VTPin: Protecting Legacy Software from VTable Hijacking

Pawel Sarbinowski, Vasileios P. Kemerlis (Brown)
Cristiano Giuffrida, Elias Athanasopoulos (Vrije U.)

Outline

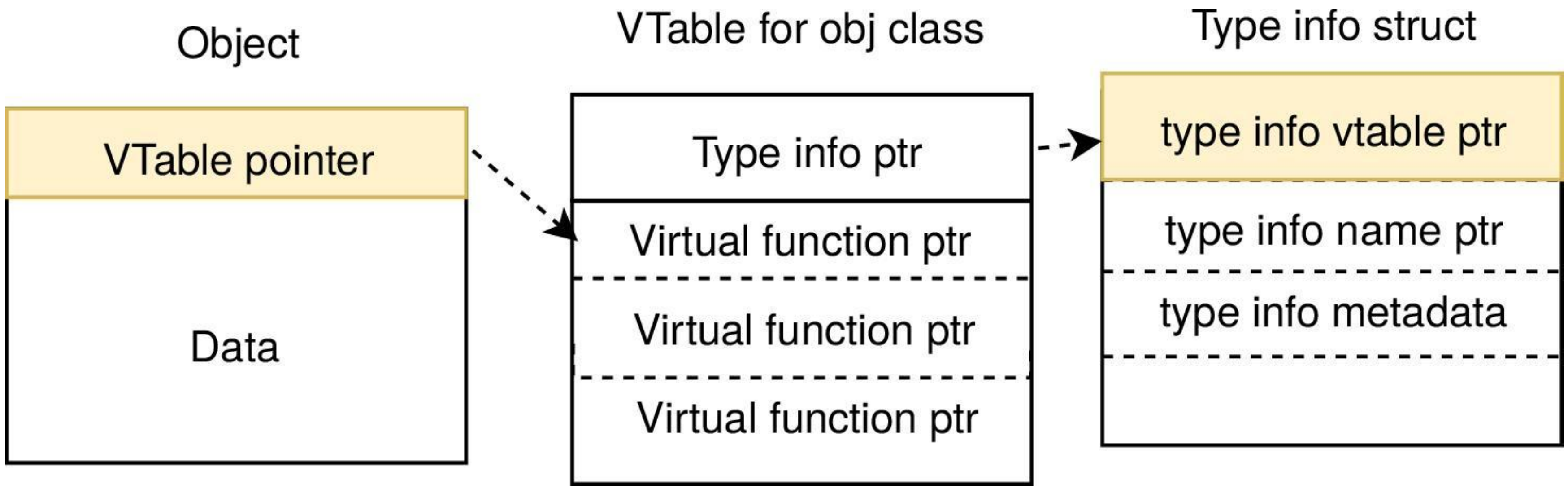
Introduction

- ❑ VTable Hijacking with Use-after-Free common attack
- ❑ Examples in Pwn2Own 2014/15/16 in Adobe Flash Player, Firefox, Chrome.

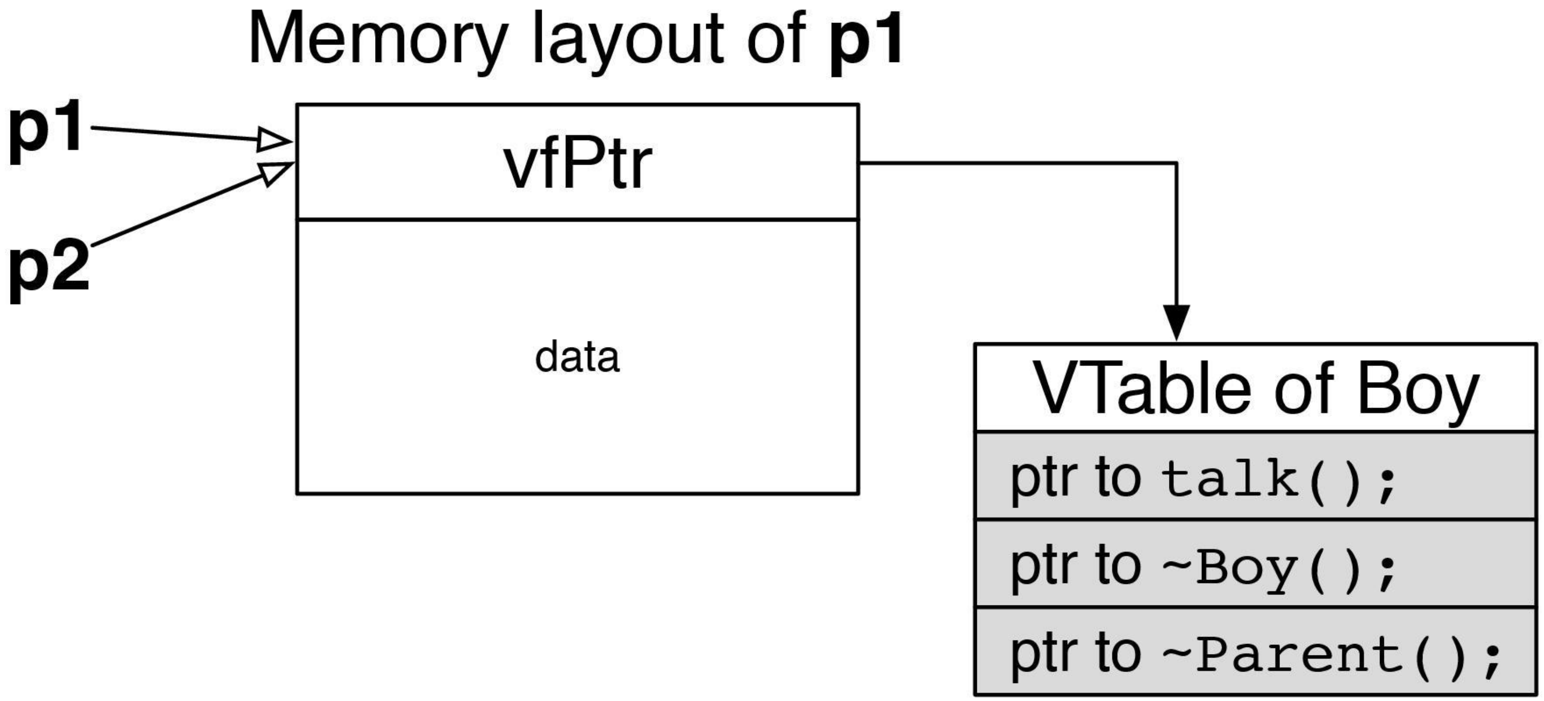
Overview

- ❑ VTPin transparently protects binaries without re-writing them.
- ❑ Applies to different OS’s and allocators
- ❑ LD_PRELOAD=”./libvtpin.so” my_cpp_binary

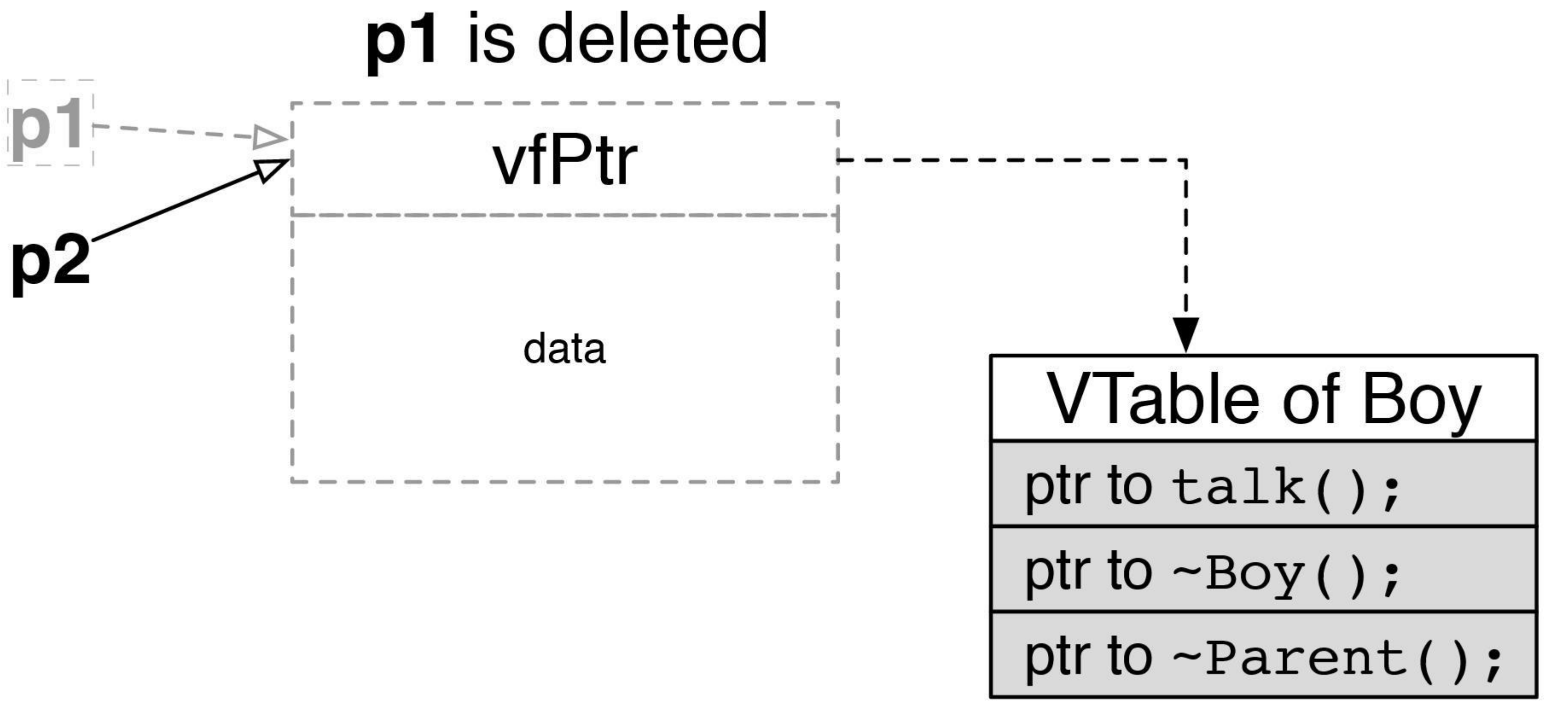
VTables in memory



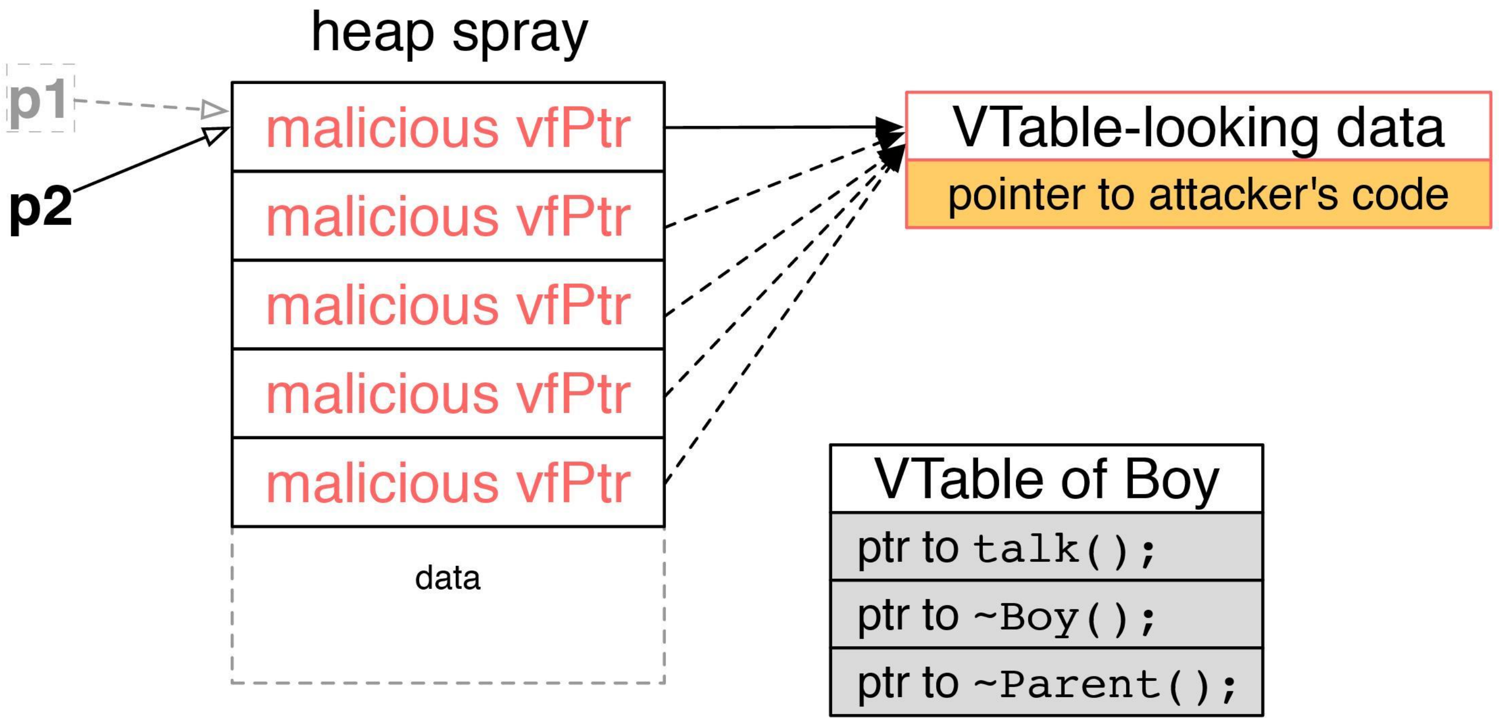
Use-after-free/Vtable Hijacking



Use-after-free/Vtable Hijacking



Use-after-free/Vtable Hijacking



What can we do

Prevent overwrite with malicious VT Ptr.

How:

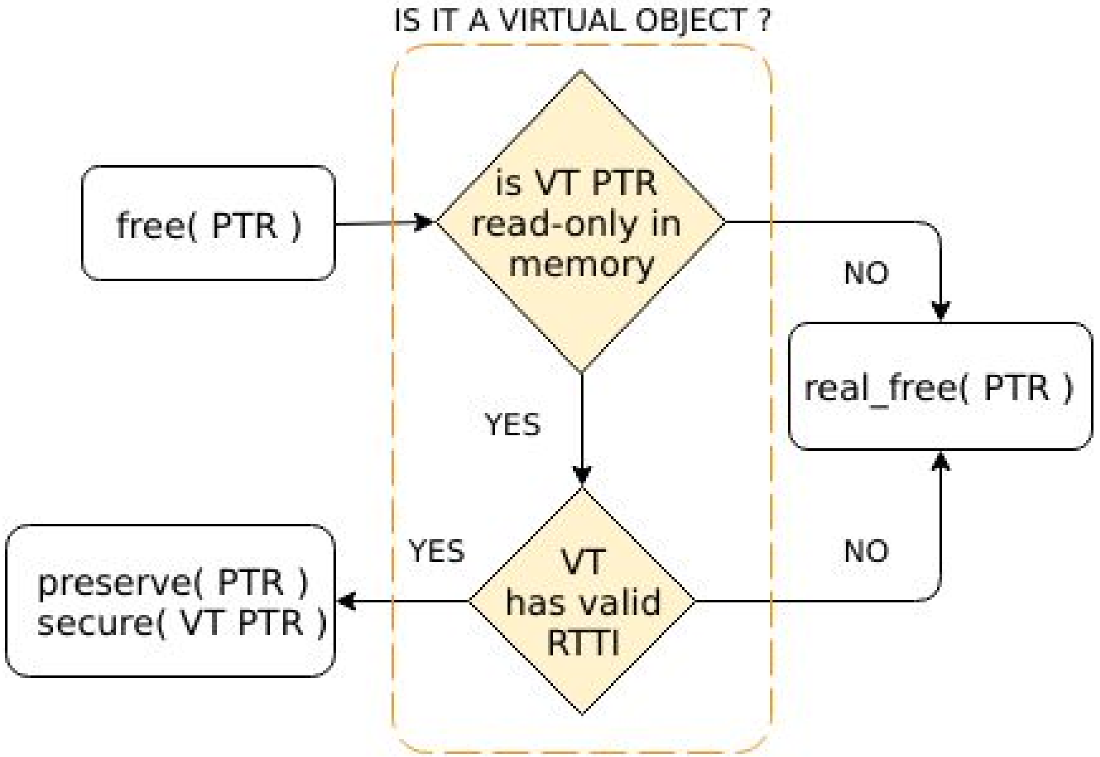
❏ Intercept free()
❏ If you're freeing a virtual c++ object, don't.

What do we know

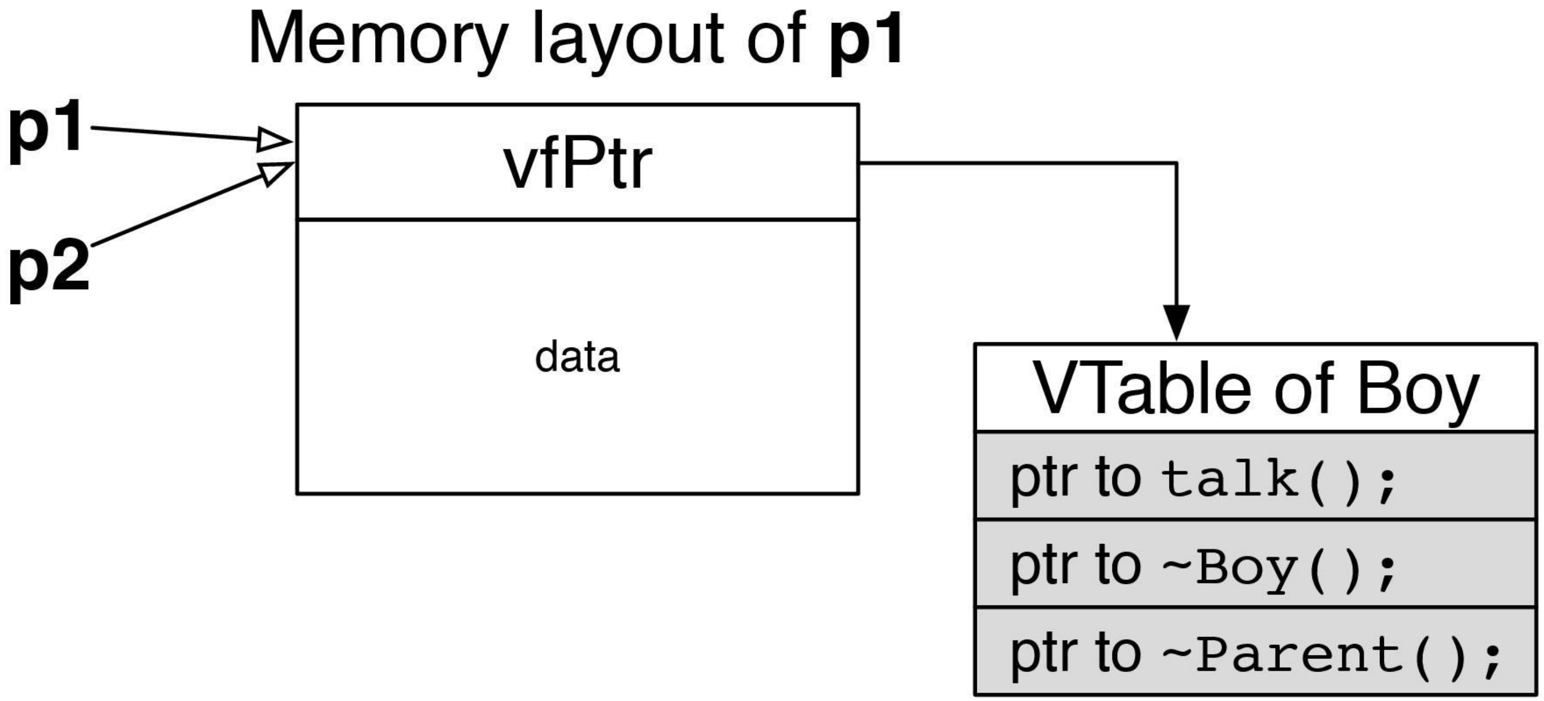
From Itanium C++ ABI :

- ❏ VTables are stored in Read-Only memory
- ❏ VTable Type Information has a specific structure

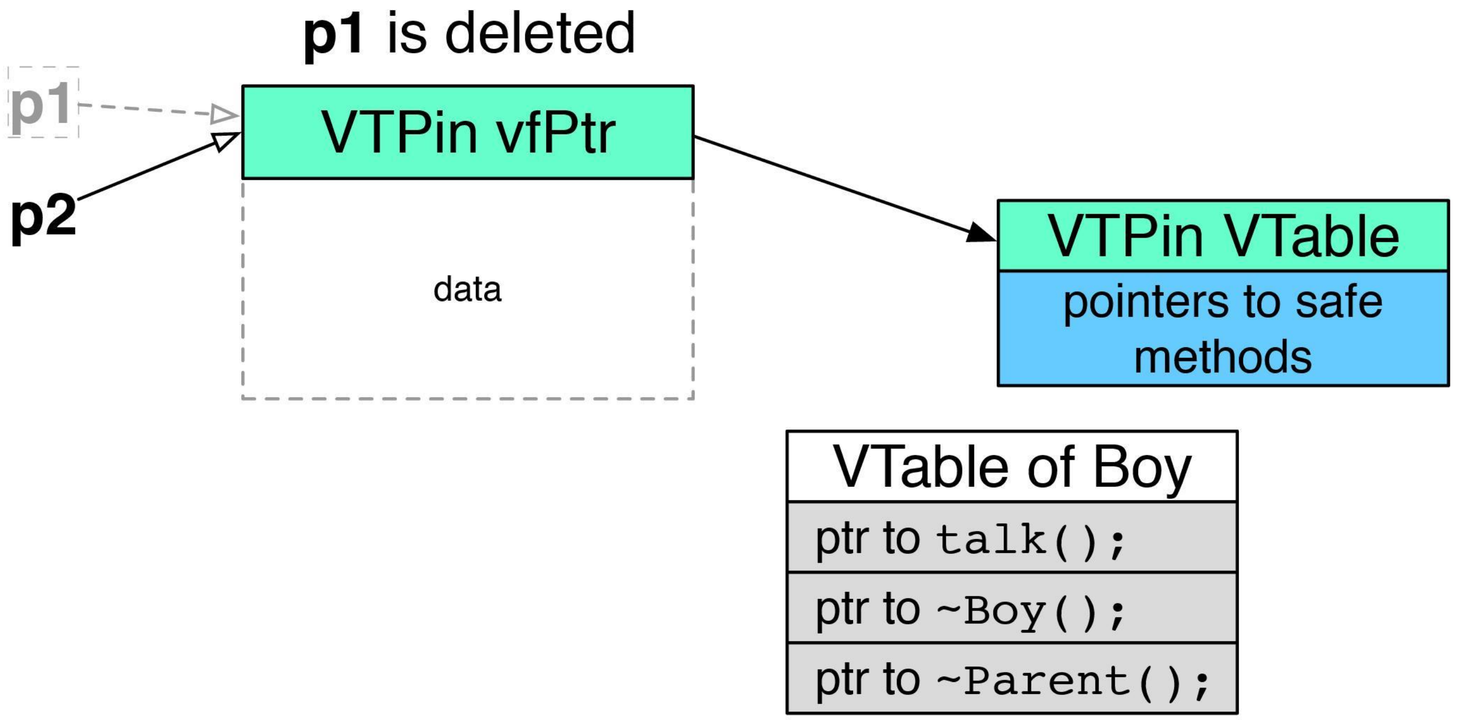
VTPin runthrough



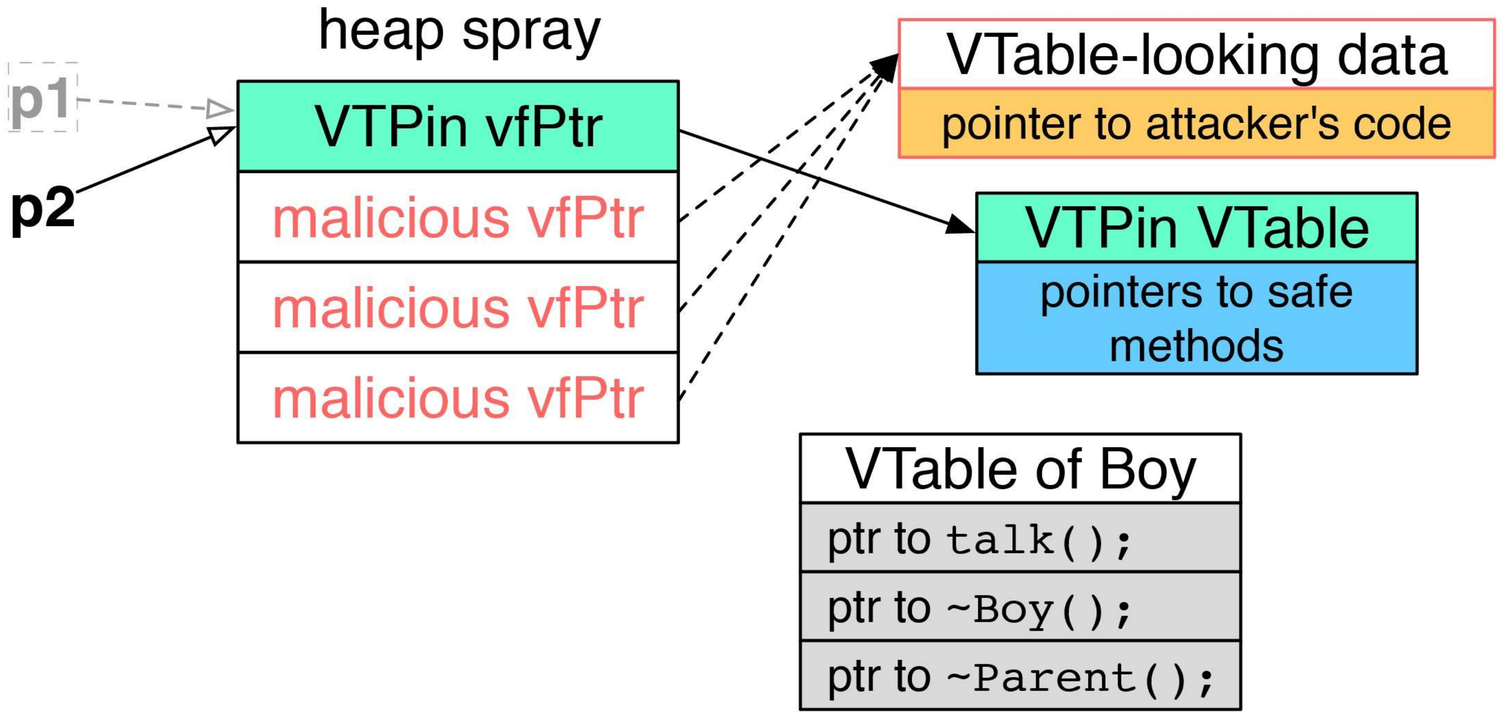
VTPin


Memory layout of **p1**
p1
p2
vfPtr
data
VTable of Boy
ptr to `talk();`
ptr to `~Boy();`
ptr to `~Parent();`

VTPin


p1 is deleted
p1
p2
VTPin vfPtr
data
VTPin VTable
pointers to safe methods
VTable of Boy
ptr to talk();
ptr to ~Boy();
ptr to ~Parent();

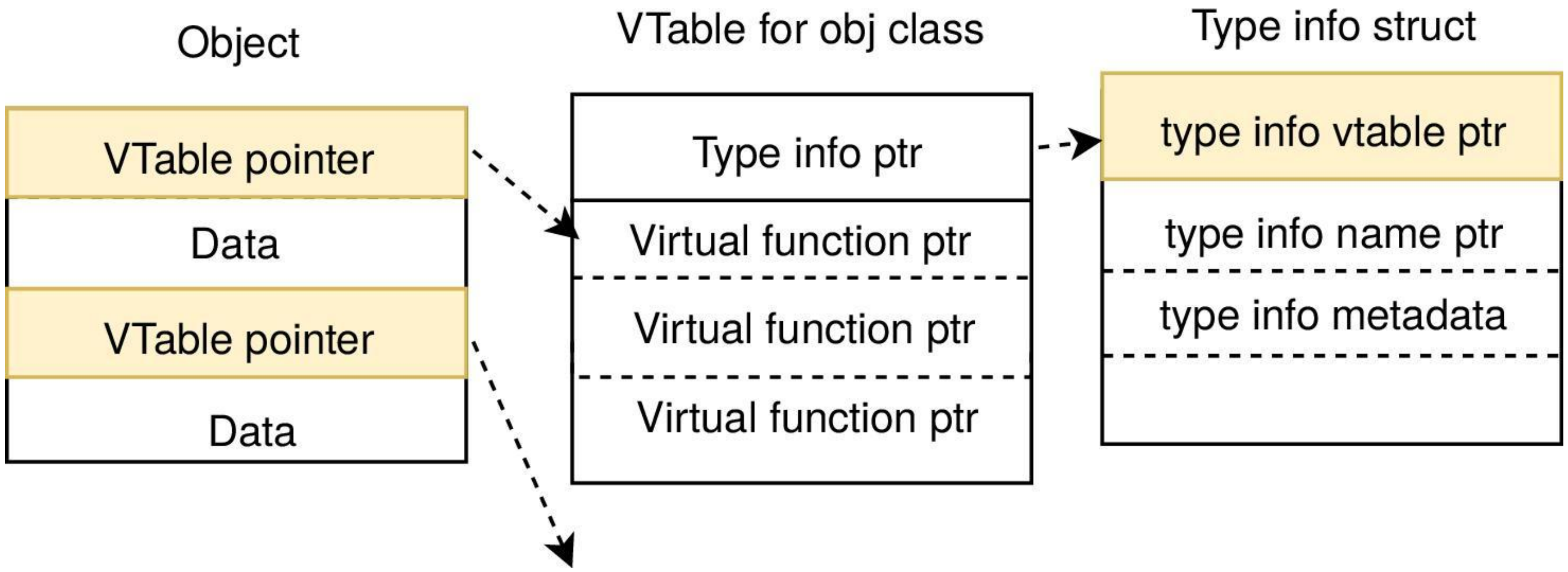
VTPin


heap spray
p1
p2
VTPin vfPtr
malicious vfPtr
malicious vfPtr
malicious vfPtr
VTable-looking data
pointer to attacker's code
VTPin VTable
pointers to safe methods
VTable of Boy
ptr to talk();
ptr to ~Boy();
ptr to ~Parent();

VTPin prerequisites

1. hooking free() and dl_open()
2. allocating memory with placement
3. RTTI (Run Time Type Information)
4. handling invalid memory accesses

Multiple inheritance and slab allocators



Handle invalid memory accesses

1. Use **segfaults** and handle in user-space
   a. Not ideal, has undefined behaviour
   b. Can create intermediate c++ objects

OR

2. Use **system calls** and let kernel handle it
   a. Lots of system calls on linux return EFAULT if accessing un-paged address (e.g. write, mincore)

Evaluation (performance)

Performance overhead: 0.7-4.9% on SPEC 2006, 0.3-4.1% on Firefox, 0.9-3.6% on Chromium. Only impacts free( ) function.

Tradeoff: Memory overhead. Depends on allocator. Can range from 0.2-4% or from 1-30% for slab allocators. Garbage collection is needed.

Evaluation (security)

Tested with:

- ❑ CVE-2013-1690 (Firefox v17.0)
- ❑ CVE-2011-0065 (Firefox v3.5)
- ❑ CVE-2013-0753 (Firefox v17.0.1)

VTPin

- ❑ Transparent to binary
- ❑ Lightweight
- ❑ Portable

Read more on
https://github.com/uberspot/VTPin