Сырые безопасные указатели. MiraclePtr

Александр Жиров
VK
https://t.me/Ciberst

MiraclePtr

1. Безопасность использования сырых указателей

MiraclePtr

1. Безопасность использования сырых указателей
2. В основном, для легаси проектов

MiraclePtr

1. Безопасность использования сырых указателей
2. В основном, для легаси проектов, которые хотят уменьшить количество возможных уязвимостей

MiraclePtr

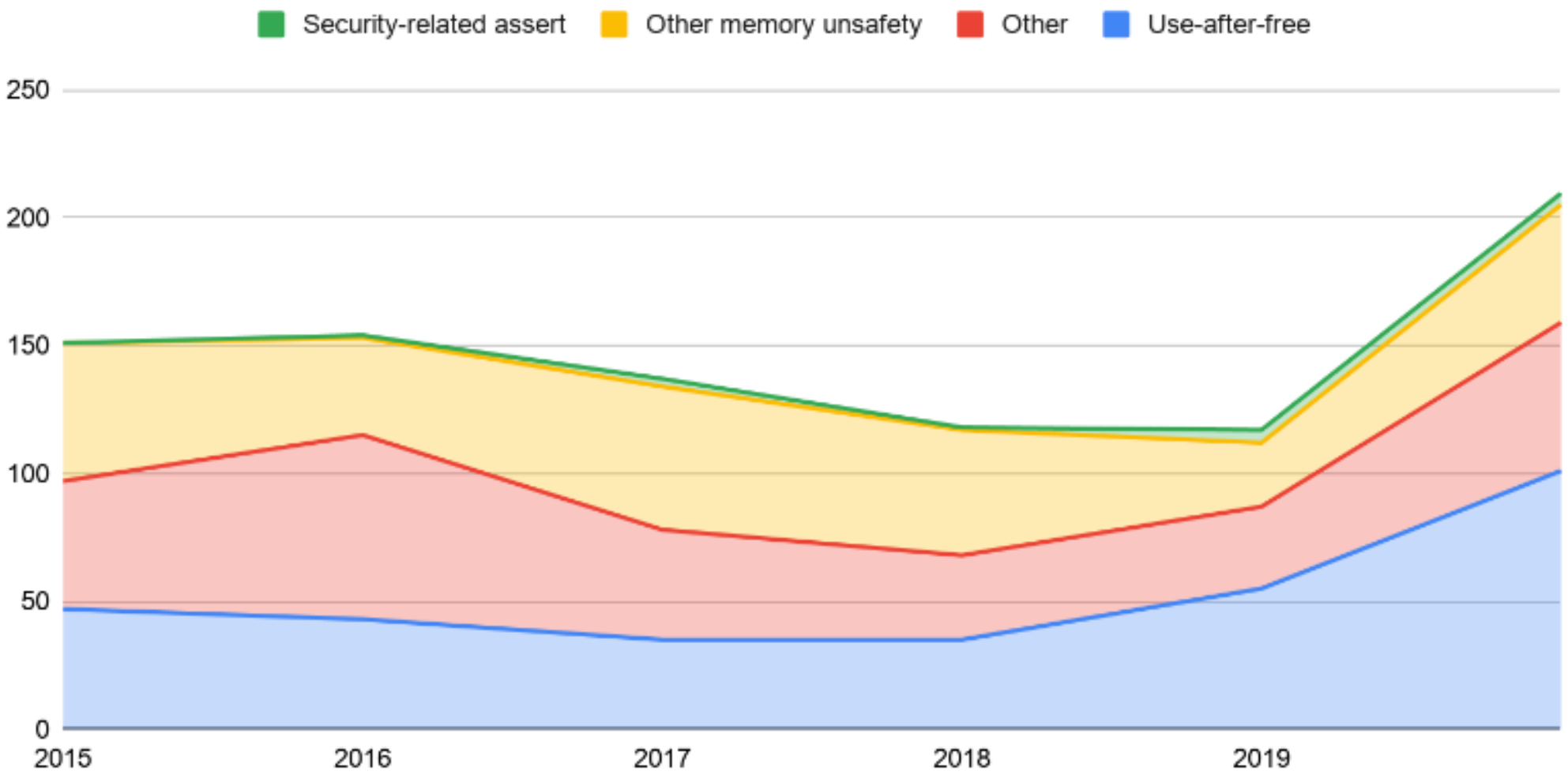
1. Безопасность использования сырых указателей
2. В основном, для легаси проектов, которые хотят уменьшить количество возможных уязвимостей
3. На примере Chromium

Кодовая база Chromium

Более миллиона строк кода на C++

Активно использовали сырые указатели

Что получили?

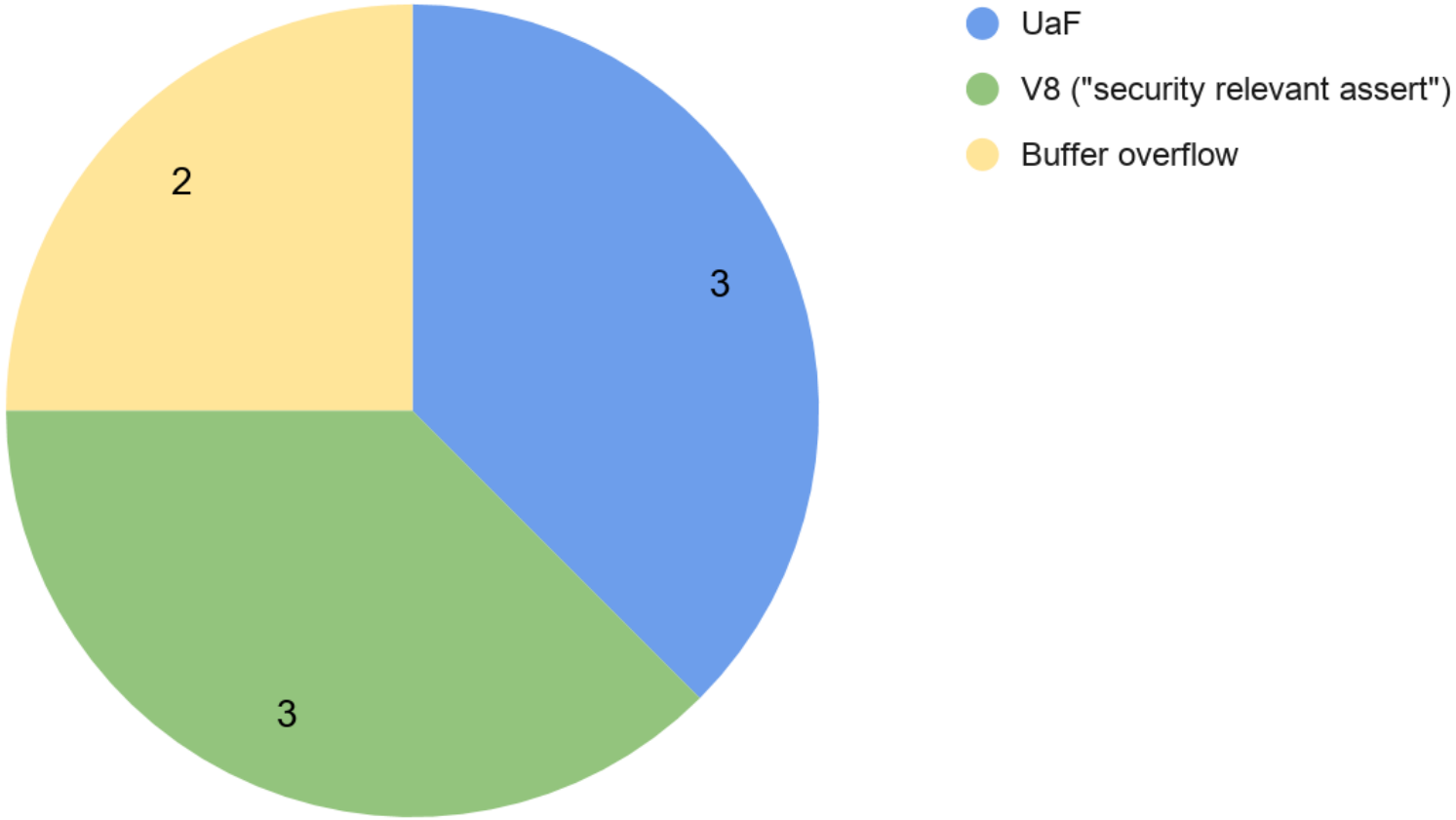


C++ и Use After Free (UaF)

Use after Free (Использование памяти после освобождения).

Обращение к памяти после ее освобождения может привести к сбою программы, использованию непредвиденных значений или выполнению произвольного кода.

Эксплоиты в Chrome 2019 – 2020



Chromium And Big Rewrite

2021 Nov 27 https://chromium-review.googlesource.com/c/chromium/src/+/3305132

+23727 -15551

8126 файлов было изменено

Большинство полей Foo* field_ были преобразованы в raw_ptr<Foo> field_

MiraclePtr

MiraclePtr – это проект внедрения **невладеющего** умного указателя.

MiraclePtr

В chromium именуется как **raw_ptr<T>,** ранее **CheckedPtr<T>** (*не путать с одноименным алгоритмом*).
Впервые появился в Blink (движок для отображения веб-страниц)

MiraclePtr

В chromium именуется как **raw_ptr<T>,** ранее **CheckedPtr<T>** (*не путать с одноименным алгоритмом*).
Впервые появился в Blink (движок для отображения веб-страниц)

Основная идея: **трансформация ошибок в крэши или утечки памяти**.

MiraclePtr. Подведем итоги

1. это невладеющий умный указатель.

MiraclePtr. Подведем итоги

1. это невладеющий умный указатель.
2. по прежнему приходится самостоятельно управлять временем жизни объекта

MiraclePtr. Подведем итоги

1. это невладеющий умный указатель.
2. по прежнему приходится самостоятельно управлять временем жизни объекта
3. работает более безопасно с памятью в отличие от обычных сырых указателей

MiraclePtr aka raw_ptr

Было:

```
WebContents* contents_ = nullptr;
```

MiraclePtr aka raw_ptr

Было:

```
WebContents* contents_ = nullptr;
```

Стало:

```
raw_ptr<WebContents> contents_ = nullptr;
```

MiraclePtr. Несовместимость с сырыми указателями

- `auto* raw_ptr_var = wrapped_ptr.get()`

MiraclePtr. Несовместимость с сырыми указателями

- `auto* raw_ptr_var = wrapped_ptr.get()`
- `return condition ? raw_ptr : wrapped_ptr.get();`

MiraclePtr. Несовместимость с сырыми указателями

- `auto* raw_ptr_var = wrapped_ptr.get()`
- `return condition ? raw_ptr : wrapped_ptr.get();`
- `raw_ptr<T> не поддерживает операцию взятие адреса`

MiraclePtr. Нет операции взятия адреса

```
void GetSomeClassPtr(SomeClass** out_arg) {
    *out_arg = ...;
}

struct MyStruct {
    void Example() {
        GetSomeClassPtr(&wrapped_ptr_);
    }

    raw_ptr<SomeClass> wrapped_ptr_;
};
```

MiraclePtr. Нет операции взятия адреса

```
void GetSomeClassPtr(SomeClass** out_arg) {
    *out_arg = ...;
}

struct MyStruct {
    void Example() {
        GetSomeClassPtr(&wrapped_ptr_);
    }

    raw_ptr<SomeClass> wrapped_ptr_;
};
```

MiraclePtr. Нет операции взятия адреса

```
void GetSomeClassPtr(raw_ptr<SomeClass>*
out_arg) {
    *out_arg = ...;
}
```

MiraclePtr

1. Алгоритм преобразования из сырого указателя в умный

MiraclePtr

1. Алгоритм преобразования из сырого указателя в умный
2. Аллокатор PartitionAlloc

MiraclePtr. Алгоритмы

Существует большое количество идей реализации MiraclePtr.

- **BackupRefPtr**
- **BorrowPtr**
- **CheckedPtr**
- **MTECheckedPtr**
- **SafePtr**
- **UnownedPtr**

MiraclePtr aka BorrowPtr

счетчик ссылок,

если больше 0 при освобождении => падение

MiraclePtr aka CheckedPtr

хранит tag

падает при разыменовании если tag не совпадает

MiraclePtr aka UnownedPtr

похож на WeakPtr

вызывает сбой при попытке разыменования уже освобождённого указателя.

MiraclePtr aka BackupRefPtr

Освобожденная память помещается в карантин до тех пор пока на нее есть ссылки.

На данный момент в кодовой базе Chromium используется данная идея.

BackupRefPtr. Псевдокод

```
template <typename T>
class BackupRefPtr {
    BackupRefPtr(T* ptr) : ptr_(ptr) {
        if (!isSupportedAllocation(ptr))
            return;

        atomic_int& ref_count = *(cast<atomic_int*>(ptr) - 1);
        CHECK(++ref_count);
    }
};
```

BackupRefPtr. Псевдокод

```
~BackupRefPtr() {
    if (!isSupportedAllocation(ptr_))
        return;

    atomic_int& ref_count = *(cast<atomic_int*>(ptr) - 1);
    if (--ref_count == 0)
        PartitionAlloc::ActuallyFree(ptr_);
}
```

BackupRefPtr. Псевдокод.

```
void* Alloc(size_t size) {
    void* ptr = ActuallyAlloc(size);
    if (isSupportedAllocation(ptr)) {
        int& ref_count = *(cast<int*>(ptr) - 1);
        ref_count = 1; }
    return ptr;
}
```

BackupRefPtr. Псевдокод.

```
void Free(void* ptr) {
    if (isSupportedAllocation(ptr)) {
        atomic_int& ref_count = *(cast<atomic_int*>(ptr) - 1);
        if (ref_count != 1)
            memset(ptr, 0xcc, getAllocationSize(ptr));
        if (--ref_count != 0)
            return;
    }

    ActuallyFree(ptr);
}
```

BackupRefPtr. Псевдокод.

```
void Free(void* ptr) {
    if (isSupportedAllocation(ptr)) {
        atomic_int& ref_count = *(cast<atomic_int*>(ptr) - 1);
        if (ref_count != 1)
            memset(ptr, 0xcc, getAllocationSize(ptr));
        if (--ref_count != 0)
            return;
    }

    ActuallyFree(ptr);
}
```

BackupRefPtr. Итоги

Потокобезопасный класс, предназначен для замены сырых указателей. Счетчик ссылок хранится в виде метаданных в аллокаторе PartitionAlloc.

Аллокатор PartitionAlloc

это аллокатор, прежде всего, нацеленный на решение проблем, связанных с безопасностью. Помимо это он содержит различные оптимизации, направленные на эффективное и быстрое использование пространства памяти.

Пример работы

```
base::PartitionAllocGlobalInit(HandleOOM);
base::PartitionAllocator allocator;
allocator.init(kOpts);

raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));

*P = 42;

allocator.root()->Free(p.get());


*p = 0; // CRASH
```

PartitionAlloc. Настройка

```
static constexpr base::PartitionOptions kOpts = {
    base::PartitionOptions::AlignedAlloc::kDisallowed,
    base::PartitionOptions::ThreadCache::kDisabled,
    base::PartitionOptions::Quarantine::kAllowed,
    base::PartitionOptions::Cookie::kAllowed,
    base::PartitionOptions::BackupRefPtr::kEnabled,
    base::PartitionOptions::UseConfigurablePool::kNo,
};
```

Пример работы

```
base::PartitionAllocGlobalInit(HandleOOM);
base::PartitionAllocator allocator;
allocator.init(kOpts);

raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));

*P = 42;

allocator.root()->Free(p.get());


*p = 0; // CRASH
```

Пример работы

```
base::PartitionAllocGlobalInit(HandleOOM);
base::PartitionAllocator allocator;
allocator.init(kOpts);

raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));

*P = 42;

allocator.root()->Free(p.get());


*p = 0; // CRASH
```

Аллокатор PartitionAlloc. Итоги

1. На каждую аллокация аллокатор PartitionAlloc обычно дополнительно выделяет 16 байтов или 8 байтов (4 байта для хранения счетчика ссылок, и оставшиеся на требования по выравниванию)
2. Освобожденная память помещается в “карантин” и недоступна для переиспользования
3. Включенная защита требует дополнительные разделы в PartitionAlloc, что увеличивает фрагментацию памяти.

PartitionAlloc Everywhere

это название проекта, который направлен на то, чтоб перехватывать malloc, free, realloc и т.д. и перенаправлять в PartitionAlloc.

PartitionAlloc используется только в Blink. Что создает некоторые неудобства, необходимо использовать api PartitionAlloc напрямую.

Пример работы с PartitionAlloc-Everywhere

```
raw_ptr<int> p = new int(42);
*p = 42;
delete p;
*p = 0; // CRASH
```

Tooling (инструменты для перехода)

Утилита rewrite_raw_ptr_fields

Основана clangAST

Автоматически переписывает **T*** → **raw_ptr<T>**, кроме тех файлов или директорий, которые добавлены в список игнорирования

MiraclePtr и PartitionAlloc. Больше примеров

```
{
  raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
  allocator.root()->Free(p.get());
  std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
}
std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
```

MiraclePtr и PartitionAlloc. Больше примеров

```
{
  raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
  allocator.root()->Free(p.get());
  std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
}
```

MiraclePtr и PartitionAlloc. Больше примеров

```
{
  raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
  allocator.root()->Free(p.get());
  std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
}
```

OUTPUT: 32

MiraclePtr и PartitionAlloc. Больше примеров

```
{
  raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
  allocator.root()->Free(p.get());
  std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
}
std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
```

MiraclePtr и PartitionAlloc. Больше примеров

```
{
  raw_ptr<int> p =
      reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
  allocator.root()->Free(p.get());
  std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
}
std::cout << allocator.root()->total_size_of_brp_quarantined_bytes;
```

OUTPUT: 0

MiraclePtr и PartitionAlloc. Больше примеров

```
raw_ptr<int> p =
    reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
allocator.root()->Free(p.get());

*p = 42;
```

MiraclePtr и PartitionAlloc. Больше примеров

```
raw_ptr<int> p =
    reinterpret_cast<int*>(allocator.root()->Alloc(sizeof(int), ""));
allocator.root()->Free(p.get());

*p = 42;
```

```
C:\Users\Ciberst\source\repos\chromium\src>out\Debug\cpprussia2022.exe
[0604/145509.595:FATAL:raw_ptr.h(394)] Check failed: IsPointeeAlive(address).
Backtrace:
```

MiraclePtr (BackupRefPtr). Вопросы производительности

Негативное влияние:

- Счетчик ссылок
- Фрагментация памяти с PartitionAlloc

В Chromium используется только в некоторых местах (Blink) путем прямого взаимодействия с PartitionAlloc API. Постепенное включение в других местах.

Итоги

https://chromereleases.googleblog.com/2022/05/stable-channel-update-for-desktop_24.html

24 мая 2022

[$TBD][1324864] **Critical** CVE-2022-1853: Use after free in Indexed DB. *Reported by Anonymous on 2022-05-12*
[$10000][1320024] **High** CVE-2022-1854: Use after free in ANGLE. *Reported by SeongHwan Park (SeHwa) on 2022-04-27*
[$7500][1228661] **High** CVE-2022-1855: Use after free in Messaging. *Reported by Anonymous on 2021-07-13*
[$3000][1323239] **High** CVE-2022-1856: Use after free in User Education. *Reported by Nan Wang(@eternalsakura13) and Guang Gong of 360 Alpha Lab on 2022-05-06*
[$1000][1322744] **High** CVE-2022-1859: Use after free in Performance Manager. *Reported by Guannan Wang (@Keenan7310) of Tencent Security Xuanwu Lab on 2022-05-05*
[$TBD][1297209] **High** CVE-2022-1860: Use after free in UI Foundations. *Reported by @ginggilBesel on 2022-02-15*
[$TBD][1316846] **High** CVE-2022-1861: Use after free in Sharing. *Reported by Khalil Zhani on 2022-04-16*
[$5000][1292870] **Medium** CVE-2022-1863: Use after free in Tab Groups. *Reported by David Erceg on 2022-02-01*
[$5000][1320624] **Medium** CVE-2022-1864: Use after free in WebApp Installs. *Reported by Yuntao You (@GraVity0) of Bytedance Wuheng Lab on 2022-04-28*
[$3000][1289192] **Medium** CVE-2022-1865: Use after free in Bookmarks. *Reported by Rong Jian of VRI on 2022-01-20*
[$3000][1292264] **Medium** CVE-2022-1866: Use after free in Tablet Mode. *Reported by @ginggilBesel on 2022-01-29*
[$TBD][1323236] **Medium** CVE-2022-1870: Use after free in App Service. *Reported by Nan Wang(@eternalsakura13) and Guang Gong of 360 Alpha Lab on 2022-05-06*

Спасибо за внимание!

Александр Жиров, программист