Klever Tutorial

This tutorial describes a basic workflow of using Klever.

Deployment

There is rather good documentation about Klever deployment. You can find it at https://klever.readthedocs.io/en/latest/. In this tutorial we assume that you deploy Klever locally on Debian 9 in the production mode with default settings from the latest master. In addition, we assume that your username is ***debian*** and your home directory is ***/home/debian***[1].

Preparing Build Bases

After a successful deployment of Klever you need to prepare a *build base* on the same machine where you deployed Klever. This tutorial treats just build bases for Linux kernel loadable modules since the publicly available version of Klever supports verification of other software in the experimental stage. You should not expect that Klever supports all versions and configurations of the Linux kernel well. There is a big list of things to do in this direction.

Below we consider as an example preparation of a build base for verification of Linux 3.14.79 modules (architecture *x86_64*, configuration *allmodconfig*, GCC 4.8.5). You can try to execute similar steps for other versions and configurations of the Linux kernel at your own risks. To build new versions of the Linux kernel you may need newer versions of GCC.

You can download the archive of the target build base prepared in advance from here. Let's assume that you decompress this archive into directory ***/home/debian/build-base-linux-3.14.79-x86_64-allmodconfig*** so that there should be file *meta.json* directly at the top level in that directory.

To prepare the target build base from scratch you can follow the next steps:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v3.x/linux-3.14.79.tar.xz

$ tar -xvf linux-3.14.79.tar.xz

$ cd linux-3.14.79/
```

[1] If this is not the case, you should adjust paths to build bases below respectively.

```
$ make allmodconfig
$ clade -w ~/build-base-linux-3.14.79-x86_64-allmodconfig -p klever_linux_kernel make -j8 modules
```

Then you will need to wait for quite a long period of time depending on the performance of your machine.

Signing in

Before performing all other actions described further in this tutorial you need to sign in to a Klever web interface:

1. Open page *http://localhost:8998* in your web-browser[2].
2. Input ***manager*** as a username and a password and sign in (Fig. 1).

Then you will be automatically redirected to a *job tree* page presented in the following sections.

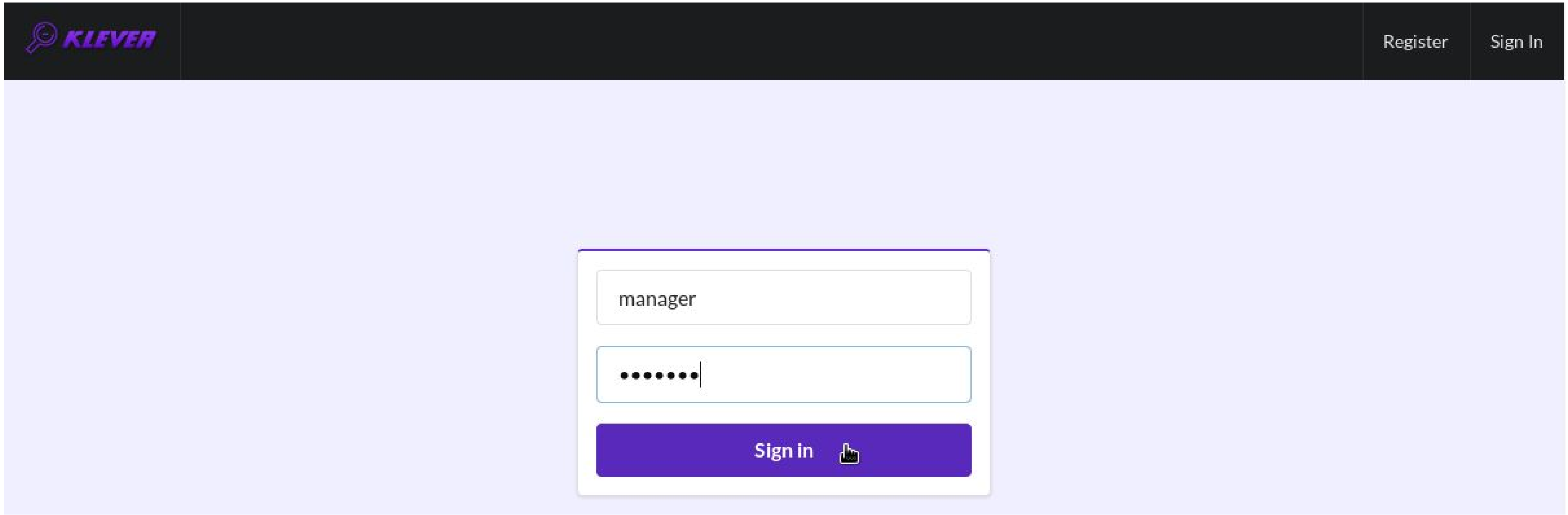


Figure 1. Signing in

[2] You can open the Klever web interface from other machines as well, but you need to set up appropriate access for that.

Starting Verification

As an example we consider checking usage of clocks in device drivers. To start up verification you need to do as follows:

1. Start the creation of a new *job* (Fig. 2).
2. Specify an appropriate title and create the new job (Fig. 3).
3. To configure a first *job version* you need to specify (Fig. 4):
    a. The path to the prepared build base that is ***/home/debian/build-base-linux-3.14.79-x86_64-allmodconfig***.
    b. Targets, e.g. device drivers, i.e. all modules from directory ***drivers*** in our example.
    c. Requirement specifications to be checked, e.g. ***drivers:clk1*** and ***drivers:clk2*** in our example (you can see a complete list of supported requirement specifications at the end of this section).
4. Press *Ctrl-S* when being at the editor window to save changes.
5. Start a *decision of the job version* (Fig. 4).

After that Klever automatically redirects you to a job version/decision page that is described in detail in the following sections.

Later you can create new jobs by opening the job tree page, e.g. through clicking on the Klever logo (Fig. 5), and by executing steps above. You can create new jobs even when some job version is being decided, but job versions are decided one by one by default.

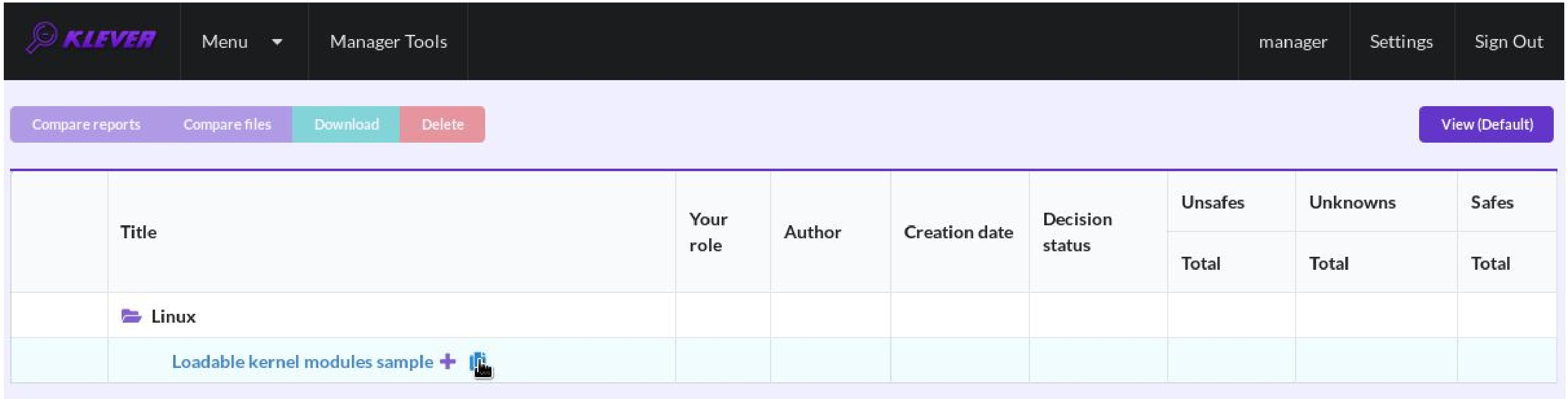
Below there are requirement specifications that you can choose for verification of Linux loadable kernel modules (we do not recommend to check requirement specifications which identifiers are italicised since they produce either many false alarms or there are just a few violations of these requirements at all):

1. alloc:irq
2. alloc:spinlock
3. alloc:usb lock
4. arch:asm:dma-mapping
5. arch:mm:ioremap
6. *block:blk-core:queue*
7. *block:blk-core:request*
8. *block:genhd*
9. *concurrency safety*
10. drivers:base:class
11. drivers:usb:core:usb:coherent
12. drivers:usb:core:usb:dev
13. drivers:usb:core:driver
14. drivers:usb:core:urb
15. drivers:usb:gadget:udc-core
16. drivers:clk1
17. drivers:clk2
18. fs:sysfs:group
19. kernel:locking:mutex
20. kernel:locking:rwlock
21. kernel:locking:spinlock
22. kernel:module
23. *kernel:rcu:update:lock bh*
24. *kernel:rcu:update:lock shed*
25. kernel:rcu:update:lock
26. *kernel:rcu:srcu*
27. *kernel:sched:completion*
28. *lib:find_next_bit*
29. *lib:idr*
30. memory safety
31. net:core:dev
32. *net:core:rtnetlink*
33. *net:core:sock*

In case of verification of the Linux kernel rather than vanilla 3.14.79, you may need to specify one extra parameter ***specifications set***, when configuring the job version (Fig. 4), with a value from the following list:

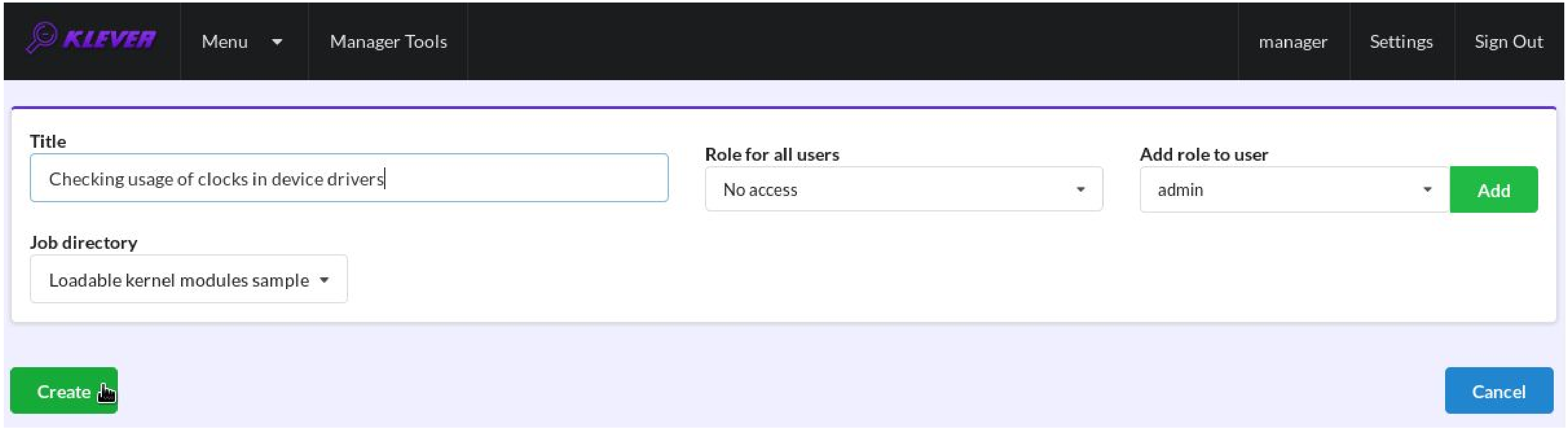
1. 2.6.33
2. 4.6.7
3. 4.15
4. 4.17
5. 5.5

These specification sets correspond to vanilla versions of the Linux kernel. You should select such a specifications set that matches your custom version of the Linux kernel better through trial and error.



	Title	Your role	Author	Creation date	Decision status	Unsafes	Unknowns	Safes
						Total	Total	Total
	Linux							
	Loadable kernel modules sample +							

Figure 2. Starting the creation of a new job



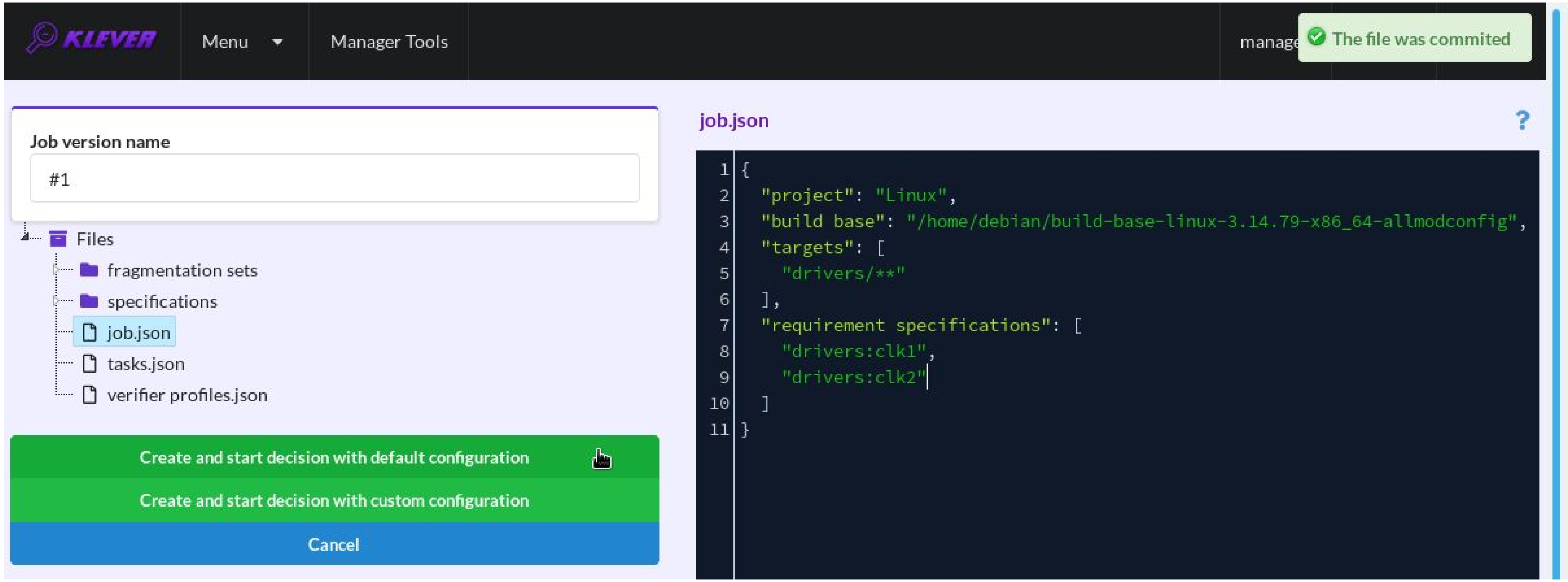
Title: Checking usage of clocks in device drivers

Role for all users: No access

Add role to user: admin

Job directory: Loadable kernel modules sample

Figure 3. The creation of the new job

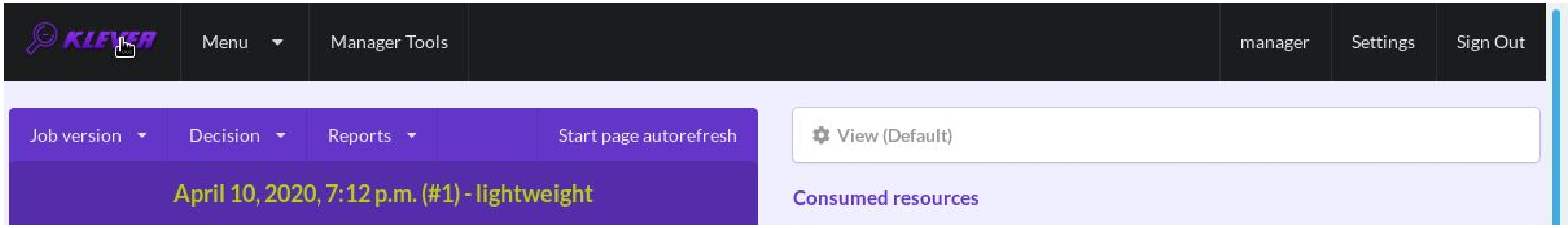


The file was commited

Job version name: #1

```
{
  "project": "Linux",
  "build base": "/home/debian/build-base-linux-3.14.79-x86_64-allmodconfig",
  "targets": [
    "drivers/**"
  ],
  "requirement specifications": [
    "drivers:clk1",
    "drivers:clk2"
  ]
}
```

Figure 4. Configuring the first job version and starting its decision



April 10, 2020, 7:12 p.m. (#1) - lightweight

Figure 5. Opening the job tree page

Decision Progress

At the beginning of the decision of the job version Klever indexes each new build base. This can take rather much time before it starts to generate and to decide first *tasks*[3] for large build bases. In about 15 minutes you can refresh the page and see some tasks and their decisions there. Please, note that the automatic refresh of the job version/decision page stops after 5 minutes, so you either need to refresh it through web browser means or request Klever to switch it on back (Fig. 6).

Before the job version is eventually decided Klever estimates and provides a *decision progress* (Fig. 7-8). You should keep in mind that Klever collects statistics for 10% of tasks before it starts predicting an approximate remaining time for their decision. After that, it recalculates it on the base of new, accumulated statistics. In our example it takes 1 day and 2 hours to decide the job version completely (Fig. 9).

At the job tree page you can see all versions of particular jobs (Fig. 10) and their *decision statutes* (Fig. 11). Besides, you can open the page with details of the decision of the latest job version (Fig. 12) or the page describing the decision of the particular job version (Fig. 13).

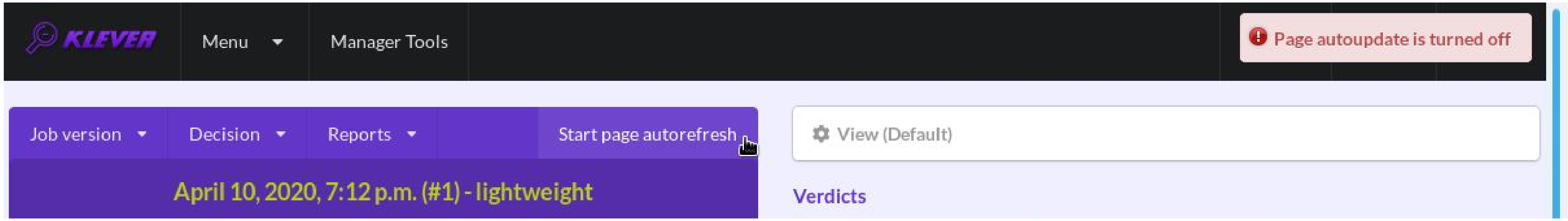
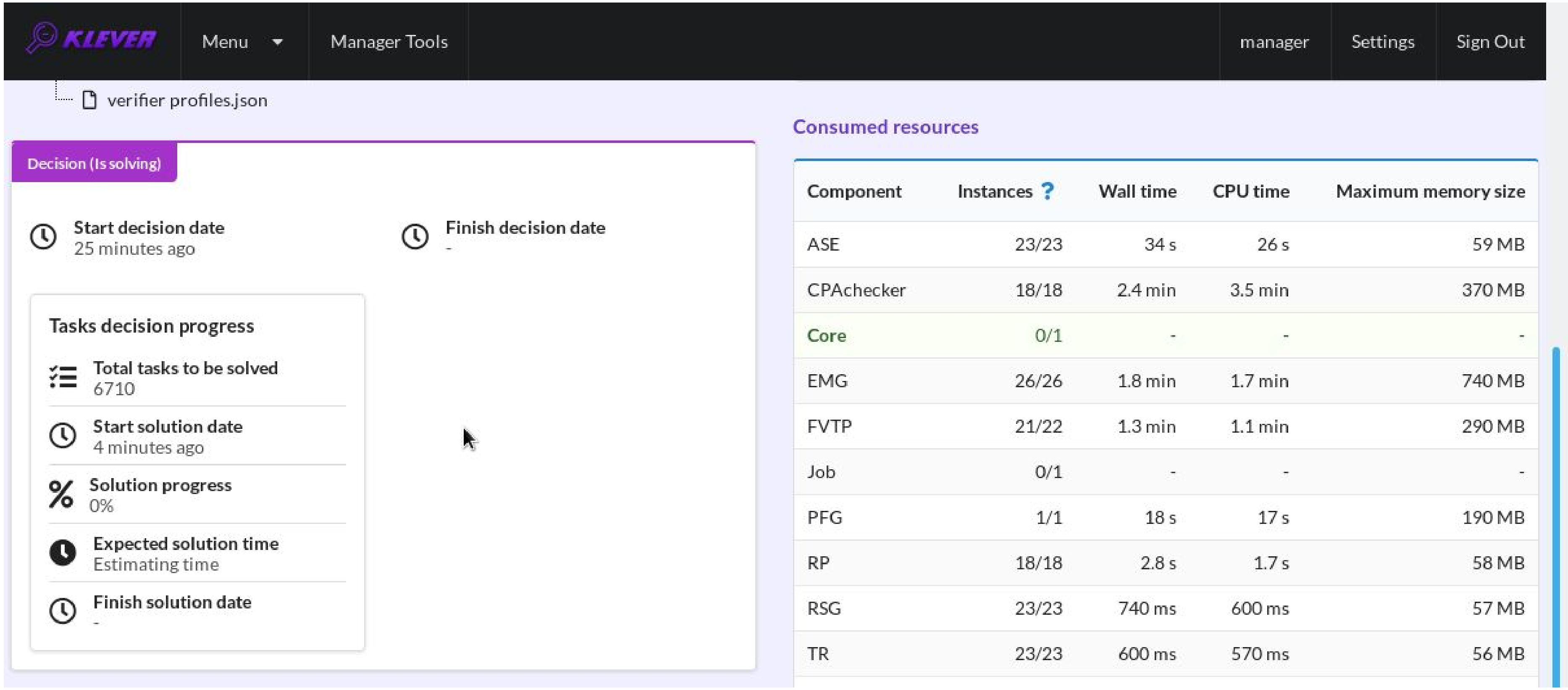


Figure 6. Switching on the automatic refresh of the job version/decision page

[3] For the considered example each task is a pair of a Linux loadable kernel module and a requirements specification. There are 3355 modules under verification and 2 requirement specifications to be checked, so there are 6710 tasks in total.



Start decision date
25 minutes ago

Finish decision date
-

Tasks decision progress

Total tasks to be solved
6710

Start solution date
4 minutes ago

Solution progress
0%

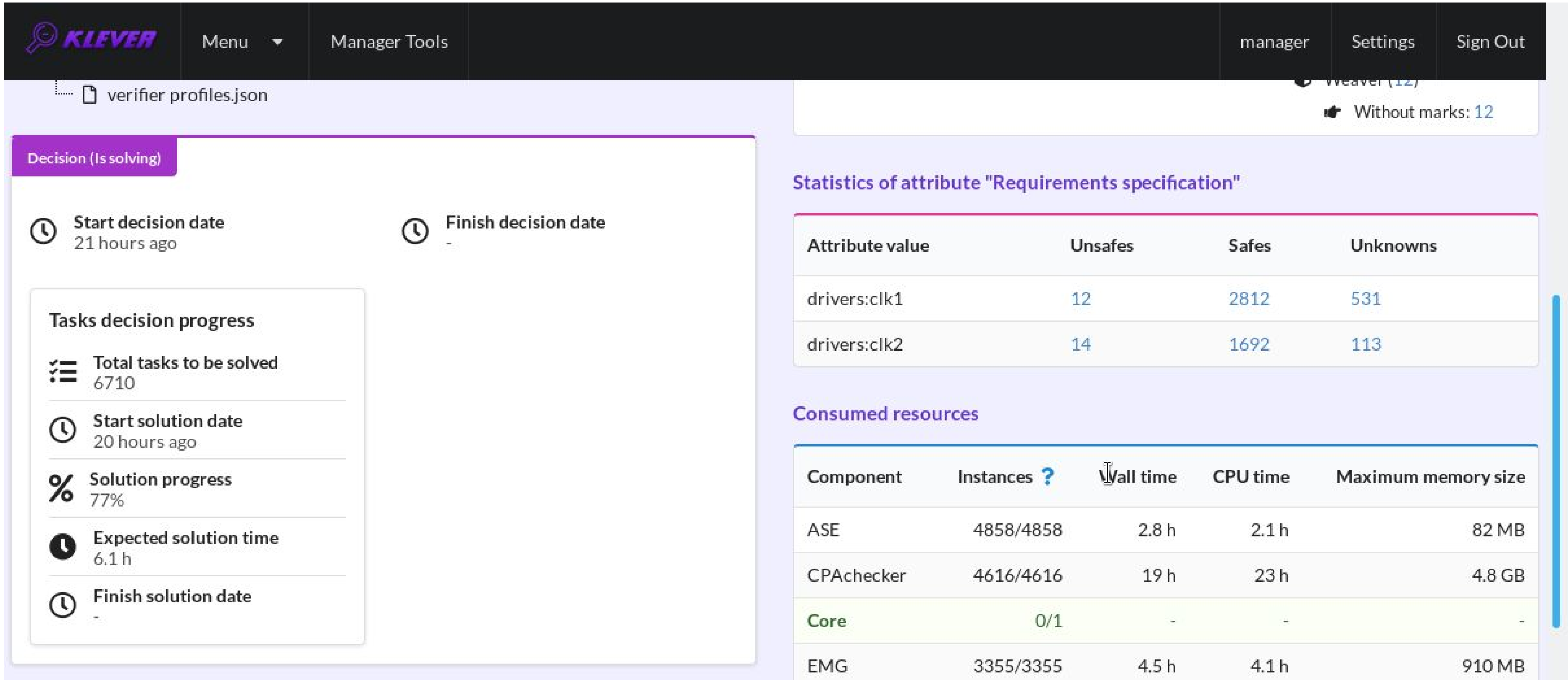
Expected solution time
Estimating time

Finish solution date
-

Consumed resources

Component	Instances ?	Wall time	CPU time	Maximum memory size
ASE	23/23	34 s	26 s	59 MB
CPAchecker	18/18	2.4 min	3.5 min	370 MB
Core	0/1	-	-	-
EMG	26/26	1.8 min	1.7 min	740 MB
FVTP	21/22	1.3 min	1.1 min	290 MB
Job	0/1	-	-	-
PFG	1/1	18 s	17 s	190 MB
RP	18/18	2.8 s	1.7 s	58 MB
RSG	23/23	740 ms	600 ms	57 MB
TR	23/23	600 ms	570 ms	56 MB

Figure 7. The progress of the decision of the job version (estimating a remaining time)



Start decision date
21 hours ago

Finish decision date
-

Tasks decision progress

Total tasks to be solved
6710

Start solution date
20 hours ago

Solution progress
77%

Expected solution time
6.1 h

Finish solution date
-

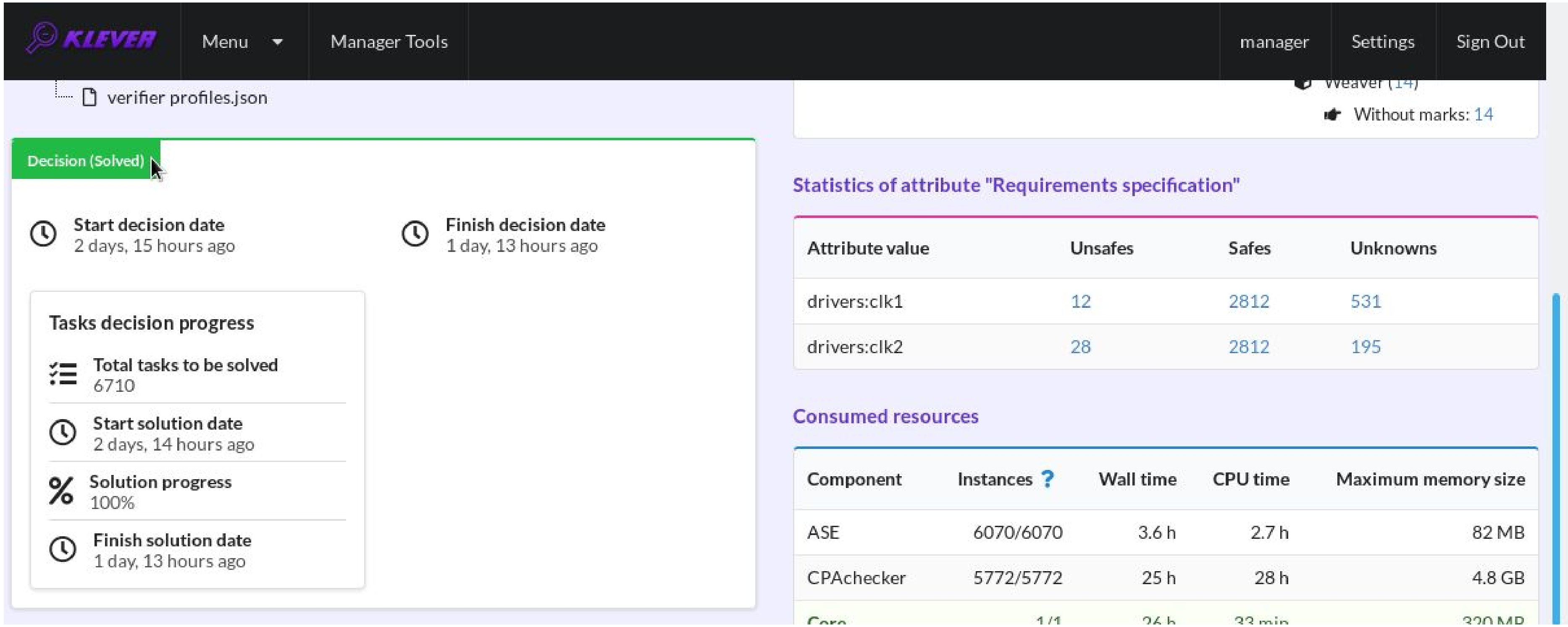
Statistics of attribute "Requirements specification"

Attribute value	Unsafes	Safes	Unknowns
drivers:clk1	12	2812	531
drivers:clk2	14	1692	113

Consumed resources

Component	Instances ?	Wall time	CPU time	Maximum memory size
ASE	4858/4858	2.8 h	2.1 h	82 MB
CPAchecker	4616/4616	19 h	23 h	4.8 GB
Core	0/1	-	-	-
EMG	3355/3355	4.5 h	4.1 h	910 MB

Figure 8. The progress of the decision of the job version (the remaining time is estimated)



Start decision date
2 days, 15 hours ago

Finish decision date
1 day, 13 hours ago

Tasks decision progress

Total tasks to be solved
6710

Start solution date
2 days, 14 hours ago

Solution progress
100%

Finish solution date
1 day, 13 hours ago

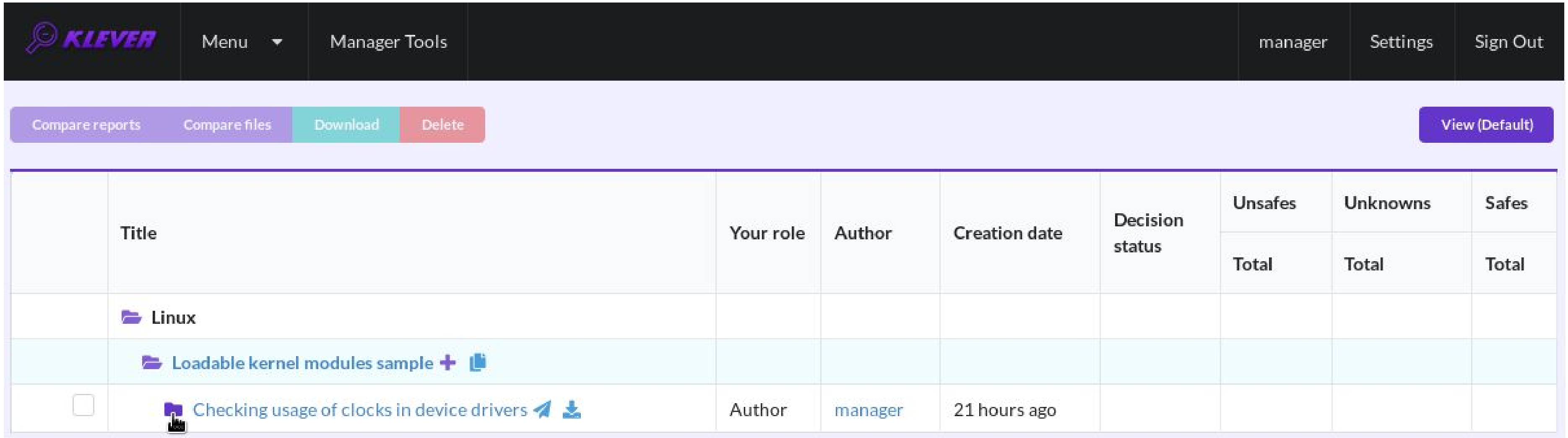
Statistics of attribute "Requirements specification"

Attribute value	Unsafes	Safes	Unknowns
drivers:clk1	12	2812	531
drivers:clk2	28	2812	195

Consumed resources

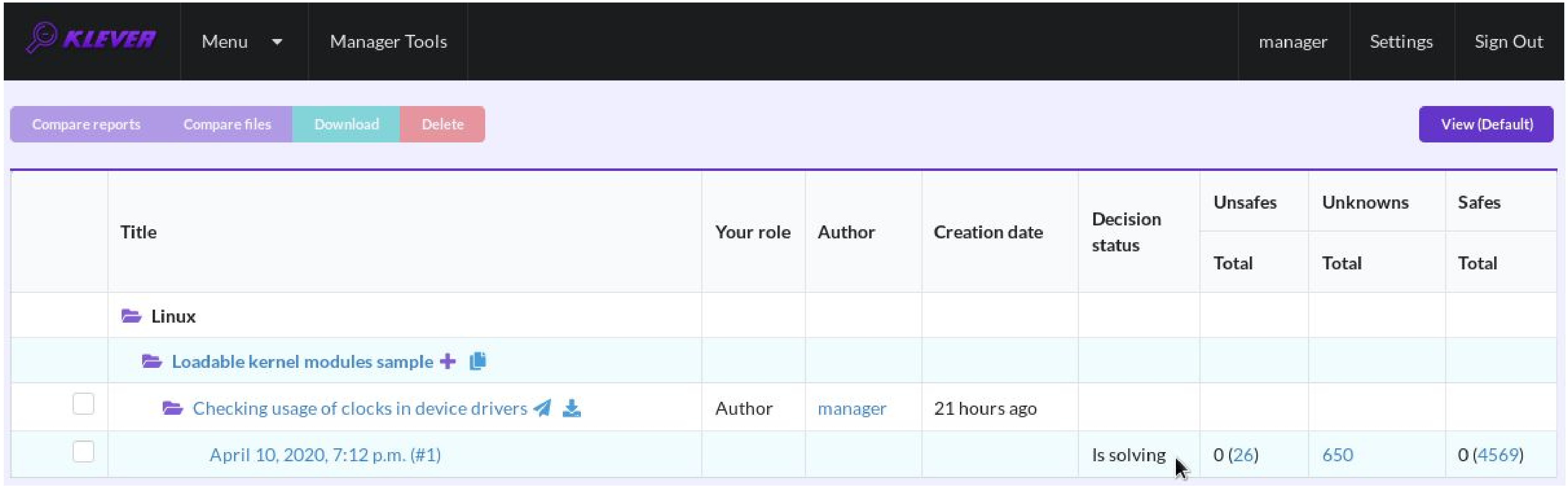
Component	Instances ?	Wall time	CPU time	Maximum memory size
ASE	6070/6070	3.6 h	2.7 h	82 MB
CPAchecker	5772/5772	25 h	28 h	4.8 GB
Core	1/1	26 h	33 min	320 MB

Figure 9. The completed decision of the job version



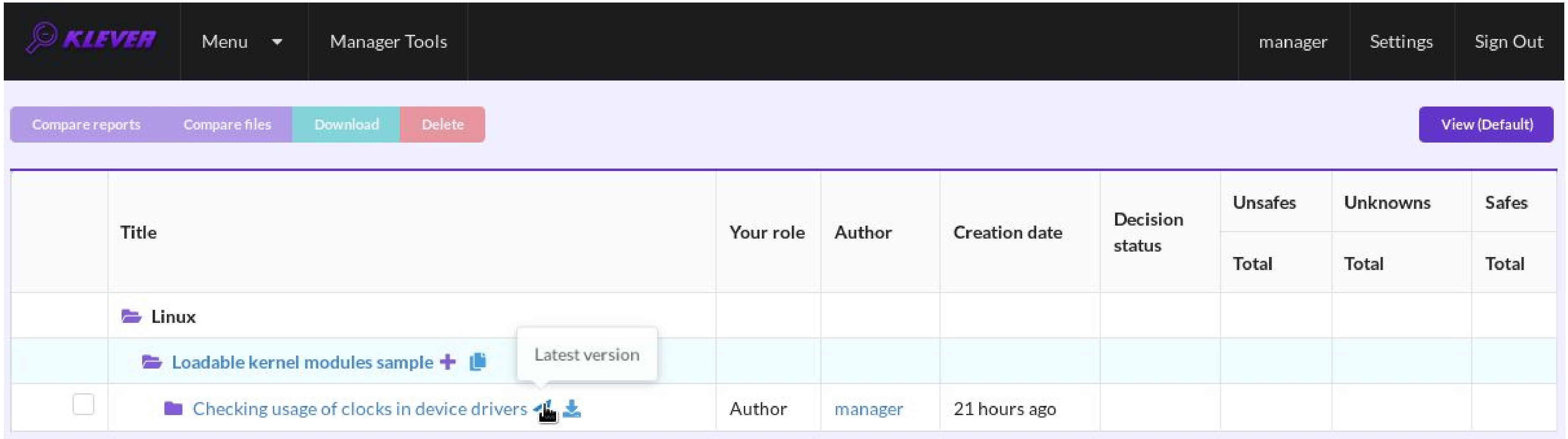
	Title	Your role	Author	Creation date	Decision status	Unsafes	Unknowns	Safes
						Total	Total	Total
	Linux							
	Loadable kernel modules sample							
□	Checking usage of clocks in device drivers	Author	manager	21 hours ago				

Figure 10. Showing job versions



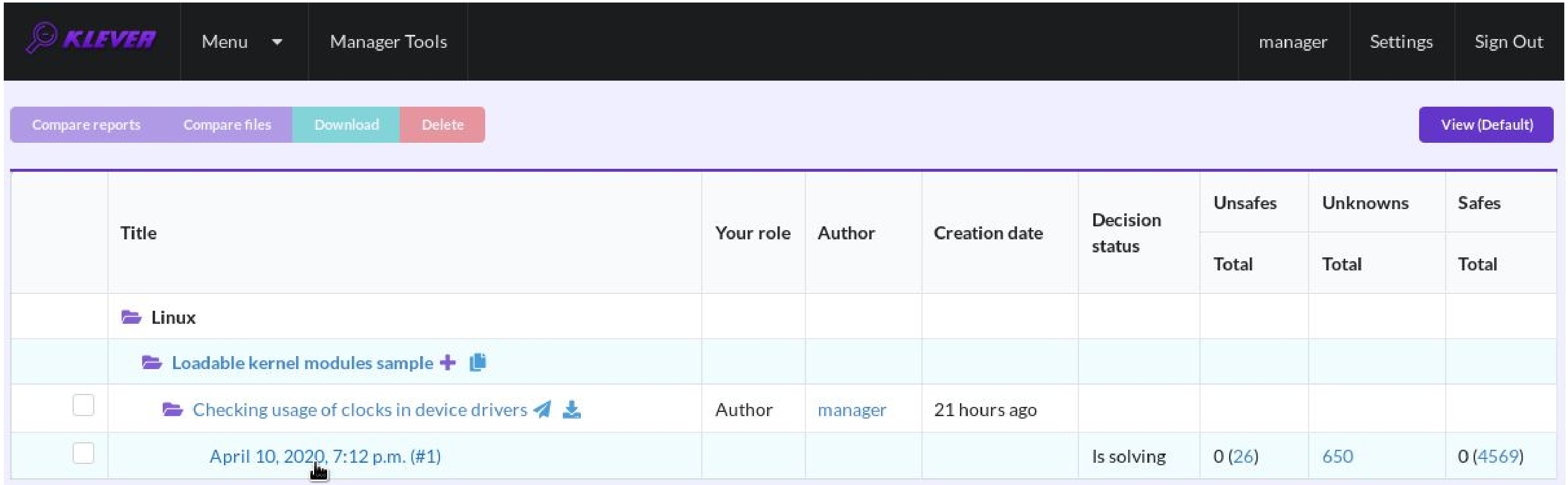
	Title	Your role	Author	Creation date	Decision status	Unsafes	Unknowns	Safes
						Total	Total	Total
	Linux							
	Loadable kernel modules sample							
□	Checking usage of clocks in device drivers	Author	manager	21 hours ago				
□	April 10, 2020, 7:12 p.m. (#1)				Is solving	0 (26)	650	0 (4569)

Figure 11. The status of the decision of the job version



	Title	Your role	Author	Creation date	Decision status	Unsafes	Unknowns	Safes
						Total	Total	Total
	Linux							
	Loadable kernel modules sample							
□	Checking usage of clocks in device drivers	Author	manager	21 hours ago				

Figure 12. Opening the page with the decision of the latest job version



	Title	Your role	Author	Creation date	Decision status	Unsafes	Unknowns	Safes
						Total	Total	Total
	Linux							
	Loadable kernel modules sample							
□	Checking usage of clocks in device drivers	Author	manager	21 hours ago				
□	April 10, 2020, 7:12 p.m. (#1)				Is solving	0 (26)	650	0 (4569)

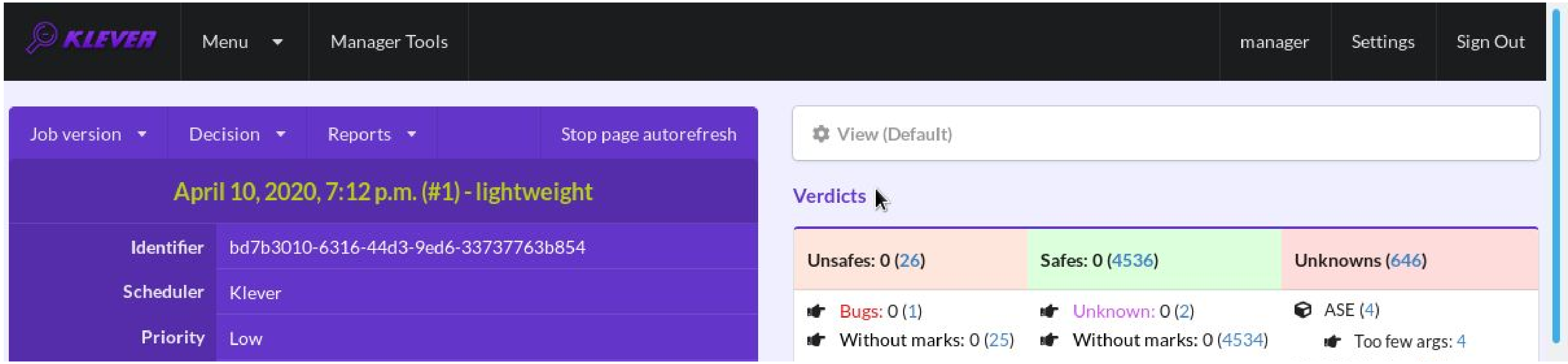
Figure 13. Opening the page with the decision of the particular job version

Analyzing Verification Results

Klever can fail to generate and to decide tasks. In this case it provides users with *unknown* verdicts, otherwise there are *safe* or *unsafe* verdicts (Fig. 14). You already saw the example with summaries of these verdicts at the job tree page (Fig. 10-11). In this tutorial we do not consider other verdicts rather than unsafes that are either violations of checked requirements or false alarms (Fig. 15). Klever reports unsafes if so during the decision of the job version and you can assess them both during the decision and after its completion.

During assessment of unsafes experts can create marks that can match other unsafes with similar error traces (we consider marks and error traces in detail within the next section). For instance, there is a preset mark for a sample job that matches one of the reported unsafes (Fig. 16). Automatic assessment can reduce efforts for analysis of verification results considerably, e.g. when verifying several versions or configurations of the same software. But experts should analyze such automatically assessed unsafes since the same mark can match unsafes with error traces that look very similar but correspond to different faults. Unsafes without marks need assessment as well (Fig. 17). When checking several requirement specifications in the same job, one is able to analyze unsafes just for a particular requirements specification (Fig. 18).

After clicking on the links in Fig. 15-18 you will be redirected to pages with lists of corresponding unsafes (e.g. Fig. 19) except for if there is the only element in this list an error trace will be shown immediately. For further analysis we recommend clicking on an unsafe index on the left to open a new page in a separate tab (Fig. 20). To return back to the job version/decision page you can click on the title of the job decision on the top left (Fig. 21). This can be done at any page with such the link.



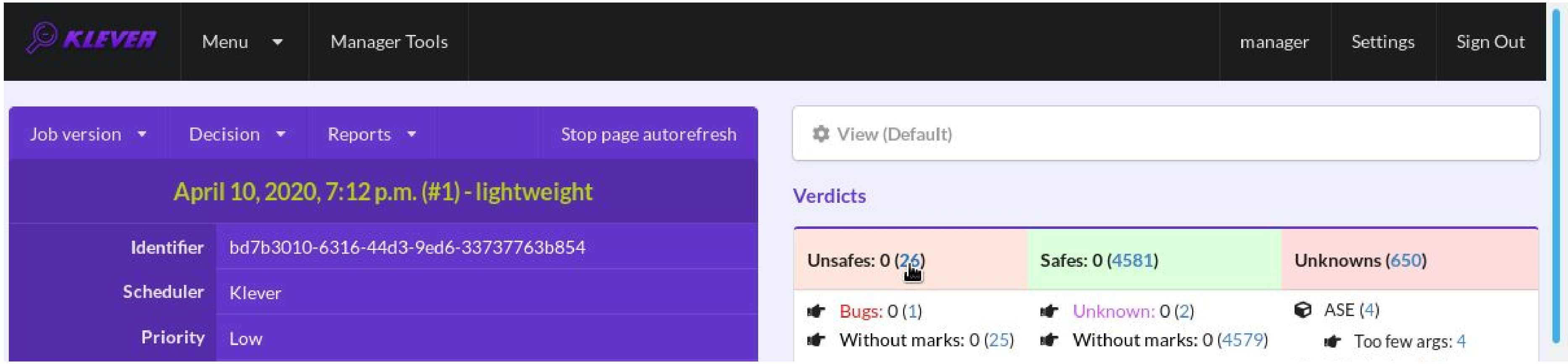
April 10, 2020, 7:12 p.m. (#1) - lightweight

Identifier	bd7b3010-6316-44d3-9ed6-33737763b854
Scheduler	Klever
Priority	Low

Verdicts

Unsafes: 0 (26)	Safes: 0 (4536)	Unknowns (646)
Bugs: 0 (1) Without marks: 0 (25)	Unknown: 0 (2) Without marks: 0 (4534)	ASE (4) Too few args: 4

Figure 14. Verdicts



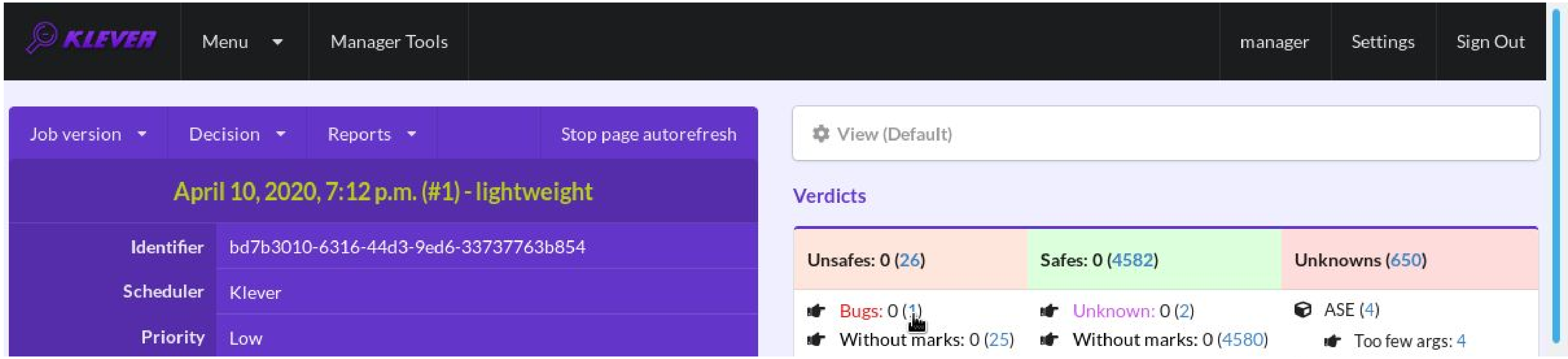
April 10, 2020, 7:12 p.m. (#1) - lightweight

Identifier	bd7b3010-6316-44d3-9ed6-33737763b854
Scheduler	Klever
Priority	Low

Verdicts

Unsafes: 0 (26)	Safes: 0 (4581)	Unknowns (650)
Bugs: 0 (1) Without marks: 0 (25)	Unknown: 0 (2) Without marks: 0 (4579)	ASE (4) Too few args: 4

Figure 15. The total number of unsafes reported thus far



April 10, 2020, 7:12 p.m. (#1) - lightweight

Identifier bd7b3010-6316-44d3-9ed6-33737763b854

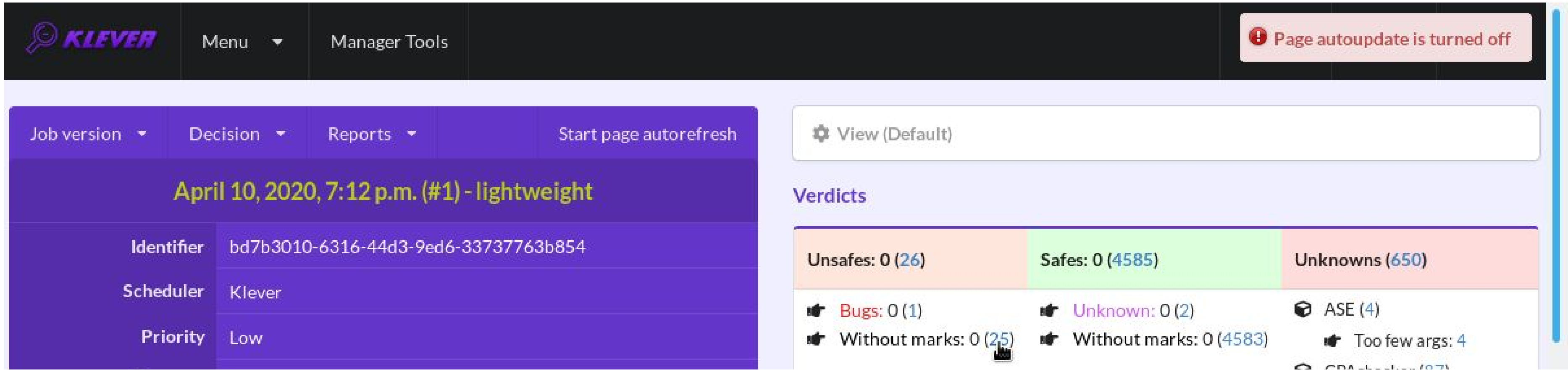
Scheduler Klever

Priority Low

Verdicts

Unsafes: 0 (26)	Safes: 0 (4582)	Unknowns (650)
Bugs: 0 (1) Without marks: 0 (25)	Unknown: 0 (2) Without marks: 0 (4580)	ASE (4) Too few args: 4

Figure 16. The total number of automatically assessed unsafes



Page autoupdate is turned off

April 10, 2020, 7:12 p.m. (#1) - lightweight

Identifier bd7b3010-6316-44d3-9ed6-33737763b854

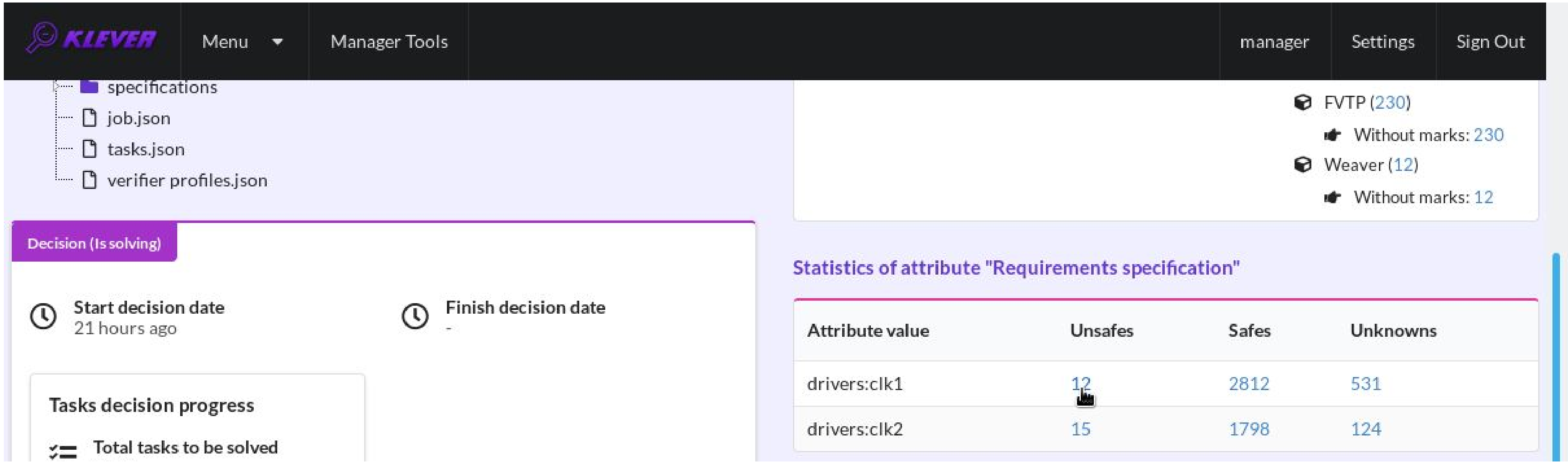
Scheduler Klever

Priority Low

Verdicts

Unsafes: 0 (26)	Safes: 0 (4585)	Unknowns (650)
Bugs: 0 (1) Without marks: 0 (25)	Unknown: 0 (2) Without marks: 0 (4583)	ASE (4) Too few args: 4

Figure 17. The total number of unsafes without any assessment



specifications
job.json
tasks.json
verifier profiles.json

Decision (Is solving)

Start decision date
21 hours ago

Finish decision date
-

Tasks decision progress

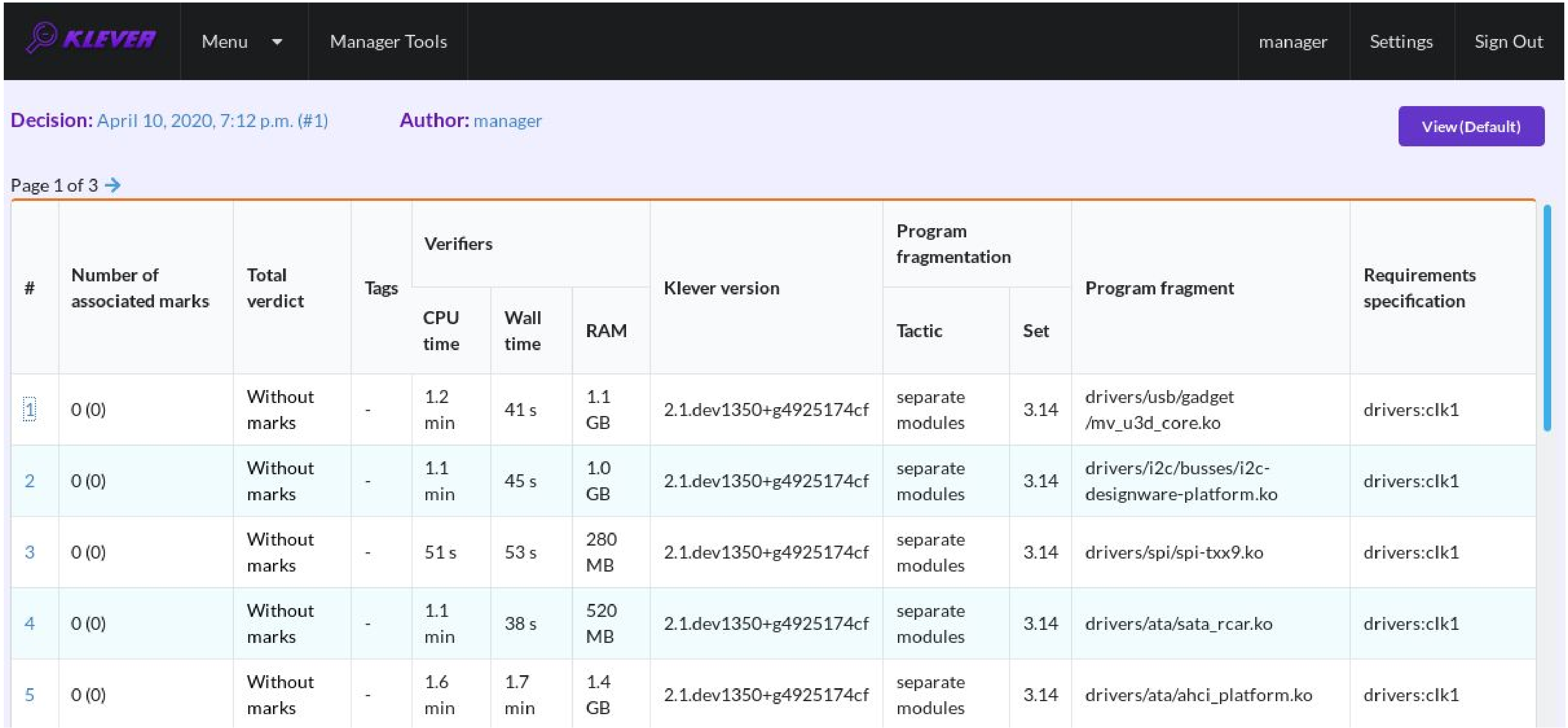
Total tasks to be solved

FVTP (230)
Without marks: 230
Weaver (12)
Without marks: 12

Statistics of attribute "Requirements specification"

Attribute value	Unsafes	Safes	Unknowns
drivers:clk1	12	2812	531
drivers:clk2	15	1798	124

Figure 18. The total number of unsafes corresponding to the particular requirements specification

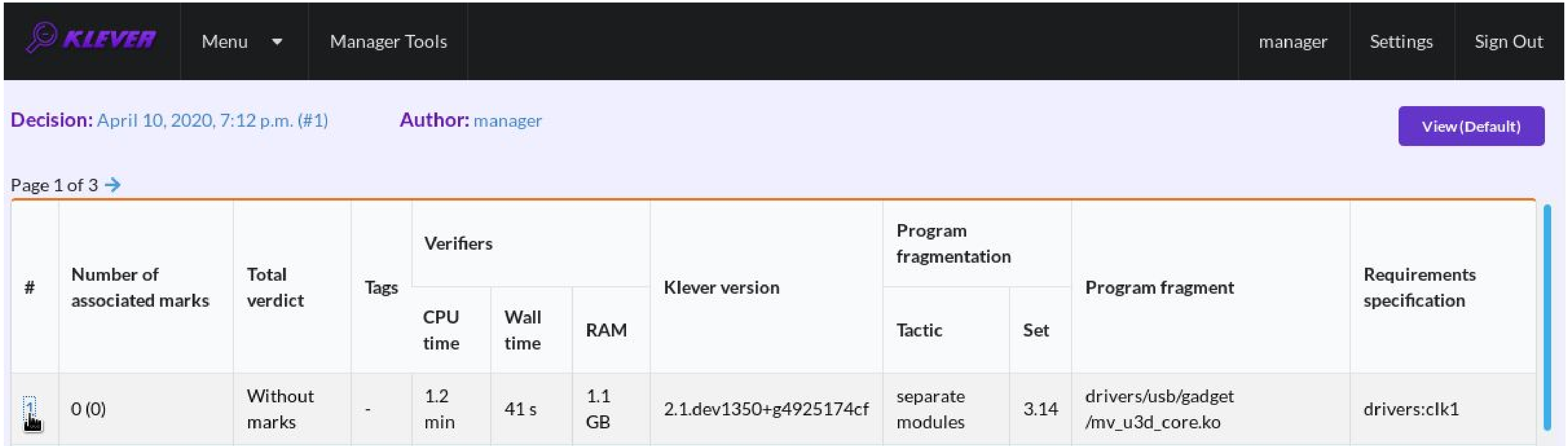


Decision: April 10, 2020, 7:12 p.m. (#1) **Author:** manager

Page 1 of 3 →

#	Number of associated marks	Total verdict	Tags	Verifiers			Klever version	Program fragmentation		Program fragment	Requirements specification
				CPU time	Wall time	RAM		Tactic	Set		
1	0 (0)	Without marks	-	1.2 min	41 s	1.1 GB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/usb/gadget /mv_u3d_core.ko	drivers:clk1
2	0 (0)	Without marks	-	1.1 min	45 s	1.0 GB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/i2c/busses/i2c-designware-platform.ko	drivers:clk1
3	0 (0)	Without marks	-	51 s	53 s	280 MB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/spi/spi-txx9.ko	drivers:clk1
4	0 (0)	Without marks	-	1.1 min	38 s	520 MB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/ata/sata_rcar.ko	drivers:clk1
5	0 (0)	Without marks	-	1.6 min	1.7 min	1.4 GB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/ata/ahci_platform.ko	drivers:clk1

Figure 19. The list of unsafes without any assessment

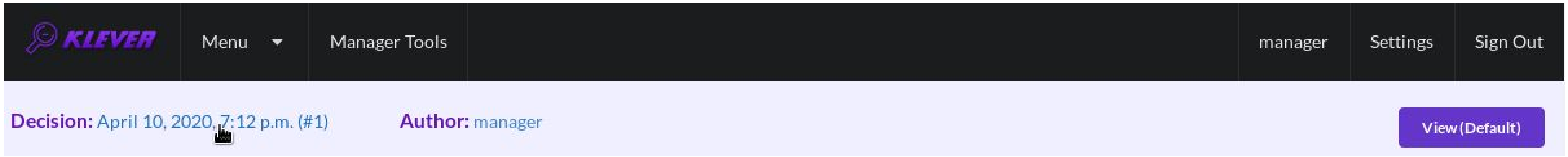


Decision: April 10, 2020, 7:12 p.m. (#1) **Author:** manager

Page 1 of 3 →

#	Number of associated marks	Total verdict	Tags	Verifiers			Klever version	Program fragmentation		Program fragment	Requirements specification
				CPU time	Wall time	RAM		Tactic	Set		
1	0 (0)	Without marks	-	1.2 min	41 s	1.1 GB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/usb/gadget/mv_u3d_core.ko	drivers:clk1

Figure 20. Opening the error trace corresponding to the unsafe without any assessment



Decision: April 10, 2020, 7:12 p.m. (#1) **Author:** manager

Figure 21. Moving back to the job version/decision page

Analyzing Error Traces

After clicking on links within the list of unsafes like in Fig. 20, you will see corresponding error traces. For instance, Fig. 22 demonstrates an error trace example for module *drivers/usb/gadget/mv_u3d_core.ko* and requirements specification *drivers:clk1*.

An *error trace* is a sequence of declarations and statements in a source code of a module under verification and an *environment model*[4] generated by Klever. Besides, within that sequence there are *assumptions* specifying conditions that a software model

[4] Environment models emulate interactions of target programs or *program fragments* like Linux kernel loadable modules with their environment like libraries, user inputs, interruptions and so on. Ideally they should cover only those interaction scenarios that are possible during real executions, but usually this is not

checker considers to be true. Declarations, statements and assumptions represent a path starting from an entry point and ending at a violation of one of checked requirements. The entry point analogue for userspace programs is the function *main* while for Linux loadable kernel modules entry points are generated by Klever as a part of environment models. Requirement violations do not always correspond to places where detected faults should be fixed. For instance, the developer can omit a check for a return value of a function that can fail. As a result various issues, such as leaks or null pointer dereferences, can be revealed somewhere later.

Numbers in the left column correspond to line numbers in source files and models. Source files and models are displayed to the right of error traces. Fig. 22 does not contain anything at the right part of the window since there should be the environment model containing the generated *main* function but by default models are not demonstrated for users in the web interface. If you click on a line number corresponding to an original source file, you will see this source file as in Fig. 23.

You can click on eyes and on rectangles to show hidden parts of the error trace (Fig. 24-25). Then you can hide them back if they are out of your interest. The difference between eyes and rectangles is that functions with eyes have either notes (Fig. 26) or warnings (Fig. 27) at some point of their execution, perhaps, within called functions. *Notes* describe important actions in models. *Warnings* represent places where Klever detects violations of checked requirements.

You can see that before calling module initialization and exit functions as well as module callbacks there is additional stuff in the error trace. These are parts of the environment model necessary to initialize models, to invoke module interfaces in the way the environment does and to check the final state. This tutorial does not consider models in detail, but you should keep in mind that Klever can detect faults not only directly in the source code under verification but also when checking something after execution of corresponding functions. For instance, this is the case for the considered error trace (Fig. 27).

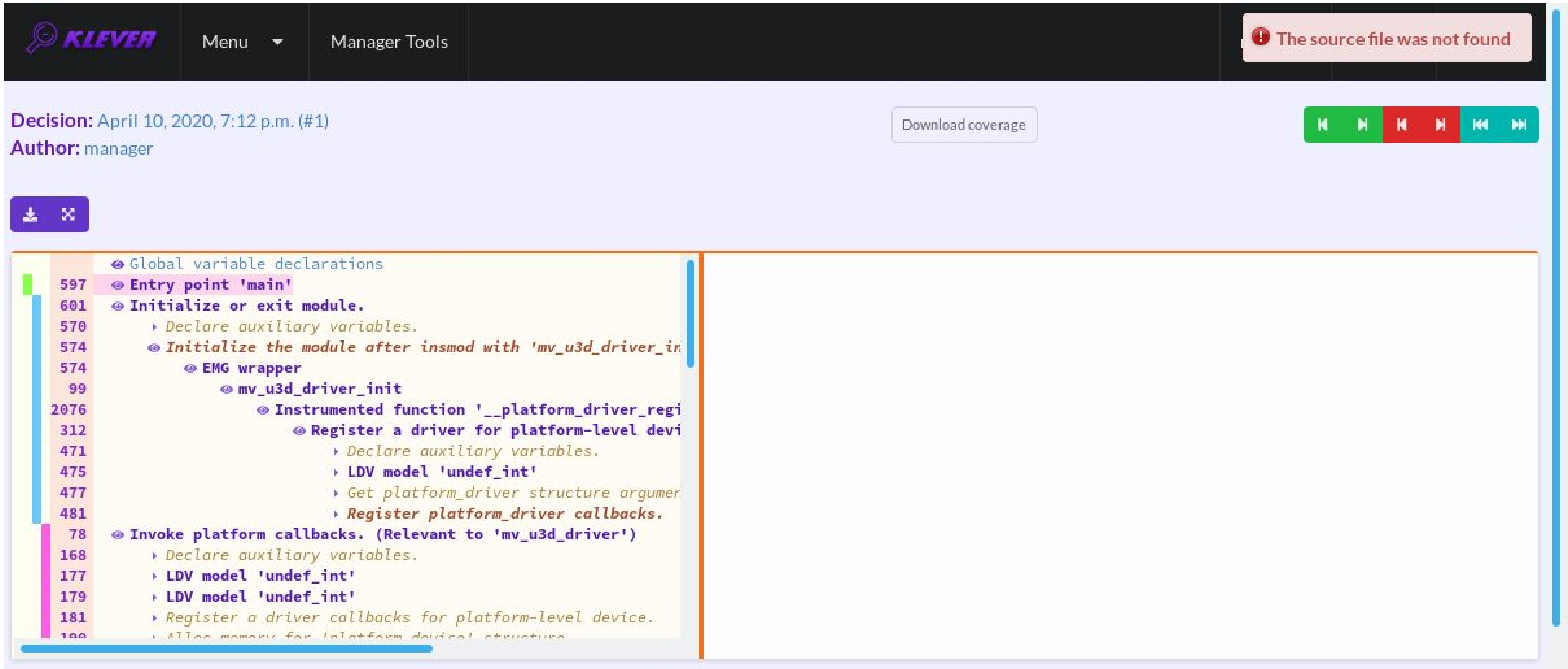
The analyzed unsafe corresponds to the fault that was fixed in commit 374a1020d21b to the Linux kernel. To finalize assessment you need to create a new *mark* (Fig. 28-30):

1. Specify a verdict (***Bug*** in our example).
2. Specify a status (***Fixed***).
3. Provide a description.
4. Save the mark.

the case, so false alarms and missing bugs take place. Each environment model is generated on the basis of specifications and it is represented as a number of additional C source files (*models*) bound with original ones through instrumentation.

After that you will be automatically redirected to the page demonstrating changes in total verdicts (Fig. 31). In our example there is the only change that corresponds to the analyzed unsafe and the new mark. But in a general case there may be many changes since the same mark can match several unsafes, and you may need to investigate these changes.

After creating the mark you can see the first manually assessed unsafe (Fig. 32). Besides, as it was already noted, you should investigate automatically assessed unsafes by analyzing corresponding error traces and marks and by (un)confirming their associations (Fig. 33-35).

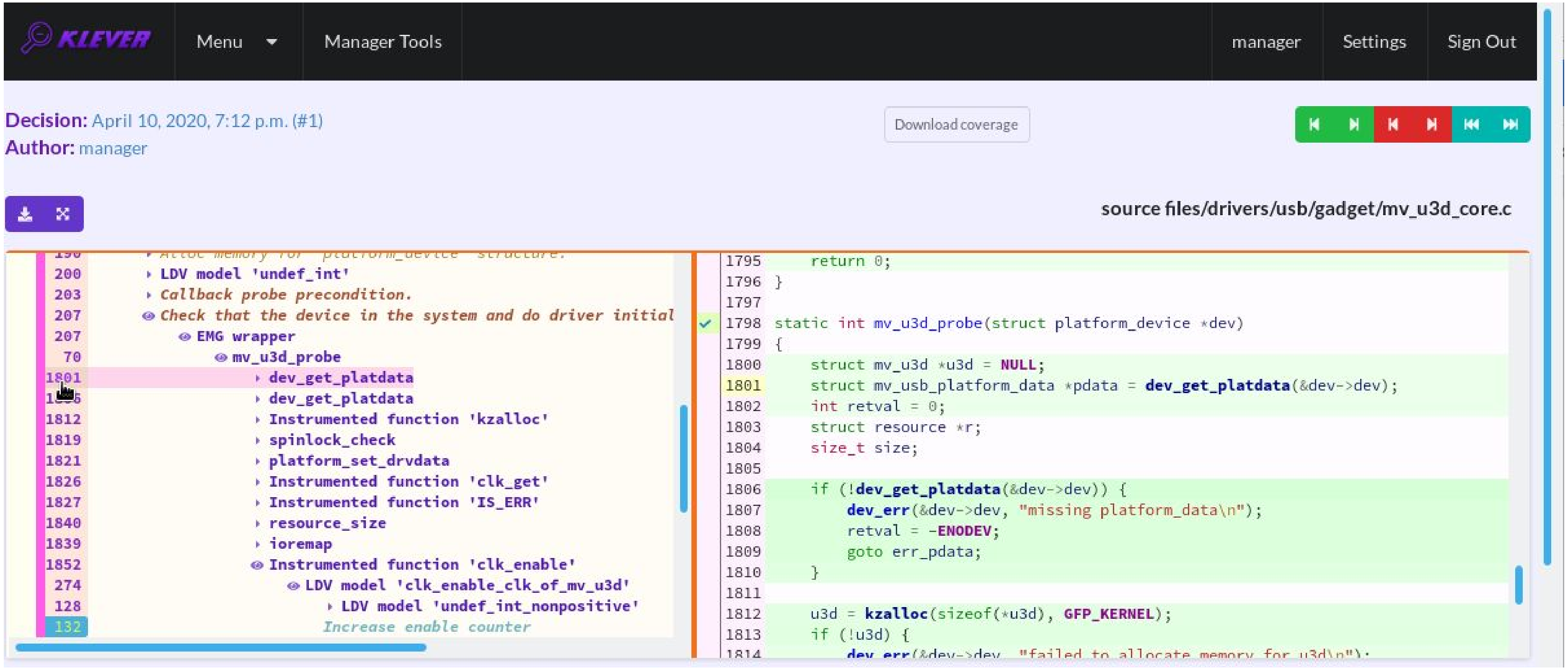


The source file was not found

Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

```
     ⊙ Global variable declarations
 597 ⊙ Entry point 'main'
 601 ⊙ Initialize or exit module.
 570     ▸ Declare auxiliary variables.
 574     ⊙ Initialize the module after insmod with 'mv_u3d_driver_in
 574         ⊙ EMG wrapper
  99             ⊙ mv_u3d_driver_init
2076                 ⊙ Instrumented function '__platform_driver_regi
 312                     ⊙ Register a driver for platform-level devi
 471                         ▸ Declare auxiliary variables.
 475                         ▸ LDV model 'undef_int'
 477                         ▸ Get platform_driver structure argumer
 481                         ▸ Register platform_driver callbacks.
  78 ⊙ Invoke platform callbacks. (Relevant to 'mv_u3d_driver')
 168     ▸ Declare auxiliary variables.
 177     ▸ LDV model 'undef_int'
 179     ▸ LDV model 'undef_int'
 181     ▸ Register a driver callbacks for platform-level device.
 190     ▸ Alloc memory for 'platform_device' structure
```

Figure 22. The error trace for module drivers/usb/gadget/mv_u3d_core.ko and requirements specification drivers:clk1



Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

source files/drivers/usb/gadget/mv_u3d_core.c

```
 196     ▸ Alloc memory for 'platform_device' structure.
 200     ▸ LDV model 'undef_int'
 203     ▸ Callback probe precondition.
 207     ⊙ Check that the device in the system and do driver initial
 207         ⊙ EMG wrapper
  70             ⊙ mv_u3d_probe
1801                 ▸ dev_get_platdata
1806                 ▸ dev_get_platdata
1812                 ▸ Instrumented function 'kzalloc'
1819                 ▸ spinlock_check
1821                 ▸ platform_set_drvdata
1826                 ▸ Instrumented function 'clk_get'
1827                 ▸ Instrumented function 'IS_ERR'
1840                 ▸ resource_size
1839                 ▸ ioremap
1852                 ⊙ Instrumented function 'clk_enable'
 274                     ⊙ LDV model 'clk_enable_clk_of_mv_u3d'
 128                         ▸ LDV model 'undef_int_nonpositive'
 132                         Increase enable counter
```

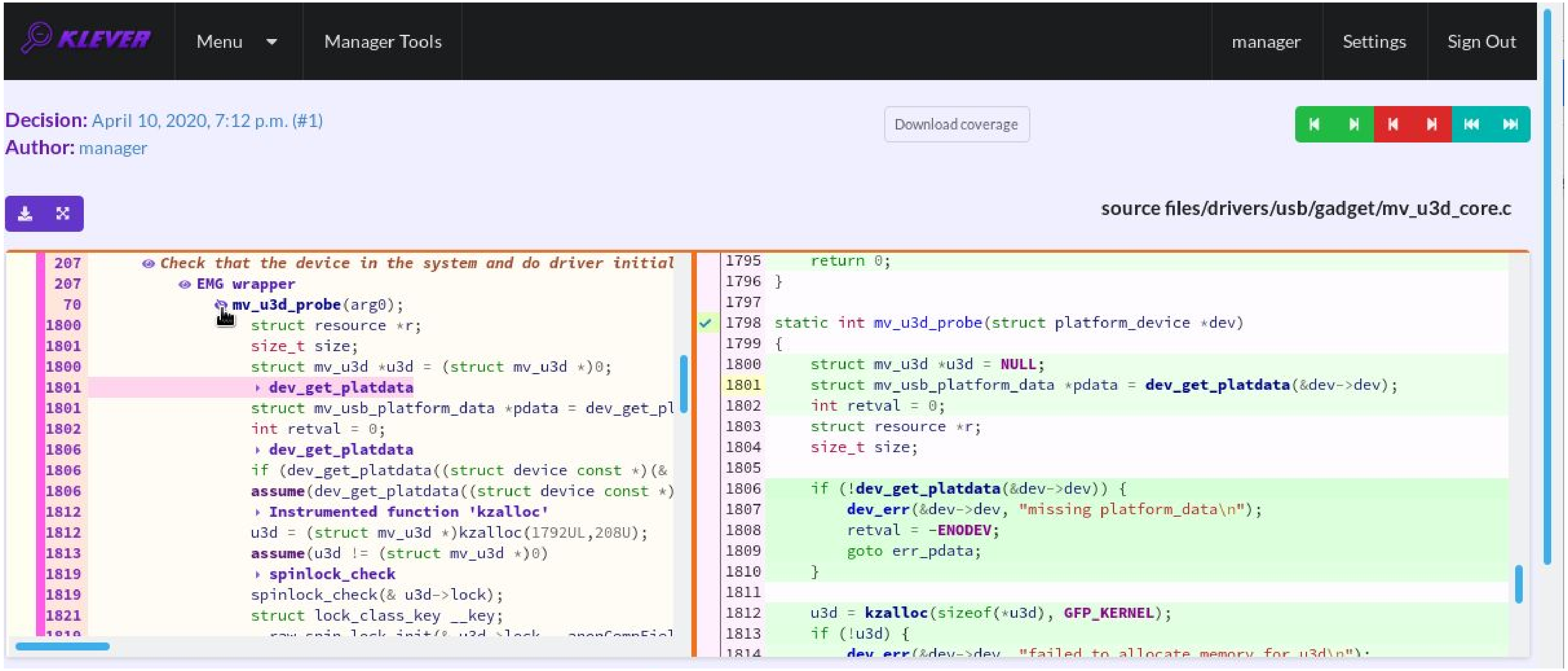
```
    return 0;
}

static int mv_u3d_probe(struct platform_device *dev)
{
    struct mv_u3d *u3d = NULL;
    struct mv_usb_platform_data *pdata = dev_get_platdata(&dev->dev);
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Figure 23. Showing the line in the original source file corresponding to the error trace statement



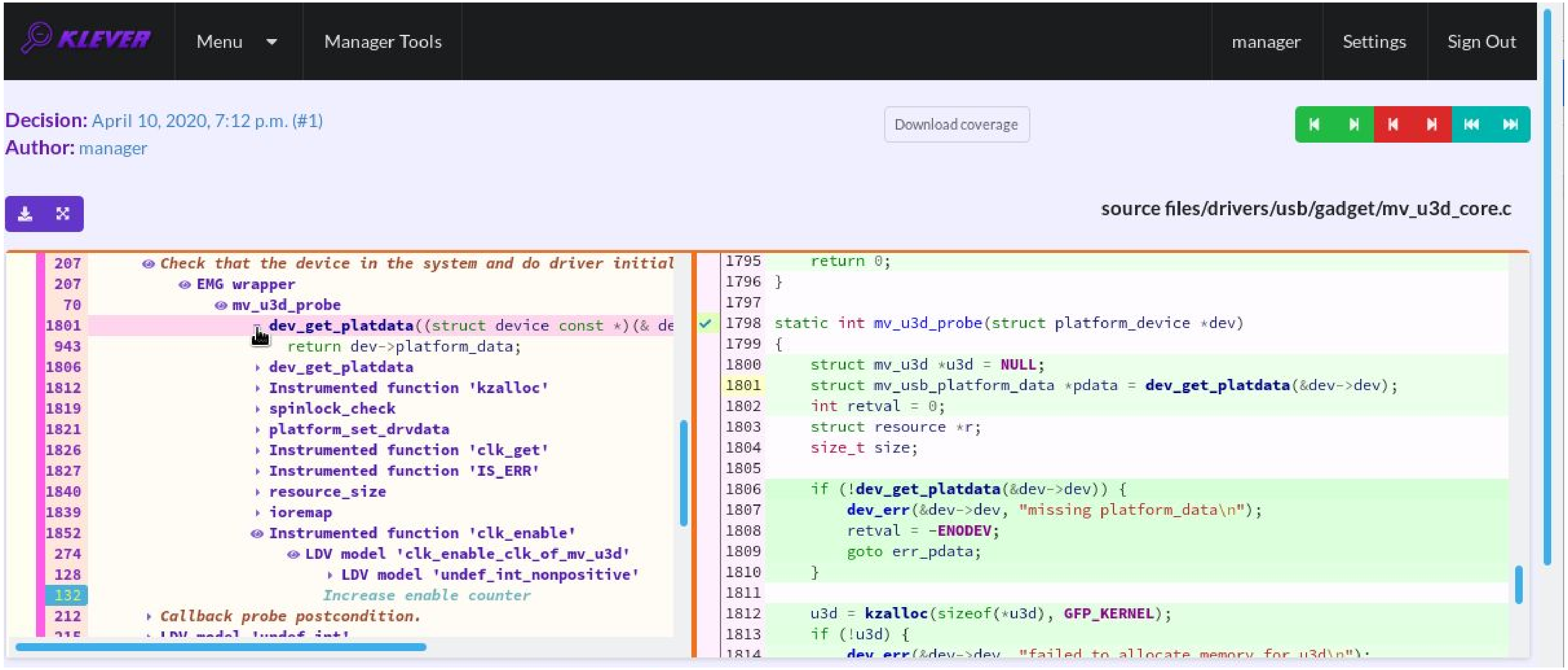
Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

source files/drivers/usb/gadget/mv_u3d_core.c

```
 207      Check that the device in the system and do driver initial
 207        EMG wrapper
  70          mv_u3d_probe(arg0);
1800            struct resource *r;
1801            size_t size;
1800            struct mv_u3d *u3d = (struct mv_u3d *)0;
1801            ‣ dev_get_platdata
1801            struct mv_usb_platform_data *pdata = dev_get_pl
1802            int retval = 0;
1806            ‣ dev_get_platdata
1806            if (dev_get_platdata((struct device const *)(&
1806            assume(dev_get_platdata((struct device const *)
1812            ‣ Instrumented function 'kzalloc'
1812            u3d = (struct mv_u3d *)kzalloc(1792UL,208U);
1813            assume(u3d != (struct mv_u3d *)0)
1819            ‣ spinlock_check
1819            spinlock_check(& u3d->lock);
1821            struct lock_class_key __key;
1819            _raw_spin_lock_init(& u3d->lock, __anonCompFiel
```

```
1795     return 0;
1796 }
1797
1798 static int mv_u3d_probe(struct platform_device *dev)
1799 {
1800     struct mv_u3d *u3d = NULL;
1801     struct mv_usb_platform_data *pdata = dev_get_platdata(&dev->dev);
1802     int retval = 0;
1803     struct resource *r;
1804     size_t size;
1805
1806     if (!dev_get_platdata(&dev->dev)) {
1807         dev_err(&dev->dev, "missing platform_data\n");
1808         retval = -ENODEV;
1809         goto err_pdata;
1810     }
1811
1812     u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
1813     if (!u3d) {
1814         dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Figure 24. Showing hidden declarations, statements and assumptions for functions with notes or warnings



Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

source files/drivers/usb/gadget/mv_u3d_core.c

```
207      ⊙ Check that the device in the system and do driver initial
207          ⊙ EMG wrapper
 70              ⊙ mv_u3d_probe
1801                 ▾ dev_get_platdata((struct device const *)(& de
 943                     return dev->platform_data;
1806                 ▸ dev_get_platdata
1812                 ▸ Instrumented function 'kzalloc'
1819                 ▸ spinlock_check
1821                 ▸ platform_set_drvdata
1826                 ▸ Instrumented function 'clk_get'
1827                 ▸ Instrumented function 'IS_ERR'
1840                 ▸ resource_size
1839                 ▸ ioremap
1852                 ⊙ Instrumented function 'clk_enable'
 274                     ⊙ LDV model 'clk_enable_clk_of_mv_u3d'
 128                         ▸ LDV model 'undef_int_nonpositive'
 132                         Increase enable counter
 212     ▸ Callback probe postcondition.
 215     ▸ LDV model 'undef_int'
```

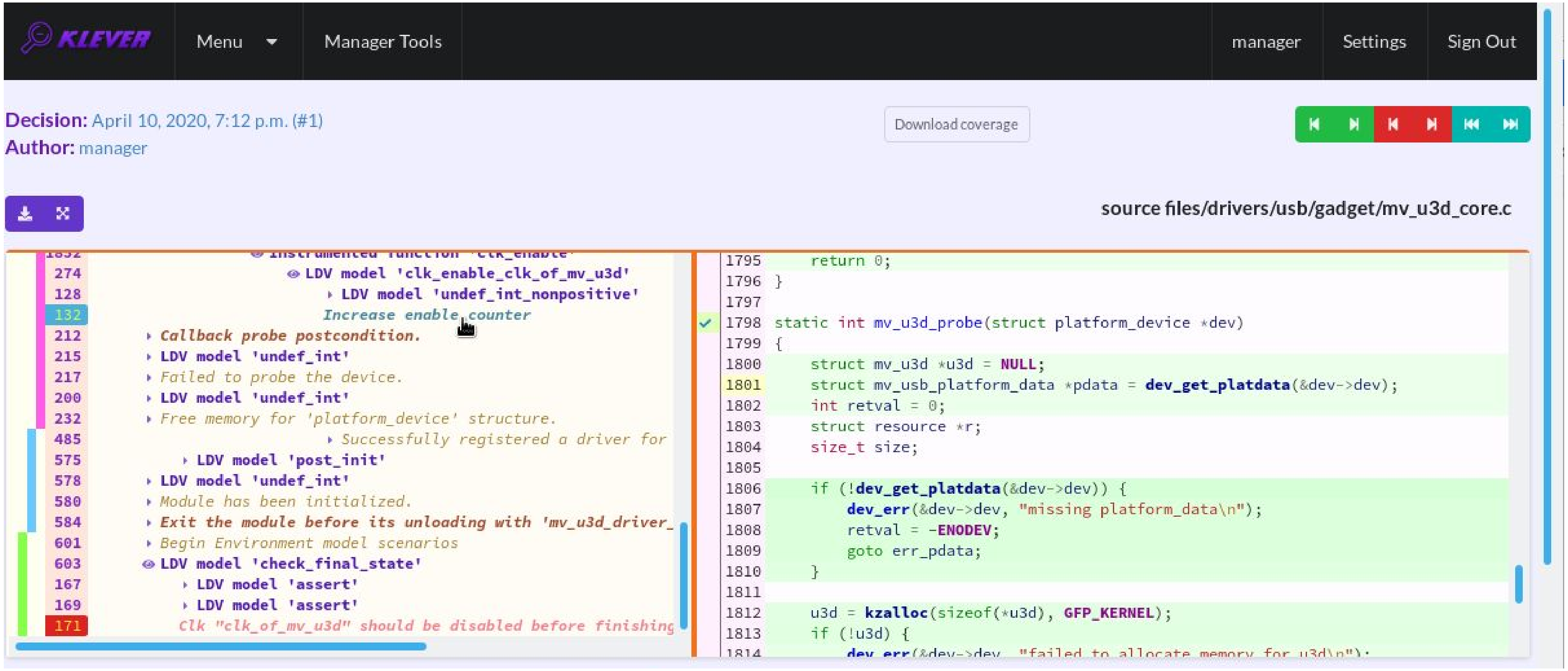
```
    return 0;
}

static int mv_u3d_probe(struct platform_device *dev)
{
    struct mv_u3d *u3d = NULL;
    struct mv_usb_platform_data *pdata = dev_get_platdata(&dev->dev);
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Figure 25. Showing hidden declarations, statements and assumptions for functions without notes and warnings



Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

source files/drivers/usb/gadget/mv_u3d_core.c

```
1832              Instrumented function 'clk_enable'
274                  LDV model 'clk_enable_clk_of_mv_u3d'
128                     LDV model 'undef_int_nonpositive'
132                    Increase enable counter
212       Callback probe postcondition.
215       LDV model 'undef_int'
217       Failed to probe the device.
200       LDV model 'undef_int'
232       Free memory for 'platform_device' structure.
485                     Successfully registered a driver for
575          LDV model 'post_init'
578       LDV model 'undef_int'
580       Module has been initialized.
584       Exit the module before its unloading with 'mv_u3d_driver_
601       Begin Environment model scenarios
603       LDV model 'check_final_state'
167          LDV model 'assert'
169          LDV model 'assert'
171          Clk "clk_of_mv_u3d" should be disabled before finishing
```

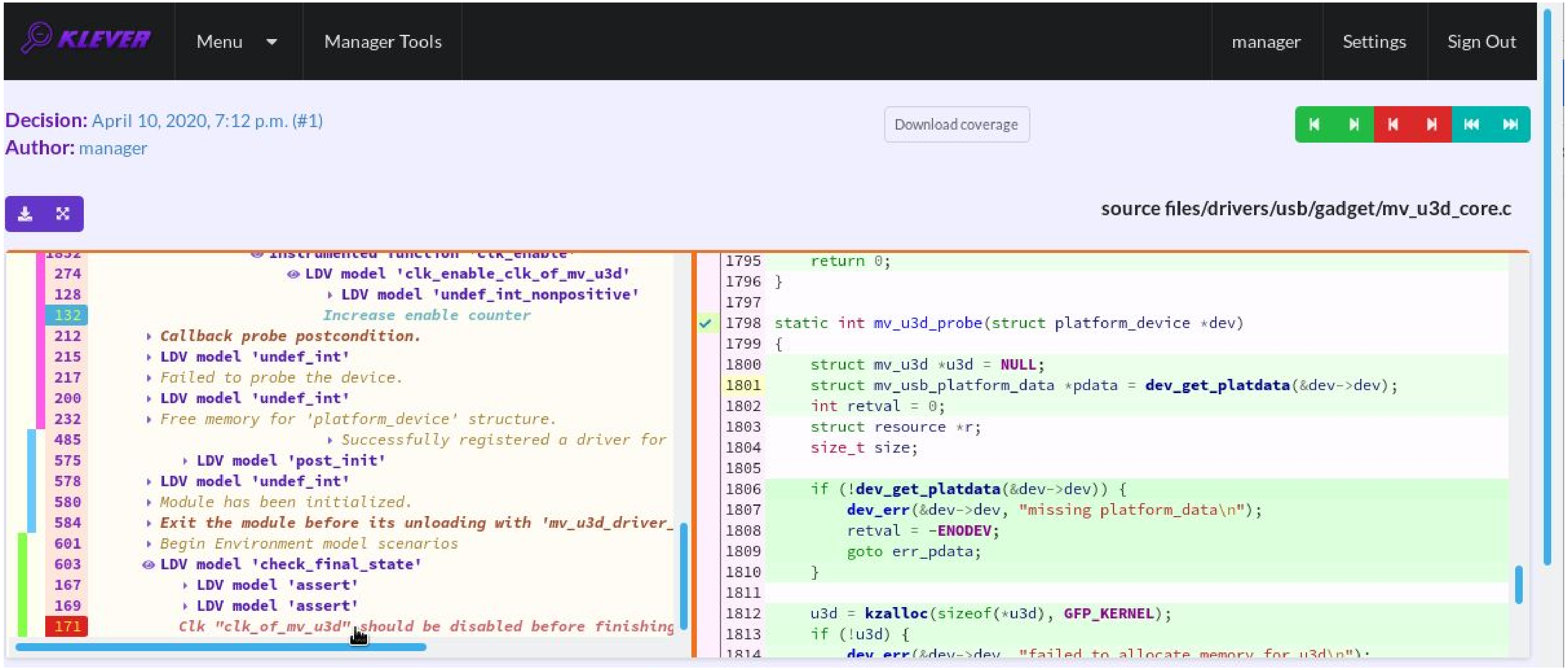
```
    return 0;
}

static int mv_u3d_probe(struct platform_device *dev)
{
    struct mv_u3d *u3d = NULL;
    struct mv_usb_platform_data *pdata = dev_get_platdata(&dev->dev);
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Figure 26. The error trace note



Decision: April 10, 2020, 7:12 p.m. (#1)
Author: manager

source files/drivers/usb/gadget/mv_u3d_core.c

```
1832                   Instrumented function 'clk_enable'
274                       LDV model 'clk_enable_clk_of_mv_u3d'
128                           LDV model 'undef_int_nonpositive'
132                         Increase enable counter
212       Callback probe postcondition.
215       LDV model 'undef_int'
217       Failed to probe the device.
200       LDV model 'undef_int'
232       Free memory for 'platform_device' structure.
485                           Successfully registered a driver for
575           LDV model 'post_init'
578       LDV model 'undef_int'
580       Module has been initialized.
584       Exit the module before its unloading with 'mv_u3d_driver_
601       Begin Environment model scenarios
603       LDV model 'check_final_state'
167           LDV model 'assert'
169           LDV model 'assert'
171         Clk "clk_of_mv_u3d" should be disabled before finishing
```

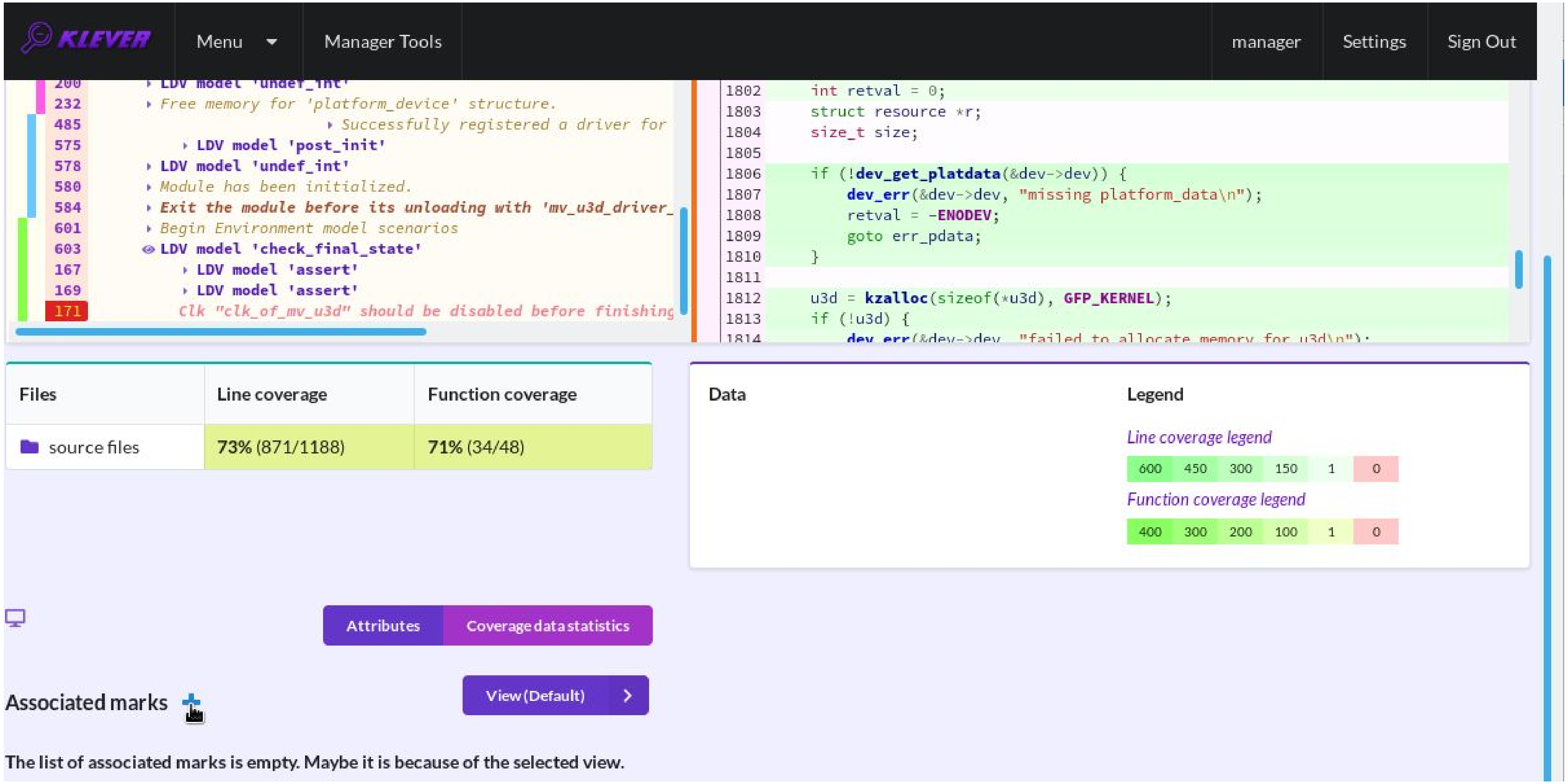
```
    return 0;
}

static int mv_u3d_probe(struct platform_device *dev)
{
    struct mv_u3d *u3d = NULL;
    struct mv_usb_platform_data *pdata = dev_get_platdata(&dev->dev);
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Figure 27. The error trace warning



```
200      ‣ LDV model 'undef_int'
232      ‣ Free memory for 'platform_device' structure.
485                            ‣ Successfully registered a driver for
575          ‣ LDV model 'post_init'
578      ‣ LDV model 'undef_int'
580      ‣ Module has been initialized.
584      ‣ Exit the module before its unloading with 'mv_u3d_driver_
601      ‣ Begin Environment model scenarios
603      ◉ LDV model 'check_final_state'
167          ‣ LDV model 'assert'
169          ‣ LDV model 'assert'
171            Clk "clk_of_mv_u3d" should be disabled before finishing
```

```
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Files	Line coverage	Function coverage
source files	73% (871/1188)	71% (34/48)

Data

Legend

Line coverage legend

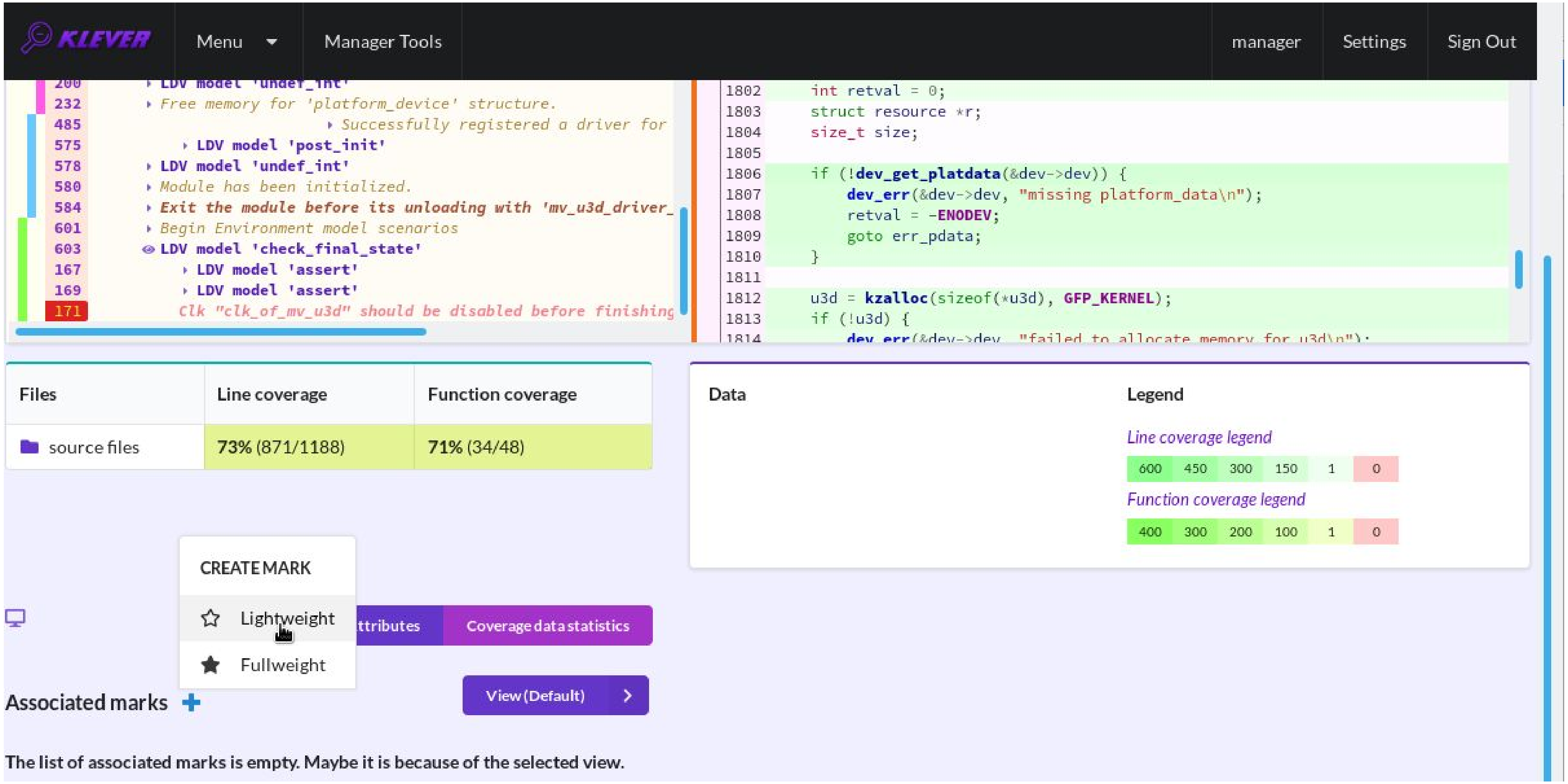
600	450	300	150	1	0

Function coverage legend

400	300	200	100	1	0

The list of associated marks is empty. Maybe it is because of the selected view.

Figure 28. Starting the creation of a new mark



```
200        ▸ LDV model 'undef_int'
232        ▸ Free memory for 'platform_device' structure.
485                             ▸ Successfully registered a driver for
575            ▸ LDV model 'post_init'
578        ▸ LDV model 'undef_int'
580        ▸ Module has been initialized.
584        ▸ Exit the module before its unloading with 'mv_u3d_driver_
601        ▸ Begin Environment model scenarios
603        LDV model 'check_final_state'
167            ▸ LDV model 'assert'
169            ▸ LDV model 'assert'
171              Clk "clk_of_mv_u3d" should be disabled before finishing
```

```
    int retval = 0;
    struct resource *r;
    size_t size;

    if (!dev_get_platdata(&dev->dev)) {
        dev_err(&dev->dev, "missing platform_data\n");
        retval = -ENODEV;
        goto err_pdata;
    }

    u3d = kzalloc(sizeof(*u3d), GFP_KERNEL);
    if (!u3d) {
        dev_err(&dev->dev, "failed to allocate memory for u3d\n");
```

Files	Line coverage	Function coverage
source files	**73%** (871/1188)	**71%** (34/48)

Data

Legend

Line coverage legend

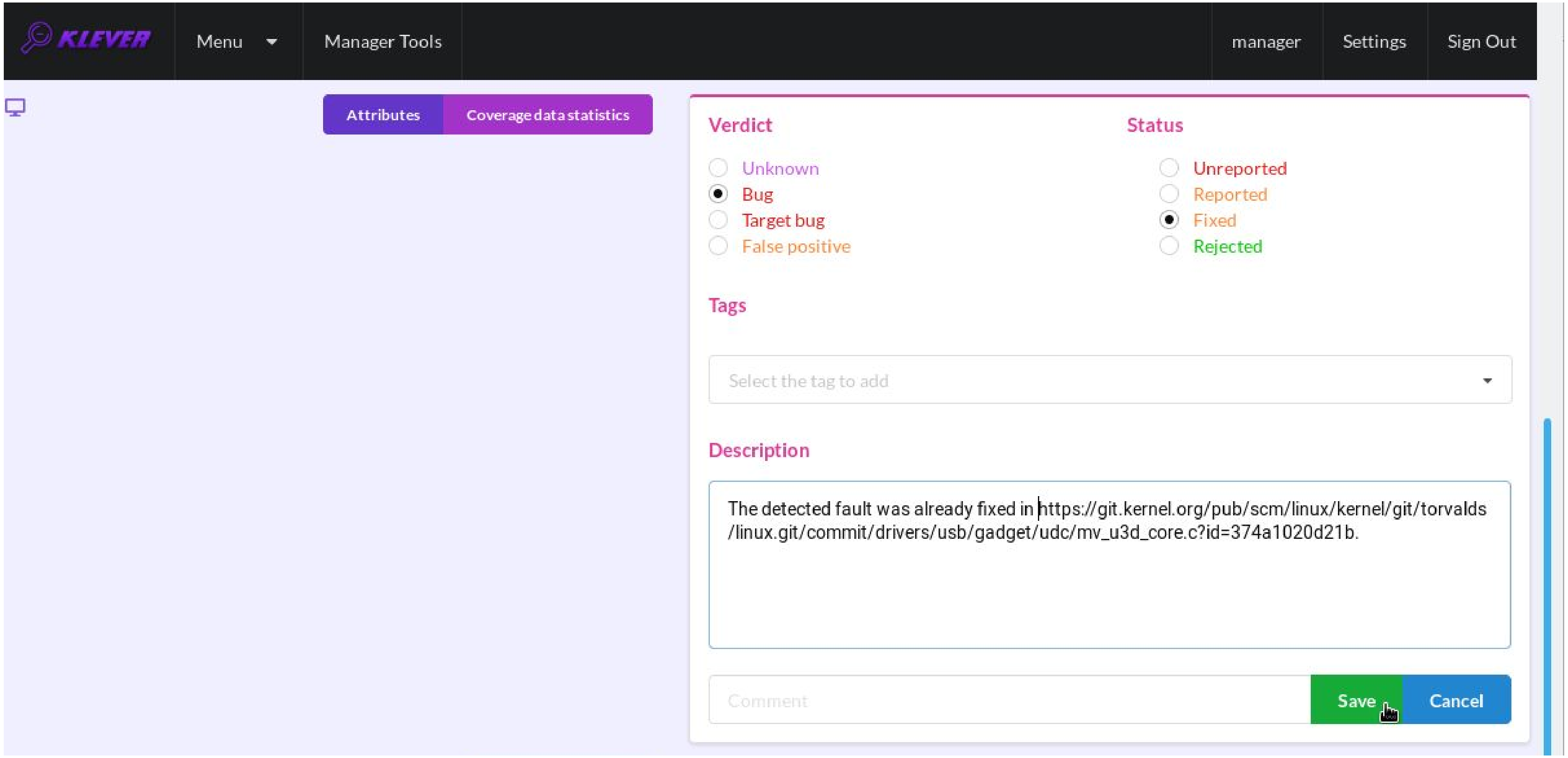
600	450	300	150	1	0

Function coverage legend

400	300	200	100	1	0

The list of associated marks is empty. Maybe it is because of the selected view.

Figure 29. Starting the creation of a new lightweight mark



Verdict

○ Unknown
● Bug
○ Target bug
○ False positive

Status

○ Unreported
○ Reported
● Fixed
○ Rejected

Tags

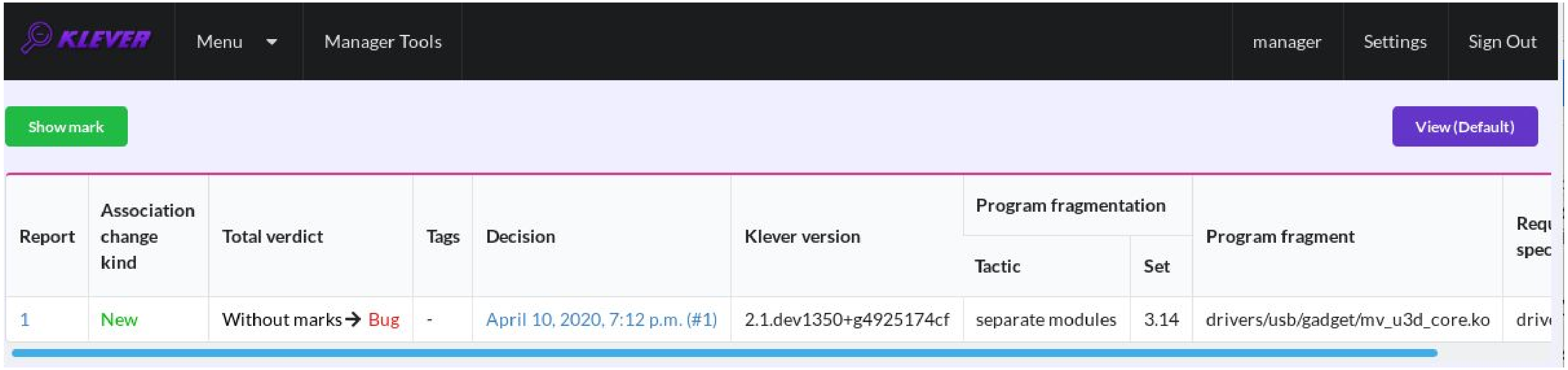
Select the tag to add

Description

The detected fault was already fixed in https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/drivers/usb/gadget/udc/mv_u3d_core.c?id=374a1020d21b.

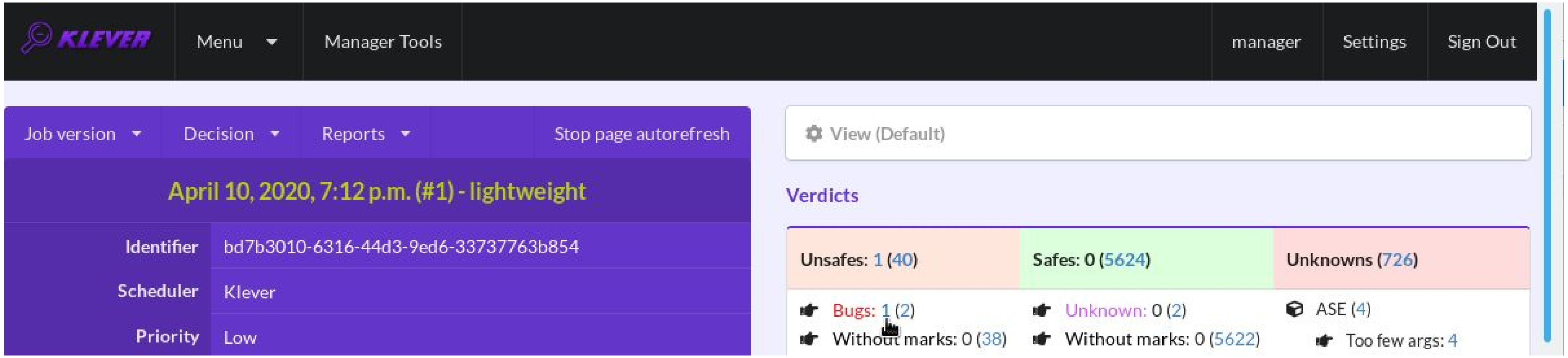
Comment

Figure 30. The creation of the new lightweight mark



Report	Association change kind	Total verdict	Tags	Decision	Klever version	Program fragmentation		Program fragment	Requ spec
						Tactic	Set		
1	New	Without marks→ Bug	-	April 10, 2020, 7:12 p.m. (#1)	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/usb/gadget/mv_u3d_core.ko	driv

Figure 31. Changes in total verdicts



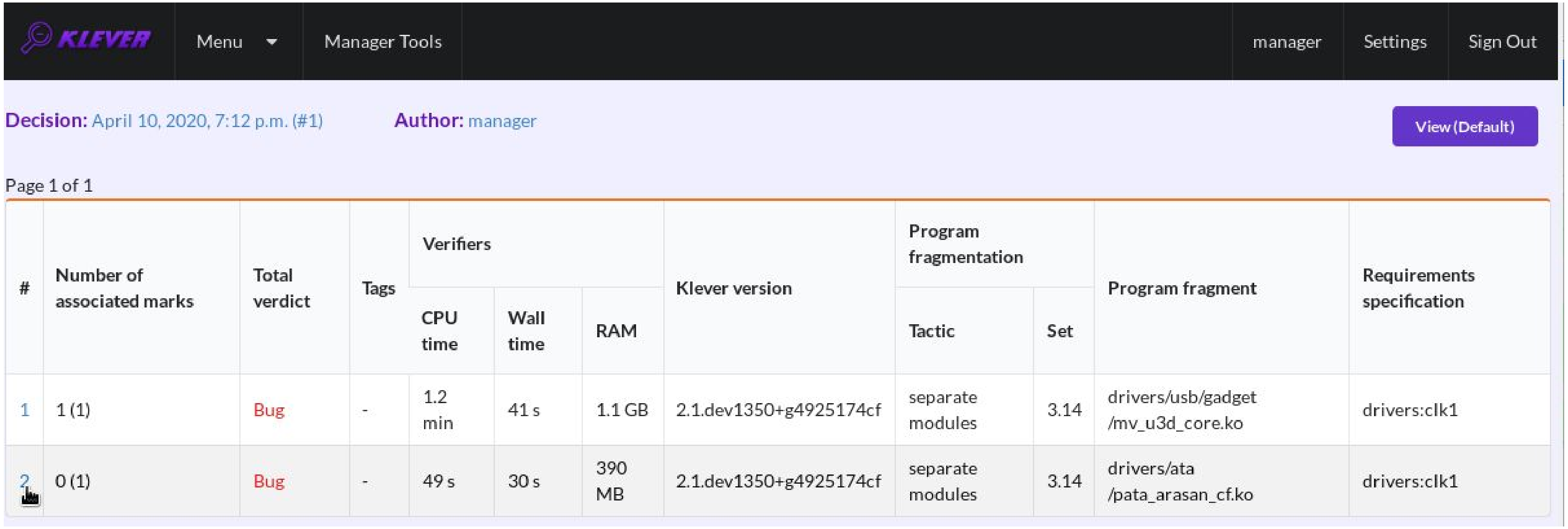
April 10, 2020, 7:12 p.m. (#1) - lightweight

Identifier	bd7b3010-6316-44d3-9ed6-33737763b854
Scheduler	Klever
Priority	Low

Verdicts

Unsafes: 1 (40)	Safes: 0 (5624)	Unknowns (726)
Bugs: 1 (2)	Unknown: 0 (2)	ASE (4)
Without marks: 0 (38)	Without marks: 0 (5622)	Too few args: 4

Figure 32. The total number of manually assessed unsafes

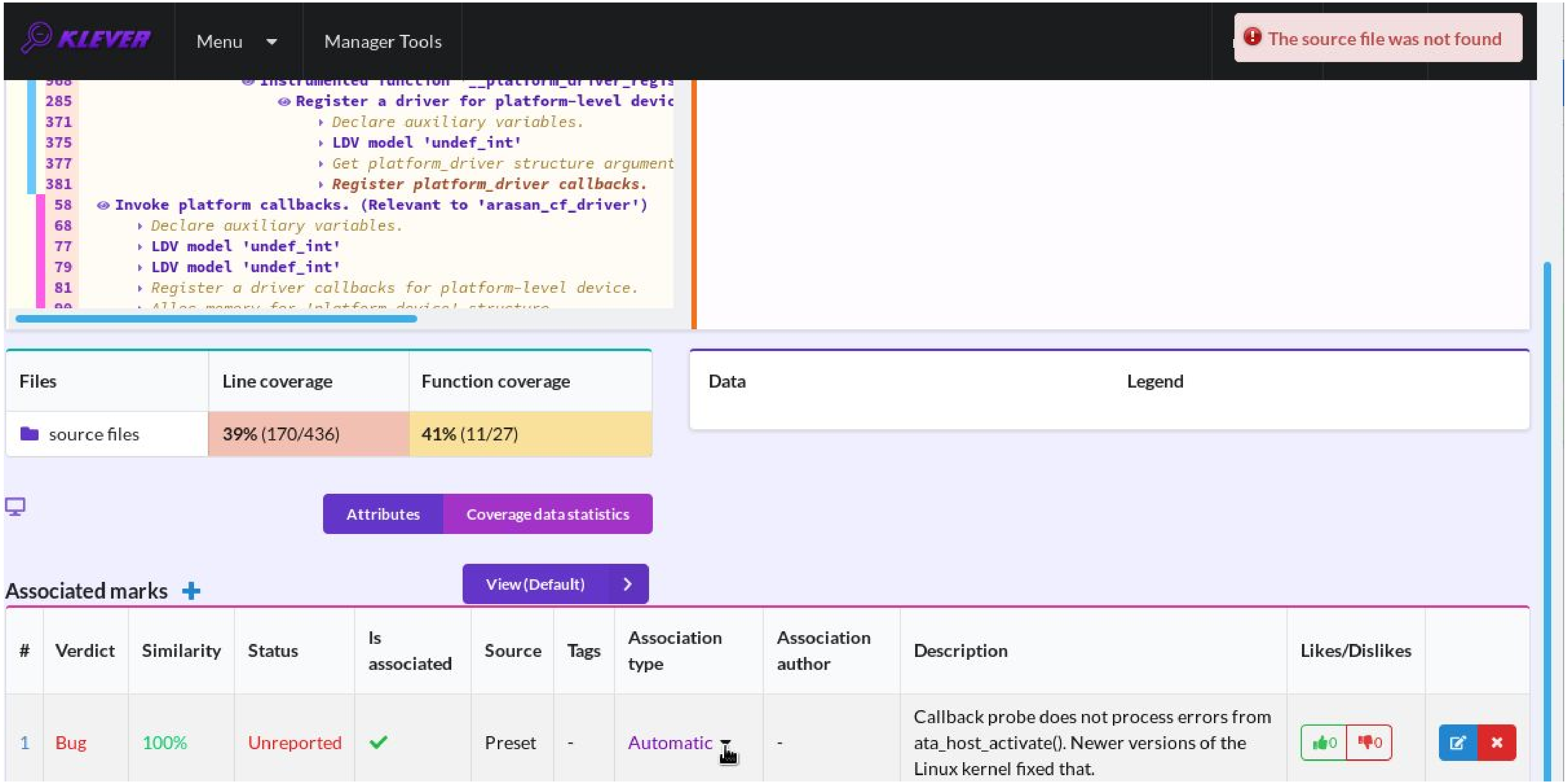


Decision: April 10, 2020, 7:12 p.m. (#1) **Author:** manager

Page 1 of 1

#	Number of associated marks	Total verdict	Tags	Verifiers			Klever version	Program fragmentation		Program fragment	Requirements specification
				CPU time	Wall time	RAM		Tactic	Set		
1	1 (1)	Bug	-	1.2 min	41 s	1.1 GB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/usb/gadget /mv_u3d_core.ko	drivers:clk1
2	0 (1)	Bug	-	49 s	30 s	390 MB	2.1.dev1350+g4925174cf	separate modules	3.14	drivers/ata /pata_arasan_cf.ko	drivers:clk1

Figure 33. Opening the error trace of the unsafe with automatic assessment



The source file was not found

```
285            Register a driver for platform-level devic
371                Declare auxiliary variables.
375                LDV model 'undef_int'
377                Get platform_driver structure argument
381                Register platform_driver callbacks.
58   Invoke platform callbacks. (Relevant to 'arasan_cf_driver')
68       Declare auxiliary variables.
77       LDV model 'undef_int'
79       LDV model 'undef_int'
81       Register a driver callbacks for platform-level device.
```

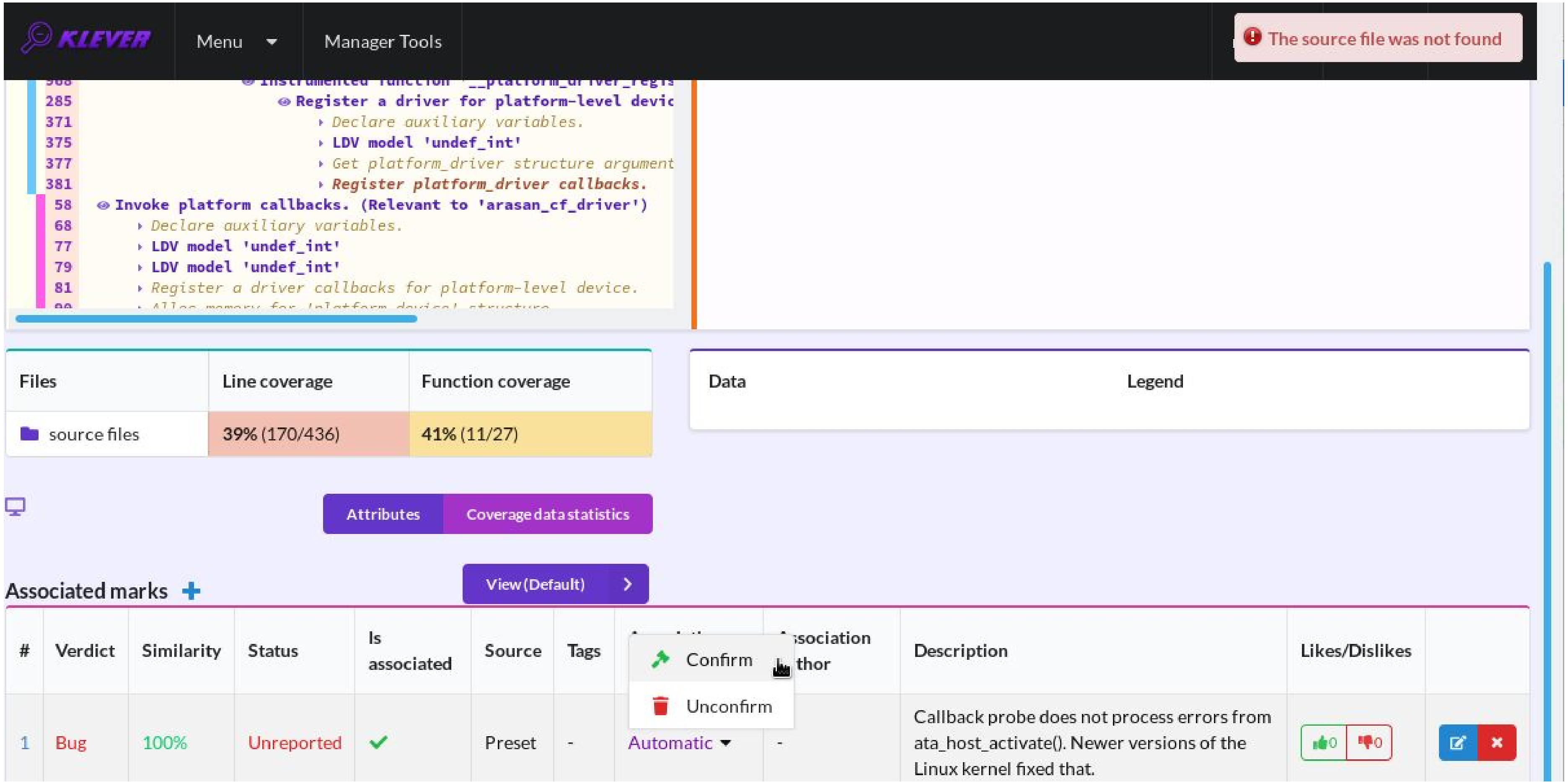
Files	Line coverage	Function coverage
source files	**39%** (170/436)	**41%** (11/27)

Data	Legend

Associated marks

#	Verdict	Similarity	Status	Is associated	Source	Tags	Association type	Association author	Description	Likes/Dislikes	
1	Bug	100%	Unreported	✓	Preset	-	Automatic	-	Callback probe does not process errors from ata_host_activate(). Newer versions of the Linux kernel fixed that.	👍0 👎0	

Figure 34. Starting changing the association type



```
568            Instrumented function '__platform_driver_regis
285              Register a driver for platform-level devic
371                 Declare auxiliary variables.
375                 LDV model 'undef_int'
377                 Get platform_driver structure argument
381                 Register platform_driver callbacks.
 58  Invoke platform callbacks. (Relevant to 'arasan_cf_driver')
 68     Declare auxiliary variables.
 77     LDV model 'undef_int'
 79     LDV model 'undef_int'
 81     Register a driver callbacks for platform-level device.
 90     Alloc memory for 'platform_device' structure
```

Files	Line coverage	Function coverage
source files	39% (170/436)	41% (11/27)

Data	Legend

Associated marks

#	Verdict	Similarity	Status	Is associated	Source	Tags	[illegible]	ssociation thor	Description	Likes/Dislikes	
1	Bug	100%	Unreported	✓	Preset	-	Automatic	-	Callback probe does not process errors from ata_host_activate(). Newer versions of the Linux kernel fixed that.	0 0	

Figure 35. Confirming the automatic association

What’s Next?

We assume that you can be non-satisfied fully with a quality of obtained verification results. Perhaps, you even could not obtain them at all. This is expected since Klever is an open source software developed in the Academy and we support verification of Linux kernel loadable modules for evaluation purposes primarily. Besides, this tutorial misses many tricky activities like development of specifications and support for verification of additional software. We are ready to discuss different issues and even to fix some crucial bugs, but we do not have the manpower to make any considerable improvements for you for free.