Linux kernel 5.7.7

Unexported kallsyms functions

Filip Pynckels

July 14, 2020

Table of contents

Introduction

Some time ago, I was installing the new Linux kernel when the CHIPSEC module refused to compile. I got the same result, at the same time, with the TYTON module. So, what was going on?

When looking into the problem, it seemed that the sources of both the routines were using kallsyms kernel functions, and that the kernel 5.7.7 build routines don't export kallsyms kernel functions any longer (for use by custom kernel modules).

This paper has as goal to show a number of alternatives to circumvent this problem.

1 Analysis of the problem

It seems that the unexporting of the kallsyms routines has been discussed at a certain moment, but without getting much attention of the Linux community. More specific, the following can be found on https://github.com/torvalds/linux/commit/0bd476e6c67190b5eb7b6e105c8db8ff61103281 :

```
kallsyms: unexport kallsyms_lookup_name() and kallsyms_on_each_symbol()
kallsyms_lookup_name() and kallsyms_on_each_symbol() are exported to
modules despite having no in-tree users and being wide open to abuse by
out-of-tree modules that can use them as a method to invoke arbitrary
non-exported kernel functions.

Unexport kallsyms_lookup_name() and kallsyms_on_each_symbol().

Signed-off-by: Will Deacon <will@kernel.org>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
Reviewed-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
Reviewed-by: Christoph Hellwig <hch@lst.de>
Reviewed-by: Masami Hiramatsu <mhiramat@kernel.org>
Reviewed-by: Quentin Perret <qperret@google.com>
Acked-by: Alexei Starovoitov <ast@kernel.org>
Link: http://lkml.kernel.org/r/20200221114404.14641-4-will@kernel.org
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
```

So, the reason seems to be:

- There are no in-tree users of these routines.
- The routines are wide open to abuse by out-of-tree modules

But what is really going on here? Is there an advanced insight, or rather a hidden goal. In each case, module developers are not happy with this decision, and even kernel developers don't estimate it usefull as can be read at
https://lwn.net/ml/linux-kernel/20200221232746.6eb84111a0d385bed71613ff@kernel.org/ :

```
Hi Will,

On Fri, 21 Feb 2020 11:44:01 +0000
Will Deacon <will@kernel.org> wrote:

What kind of issue would you like to fix with this?
There are many ways to find (estimate) symbol address, especially, if
the programmer already has the symbol map, it is *very* easy to find
the target symbol address even from one exported symbol (the distance
of 2 symbols doesn't change.) If not, they can use kprobes to find
their required symbol address. If they have a time, they can use
snprintf("%pF") to search symbol.

So, for me, this series just make it hard for casual developers (but
maybe they will find the answer on any technical Q&A site soon).

Hmm, are there other good way to detect such bad-manner out-of-tree
module and reject them? What about decoding them and monitor their
all call instructions?

Thank you,

--
Masami Hiramatsu <mhiramat@kernel.org>
```

Some more searching reveals:

```
In response, Deacon posted an interesting message about what is driving this
particular change. Kernel developers are happy to make changes just to make
life difficult for developers they see as abusing the system, but that is not
quite what is happening here. Instead, it is addressing a support issue at
Google.

...

Restricting vendors to supplying kernel modules greatly limits the kind of
changes they can make; there will be no more Android devices that replace the
CPU scheduler with some vendor's special version, for example. But that only
holds if modules are restricted to the exported-symbol interface; if they
start to reach into arbitrary parts of the kernel, all bets are off.
```

The above seems to be a more believable reason than the “wide open abuse”, althoug, other vendors replacing the CPU scheduler of Android will seem to a wide open abuse to Google...

The entire text from which the above is a small part can be found in *Appendix 1 - Unexporting kallsyms_lookup_name()* of this paper.

To make a long story short, since kernel 5.7.7 all third party modules that use some of the unexported kernel routines see the following error (or a comparable one) when compiling for installation on a Linux machine:

```
ERROR: modpost: "kallsyms_lookup_name" [kallsyms.ko] undefined!
make[2]: *** [scripts/Makefile.modpost:99: __modpost] Error 1
make[1]: *** [Makefile:1645: modules] Error 2
make: *** [Makefile:27: all] Error 2
```

The above error message is an extract of *Appendix 2 - Kallsyms module - Kallsyms lookup name no longer available* that shows a module that installed and ran perfectly under the Linux kernel 5.6.x

From here on, we will try to find a number of alternatives to find the place in memory of the kernel routines page_is_ram() and phys_mem_access_prot(). The first routine is still exported by the kernel, the second is not and has to be found in a different way.

2 Preparing the 'quest' for alternatives

Further down in this paper, we will give a number of alternatives. But to test what the result of the alternatives is, we need to create a small test environment that can compile, install and test a module that contains our alternative code.

The test environment is composed of a Linux kernel 5.7.7, a makefile, a script, a basic kernel module source, and the gcc. The script can be found in *Appendix 5 - Kallsyms testscript and Makefile* as is the makefile

The content of the makefile is the following:

```
1. obj-m += kallsyms.o
2.
3. all:
4.     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5.
6. clean:
7.     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

The source for the a basic kernel module is the following:

```
1. #include <linux/init.h>
2. #include <linux/module.h>
3. #include <linux/kernel.h>
4.
5. MODULE_LICENSE("Dual MIT/GPL");
6. MODULE_AUTHOR("Filip Pynckels");
7. MODULE_DESCRIPTION("Module for kernel 5.7.7");
8. MODULE_VERSION("0.01");
9.
10. static int __init kallsyms_init(void)
11. {
12.     printk(KERN_INFO "kallsyms: initializing...\n");
13.     printk(KERN_INFO "kallsyms: initialized\n");
14.
15.     return 0;
16. }
17.
18. static void __exit kallsyms_exit(void)
19. {
20.     printk(KERN_INFO "kallsyms: exiting...\n");
21.     printk(KERN_INFO "kallsyms: exited\n");
22. }
23.
24. module_init(kallsyms_init);
25. module_exit(kallsyms_exit);
```

The commands to build and install the module during runtime are:

```
1. make
2. sudo insmod kallsyms.ko
```

3 Alternative 1 - A custom build kernel

3.1 Method

The first alternative is to change the sources of the kernel and rebuild the kernel. The changes to the kernel are minimal. More specific, one can use the following commands to find the file(s) to change:

```
1. cd /lib/modules/$(uname -r)/build
2. Grep -r 'the-symbol' *
```

Open the file at place of the function/symbol, and use the EXPORT_SYMBOL() macro or the EXPORT_SYMBOL_GPL() macro to signal the build process to export the function/symbol in question. Please do note that

After this change, save the file, rebuild the Linux kernel and check the System.map file to assess that the function you need is indeed experted:

```
1. cat /lib/modules/$(uname -r)/build/System.map | grep 'the-symbol'
```

Finally, one can use the function __symbol_get() to get the address of an exported kernel function or symbol. See *Appendix 3 - Kallsyms module - Alternative 1 - Custom kernel with __symbol get* for an example. Note that the function page_is_ram() is exported, and hence, that we can get it's address. On the other hand, function phys_mem_access_prod() is not exported, and hence, __symbol_get returns us a 0x000..000 address.

3.2 Advantages and disadvantages

The advantage is clear and simple: it's a clean and correct solution if implemented correctly.

The disadvantage is that it's only a solution for those machines that run a modified kernel. Besides that element, changing a kernel for use on only a couple of machines makes it a "not broadly tested" kernel, and hence a kernel that may be a risc in a production environment.

Further, even if modules can run on the modified kernel, they still will produce errors during the installation on a classic kernel. Since modules are developed to run on as wide a range of machines as possible, requiring a modified kernel prohibits this goal.

Conclusion: modifying the kernel may be a risc in a production environment and prohibits a wider distribution of ones code. Updating a modified kernel is also a more difficult job than updating a default kernel. But, on the other hand, it is the most guaranteed way to have access to otherwise not exported kernel symbols. Unfortunately, each time we need access to an other symbol or function, we need to change the kernel even more, which makes maintenance rather difficult.

4 Alternative 2 - kprobes

4.1 Method

Without going to much in detail, kprobes is a debugging mechanism for the Linux kernel which can also be used for monitoring events inside a production system. You can use it to weed out performance bottlenecks, log specific events, trace problems etc.

To do what it is supposed to do, the kprobes mechanism needs (and in theory has) access to all kernal symbols, including those that are not exported to third party modules. And there we are. There is a second mechanism in the Linux kernel (one that will be difficult to eliminate the first couple of years) that gives us the information we need.

A fully functional example can be found in *Appendix 4 - Kallsyms module - Alternative 2 - kprobes*. And lo and behold, this method returns the address of a function that is exported by the kernel but also the address of a function that is not exported by the kernel. An extract of the result shows:

```
1. ==> Testing kallsyms module...
2. [ 9292.283991] kallsyms: option(3) page_is_ram(000000007cc63fd1) phys_mem_acc
   ess_prot(00000000b05c04cd)
```

4.2 Advantages and disadvantages

Except for the fact that this alternative was not easy to find, there are no real disadvantages. Up until now, all needed symbols were found this way. So, except if some symbols would not be available, this seems a rather clean way to use in third party modules.

5 Alternative 3 - Offset in symbol.map

5.1 Method

This method is risky and not very crisp, to say the least. But a module has to do what a module has to do to find a not exported kernel function/symbol if it is really needed for the functioning of the mentioned module.

The base idea of the below described method is the following: Linux does not load its modules at the same real address time after time. On the contrary, for security reasons, it puts the modules around the place (in upper real memory) and maps that real address to a virtual address that is then used by other kernel modules and eventually user space programs.

This has as a consequence (taking a short turn) that the addresses in the System.map file can and may not be interpreted as memory addresses where a certain symbol is to be found. But, lucky us, the offset between symbols in the System.map file seems to stay the same. So, a solution seems straight forward.

Look in the System.map file to find the address of the symbol we need. Also look above and below this line in the System.map file to find a symbol that is exported. We can calculate the offset between the two symbols. This offset (positive or negative) is probable to stay valid, since we stay close to the line in the System.map file of the symbol we need.

If we start from the principle that the smaller the calculated offset is, the more chance we have that it stays valid with each build of this Linux kernel, we can eventually be lucky and find the address of the symbol we need by getting the address of the nearby symbol with __symbol_get() and by adding the pre-calculated offset to it.

Searching in the System.map file for symbols can be done with the command:

```
1. cat /lib/modules/$(uname -r)/build/System.map | grep 'the-symbol'
```

To find information about 10 symbols before and after the symbol in the System.map file, the following command can be used:

```
1. grep -C 10 'the-symbol' /lib/modules/$(uname -r)/build/System.map
```

After this change, save the file, rebuild the Linux kernel and check the System.map file to assess that the function you need is indeed experted:

5.2 Advantages and disadvantages

It goes without saying that this method is rather a fishing expedition than a reliable method. But it can do the trick if you are out of other options. However, we would strongly advise against this method for production purposes.

6 Conclusion

The first conclusion is that it seems strange that Linux is changed to accommodate a profit organisation, to prevent other profit organisations of plugging into the kernel.

The second conclusion is that there is, indeed, as Mr Deacon authored and Mr. Torvalds committed on April 7th, 2020 a solution that is feasible when one knows how to use it.

We have shown in this paper that there are, in fact, even other alternatives. These are, however, less useable than the kprobes solution, since they limit the reliability, the span of distribution, the ease of maintenance, etc.

The final conclusion is that the second alternative (see *Appendix 4 - Kallsyms module - Alternative 2 - kprobes*) seems the most appropriate. It also seems to be a solution that is sustainable, since monitoring and tuning the kernel will always be an issue, wether it will be with the kprobes mechanism or an other mechanism that is still to come. But whatever mechanism will be used to monitor and tune the kernel, it will need access to as much kernel symbols as possible, and it will have to be accessible from third party kernel modules.

Appendix 1 - Unexporting kallsyms_lookup_name()

By Jonathan Corbet, February 28, 2020, lwn.net:

One of the basic rules of kernel-module development is that modules can only access symbols (functions and data structures) that have been explicitly exported. Even then, many symbols are restricted so that only modules with a GPL-compatible license can access them. It turns out, though, that there is a readily available workaround that makes it easy for a module to access any symbol it wants. That workaround seems likely to be removed soon despite some possible inconvenience for some out-of-tree users; the reason why that is happening turns out to be relatively interesting.

The backdoor in question is kallsyms_lookup_name(), which will return the address associated with any symbol in the kernel's symbol table. Modular code that wants to access a symbol ordinarily denied to it can use kallsyms_lookup_name() to get the address of its target, then dereference it in the usual ways. This function itself is exported with the GPL-only restriction, which theoretically limits its use to free software. But if a proprietary module somewhere were to falsely claim a free license to get at GPL-only symbols, it would not be the first time.

Will Deacon has posted a patch series that removes the export for kallsyms_lookup_name() (and kallsyms_on_each_symbol(), which is also open to abuse). There were some immediate positive responses; few developers are favorably inclined toward module authors trying to get around the export system, after all. There were, however, a couple of concerns expressed.

One of those is that there is, it seems, a class of out-of-tree users of kallsyms_lookup_name() that is generally considered to be legitimate: live-patching systems for the kernel. Irritatingly, kernel bugs often stubbornly refuse to restrict themselves to exported functions, so a live patch must be able to locate (and patch out) any function in the kernel;

kallsyms_lookup_name() is a convenient way to do that. After some discussion Joe Lawrence let it be known that the kpatch system has all of its needed infrastructure in the mainline kernel, and so has no further need for kallsyms_lookup_name(). The Ksplice system, though, evidently still uses it. As Miroslav Benes observed, though: "no one cares about ksplice in upstream now". So it would appear that live patching will not be an obstacle to the merging of this patch.

A different sort of concern was raised by Masami Hiramatsu, who noted that there are a number of other ways to find the address associated with a kernel symbol. User space could place some kprobes to extract that information, or a kernel module could, if time and CPU use is not a concern, use snprintf() with the "%pF" format (which prints the function associated with a given address) to search for the address of interest. He worried that the change would make life harder for casual developers while not really getting in the way of anybody who is determined to abuse the module mechanism.

In response, Deacon posted an interesting message about what is driving this particular change. Kernel developers are happy to make changes just to make life difficult for developers they see as abusing the system, but that is not quite what is happening here. Instead, it is addressing a support issue at Google.

Back in 2018, LWN reported on work being done to bring the Android kernel closer to the mainline. One of the steps in that direction is moving the kernel itself into the Android generic system image (GSI), an Android build that must boot and run on a device for that device to be considered compliant with the Android requirements. Putting the kernel into the GSI means that hardware vendors can no longer modify it; they will be limited to what they can do by adding kernel modules to the GSI.

Restricting vendors to supplying kernel modules greatly limits the kind of changes they can make; there will be no more Android devices that replace the CPU scheduler with some vendor's special version, for example. But that only holds if modules are restricted to the exported-symbol interface; if they start to reach into arbitrary parts of the kernel, all bets are off. Deacon doesn't say so, but it seems clear that some vendors are, at a minimum, thinking about doing exactly that. The business-friendly explanation for removing this capability is: "Monitoring and managing the ABI surface is not feasible if it effectively includes all data and functions via kallsyms_lookup_name()".

After seeing this explanation, Hiramatsu agreed that the patch makes sense and offered a Reviewed-by tag. So this concern, too, seems unlikely to prevent this patch set from being merged.

It's worth repeating that discouraging module developers from bypassing the export mechanism is generally seen as more than sufficient motivation to merge a change like this. But it is also interesting to see a large company supporting that kind of change as well. By more closely tying the Android kernel to the mainline, Google would appear to be better aligning its own interests with the long-term interests of the development community — on this point, at least. That, hopefully, will lead to better kernels on our devices that also happen to be a lot closer to mainline kernels.

Appendix 2 - Kallsyms module - Kallsyms lookup name no longer available

Source:

```
//============================================================================
//
// Project: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
// File:    kallsyms.c
// Version: 0.01
//
// Author : Filip Pynckels
//
// Changed: 2020 07 10
//
// Build tools: gcc
//
//============================================================================
//
// This program is free software. You can redistribute it and/or modify it under
// the terms of the MIT Public License.
//
// This software is distributed in the hope that it will be useful, but without
// any warranty. Without even the implied warranty of merchantability or fitness
// for a particular purpose.
//
//============================================================================
//
//============================================================================

// Option 1: kallsyms_lookup_name

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/kallsyms.h>

//============================================================================

MODULE_LICENSE("Dual MIT/GPL");
MODULE_AUTHOR("Filip Pynckels");
MODULE_DESCRIPTION("Test module for: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on");
MODULE_VERSION("0.01");

//============================================================================

static int __init kallsyms_init(void)
{
    int (*fnct1)(unsigned long param);
    int (*fnct2)(unsigned long param);

    printk(KERN_INFO "kallsyms: initializing...\n");
    printk(KERN_INFO "kallsyms: initialized\n");

    fnct1 = (void *)kallsyms_lookup_name("page_is_ram");
```

```
49.     fnct2 = (void *)kallsyms_lookup_name("phys_mem_access_prot");
50.
51.     printk(KERN_INFO "kallsyms: option(1) page_is_ram(%p) phys_mem_access_prot(%p)\n", fnct1, fnct2);
52.
53.     return 0;
54. }
55.
56.
57. static void __exit kallsyms_exit(void)
58. {
59.     printk(KERN_INFO "kallsyms: exiting...\n");
60.     printk(KERN_INFO "kallsyms: exited\n");
61. }
62.
63. //============================================================================
64.
65. module_init(kallsyms_init);
66. module_exit(kallsyms_exit);
```

Results:

```
1. ==> Starting kallsyms test...
2. ==> Building kallsyms module...
3.         ERROR: modpost: "kallsyms_lookup_name" [kallsyms.ko] undefined!
4.         make[2]: *** [scripts/Makefile.modpost:99: __modpost] Error 1
5.         make[1]: *** [Makefile:1645: modules] Error 2
6.         make: *** [Makefile:27: all] Error 2
7. ==> Installing kallsyms module...
8. ==> Ending kallsyms test (rc=1)...
```

Appendix 3 - Kallsyms module - Alternative 1 - Custom kernel with __symbol get

Source:

```
//============================================================================
//
// Project: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
// File:    kallsyms.c
// Version: 0.01
//
// Author : Filip Pynckels
//
// Changed: 2020 07 10
//
// Build tools: gcc
//
//============================================================================
//
// This program is free software. You can redistribute it and/or modify it under
// the terms of the MIT Public License.
//
// This software is distributed in the hope that it will be useful, but without
// any warranty. Without even the implied warranty of merchantability or fitness
// for a particular purpose.
//
//============================================================================

// Option 2:   __symbol_get

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

//============================================================================

MODULE_LICENSE("Dual MIT/GPL");
MODULE_AUTHOR("Filip Pynckels");
MODULE_DESCRIPTION("Test module for: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on");
MODULE_VERSION("0.01");

//============================================================================

static int __init kallsyms_init(void)
{
    int (*fnct1)(unsigned long param);
    int (*fnct2)(unsigned long param);

    printk(KERN_INFO "kallsyms: initializing...\n");
    printk(KERN_INFO "kallsyms: initialized\n");

    fnct1 = __symbol_get("page_is_ram");
    fnct2 = __symbol_get("phys_mem_access_prot");

```

```
    printk(KERN_INFO "kallsyms: option(2) page_is_ram(%p) phys_mem_access_prot(%p)\n", fnct1, fnct2);

    return 0;
}


static void __exit kallsyms_exit(void)
{
    printk(KERN_INFO "kallsyms: exiting...\n");
    printk(KERN_INFO "kallsyms: exited\n");
}

//========================================================================
module_init(kallsyms_init);
module_exit(kallsyms_exit);
```

Results:

```
==> Starting kallsyms test...
==> Building kallsyms module...
==> Installing kallsyms module...
        [ 9253.096736] kallsyms: initializing...
        [ 9253.096738] kallsyms: initialized
        filename:       /home/fp/Projects/kallsyms/kallsyms.ko
        version:        0.01
        description:    Test module for: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
        author:         Filip Pynckels
        license:        Dual MIT/GPL
        srcversion:     631F10709B20CD18EC65C2B
        depends:
        retpoline:      Y
        name:           kallsyms
        vermagic:       5.7.7-arch1-1 SMP preempt mod_unload
        size:           16384
==> Testing kallsyms module...
        [ 9253.096959] kallsyms: option(2) page_is_ram(000000007cc63fd1) phys_mem_access_prot(0000000000000000)
==> Removing kallsyms module...
        [ 9253.257487] kallsyms: exiting...
        [ 9253.257489] kallsyms: exited
==> Ending kallsyms test (rc=0)...
```

Appendix 4 - Kallsyms module - Alternative 2 - kprobes

Source:

```
//=============================================================================
//
// Project: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
// File:    kallsyms.c
// Version: 0.01
//
// Author : Filip Pynckels
//
// Changed: 2020 07 10
//
// Build tools: gcc
//
//=============================================================================
//
// This program is free software. You can redistribute it and/or modify it under
// the terms of the MIT Public License.
//
// This software is distributed in the hope that it will be useful, but without
// any warranty. Without even the implied warranty of merchantability or fitness
// for a particular purpose.
//
//=============================================================================

// Option 3: register_kprobes

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/kprobes.h>

//=============================================================================

MODULE_LICENSE("Dual MIT/GPL");
MODULE_AUTHOR("Filip Pynckels");
MODULE_DESCRIPTION("Test module for: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on");
MODULE_VERSION("0.01");

//=============================================================================

unsigned long lookup_name(const char *name){
    struct kprobe kp;
    unsigned long retval;

    kp.symbol_name = name;
    if (register_kprobe(&kp) < 0) return 0;
    retval = (unsigned long)kp.addr;
    unregister_kprobe(&kp);
    return retval;
}
```

```
50.
51.
52. static int __init kallsyms_init(void)
53. {
54.     int (*fnct1)(unsigned long param);
55.     int (*fnct2)(unsigned long param);
56.
57.     printk(KERN_INFO "kallsyms: initializing...\n");
58.     printk(KERN_INFO "kallsyms: initialized\n");
59.
60.     fnct1 = (void *)lookup_name("page_is_ram");
61.     fnct2 = (void *)lookup_name("phys_mem_access_prot");
62.
63.     printk(KERN_INFO "kallsyms: option(3) page_is_ram(%p) phys_mem_access_prot(%p)\n", fnct1, fnct2);
64.
65.     return 0;
66. }
67.
68.
69. static void __exit kallsyms_exit(void)
70. {
71.     printk(KERN_INFO "kallsyms: exiting...\n");
72.     printk(KERN_INFO "kallsyms: exited\n");
73. }
74.
75. //============================================================================
76.
77. module_init(kallsyms_init);
78. module_exit(kallsyms_exit);
```

Results:

```
1. ==> Starting kallsyms test...
2. ==> Building kallsyms module...
3. ==> Installing kallsyms module...
4.          [ 9292.246963] kallsyms: initializing...
5.          [ 9292.246965] kallsyms: initialized
6.          filename:       /home/fp/Projects/kallsyms/kallsyms.ko
7.          version:        0.01
8.          description:    Test module for: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
9.          author:         Filip Pynckels
10.         license:        Dual MIT/GPL
11.         srcversion:     46A6954E83E1F166C8D1C04
12.         depends:
13.         retpoline:      Y
14.         name:           kallsyms
15.         vermagic:       5.7.7-arch1-1 SMP preempt mod_unload
16.         size:           16384
17. ==> Testing kallsyms module...
18.         [ 9292.283991] kallsyms: option(3) page_is_ram(000000007cc63fd1) phys_mem_access_prot(00000000b05c04cd)
19. ==> Removing kallsyms module...
20.         [ 9292.415585] kallsyms: exiting...
21.         [ 9292.415587] kallsyms: exited
```

```
22. ==> Ending kallsyms test (rc=0)...
```

Appendix 5 - Kallsyms testscript and Makefile

Test script:

```
#!/bin/bash

#================================================================================
#
# Project: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
# File:    kallsymsmod_test.sh
# Version: 0.01
#
# Author : Filip Pynckels
#
# Changed: 2020 07 10
#
# Tools:   gcc
#
#================================================================================
#
# This program is free software. You can redistribute it and/or modify it under
# the terms of the MIT Public License.
#
# This software is distributed in the hope that it will be useful, but without
# any warranty. Without even the implied warranty of merchantability or fitness
# for a particular purpose.
#
#================================================================================

clear

# --- Set constants ----------------------------

MSG='\033[1;32m'
OUT='\033[0;37m'
NONE='\033[0m'

# --- Define helper functions ------------------

end_test()
{
    if [ -n $1 ]; then rc=$1; else rc=0; fi     # Handle function parameter(s)

    echo -e "${MSG}==> Ending kallsyms test (rc=$rc)...${OUT}"

    sudo rmmod 'kallsyms' 2>/dev/null           # Unload module if loaded
    make -s clean                               # Clean up the build files
    rm kallsyms.c 2>/dev/null                   # Clean up the test files

    exit $rc
}


```

```
50. show_help_and_exit()
51. {
52.
53.     if [ -n $1 ]; then rc=$1; else rc=0; fi     # Handle function parameter(s)
54.
55.     echo ""
56.     echo "        Usage: $0 [-1][-2][-3]"
57.     echo ""
58.     echo "          -1    Run test for option 1: kallsyms_lookup_name()"
59.     echo "          -2    Run test for option 2: __symbol_get()"
60.     echo "          -3    Run test for option 3: register_kprobe()"
61.     echo ""
62.     echo "        Default is to show this help screen"
63.     echo ""
64.
65.     end_test $rc
66. }
67.
68. # --- Start test -------------------------------
69.
70.    echo -e "${MSG}==> Starting kallsyms test...${OUT}"
71.
72. # --- Handle argument(s) -----------------------
73.
74. if [ $# -eq 0 ]; then show_help_and_exit 1; fi  # No parameters
75.
76. while getopts "123?" arg 2>/dev/null; do        # Handle all given arguments
77.   case "$arg" in
78.     1) cp kallsyms_option1.c kallsyms.c ;;      # Test module option 1
79.     2) cp kallsyms_option2.c kallsyms.c ;;      # Test module option 2
80.     3) cp kallsyms_option3.c kallsyms.c ;;      # Test module option 3
81.     *) show_help_and_exit 1             ;;      # Show help - Exit with error
82.   esac
83. done
84.
85. # --- Build kernel module ----------------------
86.
87. echo -e "${MSG}==> Building kallsyms module...${OUT}"
88.
89. make -s 2>&1                                    \
90.    |  sed 's/.*/        &/'
91.
92. rc=$?
93. if [ $rc -ne 0 ]; then end_test $rc; fi
94.
95. # --- Install kernel module for testing --------
96.
97. echo -e "${MSG}==> Installing kallsyms module...${OUT}"
98.
99. sudo insmod kallsyms.ko 2>/dev/null
100. rc=$?
101. if [ $rc -ne 0 ]; then end_test $rc; fi
102.
103. # --- Show system info -------------------------
```

```
104.
105. sudo dmesg                                      \
106.    | grep 'kallsyms:'                           \
107.    | tail -n3                                  \
108.    | head -n2                                  \
109.    | sed 's/.*/        &/'
110.
111. modinfo kallsyms.ko | sed 's/.*/        &/'
112.
113. lsmod                                           \
114.    | grep 'kallsyms'                           \
115.    | awk '{print $2}'                          \
116.    | sed 's/.*/        size:           &/'
117.
118. # --- Test installed module ---------------------
119.
120. echo -e "${MSG}==> Testing kallsyms module...${OUT}"
121.
122. ID=$(sudo dmesg                                 \
123.        | grep 'kallsyms:'                       \
124.        | tail -1                                \
125.        | sed 's/\[\(.*\)\..*/\1/')
126.
127. sudo dmesg                                      \
128.    | grep "$ID"                                \
129.    | grep -v 'initializ'                       \
130.    | grep -v 'exit'                            \
131.    | sed 's/.*/        &/'
132.
133. # Remove kernel module -------------------------
134.
135. echo -e "${MSG}==> Removing kallsyms module...${OUT}"
136.
137. sudo rmmod 'kallsyms' 2>/dev/null
138. rc=$?
139. if [ $rc -ne 0 ]; then end_test $rc; fi
140.
141. # --- Show system info -------------------------
142.
143. sudo dmesg                                      \
144.    | grep 'kallsyms:'                          \
145.    | tail -n2                                  \
146.    | sed 's/.*/        &/'
147.
148. lsmod                                           \
149.    | grep 'kallsyms'
150.
151. # --- End without error ------------------------
152.
153. end_test 0
```

Makefile:

```
#============================================================================
#
# Project: Unexported kallsyms_lookup_name from Linux kernel 5.7.7 on
# File:    Makefile
# Version: 0.01
#
# Author : Filip Pynckels
#
# Changed: 2020 07 10
#
# Tools:   gcc
#
#============================================================================
#
# This program is free software. You can redistribute it and/or modify it under
# the terms of the MIT Public License.
#
# This software is distributed in the hope that it will be useful, but without
# any warranty. Without even the implied warranty of merchantability or fitness
# for a particular purpose.
#
#============================================================================

obj-m += kallsyms.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```