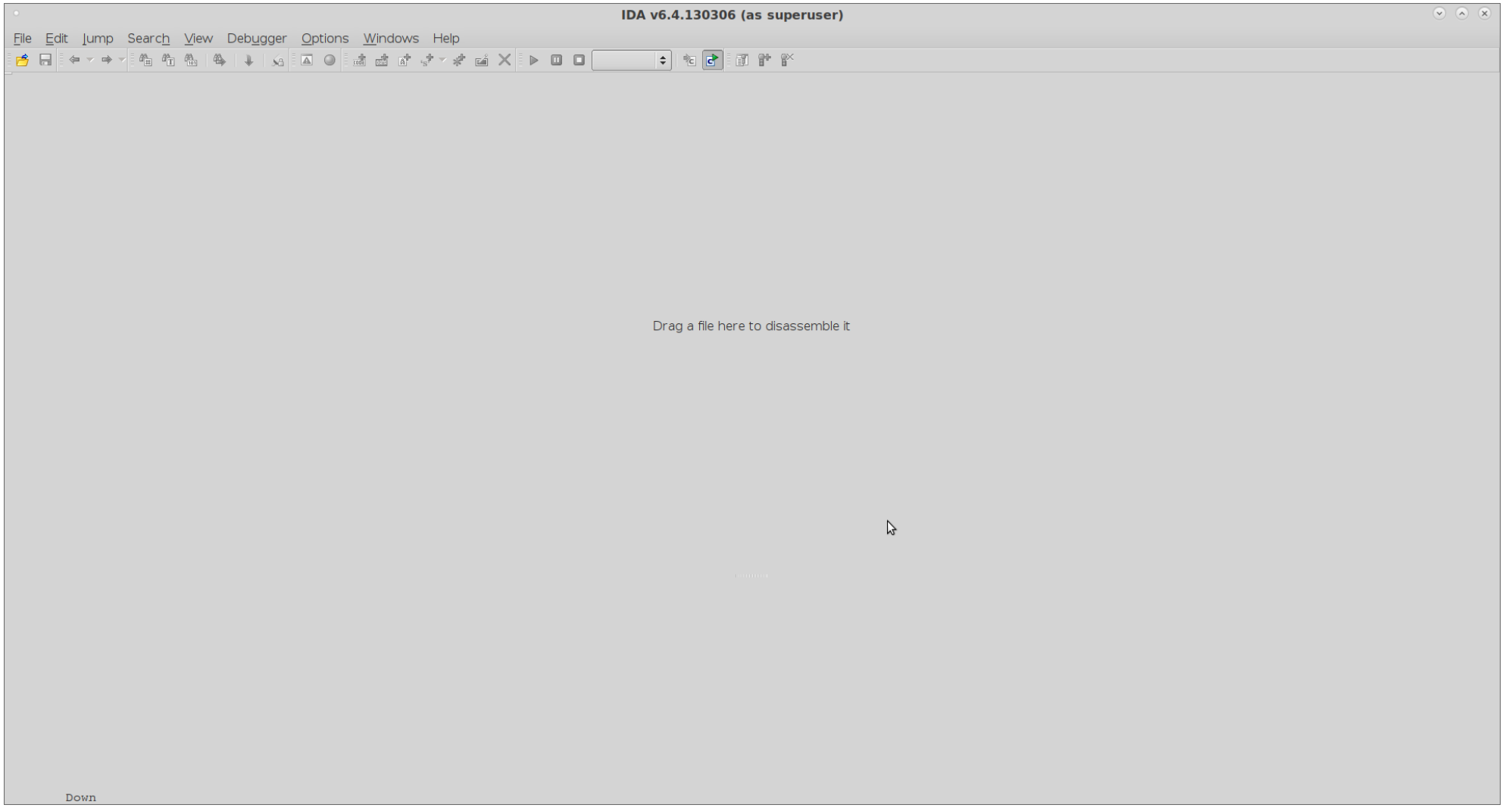
- cmp reg, reg/val ;vérifer la valeur d'un registre pour vérifier une condition (tous)
- B(cond) label ;sauter (in)conditionnellement à une fonction/procédure défini par un label (RISC)
- J(cond) label ;sauter (in)conditionnellement à une fonction/procédure défini par un label (x86)
- Définir le label dans le code l'équivalent d'un bloc if/else ou le code l'intérieur d'une boucle
- Loop en x86 sert à sauter à une boucle à partir du registre ecx

Conditions en assembleur

x86	ppc	sparc	ARM	C/C++
e	eq	e	EQ	==
ne	ne	ne	NE	!=
g/(a)	gt	g	GT	>
ge/(ae)	ge	ge	GE	>=
l/(b)	lt	l	LT	<
le/(be)	le	le	LE	<=

(cond) pour opérations non signés x86

Démo Ida pro


IDA v6.4.130306 (as superuser)
File Edit Jump Search View Debugger Options Windows Help
Drag a file here to disassemble it
Down

Exercices (ASM et C/C++)

- Télécharger les applications crackme de github folder C(++) and ASM binaries
- Utiliser Ida Pro ou Hopper pour les résoudre
- Ne pas debug aucune aide sera accordé pour ceux utilisant un debugger

Pour plus d’information (ASM)

- http://en.wikibooks.org/wiki/X86_Assembly
- http://www.ds.ewi.tudelft.nl/vakken/in101/labcourse/instruction-set/
- https://www.tutorialspoint.com/assembly_programming/index.htm
- https://en.wikibooks.org/wiki/SPARC_Assembly
- Guide to RISC Processors: for Programmers and Engineers

Outils pour la rétro-ingénérie

- Disassemblers:

  Ida Pro: https://www.hex-rays.com/products/ida/index.shtml

  Hopper: https://www.hopperapp.com/download.html

  Binary Ninja: https://binary.ninja/faq/

  Radare2: http://www.radare.org/r/

- Decompilers C/C++:

  retargetable: https://retdec.com/home/

  snowman: https://derevenets.com/

Outils pour la rétro-ingénérie(suite)

- Decompilers .Net:

  https://github.com/0xd4d/dnSpy/releases

  https://www.jetbrains.com/decompiler/
- Decompilers Java:

  JD-GUI(out of date): http://jd.benow.ca/

  http://www.javadecompilers.com/

Conclusion

- Utiliser un décompilateur pour des .jar, .apk et .exe (.net)
- Pour des elf, mach-o ou PE utiliser un désassembleur et analyser le code source de de l’application (ASM)