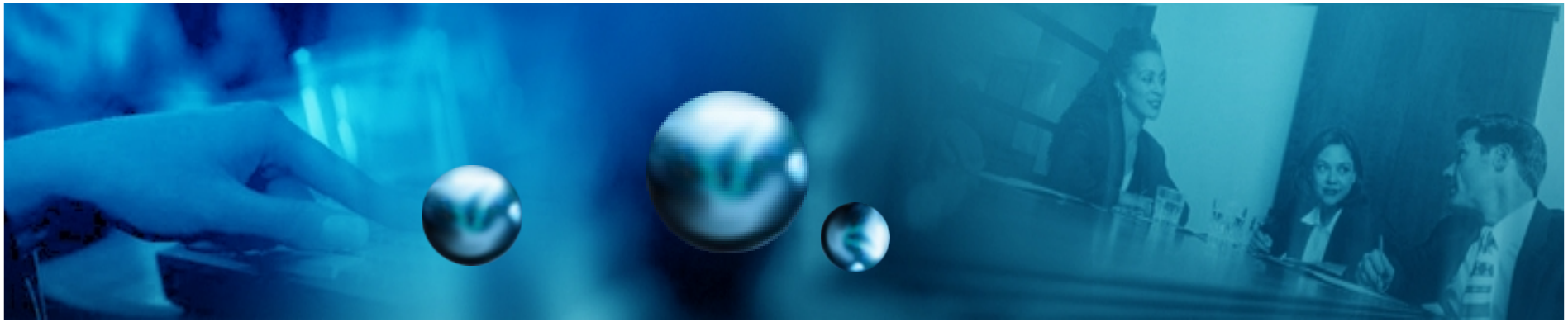
Reverse Engineering for exploit writers



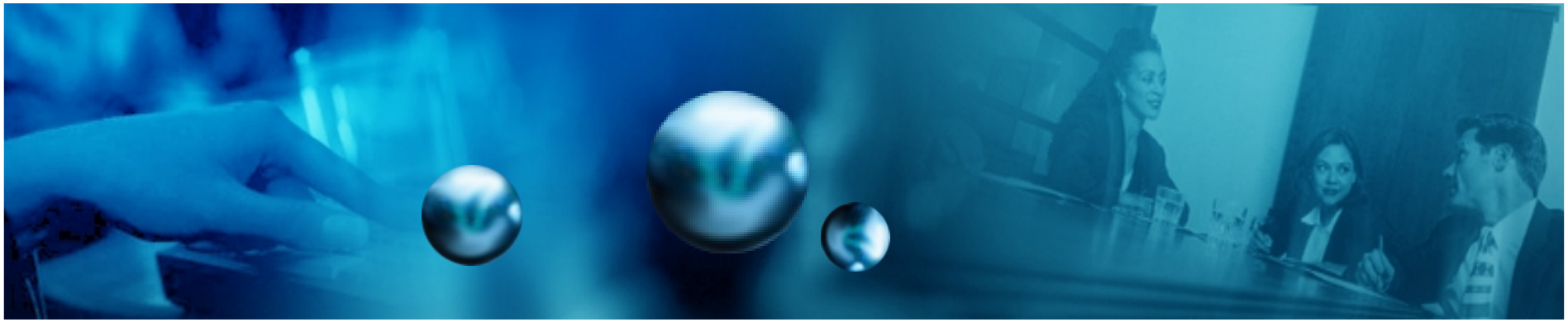
Jonathan Brossard, iViZ Research Team

Clubhack 2008

Pune, India

iViZ
intelligent vision

Who Am I ?
(and why am I writing this ??)



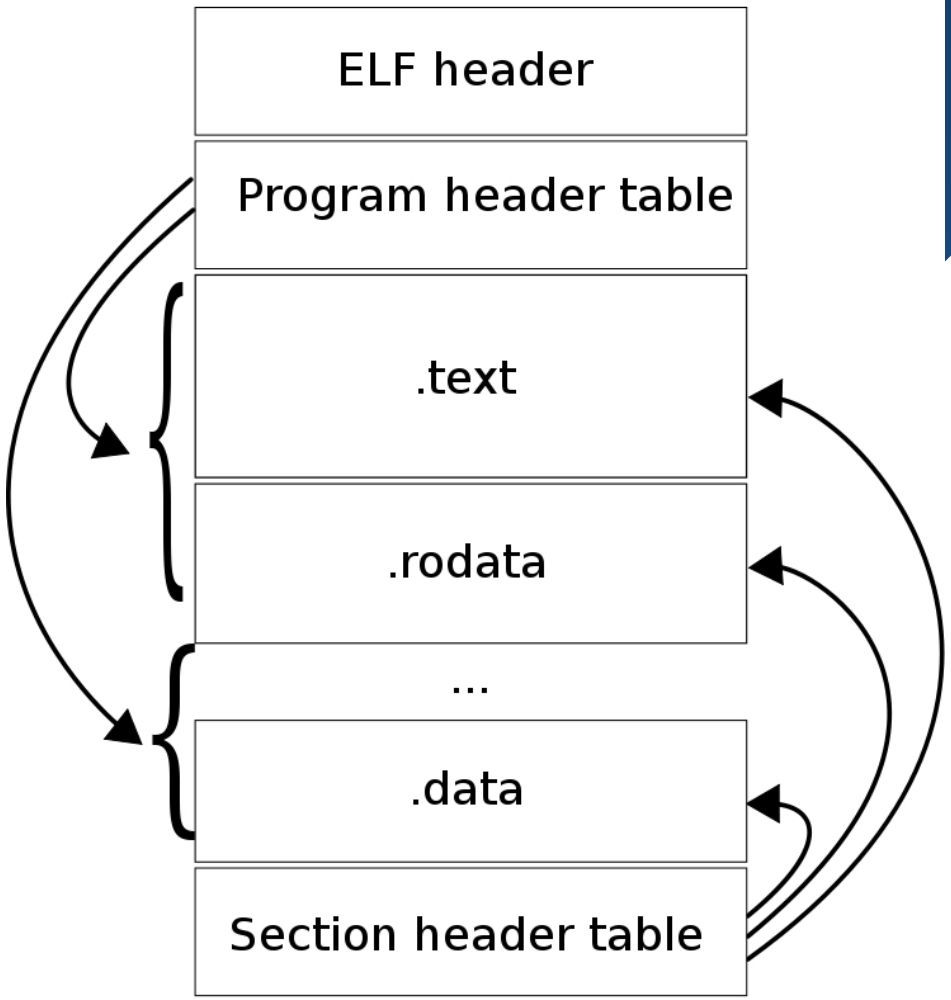
We are recruting ! Send me your CVs at :
Jonathan.brossard@ivizsecurity.com

iViZ
intelligent vision

Roadmap

- A (short) reminder of the ELF file format
- Introducing the problem
- How (not) to work with proprietary binaries anyway ?
- What to rebuild ?
- Refactoring the binary
- Refactoring in practice

A (short) reminder of the ELF format



A (short) reminder of the ELF format

The ELF header : (mandatory)

```
typedef struct {
unsigned char   e_ident[EI_NIDENT];
Elf32_Half      e_type;
Elf32_Half      e_machine;
Elf32_Word      e_version;
Elf32_Addr      e_entry;
Elf32_Off       e_phoff;    // offset to Program  Header Table
Elf32_Off       e_shoff;    // offset to Section  Header Table
Elf32_Word      e_flags;
Elf32_Half      e_ehsize;
Elf32_Half      e_phentsize;
Elf32_Half      e_phnum;
Elf32_Half      e_shentsize;    // size of a section header
Elf32_Half      e_shnum;        // number of section headers
Elf32_Half      e_shtrndx;      // offset of associated string table
} Elf32_Ehdr;
```

A (short) reminder of the ELF format

Program Headers : (mandatory, one per segment)

```
typedef struct {
        Elf32_Word      p_type; // Segment type (Alocate ? Null ?
    Dynamic ? …)
        Elf32_Off       p_offset;  // offset in file
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz;      // length in file
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;
```

A (short) reminder of the ELF format

Section Headers : (optional, one per section)

```
typedef struct
{
    Elf32_Word      sh_name;// index in string table
    Elf32_Word      sh_type;  // type of section
    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;
```

A (short) reminder of the ELF format

Symbols : (the Symbol table is an array of Elf32_sym)

```
typedef struct
{
  Elf32_Word st_name;          // Symbol name (string tbl index)
  Elf32_Addr  st_value;        // Symbol value
  Elf32_Word st_size;          // Symbol size
  unsigned char    st_info;    // Symbol type and binding
  unsigned char    st_other;   // Symbol visibility
  Elf32_Section    st_shndx;   // Section index
} Elf32_Sym;
```

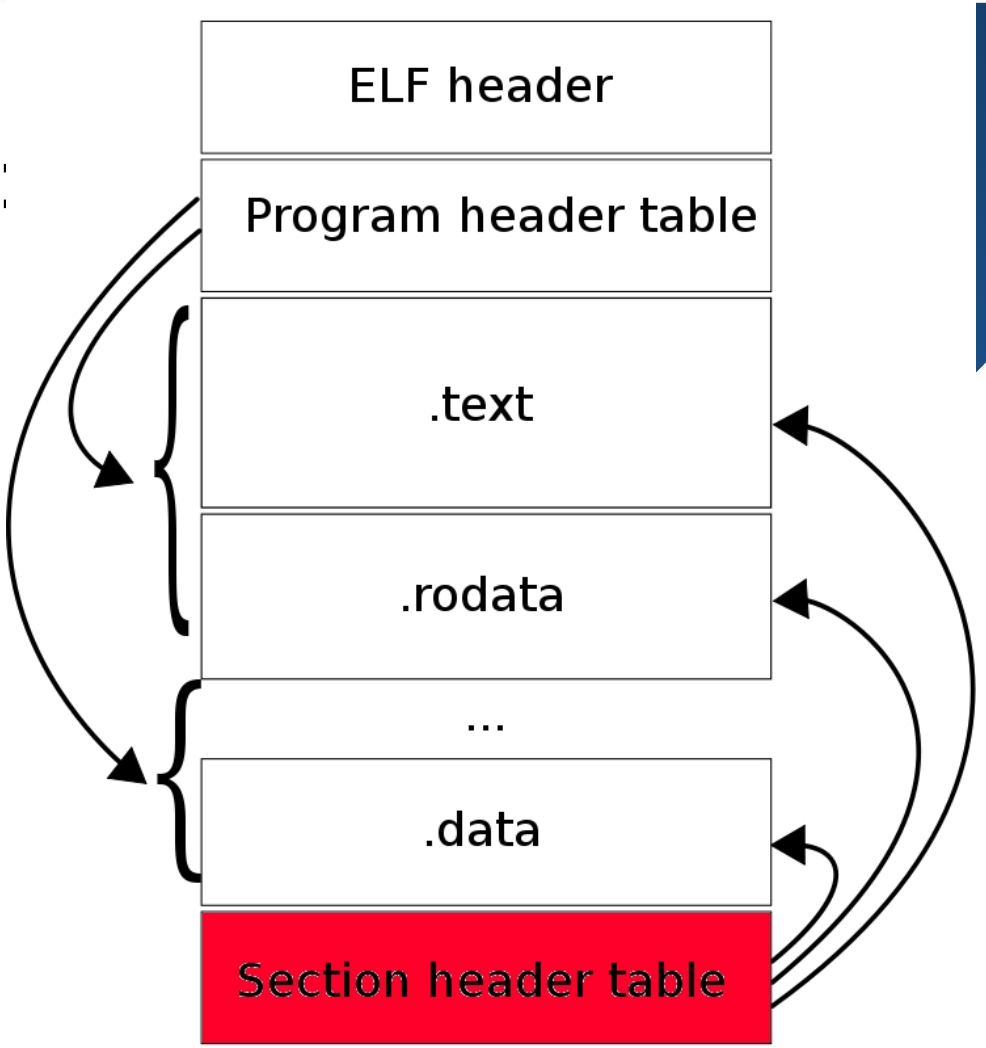
Introducing the problem

Proprietary binaries are commonly modified to make the job of security analysts difficult:

- Sometimes packed (out of topic)
- Usually don't have a symbol table (stripped)
- More and more have a missing/corrupted Section Header Table (sstripped, a la sstrip from elfkickers…) and/or zeroed Section Headers.

Introducing the problem

Before :



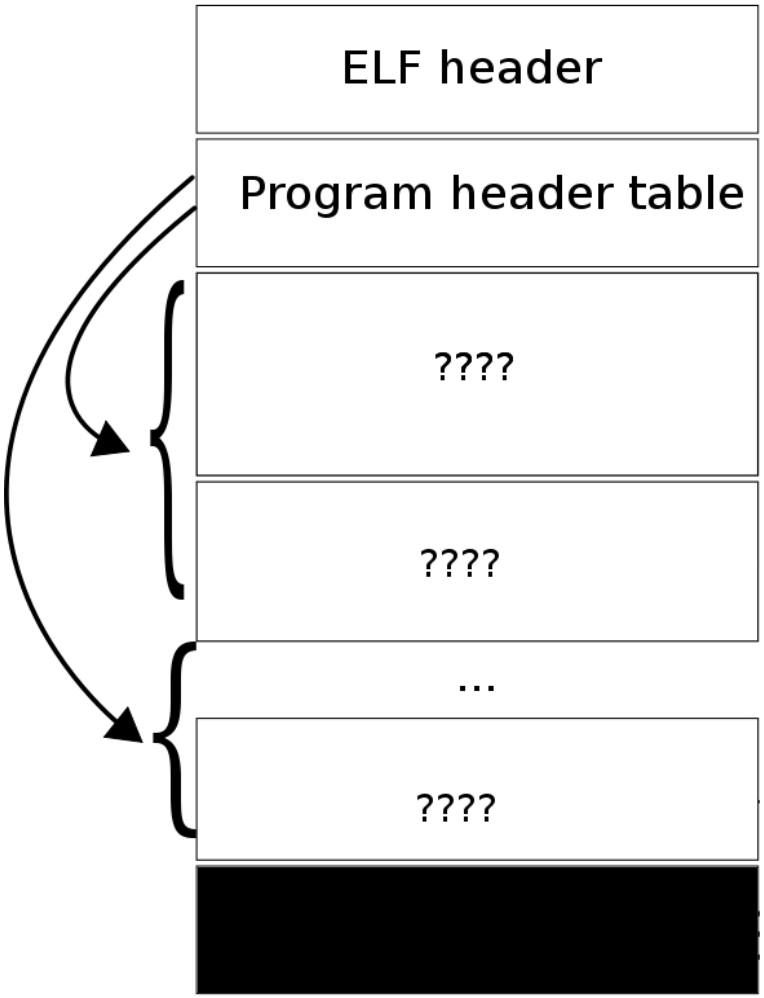
- We know where the Segments are

- We know where the Sections are located

- The application has a symbol table

Introducing the problem

After :



- We know where the Segments are : the loader/dynamic linker can still do their jobs

- We don't know where the Sections start/end

- The application has no symbol table

Introducing the problem

- Tools based on libbfd need to read the Section Headers to analyse it.
- Therefore, the handy GNU binutils utilities won't manage to analyze the target (readelf, objdump, objcopy, nm...)
- Debugging with gdb will be really uneasy :
  - no symbols, so no breakpoints on symbol names. :(
  - the application doesn't even have a “main”.
  How to get a prompt once the shared libraries are loaded ?

Introducing the problem

DEMO

iViZ intelligent vision

How (not) to work with proprietary binaries anyway ?

Use tools that aren't based on libbfd ?

- Fenris (M Zalewski) : rebuilds a symbol table for dynamically linked binaries (moderately interresting for us)
http://lcamtuf.coredump.cx/fenris/

- Elfsh from the Eresi project (attempts to rebuild the missing ELF section header and a symbol table) plus its debugger, tracer…
http://www.eresi-project.org/

How (not) to work with proprietary binaries anyway ?

The problem with existing tools...

DEMO

Hrm... so we will code our own ;)

iViZ intelligent vision

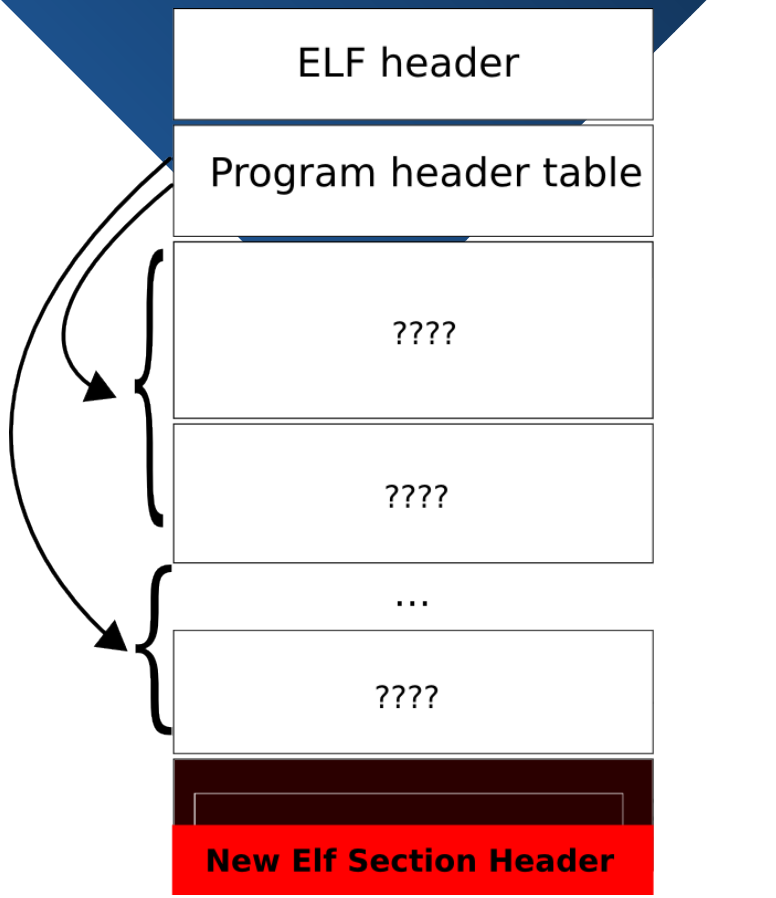
What to rebuild ?

- Instead of rewriting ELF parsers and debuggers, the idea is to refactor the binary as little as possible (do not modify the .data or .text for instance) to make it usable by the standard tools we may need (libbfd based tools like the ones of binutils, GDB, etc).
- We need a Section Header Table and Section Headers (and infos on the sections to populate them !) for all the relevant sections.
- We need a symbol table with labels for every function/control structure

Refactoring the binary :

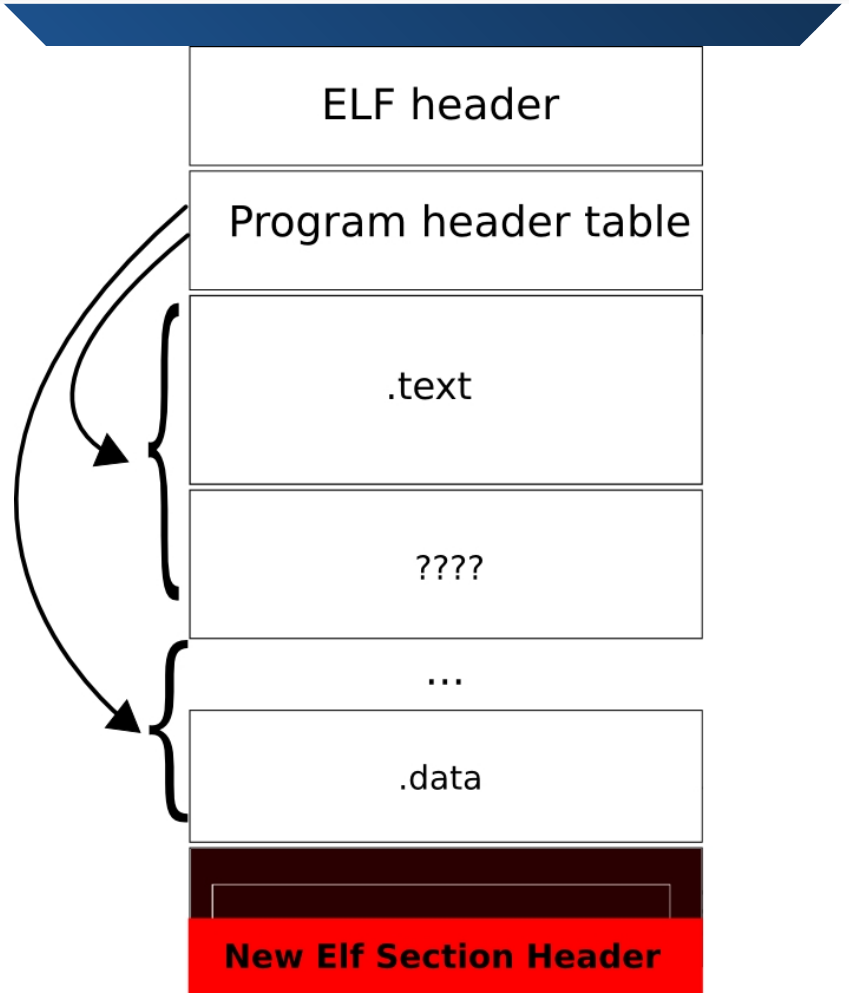
Increase the size of the binary to contain a new Section Header Table

Modify the ELF Header to point to our new Section Header Table (via e_shoff)



Refactoring the binary

retrieve information about the sections start/end (make a wild guess or use heuristics when possible)



Refactoring the binary

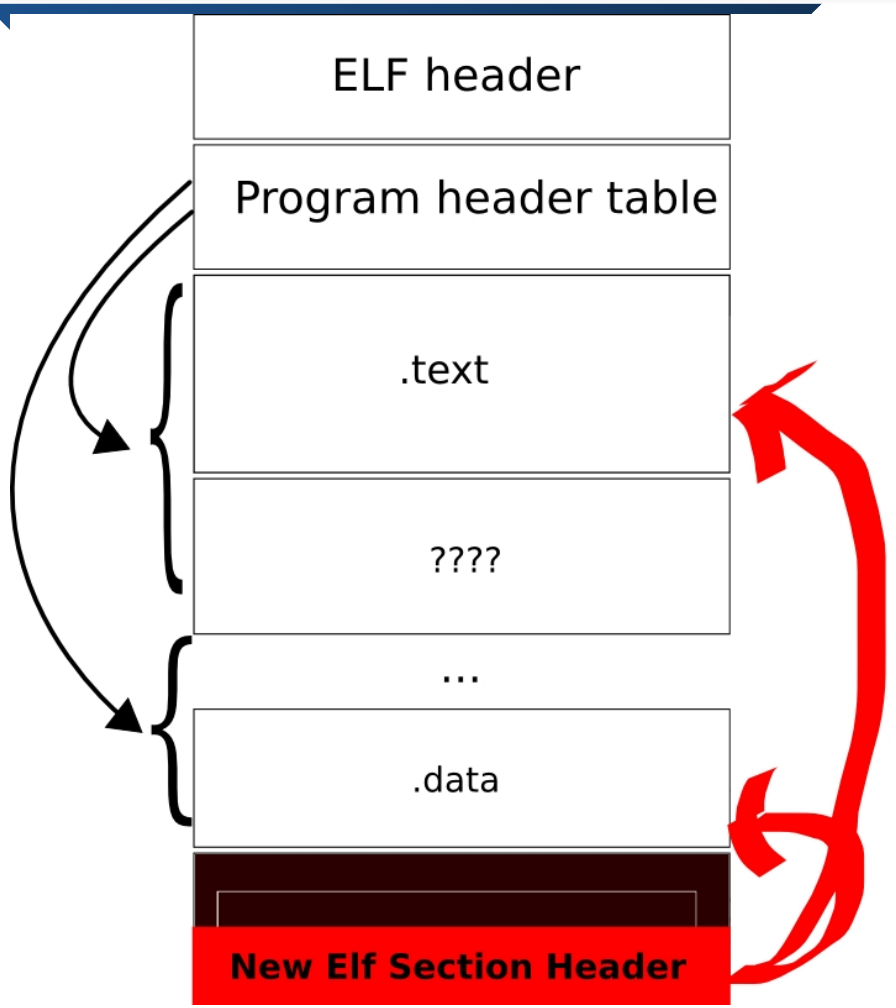
- Example of heuristics on Sections :

- Entry point points to .text
- Segment types and Flags give indications on their content
- Some sections are in a predictable order if the compiler is known
- Patterns of bytes can be found for some sections starts/ends (eg: .interp)

- NOTE: We don't care if 100% of the info is not correct !

Refactoring the binary

Allocate (append) and update Section Headers accordingly (don't forget to e_shnum++ in ELF Header).



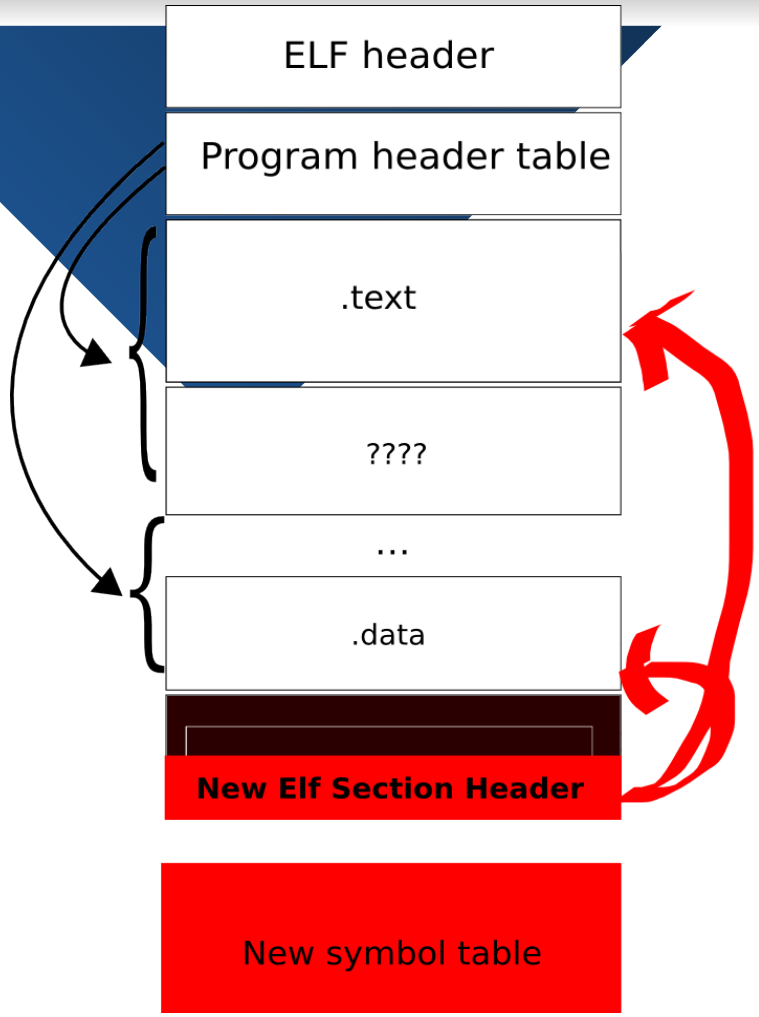
Refactoring the binary

We can now use the binary with our usual disassemblers using libbfd.

Disassemble the .text, and give names to the destination offsets of (un)conditional jumps and calls

Update this list with labels corresponding to predictable offsets (eg: main()) and the content of the .dynamic section

Add all those label/offset tuples to a symbol table (new section SHT_SYMTAB) at the end of the binary



Refactoring the binary

- Examples of heuristics :

1) Finding main()

```
objdump -d -j .text ./binary \
 2>/dev/null|tac|grep \
"__libc_start_main@plt" -A 1|grep
  push|grep \
 "0x[0-9a-fA-F]*" -o|awk '{print $0 "
  main"}'
```

Refactoring the binary

- Examples of heuristics :

2) Finding constructors

```
objdump -d -j .text ./ binary 2>/dev/null \
 |tac|grep \
 "bb [0-9a-fA-F][0-9a-fA-F] [0-9a-fA-F][0-9a \
-fA-F] 0[0-9a-fA-F] 08" -A 4|grep -w 55|grep \
 "[0-9a-fA-F][0-9a-fA-F]*" -o|head -n 1|sed \
s#"^0"##gi|awk '{print "0x" $0 " \
 __do_global_ctors_aux"}'
```

Refactoring the binary

- Examples of heuristics :

3) Finding destructors

```
objdump -d -j .text ./binary \
 2>/dev/null|tac|grep "80 3d [0-9a-fA-F][0-9a \
-fA-F] [0-9a-fA-F][0-9a-fA-F] 0[0-9a-fA-F] 08 \
 00" -A 10|grep -w 55|grep "[0-9a-fA-F][0-9a \
-fA-F]*" -o|head -n 1|sed s#"^0"##gi|awk \ '{print
  "0x" $0 " __do_global_dtors_aux"}'
```

Refactoring the binary

- It is worth noticing that the modifications we did on the binary affect non loadable parts of the binary only.
- In other words, the process actually loaded in memory is not changed : addresses in .text, stack or heap won't be modified (luckily from an exploit writer POV).
- We add information relevant to the auditor and its tools only : we don't really care if all information is not accurate (as long as it helps…)

Refactoring in practice

DEMO

Conclusion

- It is possible to unstrip (rebuild a symbol table) and even unsstrip (rebuild Section Headers) a binary.
- From a defensive point of view, it is not possible to remove more information from the binary without affecting its execution (eg: a binary without ELF header won't be loaded properly). Go for packers… or opensource :p
- We can now write exploits using our usual tools without caring about those "protective" alterations.

Greetings

- Abhisek and Nibin from the iViZ Research Team
- irc.pullthepplug.org #social, in particular Silvio Cesare and Mayhem for their ideas/tools/knowledge
- irc.blacksecurity.org
- The Clubhack staff for making the event happen
- You for coming to this talk ;)

Questions ?

Thank You!