eBPF for Python Troubleshooting

2024-11-23

PYCHINA
Python 中国社区

- Cilium datapath tech lead
- Started my career from Python
- Debug for fun

Let’s debug, in incident mode

1. CVE-2024-27351
2. Django ticket #27690
3. BentoML issue #4760

Don’ts
1. No time to read the whole code
2. No idea about impl details
3. Non-disruptive required

Dos
1. Observe symptoms
2. Collect metrics
3. Non-disruptive debuggers

Notes:

- Containerized CPython 3.10 (docker run python:3.10)
- GNU/Linux 6.8.0 (Ubuntu 22.04.1)
- x86_64, Little Endian
- CONFIG_BPF*=y, CONFIG_KPROBE*=y (cat /boot/config-$(uname -r))
- bpftrace v0.20.3

CVE-2024-27351

Symptoms

1. Django HTTP server suddenly doesn’t serve (`curl localhost:8000 hangs/timeout/slow`)
2. Django Python process consumes CPU (`top`)

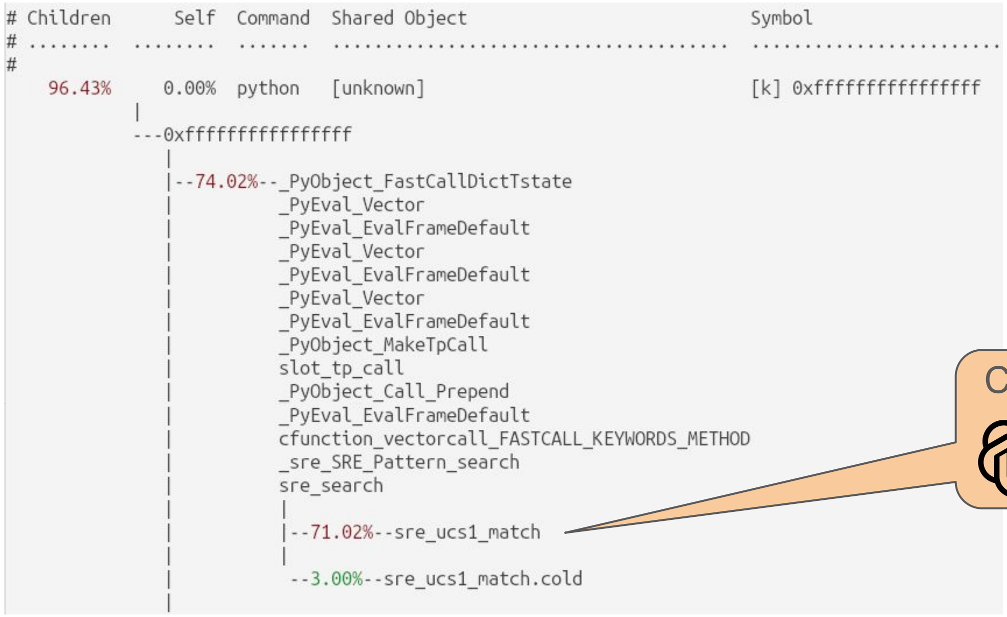
Random guesses:

- App bugs: infinite loop?
- Bad optimization: deep recursion?
- No bug: too many requests?

Idea: Find the Django code responsible for CPU spike, likely that’s the cause of slow response.

perf

```
$ perf record -F99 -g -p $PID --call-graph dwarf,16384
$ perf report -g
```



```
# Children      Self  Command  Shared Object                         Symbol
# ........  ........  .......  ....................................  ........................
#
    96.43%     0.00%  python   [unknown]                             [k] 0xffffffffffffffff
            |
            ---0xffffffffffffffff
               |
               |--74.02%--_PyObject_FastCallDictTstate
               |          _PyEval_Vector
               |          _PyEval_EvalFrameDefault
               |          _PyEval_Vector
               |          _PyEval_EvalFrameDefault
               |          _PyEval_Vector
               |          _PyEval_EvalFrameDefault
               |          _PyObject_MakeTpCall
               |          slot_tp_call
               |          _PyObject_Call_Prepend
               |          _PyEval_EvalFrameDefault
               |          cfunction_vectorcall_FASTCALL_KEYWORDS_METHOD
               |          _sre_SRE_Pattern_search
               |          sre_search
               |          |
               |          |--71.02%--sre_ucs1_match
               |          |
               |           --3.00%--sre_ucs1_match.cold
               |
```

ChatGPT, what's sre_ucs1_match in Python?

: "an internal function in `re` module"

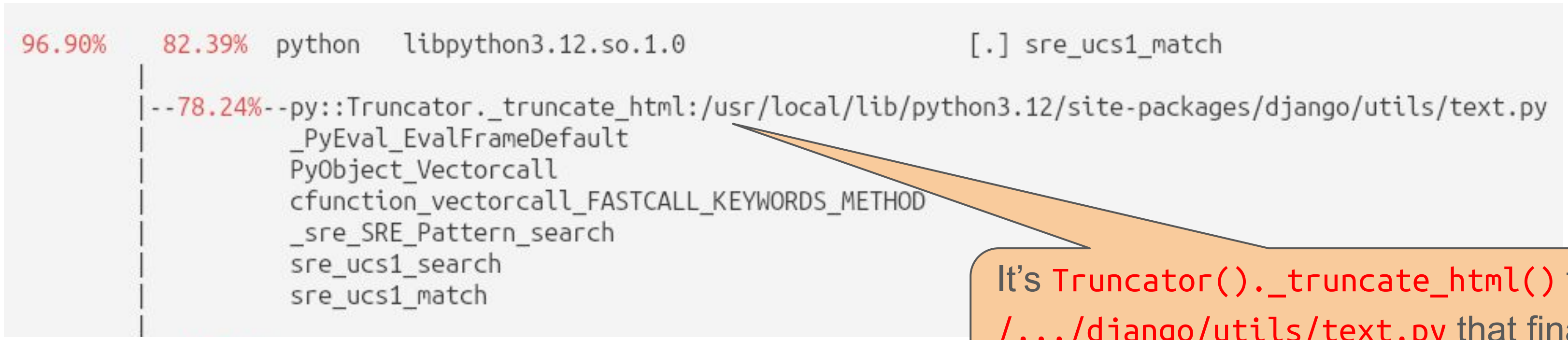
Looks like ReDoS (Regular expression Denial of Service)! But which Python code called what regex?

Dark side of perf (In the context of CPython)

1. Output C functions rather than Python functions
2. Requirements in how CPython is built
   a. `-fno-omit-frame-pointer:` with or without frame pointer (affecting algorithm and overhead)
   b. `-s`: with or without symbols (affecting address-symbol mapping)

python3.12+ -X perf

```
$ python3.12 -X perf manage.py runserver
$ perf record -F99 -g -p $PID --call-graph dwarf,16384
$ perf report -g
```



```
96.90%    82.39%  python   libpython3.12.so.1.0                [.] sre_ucs1_match
                |
                |--78.24%--py::Truncator._truncate_html:/usr/local/lib/python3.12/site-packages/django/utils/text.py
                |          _PyEval_EvalFrameDefault
                |          PyObject_Vectorcall
                |          cfunction_vectorcall_FASTCALL_KEYWORDS_METHOD
                |          _sre_SRE_Pattern_search
                |          sre_ucs1_search
                |          sre_ucs1_match
                |
```

It's `Truncator()._truncate_html()` from `/.../django/utils/text.py` that finally calls `sre_ucs1_match`.

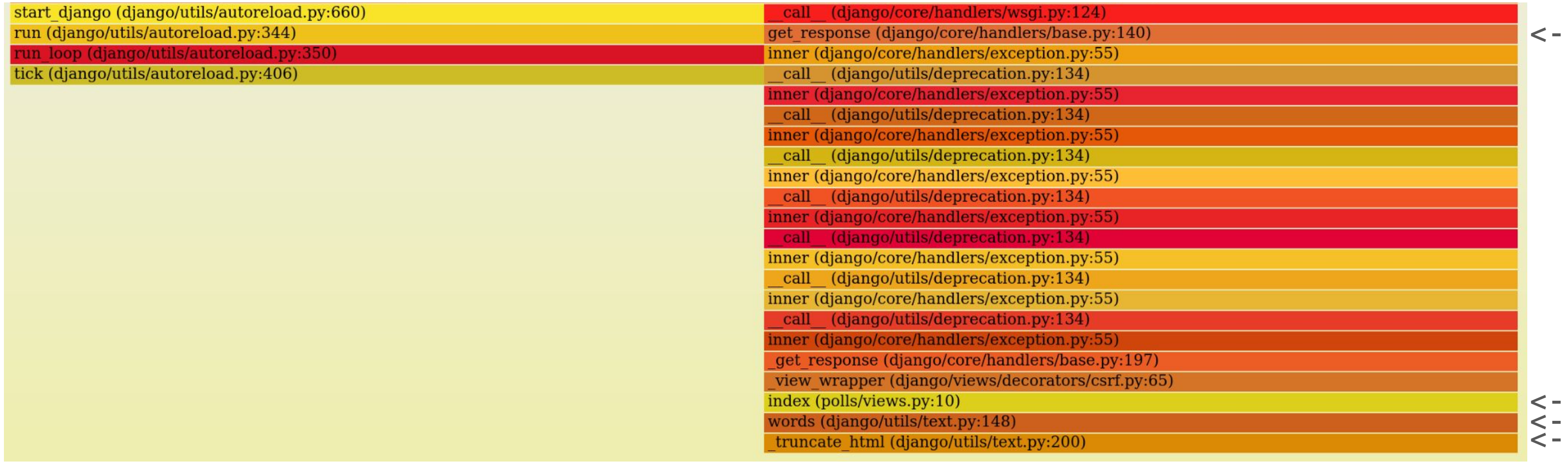
But who called the _truncate_html()? Can we see the whole Python call chain?

Dark side of -X perf

1. Official images not built with -fno-omit-frame-pointer
    a. Can't get full Python function call chain
    b. `python3.13+ -X perf_jit` (kernel>=6.7.3, higher overhead)
2. Performance overhead
    a. Requests/second: 142.63 → 123.90, -13% (`ab -n10000 localhost:8000/`)
    b. Extreme case: -56% (recursive fib)
3. Require process restart to specify -X perf
    a. Prod env doesn't allow
    b. Issues may be gone after restart
    c. Why don't `python -m cProfile`?

py-spy

```
$ py-spy record -p $PID
```



```
start_django (django/utils/autoreload.py:660)   __call__ (django/core/handlers/wsgi.py:124)
run (django/utils/autoreload.py:344)            get_response (django/core/handlers/base.py:140)           <-
run_loop (django/utils/autoreload.py:350)       inner (django/core/handlers/exception.py:55)
tick (django/utils/autoreload.py:406)           __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                __call__ (django/utils/deprecation.py:134)
                                                inner (django/core/handlers/exception.py:55)
                                                _get_response (django/core/handlers/base.py:197)
                                                _view_wrapper (django/views/decorators/csrf.py:65)
                                                index (polls/views.py:10)                                  <-
                                                words (django/utils/text.py:148)                           <-
                                                _truncate_html (django/utils/text.py:200)                  <-
```

```
-> get_response[django]
-> index (polls/views.py:10)
-> words (django/utils/text.py:148)
-> _truncate_html (django/utils/text.py:200)
```

Root cause: `index()` in `polls/views.py:10` finally called `_truncate_html()` in Django, which has a regex search that triggered ReDoS.

Dark side of py-spy

1. Can't distinguish off-CPU from on-CPU
    a. Misleading plateaus
2. Performance overhead
    a. Requests/second: 170.94 → 134.11, -21% (`ab -n10000 localhost:8000/`)
    b. `py-spy --nonblocking`: 170.94 → 164.83, -3% (but may lead to inaccurate results)

	Disruptive	Requirements in Python build flags	Overhead	Accuracy
`perf`	No	Yes	Low (sampling)	No Python stack
`perf + python -X perf`	Yes	Yes	Moderate (non-sampling)	One layer of Python stack
`py-spy`	No	No	High (sampling)	Possible misleading result

eBPF: speedrun

Like a breakpoint, but more flexible, more efficient, more powerful

```
$ gdb -p 88991
(gdb) break sre_ucs1_match
Breakpoint 1 at 0x7ee02c2d3a70: file ./Modules/sre_lib.h, line 550.
(gdb) c
Continuing.

Thread 3 "python" hit Breakpoint 1, sre_ucs1_match (state=0x7ee0291fa2f0, pattern=0x7ee02953852c,
toplevel=0) at ./Modules/sre_lib.h:550
(gdb) backtrace
```

```
$ bpftrace -e 'uprobe:/proc/88991/root/usr/local/lib/libpython3.10.so.1.0:sre_ucs1_match
/pid == 88991/ {print(ustack)}'
```

[1] target pid (filter)
[2] breakpoint (event hook)
[3] commands to execute at breakpoint (callback)

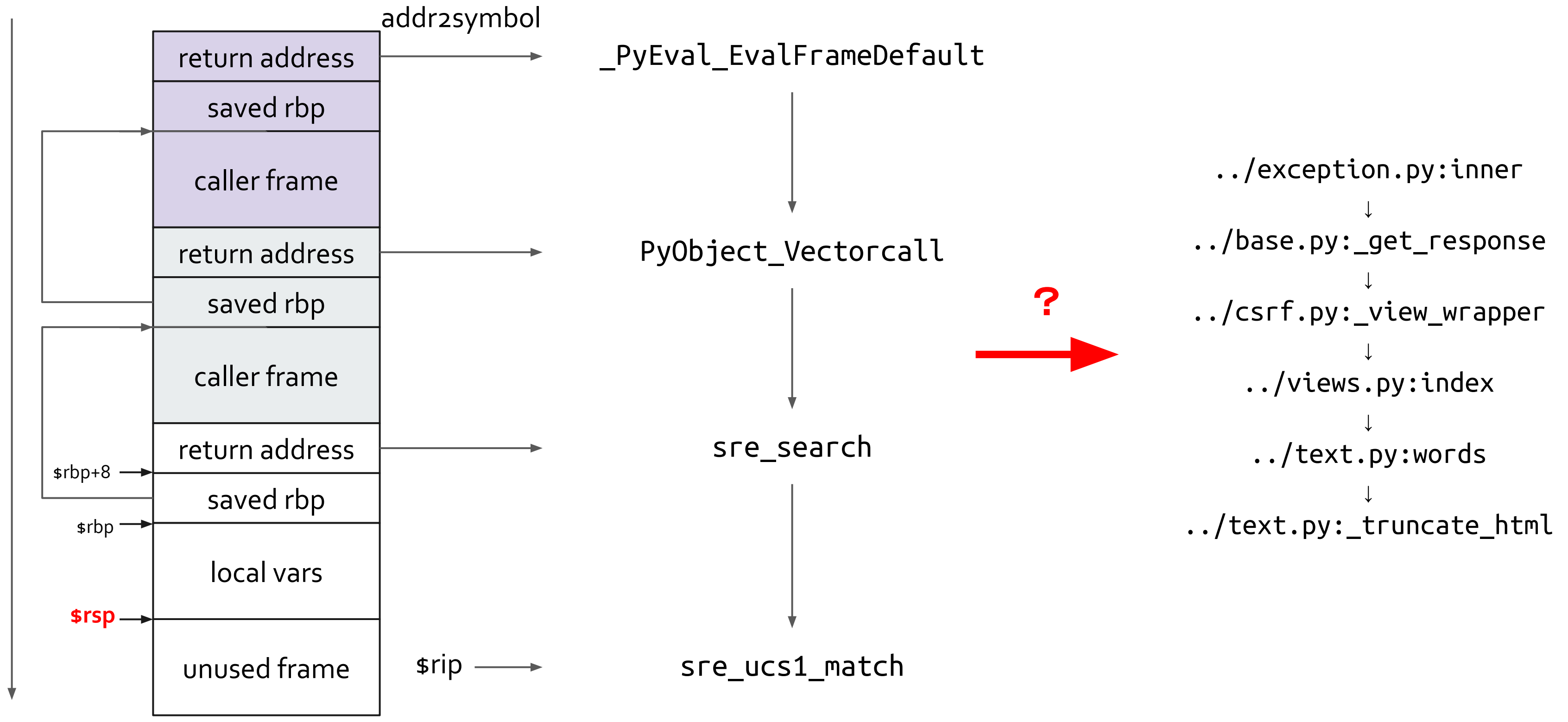
eBPF for CPython perf: Vision

1. Event hook: on-CPU samples
2. Callback: an eBPF program to unwind stack in Python VM level (HOW?)
3. Filter: target Python pid

```
/.../autoreload.py:tick
/.../autoreload.py:run_loop
/.../autoreload.py:start_django
/.../manage.py:main
/.../manage.py:<module>: 1

/.../selectors.py:select
/.../socketserver.py:serve_forever
/.../threading.py:_bootstrap_inner
/.../threading.py:_bootstrap: 2

/.../text.py:_truncate_html
/.../text.py:words
/.../views.py:index
/.../deprecation.py:__call__
/.../exception.py:inner: 260
```

This stack was seen 260 times during on-CPU sampling! Gotcha!

Technical Hurdles



PyFrameObject: an abstract of Python Frame

```
PyObject *_PyEval_EvalFrameDefault(PyThreadState *,PyFrameObject *, int);
```

```
(gdb) ptype/o PyFrameObject
type = struct _frame {
/*     0      |      24 */  PyVarObject ob_base;
/*    24      |       8 */  struct _frame *f_back;
/*    32      |       8 */  PyCodeObject *f_code;
/*    40      |       8 */  PyObject *f_builtins;
/*    48      |       8 */  PyObject *f_globals;
/*    56      |       8 */  PyObject *f_locals;
/*    64      |       8 */  PyObject **f_valuestack;
/*    72      |       8 */  PyObject *f_trace;
/*    80      |       4 */  int f_stackdepth;
/*    84      |       1 */  char f_trace_lines;
/*    85      |       1 */  char f_trace_opcodes;
/* XXX  2-byte hole       */
/*    88      |       8 */  PyObject *f_gen;
/*    96      |       4 */  int f_lasti;
/*   100      |       4 */  int f_lineno;
/*   104      |       4 */  int f_iblock;
/*   108      |       1 */  PyFrameState f_state;
/* XXX  3-byte hole       */
/*   112      |     240 */  PyTryBlock f_blockstack[20];
/*   352      |       8 */  PyObject *f_localsplus[1];

                               /* total size (bytes):  360 */
                            }
```

Docs: frame.ob_base has a reference to object's type (ob_type).
Can we rely on certain characteristic to identify a PyFrameObject pointer?

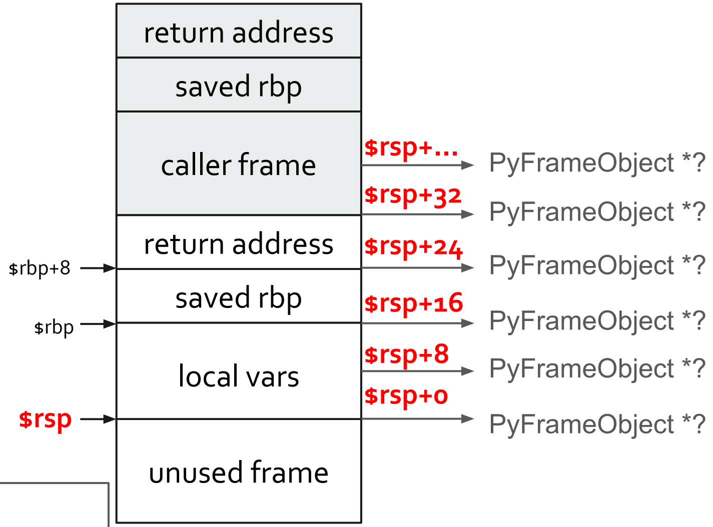
Docs: frame.f_back points to the previous stack frame (towards the caller).
Can we unwind python stack using it?

Docs: frame.f_code holds code object being executed in this frame.
Can we read python func name and filename from it?

Can we unwind Python stack based on PyFrameObject?

PyFrameObject->ob_base: to find a PyFrameObject

```
(gdb) bt
...
#7  _PyEval_EvalFrameDefault (tstate=<optimized out>, f=<optimized out>, throwflag=<optimized out>)
       at Python/ceval.c:4181
#8  0x00007b2ee9c0992a in _PyEval_EvalFrame (throwflag=<optimized out>, f=0x7b2ee78b72e0,
       tstate=0x60cadbf29190) at ./Include/internal/pycore_ceval.h:46

(gdb) f 8
(gdb) p f
$10 = (PyFrameObject *) 0x7b2ee78b72e0

(gdb) p f->ob_base
(gdb) ptype f->ob_base
(gdb) p f->ob_base.ob_base.ob_type->tp_name
$12 = 0x7b2ee9c93ba4 "frame"

(gdb) p ((PyFrameObject*)0x7b2ee78b72e0)->ob_base.ob_base.ob_type->tp_name
$12 = 0x7b2ee9c93ba4 "frame"

(gdb) p ((PyFrameObject*)0x7b2ee78b7233)->ob_base.ob_base.ob_type->tp_name
Cannot access memory at address 0x24
```



Conclusion: We can find a PyFrameObject by finding an stack address where
`(PyFrameObject*)$addr->ob_base.ob_base.ob_type->tp_name == “frame”`

PyFrameObject->f_back: to unwind Python stack

```
(gdb) p f
$10 = (PyFrameObject *) 0x7b2ee78b72e0

(gdb) ptype f
type = struct _frame {
...
} *

(gdb) p f->f_back
$11 = (struct _frame *) 0x60cadc2f9780
```

Conclusion: We can unwind stack in Python VM level via `PyFrameObject->f_back`!

```
struct PyFrameObject *f;

for (int i=0; i<20; i++) {
        f = f->f_back;
        if (!f)
                break;
}
```

PyFrameObject->f_code: to get Python func/file name

```
(gdb) p f
$2 = (PyFrameObject *) 0x7b2ee78b72e0

(gdb) p f->f_code
$3 = (PyCodeObject *) 0x7b2ee85f8df0


(gdb) ptype f->f_code
...
(gdb) p f->f_code->co_name
$4 = (PyObject *) 0x7b2ee85f0ef0

(gdb) x/100s f->f_code->co_name
(gdb) x/s (unsigned long long)f->f_code->co_name + 48
0x7b2ee85f0f20:     "tick"

(gdb) x/s (unsigned long long)f->f_code->co_filename + 48
0x7b2ee85dcae0:     "/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py"
```

```
#define STRUCT_FOR_ASCII_STR(LITERAL) \
      struct { \
      PyASCIIObject _ascii; \
      uint8_t _data[sizeof(LITERAL)]; \
      }

sizeof(PyASCIIObject) == 48
```

But to be honest, you don't need to know this CPython implementation detail.

Conclusion: We can get python function name and filename via `PyFrameObject->f_code->{co_name,co_filename} + 48`!

bpftrace script

```
profile:hz:99 /pid == $1/
{
	$rsp = reg("sp");
	$frame = (struct PyFrameObject *)0;
	$i = (uint64)0;
	$sp = (uint64)$rsp;
	while ($i <= 200) {
		$frame = *(struct PyFrameObject**)($sp + 8*$i);
		if (str($frame->ob_base.ob_base.ob_type->tp_name, 5) == "frame") {
			break;
		}
		$frame = (struct PyFrameObject *)0;
		$i += 1;
	}

	if ($frame == 0) {
		return;
	}
	$i = 0;
	printf("\n");
	while ($i < 20) {
		printf("%s:%s\n", $frame->f_code->co_filename->buf, $frame->f_code->co_name->buf);
		$i += 1;
		$frame = $frame->f_back;
		if ($frame == 0) {
			return;
		}
	}
}
```

0. eBPF hook: `perf -F99 -p $PID`

1. Get user space register

2. Search stack memory from `$rsp`, find the `PyFrameObject` pointer via `PyFrameObject->ob_base.ob_base.ob_type->tp_name`

3. Print py function name and filename via `PyFrameObject->f_code->{co_name,co_filename} + 48`

4. Unwind py stack via `PyFrameObject->f_back`

Misc: Type Definitions (cpython310.h)

```
struct PyTypeObject {
      char _[24];
      char *tp_name;
};
struct PyObject {
      char _[8];
      struct PyTypeObject *ob_type;
};
struct PyVarObject {
      struct PyObject ob_base;
      char _[8];
};

struct _PyStr {
      char _[48];
      char buf[100];
};
struct PyCodeObject {
      char _[104];
      struct _PyStr *co_filename;
      struct _PyStr *co_name;
};

struct PyFrameObject {
      struct PyVarObject ob_base;
      struct PyFrameObject *f_back;
      struct PyCodeObject *f_code;
};
```

CPython version sensitive
But method applies

Remember `PyFrameObject->f_code->{co_name,co_filename} + 48`?
We don't need to know why it's 48 bytes offset, just define an unused field of 48 bytes size

We don't need to define all fields

Misc: Output (group.py)

```
.../autoreload.py:tick
.../autoreload.py:run_loop
.../autoreload.py:run

.../autoreload.py:tick
.../autoreload.py:run_loop
.../autoreload.py:run

.../text.py:_truncate_html
.../text.py:words
.../views.py:index
```

⇩

```
.../text.py:_truncate_html
.../text.py:words
.../views.py:index: 1

.../autoreload.py:tick
.../autoreload.py:run_loop
.../autoreload.py:run: 2
```

```
import sys
from collections import defaultdict

stack = []
stacks = defaultdict(int)
try:
      for line in sys.stdin:
      stack.append(line.strip())
      if not line.strip():
            stacks['\n'.join(stack)] += 1
            stack = []
except KeyboardInterrupt:
      pass

# output by count of stacks
for stack, count in sorted(stacks.items(), key=lambda x: x[1]):
      print(f'{stack.strip()}: {count}\n')
```

Read from stdin (shell pipe)

Split by paragraph

Sort by number of occurrences

Misc: Kernel space (ubuntu2204.h)

```
#define PAGE_SIZE (1<<12)
#define KASAN_STACK_ORDER 0
#define THREAD_SIZE_ORDER (2 + KASAN_STACK_ORDER)
#define THREAD_SIZE  ((uint64)(PAGE_SIZE << THREAD_SIZE_ORDER))
#define TOP_OF_KERNEL_STACK_PADDING ((uint64)0)

profile:hz:99 /pid == $1/
{
      $rsp = reg("sp");

      // kernel mode
      $user_mode = reg("cs") & 3;
      if (!$user_mode) {
            $task = (struct task_struct *)curtask;
            $__ptr = (uint64)$task->stack;
            $__ptr += THREAD_SIZE - TOP_OF_KERNEL_STACK_PADDING;
            $pt_regs = ((struct pt_regs *)$__ptr) - 1;
            $rsp = $pt_regs->sp;
      }

...
}
```

This may point to kernel stack if process is trapped in kernel mode (syscall)

If in kernel mode, get user space registers from task_struct.
(Algorithm and macros may vary among different kernel versions and distros.)

Conclusion: We must get user space registers if process is in kernel space.

```
$ bpftrace --include ../headers/cpython310.h --include ../headers/ubuntu2204.h cpython_perf.bt $PID | python group.py
WARNING: Addrspace is not set
^CAttaching 1 probe...: 1

/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py:tick
/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py:run_loop
/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py:run
/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py:start_django
/usr/local/lib/python3.10/site-packages/django/utils/autoreload.py:run_with_reloader
/usr/local/lib/python3.10/site-packages/django/core/management/commands/runserver.py:run
/usr/local/lib/python3.10/site-packages/django/core/management/commands/runserver.py:handle
/usr/local/lib/python3.10/site-packages/django/core/management/base.py:execute
/usr/local/lib/python3.10/site-packages/django/core/management/commands/runserver.py:execute
/usr/local/lib/python3.10/site-packages/django/core/management/base.py:run_from_argv
/usr/local/lib/python3.10/site-packages/django/core/management/__init__.py:execute
/usr/local/lib/python3.10/site-packages/django/core/management/__init__.py:execute_from_command_line
/src/manage.py:main
/src/manage.py:<module>: 1

/usr/local/lib/python3.10/site-packages/django/utils/text.py:_truncate_html
/usr/local/lib/python3.10/site-packages/django/utils/text.py:words
/src/polls/views.py:index
/usr/local/lib/python3.10/site-packages/django/views/decorators/csrf.py:_view_wrapper
/usr/local/lib/python3.10/site-packages/django/core/handlers/base.py:_get_response
/usr/local/lib/python3.10/site-p
/usr/local/lib/python3.10/site-p
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inner
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inner
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inner
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inner
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inne
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inne
/usr/local/lib/python3.10/site-packages/django/utils/deprecation.py:__call__
/usr/local/lib/python3.10/site-packages/django/core/handlers/exception.py:inner: 201
```

Deepest frame points to the direct cause of CPU use

Stack backtrace tells how we end up there

201 out of 202 samples are this stack

	Disruptive	Requirements in Python build flags	Overhead	Accuracy
`perf`	No	Yes	Low (sampling)	No Python stack
`perf + python -X perf`	Yes	Yes	Moderate (non-sampling)	One layer of Python stack
`py-spy`	No	No	High (sampling)	Possible misleading result
`bpftrace`	No	No	Super low (sampling)	100% on-CPU stacks

RPS -0% (frequency 99, python stack depth 20)

CVE-2024-27351

Common Vulnerabilities and Exposures

[Previous] [Index] [Next]

Upstream information

CVE-2024-27351 at MITRE

Description

In Django 3.2 before 3.2.25, 4.2 before 4.2.11, and 5.0 before 5.0.3, the django.utils.text.Truncator.words() method (with html=True) and the truncatewords_html template filter are subject to a potential regular expression denial-of-service attack via a crafted string. NOTE: this issue exists because of an incomplete fix for CVE-2019-14232 and CVE-2023-43665.

Django ticket #27690

Symptoms

1. Django DB test runs slowly (>1s) even for empty project (`time python manage.py test`)
2. Django Python process **doesn't** consume CPU (`top`)

Random guesses:

- Mutex?
- Disk IO? Network IO?
- Kernel? (scheduler? cgroups?) (unlikely in this case)

Idea: Find the **syscalls** responsible for off-CPU, likely that's the cause of slow response.

strace

```
$ strace -fTtt -- python manage.py test
```

pselect6(2) costs 1s, who called it?

```
...
01:44:55.406993 pselect6(0, NULL, NULL, NULL, {tv_sec=1, tv_nsec=0}, NULL) = 0 (Timeout) <1.000610>
...
```

-k prints the execution stack trace of the traced processes after each system call.

```
$ strace -e pselect6 -fTtt -k  --  python manage.py test

01:47:36.402247 pselect6(0, NULL, NULL, NULL, {tv_sec=1, tv_nsec=0}, NULL) = 0 (Timeout) <1.001058>
 > /usr/lib/x86_64-linux-gnu/libc.so.6(__select+0xbd) [0x11b59d]
 > /usr/bin/python3.10(_Py_Gid_Converter+0x736) [0x279bb6]
 > /usr/bin/python3.10(PyObject_GenericGetAttr+0x624) [0x15c574]
 > /usr/bin/python3.10(_PyEval_EvalFrameDefault+0x613a) [0x14b34a]
 > /usr/bin/python3.10(_PyFunction_Vectorcall+0x7c) [0x15d42c]
 > /usr/bin/python3.10(_PyEval_EvalFrameDefault+0x8ab) [0x145abb]
 > /usr/bin/python3.10(_PyFunction_Vectorcall+0x7c) [0x15d42c]
 > /usr/bin/python3.10(_PyEval_EvalFrameDefault+0x8ab) [0x145abb]
 > /usr/bin/python3.10(_PyFunction_Vectorcall+0x7c) [0x15d42c]
...
 > /usr/bin/python3.10(_PyRun_AnyFileObject+0x43) [0x261793]
 > /usr/bin/python3.10(Py_RunMain+0x2be) [0x2542ce]
 > /usr/bin/python3.10(Py_BytesMain+0x2d) [0x22a70d]
 > /usr/lib/x86_64-linux-gnu/libc.so.6(__libc_init_first+0x90) [0x29d90]
 > /usr/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0x80) [0x29e40]
 > /usr/bin/python3.10(_start+0x25) [0x22a605]
```

strace outputs C functions rather than Python functions.

```
kprobe:__x64_sys_pselect6 /comm == "python3.10"/
{
        $task = (struct task_struct *)curtask;
        $__ptr = (uint64)$task->stack;
        $__ptr += THREAD_SIZE - TOP_OF_KERNEL_STACK_PADDING;
        $pt_regs = ((struct pt_regs *)$__ptr) - 1;
        $rsp = uptr($pt_regs->sp);

        $frame = uptr((struct PyFrameObject *)0);
        $i = (uint64)0;
        while ($i <= 200) {
                $frame = *uptr((struct PyFrameObject**)($rsp + 8*$i));
                if (str($frame->ob_base.ob_base.ob_type->tp_name, 5) == "frame") {
                        break;
                }
                $frame = (struct PyFrameObject *)0;
                $i += 1;
        }

        if ($frame == 0) {
                return;
        }

        $i = 0;
        while ($i < 20) {
                printf("%s:%s\n", $frame->f_code->co_filename->buf, $frame->f_code->co_name->buf);
                $i += 1;
                $frame = $frame->f_back;
                if ($frame == 0) {
                        return;
                }
        }
}
```

0. eBPF hook: triggered by pselect6 syscall

1. Get user space register

2. Search stack to find the `PyFrameObject` pointer

3. Print py function name and filename

4. Unwind python stack

```
$ sudo bpftrace --include ../headers/cpython310.h --include ../headers/ubuntu2204.h cpython_syscall.bt

Attaching 1 probe...

/home/liangzc/.local/lib/python3.10/site-packages/django/db/backends/sqlite3/creation.py:_destroy_test_db
/home/liangzc/.local/lib/python3.10/site-packages/django/db/backends/base/creation.py:destroy_test_db
/home/liangzc/.local/lib/python3.10/site-packages/django/test/utils.py:teardown_databases
/home/liangzc/.local/lib/python3.10/site-packages/django/test/runner.py:teardown_databases
/home/liangzc/.local/lib/python3.10/site-packages/django/test/runner.py:run_tests
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/commands/test.py:handle
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/base.py:execute
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/base.py:run_from_argv
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/commands/test.py:run_from_argv
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/__init__.py:execute
/home/liangzc/.local/lib/python3.10/site-packages/django/core/management/__init__.py:execute_from_command_line
/home/liangzc/src/github.com/jschwinger233/pycon2024/django27690/manage.py:main
/home/liangzc/src/github.com/jschwinger233/pycon2024/django27690/manage.py:<module>
```

```
def _destroy_test_db(self, test_database_name, verbosity=1):
    ...
    cursor = self._maindb_connection.cursor()
    time.sleep(1) # To avoid "database is being accessed by other users" errors.
    if self._test_user_create():
        ...
```

Root cause: `sleep(1)` in `_destroy_test_db()`

Off-CPU profile

Off-CPU profiling explains where a process goes off-CPU + how long been off-CPU.

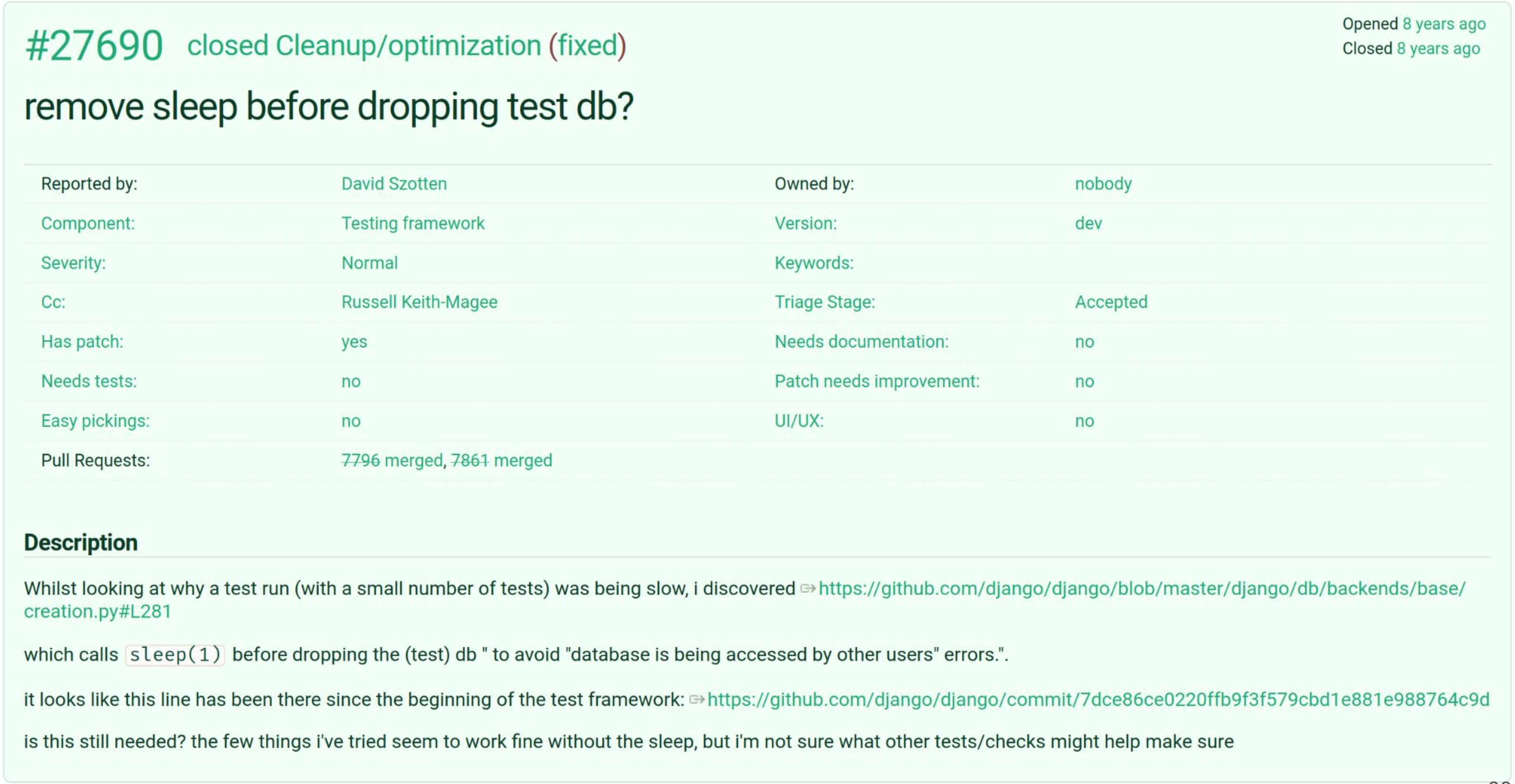
```
kprobe:finish_task_switch.isra.0
{
      $prev = (struct task_struct *)arg0;
      if ($prev->tgid == $1) {
            @start[$prev->pid] = nsecs;
      }

      $last = @start[tid];
      if ($last != 0) {
           // Calculate user space $rsp
            ...

            // Search PyFrameObject *
            ...

           // Unwind stack, print function and filename
            ...

            printf("offcpu time: %lld\n", nsecs - $last);

            delete(@start[tid]);
      }
}
```

eBPF hook: triggered by kernel task schedule

process switched from on-cpu to off-cpu

process switched from off-cpu to on-cpu

Off-CPU profiling is useful for investigating “process can’t utilize 100% CPU”.



#27690 closed Cleanup/optimization (fixed)

Opened 8 years ago
Closed 8 years ago

remove sleep before dropping test db?

Reported by:	David Szotten	Owned by:	nobody
Component:	Testing framework	Version:	dev
Severity:	Normal	Keywords:	
Cc:	Russell Keith-Magee	Triage Stage:	Accepted
Has patch:	yes	Needs documentation:	no
Needs tests:	no	Patch needs improvement:	no
Easy pickings:	no	UI/UX:	no
Pull Requests:	~~7796~~ merged, ~~7861~~ merged		

Description

Whilst looking at why a test run (with a small number of tests) was being slow, i discovered https://github.com/django/django/blob/master/django/db/backends/base/creation.py#L281

which calls `sleep(1)` before dropping the (test) db " to avoid "database is being accessed by other users" errors.".

it looks like this line has been there since the beginning of the test framework: https://github.com/django/django/commit/7dce86ce0220ffb9f3f579cbd1e881e988764c9d

is this still needed? the few things i've tried seem to work fine without the sleep, but i'm not sure what other tests/checks might help make sure

BentoML issue #4760

Symptoms

```
$ bentoml serve service.py:TestService
$ while :; do curl localhost:3000; done &>/dev/null
```

1. Memory leak (`docker stats $container`)
2. No memory leak (`ps -p $PID -o %mem,rss,vsz`)

How “`ps`” and “`docker stats`” are implemented?

```
      ps -o rss,vsz                              docker stats
            ↓                                          ↓
    /proc/$PID/status                    /sys/fs/cgroup/.../memory.stat
            ↓                                          ↓
    user space memory                       also kernel space memory
```

Yeah it’s definitely a case of kernel space memory leak triggered by user space Python code, we need to:

1. Find a hook of kernel space memory allocation.
2. Filter events that are triggered by Python process
3. Do user space Python stack unwind to see the call chain causing kmem leak.

eBPF hook: triggered when kmem is allocated

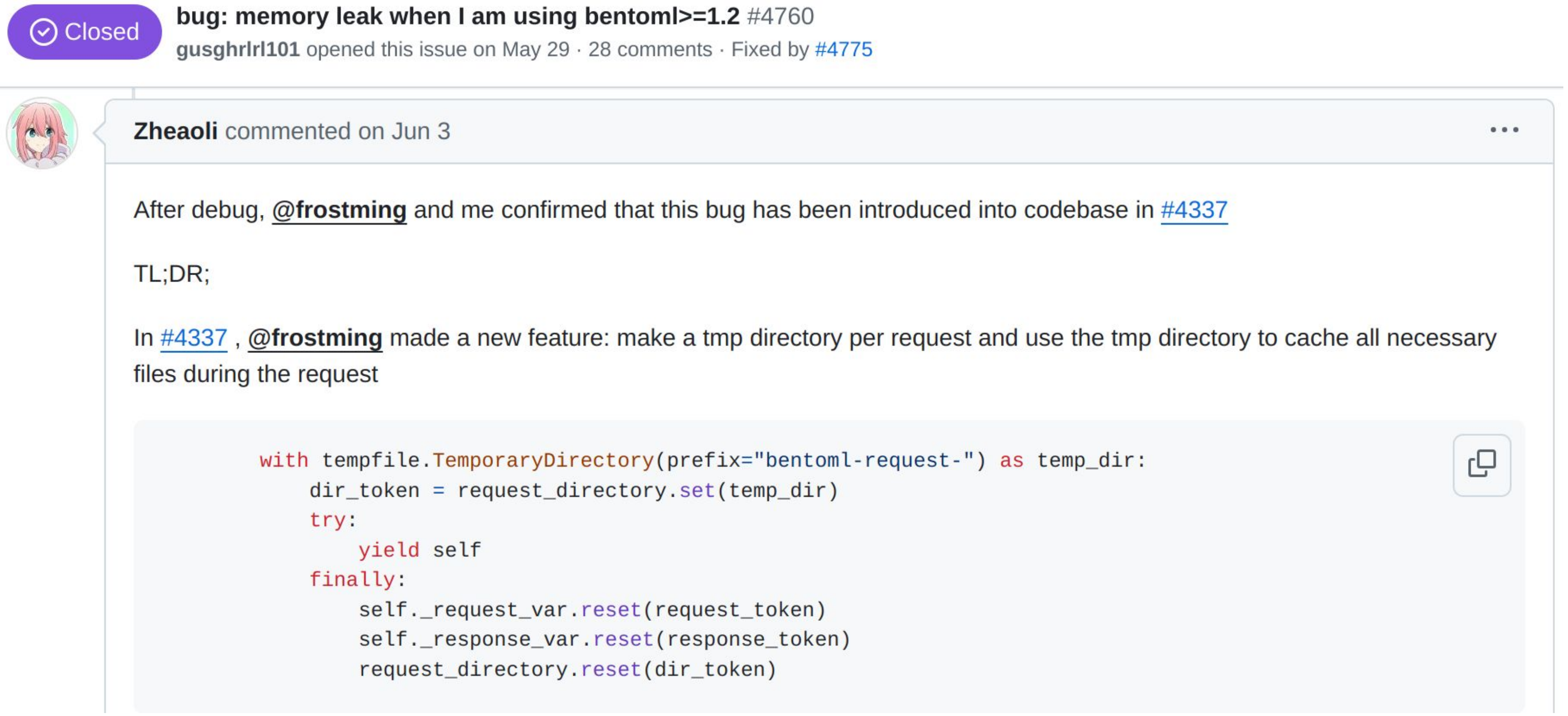
```
tracepoint:kmem:mm_page_alloc  /pid == $1/
{
      // Calculate user space $rsp in kernel mode
      ...
      // Search PyFrameObject * from user space stack
      ...
      // Unwind Python stack, print function and filename
      ...
}
```

Exactly same code as last section

```
$ bpftrace --include ../headers/cpython310.h --include ../headers/ubuntu2204.h cpython_kmemleak.bt $PID | group.py
```

Root cause: mktemp dir for each request

```
/usr/local/lib/python3.10/tempfile.py:mkdtemp
/usr/local/lib/python3.10/tempfile.py:__init__
/usr/local/lib/python3.10/site-packages/bentoml/_internal/context.py:in_request
/usr/local/lib/python3.10/contextlib.py:__enter__
/usr/local/lib/python3.10/site-packages/_bentoml_impl/server/app.py:__call__
/usr/local/lib/python3.10/asyncio/events.py:_run
/usr/local/lib/python3.10/site-packages/click/core.py:main
/usr/local/lib/python3.10/site-packages/click/core.py:__call__
/usr/local/lib/python3.10/site-packages/_bentoml_impl/worker/service.py:<module>
/usr/local/lib/python3.10/runpy.py:_run_code
/usr/local/lib/python3.10/runpy.py:_run_module_as_main:54
```


Closed

bug: memory leak when I am using bentoml>=1.2 #4760

gusghrlrl101 opened this issue on May 29 · 28 comments · Fixed by #4775

Zheaoli commented on Jun 3

After debug, **@frostming** and me confirmed that this bug has been introduced into codebase in #4337

TL;DR;

In #4337 , **@frostming** made a new feature: make a tmp directory per request and use the tmp directory to cache all necessary files during the request

```
        with tempfile.TemporaryDirectory(prefix="bentoml-request-") as temp_dir:
            dir_token = request_directory.set(temp_dir)
            try:
                yield self
            finally:
                self._request_var.reset(request_token)
                self._response_var.reset(response_token)
                request_directory.reset(dir_token)
```

Summary

1. Traditional debugging tools have limitations
2. eBPF is able to unwind CPython stack
3. Using different Linux hooks, we debug different issues
    a. On-CPU performance issue: sampling(perf_event)
    b. Off-CPU performance issue: syscall(kprobe), off-CPU scheduling(kprobe)
    c. Kernel space memory leak: kmem alloc(tracepoint)
4. Super fast, non-disruptive, zero CPython requirement, 100% accuracy
5. Other interesting topics
    a. Collect Python function arguments/local vars
    b. Execution flows (Python3.11- DTrace probes)
    c. Topdown analysis (super super fun)

Thank you! Please don’t ask tough questions 😬😭

eBPF