Melkor fuzzing rules based on specification violations of:

- Tool Interface Standard (TIS) ELF Specification 1.2 (May 1995)
- ELF-64 Object File Format 1.5 (May 1998)

and:

- Ideas & Considerations
- ELF Parsing Patterns

Alejandro Hernández H. (nitr0us)
@nitr0usmx
http://www.brainoverflow.org

Metadata		Number of Rules
HDR	Header	**19**
PHT	Program Header Table	**22**
SHT	Section Header Table	**37**
STRS	String Table	**3**
DYN	Dynamic Section	**18**
NOTE	Note Section	**4**
SYM	Symbols Table	**15**
REL	Relocations Table	**3**
HASH	Hash Table	**2**
ENV	OS Environment Variables	**3**
	Total:	**126**

ELF Specification 1.2 Violations
XXX: Value to be fuzzed with semi-valid semantics

Rule	Specification	Violation description	ELF metadata	Page
hdr1	A program header table, if present, tells the system how to create a process image. Files used to build a process image (execute a program) must have a program header table;	Executable ELF without a PHT	- HDR e_type = ET_EXEC | ET_DYN e_phoff = 0 e_phentsize = 0 e_phoff = Valid offset e_phnum = 0 e_phentsize = Valid number	16
hdr2	A section header table contains information describing the file's sections. Every section has an entry in the table; Files used during linking must have a section header table;	Relocatable file without a SHT Empty SHT	- HDR e_type = ET_REL e_shoff = 0 e_shentsize = 0 e_shoff = Valid offset e_shnum = 0 e_shentsize = Valid number	16
hdr3	e_type This member identifies the object file type. Values from ET_LOPROC through ET_HIPROC (inclusive) are reserved for processor-specific semantics. Other values are reserved and will be assigned to new object file types as necessary	ELF type set to normal values (< 5), invalid and uncommon values (>= 5) or zero.	- HDR e_type < 5 e_type >= 5 <= ET_HIPROC e_type = 0	19
hdr4	e_machine This member's value specifies the required architecture for an individual file	ELF machine with invalid / uncommon values	- HDR e_machine > 16 e_machine = 0	19
hdr5	e_entry If the file has no associated entry point, this member holds zero.	Invalid entry point (0 or out of range) Some point in kernel-land	- HDR e_entry = XXX e_entry = 0	19
hdr6	e_phoff This member holds the program header table's file offset in bytes. If the file has no program header table, this member holds zero.	PHT out of bounds	- HDR e_phoff = XXX	19
hdr7	e_ehsize This member holds the ELF header's size in bytes.	Random ELF header size	- HDR e_ehsize = XXX	19
hdr8	e_phentsize This member holds the size in bytes of one entry in the file's program header table; all entries are the same size. e_phnum This member holds the number of entries in the program header table. Thus the product of e_phentsize and e_phnum gives the table's size in bytes. If a file has no program header table, e_phnum holds the value zero.	Combination of low and high values e_phentsize to zero	- HDR e_phentsize = XXX e_phnum = XXX e_phentsize = 0	20
hdr9	e_shentsize This member holds a section header's size in bytes. A section header is one entry in the section header table; all entries are the same size. e_shnum This member holds the number of entries in the section header table. Thus the product of e_shentsize and e_shnum gives the section header table's size in bytes. If a file has no section header table, e_shnum holds the value zero.	Combination of low and high values e_shentsize to zero	- HDR e_shentsize = XXX e_shnum = XXX e_shentsize = 0	20
hdr10	e_shstrndx This member holds the section header table index of the entry associated with the section name string table. If the file has no section name string table, this member holds the value SHN_UNDEF.	Index to zero and out of bounds	- HDR e_shstrndx = XXX e_shstrndx = 0	20
hdr11	EI_CLASS The next byte, e_ident[EI_CLASS], identifies the file's class, or capacity.	ELF class with invalid / uncommon values	- HDR e_ident[EI_CLASS] > ELFCLASS64 e_ident[EI_CLASS] = 0	21
hdr12	EI_DATA Byte e_ident[EI_DATA] specifies the data encoding of the processor-specific data in the object file.	ELF encoding with invalid / uncommon values	- HDR e_ident[EI_DATA] > ELFDATA2MSB e_ident[EI_DATA] = 0	21
hdr13	EI_VERSION Byte e_ident[EI_VERSION] specifies the ELF header version number. Currently, this value must be EV_CURRENT, as explained above for e_version.	ELF version different than EV_CURRENT	- HDR e_version > EV_CURRENT e_version = 0 e_ident[EI_VERSION] > EV_CURRENT e_ident[EI_VERSION] = 0	21
hdr14	The ELF header's e_shoff member gives the byte offset from the beginning of the file to the section header table; e_shnum tells how many entries the section header table contains; e_shentsize gives the size in bytes of each entry	SHT offset out of bounds Combination of low and high values for sh_num and e_shentsize	- HDR e_shoff = XXX e_shnum = XXX e_shentsize = XXX	23
hdr15	SHN_LORESERVE This value specifies the lower bound of the range of reserved indexes.	ELF file with e_shnum = SHN_LORESERVE	- HDR e_shnum = SHN_LORESERVE	23
pht1	p_type This member tells what kind of segment this array element describes or how to interpret the array element's information.	Change the program header type with random common / invalid values Change the program headers type to cero	- PHT p_type = XXX p_type = 0	40
pht2	p_offset This member gives the offset from the beginning of the file at which the first byte of the segment resides.	Out of bounds Offset unaligned	- PHT (p_offset = XXX) % PAGESIZE = 0 (p_offset = XXX) % PAGESIZE != 0	40
pht3	p_vaddr This member gives the virtual address at which the first byte of the segment resides in memory. p_paddr On systems for which physical addressing is relevant, this member is reserved for the segment's physical address. This member requires operating system specific information, which is described in the appendix at the end of Book III.	Valid p_vaddr but invalid p_paddr and vice versa Invalid address Some point in kernel-land Addresses unaligned	- PHT p_vaddr = p_vaddr p_paddr = XXX p_vaddr = XXX p_paddr = p_paddr (p_vaddr = XXX) % PAGESIZE = 0 (p_paddr = XXX) % PAGESIZE = 0 (p_vaddr = XXX) % PAGESIZE != 0 (p_paddr = XXX) % PAGESIZE != 0	40
pht4	p_filesz This member gives the number of bytes in the file image of the segment; it may be zero. p_memsz This member gives the number of bytes in the memory image of the segment; it may be zero.	Bytes in memory but not in file Bytes in file but not in memory Loadable segment with cero bytes A big value of bytes in memory	- PHT p_filesz = 0 (p_memsz = XXX) % PAGESIZE = 0 p_filesz = XXX p_memsz = 0 p_filesz = 0 p_memsz = 0 p_memsz = 0xffffffff	40
pht5	p_align Loadable process segments must have congruent values for p_vaddr and p_offset, modulo the page size. This member gives the value to which the segments are aligned in memory and in the file. Values 0 and 1 mean that no alignment is required. Otherwise, p_align should be a positive, integral power of 2, and p_addr should equal p_offset, modulo p_align.	p_align not power of two (3,5,7,9,10,11,12,13,14,15,17, etc.)	- PHT p_align = PAGESIZE - 1 p_align = PAGESIZE +1 p_align = XXX	40
pht6	PT_LOPROC 0x70000000 PT_HIPROC 0x7fffffff	Program headers' types between these two constants	- PHT p_type >= PT_LOPROC <= PT_HIPROC	41
pht7	PT_LOAD The array element specifies a loadable segment, described by p_filesz and p_memsz. The bytes from the file are mapped to the beginning of the memory segment. If the segment's memory size (p_memsz) is larger than the file size (p_filesz), the "extra" bytes are defined to hold the value 0 and to follow the segment's initialized area. The file size may not be larger than the memory size.	More file image bytes than in memory A big value of bytes in file image	- PHT p_filesz > p_memsz p_filesz = 0xffffffff	41

pht8	Padding is present, if necessary, to ensure 4-byte alignment for the descriptor. Such padding is not included in namesz.	Make the note section's size unaligned	- PHT p_type = PT_NOTE p_filesz % 4 != 0	42
pht9	PT_INTERP The array element specifies the location and size of a null-terminated path name to invoke as an interpreter. This segment type is meaningful only for executable files (though it may occur for shared objects); it may not occur more than once in a file.	Point the path to itself Delete the PT_INTERP program header from the PHT Put two more PT_INTERP headers to the PHT	- PHT e_type = ET_EXEC | ET_DYN p_type != PT_INTERP String replacement to point to itself Algorithm to add two PT_INTERP before and after the original one	72
pht10	When the system creates loadable segments' memory images, it gives access permissions as specified in the p_flags member. All bits included in the PF_MASKPROC mask are reserved for processor-specific semantics.	Fuzz the flags combining the common values Add the PF_MASKPROC to the flags	- PHT p_flags = XXX p_flags |= PF_MASKPROC	73
pht11	For example, typical text segments have read and execute —but not write —permissions.	Locate the text segment and set the write flag	- PHT p_flags |= PF_W	74
pht12	A PT_DYNAMIC program header element points at the .dynamic section, explained in "Dynamic Section" below.	Change the PT_DYNAMIC header to point somewhere else	- PHT p_type = PT_DYNAMIC p_offset = XXX p_vaddr = XXX	75
pht13	PT_SHLIB This segment type is reserved but has unspecified semantics. Programs that contain an array element of this type do not conform to the ELF specification for UNIX System V.	Add a PT_SHLIB at the end of the PHT	- PHT PHT[e_phnum - 1] = PT_SHLIB	72
pht19	PT_PHDR The array element, if present, specifies the location and size of the program header table itself, both in the file and in the memory image of the program. This segment type may not occur more than once in a file. Moreover, it may occur only if the program header table is part of the memory image of the program. If it is present, it must precede any loadable segment entry. See "Program Interpreter" in the appendix at the end of Book III for further information.	Delete the PT_PHDR from the PHT Create an extra PT_PHT right after the first one found in the PHT Move the PT_PHDR to the end of the PHT (after the PT_LOAD segments)	- PHT p_type != PT_PHDR PHT[x].p_type = PT_PHDR PHT[x+1].p_type = PT_PHDR PHT[e_phnum - 1] = PT_PHDR	41
pht20	Loadable segment entries in the program header table appear in ascending order, sorted on the p_vaddr member.	Re-order the PT_LOAD segments in descending order Re-order the PT_LOAD entries randomly	- PHT Algorithm to re-order the PT_LOAD entries' p_vaddr address (descending or randomly)	41
pht21	PT_INTERP If it is present, it must precede any loadable segment entry. See "Program Interpreter" below for further information.	Move the PT_INTERP to the end of the PHT (after the PT_LOAD segments)	- PHT Algorithm to relocate the PT_INTERP to the end of the PHT. The program header in the end will take place where the original PT_INTERP is. PHT[PT_INTERP] = PHT[e_phnum - 1] PHT[e_phnum - 1] = PT_INTERP	72
sht1	sh_name This member specifies the name of the section. Its value is an index into the section header string table section [see "String Table" below], giving the location of a null-terminated string.	A sh_name out of bounds	- SHT sh_name = XXX	24
sht2	sh_addr If the section will appear in the memory image of a process, this member gives the address at which the section's first byte should reside. Otherwise, the member contains 0.	Invalid address Some point in kernel-land	- SHT sh_addr = XXX	24
sht3	sh_offset This member's value gives the byte offset from the beginning of the file to the first byte in the section. One section type, SHT_NOBITS described below, occupies no space in the file, and its sh_offset member locates the conceptual placement in the file.	A sh_offset out of bounds	- SHT sh_offset = XXX	25
sht4	sh_size This member gives the section's size in bytes. Unless the section type is SHT_NOBITS, the section occupies sh_size bytes in the file. Sections in a file may not overlap. No byte in a file resides in more than one section.	Combination of low and high values	- SHT sh_size = XXX	25
sht5	sh_addralign That is, the value of sh_addr must be congruent to 0, modulo the value of sh_addralign. Currently, only 0 and positive integral powers of two are allowed. Values 0 and 1 mean the section has no alignment constraints.	sh_addralign not power of two (3,5,7,9,10,11,12,13,14,15,17, etc.) sh_addralign = PAGESIZE +/- 1	- SHT sh_addralign = XXX sh_addralign = PAGESIZE - 1 sh_addralign = PAGESIZE + 1	25
sht6	sh_entsize Some sections hold a table of fixed-size entries, such as a symbol table. For such a section, this member gives the size in bytes of each entry.	Combination of low and high values Zero	- SHT sh_entsize = XXX sh_entsize = 0	25
sht7	A section header's sh_type member specifies the section's semantics.	Change the section types with random common / invalid values	- SHT sh_type = XXX	25
sht8	SHT_LOPROC through SHT_HIPROC Values in this inclusive range are reserved for processor-specific semantics. Section types between SHT_LOUSER and SHT_HIUSER may be used by the application, without conflicting with current or future system-defined section types.	Section types set to these four constants	- SHT sh_type = SHT_LOPROC + 1 sh_type = SHT_HIPROC sh_type = SHT_LOUSER + 1 sht_type = SHT_HIUSER	27
sht9	The section header for index 0 (SHN_UNDEF) exists, even though the index marks undefined section references. This entry holds the following. Figure 1-10. Section Header Table Entry: Index 0	SHT index 0 with random / uncommon values	- SHT SHT[0]: sh_* = XXX	27
sht10	A section header's sh_flags member holds 1-bit flags that describe the section's attributes.	sh_flags with random combinations of the valid flags sh_flags with invalid / undefined values	- SHT sh_flags = SHF_WRITE|SHF_ALLOC|SHF_EXECINSTR|SHF_MASKPROC sh_flags &= ~ SHF_WRITE sh_flags &= ~ SHF_ALLOC sh_flags &= ~ SHF_EXECINSTR sh_flags = XXX	27
sht11	SHT_DYNAMIC: sh_link = The section header index of the string table used by entries in the section; sh_info = 0 SHT_HASH: sh_link = The section header index of the symbol table to which the hash table applies; sh_info = 0	sh_link pointing to a valid SHT index but not to a string table sh_link pointing to 0 sh_link out of bounds sh_info with random values	- SHT sh_type = SHT_DYNAMIC sh_link = (1 to e_shnum - 1) != SHT_STRTAB sh_link = XXX sh_info = (1 to e_shnum - 1) sh_info = XXX sh_type = (SHT_HASH|SHT_GNU_HASH) sh_link = (1 to e_shnum - 1) != (SHT_SYMTAB|SHT_DYNSYM) sh_link = XXX sh_info = (1 to e_shnum - 1) sh_info = XXX	28
sht12	SHT_REL and SHT_RELA: sh_link = The section header index of the associated symbol table; sh_info = The section header index of the section to which the relocation applies.	sh_link and sh_info pointing to a valid SHT index but not to a symbol table nor relocation sh_link out of bounds sh_link pointing to 0 sh_info out of bounds sh_info pointing to 0	- SHT sh_type = (SHT_REL|SHT_RELA) sh_link = (1 to e_shnum - 1) != (SHT_SYMTAB|SHT_DYNSYM) sh_link = XXX sh_info = (1 to e_shnum - 1) sh_info = XXX	28

sht13	SHT_SYMTAB and SHT_DYNSYM: sh_link = The section header index of the associated string table; sh_info = One greater than the symbol table index of the last local symbol (binding STB_LOCAL).	sh_link and sh_info with random values sh_link out of bounds sh_link pointing to 0 sh_info out of bounds sh_info pointing to 0	- SHT sh_type = (SHT_SYMTAB|SHT_DYNSYM) sh_link = (1 to e_shnum - 1) != (SHT_STRTAB) sh_link = XXX sh_info = (1 to e_shnum - 1) sh_info = XXX	66
sht14	.bss This section holds uninitialized data that contribute to the program's memory image. By definition, the system initializes the data with zeros when the program begins to run. The section occupies no file space, as indicated by the section type, SHT_NOBITS.	Fuzz its attributes Set a random size	- SHT sh_type = SHT_NOBITS sh_flags = XXX sh_size = XXX	29
sht15	.data and .data1 These sections hold initialized data that contribute to the program's memory image.	Fuzz its attributes	- SHT sh_type = XXX sh_flags = XXX sh_size = XXX	29
sht16	.hash This section holds a symbol hash table.	Fuzz its attributes	- SHT sh_type = (SHT_HASH | SHT_GNU_HASH) sh_flags = XXX sh_size = XXX sh_entsize = XXX	29
sht17	.debug This section holds information for symbolic debugging. The contents are unspecified. All section names with the prefix .debug are reserved for future use. .line This section holds line number information for symbolic debugging, which describes the correspondence between the source program and the machine code. The contents are unspecified.	Fuzz the attributes Fuzz the debugging information (DWARF). GCC adds debug_*	- SHT sh_name = ".debug_*" sh_type = XXX sh_flags = XXX sh_size = XXX sh_entsize = XXX Fuzz its content. Debugging information. DWARF. [NOT IMPLEMENTED YET]	29
sht18	.dynamic This section holds dynamic linking information and has attributes such as SHF_ALLOC and SHF_WRITE. Whether the SHF_WRITE bit is set is determined by the operating system and processor.	Clear those flags	- SHT sh_type = SHT_DYNAMIC sh_flags &= ~ SHF_ALLOC sh_flags &= ~ SHF_WRITE	29
sht19	.rodata and .rodata1 These sections hold read-only data that typically contribute to a non-writable segment in the process image. See "Program Header" in Chapter 2 for more information.	Fuzz its attributes	- SHT sh_type = XXX sh_flags = XXX sh_size = XXX sh_entsize = XXX	30
sht20	.note This section holds information in the format that is described in the "Note Section" in Chapter 2. Padding is present, if necessary, to ensure 4-byte alignment for the descriptor. Such padding is not included in namesz.	Fuzz its attributes Sections's size less or equal the size of the struct Nhdr. Section's size unaligned	- SHT sh_type = SHT_NOTE sh_flags = XXX sh_size = sizeof(Elf32_Nhdr | Elf64_Nhdr) sh_size % 4 != 0 sh_addralign = XXX	30
sht21	.strtab This section holds strings, most commonly the strings that represent the names associated with symbol table entries. If a file has a loadable segment that includes the symbol string table, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.	Fuzz its attributes	- SHT sh_type = SHT_STRTAB sh_flags = XXX sh_size = XXX sh_entsize = XXX sh_addralign = XXX	30
sht22	.symtab This section holds a symbol table, as "Symbol Table" in this chapter describes. If a file has a loadable segment that includes the symbol table, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.	Fuzz its attributes	- SHT sh_type = SHT_SYMTAB | SHT_DYNSYM sh_flags = XXX sh_size = XXX sh_entsize = XXX sh_addralign = XXX	30
sht23	.text This section holds the "text," or executable instructions, of a program.	Change the section type to != SHT_PROGBITS and fuzz its attributes	- SHT sh_type != SHT_PROGBITS sh_flags &= ~ SHF_ALLOC sh_flags &= ~ SHF_EXECINSTR sh_flags = XXX sh_flags = 0	30
sht24	.fini & .init	Delete the SHF_EXECINSTR flag and/or SHF_ALLOC	- SHT sh_type = SHT_PROGBITS sh_flags &= ~ SHF_ALLOC sh_flags &= ~ SHF_EXECINSTR sh_flags = XXX sh_flags = 0	67
sht25	.interp This section holds the path name of a program interpreter. If the file has a loadable segment that includes the section, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.	Modify its attributes Point the path to itself	- SHT sh_type = SHT_NULL sh_flags &= ~ SHF_ALLOC sh_flags = XXX String replacement to point to itself	67
sht26	.got SHT_PROGBITS SHF_ALLOC+SHF_WRITE	Remove the SHF_WRITE permission	- SHT sh_type = SHT_PROGBITS sh_flags &= ~ SHF_WRITE	90
sht27	.plt SHT_PROGBITS SHF_ALLOC+SHF_EXECINSTR	Remove the SHF_EXECINSTR permission Leave the SHF_EXECINSTR permission but patch the second instruction (jmp) in the PLT to the original entrypoint. random address or .init or .fini: Disassembly of section .plt: 08048470 <printf@plt-0x10>: 8048470: ff 35 f8 9f 04 08 pushl 0x8049ff8 8048476: ff 25 fc 9f 04 08 jmp *0x8049ffc	- SHT sh_type = SHT_PROGBITS sh_flags &= ~ SHF_EXECINSTR plt + 6 = jmp entrypoint (HDR.e_entry) plt + 6 = jmp _init (.init.sh_addr) plt + 6 = jmp _fini (.fini.sh_addr) plt + 6 = XXX	90
sht28	An empty string table section is permitted; its section header's sh_size member would contain zero. Non-zero indexes are invalid for an empty string table.	Modify an string table different than e_shstrndx so this will be an empty string table or the offset at the end of the file with a random size	- SHT sh_type = SHT_STRTAB shndx != e_shstrndx sh_size = 0 sh_type = SHT_STRTAB shndx != e_shstrndx sh_offset = stat_info.st_size sh_size = 0x1337	31
sht29	SHT_HASH All objects participating in dynamic linking must contain a symbol hash table.	Delete the hash table	- SHT sh_type != (SHT_HASH | SHT_GNU_HASH)	66
sht30	An object file may have more than one section with the same name	Add the same section name more than two times in the file	- SHT sh_name = .text | .data | .got | .got.plt | .bss	30

str1	sh_name This member specifies the name of the section. Its value is an index into the section header string table section [see "String Table" below], giving the location of a null-terminated string. These single-byte characters use the 7-bit ASCII character set. Character values outside the range of 0 to 127 may occupy one or more bytes, according to the character encoding.	Fuzz the strings inside without deleting the NULL bytes	- STR sh_type = SHT_STRTAB Chars > 0x7f	24
str2	String table sections hold null-terminated character sequences, commonly called strings. The first byte, which is index zero, is defined to hold a null character. Likewise, a string table's last byte is defined to hold a null character, ensuring null termination for all strings.	Delete some NULL bytes between strings Change the first byte in the string table Delete the NULL byte from the end	- STR sh_type = SHT_STRTAB string_table[x] != 0 string_table[0] = XXX string_table[sh_size - 1] > 0x7f	31
str3	String table sections hold null-terminated character sequences, commonly called strings.	Fuzz every section that contains a string table with format string vulnerability triggers	- STR sh_type = SHT_STRTAB Replace the strings with format string vulnerability triggers such as %x or %n without deleting the NULL byte	31
note1	Note Section - Sometimes a vendor or system builder needs to mark an object file with special information that other programs will check for conformance, compatibility, etc. Sections of type SHT_NOTE and program header elements of type PT_NOTE can be used for this purpose. The note information in sections and program header elements holds any number of entries, each of which is an array of 4-byte words in the format of the target processor. namesz and name The first namesz bytes in name contain a null-terminated character representation of the entry's owner or originator. There is no formal mechanism for avoiding name conflicts. By convention, vendors use their own name, such as "XYZ Computer Company," as the identifier. If no name is present, namesz contains 0.	namesz with a high value	- NOTE namesz = XXX	42
note2	The first namesz bytes in name contain a null-terminated character representation of the entry's owner or originator.	The NULL bytes in name will be overwritten Some format strings within name	- NOTE name = XXX	42
note3	descsz and desc The first descsz bytes in desc hold the note descriptor. ELF places no constraints on a descriptor's contents. If no descriptor is present, descsz contains 0. Padding is present, if necessary, to ensure 4-byte alignment for the next note entry. Such padding is not included in descsz.	Fuzz the first descsz bytes in desc descsz with a random value	- NOTE desc = XXX descsz = XXX	42
note4	type This word gives the interpretation of the descriptor. Each originator controls its own types; multiple interpretations of a single type value may exist. Thus, a program must recognize both the name and the type to "understand" a descriptor. Types currently must be non-negative. ELF does not define what descriptors mean.	Put in type negative values (>= 0x80000000)	- NOTE type >= 0x80000000	42
dyn1	DT_NEEDED This element holds the string table offset of a null-terminated string, giving the name of a needed library. The offset is an index into the table recorded in the DT_STRTAB entry. DT_SONAME This element holds the string table offset of a null-terminated string, giving the name of the shared object. The offset is an index into the table recorded in the DT_STRTAB entry. DT_RPATH This element holds the string table offset of a null-terminated search library search path string, discussed in "Shared Object Dependencies". The offset is an index into the table recorded in the DT_STRTAB entry.	d_val pointing to an out of bounds index Pointing to an invalid name / path.	- DYN d_tag = DT_NEEDED | DT_SONAME | DT_RPATH d_val = XXX	80
dyn2	DT_PLTRELSZ This element holds the total size, in bytes, of the relocation entries associated with the procedure linkage table. If an entry of type DT_JMPREL is present, a DT_PLTRELSZ must accompany it. DT_RELSZ This element holds the total size, in bytes, of the DT_REL relocation table. DT_RELASZ This element holds the total size, in bytes, of the DT_RELA relocation table. DT_STRSZ This element holds the size, in bytes, of the string table.	Less bytes in buffer size to trigger memory corruption vulnerabilities High values to allocate big memory chunks Zero	- DYN d_tag = DT_PLTRELSZ | DT_RELSZ | DT_RELASZ | DT_STRSZ d_val = XXX d_val = 0	80
dyn3	DT_RELAENT This element holds the size, in bytes, of the DT_RELA relocation entry. DT_RELENT This element holds the size, in bytes, of the DT_REL relocation entry. DT_SYMENT This element holds the size, in bytes, of a symbol table entry.	Combination of low and high values Zero	- DYN d_tag = DT_RELAENT | DT_RELENT | DT_SYMENT d_val = XXX d_val = 0	80
dyn4	DT_PLTGOT This element holds an address associated with the procedure linkage table and/or the global offset table.	Pointer to an invalid address and to some point in kernel-land	- DYN d_tag = DT_PLTGOT d_ptr = XXX	80
dyn5	DT_HASH This element holds the address of the symbol hash table, described in "Hash Table". This hash table refers to the symbol table referenced by the DT_SYMTAB element. DT_SYMTAB This element holds the address of the symbol table, described in Chapter 1, with Elf32_Sym entries for the 32-bit class of files.	DT_HASH entry with valid values but DT_SYMTAB with an invalid address or to some point in kernel-land, and vice versa.	- DYN d_tag = DT_HASH | DT_GNU_HASH d_ptr = d_ptr d_ptr = XXX d_tag = DT_SYMTAB d_ptr = d_ptr d_ptr = XXX	80
dyn6	DT_INIT This element holds the address of the initialization function, discussed in "Initialization and Termination Functions" below. DT_FINI This element holds the address of the termination function, discussed in "Initialization and Termination Functions" below.	Make the address to point to some point in kernel-land Make the address to point to the executable's entrypoint Endless loop through DT_INIT to DT_FINI and DT_FINI to DT_INIT	- DYN d_tag = (DT_INIT | DT_FINI) d_ptr = XXX d_ptr = HDR.e_entry d_tag = DT_INIT d_ptr = DT_FINI.d_ptr d_tag = DT_FINI d_ptr = DT_INIT.d_ptr	80
dyn7	DT_SYMBOLIC This element's presence in a shared object library alters the dynamic linker's symbol resolution algorithm for references within the library. Instead of starting a symbol search with the executable file, the dynamic linker starts from the shared object itself. If the shared object fails to supply the referenced symbol, the dynamic linker then searches the executable file and other shared objects as usual. DT_TEXTREL This member's absence signifies that no relocation entry should cause a modification to a non-writable segment, as specified by the segment permissions in the program header table. If this member is present, one or more relocation entries might request modifications to a non-writable segment, and the dynamic linker can prepare accordingly.	Replace the DT_DEBUG for thes ones	- DYN d_tag = DT_DEBUG d_tag = DT_SYMBOLIC | DT_TEXTREL	81
dyn8	DT_PLTREL This member specifies the type of relocation entry to which the procedure linkage table refers. The d_val member holds DT_REL or DT_RELA, as appropriate. All relocations in a procedure linkage table must use the same relocation.	Set to unknown types	- DYN d_tag = DT_PLTREL d_val != (DT_REL | DT_RELA)	81
dyn9	DT_BIND_NOW If present in a shared object or executable, this entry instructs the dynamic linker to process all relocations for the object containing this entry before transferring control to the program. The presence of this entry takes precedence over a directive to use lazy binding for this object when specified through the environment or via dlopen.	Replace the DT_DEBUG for this one	- DYN d_tag = DT_BIND_NOW	81
dyn10	If a shared object name has one or more slash (/) characters anywhere in the name, such as /usr/lib/lib2 above or directory/file, the dynamic linker uses that string directly as the path name. If the name has no slashes, such as lib1 above, three facilities specify shared object path searching, with the following precedence: • First, the dynamic array tag DT_RPATH may give a string that holds a list of directories, separated by colons (:). For example, the string /home/dir/lib:/home/dir2/lib:	Pointing to an fuzzed path	- DYN d_tag = DT_RPATH | DT_RUNPATH d_val = XXX	82
dyn11	The relocation type specifies which bits to change and how to calculate their values. The Intel architecture uses only Elf32_Rel relocation entries, the field to be relocated holds the addend.	Change the rellocation types to DT_RELA for 32-bit files and vice versa	- DYN d_tag = DT_PLTREL d_val = DT_REL | DT_RELA	93

dyn12	DT_PLTGOT On the Intel architecture, this entry's d_ptr member gives the address of the first entry in the global offset table. As mentioned below, the first three global offset table entries are reserved, and two are used to hold procedure linkage table information.
	
	The table's entry zero is reserved to hold the address of the dynamic structure, referenced with the symbol _DYNAMIC. This allows a program, such as the dynamic linker, to find its own dynamic structure without having yet processed its relocation entries. This is especially important for the dynamic linker, because it must initialize itself without relying on other programs to relocate its memory image.
		Patch the three addresses in GOT to point to invalid addresses or key addresses in kernel-land
		
		$ dissector -s hydra | grep -i _DYNAMIC
		[191] 0x080a1f08 ... _DYNAMIC
		$ dissector -a hydra | grep -i got
		[22] .got 0x080a1ff0 0x0058ff0
		[23] .got.plt 0x080a1ff4 0x0058ff4
		[13] DT_PLTGOT 0x080a1ff4
		$ objdump -s -j .got.plt hydra | head
		Contents of section .got.plt:
		80a1ff4 081f0a08 00000000 00000000
		
		The first address is _DYNAMIC. Not always modify this one.
			- DYN
			d_tag = DT_PLTGOT
			*(d_ptr++) = XXX
			*(d_ptr++) = XXX
			*(d_ptr) = XXX
				99

dyn13	DT_JMPREL If present, this entries d_ptr member holds the address of relocation entries associated solely with the procedure linkage table. Separating these relocation entries lets the dynamic linker ignore them during process initialization, if lazy binding is enabled. If this entry is present, the related entries of types DT_PLTRELSZ and DT_PLTREL must also be present.	Leave the DT_JMPREL entry and delete DT_PLTRELSZ	- DYN d_tag = DT_PLTRELSZ d_tag != DT_PLTRELSZ	81
dyn14	DT_RELA This element holds the address of a relocation table, described in Chapter 1. Entries in the table have explicit addends, such as Elf32_Rela for the 32-bit file class. If this element is present, the dynamic structure must also have DT_RELASZ and DT_RELAENT elements. DT_REL This element is similar to DT_RELA, except its table has implicit addends, such as Elf32_Rel for the 32-bit file class. If this element is present, the dynamic structure must also have DT_RELSZ and DT_RELENT elements.	Leave the DT_RELA entry and delete DT_RELASZ and/or DT_RELAENT Leave the DT_REL entry and delete DT_RELSZ and DT_RELENT	- DYN d_tag = DT_RELASZ | DT_RELAENT d_tag != DT_RELASZ d_tag != DT_RELAENT d_tag = DT_RELSZ | DT_RELENT d_tag != DT_RELSZ d_tag != DT_RELENT	80
dyn15	DT_NULL An entry with a DT_NULL tag marks the end of the _DYNAMIC array.	Delete the NULL entry	- DYN d_tag != DT_NULL	80
sym1	Symbol Table An object file's symbol table holds information needed to locate and relocate a program's symbolic definitions and references. A symbol table index is a subscript into this array. Index 0 both designates the first entry in the table and serves as the undefined symbol index. Name Value STN_UNDEF 0	First entry different of STN_UNDEF with fuzzed values	- SYM ST[0].st_name != STN_UNDEF ST[0].* = XXX	32
sym2	st_name This member holds an index into the object file's symbol string table, which holds the character representations of the symbol names.	A st_name out of bounds	- SYM st_name = XXX	32
sym3	st_value This member gives the value of the associated symbol. Depending on the context, this may be an absolute value, an address, and so on; details appear below.	Invalid address Some point in kernel-land High values	- SYM st_value = XXX	32
sym4	st_size Many symbols have associated sizes. For example, a data object's size is the number of bytes contained in the object. This member holds 0 if the symbol has no size or an unknown size.	Combination of low and high values	- SYM st_size = XXX	32
sym5	st_shndx Every symbol table entry is "defined" in relation to some section; this member holds the relevant section header table index. As Figure 1-7 and the related text describe, some section indexes indicate special meanings.	Index to zero and out of bounds Set to a random but valid index within the SHT.	- SYM st_shndx = 0 - HDR->e_shnum st_shndx = XXX orcSYM->st_size = getElf_Word();	33
sym6	STT_SECTION The symbol is associated with a section. Symbol table entries of this type exist primarily for relocation and normally have STB_LOCAL binding.	Change the STB_LOCAL binding type.	- SYM ELF_ST_TYPE(st_info) = STT_SECTION ELF_ST_BIND(st_info) != STB_LOCAL	34
sym7	STT_FILE A file symbol has STB_LOCAL binding, its section index is SHN_ABS, and it precedes the other STB_LOCAL symbols for the file, if it is present.	Change the STB_LOCAL binding type and st_shndx != SHN_ABS	- SYM ELF_ST_TYPE(st_info) = STT_FILE ELF_ST_BIND(st_info) != STB_LOCAL st_shndx != SHN_ABS	34
sym8	In relocatable files, st_value holds alignment constraints for a symbol whose section index is SHN_COMMON.	For those symbols whose st_shndx = SHN_COMMON, fuzz st_value with inconsistent alignment values	- SYM e_type = ET_REL st_shndx = SHN_COMMON st_value != 1,2,4,8,16,32,64,128,256,512…	35
sym9	In relocatable files, st_value holds a section offset for a defined symbol. That is, st_value is an offset from the beginning of the section that st_shndx identifies.	For those symbols whose st_shndx != SHN_COMMON, fuzz st_value with values out of bounds	- SYM e_type = ET_REL st_shndx != SHN_COMMON st_value = XXX	35
sym10	In executable and shared object files, st_value holds a virtual address.	Invalid address Some point in kernel-land	- SYM e_type = ET_EXEC | ET_DYN st_value = XXX	35
sym11	If an executable file contains a reference to a function defined in one of its associated shared objects, the symbol table section for that file will contain an entry for that symbol. The st_shndx member of that symbol table entry contains SHN_UNDEF. This signals to the dynamic linker that the symbol definition for that function is not contained in the executable file itself.	Low chances to set st_shndx to a value different than SHN_UNDEF	- SYM e_type = ET_EXEC | ET_DYN st_shndx != SHN_UNDEF	91
sym12	If that symbol has been allocated a procedure linkage table entry in the executable file, and the st_value member for that symbol table entry is non-zero, the value will contain the virtual address of the first instruction of that procedure linkage table entry. Otherwise, the st_value member contains zero	For those symbols with st_shndx = SHN_UNDEF, set the st_value to an invalid pointer or at some point to kernel-land	- SYM st_shndx = SHN_UNDEF st_value != 0 st_value = XXX	91
sym13	st_info This member specifies the symbol's type and binding attributes. A list of the values and meanings appears below. The following code shows how to manipulate the values. #define ELF32_ST_BIND(i) ((i)>>4) #define ELF32_ST_TYPE(i) ((i)&0xf) #define ELF32_ST_INFO(b,t) (((b)<<4)+((t)&0xf))	Combination of low and high values Specific combinations to escape from common symbol types and binding types	- SYM st_info = XXX	32
rel1	r_offset This member gives the location at which to apply the relocation action. For a relocatable file, the value is the byte offset from the beginning of the section to the storage unit affected by the relocation. For an executable file or a shared object, the value is the virtual address of the storage unit affected by the relocation.	In relocatable files r_offset out of bounds In exec and shared objects, r_offset will hold invalid addresses and addresses in kernel-land	- REL e_type = ET_REL r_offset = XXX e_type = ET_EXEC | ET_DYN r_offset = XXX	36
rel2	r_info This member gives both the symbol table index with respect to which the relocation must be made, and the type of relocation to apply. Relocation types are processor-specific; descriptions of their behavior appear in the processor supplement. When the text in the processor supplement refers to a relocation entry's relocation type or symbol table index, it means the result of applying ELF32_R_TYPE or ELF32_R_SYM, respectively, to the entry's r_info member. #define ELF32_R_SYM(i) ((i)>>8) #define ELF32_R_TYPE(i) ((unsigned char)(i)) #define ELF32_R_INFO(s,t) (((s)<<8)+(unsigned char)(t))	Combination of low and high values Specific combinations to escape from the macros Make that ELF32_R_SYM() returns an invalid section index (its related symbol table)	- REL r_info = XXX ELF_R_SYM(r_info) > e_shnum	36
rel3	r_addend This member specifies a constant addend used to compute the value to be stored into the relocatable field.	r_addend is Sword (Signed Word). Combination of high and low values. Negative values	- REL r_addend >= 0x80000000 r_addend = XXX	36
env1	If the process environment contains a variable named LD_BIND_NOW with a non-null value, the dynamic linker processes all relocation before transferring control to the program. For example, all the following environment entries would specify this behavior. • LD_BIND_NOW=1 • LD_BIND_NOW=on • LD_BIND_NOW=off Otherwise, LD_BIND_NOW either does not occur in the environment or has a null value. The dynamic linker is permitted to evaluate procedure linkage table entries lazily, thus avoiding symbol resolution and relocation overhead for functions that are not called.	Fuzz the environment variable before executing the malformed ELFs or testing the programs	- ENVIRON export LD_BIND_NOW = XXX	77

env2	Second, a variable called LD_LIBRARY_PATH in the process environment [see exec(BA_OS)] may hold a list of directories as above, optionally followed by a semicolon (;) and another directory list. The following values would be equivalent to the previous example: • LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib: • LD_LIBRARY_PATH=/home/dir/lib;/home/dir2/lib: • LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib:;	Fuzz the environment variable before executing the malformed ELFs or testing the programs	- ENVIRON export LD_LIBRARY_PATH = XXX	82
hash1	The bucket array contains nbucket entries, and the chain array contains nchain entries; indexes start at 0.	nbucket and nchain with high and low values	- HASH nbucket = XXX nchain = XXX	84
hash2	Both bucket and chain hold symbol table indexes. Chain table entries parallel the symbol table. The number of symbol table entries should equal nchain;	Take the original indexes and set them to out of bounds values	- HASH nbucket = nbucket nchain = nchain bucket[1..nbucket-1] = XXX chain[1..nchain-1] = XXX	84

ELF-64 File Format Violations

XXX: Value to be fuzzed with semi-valid semantics

Rule	Specification	Violation description	ELF metadata	Page
hdr16	e_ident[EI_ABIVERSION] identifies the version of the ABI for which the object is prepared. This field is used to distinguish among incompatible versions of an ABI. The interpretation of this version number is dependent on the ABI identified by the EI_OSABI field. For applications conforming to the System V ABI, third edition, this field should contain 0.	Set to uncommon values	- HDR e_ident[EI_ABIVERSION] = XXX	4
hdr17	Table 5. Operating System and ABI Identifiers, e_ident[EI_OSABI] Name Value Meaning ELFOSABI_SYSV 0 System V ABI ELFOSABI_HPUX 1 HP-UX operating system ELFOSABI_STANDALONE 255 Standalone (embedded) application	Set to uncommon values	- HDR e_ident[EI_OSABI] = XXX	5
hdr18	Table 6. Object File Types, e_type Name Value Meaning ET_LOOS 0xFE00 Environment-specific use ET_HIOS 0xFEFF	Set to these types	- HDR e_type = ET_LOOS + 1 e_type = ET_HIOS	5
pht14	Table 16. Segment Types, p_type (Continued) Name Value Meaning PT_LOOS 0x60000000 Environment-specific use PT_HIOS 0x6FFFFFFF	Set some segments to these types	- PHT p_type = (PT_LOOS | PT_HIOS)	13
pht15	Table 17. Segment Attributes, p_flags Name Value Meaning PF_MASKOS 0x00FF0000 These flag bits are reserved for environment-specific use	Enable this flag in some segments	- PHT p_flags |= PF_MASKOS	13
sht31	Table 8. Section Types, sh_type Name Value Meaning SHT_LOOS 0x60000000 Environment-specific use SHT_HIOS 0x6FFFFFFF	Set some sections to these types	- SHT sh_type = (SHT_LOOS | SHT_HIOS)	7
sht32	Table 9. Section Attributes, sh_flags Name Value Meaning SHF_MASKOS 0x0F000000 Environment-specific use	Enable this flag in some sections	- SHT sh_flags |= SHF_MASKOS	8
dyn16	Table 18. Dynamic Table Entries Name Value d_un Meaning DT_INIT_ARRAY 25 d_ptr Pointer to an array of pointers to initialization functions. DT_FINI_ARRAY 26 d_ptr Pointer to an array of pointers to termination functions.	Pointer to an invalid address and to some point in kernel-land The first entry normally is a helper function in GLIBC, so, fuzz: DT_INIT_ARRAY[1] and/or DT_FINI_ARRAY[1] if DT_INIT_ARRAYSZ > sizeof(void *) with invalid pointers	- DYN d_tag = DT_INIT_ARRAY | DT_FINI_ARRAY *(++d_ptr) = XXX	15
dyn17	DT_INIT_ARRAYSZ 27 d_val Size, in bytes, of the array of initialization functions. DT_FINI_ARRAYSZ 28 d_val Size, in bytes, of the array of termination functions.	Combination of low and high values d_val % 4 shouldn't be zero	- DYN d_tag = DT_INIT_ARRAYSZ | DT_FINI_ARRAYSZ d_val += (4 | 8) d_val % 4 != 0	15
sym14	Table 14. Symbol Bindings Name Value Meaning STB_LOOS 10 Environment-specific use STB_HIOS 12 Table 15. Symbol Types (Continued) Name Value Meaning STT_LOOS 10 Environment-specific use STT_HIOS 12	Set some entries to these types	- SYM ELF_ST_BIND(st_info) = (STB_LOOS | STB_HIOS) ELF_ST_TYPE(st_info) = (STT_LOOS | STT_HIOS)	10

Ideas & Considerations

XXX: Value to be fuzzed with semi-valid semantics

Rule	Idea / Consideration	Description	ELF metadata
hdr19	OPENBSD kern/exec_elf.c:129:#define ELF_MAX_VALID_PHDR 32 kern/exec_elf.c:205: if (ehdr->e_phnum > ELF_MAX_VALID_PHDR)	Modify the PHT to have 32 program headers	- HDR e_phnum = 32
pht16	Modify some security specific features seen in the PHT such as executable stack, PAX flags or RELRO.	Fuzz the metadate related to these features	- PHT p_type = PT_GNU_STACK | PT_PAX_FLAGS | PT_GNU_RELRO p_* = XXX
pht17	Static binaries add PT_TLS	From the PHT fuzz its values	- PHT p_type = PT_TLS p_* = XXX
pht18	GNU extension PT_GNU_EH_FRAME sorted table of unwind information. GCC uses this table to find the appropriate handler for an exception.	Fuzz p_filesz bytes from p_offset with invalid memory addresses / kernel addresses	- PHT p_type = PT_GNU_EH_FRAME p_* = XXX
pht22	Static ELFs don't have PT_DYNAMIC	Add a PT_DYNAMIC entry with invalid information Add a PT_DYNAMIC entry pointing to anywhere in the file	- PHT Algorithm to find the PT_DYNAMIC entry and fuzz its values. If not found, look for a PT_NULL entry and set its p_type to PT_DYNAMIC.
sht33	A section header's sh_type member specifies the section's semantics.	Also include the SHT_GNU_* types	- SHT sh_type = SHT_GNU_*
sht34	Seen that all the SHF_WRITE and SHF_EXECINSTR need the SHF_ALLOC	Clear the SHF_ALLOC of those sections	- SHT sh_flags &= ~ SHF_ALLOC
sht35	.init_array and .fini_array are arrays of function pointers to functions	Overwrite the pointers with invalid addresses or pointers in kernel-land	- SHT sh_type = (SHT_INIT_ARRAY | SHT_FINI_ARRAY) (sh_offset) = jmp entrypoint (HDR.e_entry) (sh_offset) = jmp _init (.init.sh_addr) (sh_offset) = jmp _fini (.fini.sh_addr) (sh_offset) = XXX
sht36	Trusting in sizes such as: for(I = 0; I < shdr->sh_size / shdr->sh_entsize; I++, rel++){	To make loops run one more time: Add +1 to sh_size Substract -1 to sh_entsize	- SHT sh_size += 1 sh_entsize -= 1
sht37	Static binaries add .tdata and .tbss sections	From the SHT delete the SHF_TLS flag from all the sections	- SHT sh_name ".tdata" | ".tbss" sh_flags &= ~ SHF_TLS
dyn18	Set d_tag to high / random / negative values (> 0x7fffffff)	Low chances to fuzz d_tag. Random or negative values Set to: DT_LOOS 0x6000000d DT_HIOS 0x6ffff000 DT_LOPROC 0x70000000 DT_HIPROC 0x7fffffff	- DYN d_tag = XXX d_tag > 0x7fffffff d_tag = DT_LOOS | DT_HIOS | DT_LOPROC | DT_HIPROC
sym15	ELF_ST_VISIBILITY() uses st_other which is mentioned in the original specification as unused: st_other This member currently holds 0 and has no defined meaning.	Fuzz its content randomly	- SYM st_other = XXX
env3	The testing script should allow to fuzz the LD_PRELOAD environment variable	Fuzz the environment variable before executing the malformed ELFs or testing the programs	- ENVIRON export LD_PRELOAD = XXX

ELF Parsing Patterns

Software	Pattern
Memory Mapping (Filesystem to Memory)	

Linux Kernel 3.13.6	arch/sh/boot/Makefile:15:CONFIG_PAGE_OFFSET ?= 0x80000000
	arch/x86/include/asm/page_types.h:30:#define PAGE_OFFSET ((unsigned long)__PAGE_OFFSET)
	arch/x86/include/asm/page_32_types.h:16:#define __PAGE_OFFSET _AC(CONFIG_PAGE_OFFSET, UL)
	arch/x86/include/asm/page_64_types.h:33:#define __PAGE_OFFSET _AC(0xffff880000000000, UL)
	
	fs/binfmt_elf.c:66:#define ELF_MIN_ALIGN PAGE_SIZE
	fs/binfmt_elf.c:73:#define ELF_PAGESTART(_v) ((_v) & ~(unsigned long)(ELF_MIN_ALIGN-1))
	fs/binfmt_elf.c:74:#define ELF_PAGEOFFSET(_v) ((_v) & (ELF_MIN_ALIGN-1))
	fs/binfmt_elf.c:75:#define ELF_PAGEALIGN(_v) (((_v) + ELF_MIN_ALIGN - 1) & ~(ELF_MIN_ALIGN - 1))
	
	#define BAD_ADDR(x) ((unsigned long)(x) >= TASK_SIZE)
	
	static unsigned long total_mapping_size(struct elf_phdr *cmds, int nr)
	{
	...
	return cmds[last_idx].p_vaddr + cmds[last_idx].p_memsz -
	ELF_PAGESTART(cmds[first_idx].p_vaddr);
	
	total_size = total_mapping_size(elf_phdata, interp_elf_ex->e_phnum);
	
	static unsigned long elf_map(struct file *filep, unsigned long addr,
	struct elf_phdr *eppnt, int prot, int type,
	unsigned long total_size)
	{
	unsigned long map_addr;
	unsigned long size = eppnt->p_filesz + ELF_PAGEOFFSET(eppnt->p_vaddr);
	unsigned long off = eppnt->p_offset - ELF_PAGEOFFSET(eppnt->p_vaddr);
	addr = ELF_PAGESTART(addr);
	size = ELF_PAGEALIGN(size);
	
	if (total_size) {

ClamAV 0.98.1	/* Read rest of 64-bit header */
	if(fmap_readn(map, file_hdr->hdr32.pad, sizeof(struct elf_file_hdr32), ELF_HDR_SIZEDIFF)
	!= ELF_HDR_SIZEDIFF) {
	
	if(phnum) {
	program_hdr = (struct elf_program_hdr32 *) cli_calloc(phnum, sizeof(struct elf_program_hdr32));
	if(!program_hdr) {
	cli_errmsg("ELF: Can't allocate memory for program headers\n");
	
	for(i = 0; i < phnum; i++) {
	err = 0;
	if(fmap_readn(map, &program_hdr[i], phoff, sizeof(struct elf_program_hdr32)) != sizeof(struct elf_program_hdr32))
	err = 1;

Malelficus 1.0.0	if (-1 == fstat(bin->fd, &st_info)) {
	error = errno;
	MALELF_DEBUG_ERROR("Failed to stat file '%s'.",
	bin->fname);
	return error;
	}
	
	if (0 == st_info.st_size && !is_creat) {
	return MALELF_EEMPTY_FILE;
	}
	
	bin->size = st_info.st_size;
	
	bin->mem = mmap(0,
	bin->size,
	PROT_READ|PROT_WRITE,
	MAP_PRIVATE,
	bin->fd,
	0);

Section Header Table's String Table Location	

Binutils 2.24 readelf	else if (elf_header.e_shstrndx != SHN_UNDEF
	&& elf_header.e_shstrndx >= elf_header.e_shnum)
	printf (_(" <corrupt: out of range>"));
	
	/* Read in the string table, so that we have names to display. */
	if (elf_header.e_shstrndx != SHN_UNDEF
	&& elf_header.e_shstrndx < elf_header.e_shnum)
	{
	section = section_headers + elf_header.e_shstrndx;
	
	if (section->sh_size != 0)
	{
	string_table = (char *) get_data (NULL, file, section->sh_offset,
	1, section->sh_size,
	_("string table"));
	
	string_table_length = string_table != NULL ? section->sh_size : 0;

Validation of Class, Machine and Endianness	
Linux Kernel 3.13.6	/* * This is used to ensure we don't load something for the wrong architecture. */ #define elf_check_arch_ia32(x) \ (((x)->e_machine == EM_386) || ((x)->e_machine == EM_486)) if (!elf_check_arch(interp_elf_ex)) goto out;

Binutils 2.24 readelf	if ((elf_header.e_machine == EM_ALPHA
	|| elf_header.e_machine == EM_S390
	|| elf_header.e_machine == EM_S390_OLD)
	&& elf_header.e_ident[EI_CLASS] == ELFCLASS64)
	hash_ent_size = 8;

ClamAV 0.98.1

```
        default:
            cli_dbgmsg("ELF: Unknown ELF class (%u)\n", file_hdr->hdr64.e_ident[4]);
            return CL_EFORMAT;

    /* Need to know to endian convert */
    if(file_hdr->hdr64.e_ident[5] == 1) {

        /* Now endian convert, if needed */
        if(conv) {
            file_hdr->hdr64.e_entry = EC64(file_hdr->hdr64.e_entry, conv);
            file_hdr->hdr64.e_phoff = EC64(file_hdr->hdr64.e_phoff, conv);
            file_hdr->hdr64.e_shoff = EC64(file_hdr->hdr64.e_shoff, conv);
```

Section Header Table

Binutils 2.24 readelf

```
  for (i = 0, section = section_headers;
       i < elf_header.e_shnum;
       i++, section++)
    {


find_section (const char * name)
{
  unsigned int i;

  for (i = 0; i < elf_header.e_shnum; i++)
    if (streq (SECTION_NAME (section_headers + i), name))
      return section_headers + i

find_section_by_address (bfd_vma addr)
{
  unsigned int i;

  for (i = 0; i < elf_header.e_shnum; i++)
    {
      Elf_Internal_Shdr *sec = section_headers + i;

  if (elf_header.e_shnum == 0)
    {
      /* PR binutils/12467.  */
      if (elf_header.e_shoff != 0)
        warn (_("possibly corrupt ELF file header - it has a non-zero"
                " section header offset, but no section headers\n"));
```

ClamAV 0.98.1

```
    shnum = file_hdr->e_shnum;
    cli_dbgmsg("ELF: Number of sections: %d\n", shnum);
    if(ctx && (shnum > 2048)) {
        cli_dbgmsg("ELF: Number of sections > 2048, skipping\n");

    shentsize = file_hdr->e_shentsize;
    /* Sanity check */
    if(shentsize != sizeof(struct elf_section_hdr32)) {
        cli_dbgmsg("ELF: shentsize != sizeof(struct elf_section_hdr32)\n");
        if(ctx && DETECT_BROKEN) {
            cli_append_virus(ctx, "Heuristics.Broken.Executable");
```

Malelficus 1.0.0

```
        for (i = 0; i < shnum; i++) {
                Elf32_Shdr *s = &sections[i];
                malelf_table_add_int_value(&table, i);
                malelf_table_add_hex_value(&table, s->sh_addr);
                malelf_table_add_int_value(&table, s->sh_offset);
                MALELF_CHECK(malelf_shdr_get_mstype, &shdr, &ms_type, i);
                malelf_table_add_str_value(&table, ms_type.name);
                if (s->sh_type != SHT_NULL && shstrndx != 0x00) {

        for (i = host_ehdr->e_shnum; i-- > 0; host_shdr++) {
                if (host_shdr->sh_offset >= parasite_end_offset) {
                        host_shdr->sh_offset += MALELF_PAGE_SIZE;
```

Program Header Table

Linux Kernel 3.13.6

```
        if (interp_elf_ex->e_phnum < 1 ||
                interp_elf_ex->e_phnum > 65536U / sizeof(struct elf_phdr))
                goto out;

        eppnt = elf_phdata;
        for (i = 0; i < interp_elf_ex->e_phnum; i++, eppnt++) {

        for (i = 0; i < loc->elf_ex.e_phnum; i++) {
                if (elf_ppnt->p_type == PT_INTERP) {

        elf_ppnt = elf_phdata;
        for (i = 0; i < loc->elf_ex.e_phnum; i++, elf_ppnt++)
                if (elf_ppnt->p_type == PT_GNU_STACK) {
                        if (elf_ppnt->p_flags & PF_X)
                                executable_stack = EXSTACK_ENABLE_X;
                        else
                                executable_stack = EXSTACK_DISABLE_X;
                        break;
                }
```

Binutils 2.24 readelf

```
  for (i = 0, segment = program_headers;
       i < elf_header.e_phnum;
       i++, segment++)
    {

      printf (_("\n Section to Segment mapping:\n"));
      printf (_("  Segment Sections...\n"));

      for (i = 0; i < elf_header.e_phnum; i++)
        {
```

ClamAV 0.98.1	phnum = file_hdr->e_phnum;
	cli_dbgmsg("ELF: Number of program headers: %d\n", phnum);
	if(phnum > 128) {
	cli_dbgmsg("ELF: Suspicious number of program headers\n");
	
	if(phnum && entry) {
	phentsize = file_hdr->e_phentsize;
	/* Sanity check */
	if(phentsize != sizeof(struct elf_program_hdr32)) {
	cli_dbgmsg("ELF: phentsize != sizeof(struct elf_program_hdr32)\n");
	if(ctx && DETECT_BROKEN) {
	cli_append_virus(ctx, "Heuristics.Broken.Executable");
	
	for(i = 0; i < phnum; i++) {
	if(EC32(ph[i].p_vaddr, conv) <= vaddr && EC32(ph[i].p_vaddr, conv) + EC32(ph[i].p_memsz, conv) > vaddr) {
	found = 1;

Malelficus 1.0.0	for (phdr = host_phdr, i = host_ehdr->e_phnum;
	i-- > 0;
	phdr++) {

Symbol(s) Table(s)	

Binutils 2.24 readelf	if (ELF_ST_TYPE (psym->st_info) == STT_SECTION)
	{
	if (psym->st_shndx < elf_header.e_shnum)
	
	if (section->sh_entsize == 0)
	{
	printf (_("\nSymbol table '%s' has a sh_entsize of zero!\n"),
	SECTION_NAME (section));
	continue;
	}
	
	printf (_("\nSymbol table '%s' contains %lu entries:\n"),
	SECTION_NAME (section),
	(unsigned long) (section->sh_size / section->sh_entsize));
	
	symtab = GET_ELF_SYMBOLS (file, section, & num_syms);
	...
	for (si = 0, psym = symtab; si < num_syms; si++, psym++)
	{

Dynamic Information	

Binutils 2.24 readelf	for (ext = edyn, dynamic_nent = 0;
	(char *) ext < (char *) edyn + dynamic_size;
	ext++)
	{
	dynamic_nent++;
	if (BYTE_GET (ext->d_tag) == DT_NULL)
	break;
	}
	
	printf (_("\nDynamic section at offset 0x%lx contains %u entries:\n"),
	dynamic_addr, dynamic_nent);
	
	for (entry = dynamic_section;
	entry < dynamic_section + dynamic_nent;
	entry++)
	{

Relocations	

Binutils 2.24 readelf	rel_size = section->sh_size;
	
	if (rel_size)
	...
	printf (_(" at offset 0x%lx contains %lu entries:\n"),
	rel_offset, (unsigned long) (rel_size / section->sh_entsize));
	
	symsec = section_headers + section->sh_link;
	if (symsec->sh_type != SHT_SYMTAB
	&& symsec->sh_type != SHT_DYNSYM)

Misc	
Malelficus 1.0.0	end_of_text = target_phdr->p_offset + target_phdr->p_filesz; /* last_chunk = input->size - end_of_text;*/ error = malelf_binary_copy_data(output, input, end_of_text, input->size);