Obfuscation of Executable Code.

(Really, Really, Really Briefly)

Edd Barrett

Computing Lab
University of Kent

December 7, 2009

1. Reverse Engineering in General
   - A definition
   - For Example
   - Why do People Reverse Software?
2. Where Does Executable Code Come From?
   - Code From Compilers
   - Other Ways Code is Made
3. First Steps in Reverse Engineering
   - The Disassembler
   - Disassembler Output
   - Slicing
4. Rounding Up
   - It is Never That Easy!
   - Conclusions

A definition

The act of determining the function of an “article” without prior knowledge or documentation on it’s inner workings.

For Example

- You could reverse engineer a recipe.

For Example

- You could reverse engineer a recipe.
  - http://www.fourmilab.ch/chez-nuke/SubMarie/
    Reverse Engineering Marie's Chunky Blue Cheese Dressing.

For Example

- You could reverse engineer a recipe.
  - `http://www.fourmilab.ch/chez-nuke/SubMarie/` Reverse Engineering Marie's Chunky Blue Cheese Dressing.
- You might try to reverse engineer a mechanical device.
  - Eg. To make cheap replacement parts when they become discontinued.

For Example

- You could reverse engineer a recipe.
  - `http://www.fourmilab.ch/chez-nuke/SubMarie/` Reverse Engineering Marie's Chunky Blue Cheese Dressing.
- You might try to reverse engineer a mechanical device.
  - Eg. To make cheap replacement parts when they become discontinued.
- Some people reverse engineer software.

Why do People Reverse Software?

- To break copyright/license restrictions. “Cracking”.

Why do People Reverse Software?

- ~~To break copyright/license restrictions. “Cracking”.~~
  - (I am not interested in that).

Why do People Reverse Software?

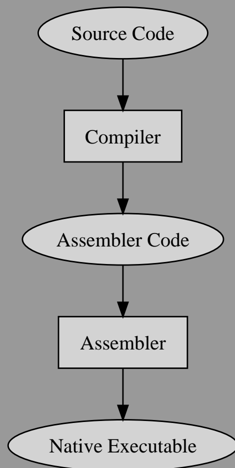
- ~~To break copyright/license restrictions. “Cracking”.~~
  - (I am not interested in that).
- To recover the functionality of a program whose source code was lost.
  - It happens!

Why do People Reverse Software?

- ~~To break copyright/license restrictions. “Cracking”.~~
  - (I am not interested in that).
- To recover the functionality of a program whose source code was lost.
  - It happens!
- For security purposes.
  - Is the software “malware”?

1. Reverse Engineering in General
   - A definition
   - For Example
   - Why do People Reverse Software?
2. Where Does Executable Code Come From?
   - Code From Compilers
   - Other Ways Code is Made
3. First Steps in Reverse Engineering
   - The Disassembler
   - Disassembler Output
   - Slicing
4. Rounding Up
   - It is Never That Easy!
   - Conclusions

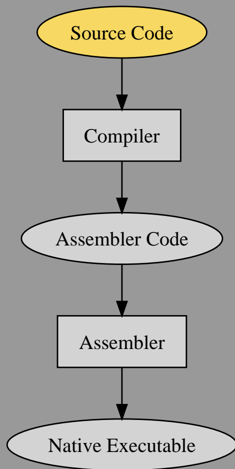
Code From Compilers



A simplified view of the production of a native executable using a compiler.

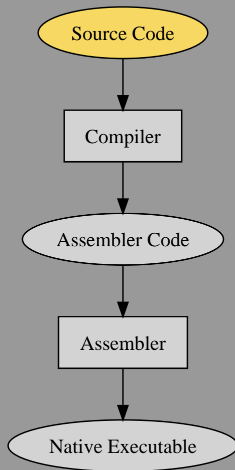
NOTE*:
I am not talking about byte-code.

Code From Compilers



You start with source code, for example C, C++ or Fortran.

Code From Compilers

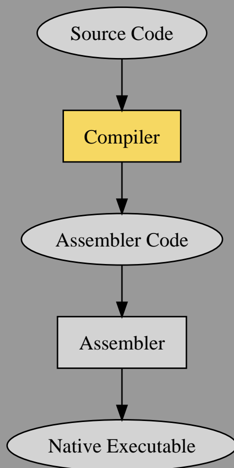


You start with source code, for example C, C++ or Fortran.

```
#include <stdio.h>

int
main(void)
{
	do_stuff();
	do_more_stuff();
}
```

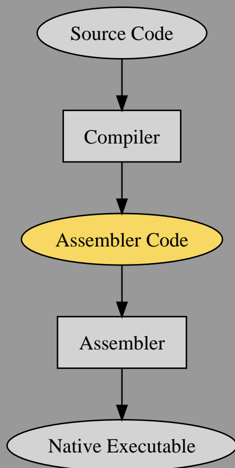
Code From Compilers



Source is compiled, for example with GCC or Visual Studio.

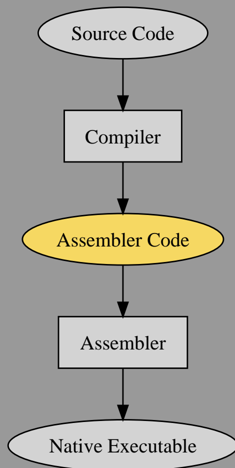
- Parser.
- Semantic checks.
- Some optimisation.

Code From Compilers



The resulting code is “assembler code”

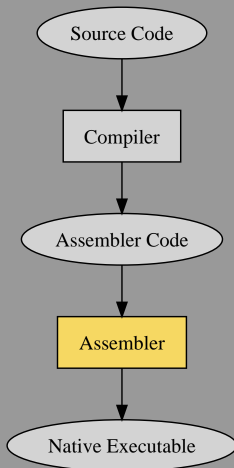
Code From Compilers



The resulting code is "assembler code"

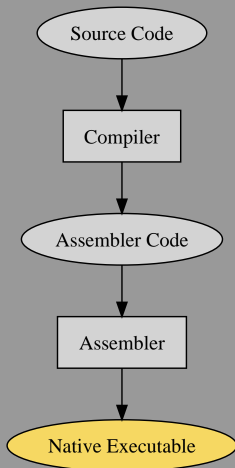
```
bgu   %icc, 100cac
srl   %o0, 0, %g1
sethi  %hi(0), %g2
sllx  %g1, 3, %g1
```

Code From Compilers



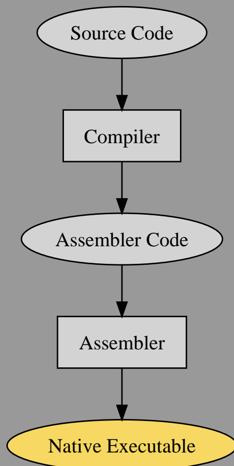
Assembler code is then assembled (and linked).

Code From Compilers



The final product is a native executable (or binary).

Code From Compilers



The final product is a native executable (or binary).

```
^?ELF^A^A^A^@^@^@^@^@^@^@^
    @^@^B^@^C^@^A^@^@^@8^P^
    @^\4^@^@^ @    ^V^A^@^@^@
    ^@^@4^@  ^@^@(^@^X^@^W^@
    ^F^@^@^@4^@^@^@4^@^@
    ^\4^@^@^\  ^A^@^@  ^A^@^@
    ^E^@^@^@^D^@^@^@
```

Other Ways Code is Made

- Hand coded assembler.
  - Not as portable as a high level language.
  - Takes longer.
  - Can make very optimised code.

Other Ways Code is Made

- Hand coded assembler.
  - Not as portable as a high level language.
  - Takes longer.
  - Can make very optimised code.
- Modified compiler output.
  - To add low level instrumentation.
  - To hide nasty code in innocent looking binaries.

1. Reverse Engineering in General
   - A definition
   - For Example
   - Why do People Reverse Software?
2. Where Does Executable Code Come From?
   - Code From Compilers
   - Other Ways Code is Made
3. First Steps in Reverse Engineering
   - The Disassembler
   - Disassembler Output
   - Slicing
4. Rounding Up
   - It is Never That Easy!
   - Conclusions

The Disassembler

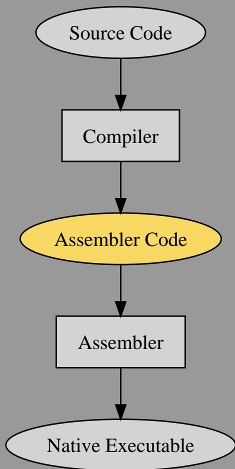
So where do we start with reverse engineering binaries?

The Disassembler

So where do we start with reverse engineering binaries?

- For a start, it helps to get something human readable(ish)?

The Disassembler



$\Leftarrow$ We can get back here using a “disassembler”.

Disassembler Output

```
1c000728 <main>:
1c000728:       8d 4c 24 04             lea    0x4(%esp),%ecx
1c00072c:       83 e4 f0                and    $0xfffffff0,%esp
1c00072f:       ff 71 fc                pushl  0xfffffffc(%ecx)
1c000732:       55                      push   %ebp
1c000733:       89 e5                   mov    %esp,%ebp
1c000735:       51                      push   %ecx
1c000736:       83 ec 10                sub    $0x10,%esp
1c000739:       8b 41 04                mov    0x4(%ecx),%eax
1c00073c:       ff 70 04                pushl  0x4(%eax)
1c00073f:       e8 84 fd ff ff          call   1c0004c8 <__init+0x48>
1c000744:       83 c4 10                add    $0x10,%esp
1c000747:       83 f8 05                cmp    $0x5,%eax
1c00074a:       77 18                   ja     1c000764 <main+0x3c>
1c00074c:       ff 24 85 0c 00 00 3c    jmp    *0x3c00000c(,%eax,4)
1c000753:       83 ec 0c                sub    $0xc,%esp
1c000756:       68 0a 00 00 3c          push   $0x3c00000a
1c00075b:       e8 38 fd ff ff          call   1c000498 <__init+0x18>
1c000760:       83 c4 10                add    $0x10,%esp
1c000763:       90                      nop
1c000764:       31 c0                   xor    %eax,%eax
```

Now What?

- Well you can try some basic techniques such as “slicing”.

Now What?

- Well you can try some basic techniques such as “slicing”.
- With slicing, number of CPU instructions are recognised as a high level language construct.

Now What?

- Well you can try some basic techniques such as “slicing”.
- With slicing, number of CPU instructions are recognised as a high level language construct.
- Lets look at an example...

```
1c000728 <main>:
1c000728:       8d 4c 24 04             lea     0x4(%esp),%ecx
1c00072c:       83 e4 f0                and     $0xfffffff0,%esp
1c00072f:       ff 71 fc                pushl   0xfffffffc(%ecx)
1c000732:       55                      push    %ebp
1c000733:       89 e5                   mov     %esp,%ebp
1c000735:       51                      push    %ecx
1c000736:       83 ec 10                sub     $0x10,%esp
1c000739:       8b 41 04                mov     0x4(%ecx),%eax
1c00073c:       ff 70 04                pushl   0x4(%eax)
1c00073f:       e8 84 fd ff ff          call    1c0004c8 <__init+0x48>
1c000744:       83 c4 10                add     $0x10,%esp
1c000747:       83 f8 05                cmp     $0x5,%eax
1c00074a:       77 18                   ja      1c000764 <main+0x3c>
1c00074c:       ff 24 85 0c 00 00 3c    jmp     *0x3c00000c(,%eax,4)
1c000753:       83 ec 0c                sub     $0xc,%esp
1c000756:       68 0a 00 00 3c          push    $0x3c00000a
1c00075b:       e8 38 fd ff ff          call    1c000498 <__init+0x18>
1c000760:       83 c4 10                add     $0x10,%esp
1c000763:       90                      nop
1c000764:       31 c0                   xor     %eax,%eax
```

Back to our disassembler output.

```
1c000728 <main>:
1c000728:       8d 4c 24 04             lea    0x4(%esp),%ecx
1c00072c:       83 e4 f0                and    $0xfffffff0,%esp
1c00072f:       ff 71 fc                pushl  0xfffffffc(%ecx)
1c000732:       55                      push   %ebp
1c000733:       89 e5                   mov    %esp,%ebp
1c000735:       51                      push   %ecx
1c000736:       83 ec 10                sub    $0x10,%esp
1c000739:       8b 41 04                mov    0x4(%ecx),%eax
1c00073c:       ff 70 04                pushl  0x4(%eax)
1c00073f:       e8 84 fd ff ff          call   1c0004c8 <__init+0x48>
1c000744:       83 c4 10                add    $0x10,%esp
1c000747:       83 f8 05                cmp    $0x5,%eax
1c00074a:       77 18                   ja     1c000764 <main+0x3c>
1c00074c:       ff 24 85 0c 00 00 3c    jmp    *0x3c00000c(,%eax,4)
1c000753:       83 ec 0c                sub    $0xc,%esp
1c000756:       68 0a 00 00 3c          push   $0x3c00000a
1c00075b:       e8 38 fd ff ff          call   1c000498 <__init+0x18>
1c000760:       83 c4 10                add    $0x10,%esp
1c000763:       90                      nop
1c000764:       31 c0                   xor    %eax,%eax
```

The tell tale signs of a `switch()` statement using a jump table.

```
1c000728 <main>:
1c000728:       8d 4c 24 04             lea     0x4(%esp),%ecx
1c00072c:       83 e4 f0                and     $0xfffffff0,%esp
1c00072f:       ff 71 fc                pushl   0xfffffffc(%ecx)
1c000732:       55                      push    %ebp
1c000733:       89 e5                   mov     %esp,%ebp
1c000735:       51                      push    %ecx
1c000736:       83 ec 10                sub     $0x10,%esp
1c000739:       8b 41 04                mov     0x4(%ecx),%eax
1c00073c:       ff 70 04                pushl   0x4(%eax)
1c00073f:       e8 84 fd ff ff          call    1c0004c8 <__init+0x48>
1c000744:       83 c4 10                add     $0x10,%esp
1c000747:       83 f8 05                cmp     $0x5,%eax
1c00074a:       77 18                   ja      1c000764 <main+0x3c>
1c00074c:       ff 24 85 0c 00 00 3c    jmp     *0x3c00000c(,%eax,4)
1c000753:       83 ec 0c                sub     $0xc,%esp
1c000756:       68 0a 00 00 3c          push    $0x3c00000a
1c00075b:       e8 38 fd ff ff          call    1c000498 <__init+0x18>
1c000760:       83 c4 10                add     $0x10,%esp
1c000763:       90                      nop
1c000764:       31 c0                   xor     %eax,%eax
```

Bounds check.

```
1c000728 <main>:
1c000728:   8d 4c 24 04             lea    0x4(%esp),%ecx
1c00072c:   83 e4 f0                and    $0xfffffff0,%esp
1c00072f:   ff 71 fc                pushl  0xfffffffc(%ecx)
1c000732:   55                      push   %ebp
1c000733:   89 e5                   mov    %esp,%ebp
1c000735:   51                      push   %ecx
1c000736:   83 ec 10                sub    $0x10,%esp
1c000739:   8b 41 04                mov    0x4(%ecx),%eax
1c00073c:   ff 70 04                pushl  0x4(%eax)
1c00073f:   e8 84 fd ff ff          call   1c0004c8 <__init+0x48>
1c000744:   83 c4 10                add    $0x10,%esp
1c000747:   83 f8 05                cmp    $0x5,%eax
1c00074a:   77 18                   ja     1c000764 <main+0x3c>
1c00074c:   ff 24 85 0c 00 00 3c    jmp    *0x3c00000c(,%eax,4)
1c000753:   83 ec 0c                sub    $0xc,%esp
1c000756:   68 0a 00 00 3c          push   $0x3c00000a
1c00075b:   e8 38 fd ff ff          call   1c000498 <__init+0x18>
1c000760:   83 c4 10                add    $0x10,%esp
1c000763:   90                      nop
1c000764:   31 c0                   xor    %eax,%eax
```

Jump elsewhere if out of bounds.

```
1c000728 <main>:
1c000728:       8d 4c 24 04             lea     0x4(%esp),%ecx
1c00072c:       83 e4 f0                and     $0xfffffff0,%esp
1c00072f:       ff 71 fc                pushl   0xfffffffc(%ecx)
1c000732:       55                      push    %ebp
1c000733:       89 e5                   mov     %esp,%ebp
1c000735:       51                      push    %ecx
1c000736:       83 ec 10                sub     $0x10,%esp
1c000739:       8b 41 04                mov     0x4(%ecx),%eax
1c00073c:       ff 70 04                pushl   0x4(%eax)
1c00073f:       e8 84 fd ff ff          call    1c0004c8 <__init+0x48>
1c000744:       83 c4 10                add     $0x10,%esp
1c000747:       83 f8 05                cmp     $0x5,%eax
1c00074a:       77 18                   ja      1c000764 <main+0x3c>
1c00074c:       ff 24 85 0c 00 00 3c    jmp     *0x3c00000c(,%eax,4)
1c000753:       83 ec 0c                sub     $0xc,%esp
1c000756:       68 0a 00 00 3c          push    $0x3c00000a
1c00075b:       e8 38 fd ff ff          call    1c000498 <__init+0x18>
1c000760:       83 c4 10                add     $0x10,%esp
1c000763:       90                      nop
1c000764:       31 c0                   xor     %eax,%eax
```

Otherwise use an indirect jump into a jump table.

In-fact this was the source code of the previous disassembly:

```
int
main(int argc, char **argv)
{
    int a = atoi(argv[1]);

    switch (a) {
        case 0:
            printf("0\n");
            break;
        case 1:
            printf("1\n");
            break;
        case 2:
            printf("2\n");
            break;
        case 3:
            printf("3\n");
            break;
        case 4:
            printf("4\n");
            break;
        case 5:
            printf("5\n");
            break;
        };

    return 0;
}
```

1. Reverse Engineering in General
   - A definition
   - For Example
   - Why do People Reverse Software?
2. Where Does Executable Code Come From?
   - Code From Compilers
   - Other Ways Code is Made
3. First Steps in Reverse Engineering
   - The Disassembler
   - Disassembler Output
   - Slicing
4. Rounding Up
   - It is Never That Easy!
   - Conclusions

- Every CPU has it’s own instruction set and quirks.

- Every CPU has it’s own instruction set and quirks.
- It is difficult to know all possible routes through a program.

- Every CPU has it's own instruction set and quirks.
- It is difficult to know all possible routes through a program.
  - Without a comprehensive set of test cases.

- Every CPU has it’s own instruction set and quirks.
- It is difficult to know all possible routes through a program.
  - Without a comprehensive set of test cases.
  - But running the code is cheating!

- Every CPU has it's own instruction set and quirks.
- It is difficult to know all possible routes through a program.
  - Without a comprehensive set of test cases.
  - But running the code is cheating!
- If people do not want their code reverse engineered, they will obfuscate their code.

- Every CPU has it’s own instruction set and quirks.
- It is difficult to know all possible routes through a program.
  - Without a comprehensive set of test cases.
  - But running the code is cheating!
- If people do not want their code reverse engineered, they will obfuscate their code.
  - Hand coded assembler / modified compiler output.
  - Inserting junk/fake code.
  - Misaligned jumps.
  - Hiding control flow edges with signals.
  - Crypto! Ouch!

Conclusions

- Reverse engineering can be used for both good and bad purposes.

Conclusions

- Reverse engineering can be used for both good and bad purposes.
- Reverse engineering is not trivial.

Conclusions

- Reverse engineering can be used for both good and bad purposes.
- Reverse engineering is not trivial.
- We need better disassembler algorithms.

Conclusions

- Reverse engineering can be used for both good and bad purposes.
- Reverse engineering is not trivial.
- We need better disassembler algorithms.
- We need better ways of retrieving control flow from binaries.

Thanks $T^i n_k e^r S_o c$!