

ExecVus

Alexandru Totolici

What? Why? How?

a way to visualize control-flow in
software execution

What? Why? How?

underlying data collected by Tralfamadore

<http://tralfamadore.cs.ubc.ca/>

What? Why? How?

existing visualization...

What? Why? How?

```
673
674 lock_sock(sk);
675 TCP_CHECK_TIMER(sk);
676
677 flags = msg->msg_flags;
678 timeo = sock_sndtimeo(sk, flags & MSG_DONTWAIT);
679
680 /* Wait for a connection to finish. */
681 if ((1 << sk->sk_state) & ~(TCPF_ESTABLISHED | TCPF_CLOSE_WAIT))
682     if ((err = sk_stream_wait_connect(sk, &timeo)) != 0)
683         goto out_err;
684
685 /* This should be in poll */
686 clear_bit(SOCK_ASYNC_NOSPACE, &sk->sk_socket->flags);
687
688 mss_now = tcp_current_mss(sk, !(flags&MSG_OOB));
```

call tcp_current_mss(114) tcp_current_mss(28) tcp_current_mss(23) tcp_current_mss(23)
tcp_current_mss(4) tcp_current_mss(3) tcp_current_mss(2) tcp_current_mss(1)
tcp_current_mss(1) tcp_current_mss(1)

```
692 iovlen = msg->msg_iovlen;
693 iov = msg->msg_iov;
694 copied = 0;
695
696 err = -EPIPE;
697 if (sk->sk_err || (sk->sk_shutdown & SEND_SHUTDOWN))
698     goto do_error;
699
700 while (--iovlen >= 0) {
701     int seglen = iov->iov_len;
702     unsigned char __user *from = iov->iov_base;
```

inline tcp_mark_urg

```
704     iov++;
```

```
705
```

```
102
```

```
104
```

```
iov++;
```

What? Why? How?

but:

occlusion (of paths)

What? Why? How?

```
673
674 lock_sock(sk);
675 TCP_CHECK_TIMER(sk);
676
677 flags = msg->msg_flags;
678 timeo = sock_sndtimeo(sk, flags & MSG_DONTWAIT);
679
680 /* Wait for a connection to finish. */
681 if ((1 << sk->sk_state) & ~(TCPF_ESTABLISHED | TCPF_CLOSE_WAIT))
682     if ((err = sk_stream_wait_connect(sk, &timeo)) != 0)
683         goto out_err;
684
685 /* This should be in poll */
686 clear_bit(SOCK_ASYNC_NOSPACE, &sk->sk_socket->flags);
687
688 mss_now = tcp_current_mss(sk, !(flags&MSG_OOB));
```

call tcp_current_mss(114) tcp_current_mss(28) tcp_current_mss(23) tcp_current_mss(23)
tcp_current_mss(4) tcp_current_mss(3) tcp_current_mss(2) tcp_current_mss(1)
tcp_current_mss(1) tcp_current_mss(1)

```
692 iovlen = msg->msg_iovlen;
693 iov = msg->msg_iov;
694 copied = 0;
695
696 err = -EPIPE;
697 if (sk->sk_err || (sk->sk_shutdown & SEND_SHUTDOWN))
698     goto do_error;
```

how many paths here?

```
inline tcp_mark_urg
inline tcp_mark_urg
```

What? Why? How?

but:

occlusion (of paths)

selection (of flow)

What? Why? How?

```
673
674 lock_sock(sk);
675 TCP_CHECK_TIMER(sk);
676
677 flags = msg->msg_flags;
678 timeo = sock_sndtimeo(sk, flags & MSG_DONTWAIT);
679
680 /* Wait for a connection to finish. */
681 if ((1 << sk->sk_state) & ~(TCPF_ESTABLISHED | TCPF_CLOSE_WAIT))
682     if ((err = sk_stream_wait_connect(sk, &timeo)) != 0)
683         goto out_err;
684
685 /* This should be in poll */
686 clear_bit(SOCK_ASYNC_NOSPACE, &sk->sk_socket->flags);
687
688 mss_now = tcp_current_mss(sk, !(flags&MSG_OOB));
```

call tcp_current_mss(114) tcp_current_mss(28) tcp_current_mss(23) tcp_current_mss(23)
tcp_current_mss(4) tcp_current_mss(3) tcp_current_mss(2) tcp_current_mss(1)
tcp_current_mss(1) tcp_current_mss(1)

which paths go where?

```
692 iovlen = msg->msg_iovlen;
693 iov = msg->msg_iov;
694
695 err = -EPIPE;
696 if (sk->sk_err || (sk->sk_shutdown & SEND_SHUTDOWN))
697     goto do_error;
698
699
700 while (--iovlen >= 0) {
701     int seglen = iov->iov_len;
702     unsigned char __user *from = iov->iov_base;
```

inline tcp_mark_urg

What? Why? How?

but:

occlusion (of paths)

selection (of flow)

location (in execution)

What? **Why?** How?

control-flow is non-trivial to “see”

What? **Why?** How?

scenario 1: debug

What? **Why?** How?

scenario 2: malware

What? Why? **How?**

code collapse

What? Why? **How?**

code reordering

What? Why? **How?**

improve selection of execution path

What? Why? **How?**

“zoom” in and out (in the control-flow)

What? Why? **How?**

The screenshot shows the ExecVus web application interface. At the top, the browser address bar displays `http://execv.us/linux/deep-inside-the-kernel-source-tree/somefile.c`. The main content area is a code editor showing the following C code:

```
 /
 * ©HappyGoLucky Inc.
 * Hello <user> program to demonstrate our superior Linux Module Development
 strategies
 */

#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
MODULE_LICENSE("GPL or BSD/GPL");

#define BITESIZE 64000 MORE than enough to fit anyone's first name

static char name[BITESIZE];

static int proc_read(char *buf, char **start, off_t offset, int count, int *eof, void
 *data) {
    int len;
    len = sprintf(buf, "Hello %s", name);
    *eof = someOtherFunction(1);
    return len;
}
```

Annotations on the code editor:

- A callout box labeled "Click node" with the text "...for radial menu with options to linearize" points to a radial menu at the bottom left of the code editor.
- A callout box labeled "Collapse" with the text "... the function below" points to a minus sign icon next to the `static int proc_read` function definition.

At the bottom left, a file browser shows three files: `api.c`, `640k.h`, and `somefile.c`. The `somefile.c` file is highlighted in green.

On the right side, a sidebar titled "Call Sources" contains a list of callers, each represented by a colored circle icon and the text `<caller>`. Below this list are sections for "Histogram" and "Search".

What? Why? **How?**

The screenshot shows the ExecVus web interface. The browser address bar contains `http://execv.us/linux/deep-inside-the-kernel-source-tree/somefile.c`. The main content area displays C code with several annotations:

- Code Annotations:**
 - Four colored circles (green, cyan, yellow, red) are placed above the function signature `static int proc_read(char *buf, char **start, char **eof, int len) {`. A note below them says: "Tabs keep track of where we've been (it's easy to get lost looking at code)".
 - A box points to the line `*eof = someOtherFunction(1);` with the text: "Click to 'zoom in'". Below this box, it says: "New view will have exec flow selected automatically. New tab created or we're moving a previous on to the top of the pile."
 - A box points to the `<caller>` entries in the sidebar with the text: "Click to 'zoom out' to caller.". Below this box, it says: "New view will have exec flow selected automatically, new tab added or moved to the front of the line."
 - A box points to the sidebar with the text: "Click to see call graph". Below this box, it says: "Who called this caller higher up?"
- Call Sources Sidebar:** A vertical list of entries, each starting with `<caller>` and followed by `<...>`. Each entry has a small colored circle to its left. The top entry is highlighted in green.
- Bottom Panel:** A tab bar at the bottom shows three tabs: `api.c`, `640k.h`, and `somefile.c`. The `somefile.c` tab is selected and highlighted in green. Below the tabs are sections for "Histogram" and "Search".

What? Why? **How?**

Mercurial, jQuery, Python, Cappuccino