How to port a DOS game to modern systems
Authors

- Unavowed
- Gynvael Coldwind

Additional help / code / art:

- j00ru, MeMeK, oshogbo, Blount, xa
What did we do

- Port Syndicate Wars
- DOS → modern x86 systems
  - GNU/Linux, Windows, Mac OS X
  - others with SDL/OpenAL support
- no source code
About Syndicate Wars

• DOS 3D tactical squad action game by Bullfrog
• sequel to Syndicate
About Syndicate Wars

- Demo
Why?

- For fun and profit
- Did not work on DOSBox
- Inspired by the John Jordan's Frontier: First Encounters (JJFFE) project
How we did it

- Using recompilation techniques:
  1. Disassemble $\rightarrow$ recompilable form
  2. Find & replace DOS-specific parts with portable C code, using free software portable libraries
  3. Compile $\rightarrow$ native executable
Disassembling

- Executable type: 32-bit Linear Executable (LE) with a DOS extender (dos4gw)
- Compiled with the WATCOM C/C++ compiler
- No applicable disassembler
- We created our own: swdisasm
Disassembling - LE

- 16-bit loader (that called dos4gw.exe)
- 32-bit application code (loaded by dos4gw.exe)
- Sections (called objects)
- Relocations
Goal:
- all static addressing → labels
- code and data separation
- compilable output

Three attempts:
- ledisasm v1 (C)
- tracing ledisasm (Python)
- swdisasm (C++)
ledisasm v1:

- ndisasm engine
- multiple linear passes
  - detecting functions
  - detecting padding
  - detecting “vtables” (?)
- output: nasm
ledisasm v1 problems:

- mixed consts/valid addresses
- alignment problems
- didn't detect all "labels"

reasons:
- linear passes
- insufficient use of relocs
swdisasm:

- tracing disassembler
- prototype in Python (slow)
- using binutils instead of ndisasm
- region map
  - 1 region per section at start, subdivided into smaller regions with assigned types (code/data/vtable)
- label list
swdisasm:

- how does it work:
  - has a trace address queue
  - add OEP to the queue
  - trace through the queue until empty
    - adds new addresses to the queue while tracing
    - subdivides regions
  - trace through the reloc targets
  - add labels for data relocations
ledisasm → swdisasm

swdisasm problems:

• padding is ignored → data arrays in code sections are lost
• a few unusual cases in the source executable
• workaround: assign 14 regions manually
swdisasm summary:

• ~2 seconds to disassemble a 1.7MB exec

$ time ./swdisasm main.exe > swars.S
Tracing code directly accessible from the entry point...
Tracing text relocs for vtables...
Warning: Reloc pointing to unmapped memory at 0x140096.
Tracing remaining relocs for functions and data...
0 guess(es) to investigate.
Region count: 3954

real 0m1.755s
user 0m1.716s
sys 0m0.024s
What's left to recompile?

• add underscores on OSX/W32

```c
#ifdef NEED_UNDERSCORE
#define TRANSFORM_SYMBOLS
#define EXPORT_SYMBOL(sym) _ ## sym
#endif
```

• call swars main from C

```c
// Call game main
asm volatile ("call asm_main
 : ":a" (retval) : "a" (argc), "d" (argv));
```
What's left to recompile?

• it works!!!
• (well, actually it just compiles/links)
What next?

• Now the game would execute and immediately crash

• Next goal:
  • finding/replacing asm code with C code/calls to libc

• Things to look for:
  • interrupts (int $0xNN, int386(), setvect()), port accesses (in, out instructions)
  • DPMI / dos4gw “API”
  • statically-linked libc functions
C Lib

- Manually look for functions:
  - Find a function using interrupts
  - Easy picks: file operations
  - Compare to Open WATCOM source
  - Look at nearby code for other likely libc functions
  - Time consuming!
  - Did not finish (got 40% of used functions)

- Received a .map of libc in main.exe from an IDA user (thank you :)
Replacing code: `asm → C`

- Incompatible calling conventions
- x86 cdecl: arguments pushed on stack
- “watcall”: passing in registers:
  - eax, edx, ebx, ecx, rest pushed on stack
  - but cdecl for vararg functions
- Different registers preserved
MKWRAPPERS

- **mkwrappers**: asm → C ABI wrapper generation in python
- **input**: configuration file, parameters
  - e.g. wrappers_libc.conf

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>args</th>
</tr>
</thead>
<tbody>
<tr>
<td>rename</td>
<td>w</td>
<td>ss</td>
</tr>
<tr>
<td>rmdir</td>
<td>w</td>
<td>s</td>
</tr>
<tr>
<td>setbuf</td>
<td>w</td>
<td>pp</td>
</tr>
<tr>
<td>sprintf</td>
<td>v</td>
<td>psv</td>
</tr>
<tr>
<td>srand</td>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>sscanf</td>
<td>v</td>
<td>ssv</td>
</tr>
</tbody>
</table>
configuration file syntax

# type is one of:
# w - watcom: args passed in
#     eax, edx, ebx, ecx
# c - cdecl
# v - vararg
#
# args is a sequence of zero or more of:
# i - int
# x - int (displayed in hex)
# p - void * (general pointer)
# s - char *
# c - char
# v - ...
# l - va_list
MKWRAPPERS

Output: wrappers in asm:

```assembly
.global ac_rename
ac_rename: /* w ss */
push %ecx
push %edx
push %edx
push %eax
call _rename
add $0x8,%esp
pop %edx
pop %ecx
ret
```
Output: wrappers in asm (debug):

```assembly
.global ac_rename
ac_rename: /* w ss */
push %ecx
push %edx
push %edx
push %eax
push %edx
push %eax
push %eax
push $0f
call _printf
add $0xc, %esp
call _rename
add $0x8,%esp
pop %edx
pop %ecx
ret
.data
 0: .string "rename ("%s", "%s")\n"
.text
```
MKWRAPPERS

• Output: wrappers in asm (vararg):

```assembly
.global ac_printf
ac_printf: /* v sv */
push %ebp
mov %esp,%ebp

push %ecx
push %edx

lea 0xc(%ebp),%eax
push %eax
push 0x8(%ebp)

call _vprintf
add $0x8,%esp

pop %edx
pop %ecx
leave
ret
```
MKWRAPPERS

- Function renames (strcasecmp vs stricmp)
- Additional parameters:
  - Underscores in symbols
  - Call stack alignment
Replacing libc calls

- We now had mkwrappers and the libc symbol map
- We made substitutions: `s/_printf/ac_printf/g`
Replacing libc calls

- Game started working!

(as in: displaying debug output before crashing hard)
Approach to replacing game code

- Having found DOS-specific functions, we:
  - Identified purpose using DPMI spec/Ralf Brown's interrupt list
  - Noted what data they touch
  - Looked for other functions touching it
- After finding interesting functions:
  - Manually translated functions into C
  - Got an understanding of how a subsystem works
  - Wrote replacements
Replacing unportable code

• The aim of replaced functions:
  • Communication with game by reading/writing variables according to some protocol
  • In a portable manner
  • Call free software portable libraries for video, audio, keyboard and mouse input
Things replaced

- Low level DOS/hardware functions
- Video code
- Audio
- Mouse and keyboard input
- Event loops
Low level DOS/hardware

• Path handling
  • Case-insensitive file names on case-sensitive file systems
  • Support for per-user profiles
  • Date and time (gettime/getdate), file handling (sopen, setmode)

• Timing
  • 8254 Programmable Interrupt Timer (PIT) used in intro playback
• Game uses 3D software rendering
• Originally implemented using VESA
• 8-bit palette mode
• Needed to set video mode and provide a framebuffer
• Reimplemented with SDL
Program wykonał nieprawidłową operację i nastąpi jego zamknięcie. Jeśli problem będzie się powtarzał, skontaktuj się ze sprzedawcą.
Video

Video output...

```
<silent@silent:silent>$ vim swars.asm
<silent@silent:silent>$ make clean
rm -f swars swars.o main.o clib.o graphics.o timer.o debug.o clib.inc core
<silent@silent:silent>$ make
../util/wrappers.py -g ../util/wrappers.conf > clib.inc
nasm -f elf -o swars.o swars.asm
gcc -c -g -Wall -I/usr/include/SDL -D_REENTRANT -o main.o main.c
gcc -o -Wall -I/usr/include/SDL -D_REENTRANT -o clib.o clib.c
```

```
not be used.
```

```
<silent@silent:silent>$ python
Python 2.3.5 (#1, Apr 28 2005, 02:32:18)
[GCC 3.3.5-20050130 (Gentoo Linux 3.3.5.20050130-r1, ss on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
jump_19576:
esi = &local_4;
push esi
local_8 = eax;
push eax
push ds
```
Input

- Also SDL-based replacements
- Keyboard
  - Keyboard controller interrupt handler
  - Needed to fill key-event ring buffer and set key state table
  - Talking with 8041/8042
- Mouse
  - Mouse interrupt handler (with “Mickey’s”)
  - Set location, motion and button state variables according to a locking protocol
Audio

- Originally statically-linked Miles Sound System library
- Pluggable drivers
- Analysed top-down, not bottom-up
  - Found getenv("AIL_DEBUG") which controlled debug output
  - Found newer headers for this library
  - Used the two to identify functions and data structures
Audio

- Originally samples polled by sound card interrupts
- Reimplemented using OpenAL
- CDDA music → Ogg/Vorbis with libvorbis, needs to be ripped and encoded
Event Loops

- Originally sound/input updates triggered asynchronously – no longer an option
- Needed to periodically call game_update() to flip frames / poll input / push audio
- 4 “main loops” in the game, depending on mode:
  - intro, menu gui, mission display, paused mission gui
- Easy to find: game would freeze!
OS X issues

- 16-byte call stack alignment
- Ancient version of GNU as in XCode
  - No support for .global, .string, .fill, C-style escapes and instruction syntax differences
  - Bug which miscalculated loop* instruction target relative addresses
- Workaround:
  - Add stack alignment to mkwrappers
  - asfilter script in python implementing missing features and replacing loops with sub/jmp
Release

• Wrote installers:
  • bash script with cdparanoia for the UNIX-like
  • Nullsoft with AKRIP for Windows (by j00ru)
  • bash script for making Mac OS bundles
• Port released 5 years since inception
• Available at http://swars.vexillium.org/
Post-release

• Bug reports:
  • Missing bounds checking on paths, overflow after 100 bytes

• Things left to do:
  • Network multiplayer support
  • Joystick/game controller support
Conclusion

• Final code size:
  • asm: 380 kLOC
  • portable C: 2.5 kLOC

• Time to completion: 5 years
  • (of which a few months of real work)

• Countless nights spent debugging

• A cool working game!
Questions?
Contact

- http://swars.vexillium.org
- http://unavowed.vexillium.org
- http://gynvael.coldwind.pl
- http://vexillium.org