Witchcraft Compiler Collection :
User Manual

Jonathan Brossard

v1.0 November 2016

The Witchcraft Compiler Collection User Manual

Welcome to the Witchcraft Compiler Collection User Manual.

The latest version of this manual is available at:

https://github.com/endrazine/wcc/wiki

The Witchcraft Compiler Collection User Manual

Getting started

Other resources

wcc

wcch

wld

wldd

wsh

wsh : commands

wsh : Core API

wsh : the punk-C language

wsh with ARM

This page documents how to download, compile and install WCC.

Downloading the source code

The official codebase of the Witchcraft Compiler Collection is hosted on github at https://github.com/endrazine/wcc/ . It uses git modules, so some extra steps are needed to fetch all the code including depedencies. To download the source code of wcc, in a terminal, type:

```
git clone https://github.com/endrazine/wcc.git
cd wcc
git submodule init
git submodule update
```

This will create a directory named wcc and fetch all required source code in it.

Prerequisites

Installing requirements

The Witchcraft Compiler Collection requires the following software to be installed:

```
    Glibc, libbfd, libdl, zlib, libelf, libreadline, libgsl.
```

Installing requirements on Ubuntu/Debian

Under ubuntu/debian those dependancies can be installed with the following command:

```
sudo apt-get install clang libbfd-dev uthash-dev libelf-dev libcapstone-dev
libreadline6 libreadline6-dev libiberty-dev libgsl-dev
```

Building and Installing:

Building WCC

From your root wcc directory, type:

```
make
```

Installing WCC

Then to install wcc, type:

```
sudo make install
```

Building the WCC documentation

WCC makes use of doxygen to generate its documentation. From the root wcc directory, type

```
make documentation
```

Presentations

The slides of the presentation given at the DEF CON 24 Conference in August 2016 are available at: https://github.com/endrazine/wcc/raw/master/doc/presentations/Jonathan_Brossard_Witchract_Compiler_Collection_Defcon24_2016.pdf

More demos

The source code of the all demos of the presentation given at DEF CON can be found here : https://github.com/endrazine/wcc/tree/master/doc/presentations/demos_defcon24_2016

Developper Manual

The Doxygen documentation of the Witchcraft Compiler Collection is available at: https://github.com/endrazine/wcc/raw/master/doc/WCC_internal_documentation.pdf

wcc : The Witchcraft Core Compiler

The wcc compiler takes binaries (ELF, PE, ...) as an input and creates valid ELF binaries as an output. It can be used to create relocatable object files from executables or shared libraries.

wcc command line options

```
jonathan@blackbox:~$ wcc
Witchcraft Compiler Collection (WCC) version:0.0.1    (01:47:53 Jul 29 2016)

Usage: wcc [options] file

options:

    -o, --output           <output file>
    -m, --march            <architecture>
    -e, --entrypoint       <0xaddress>
    -i, --interpreter      <interpreter>
    -p, --poison           <poison>
    -s, --shared
    -c, --compile
    -S, --static
    -x, --strip
    -X, --sstrip
    -E, --exec
    -C, --core
    -O, --original
    -D, --disasm
    -d, --debug
    -h, --help
    -v, --verbose
    -V, --version

jonathan@blackbox:~$
```

Options description

```
    -o, --output           <output file>
```

Speficy the desired output file name. Default: a.out

```
    -m, --march            <architecture>
```

Specify the desired output architecture. This option is ignored. Run the 64bit or the 32bit versions of wcc to produce 64 bits or 32 bits binaries respectively.

```
    -e, --entrypoint       <0xaddress>
```

Specify the address of the entry point as found in the ELF header manually.

```
    -i, --interpreter      <interpreter>
```

Specify a new program interpreter to be written to the interpreter segment of the output program.

```
    -p, --poison           <poison>
```

Specify a poison byte to be written in the unused bytes of the output file.

```
    -s, --shared
```

Produce a shared library.

```
    -c, --compile
```

Produce relocatable object files.

```
    -S, --static
```

Produce a static binary.

```
    -x, --strip
```

Do not use the Dynamic symbol table to unstrip the binary. Default: off.

```
    -X, --sstrip
```

Strip more.

```
    -E, --exec
```

Set binary type to ET_EXEC in the ELF header.

```
    -C, --core
```

Set binary type to a Core file in the ELF header.

```
    -O, --original
```

Copy original section headers from input file (which must be an ELF) instead of guessing them from bfd sections. Default: off.

```
    -D, --disasm
```

Display application disassembly.

```
    -d, --debug
```

Enable debug mode (very verbose).

```
    -h, --help
```

Display help.

```
    -v, --verbose
```

Be verbose.

```
    -V, --version
```

Display version number.

Example usage of wcc

The primary use of wcc is to "unlink" (undo the work of a linker) ELF binaries, either executables or shared libraries, back into relocatable shared objects. The following command line attempts to unlink the binary /bin/ls (from GNU binutils) into a relocatable file named /tmp/ls.o

```
jonathan@blackbox:~$ wcc -c /bin/ls -o /tmp/ls.o
jonathan@blackbox:~$
```

This relocatable file can then be used as if it had been directly produced by a compiler. The following command would use the gcc compiler to link /tmp/ls.o into a shared library /tmp/ls.so

```
jonathan@blackbox:~$ gcc /tmp/ls.o -o /tmp/ls.so -shared
jonathan@blackbox:~$
```

Limits of wcc

wcc will process any file supported by libbfd and produce ELF files that will contain the same mapping when relinked and executed. This includes PE or OSX COFF files in 32 or 64 bits. However, rebuilding relocations is currently supported only for Intel ELF x86_64 binaries. Transforming a PE into an ELF and invoking pure functions is for instance supported.

How does it work ?

wcc uses libbfd to parse the sections of the input binary, and generates an ELF file with the corresponding Sections and Segments. wcc also handles symbols and symbol tables and attempts to unstrip stripped binaries by parsing their dynamic symbol tables. Relocations are recreated as needed for ELF Intel x86_64 input files. Help on extending to other cpus and relocation types very welcome :)

What does the resulting /tmp/ls.o look like in details ?

In order to observe more closely the output of wcc, let's take a look at /tmp/ls.o as parsed by readelf (GNU binutils package) editted for brevity:

```
jonathan@blackbox:~$ readelf -a /tmp/ls.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          2348624 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         9
  Section header string table index: 8

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  0001ae00
       00000000002191ec  0000000000000000 WAX       0     0     16
  [ 2] .rodata           PROGBITS         0000000000000000  00011f20
       00000000000050fc  0000000000000000   A       0     0     32
  [ 3] .data             PROGBITS         0000000000000000  0001a3a0
       0000000000000254  0000000000000000  WA       0     0     32
```

```
  [ 4] .bss              NOBITS           0000000000000000  0001a5f4
       0000000000000d60  0000000000000000  WA       0     0     32
  [ 5] .rela.all         RELA             0000000000000000  00233fe0
       0000000000007158  0000000000000018   A       7     1     8
  [ 6] .strtab           STRTAB           0000000000000000  0023b138
       0000000000000dee  0000000000000000           0     0     1
  [ 7] .symtab           SYMTAB           0000000000000000  0023bf26
       00000000000016f8  0000000000000018           6     5     8
  [ 8] .shstrtab         STRTAB           0000000000000000  0023d890
       000000000000003e  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

There are no program headers in this file.

Relocation section '.rela.all' at offset 0x233fe0 contains 1209 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000217eb0  000600000001 R_X86_64_64       0000000000000000 __ctype_toupper_loc +
0
000000217eb8  000700000001 R_X86_64_64       0000000000000000 __uflow + 0
000000217ec0  000800000001 R_X86_64_64       0000000000000000 getenv + 0
000000217ec8  000900000001 R_X86_64_64       0000000000000000 sigprocmask + 0
000000217ed0  000a00000001 R_X86_64_64       0000000000000000 raise + 0
000000217ed8  007b00000001 R_X86_64_64       00000000004021f0 free + 0
000000217ee0  000b00000001 R_X86_64_64       0000000000000000 localtime + 0
000000217ee8  000c00000001 R_X86_64_64       0000000000000000 __mempcpy_chk + 0
000000217ef0  000d00000001 R_X86_64_64       0000000000000000 abort + 0
000000217ef8  000e00000001 R_X86_64_64       0000000000000000 __errno_location + 0
000000217f00  000f00000001 R_X86_64_64       0000000000000000 strncmp + 0
...
00000000091f  000400000002 R_X86_64_PC32     0000000000000000 .bss + abd
000000000971  000400000002 R_X86_64_PC32     0000000000000000 .bss + ac1
000000000976  00020000000a R_X86_64_32       0000000000000000 .rodata + 1924
000000000988  000400000002 R_X86_64_PC32     0000000000000000 .bss + acd
0000000009b6  000400000002 R_X86_64_PC32     0000000000000000 .bss + ad1
0000000009ce  00020000000a R_X86_64_32       0000000000000000 .rodata + 1160
0000000009d3  00020000000a R_X86_64_32       0000000000000000 .rodata + 3ca8
000000000a0b  000400000002 R_X86_64_PC32     0000000000000000 .bss + b3e
000000000a12  000400000002 R_X86_64_PC32     0000000000000000 .bss + b46
000000000a26  000400000002 R_X86_64_PC32     0000000000000000 .bss + b0d
000000000a2f  000400000002 R_X86_64_PC32     0000000000000000 .bss + b36
000000000a39  000400000002 R_X86_64_PC32     0000000000000000 .bss + b2a
...
000000000b25  008500000002 R_X86_64_PC32     0000000000000000 optarg - 4
000000000b45  000400000002 R_X86_64_PC32     0000000000000000 .bss + ad1
000000000b50  000400000002 R_X86_64_PC32     0000000000000000 .bss + b3e
00000000240f  008200000002 R_X86_64_PC32     0000000000000000 stderr - 4
...

The decoding of unwind sections for machine type Advanced Micro Devices X86-64 is
not currently supported.

Symbol table '.symtab' contains 245 entries:
```

```
    Num:    Value          Size Type    Bind   Vis      Ndx Name
      0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
      1: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
      2: 0000000000000000     0 SECTION LOCAL  DEFAULT    2 .rodata
      3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
      4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
      5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .unknown
      6: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __ctype_toupper_loc
      7: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __uflow
      8: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND getenv
      9: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND sigprocmask
     10: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND raise
     11: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND localtime
     12: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __mempcpy_chk
      ...
    132: 0000000000411efc     0 NOTYPE  WEAK   DEFAULT  UND old__fini
    133: 0000000000000000     8 OBJECT  GLOBAL DEFAULT  UND optarg
    134: 0000000000000000   100 FUNC    GLOBAL DEFAULT    1 old_plt
    135: 0000000000000738   100 FUNC    GLOBAL DEFAULT    1 old_text
    136: 00000000000104d5   100 FUNC    GLOBAL DEFAULT    1 old_text_end
    137: 000000000000b538   100 FUNC    GLOBAL DEFAULT    1 internal_0040d6a0
    138: 000000000000fd78   100 FUNC    GLOBAL DEFAULT    1 internal_00411ee0
    139: 000000000000c4d8   100 FUNC    GLOBAL DEFAULT    1 internal_0040e640
    140: 00000000000007ce8  100 FUNC    GLOBAL DEFAULT    1 internal_00409e50
    141: 000000000000ed28   100 FUNC    GLOBAL DEFAULT    1 internal_00410e90
    142: 000000000000ead8   100 FUNC    GLOBAL DEFAULT    1 internal_00410c40
    143: 00000000000075e8   100 FUNC    GLOBAL DEFAULT    1 internal_00409750
    144: 000000000000e9c8   100 FUNC    GLOBAL DEFAULT    1 internal_00410b30
    145: 0000000000007fb8   100 FUNC    GLOBAL DEFAULT    1 internal_0040a120
    146: 000000000000a6a8   100 FUNC    GLOBAL DEFAULT    1 internal_0040c810
    147: 000000000000c7c8   100 FUNC    GLOBAL DEFAULT    1 internal_0040e930
    148: 000000000000c498   100 FUNC    GLOBAL DEFAULT    1 internal_0040e600
    149: 000000000000c4c8   100 FUNC    GLOBAL DEFAULT    1 internal_0040e630
    150: 000000000000c4e8   100 FUNC    GLOBAL DEFAULT    1 internal_0040e650
    151: 0000000000002c68   100 FUNC    GLOBAL DEFAULT    1 internal_00404dd0
     ...
    241: 000000000000e958   100 FUNC    GLOBAL DEFAULT    1 internal_00410ac0
    242: 000000000000fbc8   100 FUNC    GLOBAL DEFAULT    1 internal_00411d30
    243: 000000000000fc48   100 FUNC    GLOBAL DEFAULT    1 internal_00411db0
    244: 000000000000fc88   100 FUNC    GLOBAL DEFAULT    1 internal_00411df0

No version information found in this file.
jonathan@blackbox:~$
```

It is worth in particular noticing that wcc rebuilt different types of relocations under the new .rela.all section. It also stripped the sections non essential to a relocatable object file from the input binary, and rebuilt a symbol table. On this last topic, it is also worth noticing that wcc created new symbols named internal_00XXXXXX where 0xXXXXXX is the address of a static function within the binary, not normally exported. Finally, wcc also makes used of additional symbol tables to find the address of additional functions if any are available (parsing both symbol tables and dynamic symbol tables).

wcch command line options

wcch takes a single mandatory argument : the path to an ELF executable or shared library.

```
wcch </path/to/binary>
```

wcch will generate minimal C header files suitable for compiling C code against the binary given as argument.

Example usage of wcch

The following command instructs wcch to generate C headers from the apache2 executable and redirects the output from the standard output to a file named /tmp/apache2.h ready for use as a header in a C application.

```
jonathan@blackbox:~$ wcch /usr/sbin/apache2 >/tmp/apache2.h
jonathan@blackbox:~$
```

Here is the actual content of the generated /tmp/apache2.h file, edited because of its large size:

```
/**
*
* Automatically generated by the Whitchcraft Compiler Collection 0.0.1
*
* 23:17:22 Jul 26 2016
*
*/


/**
* Imported objects
**/
extern void *_dlfcn_hook;
extern void *daylight;
extern void *_sys_nerr;
extern void *getdate_err;
extern void *__rcmd_errstr;
extern void *optind;
extern void *argp_program_version;
extern void *__free_hook;
extern void *__tzname;
extern void *__progname;
extern void *_environ;
...
extern void *ap_hack_ap_build_cont_config;
extern void *ap_hack_ap_find_etag_weak;
extern void *ap_hack_ap_hook_get_post_read_request;
extern void *ap_hack_apr_file_name_get;
extern void *ap_hack_apr_sdbm_unlock;
extern void *ap_hack_ap_is_rdirectory;
extern void *ap_hack_ap_request_has_body;
```

```
extern void *ap_hack_apr_pool_cleanup_run;
extern void *ap_hack_ap_hook_get_type_checker;
extern void *ap_hack_apr_global_mutex_pool_get;
extern void *ap_hack_apr_file_data_set;
extern void *ap_hack_ap_hook_get_child_status;
extern void *ap_hack_ap_set_server_protocol;
extern void *ap_hack_apr_hash_make_custom;
extern void *ap_hack_ap_malloc;
extern void *ap_hack_ap_pool_cleanup_set_null;
extern void *ap_hack_apr_dbm_firstkey;
extern void *ap_hack_apr_strmatch_precompile;
...


/**
* Imported functions
**/
void *dlclose();
void *dlinfo();
void *dladdr1();
void *dlsym();
void *dladdr();
void *dlopen();
void *dlmopen();
void *dlerror();
void *dlvsym();
void *putwchar();
void *__strspn_c1();
void *__gethostname_chk();
void *__strspn_c2();
void *setrpcent();
void *__wcstod_l();
void *__strspn_c3();
void *epoll_create();
void *sched_get_priority_min();
void *__getdomainname_chk();
void *klogctl();
void *__tolower_l();
void *dprintf();
void *setuid();
...
void *ap_mpm_pod_killpg();
void *ap_register_hooks();
void *ap_remove_output_filter_byhandle();
void *ap_hook_create_request();
void *ap_expr_exec_ctx();
void *ap_send_http_options();
void *ap_mpm_set_max_requests();
void *ap_os_escape_path();
void *ap_file_walk();
void *ap_build_cont_config();
void *ap_start_lingering_close();
void *ap_hook_generate_log_id();
void *ap_varbuf_cfg_getline();
void *ap_hook_test_config();
void *ap_fcgi_header_to_array();
void *ap_http_chunk_filter();
```

```
void *ap_random_insecure_bytes();
void *ap_pcfg_open_custom();
void *ap_hook_get_auth_checker();
void *ap_expr_yyfree();
...
void *uuid_copy();
void *uuid_generate();
```

The functions prototypes and imported objects cover all of the API exported by executables and shared libraries including their recursive dependancies. All the programmable API in the address space. #Witchcraft

How is this useful ?

Both gcc and clang will happily use the above mention function prototypes when compiling C code making use of them instead of issuing errors due to missing function prototypes. This is a great feature : it means we can now call those functions from C without actually knowing their exact prototypes (such as arguments number and types).

wld : The Witchcraft Linker.

wld takes an ELF executable as an input and modifies it to create a shared library.

wld command line options

```
jonathan@blackbox:~$ wld
Witchcraft Compiler Collection (WCC) version:0.0.1    (23:11:13 Jul 21 2016)

Usage: wld [options] file

options:

    -libify          Set Class to ET_DYN in input ELF file.

jonathan@blackbox:~$
```

Example usage of wld

The following example libifies the executable /bin/ls into a shared library named /tmp/ls.so.

```
jonathan@blackbox:~$ cp /bin/ls /tmp/ls.so
jonathan@blackbox:~$ wld -libify /tmp/ls.so
jonathan@blackbox:~$
```

Limits of wld

wld currently only works on ELF binaries. However wld can process ELF executables irrelevant of their architecture or operating system. wld could for instance process Intel, ARM or SPARC executables from Android, Linux, BSD or UNIX operating systems and transform them into "non relocatable shared libraries". Feel free to refer to the documentation under the /doc directory for more ample details.

Do I even need wld ?

If the ELF executable you whish to work with has been compiled with as Position Independant Executable (-pie -fpie compiler flags with gcc or clang), it already is a functional shared library and doesn't need to be libified. In particular, its ELF header is already set to ET_DYN.

Here is an example executable that is of type ET_EXEC and can be libified. Mind the Type field set to EXEC:

```
jonathan@blackbox:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=8d0966ce81ec6609bbf4aa439c77138e2f48a471, stripped
jonathan@blackbox:~$ readelf -h /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
```

```
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x404890
  Start of program headers:          64 (bytes into file)
  Start of section headers:          108288 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         28
  Section header string table index: 27
jonathan@blackbox:~$
```

Here is an exemple binary compiled as Position Independant Executable and not requiring libification to be used as a shared library or loaded in wsh. Mind the Type field set to DYN:

```
jonathan@blackbox:~$ file /usr/sbin/apache2
/usr/sbin/apache2: ELF 64-bit LSB  shared object, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=02c74092325980f41ca3e1c2995daec1f3b30ea2, stripped
jonathan@blackbox:~$ readelf -h /usr/sbin/apache2
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x37156
  Start of program headers:          64 (bytes into file)
  Start of section headers:          635736 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         28
  Section header string table index: 27
jonathan@blackbox:~$
```

Finally, here is what a libified shared library looks like. The Type field has been set to DYN by wld during the libification process:

```
jonathan@blackbox:~$ file /tmp/ls.so
/tmp/ls.so: ELF 64-bit LSB  shared object, x86-64, version 1 (SYSV), dynamically
linked, BuildID[sha1]=04fd28208b659339be2711ea5f6d3485b6117da6, not stripped
jonathan@blackbox:~$ readelf -h /tmp/ls.so
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x6200
  Start of program headers:          64 (bytes into file)
  Start of section headers:          2261504 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         6
  Size of section headers:           64 (bytes)
  Number of section headers:         27
  Section header string table index: 24
jonathan@blackbox:~$
```

wldd : print shared libraries compilation flags

When compiling C code, it is often required to pass extra arguments to the compiler to signify which shared libraries should explicitely linked against the compile code. Figuring out those compilation parameters can be cumbersome. The wldd commands displays the shared libraries compilation flags given at compile time for any given ELF binary.

wldd command line options

```
jonathan@blackbox:~$ wldd
Usage: /usr/bin/wldd </path/to/bin>

  Returns libraries to be passed to gcc to relink this application.

jonathan@blackbox:~$
```

Example usage of wldd

On /bin/ls (ET_EXEC ELF executable)

The following command displays shared libraries compilation flags as passed to gcc when compiling /bin/ls from GNU binutils:

```
jonathan@blackbox:~$ wldd /bin/ls
-lselinux -lacl -lc -lpcre -ldl -lattr
jonathan@blackbox:~$
```

On apache2 (ET_DYN ELF executable compiled as Position Independant Executable / full ASLR)

The following command displays the compilation flags relative to shared libraries used when compiling /usr/sbin/apache2:

```
jonathan@blackbox:~$ wldd /usr/sbin/apache2
-lpcre -laprutil-1 -lapr-1 -lpthread -lc -lcrypt -lexpat -luuid -ldl
jonathan@blackbox:~$
```

On the openssl shared library

This command can also be ran on shared libraries. The following example displays the same compiler options for the openssl shared library:

```
jonathan@blackbox:~$ wldd /usr/lib/x86_64-linux-gnu/libssl.so.0.9.8
-lcrypto -lc -ldl -lz
jonathan@blackbox:~$
```

Security Caveat

wldd invokes binutils' ldd which in turns loads the binary passed as an argument using its hardcoded dynamic linker. This does run code inside the analysed binary. As such, running wldd on potentially hostile code (eg: malware) is not safe.

Note: We could get the name of the shared libraries linked with this binary from the content of its .dynamic section without having to rely on ldd nor run the binary. That would be very useful. It would also produce a non recursive answer (unlike wldd currenty), which would reflect more the actual linking of the binary. Feel free to implement it :)

wsh : The Witchcraft shell

The witchcraft shell accepts ELF shared libraries, ELF ET_DYN executables and Witchcraft Shell Scripts written in Punk-C as an input. It loads all the executables in its own address space and make their API available for programming in its embedded interpreter. This provides for binaries functionalities similar to those provided via reflection on languages like Java.

wsh command line options

```
jonathan@blackbox:~$ wsh -h
Usage: wsh [script] [options] [binary1] [binary2] ... [-x] [script_arg1]
[script_arg2] ...

Options:

    -x, --args                Optional script argument separator.
    -v, --verbose
    -V, --version

Script:

    If the first argument is an existing file which is not a known binary file
format,
    it is assumed to be a lua script and gets executed.

Binaries:

    Any binary file name before the -x tag gets loaded before running the script.
    The last binary loaded is the main binary analyzed.

jonathan@blackbox:~$
```

Example usage of wsh

The following command loads the /usr/sbin/apache2 executable within wsh, calls the ap_get_server_banner() function within apache to retreive its banner and displays it within the wsh intterpreter.

```
jonathan@blackbox:~$ wsh /usr/sbin/apache2
> a = ap_get_server_banner()
> print(a)
Apache/2.4.7
>
```

To get help at any time from the wsh interpreter, simply type help. To get help on a particular topic, type help("topic").

The following example illustrates how to display the main wsh help from the interpreter and how to get detailed help on the grep command by calling help("grep") from the wsh interpreter.

```
> help
  [Shell commands]

    help, quit, exit, shell, exec, clear

  [Functions]

 + basic:
    help(), man()

 + memory display:
     hexdump(), hex_dump(), hex()

 + memory maps:
    shdrs(), phdrs(), map(), procmap(), bfmap()

 + symbols:
    symbols(), functions(), objects(), info(), search(), headers()

 + memory search:
    grep(), grepptr()

 + load libaries:
    loadbin(), libs(), entrypoints(), rescan()

 + code execution:
    libcall()

 + buffer manipulation:
    xalloc(), ralloc(), xfree(), balloc(), bset(), bget(), rdstr(), rdnum()

 + control flow:
     breakpoint(), bp()

 + system settings:
    enableaslr(), disableaslr()

 + settings:
     verbose(), hollywood()

 + advanced:
    ltrace()

Try help("cmdname") for detailed usage on command cmdname.

> help("grep")

    WSH HELP FOR FUNCTION grep


NAME

    grep

SYNOPSIS

    table match = grep(<pattern>, [patternlen], [dumplen], [before])
```

```
DESCRIPTION

    Search <pattern> in all ELF sections in memory. Match [patternlen] bytes, then
display [dumplen] bytes, optionally including [before] bytes before the match.
Results are displayed in enhanced decimal form

RETURN VALUES

    Returns 1 lua table containing matching memory addresses.


>
```

Extending wsh with Witchcraft Shell Scripts

The combination of a full lua interpreter in the same address space as the loaded executables and shared libraries in combination with the reflection like capabilities of wsh allow to call any function loaded in the address space from the wsh interpreter transparently. The resulting API, a powerfull combination of lua and C API is called Punk-C. Wsh is fully scriptable in Punk-C, and executes Punk-C on the fly via its dynamic interpreter. Scripts in Punk C can be invoked by specifying the full path to wsh in the magic bytes of a wsh shell. The following command displays the content of a Witchcraft shell script:

```
jonathan@blackbox:/usr/share/wcc/scripts$ cat md5.wsh
#!/usr/bin/wsh

-- Computing a MD5 sum using cryptographic functions from foreign binaries (eg:
sshd/OpenSSL)

function str2md5(input)

    out = calloc(33, 1)
    ctx = calloc(1024, 1)

    MD5_Init(ctx)
    MD5_Update(ctx, input, strlen(input))
    MD5_Final(out, ctx)

    free(ctx)
    return out
end

input = "Message needing hashing\n"
hash = str2md5(input)
hexdump(hash,16)

exit(0)
jonathan@blackbox:/usr/share/wcc/scripts$
```

To run this script using the API made available inside the address space of sshd, simply run:

```
jonathan@blackbox:/usr/share/wcc/scripts$ ./md5.wsh /usr/sbin/sshd
0x43e8b280    d6 fc 46 91 b0 6f ab 75 4d 9c a7 58 6d 9c 7e 36    V|F.0o+uM.'Xm.~6
jonathan@blackbox:/usr/share/wcc/scripts$
```

Limits of wsh

wsh can only load shared libraries and ET_DYN dynamically linked ELF executables directly. This means ET_EXEC executables may need to be libified using wld before use in wsh. Binaries in other file formats might need to be turned into ELF files using wcc.

Analysing and Executing ARM/SPARC/MIPS binaries "natively" on Intel x86_64 cpus via JIT binary translation

wsh can be cross compiled to ARM, SPARC, MIPS and other plateforms and used in association with the qemu's user space emulation mode to provide JIT binary translation on the fly and analyse shared libraries and binaries from other cpus without requiring emulation a full operating system in a virtual machine. On the the analyzed binaries are translated from one CPU to an other, and the analysed binaries, the wsh cross compiled analyser and the qemu binary translator share the address space of a single program. This significantly diminishes the complexity of analysing binaries accross different hardware by seemingly allowing to run ARM or SPARC binaries on a linux x86_64 machine natively and transparently.

Core API Overview

basic functions

```
help(), man()
```

memory display functions

```
 hexdump(), hex_dump(), hex()
```

memory maps functions

```
shdrs(), phdrs(), map(), procmap(), bfmap()
```

symbols functions

```
symbols(), functions(), objects(), info(), search(), headers()
```

memory search functions

```
grep(), grepptr()
```

load libaries functions

```
loadbin(), libs(), entrypoints(), rescan()
```

code execution functions

```
libcall()
```

buffer manipulation functions

```
xalloc(), ralloc(), xfree(), balloc(), bset(), bget(), rdstr(), rdnum()
```

control flow functions

```
 breakpoint(), bp()
```

system settings functions

```
enableaslr(), disableaslr()
```

settings functions

```
 verbose(), hollywood()
```

advanced functions

```
ltrace()
```

API speficitations

function help()

```
    WSH HELP FOR FUNCTION help


NAME

    help

SYNOPSIS

    help([topic])

DESCRIPTION

    Display help on [topic]. If [topic] is ommitted, display general help.

RETURN VALUES

    None
```

function man()

```
    WSH HELP FOR FUNCTION man


NAME

    man

SYNOPSIS

    man([page])

DESCRIPTION

    Display system manual page for [page].
```

```
RETURN VALUES

    None
```

function hexdump()

```
    WSH HELP FOR FUNCTION hexdump


NAME

    hexdump

SYNOPSIS

    hexdump(<address>, <num>)

DESCRIPTION

    Display <num> bytes from memory <address> in enhanced hexadecimal form.

RETURN VALUES

    None
```

function hex_dump()

```
    WSH HELP FOR FUNCTION hex_dump


NAME

    hex

SYNOPSIS

    hex(<object>)

DESCRIPTION

    Display lua <object> in enhanced hexadecimal form.

RETURN VALUES

    None
```

function hex()

```
    WSH HELP FOR FUNCTION hex


NAME
```

```
    hex

SYNOPSIS

    hex(<object>)

DESCRIPTION

    Display lua <object> in enhanced hexadecimal form.

RETURN VALUES

    None
```

function shdrs()

```
    WSH HELP FOR FUNCTION shdrs


NAME

    shdrs

SYNOPSIS

    shdrs()

DESCRIPTION

    Display ELF section headers from all binaries loaded in address space.

RETURN VALUES

    None
```

function phdrs()

```
    WSH HELP FOR FUNCTION phdrs


NAME

    phdrs

SYNOPSIS

    phdrs()

DESCRIPTION

    Display ELF program headers from all binaries loaded in address space.

RETURN VALUES
```

```
    None
```

function map()

```
    WSH HELP FOR FUNCTION map


NAME

    map

SYNOPSIS

    map()

DESCRIPTION

    Display a table of all the memory ranges mapped in memory in the address space.

RETURN VALUES

    None
```

function procmap()

```
    WSH HELP FOR FUNCTION procmap


NAME

    procmap

SYNOPSIS

    procmap()

DESCRIPTION

    Display a table of all the memory ranges mapped in memory in the address space
as displayed in /proc/<pid>/maps.

RETURN VALUES

    None
```

function bfmap()

```
    WSH HELP FOR FUNCTION bfmap


NAME
```

```
    bfmap

SYNOPSIS

    bfmap()

DESCRIPTION

    Bruteforce valid mapped memory ranges in address space.

RETURN VALUES

    None
```

function symbols()

```
    WSH HELP FOR FUNCTION symbols


NAME

    symbols

SYNOPSIS

    symbols([sympattern], [libpattern], [mode])

DESCRIPTION

    Display all the symbols in memory matching [sympattern], from library
[libpattern]. If [mode] is set to 1 or 2, do not wait user input between pagers.
[mode] = 2 provides a shorter output.

RETURN VALUES

    None
```

function functions()

```
    WSH HELP FOR FUNCTION functions


NAME

    functions

SYNOPSIS

    table func = functions([sympattern], [libpattern], [mode])

DESCRIPTION

    Display all the functions in memory matching [sympattern], from library
[libpattern]. If [mode] is set to 1 or 2, do not wait user input between pagers.
```

```
[mode] = 2 provides a shorter output.

RETURN VALUES

    Return 1 lua table _func_ whose keys are valid function names in address space,
and values are pointers to them in memory.
```

function objects()

```
    WSH HELP FOR FUNCTION objects


NAME

    objects

SYNOPSIS

    objects([pattern])

DESCRIPTION

    Display all the functions in memory matching [sympattern]

RETURN VALUES

    None
```

function info()

```
    WSH HELP FOR FUNCTION info


NAME

    info

SYNOPSIS

    info([address] | [name])

DESCRIPTION

    Display various informations about the [address] or [name] provided : if it is
mapped, and if so from which library and in which section if available.

RETURN VALUES

    None
```

function search()

```
    WSH HELP FOR FUNCTION search

NAME
    search
SYNOPSIS
    search(<pattern>)
DESCRIPTION
    Search all object names matching <pattern> in address space.
RETURN VALUES
    None
```

function headers()

```
    WSH HELP FOR FUNCTION headers

NAME
    headers
SYNOPSIS
    headers()
DESCRIPTION
    Display C headers suitable for linking against the API loaded in address space.
RETURN VALUES
    None
```

function grep()

```
    WSH HELP FOR FUNCTION grep

NAME
    grep
SYNOPSIS
    table match = grep(<pattern>, [patternlen], [dumplen], [before])
```

```
DESCRIPTION

    Search <pattern> in all ELF sections in memory. Match [patternlen] bytes, then
display [dumplen] bytes, optionally including [before] bytes before the match.
Results are displayed in enhanced decimal form

RETURN VALUES

    Returns 1 lua table containing matching memory addresses.
```

function grepptr()

```
    WSH HELP FOR FUNCTION grepptr


NAME

    grep

SYNOPSIS

    table match = grep(<pattern>, [patternlen], [dumplen], [before])

DESCRIPTION

    Search <pattern> in all ELF sections in memory. Match [patternlen] bytes, then
display [dumplen] bytes, optionally including [before] bytes before the match.
Results are displayed in enhanced decimal form

RETURN VALUES

    Returns 1 lua table containing matching memory addresses.
```

function loadbin()

```
    WSH HELP FOR FUNCTION loadbin


NAME

    loadbin

SYNOPSIS

    loadbin(<pathname>)

DESCRIPTION

    Load binary to memory from <pathname>.

RETURN VALUES

    None
```

function libs()

```
    WSH HELP FOR FUNCTION libs


NAME

    libs

SYNOPSIS

    table libraries = libs()

DESCRIPTION

    Display all libraries loaded in address space.

RETURN VALUES

    Returns 1 value: a lua table _libraries_ whose values contain valid binary
names (executable/libraries) mapped in memory.
```

function entrypoints()

```
    WSH HELP FOR FUNCTION entrypoints


NAME

    entrypoints

SYNOPSIS

    entrypoints()

DESCRIPTION

    Display entry points for each binary loaded in address space.

RETURN VALUES

    None
```

function rescan()

```
    WSH HELP FOR FUNCTION rescan


NAME

    rescan
```

```
SYNOPSIS

    rescan()

DESCRIPTION

    Re-perform address space scan.

RETURN VALUES

    None
```

function libcall()

```
    WSH HELP FOR FUNCTION libcall


NAME

    libcall

SYNOPSIS

    void *ret, table ctx = libcall(<function>, [arg1], [arg2], ... arg[6])

DESCRIPTION

    Call binary <function> with provided arguments.

RETURN VALUES

    Returns 2 return values: _ret_ is the return value of the binary function (nill
if none), _ctx_ a lua table representing the execution context of the library call.
```

function xalloc()

```
No help available for function xalloc()
```

function ralloc()

```
No help available for function ralloc()
```

function xfree()

```
No help available for function xfree()
```

function balloc()

```
No help available for function balloc()
```

function bset()

```
No help available for function bset()
```

function bget()

```
No help available for function bget()
```

function rdstr()

```
No help available for function rdstr()
```

function rdnum()

```
No help available for function rdnum()
```

function breakpoint()

```
    WSH HELP FOR FUNCTION breakpoint


NAME

    breakpoint

SYNOPSIS

    breakpoint(<address>, [weight])

DESCRIPTION

    Set a breakpoint at memory <address>. Optionally add a <weight> to breakpoint
score if hit.

RETURN VALUES

    None
```

function bp()

```
    WSH HELP FOR FUNCTION bp


NAME

    bp
```

```
SYNOPSIS

    bp(<address>, [weight])

DESCRIPTION

    Set a breakpoint at memory <address>. Optionally add a <weight> to breakpoint
score if hit. Alias for breakpoint() function.

RETURN VALUES

    None
```

function enableaslr()

```
    WSH HELP FOR FUNCTION enableaslr


NAME

    enableaslr

SYNOPSIS

    enableaslr()

DESCRIPTION

    Enable Address Space Layout Randomization (requires root privileges).

RETURN VALUES

    None
```

function disableaslr()

```
    WSH HELP FOR FUNCTION disableaslr


NAME

    disableaslr

SYNOPSIS

    disableaslr()

DESCRIPTION

    Disable Address Space Layout Randomization (requires root privileges).

RETURN VALUES

    None
```

function verbose()

```
    WSH HELP FOR FUNCTION verbose


NAME

    verbose

SYNOPSIS

    verbose(<verbosity>)

DESCRIPTION

    Change verbosity setting to <verbosity>.

RETURN VALUES

    None
```

function hollywood()

```
    WSH HELP FOR FUNCTION hollywood


NAME

    hollywood

SYNOPSIS

    hollywood(<level>)

DESCRIPTION

    Change hollywood (fun) display setting to <level>, impacting color display
(enable/disable).

RETURN VALUES

    None
```

The following commands are built into wsh

help

Simply typing help in the wsh interpreter displays the following help

```
> help
  [Shell commands]

    help, quit, exit, shell, exec, clear

  [Functions]

 + basic:
    help(), man()

 + memory display:
     hexdump(), hex_dump(), hex()

 + memory maps:
    shdrs(), phdrs(), map(), procmap(), bfmap()

 + symbols:
    symbols(), functions(), objects(), info(), search(), headers()

 + memory search:
    grep(), grepptr()

 + load libaries:
    loadbin(), libs(), entrypoints(), rescan()

 + code execution:
    libcall()

 + buffer manipulation:
    xalloc(), ralloc(), xfree(), balloc(), bset(), bget(), rdstr(), rdnum()

 + control flow:
     breakpoint(), bp()

 + system settings:
    enableaslr(), disableaslr()

 + settings:
     verbose(), hollywood()

 + advanced:
    ltrace()

Try help("cmdname") for detailed usage on command cmdname.

>
```

The advanced help for help follow:

```
> help("help")

    WSH HELP FOR FUNCTION help


NAME

    help

SYNOPSIS

    help([topic])

DESCRIPTION

    Display help on [topic]. If [topic] is ommitted, display general help.

RETURN VALUES

    None


>
```

quit

The quit command terminates the main wsh process and exits the wsh interpreter.

Here is the help page for quit

```
> help("quit")

    WSH HELP FOR COMMAND quit


NAME

    quit

SYNOPSIS

    quit

DESCRIPTION

    Exit wsh.

RETURN VALUES

    Does not return : exit wsh



>
```

exit

The exit command behaves much like the quit command.

Here is the detailed help for the exit command:

```
> help("exit")

    WSH HELP FOR COMMAND exit


NAME

    exit

SYNOPSIS

    exit

DESCRIPTION

    Exit wsh.

RETURN VALUES

    Does not return : exit wsh



>
```

Note on the exit command versus exit() function

It is worth noticing that typing exit(0) in the terminal does something different entirely : this will result in calling the function exit(), typically from the C library, with the parameter 0.

shell

The shell command instanciates an instance of /bin/sh from the wsh interpreter. Terminating the /bin/sh session will allow returning in the parent wsh session.

```
> help("shell")

    WSH HELP FOR COMMAND shell


NAME

    shell

SYNOPSIS
```

```
    shell [command]

DESCRIPTION

    Run a /bin/sh shell.

RETURN VALUES

    None. Returns uppon shell termination.


>
```

example usage of the shell command

From the wsh interpreter, the following commands start a /bin/sh shell, run the /bin/id application from this shell, and finally calls exit, which terminates the /bin/sh session and returns into the wsh interpreter.

```
> shell
$ id
uid=1001(jonathan) gid=1001(jonathan) groups=1001(jonathan)
$ exit
>
```

exec

The exec command allows running an external command from the wsh interpreter.

Here is the detailed help page for the exec command :

```
> help("exec")

    WSH HELP FOR COMMAND exec


NAME

    exec

SYNOPSIS

    exec <command>

DESCRIPTION

    Run <command> via the system() library call.

RETURN VALUES

    None. Returns uppon <command> termination.
```

```
>
```

Example usage of the exec command

The following command exemplifies calling the uname system utility with the "-a" argument:

```
> exec uname -a
Linux blackbox 3.13.0-68-generic #111-Ubuntu SMP Fri Nov 6 18:17:06 UTC 2015 x86_64
x86_64 x86_64 GNU/Linux
>
```

clear

The clear command clears the terminal. Its detailed help follows:

```
> help("clear")

    WSH HELP FOR COMMAND clear


NAME

    clear

SYNOPSIS

    clear

DESCRIPTION

    Clear terminal.

RETURN VALUES

    None.


>
```

Disclaimer

If you are an academic C teacher, your feelings may be hurt by what you are going to read in this page and what we are doing to your very dear and beautiful language for the purpose of binary wizardry. #Enjoy

What is Punk-C ?

Punk-C is the language wsh implements by extending a core lua interpreter with the API "reflected" from all the executables and shared libraries loaded in its address space.

How is Punk C different from C ?

Punk C is not compiled but interpreted. Punk C has no types declarations, does not enforce functions prototypes (wtf?) nor any of the notorious C nightmares. Think C without the problems.

The control statements such as loop iterrators are inherited from lua and do not ressemble those of C.

Note/TODO: Can we hack this last statement by modifying the lua grammars ? :)

What is lua ?

Lua is an amazing open source programming language and implementation. Its interpreter is very tiny yet very powerful. For more information on the Lua language, feel free to visit : https://www.lua.org/

How does binary "reflection" work ?

We use quotes around the word "reflected" because strictly speaking there is no Virtual Machine. wsh and the loaded programs share the same address space. The functionality is made possible by parsing the struct link_map returned by dlopen() when loading a binary. It alows in particular dumping all the symbols known by the dynamic linker and their respective addresses in the address space. This allows providing reflection like functionalities on raw binaries.

From a user perspective, this mechanism is transparent. We can call all of the C API present in memory directly from lua. In particular pass arguments to a C function and retrieve its return value.

Punk-C by example

The following commands examplify how to start wsh by loading the OpenSSH in memory from the path /usr/sbin/sshd. Wsh is then instructed to call the getpid() and getenv() functions and print their results. Those two functions do not exist in the Lua API : they are really made available directly from the libc by wsh's reflection mechanism.

```
jonathan@blackbox:~$ wsh /usr/sbin/sshd
```

```
> a = getpid()
> print(a)
22453
> b = getenv("PWD")
> print(b)
/home/jonathan
> exit(3)
jonathan@blackbox:~$ echo $?
3
jonathan@blackbox:~$
```

It is worth noticing that the exit() function was too called here via reflection from the C library loaded as an OpenSSH server depedency, and its parameter returned to the parent shell as expected.

Example witchcraft shell scripts

If you installed the Witchrcaft Compiler Collection on your computer, the directory /usr/share/wcc/scripts should contain example scripts.

Let's take a look at the following script:

```
jonathan@blackbox:/usr/share/wcc/scripts$ cat read.wsh
#!/usr/bin/wsh

 fname="/etc/passwd"
 printf("\n ** Reading file %s\n", fname)
 mem = malloc(1024)
 nread = read(open(fname), mem, 100) -- Composition works
 printf(" ** Displaying content (%u bytes) of file %s:\x0a\x0a%s\n", nread, fname,
mem)
 free(mem)
 c = close(fd)
 exit(0);

jonathan@blackbox:/usr/share/wcc/scripts$
```

Conventionally, wsh scripts names end with the ".wsh" extension.

This script attempts to open the /etc/password file and read 100 bytes of its content into a buffer of 1024 bytes pre allocated in the heap. This content is then displayed, the allocated heap memory freed and the opened file descriptor closed, before exiting with return value 0 (success, no errors).

The first line of the script instructs the linux kernel where to find the interpreter to execute it. We set this line to the full path of wsh.

A few things are worth noticing : the open function is only given one parameter when the POSIX standard specifies 2 or 3 :

```
Posix prototypes for function open():

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

This is made possible by the fact that wsh doesn't need to know the exact type of a function to craft arguments to call it and invoke it. Non provided arguments are implicitely casted to the value 0.

It is also worth noticing that arguments have no explicit types. This is made possible by the Lua typing mechanism.

Comments start with the "--" marker, and end with the line return as in lua.

Running a Witchraft shell script as a wsh argument

Let us now call this script with wsh, using sshd (and its dependancies) as the API provided for all the functions we will use:

```
jonathan@blackbox:/usr/share/wcc/scripts$ wsh ./read.wsh /usr/sbin/sshd

 ** Reading file /etc/passwd
 ** Displaying content (100 bytes) of file /etc/passwd:

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/
jonathan@blackbox:/usr/share/wcc/scripts$
```

We just invoked C functions from wsh dynamically, without compilation, and without knowing their prototypes ! #Witchcraft

Running a Witchcraft shell script as a standalone program

Let us start by making the read.wsh script executable:

```
jonathan@blackbox:/usr/share/wcc/scripts$ sudo chmod +x read.wsh
jonathan@blackbox:/usr/share/wcc/scripts$
```

We can now execute this script on any ELF executable or shared library by passing it as an argument to the script:

```
jonathan@blackbox:/usr/share/wcc/scripts$ ./read.wsh /usr/sbin/sshd

 ** Reading file /etc/passwd
 ** Displaying content (100 bytes) of file /etc/passwd:

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

```
bin:x:2:2:bin:/bin:/
jonathan@blackbox:/usr/share/wcc/scripts$
```

Registering a custom binfmt for .wsh scripts

Linux allows to define a binfmt so that the interpreter path can be ommited in wsh scripts. Any file named with the ".wsh" extension and executed will then be executed via the wsh interpreter automatically.

This is achieve via the following command:

```
sudo update-binfmts --package wsh --install wsh /usr/bin/wsh --extension wsh
```

You can verify if this command worked by viewing the corresponding entry under /proc :

```
jonathan@blackbox:~$ cat /proc/sys/fs/binfmt_misc/wsh
enabled
interpreter /usr/bin/wsh
flags:
extension .wsh
jonathan@blackbox:~$
```

We can now run .wsh scripts directly within wsh without specifying an interpreter :

```
jonathan@blackbox:~$ echo 'printf("Hello %s !\n", "World"); exit(3);'
>/tmp/hello.wsh
jonathan@blackbox:~$
jonathan@blackbox:~$ chmod +x /tmp/hello.wsh
jonathan@blackbox:~$
jonathan@blackbox:~$ cat /tmp/hello.wsh
printf("Hello %s !\n", "World"); exit(3);
jonathan@blackbox:~$
jonathan@blackbox:~$ /tmp/hello.wsh /usr/sbin/apache2
Hello World !
jonathan@blackbox:~$ echo $?
3
jonathan@blackbox:~$
```