Teaching the stack for fun and profit

Hugh Nowlan

A HackerWeek presentation

Me

- TCD
  - Grand
- DCU
  - Grand

Warnings

- Lecture pertains to IA32 for the most part
- No insta-l33t
  - Frames, pointers, bounds
  - Stack discipline is boooring
  - Complex topic

Apology before one is required

- Target audience

The stack

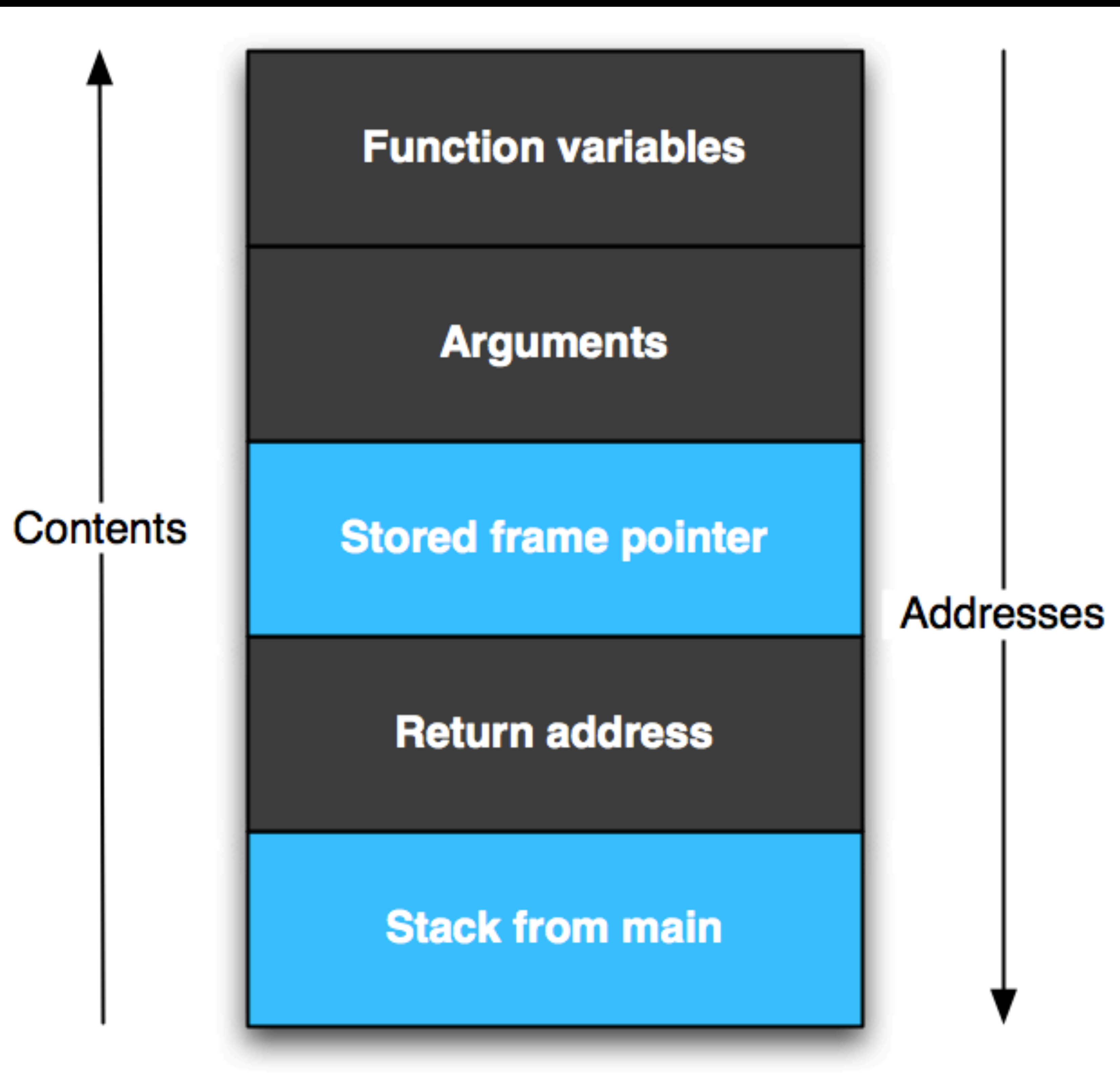
- Keep track of functions called
  - Arguments
  - *Where to return to*
  - Local variables
- Stored in frames

Stack buffer overflows

- Debilitating
- Widespread (still)
- Easily introduced
- Successful exploitation alters control flow
- Flow altered via modification of return values

Function execution

- Before function execution
  - Address of next instruction stored
  - Frame pointer stored
  - Frame pointer becomes function address
- After function execution - ret
  - Return address popped from stack



Contents
Function variables
Arguments
Stored frame pointer
Return address
Stack from main
Addresses

Bounds

- User controlled input = potential threat
  - SQL injection, XSS, BoFs
- Programming practice should protect
  - Some core elements are insecure
- Input should go into input buffers
  - Nowhere else...

C checklist

- gets
- printf
- strcpy
- memcpy
- Others...

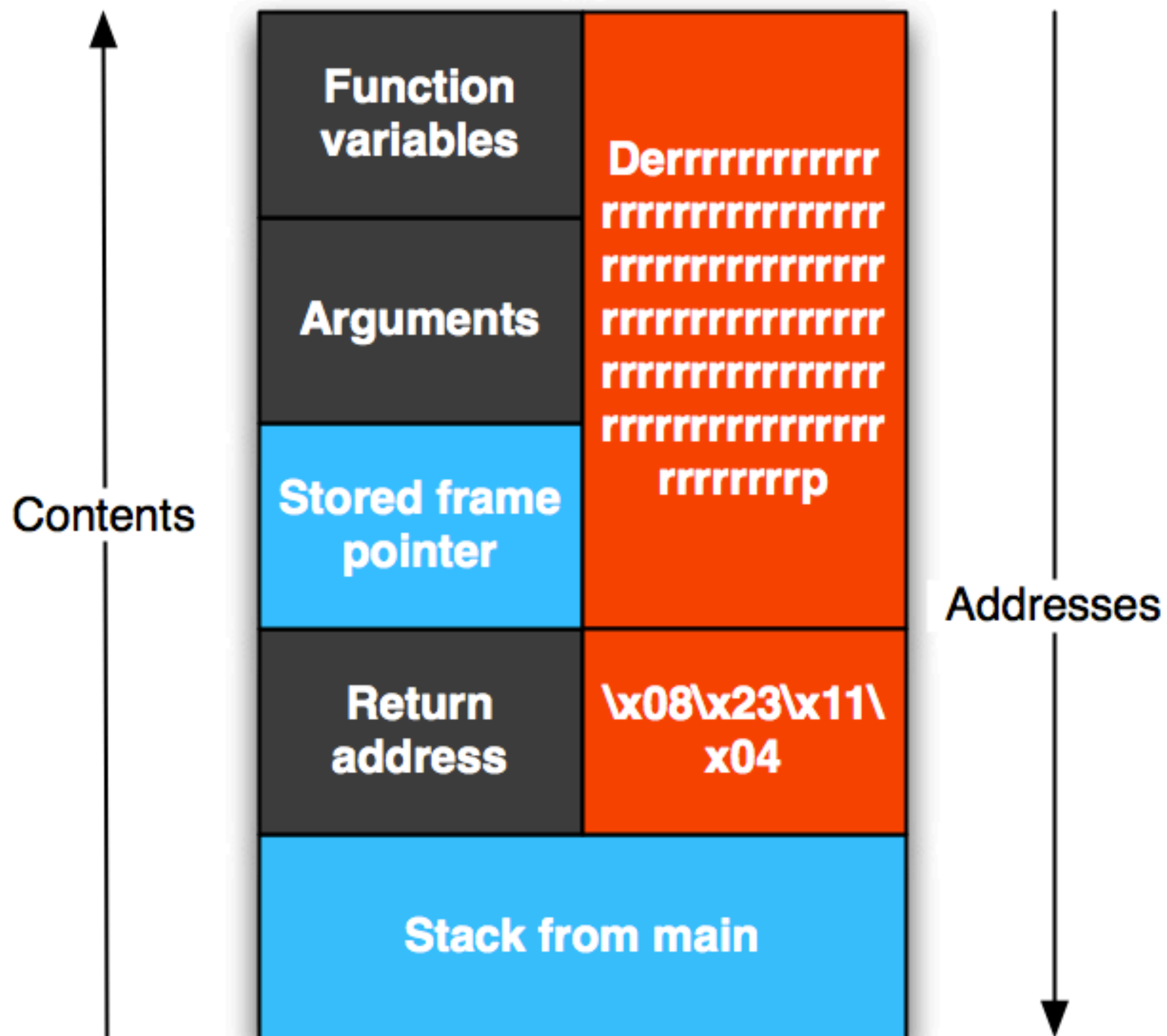
Unchecked bounds

- Unchecked bounds can mean compromise
- What happens here? Inside function x:
  - input = pointer to 18 bytes of data
  - destination = 10 byte array local to x
  - strcpy(destination, input)

Oh no.

What now?

- Unsurprisingly, a crash
  - Function postlude goes ahead unaware
  - Stored frame pointer becomes “rrrp”
  - Derp
- Let’s boost it a little


Contents
Function variables
Arguments
Stored frame pointer
Return address
Stack from main
Derrrrrrrrrrrr rrrrrrrrrrrrrrrr rrrrrrrrrrrrrrrr rrrrrrrrrrrrrrrr rrrrrrrrrrrrrrrr rrrrrrrrrrrrrrrr rrrrrrrrp
\x08\x23\x11\x04
Addresses

Exploitation at last

- Input is simply a \x escaped hex memory address
- This points to a function from which to return to
- Handy if you have the right function
  - No arguments, calls execl("/bin/sh",null);

Best-case

- Assuming this function
- Get the address (say 0x04112308)
- Exploited function executes “ret”
- Code “returns” to the exploit
- Usually not an option

Shellcode

- Expansion of previous technique
- Injection of code alongside memory redirection
- Encoded assembly instructions to execute exploit code
- The shorter your code, the leeter your skilz

Pre-assembly

```
void shellcode(void) {
    char *execarg[2];
    execarg[0] = “/bin/sh”;
    execarg[1] = NULL;
    execve(execarg[0],execarg,NULL);
}
```

Preparing payloads

- Injecting the code isn’t enough
- NOPs used to make up loading space
- Buffer location is unclear to attacker

Assembly

- Produced assembly is optimised
- \xeb\x1f\x5e\xb8\x00\x00\x00\x00\xc7 \x46\x07\x00\x00\x00\x00\x50...
- Nulls need to be removed
- Strings need to be loaded carefully

Caveats

- Executing /bin/sh runs as exploiting user
- SUID bit changes this
  - Program executes as owner
  - Usually aids less-privileged users
  - chmod +s aprogram

Prevention

- Writing safe code
- Non-executable stack
- Bounds checking
- Stack Guard
- Instruction randomisation

Correct code

- Easier than it sounds
- Cliches appear time and time again
- Static analysis can help

Bounds checking

- Ensure reads and writes are in-bounds
- Only array references
- Can’t see variables beyond where they are declared
- Some operations slowed by large factors

Randomised instructions

- Encrypted executables
- On the fly decryption
- Most injected code lacks valid instructions
- Only protects against injected code
- Performance hit

Canaries

- Present in GCC 4.x
- Augmented function prologues and epilogues
- Canary near return address
- Value integrity is checked on return

No-exec stack

- Non-executable segments in stack
- Disable execute on all but .text
  - Code section
- Still vulnerable to
  - Data modification
  - Existing dangerous code

Prognosis

- Grim
- Only you etc. etc.

Further reading

- "Smashing the Stack for Fun and Profit"
  - Aleph One
- Real world examples!
  - exploit-db
    - Shellcode archive

Reading is only so entertaining

- intruded.net
  - Great for learning
- overthewire.org
  - A little more advanced

kthxbai

- Questions?
- Presentation available soon
  - http://www.netsoc.tcd.ie/~nosmo