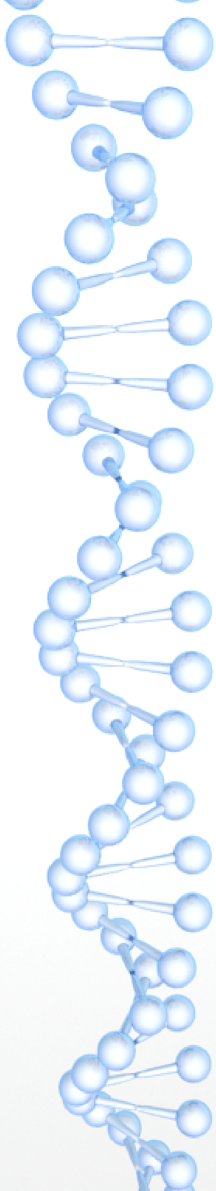


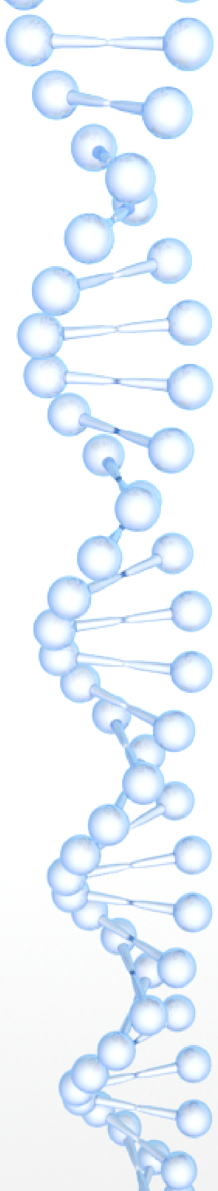
Libelfmaster, the future of intelligent binary parsing

- https://github.com/elfmaster/libelfmaster
 Open Source

- Ryan O’Neill AKA ElfMaster

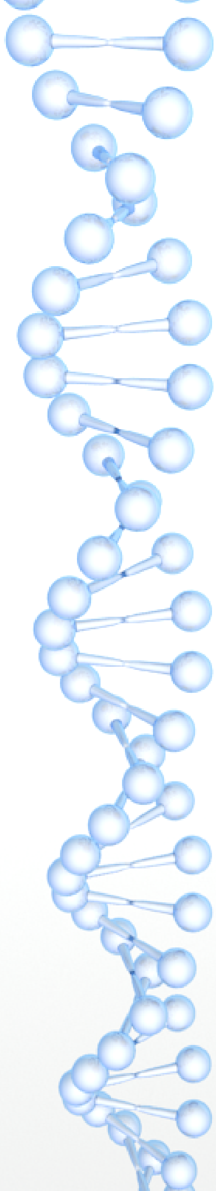
- ryan@bitlackeys.org

- Ryan.oneill@leviathansecurity.com



What are we discussing?

- The future of intelligent ELF binary parsing as it pertains to:
- Designing secure and innovative reverse engineering applications for multiple classes and architectures of ELF binaries
- The problems and pitfalls of existing parsing solutions
- The motivation and design intention behind libelfmaster
- The development of Arcana, an automated binary forensics software that is built using libelfmaster

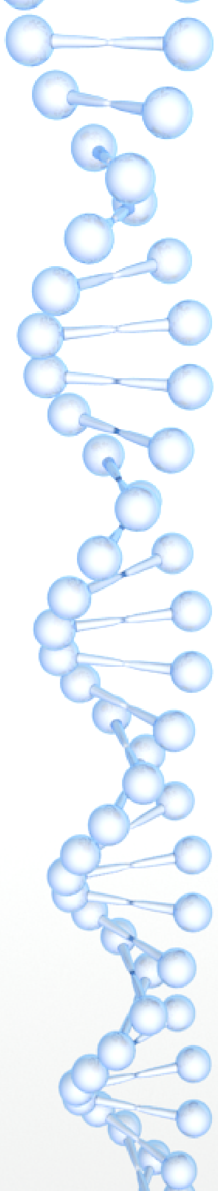


What prompted the design of libelfmaster

- A universal ELF parser built with innovation and for intuitive and easy use.
- Unique forensics reconstruction capabilities.
- Replacing the need for libelf and other libraries that are not able to handle reconstructing and parsing “broken” binaries
- A library that can be used to build Arcana. *(An automated binary analysis software for protection, detection, and classification of malware, backdoors and viruses in Linux.)*

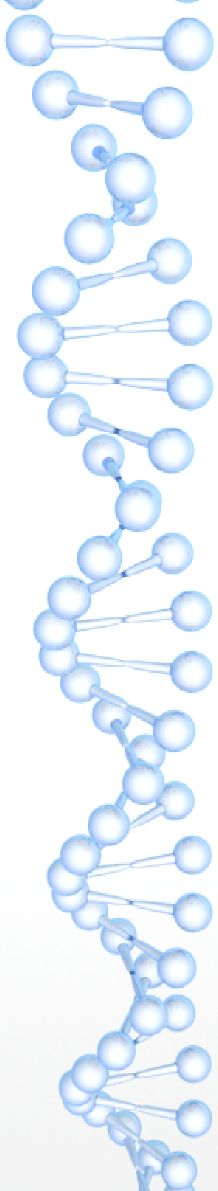
Cisco and Eurocom research on ELF malware

- An in-depth study of ELF malware was recently released by researchers: Emdel, Ivano, and several others http://www.s3.eurecom.fr/docs/oakland18_cozzi.pdf
- Cellular phones
- Misc. IOT devices
- Servers, workstations etc.



Problems with existing ELF parsing

- Unable to handle binaries that have been tampered with
- Invalid section header table offset’s etc.
- Inability to reconstruct section header tables
- Inability to reconstruct symbol tables
- Essentially off-the-shelf goto’s such as libelf break when used on malware
- Malware binaries avoid static analysis by exploiting parser differentials and parser vulnerabilities



Most common Malformed fields

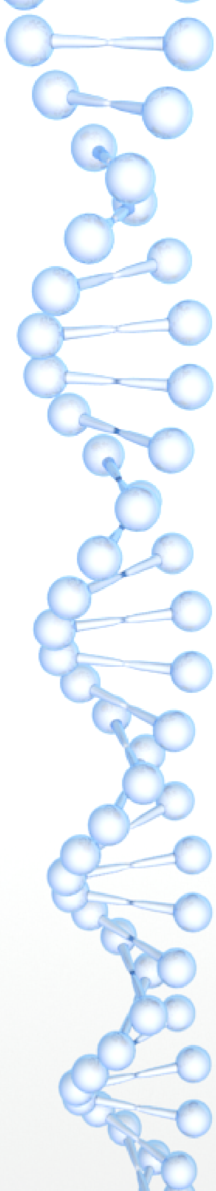
- Invalid e_shoff pointing outside of file
- Invalid e_shentsize that differs from sizeof(ElfN_Shdr)
- Invalid e_shnum extending beyond the number of section headers
- Invalid e_shstrndx pointing to an incorrect section index
- Invalid symbol table string offsets
- Invalid sh_link’s pointing outside e_shnum index
- Overlapping program header segments
- 5% of samples taken by the researchers had invalid section header table offsets

Observable impact in common software

- IDA Pro 7
- GDB (GNU Debugger)
- readelf (From GNU binutils package)
- pyelftools
- libelf
- The list goes on...

How does libelfmaster solve these problems?

- Flags passed to elf_open_object() determine how the file is parsed
- ELF_LOAD_F_STRICT
- ELF_LOAD_F_SMART
- ELF_LOAD_F_FORENSICS

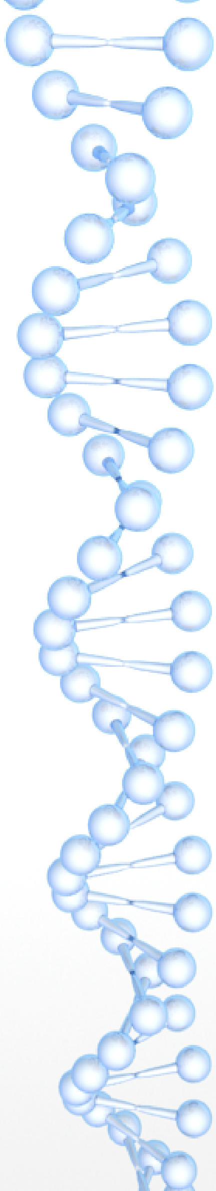


Strict binary parsing

- ELF_LOAD_F_STRICT
- Only parse an ELF object that has completely sane headers
- Cleanly and securely exit if there are any offsets or values that are not sane
- Very useful for software **that only expects sane binaries,** such as a linker which requires perfect sanity

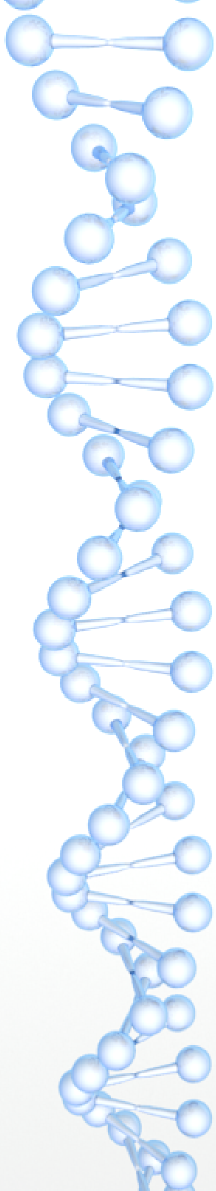
Smart binary parsing

- Parse only the headers that are sane
- Leave malformed headers alone
- Usually results acquiring a lot less information
- i.e. section headers are corrupted so a parser will only find the program header segments
- won’t crash on insane section headers, because it ignores parsing them
- Smart because it doesn’t crash, dumb in that it cannot reconstruct high-resolution ELF meta-data



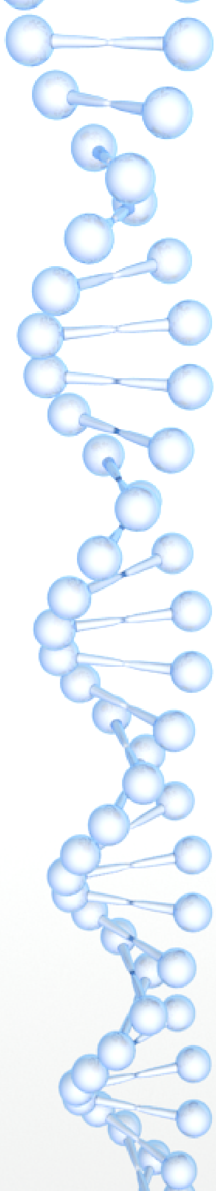
Forensic reconstruction binary parsing

- State-of-the-art techniques for reconstructing malformed binaries
- Reconstructs section header table: 90%
- Reconstructs dynamic symbol table: 100%
- Reconstructs STT_FUNC symbols: 90%
- Doesn’t actually add new sections and symbols to the binary file
- Stores them internally within the libelfmaster API
- Example? …



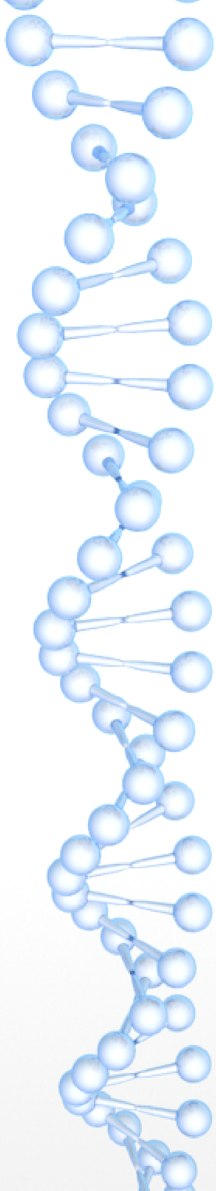
Readelf failing to get section headers

```
$ readelf -S test_stripped

There are no sections in this file.
readelf: Error: Reading 8272 bytes
extends past end of file for dynamic
string table
```



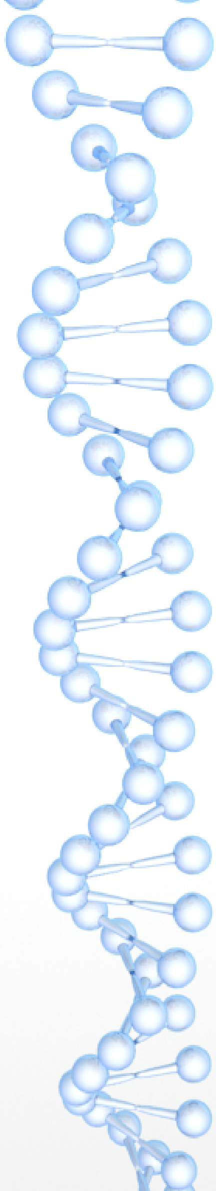
Libelfmaster program succeeding in section reconstruction

```
$ ./sections ./test_stripped
.gnu.hash: 0x400298-0x4002b8
.dynsym: 0x4002b8-0x400330
.dynstr: 0x400330-0x400373
.got.plt: 0x601000-0x601040
.plt: 0x400420-0x400450
.rela.plt: 0x4003d0-0x400400
.init: 0x400400-0x400420
.fini: 0x4005d4-0x400420
.text: 0x400238-0x4005d4
.init_array: 0x600e10-0x600e18
.fini_array: 0x600e18-0x600e20
.dynamic: 0x600e20-0x600ff0
.eh_frame_hdr: 0x4005ec-0x400628
.eh_frame: 0x40062c-0x400538
.symtab: 0-0
.strtab: 0-0
```



Readelf and nm tools fail on symbol table reconstruction

```
$ readelf -s test_stripped
readelf: Error: Reading 8272 bytes extends past end of
file for dynamic string table
Dynamic symbol information is not available for
displaying symbols.

$ nm -C test_stripped
nm: test_stripped: no symbols
```

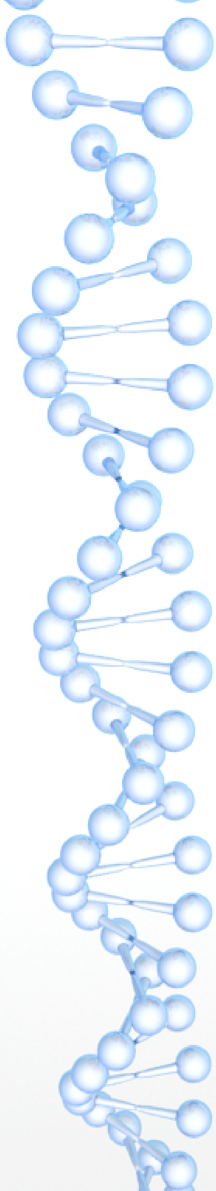


Libelfmaster symbol reconstruction example

```
$ ./symbols  test_stripped
pause: 0-0
__gmon_start__: 0-0
__libc_start_main: 0-0
puts: 0-0
: 0-0
sub_400420: 0x400420-0x400450
sub_400450: 0x400450-0x40046c
sub_400470: 0x400470-0x40049b
sub_4004a0: 0x4004a0-0x4004a2
sub_400560: 0x400560-0x4005c5
sub_4005d0: 0x4005d0-0x4005d2
```

Code example-- symbols.c

```
int main(int argc, char **argv)
{
        elfobj_t obj;
        elf_error_t error;
        elf_dynsym_iterator_t ds_iter;
        elf_symtab_iterator_t sm_iter;
        struct elf_symbol symbol;

        if (argc < 2) {
                printf("Usage: %s <binary>\n", argv[0]);
                exit(EXIT_SUCCESS);
        }
        if (elf_open_object(argv[1], &obj,
            ELF_LOAD_F_SMART|ELF_LOAD_F_FORENSICS, &error) == false) {
                fprintf(stderr, "%s\n", elf_error_msg(&error));
                return -1;
        }
        elf_dynsym_iterator_init(&obj, &ds_iter);
        while (elf_dynsym_iterator_next(&ds_iter, &symbol) == ELF_ITER_OK) {
                printf("%s: %#lx-%#lx\n",symbol.name, symbol.value,
                    symbol.value + symbol.size);
        }
        elf_symtab_iterator_init(&obj, &sm_iter);
        while (elf_symtab_iterator_next(&sm_iter, &symbol) == ELF_ITER_OK) {
                printf("%s: %#lx-%#lx\n",symbol.name, symbol.value,
                    symbol.value + symbol.size);
        }
        elf_close_object(&obj);
}
```

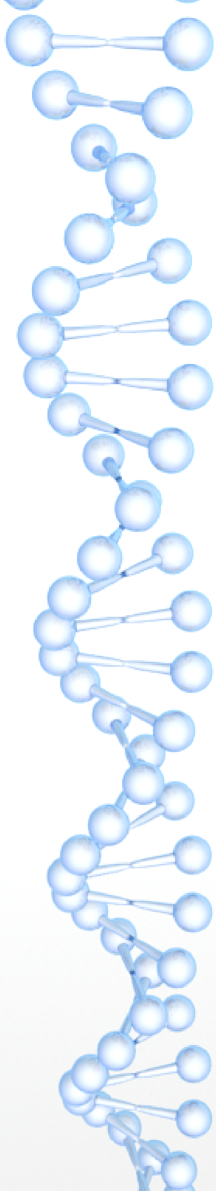


Forensics reconstruction with libelfmaster

- ELF_LOAD_F_FORENSICS
- Uses techniques similar to ECFS (extended core-file-snapshot technology) https://github.com/elfmaster/ecfs
- Still a work in progress, being fuzzed with AFL
- Requires 100 times the amount of sanity checking as ELF_LOAD_F_STRICT
- The Dynamic segment must be in-tact for reconstructing dynamic symbols
- The PT_GNU_EH_FRAME segment is used for locating the address and size of every local function.

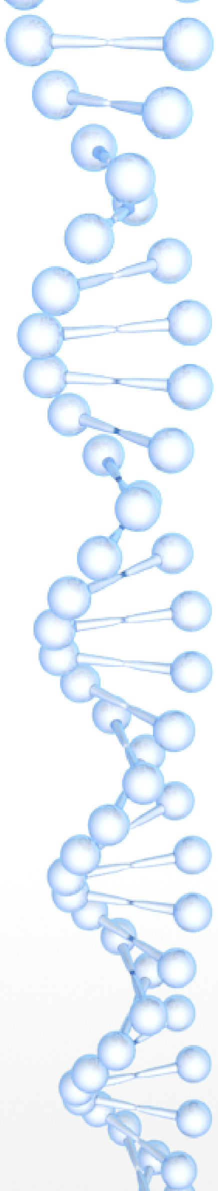
Forensics reconstruction continued

- Another well known “Progressive” parser that I will leave un-named, relies on in-tact section headers before it can reconstruct symbols
- Libelfmaster relies only on the bare-minimum components necessary to reconstruct section headers and symbols
- Libelfmaster support for binaries that use custom section header sizes is on the way, which is an intuitive leap forward



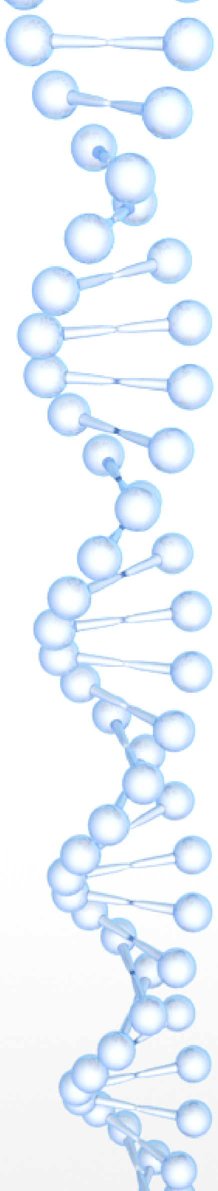
Reverse engineering tools may consider adopting libelfmaster for loading ELF objects

- Tools such as objdump, and even IDA are not able to forensically reconstruct sections & symbols
- Reverse engineering software will want to use the ELF_LOAD_F_FORENSICS
- We will show more examples of this later in the presentation



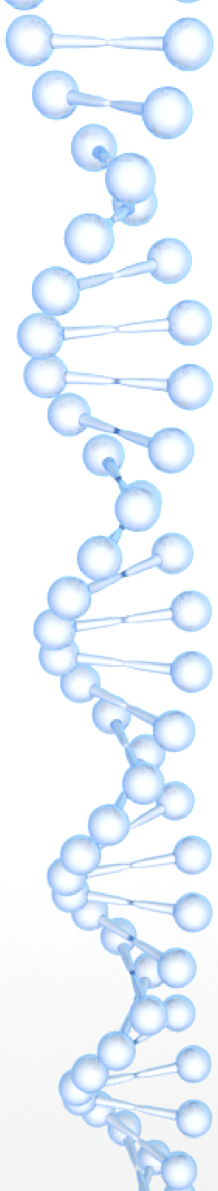
Libelfmaster encapsulation

- Simple API, seamlessly parses 32bit/64bit class binaries
- Abstracted out API based on simple iterators and accessor functions
- sophisticated tasks such as transitive shared library dependency iteration is as simple as using two functions
- `elfobj_t` maintains state of a single ELF object



Innovation and intuitive use

- The following slides will demonstrate some code that accomplishes non-trivial tasks with ease
- The examples directory contains tests and use-cases for libelfmaster, we will demonstrate several of these
- Checksec.sh re-written in C using libelfmaster
- ldd re-written in C using libelfmaster
- plt_dump.c which retrieves the actual PLT addresses for every symbol
- objdump_libelfmaster.c which correctly reconstructs sections and symbols for disassembly
- We will discuss Arcana, the future of binary forensics (malware analysis) for executables, shared libraries, kernel drivers, and core-dumps.



Ldd.c source code

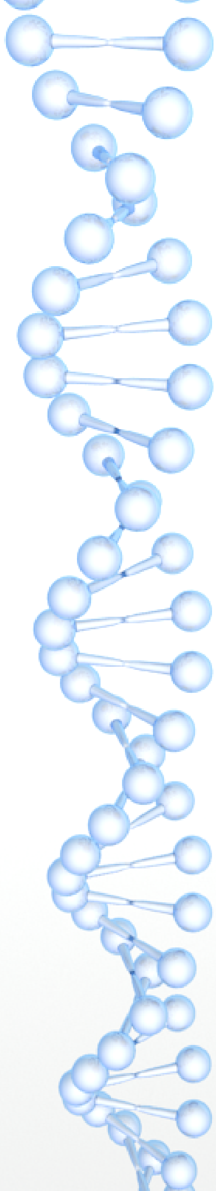
```
if (elf_open_object(argv[1], &obj, ELF_LOAD_F_FORENSICS, &error) == false) {
   fprintf(stderr, "%s\n", elf_error_msg(&error));
   return -1;
   }

   if (elf_shared_object_iterator_init(&obj, &so_iter,
       NULL, ELF_SO_RESOLVE_ALL_F, &error) == false) {
           fprintf(stderr, "elf_shared_object_iterator_init failed: %s\n",
           elf_error_msg(&error));
           return -1;
     }
   for (;;) {
           elf_iterator_res_t res;
           res = elf_shared_object_iterator_next(&so_iter, &object, &error);
            ... truncated ...
          if (res == ELF_ITER_OK) {
                 printf("%-30s -->\t%s\n", object.basename, object.path);
          } else if (res == ELF_ITER_NOTFOUND) {
                 printf("%-30s -->\t%s\n", object.basename, object.path);
           }
   }
   exit(0);
```

/bin/ldd

- Github.com/elfmaster/libelfmaster/tree/master/examples
- ldd.c
- elf_shared_object_iterator API
- ELF_SO_RESOLVE_F: Resolve top level basenames
- ELF_SO_RESOLVE_F: Recursively resolves all shared libraries

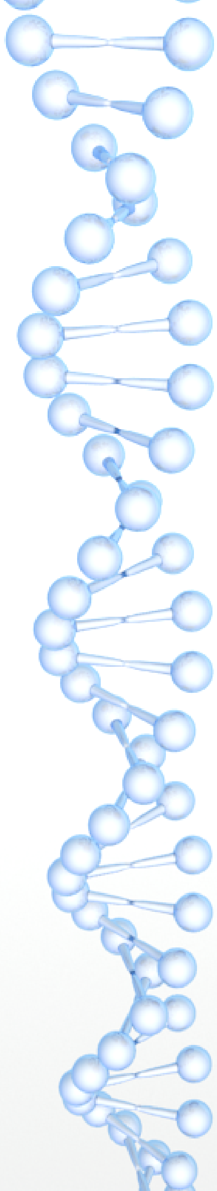
```
typedef struct elf_shared_object {
        char *basename;
        char *path;
} elf_shared_object_t;
```

- Still doesn’t support DT_RUNPATH/DT_RPATH



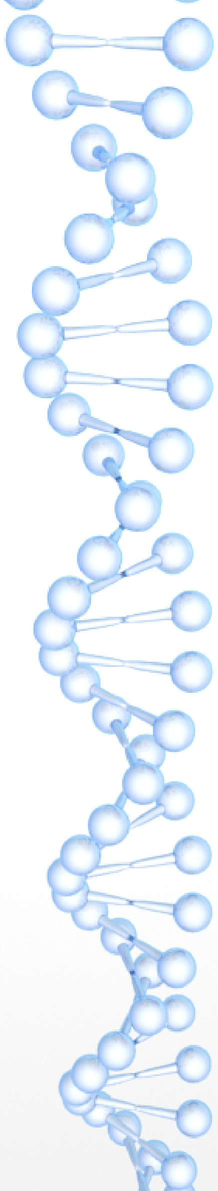
Ldd example

```
$ ./ldd /usr/sbin/sshd
libc.so.6                      --> /lib/x86_64-linux-gnu/libc.so.6
ld-linux-x86-64.so.2           --> /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
libcom_err.so.2                --> /lib/x86_64-linux-gnu/libcom_err.so.2
libpthread.so.0                --> /lib/x86_64-linux-gnu/libpthread.so.0
libkrb5.so.3                   --> /usr/lib/x86_64-linux-gnu/libkrb5.so.3
libk5crypto.so.3               --> /usr/lib/x86_64-linux-gnu/libk5crypto.so.3
libdl.so.2                     --> /lib/x86_64-linux-gnu/libdl.so.2
libkrb5support.so.0            --> /usr/lib/x86_64-linux-gnu/libkrb5support.so.0
libkeyutils.so.1               --> /lib/x86_64-linux-gnu/libkeyutils.so.1
libresolv.so.2                 --> /lib/x86_64-linux-gnu/libresolv.so.2
libgssapi_krb5.so.2            --> /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2
libcrypt.so.1                  --> /lib/x86_64-linux-gnu/libcrypt.so.1
libz.so.1                      --> /lib/x86_64-linux-gnu/libz.so.1
libutil.so.1                   --> /lib/x86_64-linux-gnu/libutil.so.1
libcrypto.so.1.0.0             --> /usr/lib/x86_64-linux-gnu/libcrypto.so.1.0.0
libsystemd.so.0                --> /lib/x86_64-linux-gnu/libsystemd.so.0
librt.so.1                     --> /lib/x86_64-linux-gnu/librt.so.1
liblzma.so.5                   --> /lib/x86_64-linux-gnu/liblzma.so.5
liblz4.so.1                    --> /usr/lib/x86_64-linux-gnu/liblz4.so.1
libgpg-error.so.0              --> /lib/x86_64-linux-gnu/libgpg-error.so.0
libgcrypt.so.20                --> /lib/x86_64-linux-gnu/libgcrypt.so.20
libselinux.so.1                --> /lib/x86_64-linux-gnu/libselinux.so.1
libpcre.so.3                   --> /lib/x86_64-linux-gnu/libpcre.so.3
libpam.so.0                    --> /lib/x86_64-linux-gnu/libpam.so.0
libcap-ng.so.0                 --> /lib/x86_64-linux-gnu/libcap-ng.so.0
libaudit.so.1                  --> /lib/x86_64-linux-gnu/libaudit.so.1
libaudit.so.1                  --> /lib/x86_64-linux-gnu/libaudit.so.1
libwrap.so.0                   --> /lib/x86_64-linux-gnu/libwrap.so.0
libnsl.so.1                    --> /lib/x86_64-linux-gnu/libnsl.so.1
```



PLT Entry addresses

- Figuring out the address of a shared library function’s PLT entry is somewhat tricky
- Requires parsing JUMP_SLOT relocation records found in `.rela.plt` section
- Requires matching up the symbol for each relocation record to the correspnding PLT stubs in the `.plt` section.



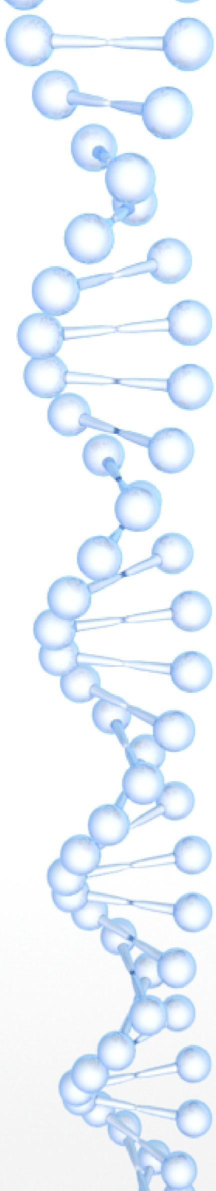
plt_dump.c example

```
elfobj_t obj;
elf_error_t error;
elf_plt_iterator_t iter;
struct elf_plt plt;

if (argc < 2) {
        printf("Usage: %s <binary>\n", argv[0]);
        exit(EXIT_SUCCESS);
}
if (elf_open_object(argv[1], &obj, ELF_LOAD_F_FORENSICS, &error) == false) {
        fprintf(stderr, "%s\n", elf_error_msg(&error));
        return -1;
}
elf_plt_iterator_init(&obj, &iter);
while(elf_plt_iterator_next(&iter, &plt) == ELF_ITER_OK)
        printf("%#08lx: %s\n", plt.addr, plt.symname);
    elf_close_object(&obj);
return 0;
```

PLT dump example

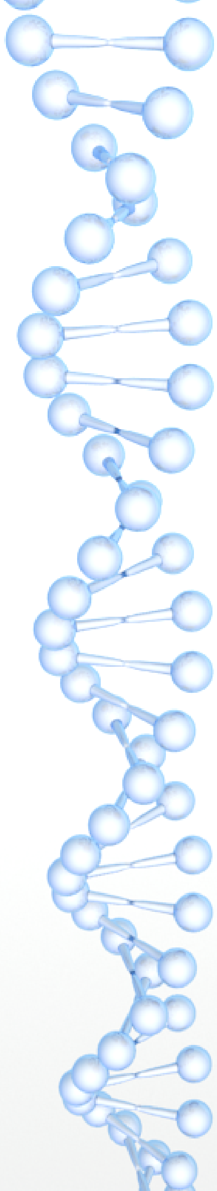
- Notice it prints them in reverse order; PLT-0 is always first. (I will fix this)

```
$ ./plt_dump test_stripped
0x400440: pause
0x400430: puts
0x400420: PLT-0
```



Checksec.sh re-written

- This version of checksec does not attach to processes like the original one
- Properly analyzes statically linked binaries for RELRO
- Supports PaX flags
- Supports SCOP (Secure code partitioning) a brand new binary mitigation, read about it here:
- http://www.bitlackeys.org/papers/secure_code_partitioning_2018.txt



Checksec example

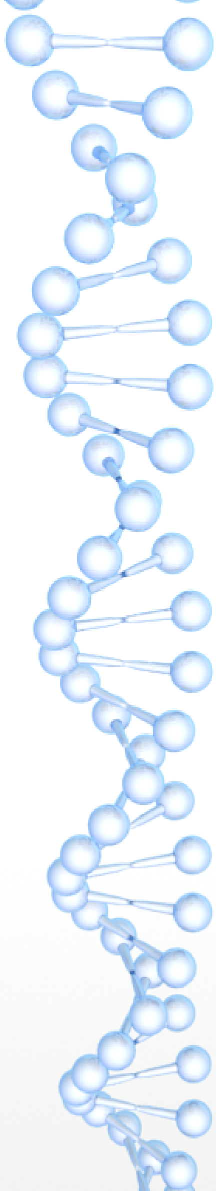
```
$ ./checksec test_scop
SCOP (Secure code partitioning) is enabled
RELRO: Full RELRO enabled
Stack canaries: Enabled
Full ASLR: Enabled
DEP: Enabled-- with PaX mprotect restrictions
PaX: |MPROTECT|RANDMMAP
```

SCOP support in libelfmaster

- SCOP is a new binary mitigation feature discovered by Justin Michael’s (sblip) and myself.
- Designed by the humble folk with GNU ldd/gcc
- This can break various parsers out there, especially when they make the (Once safe) assumption that the text segment and data segment are two contiguous segments.
- Libelfmaster aims to stay at the cutting edge of parsing, debugging, and code injection
- http://bitlackeys.org/papers/secure_code_partitioning_2018.txt

Objdump failure

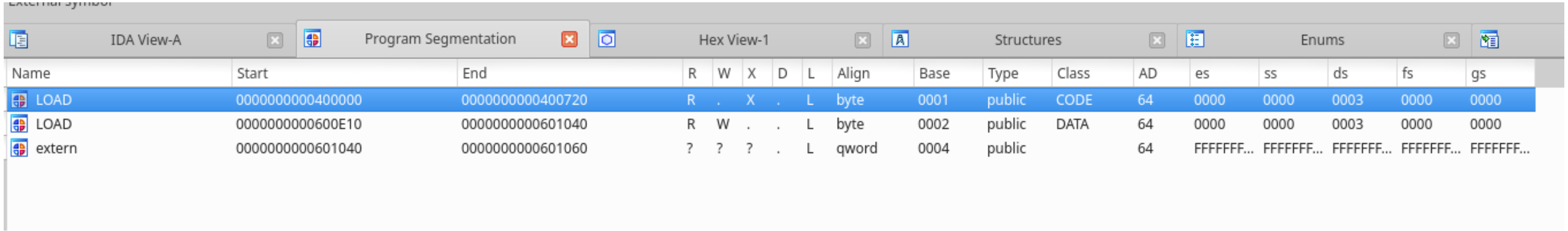
```
$ objdump -d test_stripped


test_stripped:     file format elf64-x86-64

$
```



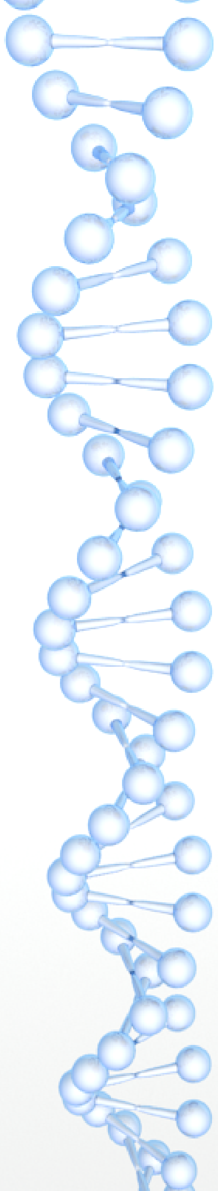
IDA Pro does not know sections

- IDA does seem to reconstruct dynamic symbols
- IDA Uses control flow analysis to find functions which will fail if the functions are encrypted
- IDA does not reconstruct any section headers and therefore it can only show the LOAD segments

IDA Pro only showing program segments



Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
LOAD	0000000000400000	0000000000400720	R	.	X	.	L	byte	0001	public	CODE	64	0000	0000	0003	0000	0000
LOAD	0000000000600E10	0000000000601040	R	W	.	.	L	byte	0002	public	DATA	64	0000	0000	0003	0000	0000
extern	0000000000601040	0000000000601060	?	?	?	.	L	qword	0004	public		64	FFFFFFF...	FFFFFFF...	FFFFFFF...	FFFFFFF...	FFFFFFF...



Libelfmaster disassembler written in 5 minutes

- Using libcapstone and libelfmaster
- Against the same binary that objdump refused to disassemble
- And that IDA could not find the section headers of

… next slide ...

examples/objdump_libelfmaster.c

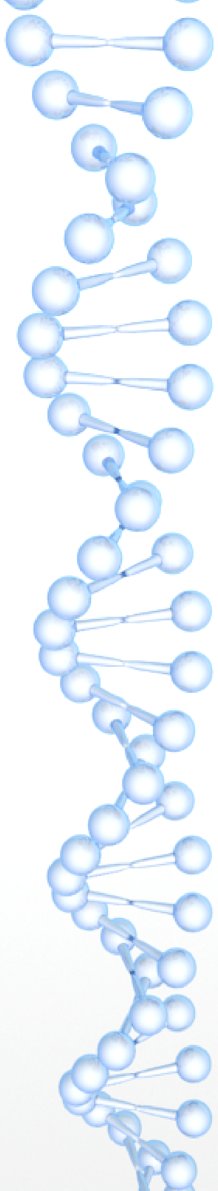
```
.plt:sub_400420:0x400426:      jmp        qword ptr [rip + 0x200be4]
.plt:sub_400420:0x40042c:      nop        dword ptr [rax]
.plt:sub_400420:0x400430:      jmp        qword ptr [rip + 0x200be2]
.plt:sub_400420:0x400436:      push        0
.plt:sub_400420:0x40043b:      jmp        0x400420
.plt:sub_400420:0x400440:      jmp        qword ptr [rip + 0x200bda]
.plt:sub_400420:0x400446:      push        1
.plt:sub_400420:0x40044b:      jmp        0x400420
.text:sub_400450:0x400450:      lea       rdi, qword ptr [rip + 0x18d]
.text:sub_400450:0x400457:      sub        rsp, 8
.text:sub_400450:0x40045b:      call      0x400430
.text:sub_400450:0x400460:      call      0x400440
.text:sub_400450:0x400465:      xor        eax, eax
.text:sub_400450:0x400467:      add        rsp, 8
.text:sub_400450:0x40046b:      ret
```

Initial conception of libelfmaster 2016

- Original inspiration for Libelfmaster was to write a parsing Library capable and suited for parsing malware
- As mentioned in previous slides malware binaries are often malformed
- I have written dozens of ELF parsers for separate projects I decided it was time to write the one-for-all parser
- Specifically I wanted to write Arcana, software for detecting ELF anomalies, backdoors, and viruses within all ELF object types
- Lets discuss Arcana some...

Original UNIX/Linux anti-virus (AVU)

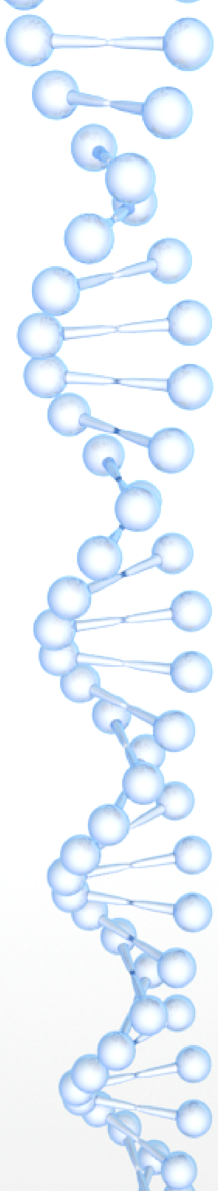
- 2008 – http://www.bitlackeys.org/projects/avu32.tgz
- Detects and disinfects binary viruses, and some memory viruses
- Unpacks UPX dynamically
- A prototype, that was purely for the purpose of research
- I re-wrote another Naive version of this and named it Arcana in 2015
- Original version of Arcana works on executables, shared libraries, and kernel drivers.
- Cannot handle edge cases, forensics reconstruction, and is a poorly written prototype
- Fast-forward to the present….



Arcana 2018-2019

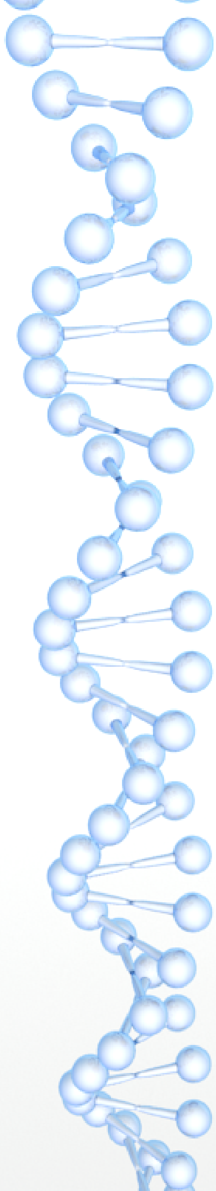
- Advanced ELF malware analysis technology for executables, shared libraries, kernel drivers
- Compliments ECFS https://github.com/elfmaster/ecfs

  and will eventually accept ECFS snapshots for analysis
- Detects sophisticated binary backdoors, trojans, and Viruses.
- Classification of malware based on infection techniques, code analysis, system calls, runtime behaviors, scan strings, etc.
- Plugin interface for easily adding new modules, i.e. a plugin that uses Unicorn emulator to further analyze identified parasite code



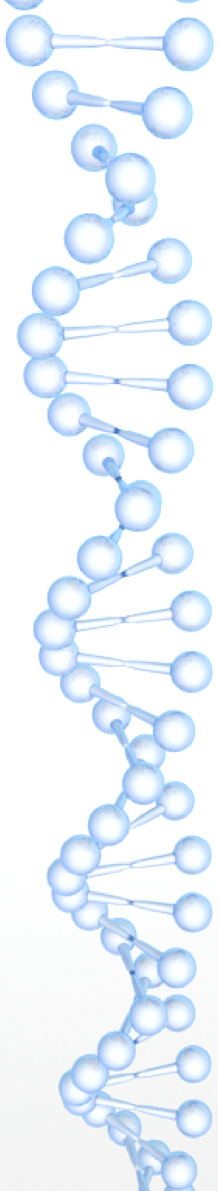
Some examples of detection features

- **Detection of many types of hooks including:**
- .got.plt hooks (PLT/GOT poisoning)
- __libc_start_main R_ARCH_GLOB_DAT relocation hook
- Initial entry point hook (i.e. ehdr→e_entry)
- .ctors/.dtors (aka .init_array/.fini_array)
- Function trampolines



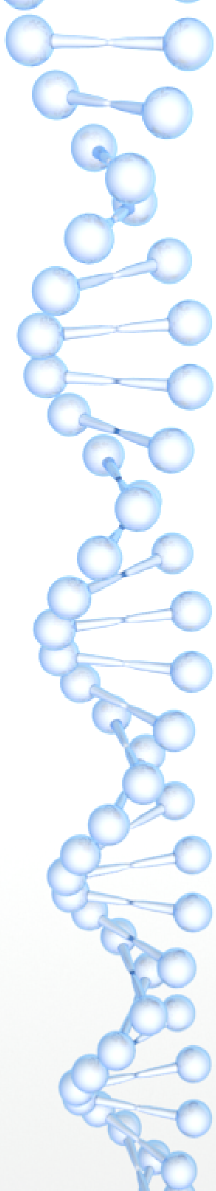
Detection of various infection types

- Text padding infections
- Reverse text infections
- Data segment infections
- PT_NOTE to PT_LOAD conversions
- PT_LOAD additions
- Takes SCOP (Secure code partitioning into consideration)



Addresses all types of ELF files

- Executables
- Shared libraries (Similar to Executable infections)
- Kernel driver infections – http://www.phrack.org/archives/issues/68/11.txt
- Eventually ECFS snapshots
- Compliments existing kernel malware analysis solutions working together as a suite with http://www.bitlackeys.org/#ikore



Prevents infected files from executing

- Programs that have been modified or are new to the system will be scanned before execution
- sys_exec(“malware.elf, args”);
- binfmt_elf.c
- This feature would have prevented me from running http://www.bitlackeys.org/#skeksi (Linux virus) on my system as root.
- Don’t ask...

A demo of the Arcana from 2015

- Running it against JPANIC’s Retaliation Virus: http://www.bitlackeys.org/#retaliation

```
$./arcana -e ../infected/jp-retal-e
-= [../infected/jp-retal-e] =-
ELF Program header [0] has segment perms [0x00000007] that violate W^X DEP
ELF Program header [10] at 0x803129 is suspicious because its not the text or data segment
ELF File header: Invalid entry point (outside of text segment): 0x80f56f
[!] A strange LOAD segment [unknown-segment-0: 0x803129] has been found with the following characteristics:
[!] segment unknown-0: has execution permissions
[!] segment unknown-0: has write permissions
[!] segment unknown-0: has read permissions
[!] The PT_NOTE segment has been changed to an 'unknown' PT_LOAD segment: unknown-0
... It is highly likely that this segment contains parasitic or malicious code
[!] The segment: unknown-0, has write+execute flags.. this may indicate malware, packers, polymorphic code etc.
[!] Suspicious program entry point detected: 0x80f56f does not point into the .text section as expected
[!] The entry point address 0x80f56f is pointing to a location within the '' section
[!] Suspicious program entry point detected: 0x80f56f does not point into the text segment
[!] It is pointing into segment: unknown-0
```

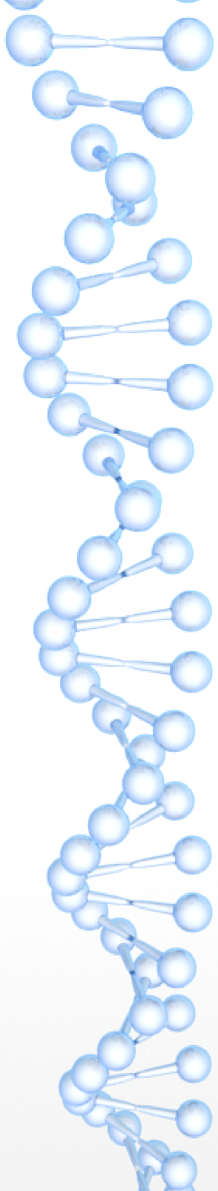
Run it on Skeksi Virus

- https://github.com/elfmaster/skeksi_virus
- `$ ./arcana -e ../infected/host1`
- `-= [../infected/host1] =-`
- `ELF Program header [0] has an invalid p_align: 00200000`
- `[!] suspicious constructor pointer 0x400550`
- `[!] suspicious destructor pointer 0x400530`
- `[!] Suspicious program entry point is smaller than expected entry 0x400000, this is a common sign of: reverse text-segment padding infection`

Run on lpv (Linux padding virus)

- http://www.bitlackeys.org/projects/lpv.c

```
./arcana32 -e ../infected/text_padding/host

-= [../infected/text_padding/host] =-

[!] Suspicious program entry point detected: 0x80485b8
does not point into the .text section as expected

[!] The entry point address 0x80485b8 is not pointing
into any valid section

-= [FINAL REPORT]: The binary file
'../infected/text_padding/host' has been analyzed and is
infected
```

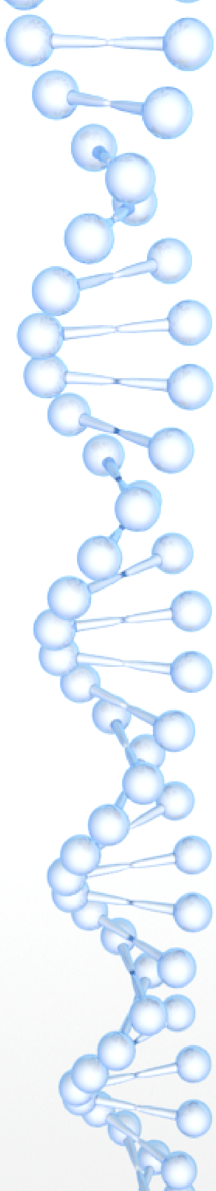


Why do I need to worry about Linux viruses?

- Although Linux viruses are a very real thing…
- Virus technology is used more commonly to create sophisticated rootkits, backdoors, and trojans
- Think of things like key-loggers, and very stealth backdoors that are too sophisticated or esoteric for existing Linux malware products to detect

What about large-sets of malware samples?

- Some of the top researchers in this area have been kind enough to give me over a thousand ELF samples to test
- Also testing with theoretical malware that I have not seen used in the wild (But suspect exists).
- Thinking outside of the box

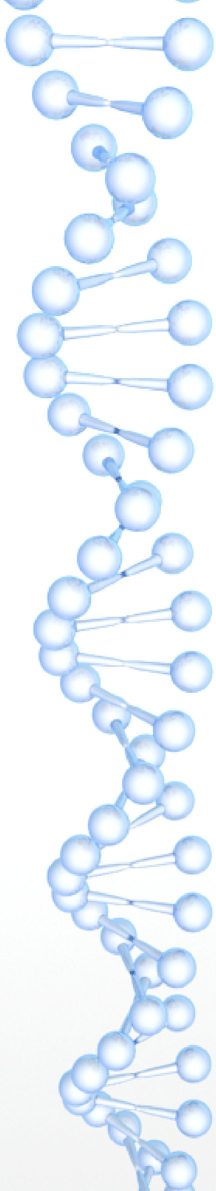


Where is REPO for the new Arcana built with libelfmaster?

- I was hoping to have more of it completed by this talk
- It will be developed quite rapidly because libelfmaster was tailored to design applications such as Arcana
- https://github.com/elfmaster/elf.arcana currently private

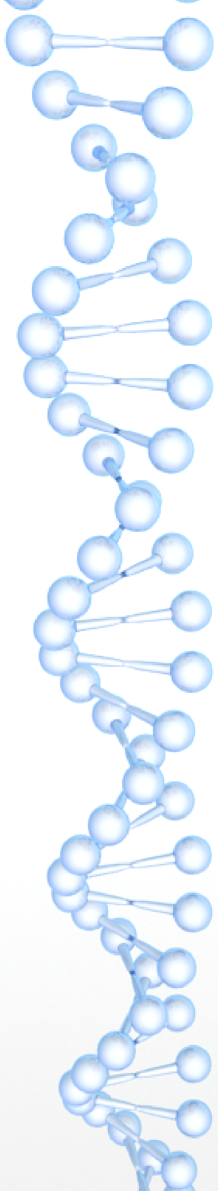
Libelfmaster injection

- @ulexec has been spearheading the instrumentation and injection features of libelfmaster
- Look forward to injection, infection, and instrumentation methods that have not yet been published to my knowledge



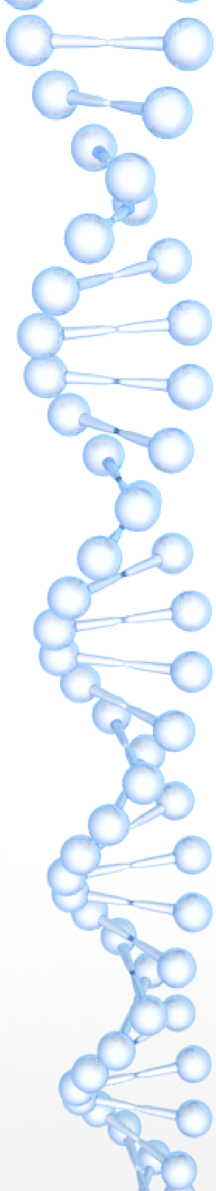
Libelfmaster Python bindings

- Emdel and Ivano (Cisco Malware researchers) are taking on writing the python bindings when time permits
- Initial python bindings created by Kaizikou (Josh)



Other future features

- Purely userland debugging API (No ptrace) similar to ERESI e2dbg
- Much more work needs to be completed, and will develop organically as people use it; necessity is the mother of all invention
- Full support (vs. partial) of other architectures. ARM is first on the list.



Questions?

- https://github.com/elfmaster/libelfmaster
- https://github.com/elfmaster/ecfs
- https://github.com/elfmaster/skeksi_virus
- http://www.bitlackeys.org/projects/avu32.tgz
- http://www.bitlackeys.org/#retaliation
- https://github.com/elfmaster/skeksi_virus
- Ryan.oneill@leviathansecurity.com
- ryan@bitlackeys.org