Reverse Engineering 101

What is Reverse engineering ?

You are given a peice of program/software without its source code figure out the mechanism of how the program works.

- can we break programs and write out keygens
- find out internal flaws in the code and bypass our way into wonderland.
- bug hunting in different programs

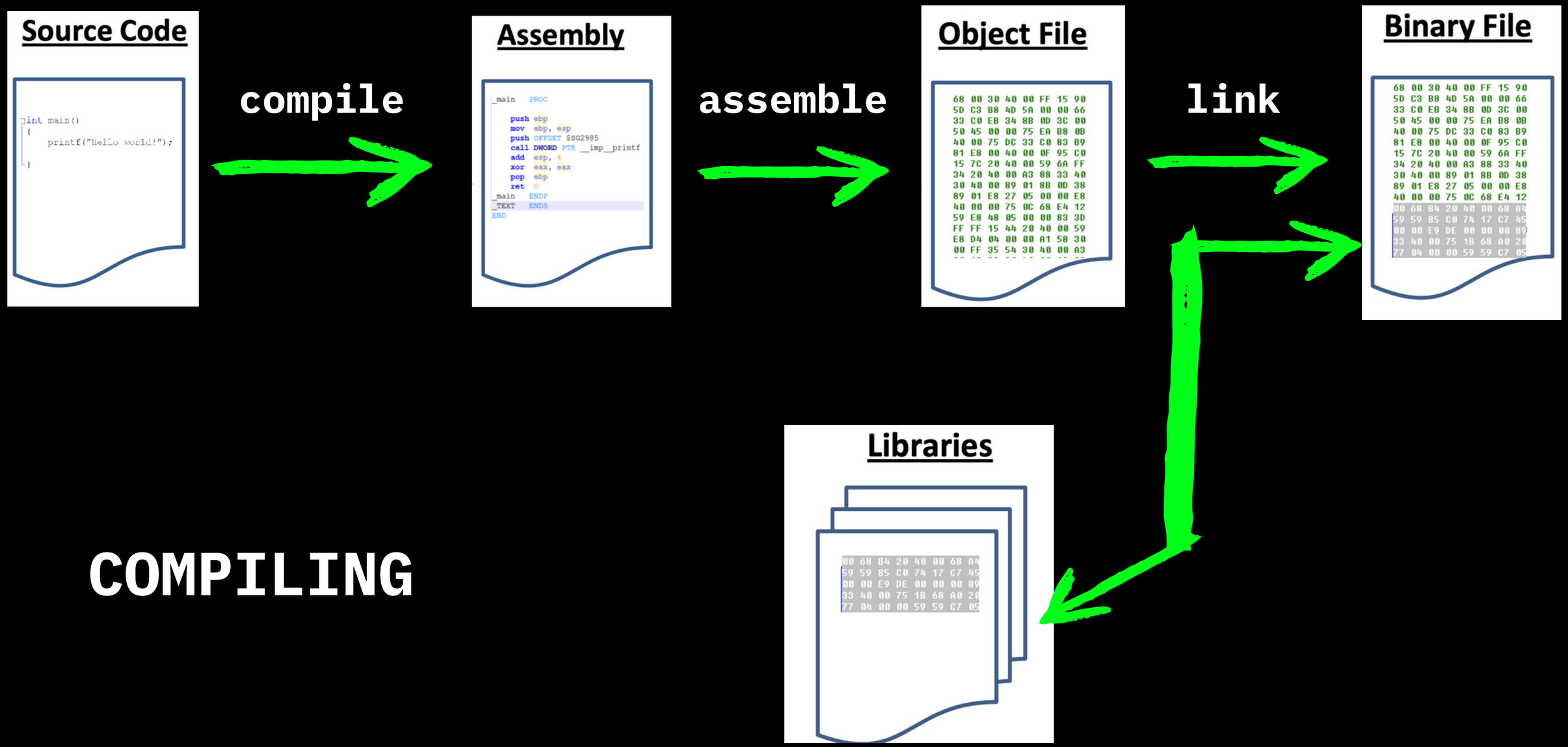
Programs are generally written using different languages like C,C++,Java,Rust,etc which require different approaches for re.

Executables

Contains machine code (x86, ARM, ...) that your processor understands

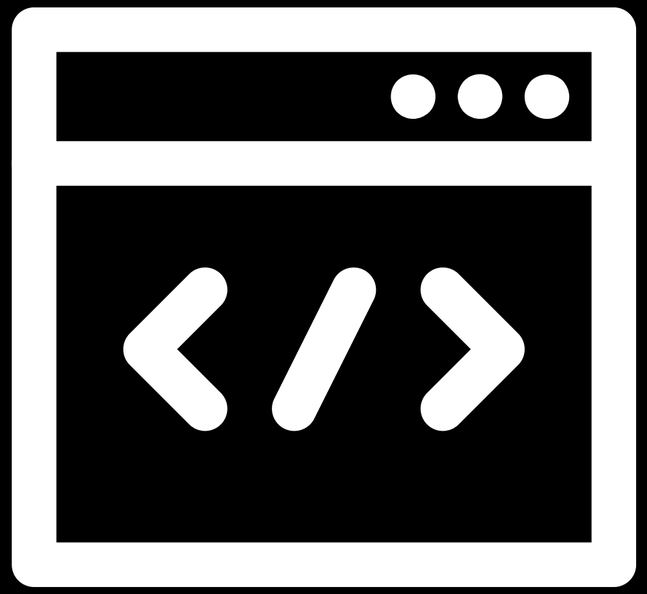
- Hard for humans to understand, though!
- Uses registers and a stack, among other things
- Register = 64 bit number (can be a number or a pointer)
- Think of this as a general purpose variable
- Stack = memory you can push and pop (used for function calls)
- Heap = malloc’d memory
- Data segment = memory where global variables are at

at every step information is lost



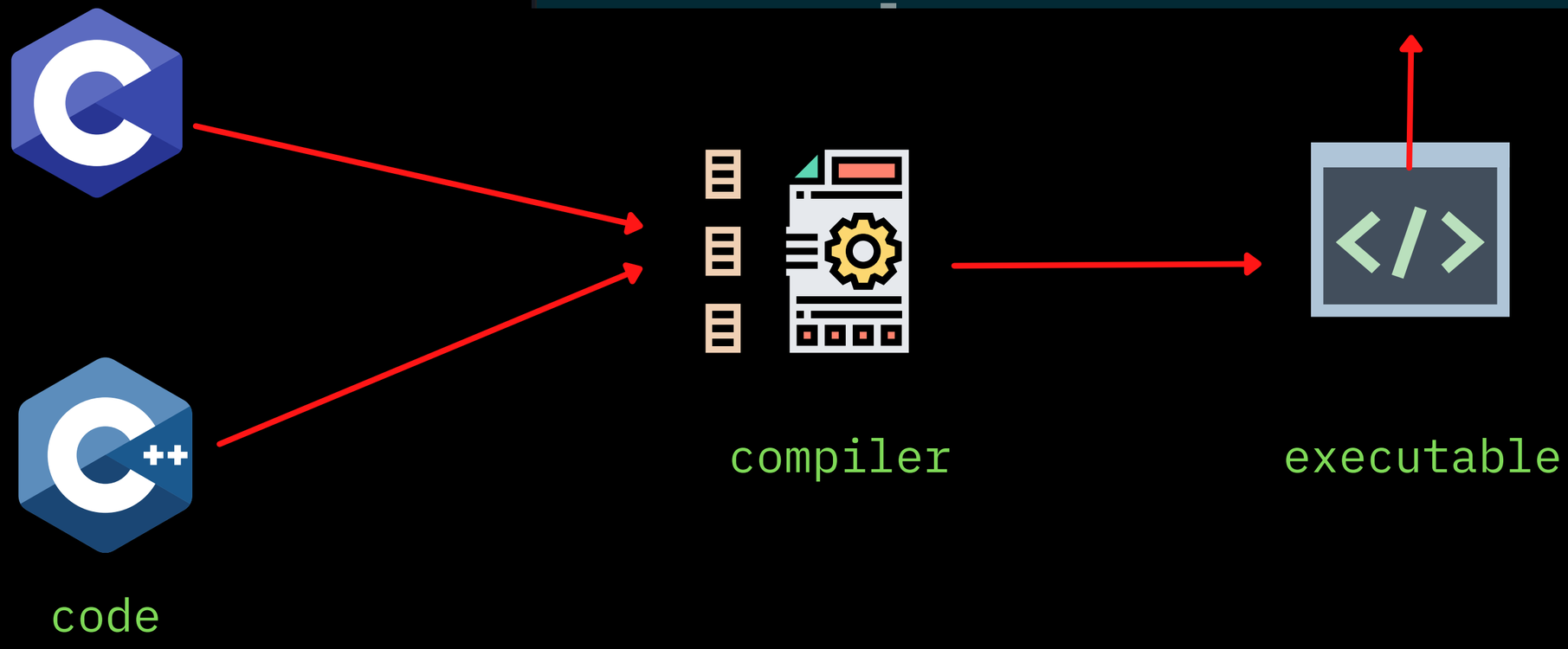
COMPILING

Let's write some C code



In this course we will look upon reverse engineering C/C++ executable.

```
ajp@jarvis-net:~/ctf$ ./basiccode
Hamcker!
```



Reverse it!

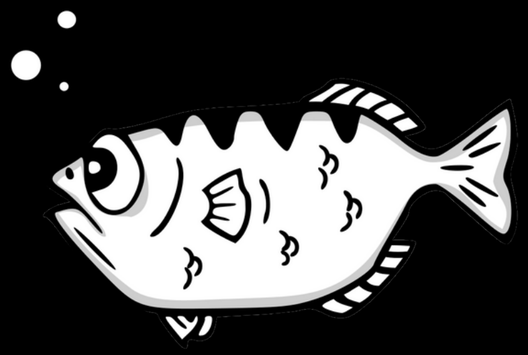
```c
#include<stdio.h>
#include<string.h>

int main()
{

    char s[18];


    gets(s);

    if(!(strcmp(s,"Hello Basic Rev!!!")))
    {
        printf("Granted!\n");
    }
    else
        printf("Denied!!!\n");

    return 0;
}
```

source code

```
00000000000011a9 <main>:
    11a9:       f3 0f 1e fa             endbr64
    11ad:       55                      push   %rbp
    11ae:       48 89 e5                mov    %rsp,%rbp
    11b1:       48 83 ec 40             sub    $0x40,%rsp
    11b5:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    11bc:       00 00
    11be:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    11c2:       31 c0                   xor    %eax,%eax
    11c4:       48 b8 48 65 6c 6c 6f    movabs $0x6142206f6c6c6548,%rax
    11cb:       20 42 61
    11ce:       48 ba 73 69 63 20 52    movabs $0x2176655220636973,%rdx
    11d5:       65 76 21
    11d8:       48 89 45 e0             mov    %rax,-0x20(%rbp)
    11dc:       48 89 55 e8             mov    %rdx,-0x18(%rbp)
    11e0:       66 c7 45 f0 21 21       movw   $0x2121,-0x10(%rbp)
    11e6:       c6 45 f2 00             movb   $0x0,-0xe(%rbp)
    11ea:       48 8d 45 c0             lea    -0x40(%rbp),%rax
    11ee:       48 89 c7                mov    %rax,%rdi
    11f1:       b8 00 00 00 00          mov    $0x0,%eax
    11f6:       e8 b5 fe ff ff          callq  10b0 <gets@plt>
    11fb:       48 8d 55 e0             lea    -0x20(%rbp),%rdx
    11ff:       48 8d 45 c0             lea    -0x40(%rbp),%rax
    1203:       48 89 d6                mov    %rdx,%rsi
    1206:       48 89 c7                mov    %rax,%rdi
    1209:       e8 92 fe ff ff          callq  10a0 <strcmp@plt>
    120e:       85 c0                   test   %eax,%eax
    1210:       75 0e                   jne    1220 <main+0x77>
    1212:       48 8d 3d eb 0d 00 00    lea    0xdeb(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    1219:       e8 62 fe ff ff          callq  1080 <puts@plt>
    121e:       eb 0c                   jmp    122c <main+0x83>
    1220:       48 8d 3d e6 0d 00 00    lea    0xde6(%rip),%rdi        # 200d <_IO_stdin_used+0xd>
    1227:       e8 54 fe ff ff          callq  1080 <puts@plt>
    122c:       b8 00 00 00 00          mov    $0x0,%eax
    1231:       48 8b 4d f8             mov    -0x8(%rbp),%rcx
    1235:       64 48 33 0c 25 28 00    xor    %fs:0x28,%rcx
    123c:       00 00
    123e:       74 05                   je     1245 <main+0x9c>
    1240:       e8 4b fe ff ff          callq  1090 <__stack_chk_fail@plt>
    1245:       c9                      leaveq
    1246:       c3                      retq
    1247:       66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
    124e:       00 00
```

assembly code

DECOMPILER AND DEBUGGER



GDB

RADARE CUTTER

Decompile with Ghidra

- Open source disassembler/decompiler
- Transforms executable to disassembly
- Can decompile disassembly to pseudo-C
- Written by the NSA!!!!

Ghidra to the rescue!

```c
#include<stdio.h>
#include<string.h>

int main()
{

    char s[18];


    gets(s);

    if(!(strcmp(s,"Hello Basic Rev!!!")))
    {
        printf("Granted!\n");
    }
    else
        printf("Denied!!!\n");

    return 0;
}
```

normal code

```c

undefined8 main(void)

{
  int iVar1;
  long in_FS_OFFSET;
  char local_28 [24];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  gets(local_28);
  iVar1 = strcmp(local_28,"Hello Basic Rev!!!");
  if (iVar1 == 0) {
    puts("Granted!");
  }
  else {
    puts("Denied!!!");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}

```

ghidra decompiled output(more understandable than normal assembly)

Common Tools in Reverse engineering

You can use tools/commands likes : (For static Analysis)

Strings

Objdump

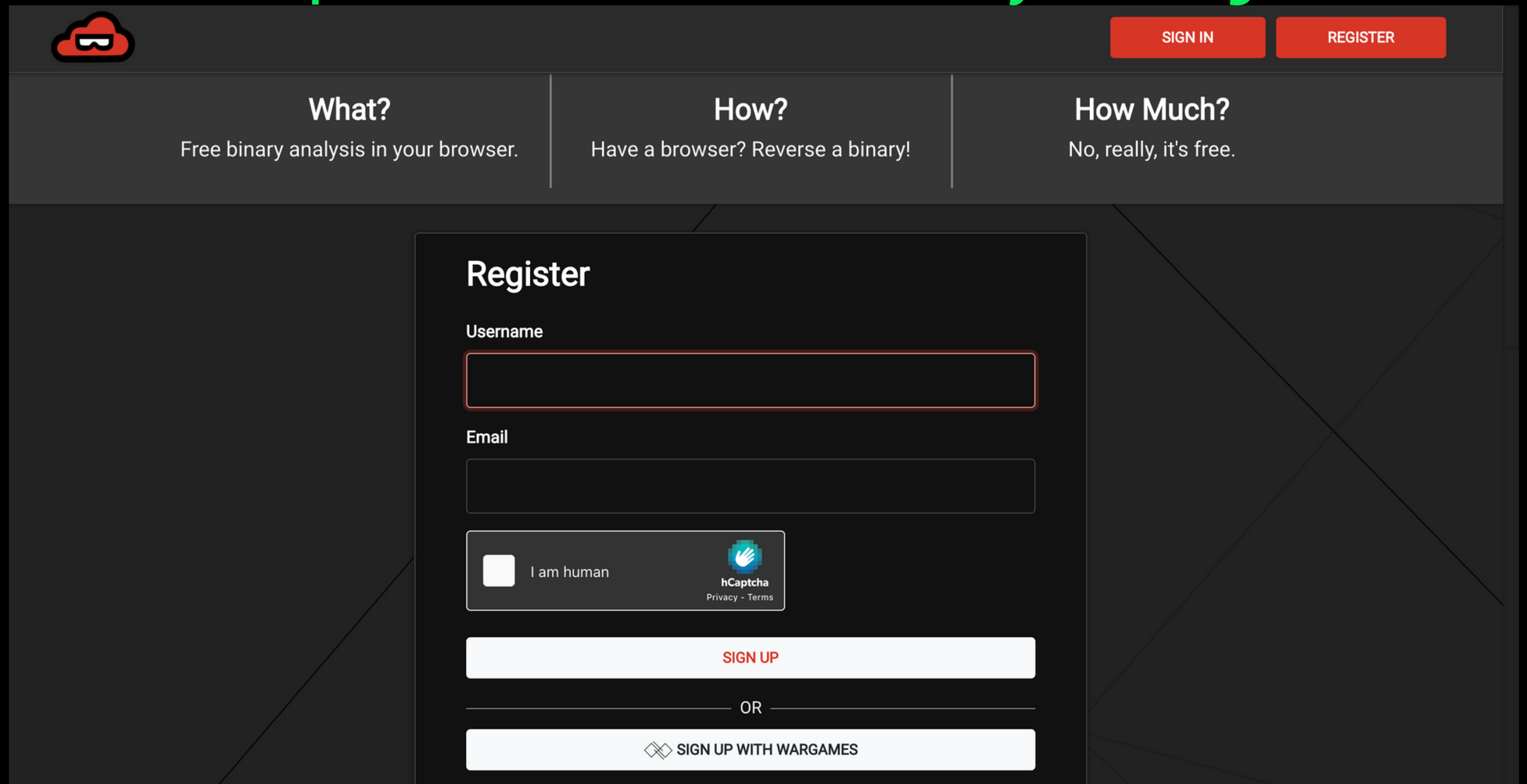
Advanced Disassembler (Binary Ninja / Ghidra, cutter etc)

(For Dynamic Analysis)

```
ltrace : tracing library  calls
stace  : tracing system calls
gdb    : gnu debugger
```

FOR UPCOMING VIDEOS
(CREATE A ACCOUNT ON BINJA CLOUD)

https://cloud.binary.ninja/



7h@NK Y0U F0R W4tch1Ng!

Ghidra Cheat Sheet

Get started:
 View all functions in list on left side of screen. Double click main to decompile main

Decompiler:

- Middle click a variable to highlight all instances in decompilation
- Type “L” to rename variable
- “Ctrl+L” to retype a variable
- Type “;” to add an inline comment on the decompilation and assembly
- Alt+Left Arrow to navigate back to previous function

General:

- Double click an XREF to navigate there
- Search -> For Strings -> Search to find all strings (and XREFs)
- Choose Window -> Function Graph for a graph view of disassembly