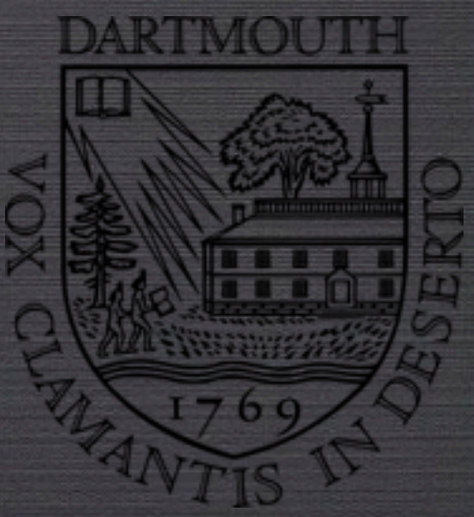
ELF Eccentricities

Julian Bangert, Rebecca Shapiro,
Sergey Bratus

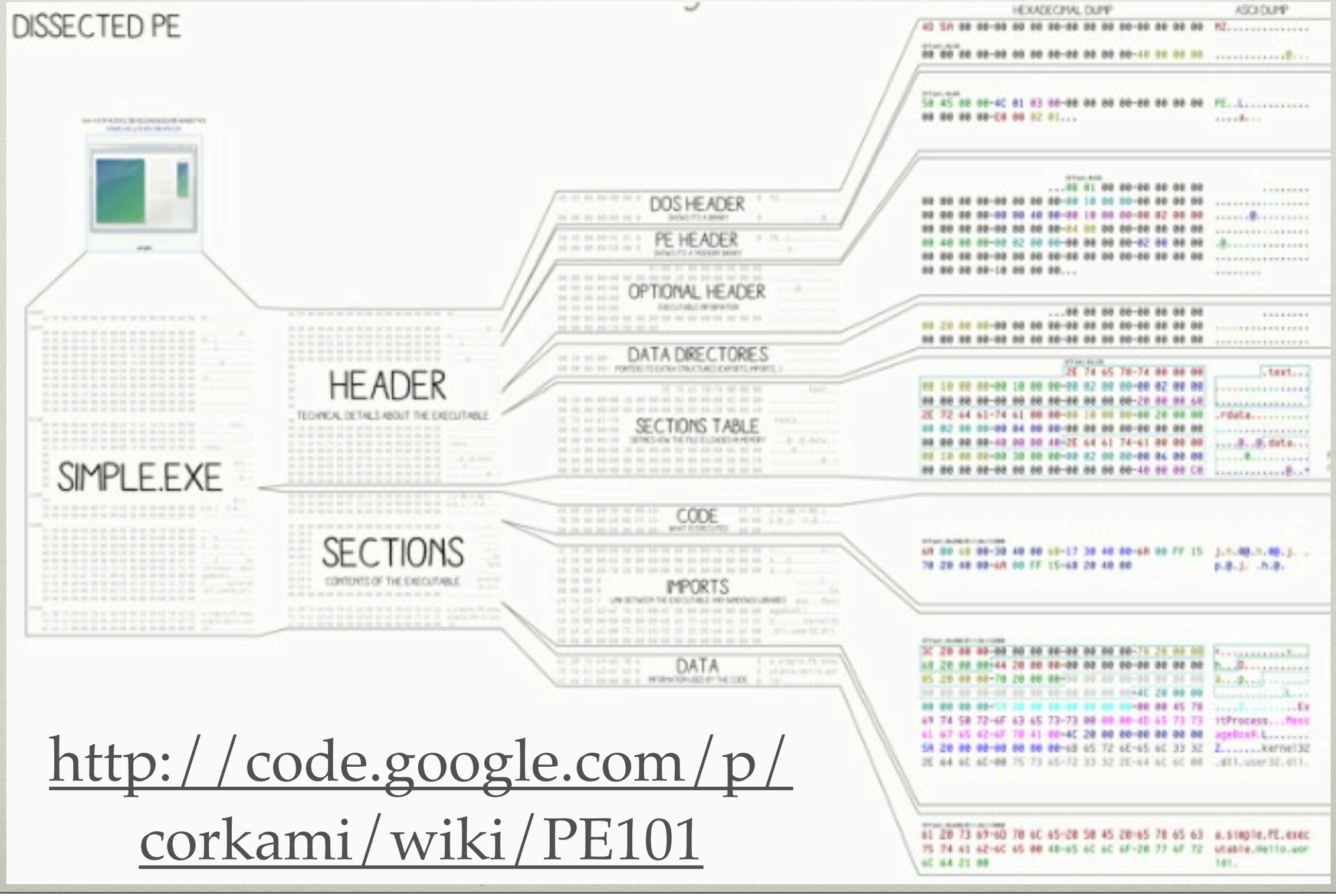
Dartmouth College



This talk brought to you by the letter "L"

- ELF: Executable Linkable Format
- Linking/loading is a complex computation on complex inputs
- Linking/loading paints magical memory trees (vma_structs, ...)
- Done by exec(), ld.so, IDA, ... - each slightly differently
- Full of mystery? Cloaked?

PE, a cousin of ELF



http://code.google.com/p/corkami/wiki/PE101

Mach-O, a cousin of ELF

"This space intentionally left blank"

Langsec angle (1)

- "**Any Input Is A Program**"
  - Input executes on input handlers (drives state changes, transitions)
    - Only a well-defined execution model can be trusted
  - Input handler is either a **recognizer** for the inputs as a well-defined language, or it's pwned

Langsec angle (2)

- "**Parse differentials create subtle vulns**"
- One input, two parsers: a tale of woe
- Two views of the same data: confusion?
  - Redundant data: which version is taken?
  - How many ELF parsers in the binary toolchain? Kernel's `binfmt` loader vs `ld.so`

Of ELFs in all the things

- Executables, .so, .o, coredumps, ...
- Replaced non-extensible AOUT
- Designed by System V working group, extremely extensible:
  - keeps code/data **intelligible** (sections, symbols, dynamic symbols)
  - allows code/data to **compose** (“link”)
  - allows code/data to **mutate** (relocs ~ASLR)

Der Tragödie erster teil

tour de Elf

(with a dash of fail)

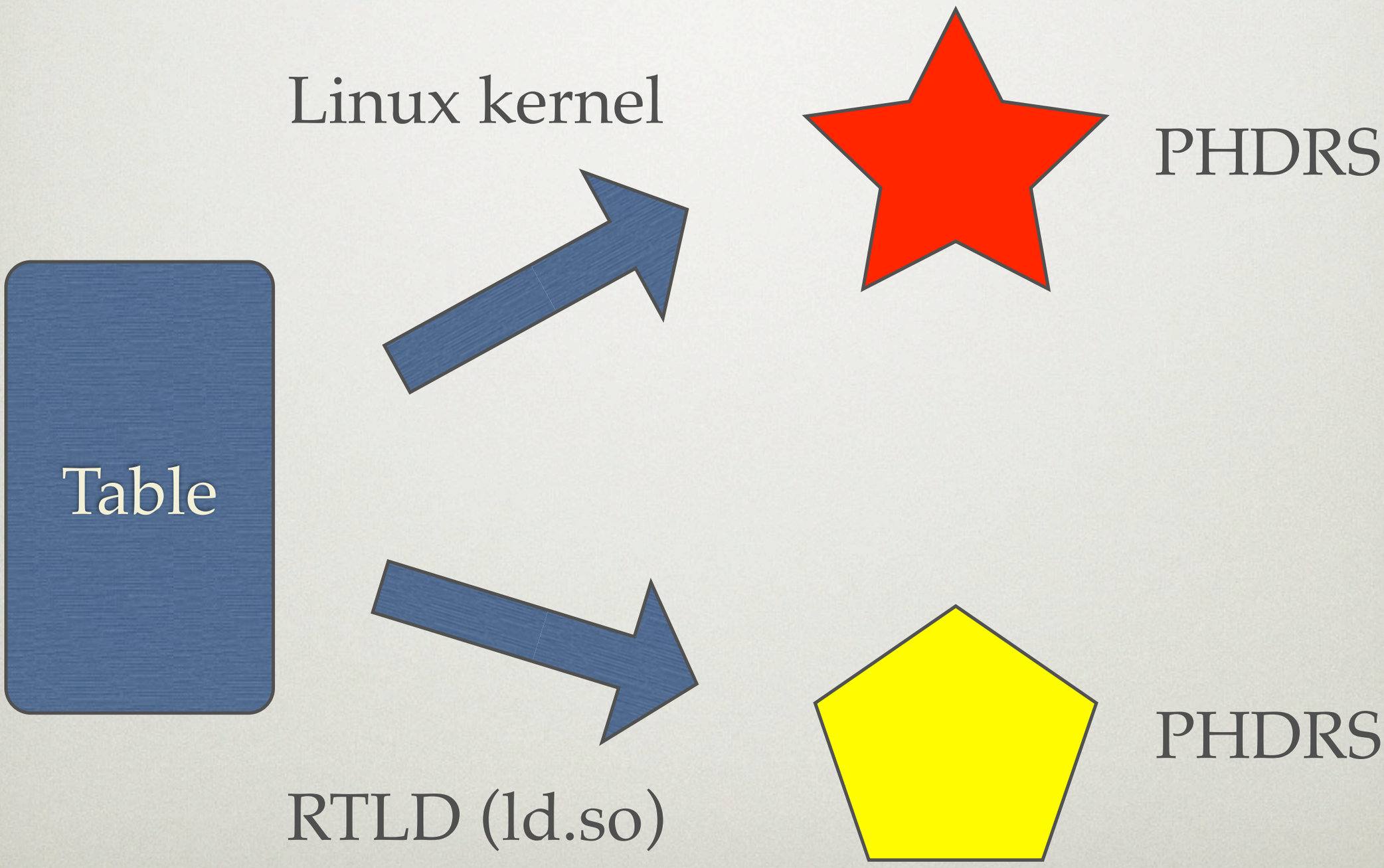
A Beautiful Dream

- Tools don't need different parsers for libraries, object files and executables
- Tools work for all platforms
- Postel's principle: Allow tools to ignore information they don't understand, change tools only as needed
- Otherwise, no .so or ASLR

AND NOW WE WAKE UP

- One libelf to parse them all?
  - Everyone builds their own
- Clean design makes linker design easy?
  - **Revenge of the Committee**
  - **Assault of the Performance Hack**

Double the parsing, double the fun



ELF header

- Generic information (Version, architecture, file type, etc.)
- Refers to section table

```
typedef struct {
        unsigned char   e_ident[16];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

Section Headers

- Label a range of offsets
- Represent 2 views of the file
  - Offset
  - Address
- Acts as ‘labels’ for byte subsets of the file

```
typedef struct
{
  Elf32_Word sh_name;
  Elf32_Word sh_type;
  Elf32_Word sh_flags;
  Elf32_Addr sh_addr;
  Elf32_Off  sh_offset;
  Elf32_Word sh_size;
  Elf32_Word sh_link;
  Elf32_Word sh_info;
  Elf32_Word sh_addralign;
  Elf32_Word sh_entsize;
} Elf32_Shdr;
```

Section header fun

- Do not guarantee non-overlay
  - "What is at offset/address x mean?"
  - Context sensitive
  - We could just reject overlap
  - Yet another obfuscation talk?
- Loader has to parse large section table
  - OMG 30 entries!

Program header

- Loader ignores sections
- Has separate ‘program header table’.
- A different label system

```
typedef struct
{
  Elf32_Word p_type;
  Elf32_Off  p_offset;
  Elf32_Addr p_vaddr;
  Elf32_Addr p_paddr;
  Elf32_Word p_filesz;
  Elf32_Word p_memsz;
  Elf32_Word p_flags;
  Elf32_Word p_align;
} Elf32_Phdr;
```

Follow us into the rabbit hole

```
S U B R O U T I N E =======================================

ased frame

blic main
oc near                 ; DATA XREF: _start+1D↑o
sh     rbp
v      rbp, rsp
v      edi, offset hello_2 ; "Hello, World\n"
ll     _puts
p      rbp
tn
dp
'Hello, World\n"
---------
```

```
$./hello
dlroW ,olleH
$
```

IDA uses sections

Kernel uses segments

Memory vs. File

- Most parsers read() file
- Some work on the memory mmap()-ed version
- Assume almost linear map
- The file describes its own memory layout

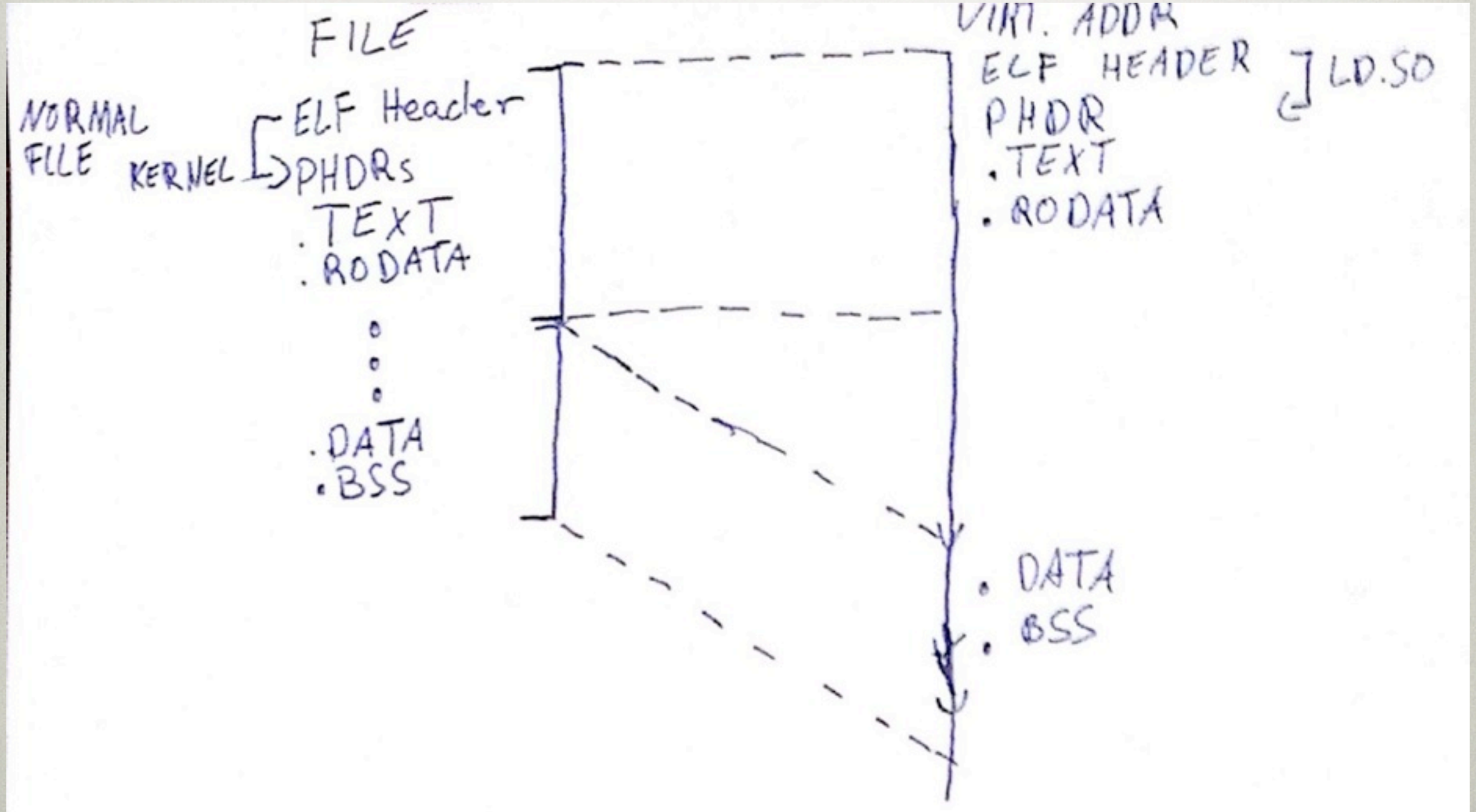
Program headers (I)

- PT_LOAD:
- “Map these bytes (+ some /dev/null) to this address range”
- Overlap: each PT_LOAD is a mmap
- Evaders note that these replace mappings
- Order of PT_LOAD is important, but that’s not in the spec.

The birth of an Elf

1. Kernel reads PHDR table into buffer
2. One mmap per PT_LOAD
3. Loads PT_INTERP
4. Sends (addr of first PT_LOAD)+ hdr.phoff to loader (AUX vector)
5. Loader uses that addr in userspace

LD.SO PHDR


FILE
NORMAL FILE
KERNEL
ELF Header
PHDRs
.TEXT
.RODATA
.DATA
.BSS
VIRT. ADDR
ELF HEADER
PHDR
.TEXT
.RODATA
LD.SO
.DATA
.BSS

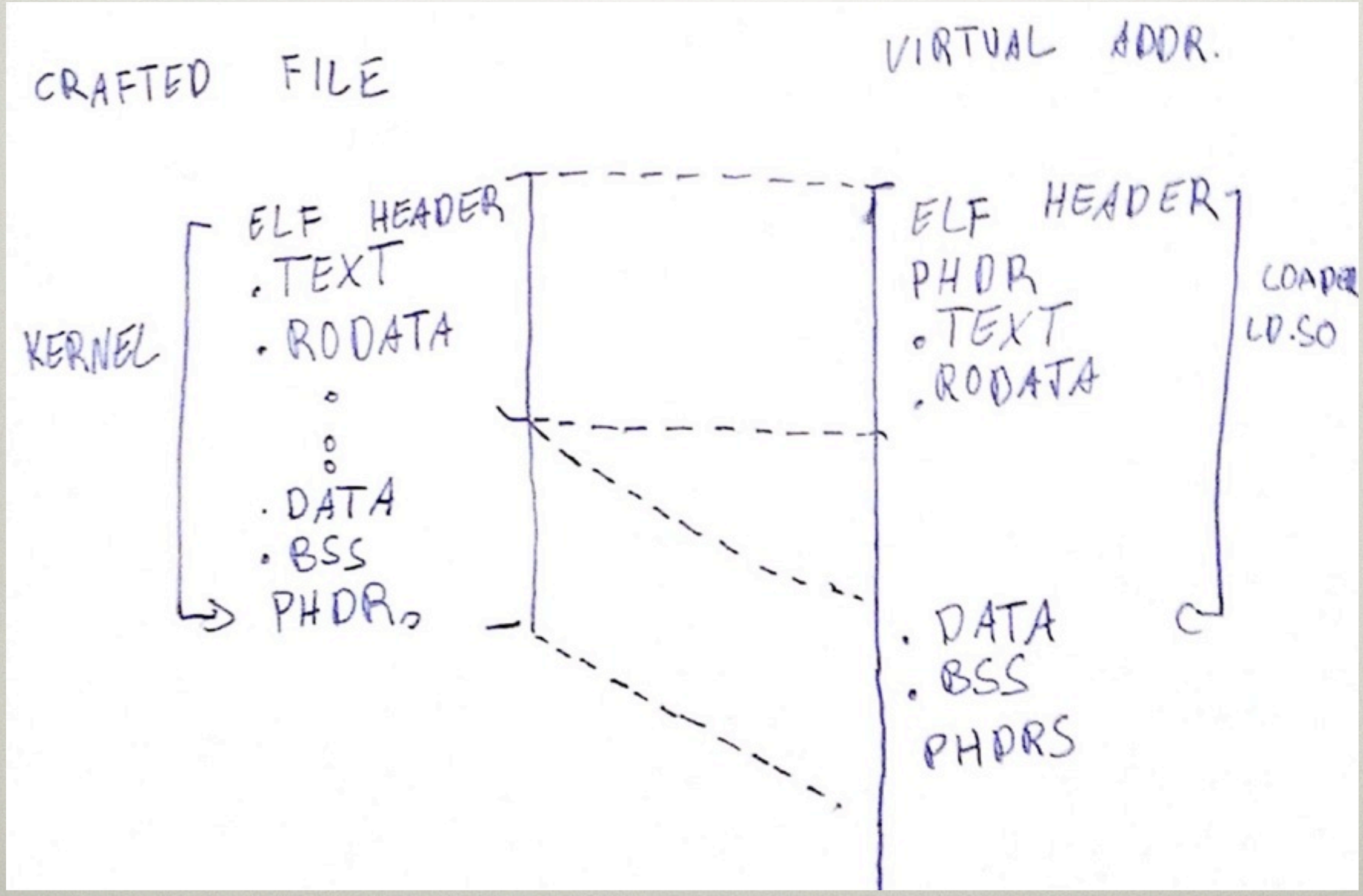
Obligatory kernel source

```
/*Redacted from binfmt_elf.c, Linux 3.4, GPLv2*/
elf_phdata = kmalloc(size, GFP_KERNEL);
kernel_read(bprm->file, loc->elf_ex.e_phoff, (char *)elf_phdata, size)
for(phdrs...){
/*..., scans for PT_INTERP and PT_LOAD*/
if(phdr->p_type == PT_LOAD)
if (!load_addr_set) {
   load_addr = (elf_ppnt->p_vaddr - elf_ppnt->p_offset);
   load_addr_set = 1;
 }
}
NEW_AUX_ENT(AT_PHDR, load_addr + exec->e_phoff); /*FAIL*/
```

PHDR bug

- Works Just Fine™ if program headers mapped as first segment.
- Otherwise, points somewhere else
- We can “finger-paint” memory to our liking, and ld.so will use **different** PHDRS
- Hide your PHDRs, the reverse engineers are coming!

Crafted file



What do we get?

- Can't map memory, the kernel does that
- PT_DYNAMIC

Dynamic section

- The loader gets symbol tables, relocations, etc., from the **.dynamic** section
  - which it finds through the PT_DYNAMIC PHDR
- Only cares about **address**, not file offsets or sections

How Tools find dynamic

- Program Header, by virtual address or offset
- Section
- Symbol

Dynamic entries

- DT_STRTAB-> Strings (e.g. symbols)
- DT_SYMTAB -> Symbols
- DT_RELA -> Relocations (ASLR, shlib)
- DT_NEEDED -> imported shared libraries

SHARED LIBRARIES

- Shared libraries (and PI executables) are loaded at a random address
- Allocate space for entire library, then remap
  - Size: End_of_last_PHDR - start_of_first_PHDR
  - **Deranged** PT_LOADs (ranges “normally” assumed sorted by loader), integer overflow?
  - Can change memory permissions elsewhere in program

"Memory finger-painting"

- PHDRS control vma structs in kernel
- Different paths: mmap(), mmap()->mprotect(), in-kernel parser
- Page-granularity of vmas not reflected in format
- Rounding in various places, overlap
- Combining different paintings

Rewriting programs in memory with ld.so?

- We control **ld.so**'s idea of all relevant sections: GOT, dyn symbols, ...
- **ld.so** resolves (what it thinks are) symbols, writes (what it thinks is) GOT
- Fun way to rewrite your program via crafted .dynamic + library symbols

Symbols

- Symbol is a name for addresses(.so and +x) or offsets(everything else). That is a convention, tools might break if you mix
  - **.dynsym** (loader) vs **.symtab** (linker,debugger)
  - Undefined symbols, imported from somewhere else
  - Weak symbols for ‘default values’
  - Can hide symbols from linker with versioning

Relocations

- Fill in missing references to other objects
- *Theory:* 'Replace this bit of program/data with some computation on this symbol'
- *Reality:* Read memory(.rela, .dynsym), compute, write memory -- **anywhere**
- @bxsays: **Turing machine!** (see elf-bf-tools on Github, our talk noon tomorrow)

What is linking?

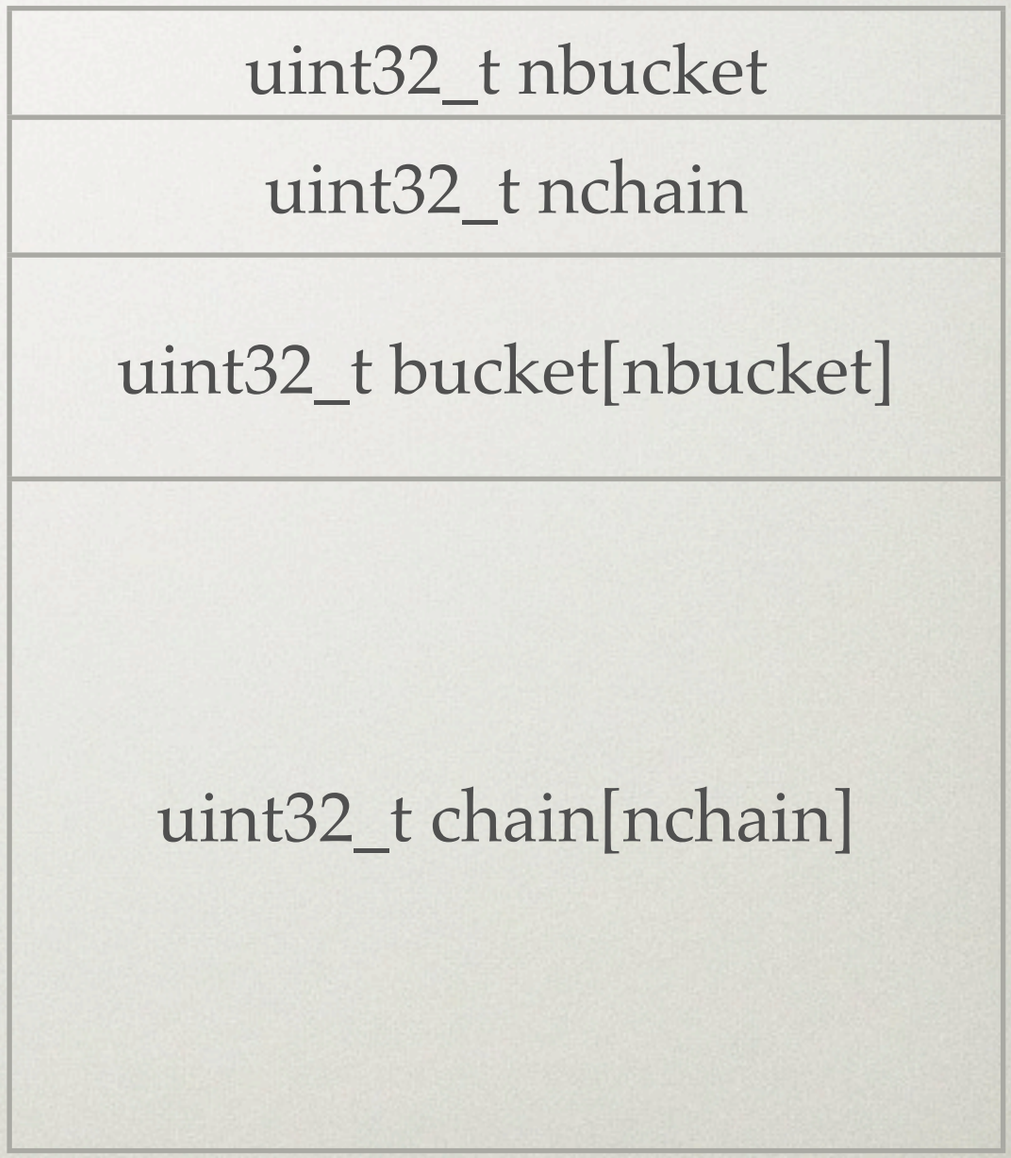
- We are working on formalizing this
- Roughly:
  - Combine sections from different ELF objects
  - Resolve symbols
  - Apply relocations

But it has to be fast

- Symbol lookups by walking string table are inefficient
- DT_HASH and DT_GNU_HASH
  - hashtable Symbol name -> string

DT_HASH

- Hashtable, simple hash
- Each hash bucket can point to one symbol
- chain has ‘next’ link for each symbol



Walking the hash table

```
/*dl-lookup.c:268*/
for (symidx = map->l_buckets[*old_hash % map->l_nbuckets];
     symidx != STN_UNDEF;
     symidx = map->l_chain[symidx])
{
    sym = check_match (&symtab[symidx]);
    if (sym != NULL)
      goto found_it;
}
```

What happens if these checks never succeed?

DT_HASH_LOOPING:
"The Red Queen Break-Point"

- Collision entry points to itself
- Resolver hangs
- Without hash(or DT_GNU_HASH) it will work fine
- Can make only some symbols hang
- Breakpoints

uint32_t nbucket
uint32_t nchain
uint32_t bucket[nbucket]
uint32_t chain[nchain]

Der Tragödie Zweiter teil

In Soviet Russia,

Mithril forges ELF

Inspiration: ElfSH

- ERESI reversing framework by Julien Vanegue and team
- **.bx** and I are big fans!
- ELF handled with libelfsh
- just went back to active development; ELF parsing developed as needed => some bugs

What to do differently

- ERESI modifies the binary as little as possible
- In-memory representation close to on-disk
  - Rewrites much of the binary on change
- Sometimes (inevitably), misses a dependency and breaks the binary

MITHRIL

- Elves smithe powerful swords and chain armour from mithril
- We cut and link ELFes with Mithril
- Parse ELF into a ‘context-free’ representation, write representation into ELF
- Files differ, but semantically identical

MITHRIL

- Ruby prototype (to make mistakes before reimplementing in C)
- Parser component stable
- Generator is more a prototype, will throw away
- github.com/jbangert/mithril

Binarysmithing 101

- Injecting code, ordinary swords:
  - Shift section headers, inject section
  - Shift sections in file, inject data
  - Update PHDRS
- Injecting with Mithril:

```
mysect = Struct.new(:data,:name,:addr,:flags,:align,:entsize)

file.progbits.push mysect.new(shellcode, ".inject", nil,
SHF::SHF_ALLOC, 0,0)
```

Graduate binarysmithing

- Inject symbol:

```
file.symbols << Symbol.new(...)
```

- Inject relocation:

```
file.relocations << Relocation.new()
```

- Mithril::Writer.to_file(“foo”,file)
- Mithril takes care of the rest

Shoutouts!

- LANGSEC conspiracy
- Corkami http://code.google.com/p/corkami/
- ERESI project team, past and present
- TinyELF http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
- The Grugq & Silvio Cesare
  (for hooking Sergey on ELF)
- Phrack \cite{Phrack XX:XX}

Questions?

- @bxsays, @julianbangert, @sergeybratus
- Soon
  - github.com/jbangert/mithril
  - gem ‘mithril’