Userspace live patching with Libpulp

Gabriel F. T. Gomes

gagomes@suse.de
gabriel@inconstante.net.br
Agenda

- Introduction
- Basics
- Live demo
- Details
- How to get started
- Contributing
- Q&A
- More details
Introduction

• Kernel live patching
• Program classes
• Shared libraries
  – Application-Library boundary
    • Two tracking methods
  – Program consistency
Basic operation
(function prologues with -fpatchable-function-entry)

Before patching

<function-22> : nop
<function-21> : nop
...
<function-1>  : nop
<function>    : nop (2-bytes)
<function+2>  : ...

After patching

<function-22> : push %rdi
<function-21> : mov $0x17,%rdi
<function-14> : jmpq *0x0(%rip)\(^1\)
<function-8>  : <data>
<function>    : jmp <function-22>
<function+2>  : ...

\(^1\) replaced with the runtime address of the version handling routine
How to change memory?

• ptrace(2)
  - PTRACE_ATTACH (to all threads)
  - Steal the context of a thread to run Libpulp functions
  - Restore the context
  - PTRACE_DETACH (from all threads)
  - PS: Like GDB does
Libpulp functions

• Provided by libpulp.so

• *Executed* from Libpulp’s tools
  - ulp_trigger, ulp_check, etc.
  - The tools ptrace into the target process.
In a nutshell

● Start a process with livepatching support:

$ LD_PRELOAD=libpulp.so program

$ echo $(pidof program)
123

● Use the tools to attach to the process:

$ ulp_check -p 123 livepatch.ulp
ulp: patch not yet applied

$ ulp_trigger -p 123 livepatch.ulp
ulp: live patching succeeded

$ ulp_check -p 123 livepatch.ulp
ulp: patch already applied
Real-world example (not so real)

Pause for live demo
(backup slides available)
# LD_PRELOAD=/usr/lib64/libpulp.so.0 /usr/sbin/nginx
[1] 1234
libpulp loaded...

# nmap -script=ssl-heartbleed -p 443 localhost
...
PORT    STATE SERVICE
443/tcp open  https
| ssl-heartbleed:
|   VULNERABLE:
|   The Heartbleed Bug is a serious vulnerability ...
Real World Testing (backup slide)

# ulp_trigger 1234 /usr/lib64/openssl-1_1-livepatches/heartbleed.ulp
Ulp: Patching 1234 successful.

# nmap -script=ssl-heartbleed -p 443 localhost
... (longer than before)
PORT    STATE SERVICE
443/tcp open    https

Nmap done: 1 IP address (1 host up) scanned in 5 seconds
Real World Testing - Heartbleed

• Buggy functions:
  – dtls1_process_heartbeat
  – tls1_process_heartbeat

• Luckily, very simple:
  – Only two functions
  – Leaf functions
Anatomy of a live patch

- Live patch source example:
  ```c
  #include <stdio.h>
  void function_new(void) {
    printf("patched\n");
  }
  ```

- Build as a library:
  ```bash
  $ gcc livepatch.c -shared -fPIC -o livepatch.so
  ```
Anatomy of a live patch

- Metadata file (.ulp)
  - Built from a description file:
    1. /path/to/livepatch.so
    2. @/path/to/target/library.so
    3. function:function_new
    4. function_2:new_function_2

- Live patch DSO

- List of replacement functions
Version handling routines

- Vertically: functions
- Horizontally: versions
How to get started - too late? ;P

$ git clone git@github.com:SUSE/libpulp.git
$ ./bootstrap
$ ./configure
$ make
$ make check

Notes
1. needs autoconf, automake and libtool
2. missing requirements should produce useful warnings
Please report bugs! :) S2
Test cases

XFAIL: blocked.py
PASS: syscall_restart.py
PASS: pagecross.py
PASS: parameters.py
PASS: asunsafe_conversion.py
PASS: constructor.py
XFAIL: hidden.py
PASS: numserv_bsymbolic.py
PASS: memory_protection.py
PASS: numserv.py
PASS: exception_handling.py
PASS: revert.py
XFAIL: contract.py
PASS: redzone.py
PASS: cancel.py
PASS: terminal.py
PASS: deadlock.py
PASS: recursion.py
Please report bugs! :) S2
Project structure

- **lib**
  - Runtime support (libpulp.so)
- **tools**
  - Command-line tools (ulp_trigger, ulp_check, etc.)
  - Process introspection (Elf, ptrace)
- **tests**
  - Python’s pexpect
  - Live patches
- **man**
  - Documentation
Contributing

- [https://github.com/SUSE/libpulp](https://github.com/SUSE/libpulp)
  - Issues, pull requests (CI tested)

- [ulp-devel@opensuse.org](mailto:ulp-devel@opensuse.org)
  - [https://lists.opensuse.org/archives/list/ulp-devel@lists.opensuse.org](https://lists.opensuse.org/archives/list/ulp-devel@lists.opensuse.org)

- README.md
  - Just guidelines
Contributing

2 Open  0 Closed

- **Live patches rely on in-disk images of libraries**
  #20 opened 10 hours ago by inconstante

- **About ptrace freeze time**
  #8 opened on Feb 22 by szhangtou
Thanks!

gagomes@suse.de
gabriel@inconstante.net.br
Backup slides
Asynchronous signal safety

• AS-Unsafe functions in use:
  - dlopen
  - dlsym
  - calloc

• Problems
  - Deadlocks during live patch application
  - AS-Unsafe conversion during regular execution
Deadlocks

Due to malloc locks

(gdb) bt
#0 0x00007f2622e54e37 in __lll_lock_wait_private () from /lib64/libc.so.6
#1 0x00007f2622e5b2a3 in calloc () from /lib64/libc.so.6
#2 0x00007f2622fbeb18 in _dlerror_run () from /lib64/libdl.so.2
#3 0x00007f2622fbeb2e4 in dlopen@GLIBC_2.2.5 () from /lib64/libdl.so.2
#4 0x00007f2622fccc780 in load_so () at /root/src/libpulp/lib/ulp.c:463
#5 0x00007f2622fcb99a in load_so_handlers () at /root/src/libpulp/lib/ulp.c:264
#6 0x00007f2622fccc751 in parse_metadata () at /root/src/libpulp/lib/ulp.c:454
#7 0x00007f2622fcbaff in load_metadata () at /root/src/libpulp/lib/ulp.c:304
#8 0x00007f2622fcbcc74c in load_patch () at /root/src/libpulp/lib/ulp.c:478
#9 0x00007f2622fcb628 in __ulp_apply_patch () at /root/src/libpulp/lib/ulp.c:99
#10 0x00007f2622fceb5e42 in __ulp_trigger () at /root/src/libpulp/lib/ulp_interface.S:44
#11 0x00007f2622f8f7e0 in ?? () from /lib64/libc.so.6
#12 0x00007f2622f8f7e0 in ?? () from /lib64/libc.so.6
#13 0x00007f2622f8f7e0 in ?? () from /lib64/libc.so.6
#14 0x0000000000000000 in ?? ()
Deadlocks

Due to dlfcn locks

(gdb) bt
#0 0x00007fbc87fb3f5c in __lll_lock_wait () from /lib64/libpthread.so.0
#1 0x00007fbc87fac8f1 in pthread_mutex_lock () from /lib64/libpthread.so.0
#2 0x00007fbc87ff5b06 in _dl_open () from /lib64/ld-linux-x86-64.so.2
#3 0x00007fbc87fc9258 in dlopen_doit () from /lib64/libdl.so.2
#4 0x00007fbc87ff5b06 in _dl_lock () from /lib64/libdl.so.2
#5 0x00007fbc87fc14a1f in _dl_error_run () from /lib64/libc.so.6
#6 0x00007fbc87fd7780 in load_so () at /root/src/libpulp/lib/ulp.c:463
#7 0x00007fbc87fd7751 in __ulp_apply_patch () at /root/src/libpulp/lib/ulp.c:478
#8 0x00007fbc87fd7751 in __ulp_activate_patch () at /root/src/libpulp/lib/ulp.c:478
#9 0x00007fbc87fd7751 in __ulp_trigger () at /root/src/libpulp/lib/ulp_interface.S:44
Deadlocks

• Take 1 (exposing internal state of locks in glibc):
  – __libpulp_dlopen_checks
  – __libpulp_malloc_checks

• Take 2 (interpose glibc functions)
  – malloc, calloc, memalign, free, etc.
  – dlopen, dlsym, etc.
Consistency checks

- Process hijacking
- Live patch application
  - Loads live-patch DSO (dlopen and friends)
  - Needs memory (malloc and friends)
- Early checking of all threads
- *Per-thread universe counter updates*
Consistency checks

ulp: beginning live patch application.
ulp: advertising live patch location to libpulp.
ulp: >>> running libpulp functions within target process...
ulp: >>> done.
ulp: libc/libdl locks were busy: patch not applied.
ulp: live patching 192846 failed (attempt #1).
ulp: exiting the critical section (process release).
AS-Unsafe conversion

- Universe handling routines
  - Executed between call and actual target function
    - Similar to how the C runtime resolves function addresses
      - e.g.: call strdup → strdup@plt → _dl_*
      - Calls are allowed: but not AS-Unsafe functions.
    - May not call AS-Unsafe functions
      - Otherwise, live patched functions become AS-Unsafe
Memory protection bits

- Patch prologues in executable segments
- Make segment writable
  - Tweak protection bits
  - Write to the memory
  - w^x violation?
    - All threads stopped (ptraced)
- Respect user settings
Patching Prologues

• Granularity
  - Concurrent execution?
    • All threads stopped under ptraced (like gdb)
    • 2-bytes nops at <function..function+1>

• Page crosses
  - Silent seg fault (under ptrace).

```c
void patch_prologue(void *old_fentry, unsigned int function_index)
{
    memcpy(old_fentry, ulp_prologue, sizeof(ulp_prologue));
    memcpy(old_fentry + 4, &function_index, 4);
}
```
Stack frame stealing

Regular call

Call site:
0x1234  call fn
0x1239  ...

Stack:
  0x1239
%rsp: ...

Call through tracker

Call site:
0x1234  call fn
0x1239  ...

Tracker (__ulp_entry):
0x4321  pop area@TLS
0x4322  call %r11 <target fn>
0x4323  ...
0x5000  push area@TLS
0x5001  ret

Stack:
  0x4323
%rsp: ...
Stack frame stealing

• Breaks unwinding:

(gdb) bt
#0  0x000007f9414e9714f in hello () from libblocked.so.0
#1  0x000007f9414e9718a in hello_loop () at libblocked.c:37
#2  0x000007f9414e97248 in __ulp_entry () at trm.S:153
Backtrace stopped: frame did not save the PC
Stack unwinding

• Thread cancellation
• Exception handling
• Upheaval in consistency checking
  - No more tracking
  - No more stealing of stack frames
  - Pushes the burden onto live patch creation
Red zone

- Signal handler
- Hijacking
  - PTRACE_GETREGS
  - Modify %rip
  - PTRACE_SETREGS
  - What about %rsp?
Library entrance tracking

- Non-standard glibc function:
  - __libpulp_tls_get_addr
  - Less pushing and popping of regs
Library entrance tracking

```c
#ifndef HAVE___LIBPULP_TLS_GET_ADDR
pushq   %r10
pushq   %r9
pushq   %r8
pushq   %rsi
pushq   %rdx
pushq   %rcx
#endif
pushq   %rdi
pushq   %rax
leaq    __ulp_ret@tlsld(%rip), %rdi
#ifndef HAVE___LIBPULP_TLS_GET_ADDR
#endif
call    __tls_get_addr@PLT
#else
call    __libpulp_tls_get_addr@PLT
#endif
```
Universe counters

• Global counter (one)
  – per process

• Local counters (many)
  – per thread
  – per library
Local counter update

leaq __ulp_ret@tlsld(%rip), %rdi
call __tls_get_addr@PLT
1
cmp $0x0, __ulp_ret@dtpoff(%rax)
jnz __ulp_entry_bypass
movq __ulp_global_universe@GOTPCREL(%rip), %rdi
movq %rdi, __ulp_thread_universe@dtpoff(%rax)

1 source of overhead
Multiple libraries

Before

.weak __ulp_global_universe
.section .data
.type __ulp_global_universe, @object
.size __ulp_global_universe, 8
__ulp_global_universe:
.zero 0x8

After

.weak __ulp_global_universe

No local definition
Multiple libraries

+// Check if the address of the global counter has been filled in the
+// GOT entry during initialization (only happens when libulp.so has
+// been LD_PRELOAD'ed). In case it has, read the contents.
+// Otherwise, the uninitialized value, zero, can be used directly.

    movq   __ulp_global_universe@GOTPCREL(%rip), %rdi
    test   %rdi, %rdi
    jz     __ulp_thread_counter_update

    movq   (%rdi), %rdi

__ulp_thread_counter_update:
    movq   %rdi, __ulp_thread_universe@dtpoff(%rax)