Utilizing High Entropy Stack Canaries for Locating Function Return Addresses

...and...

Threads WTF? (**W**here's **T**he **F**rame?)

Matt Davis (@enferex)
mattdavis9@gmail.com

BSides PDX 2017

October 21, 2017

Goal of Presentation

- Introduce Execution Stacks
- Introduce Thread Size "Problem"
- Split Stacks
- Stack Canaries
- Entropy
- 1337 Hackz
- turtle:
- goto turtle;

Goal of Presentation

- Introduce Execution Stacks
- Introduce Thread Size "Problem"
- Split Stacks
- Stack Canaries
- Entropy
- 1337 Hackz
- turtle:
- goto turtle;

"I love it when a plan comes together."

–Hannibal Smith, A-Team

About Me

About Me

$$(void*)0$$

About Me

$(void*)0$

IT DOESN'T MATTER

Motivation

We can compromise stack integrity by first understanding how stacks are created and then exploiting a security mechanism used to protect stacks.

Background: Stacks

Background...

Background: Stacks

Background...
Stacks

Background: Stacks

Background...

Stacks

Stacks

Stacks

Stacks

Stacks

Stacks

Stacks

Stack Data Structure

Stacks are a last-in-first-out (LIFO) data structure that have two operations:

- *push*: Places an item on the top of the stack.
- *pop*: Removes an item from the top of the stack.

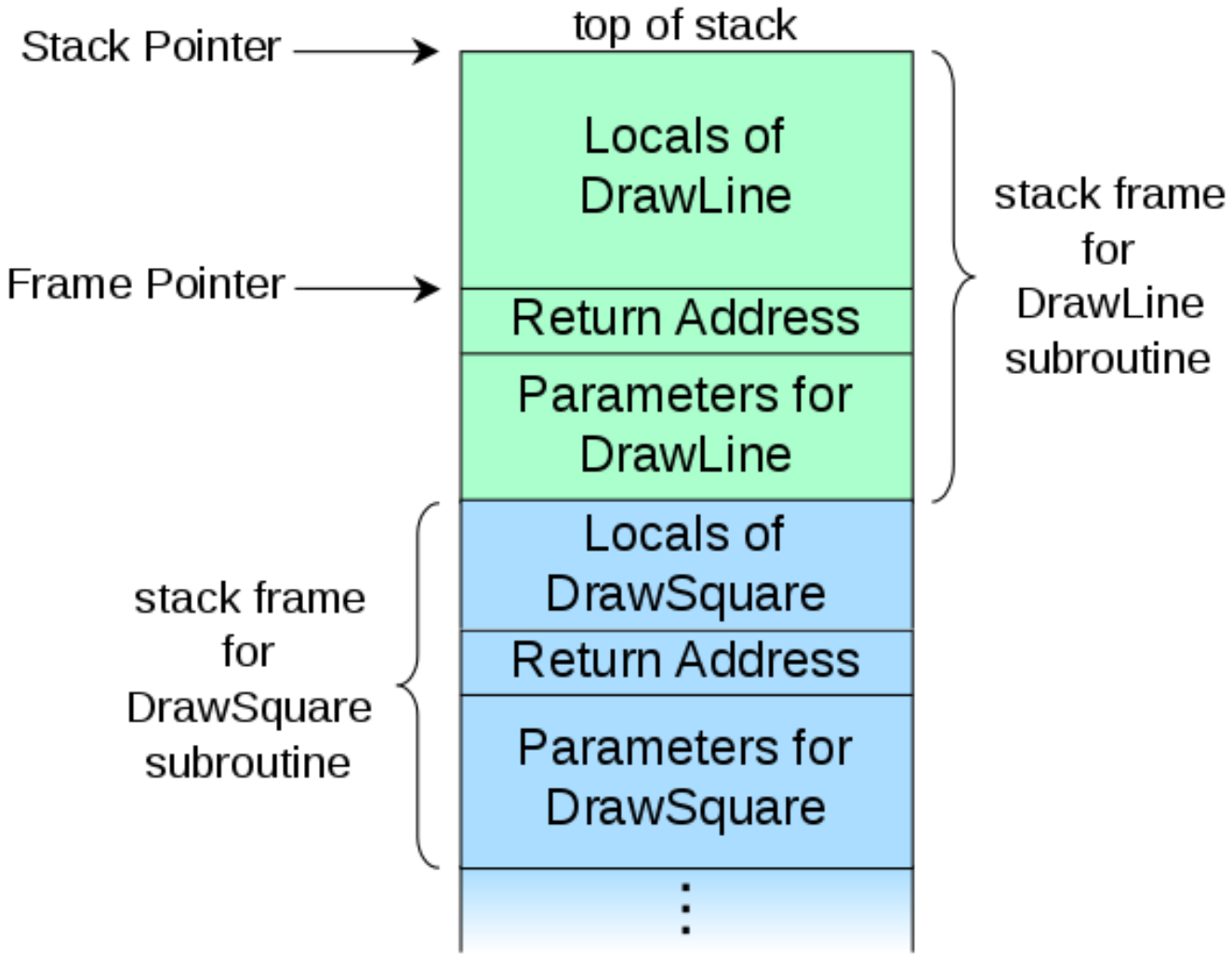
All operations occur to the top if the stack.

Execution Stack

When a program is loaded into the system's memory, a portion of its memory space is used for stack.
A program maintains a stack of execution frames.

- When a function is *called*, a new frame is "pushed" onto the stack.
- When a function *returns*, the top frame is "popped" off the stack.
- A stack frame, which is a temporary store, contains:
  - A function's local variables.
  - Return address (return to caller, in caller's frame).
  - Stack canary.

What a Stack Frame Looks Like

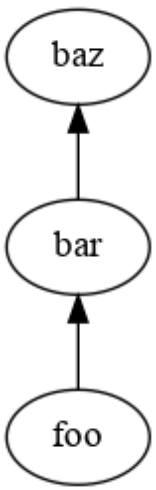


Execution Stack: Example

```
// Example in C
void baz()
{
}

void bar()
{
    int a=0xdead;
    int b=0xcafe;
    baz();
}

void foo()
{
    bar();
}
```

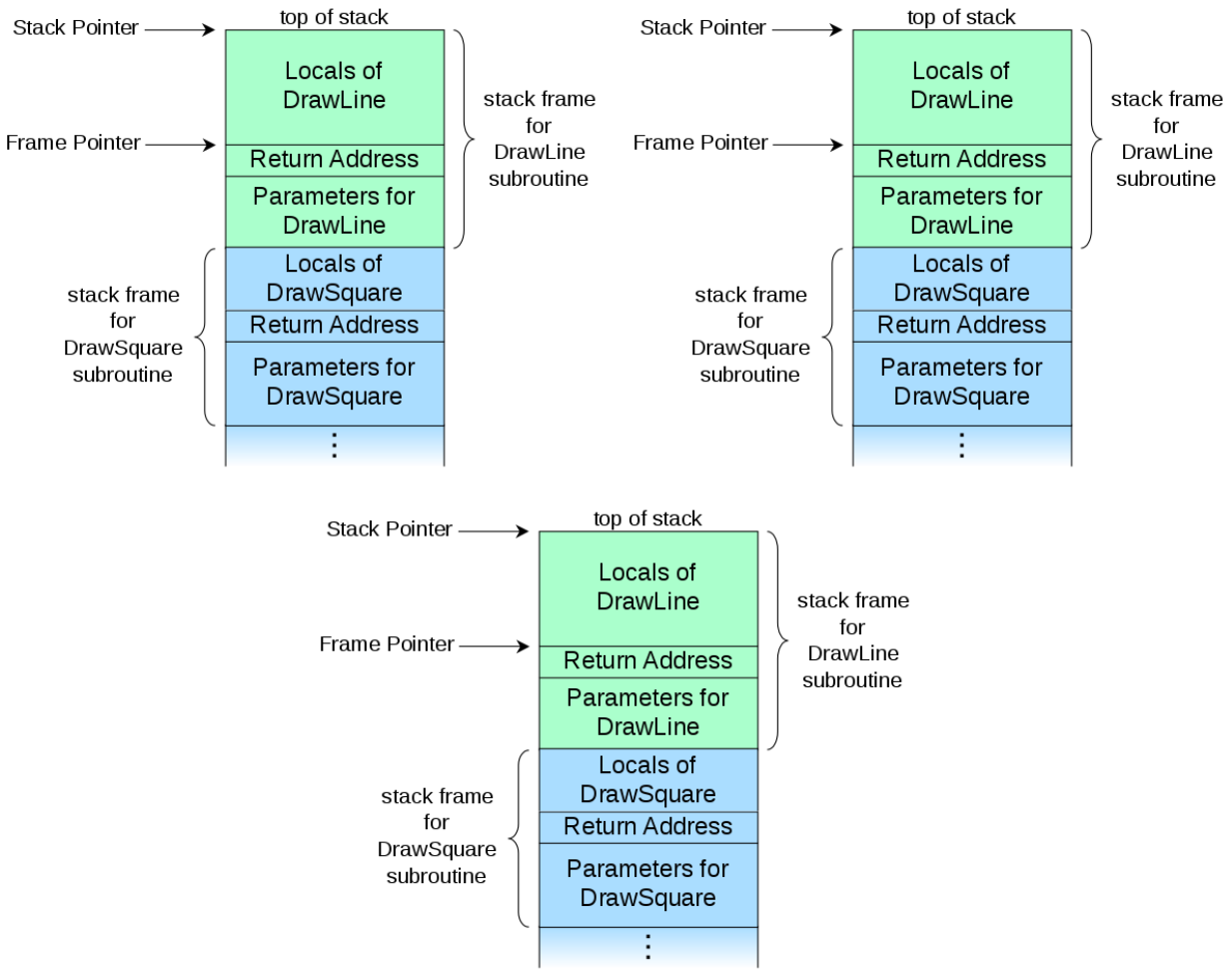



```
# Assembly output of Example
<baz>:
   push   %rbp
   mov    %rsp,%rbp
   nop
   pop    %rbp
   retq

<bar>:
   push   %rbp
   mov    %rsp,%rbp
   sub    $0x10,%rsp
   $0xdead,-0x8(%rbp)
   $0xcafe,-0x4(%rbp)
   mov    $0x0,%eax
   callq  27 <bar+0x20>
   nop
   leaveq
   retq

<foo>:
   push   %rbp
   mov    %rsp,%rbp
   mov    $0x0,%eax
   callq  38 <foo+0xe>
   nop
   pop    %rbp
   retq
```

What About a Multi-Threaded Program?



Each thread has a portion of a process' memory to maintain its own stack.

Problem: Thread Sizes are Hard Coded

- Each thread has its own stack space.
- POSIX (pthread) default will vary, but can be 8MB of stack space per thread.
- This does not scale well.
  - Lots of threads consume lots of memory.

Solution: Split Stack

Split stack: Dynamically allocated stack space.

Languages like Go are designed to spawn many threads, but with small stack spaces. They achieve lower overhead by using a "split stack" stack model.

- The function prologue checks the current thread's stack boundary.
- If the function-to-be executed requires more memory than the thread has available:
  - Allocate a new stack space.
  - Copy the current frame to the new memory.
  - Update the thread's stack pointers.
  - Resume the function.
  - When the function completes, reclaim the memory.
  - Return to the caller.

How Thread Stacks Look

Identifying thread and split-stack data from */proc* filesystem.

Before

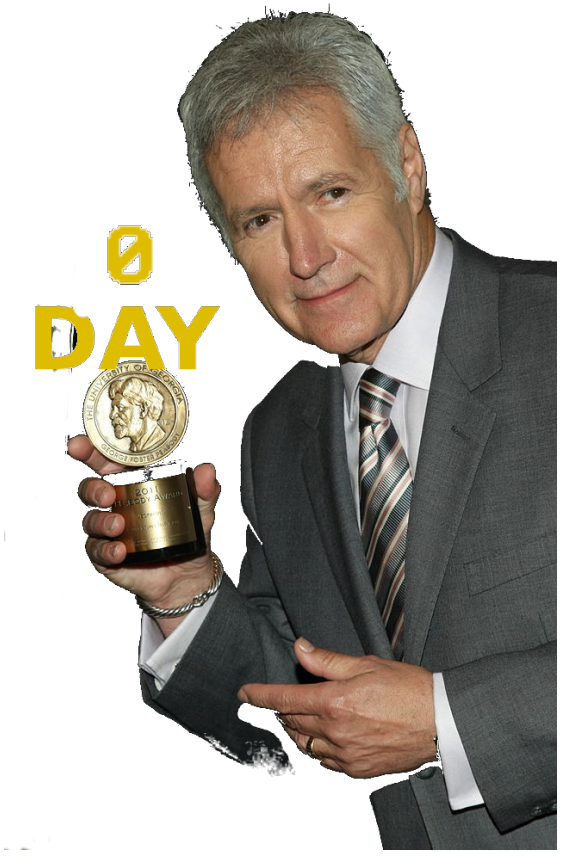
```
55ae25540000-55ae25543000 r-xp 00000000 08:01 1182693   a.out
55ae25742000-55ae25743000 r--p 00002000 08:01 1182693   a.out
55ae25743000-55ae25744000 rw-p 00003000 08:01 1182693   a.out
55ae272a6000-55ae272c7000 rw-p 00000000 00:00 0         [heap]
7fed66f77000-7fed67134000 r-xp 00000000 08:01 398918    libc-2.24.so
```

After

```
55ae25540000-55ae25543000 r-xp 00000000 08:01 1182693   a.out
55ae25742000-55ae25743000 r--p 00002000 08:01 1182693   a.out
55ae25743000-55ae25744000 rw-p 00003000 08:01 1182693   a.out
55ae272a6000-55ae272c7000 rw-p 00000000 00:00 0         [heap]
7fed64b76000-7fed66f77000 rw-p 00000000 00:00 0 <------- THREAD or SPLIT-STACK MEMORY
7fed66f77000-7fed67134000 r-xp 00000000 08:01 398918    libc-2.24.so
```

Pop Quiz, Hot Shot.

What the heck are these pages?




```
55b2998c1000-55b2998c2000 rw-p 00001000 08:01 6356756   a.out
55b29a147000-55b29a168000 rw-p 00000000 00:00 0         [heap]
7ffba9898000-7ffba9899000 ---p 00000000 00:00 0         <--- ???
7ffba9899000-7ffbaa099000 rw-p 00000000 00:00 0         <--- ???
7ffbaf09c000-7ffbaf09d000 rw-p 00025000 08:01 7080132   /usr/lib/ld-2.26.so
```

Pop Quiz, Hot Shot.

What the heck are these pages?


```
55b2998c1000-55b2998c2000 rw-p 00001000 08:01 6356756   a.out
55b29a147000-55b29a168000 rw-p 00000000 00:00 0         [heap]
7ffba9898000-7ffba9899000 ---p 00000000 00:00 0         <--- ???
7ffba9899000-7ffbaa099000 rw-p 00000000 00:00 0         <--- Possibly thread stack space.
7ffbaf09c000-7ffbaf09d000 rw-p 00025000 08:01 7080132   /usr/lib/ld-2.26.so
```

Pop Quiz, Hot Shot.

What the heck are these pages?


```
55b2998c1000-55b2998c2000 rw-p 00001000 08:01 6356756   a.out
55b29a147000-55b29a168000 rw-p 00000000 00:00 0         [heap]
7ffba9898000-7ffba9899000 ---p 00000000 00:00 0         <--- Guard page (mprotect with (PROT_NONE))
7ffba9899000-7ffbaa099000 rw-p 00000000 00:00 0         <--- Possibly thread stack space.
7ffbaf09c000-7ffbaf09d000 rw-p 00025000 08:01 7080132   /usr/lib/ld-2.26.so
```

Stack Protection and Sentinel Species

Program execution is driven by the stack, because caller return addresses make up part of the stack.

Caller and callee relationship is witnessed at runtime by the pushing and popping of stack frames.

- Stacks can get corrupted (bad use of function local variables).
    - For example: Copying too much data into a variable, and overwriting useful portions of the stack.
- Stacks can be attacked by having program input overwrite the return address of a function to point to some malware's payload.
- Stack canaries, or cookies, can help detect corrupted or compromised stacks and terminate the process immediately.

Stack Canaries

A stack canary is a known value placed onto a stack to ensure the integrity of the stack.

- Canaries can detect stack-overflow compromises.
- GCC Stack Smashing Protection (libssp) places a random word-size value onto the stack in the function prologue.
- The value is checked during function epilogue.
- If the value is not the original, then the stack is corrupted and cannot be trusted.
  - The return address on the stack might have been compromised by malware. (Stack overflow bug).
- The canary should be unpredictable, to prevent malware authors from crafting code that overflows the stack with an expected canary value.

What a Canary Looks Like

gcc canary.c -fstack-protector-all -S

canary.c

```
int foo(void)
{
    return 0xdeadbeef;
}
```

canary.s

```
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        movl    $0xdeadbeef, %eax
        movq    -8(%rbp), %rdx
        xorq    %fs:40, %rdx
        je      .L3
        call    __stack_chk_fail
.L3:
        leave
        ret
```

Entropy based Stack Canaries

Stack canaries are often generated via some pseudo-random number generator.

- This reduces the probability of an attacker guessing a canary to successfully overwrite (to thwart detection).

GCC will setup function's prologue and epilogue to store and check the function's stack canary.

- The ELF loader, *ld* (glibc source) is responsible for getting a random value every time the program starts.
- This value is generated in the kernel, and passed to *ld* via an auxiliary ELF vector.

```
/*
 * Generate 16 random bytes for userspace PRNG seeding.
 */
get_random_bytes(k_rand_bytes, sizeof(k_rand_bytes));
u_rand_bytes = (elf_addr_t __user *)
                STACK_ALLOC(p, sizeof(k_rand_bytes));
if (__copy_to_user(u_rand_bytes, k_rand_bytes, sizeof(k_rand_bytes)))
        return -EFAULT;
```

Using Canaries to find Thread Return Addresses

Can a canary, of maximum entropy, be used to find a return address?

Using Canaries to find Thread Return Addresses

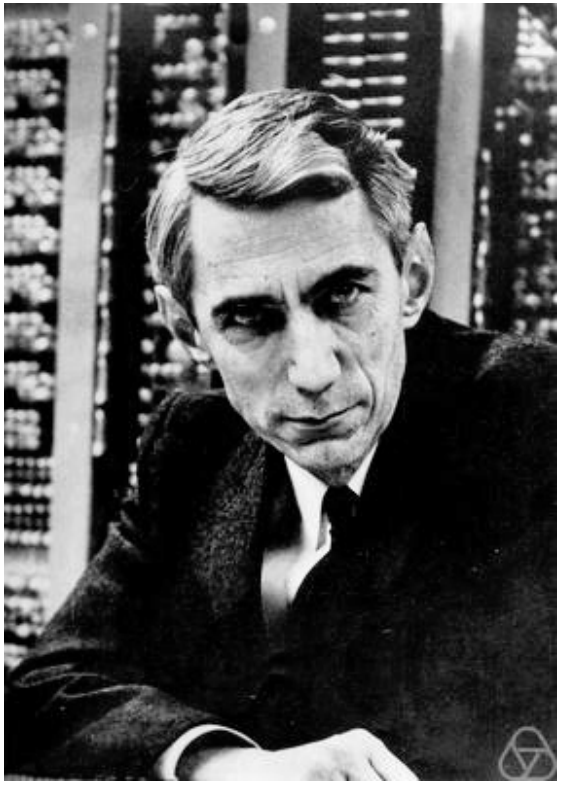
Can a canary, of maximum entropy, be used to find a return address?

Using Canaries to find Thread Return Addresses

Can a canary, of maximum entropy, be used to find a return address?

(Yes!)

Shannon Entropy



Entropy: Average amount of information for a given event.

$$H = -\sum_{i} p_i \log_b p_i$$

Where p_i: Proportion of data i occurring.

For our purposes we count the byte frequency, not bit frequency.

$$\hat{H} = \sum_{i=0}^{TotalBytes} Frequency(byte_i) \log_2(Frequency(byte_i))$$

https://en.wiktionary.org/wiki/Shannon_entropy
https://en.wikipedia.org/wiki/Diversity_index#Shannon_index

Shannon Entropy

Entropy: Average amount of information for a given event.

$$H = -\sum_{i} p_i \log_b p_i$$

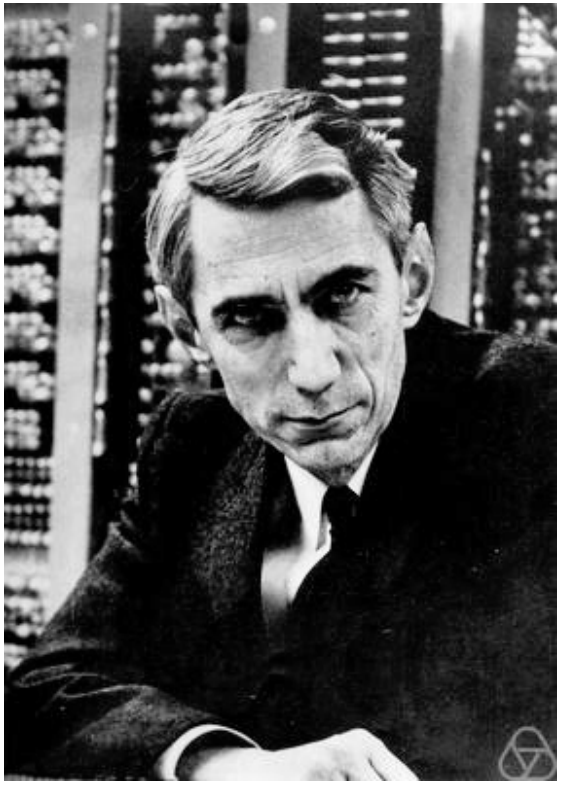
Where p_i: Proportion of data i occurring.

For our purposes we count the byte frequency, not bit frequency.

$$\hat{H} = \sum_{i=0}^{TotalBytes} Frequency(byte_i) \log_2(Frequency(byte_i))$$

https://en.wiktionary.org/wiki/Shannon_entropy
https://en.wikipedia.org/wiki/Diversity_index#Shannon_index

Shannon Entropy



Entropy: Average amount of information for a given event.

$$H = -\sum_{i} p_i \log_b p_i$$

Where p_i: Proportion of data i occurring.

For our purposes we count the byte frequency, not bit frequency.

$$\hat{H} = \sum_{i=0}^{TotalBytes} Frequency(byte_i) \log_2(Frequency(byte_i))$$

https://en.wiktionary.org/wiki/Shannon_entropy
https://en.wikipedia.org/wiki/Diversity_index#Shannon_index

Maximum Entropy

Maximum Entropy: We use a variation of Shannon's Index [1] , based off of Shannon's Entropy formula.

Instead of looking at individual bits, we look at a series of 8 bytes (word size). If each byte in the word is a different value, then we say that word has *Maximum Entropy*.

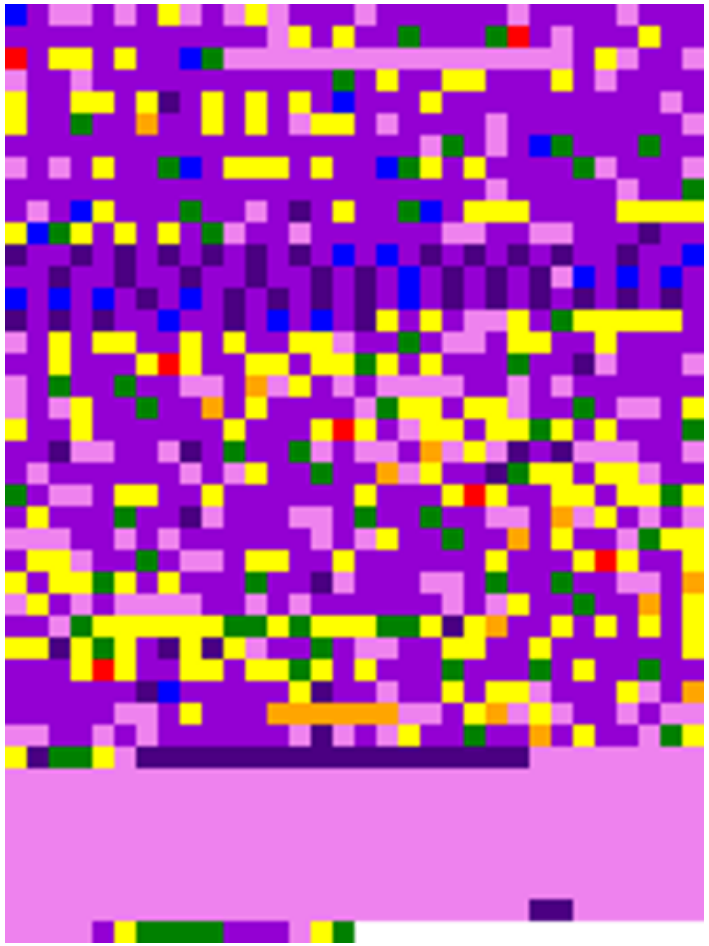
[1] https://en.wikipedia.org/wiki/Diversity_index#Shannon_index

Maximum Entropy Words

- A look at my system's memory (/proc/<pid>/maps)
- Only looking at unnamed memory mapped regions.
  - 1 word is 8 bytes.
  - 784700416 total words scanned.
  - 4337624 words of maximum entropy found.
  - 4MB of 748MB had maximum entropy.
  - 0.55% of words have maximum entropy.

Heatmap

Each block represents a word (8 bytes). Words of all zeros are removed. This follows red-orange-yellow-green-blue-indigo-violet-darkviolet, where red indicates every byte in the word is different and dark violet indicates that 1 byte is different. Note that this was from a differnt run than the previous slide.



Canary Hackery

- Scan /proc/<pid>/maps for areas of thread memory.
- For each memory mapped page:
  1. Scan word-size chunks.
  2. If a word has maximum entropy, look 2 words from it*.
  3. If the value 2 words from the high-entropy word can exist within a memory area from /proc/<pid>/maps, assume it is a return address.

Canary Hackery

- Scan /proc/<pid>/maps for areas of thread memory.
- For each memory mapped page:
  1. Scan word-size chunks.
  2. If a word has maximum entropy, look 2 words from it*.
  3. If the value 2 words from the high-entropy word can exist within a memory area from /proc/<pid>/maps, assume it is a return address.
  4. Turtles.

Done!

```
asm volatile ("ret;\n");
```

Resources (1 of 2)

- POC:
  `https://github.com/enferex/homingcanary`
- Example dynamic stack:
  `https://github.com/enferex/customstack`
- Shannon Entropy Equation:
  `https://en.wiktionary.org/wiki/Shannon_entropy`
- Shannon Index:
  `https://en.wikipedia.org/wiki/Diversity_index#Shannon_index`
- Call stack graphic:
  `https://en.wikipedia.org/wiki/Call_stack#/media/File:Call_stack_layout.svg`
- Yes image:
  `https://en.wikipedia.org/wiki/Yes_(band)#/media/File:Yes_concert.jpg`
- Claude Shannon image:
  By Jacobs, Konrad - `http://owpdb.mfo.de/detail?photo_id=3807`, CC BY-SA 2.0 de,
  `https://commons.wikimedia.org/w/index.php?curid=45380422`

Resources (2 of 2)

- GCC Source (`https://gcc.gnu.org`):
  - Setup split-stack prologue:
    gcc/config/i386/i386.c:ix86_expand_split_stack_prologue()
  - __morestack logic:
    libgcc/config/i386/morestack.S
    libgcc/generic-morestack.c
- Linux Kernel Source (`https://kernel.org`)
- GNU libc Source (`https://www.gnu.org/software/libc/`)