Introduction to the Witchcraft Compiler Collection

Jonathan Brossard

Draft Slides,
8 July 2016

DEFCON 24, Las Vegas, USA

AGENDA

- WCC components
- “Libifying” a binary
- Killing Reverse Engineering
- Crossing a Fish and a Rabbit
- Introduction to Witchcraft
- Binary “reflection” without a VM
- Towards binary self awareness
- Future work

WCC COMPONENTS

wld : witchcraft (un)linker
wcc : witchcraft core compiler
wsh : witchcraft shell : dynamic interpreter + scripting engine

Host machine : GNU/Linux (soon all POSIX systems).

WLD : "LIBIFICATION"

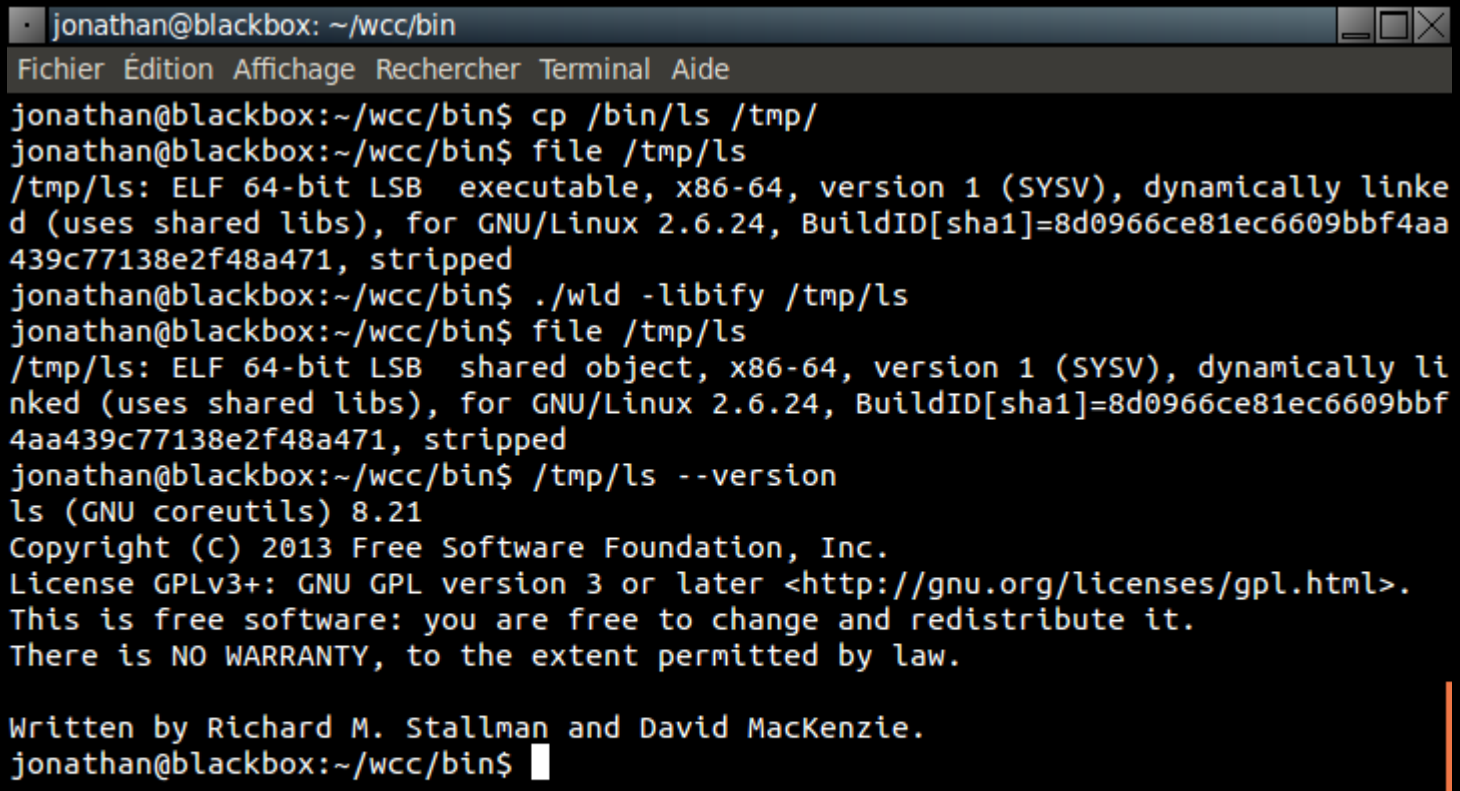
Transforming an ELF binary into an ELF shared library by simply patching the ELF type in the ELF header.

Because shared libraries are executable -see how ASLR is implemented via Position Independent Executables (PIE) on GNU/Linux - this (unexpectedly) works.

DEMOS

(Unlinking/Relinking)

LIBIFICATION OF /BIN/LS



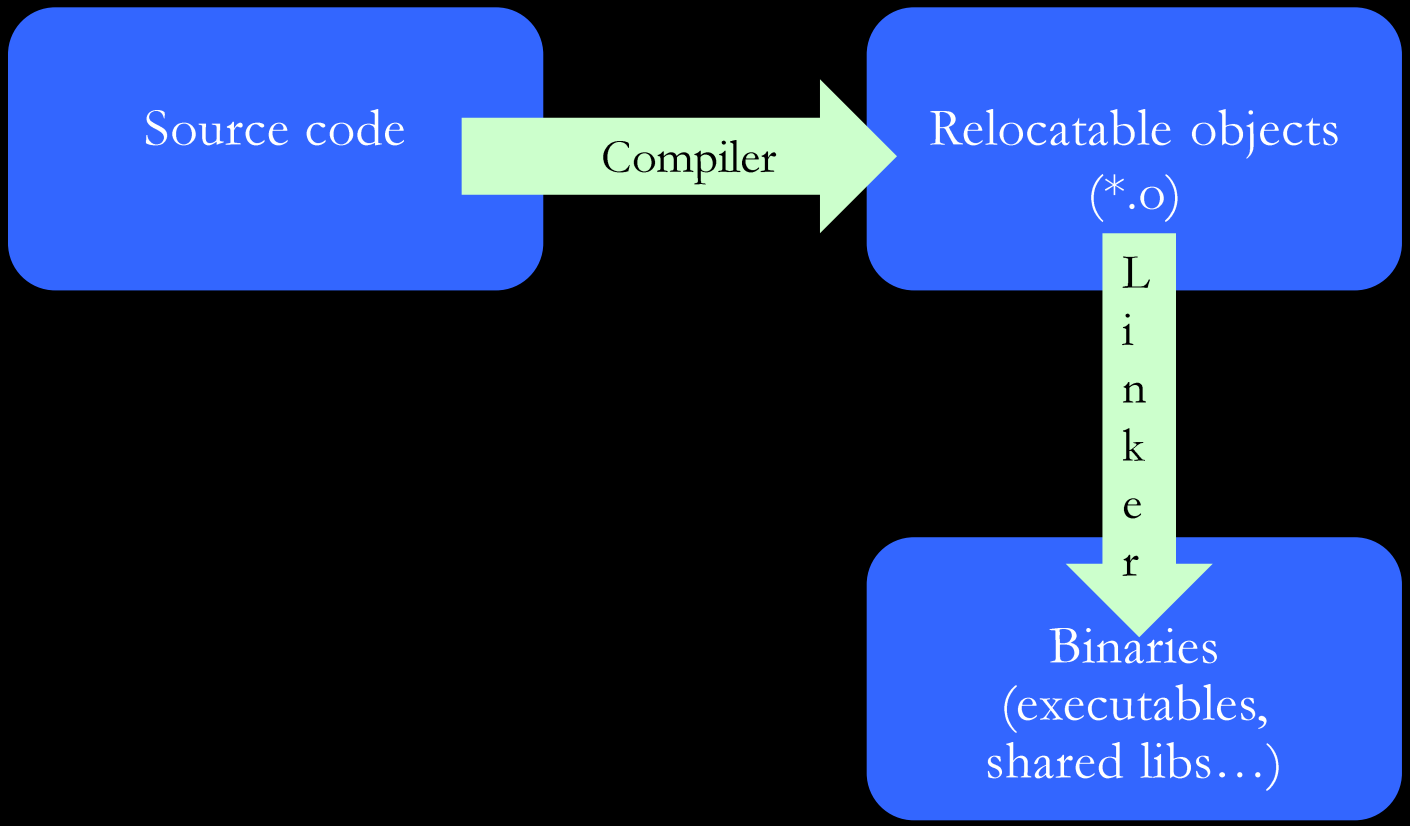
WCC : "UNLINKING"

The typical approach to reverse engineering is to transform binaries or shared libraries back to source code.

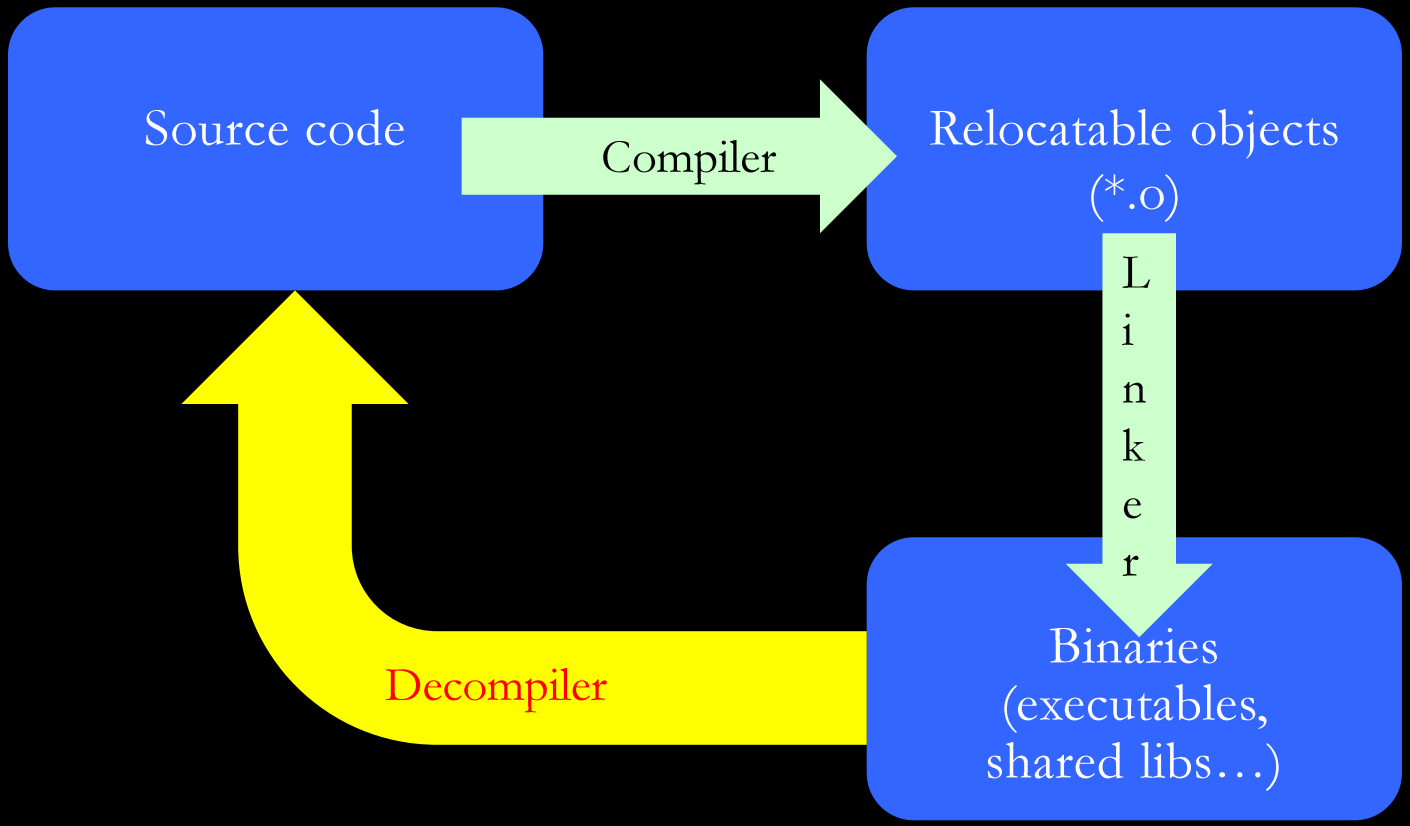
Instead, we aim at transforming final binaries or shared libraries back to ELF shared objects, that can later be relinked normally (using gcc/ld).

This binary refactoring is enough to "reuse" (steal) binary functions without any disassembly.

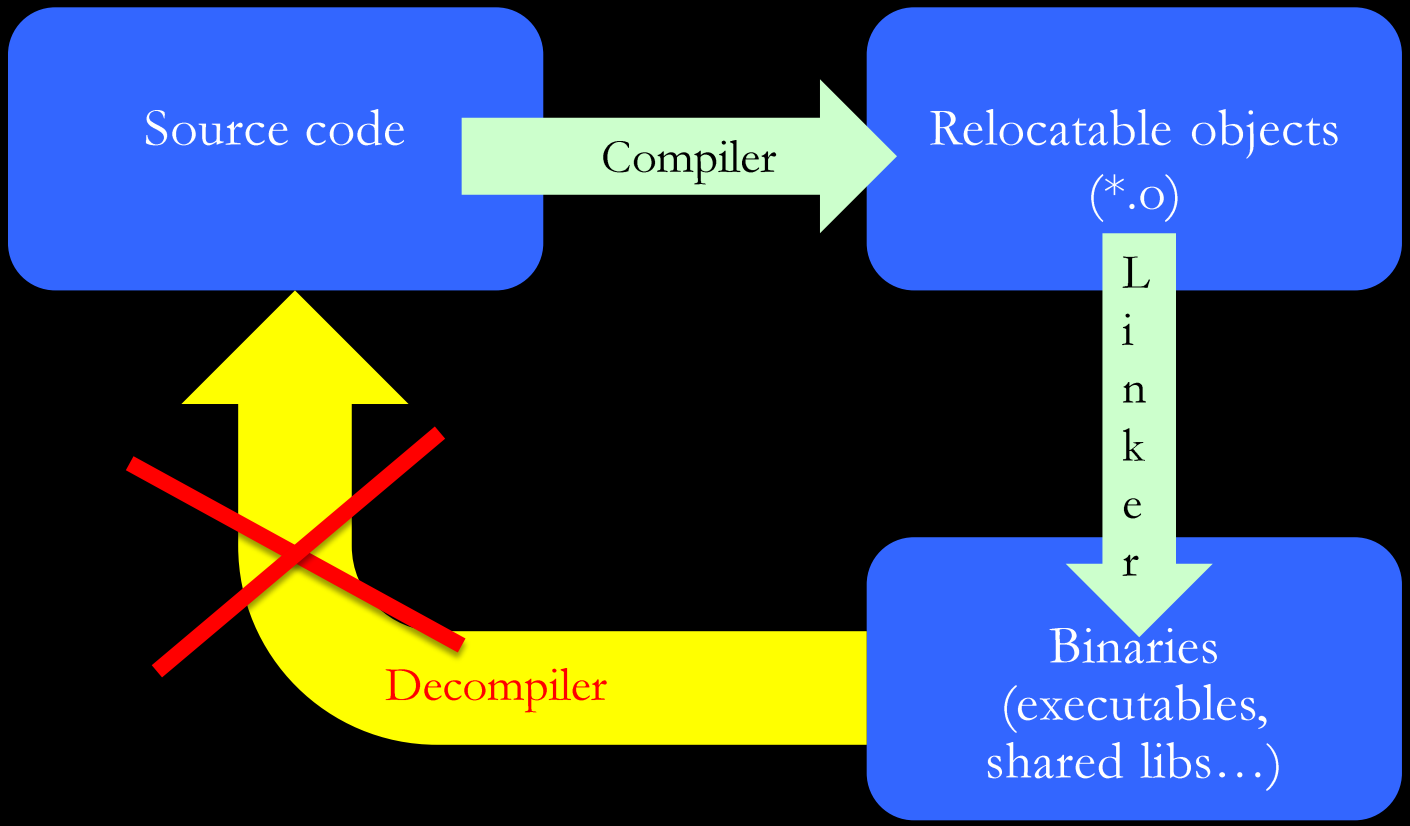
KILLING REVERSE ENGINEERING



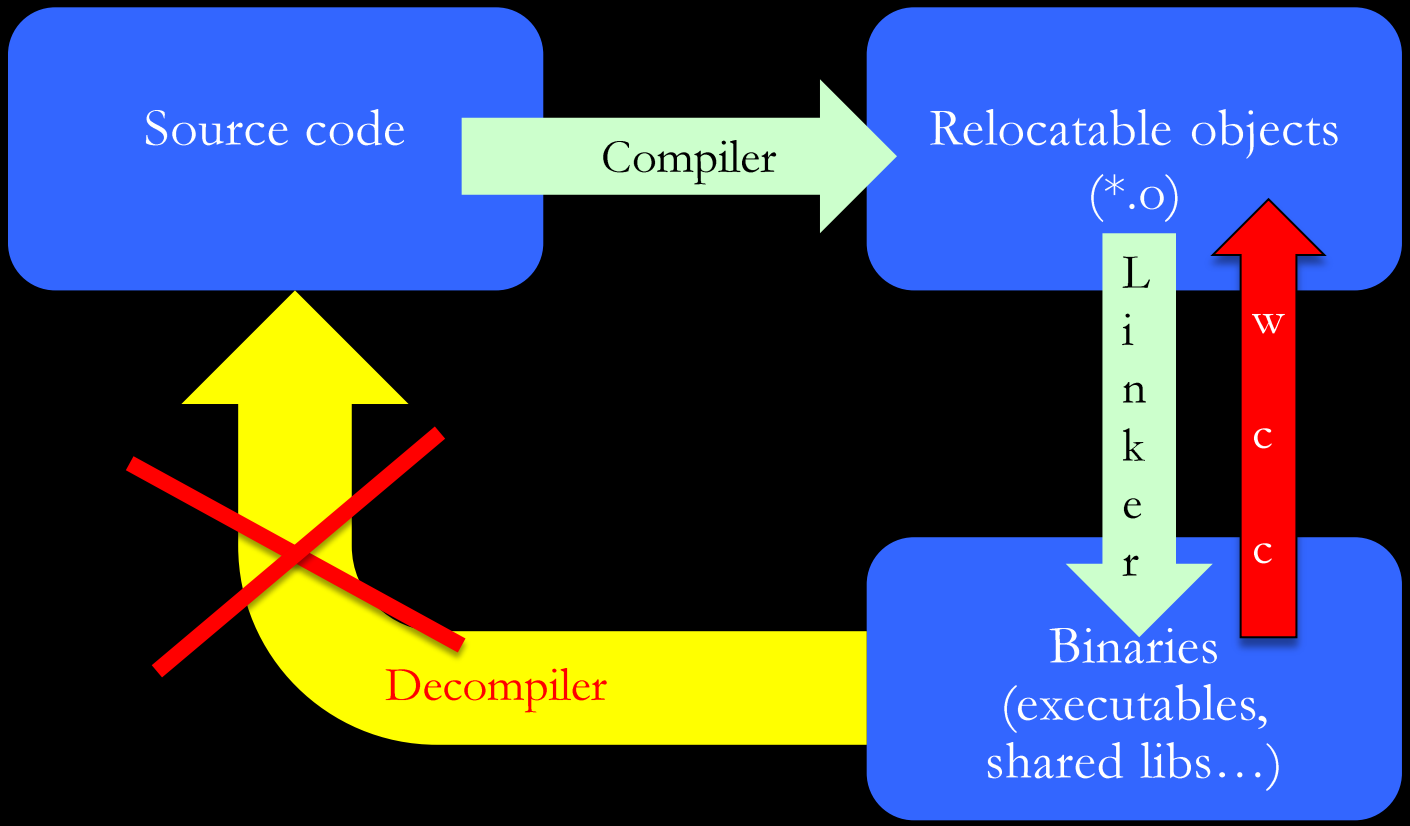
KILLING REVERSE ENGINEERING



KILLING REVERSE ENGINEERING

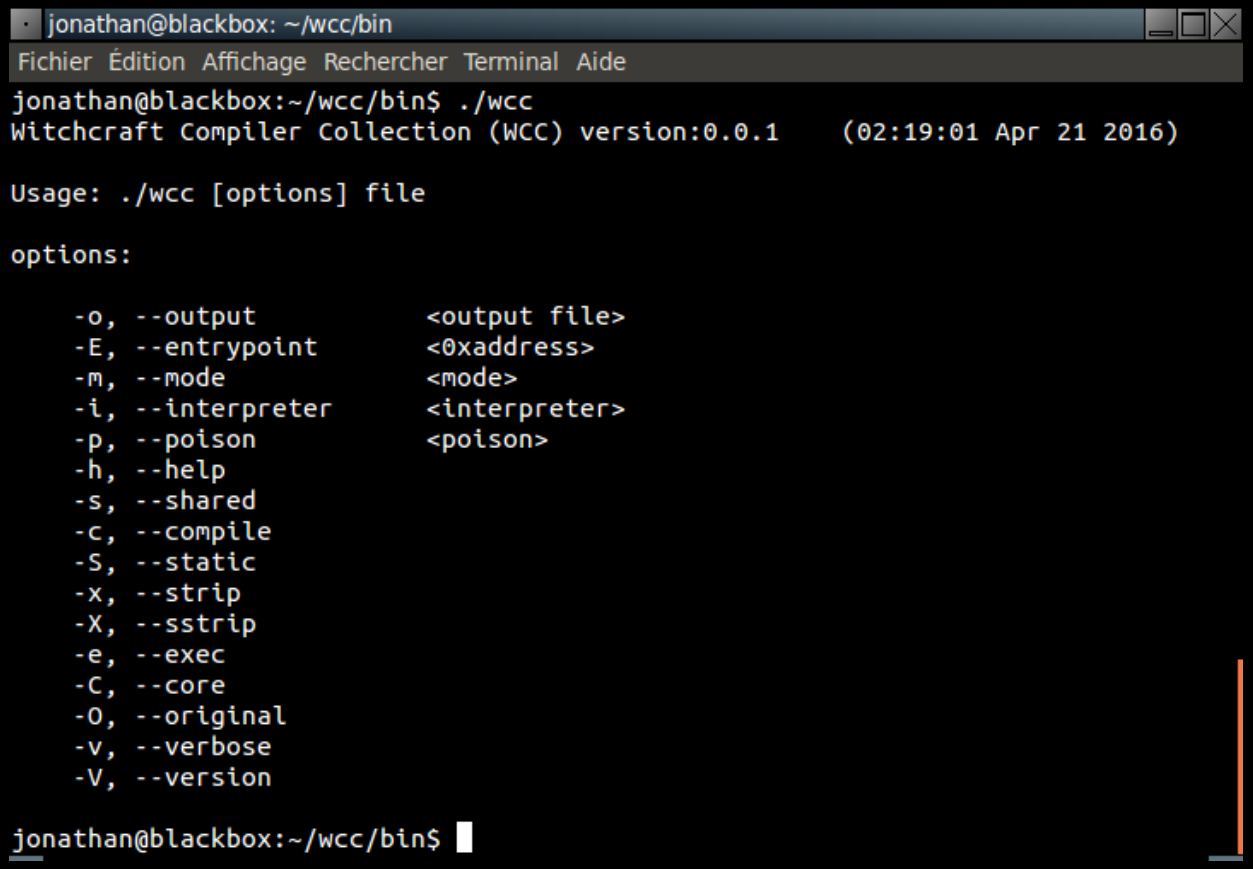


KILLING REVERSE ENGINEERING



WCC : COMMAND LINE

The command line is made to resemble the syntax of gcc :



WCC : INTERNALS

The front end is build around libbfd. The backend is trivial C to copy each mapped section of the binary, handle symbols and relocations.

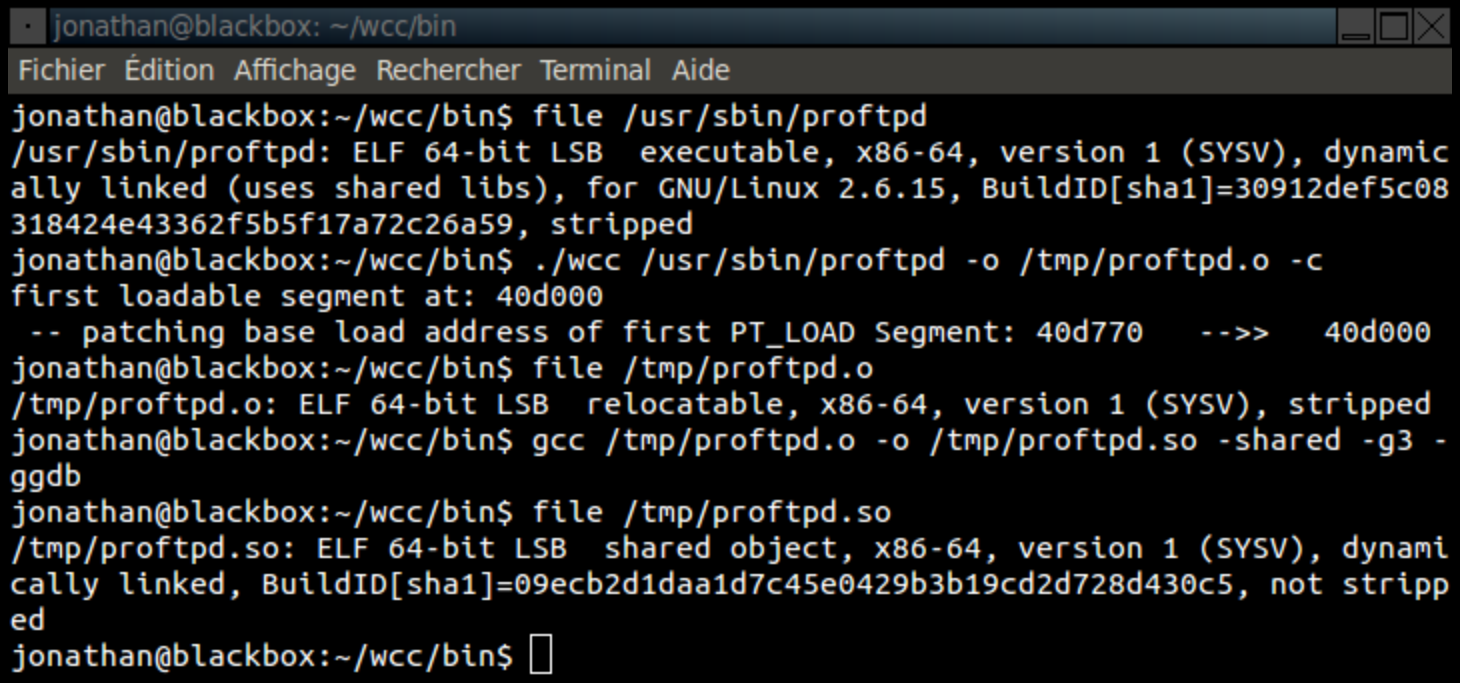
Benefit of using libbfd : the input binary doesn't need to be an ELF !

=> We can for instance transform a Win64 executable into ELF 64b relocatable objects…

DEMO

(Binary to object file to unstripped library)

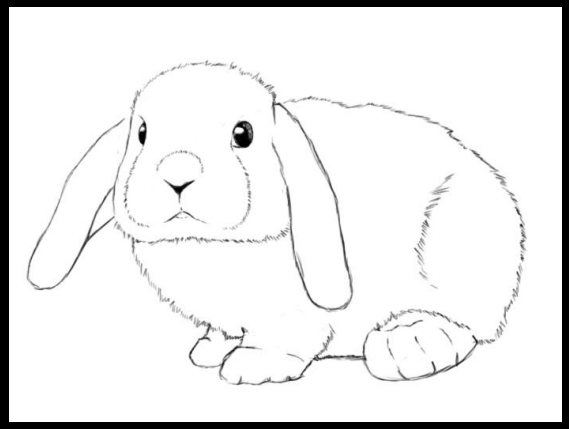
WCC : DEMO

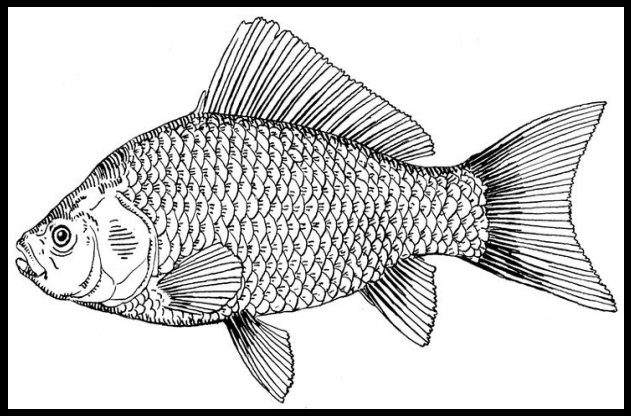

jonathan@blackbox: ~/wcc/bin
Fichier Édition Affichage Rechercher Terminal Aide
jonathan@blackbox:~/wcc/bin$ file /usr/sbin/proftpd
/usr/sbin/proftpd: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamic
ally linked (uses shared libs), for GNU/Linux 2.6.15, BuildID[sha1]=30912def5c08
318424e43362f5b5f17a72c26a59, stripped
jonathan@blackbox:~/wcc/bin$ ./wcc /usr/sbin/proftpd -o /tmp/proftpd.o -c
first loadable segment at: 40d000
-- patching base load address of first PT_LOAD Segment: 40d770 -->> 40d000
jonathan@blackbox:~/wcc/bin$ file /tmp/proftpd.o
/tmp/proftpd.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), stripped
jonathan@blackbox:~/wcc/bin$ gcc /tmp/proftpd.o -o /tmp/proftpd.so -shared -g3 -
ggdb
jonathan@blackbox:~/wcc/bin$ file /tmp/proftpd.so
/tmp/proftpd.so: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynami
cally linked, BuildID[sha1]=09ecb2d1daa1d7c45e0429b3b19cd2d728d430c5, not stripp
ed
jonathan@blackbox:~/wcc/bin$

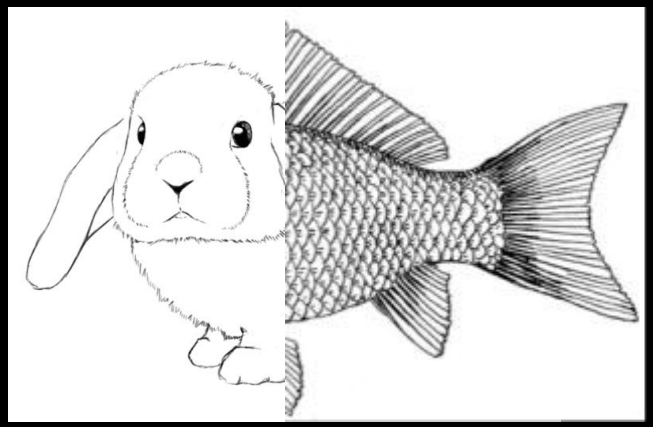
DEMO

(Crossing a Fish and a Rabbit)

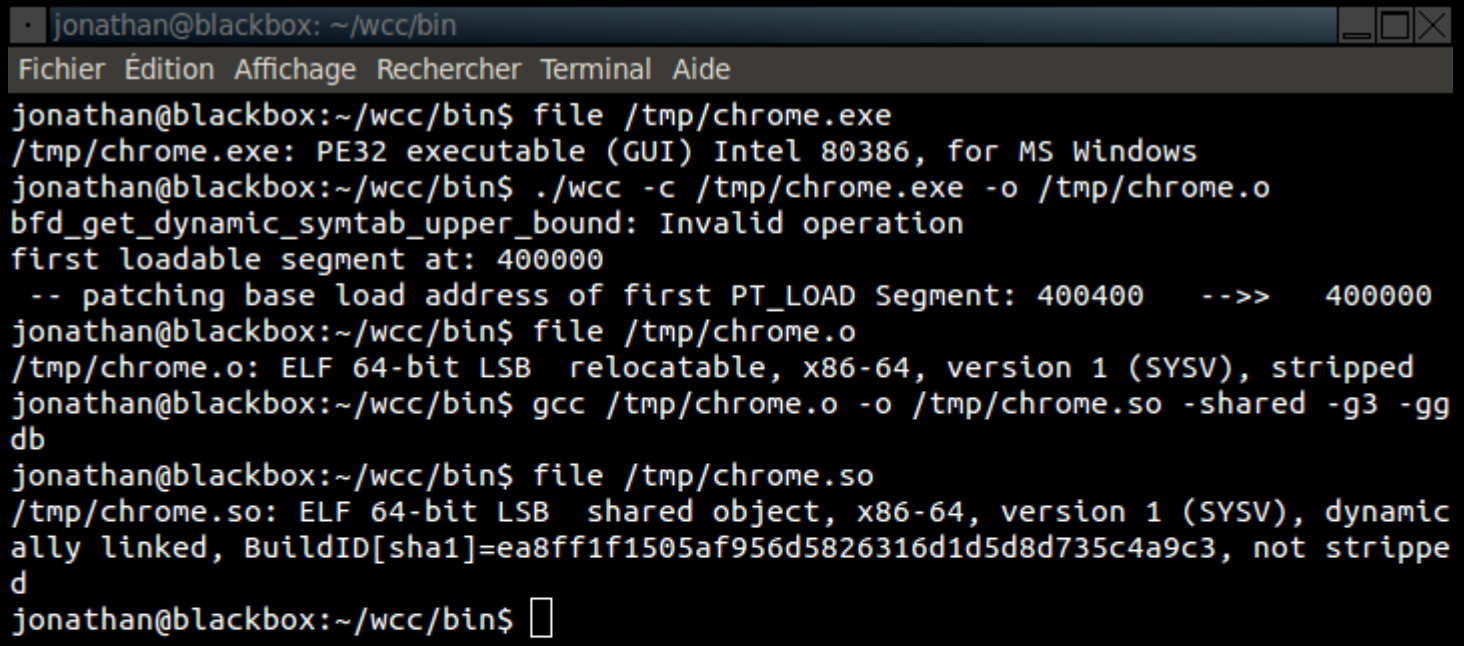
PE + ELF = PELF







WCC : PE32 TO ELF64



WITCHCRAFT

(hacking consciousness)

INTRODUCTION TO WITCHCRAFT

BINARY "REFLECTION" WITHOUT A VM

Now that we know how to transform arbitrary binaries into shared libraries, we can load them into our address space via dlopen().

Let's implement the same features as traditional virtual machines, but for raw binaries !

Whish list :

- Load arbitrary applications into memory
- Execute arbitrary functions with any arguments (and get results)
- Monitor/Trace execution
- Automated functions prototyping/annotation
- Learn new behavior
- Examine/Modify arbitrary memory

WSH : ARCHITECTURE

Loading is done via dlopen().
The core engine/shell is built around lua.
Can be compiled with luajit to get JIT compilation.
Tracing/Memory analysis doesn't rely on ptrace() : we share the address space.
Lightweight : ~5k lines of C.
No disassembler (as of writing. Subject to change).
No need for /proc support !
Function names mapped in each library is dumped from the link_map cache.

WSH : THE WICHCRAFT INTERPRETER

Distinctive features:

- We fully share the address space with analyzed applications (no ptrace() nor context switches).
- Requires no privileges/capabilities (no root, no ptrace(), no CAP_PTRACE, no /proc…)
- No disassembly : fully portable (POSIX)
- Implements “reflection” for binaries
- Full featured programming language
- Interactive and/or fully scriptable, autonomous programs
- Has no types
- Has no API : any function you load in memory becomes available in WSH
- Functions have no prototypes

=> Can call arbitrary functions without knowing their prototypes
=> Allows for extended function annotations (to be fully automated)
=> Steal/Reuse any code. Add scripting to any application.

NONE OF THIS IS SUPPOSED TO WORK

WSH : THE WICHCRAFT INTERPRETER

Advanced features:

- Loads any code via dlopen() : this solves relocations, symbols resolution, dependencies for us.
- Secondary loader bfd based (could load invalid binaries, anything in memory).
- Dumping of dynamic linker cash internals (undocumented) : linkmap
- Breakpoints without int 0x03 (use SIGINVALID + invalid opcode)
- Bruteforcing of mapped memory pages via msync() (0day, no /proc needed)
- Wsh can be compiled to do JIT compilation on the fly at runtime.
- Automated fuzzing/extended prototyping/functional testing

NONE OF THIS IS SUPPOSED TO WORK

WITCHCRAFT

DEMO

TOWARDS BINARY SELF AWARENESS

Consciousness 1.0

SELF AWARENESS

"Self-awareness is the capacity for introspection and the ability to recognize oneself as an individual separate from the environment and other individuals."

https://en.wikipedia.org/wiki/Self-awareness

"Consciousness is the state or quality of awareness, or, of being aware of an external object or something within oneself. It has been defined as: sentience, awareness, subjectivity, the ability to experience or to feel, wakefulness, having a sense of selfhood, and the executive control system of the mind."

https://en.wikipedia.org/wiki/Consciousness

WITCHCRAFT

Numerical solutions

NUMERICAL SOLUTIONS

- To do anything relevant in AI, we need a fitness function that :
* is monotonous
* is bounded
=> Therefor converges

- To craft an input that goes from a() to b() to c() to d(), we :
* assign breakpoints to a,b,c,d and assign them weights
* create a fit application that is the sum of the breakpoints reached, weighted

=> Function is bounded (sum of all weights)
=> Function is monotonous (increases)
=> Automated fuzzing of death !!

WITCHCRAFT

FUTURE WORK

FUTURE WORK

- Hyde our own presence better in memory (second heap)
- Remote debugging, running process injection
- Shadow mapping, internal libraries tracing (recursive ltrace)
- ltrace/strace to valid scripts
- system call tracing
- Skynet / World domination

TO BE CONTINUED

Questions ?