Can We Prevent Use-after-free Attacks?

2017/06/04

ssmjp Special

inaz2

About me

- @inaz2
  - https://twitter.com/inaz2

- Python programmer & Security engineer

- Blog: ももいろテクノロジー
  - http://inaz2.hatenablog.com/

Questions

- Are you a programmer?
- Can you do basic C++ programming?
- Do you know what use-after-free attack is?
- Do you know how to prevent the attack when you write your codes?

Important warning

- The purpose of this talk is to tell you how computers work, what the problem is and my current thoughts to prevent it

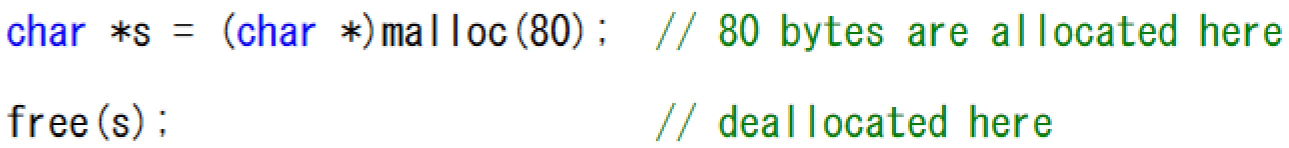
- NEVER ABUSE THE KNOWLEDGE YOU GET
  - Comply strictly with the law. Think about ethics. Never harm others definitely

- Some of Japanese laws about computer security
  - 不正アクセス行為の禁止等に関する法律（不正アクセス禁止法）
  - 刑法第168条の2及び3 不正指令電磁的記録に関する罪（ウイルス作成罪）
  - 刑法第234条の2 電子計算機損壊等業務妨害罪

Dynamic memory allocation and heap

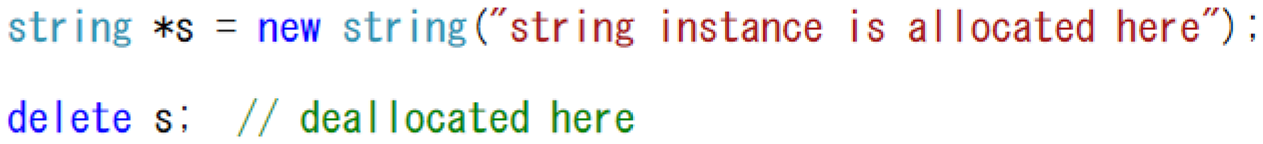
Dynamic memory allocation

- C: malloc() / free()



```
char *s = (char *)malloc(80);  // 80 bytes are allocated here
free(s);                       // deallocated here
```

- C++: new / delete



```
string *s = new string("string instance is allocated here");
delete s;  // deallocated here
```

- Win32API: HeapAlloc() / HeapFree()

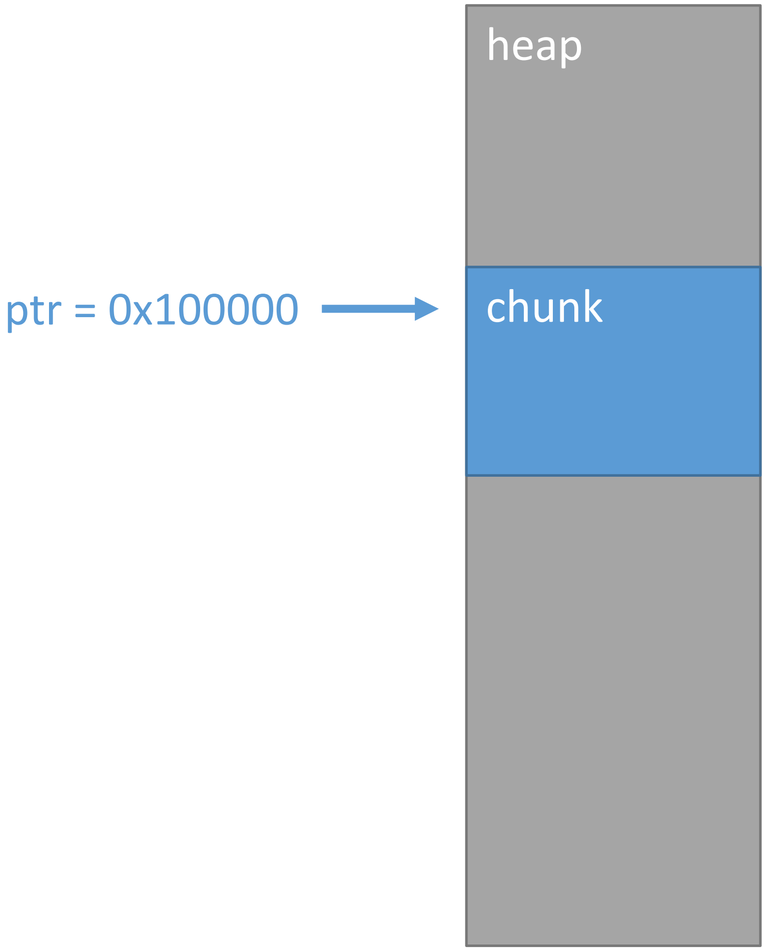
Heap

- First allocate a pool of memory
- malloc(): Use some of the pool as a *chunk*
- free(): Give it back
- Freed chunks are generally reused for efficiency

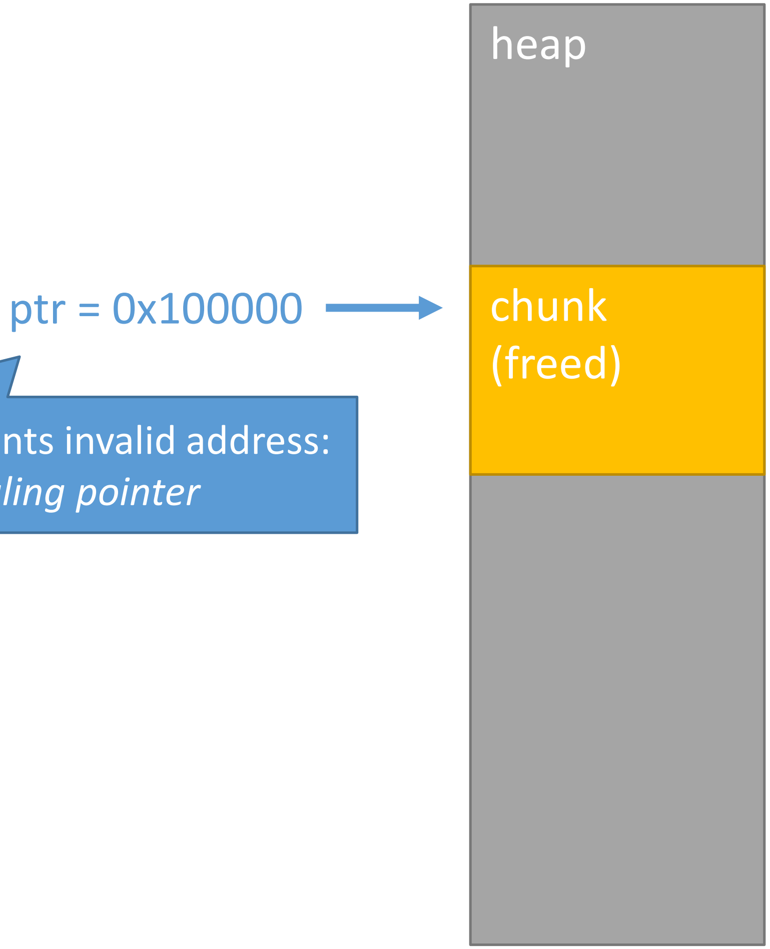
https://commons.wikimedia.org/wiki/File:Tannin_heap.jpeg

Use-after-free attacks

Allocate a chunk

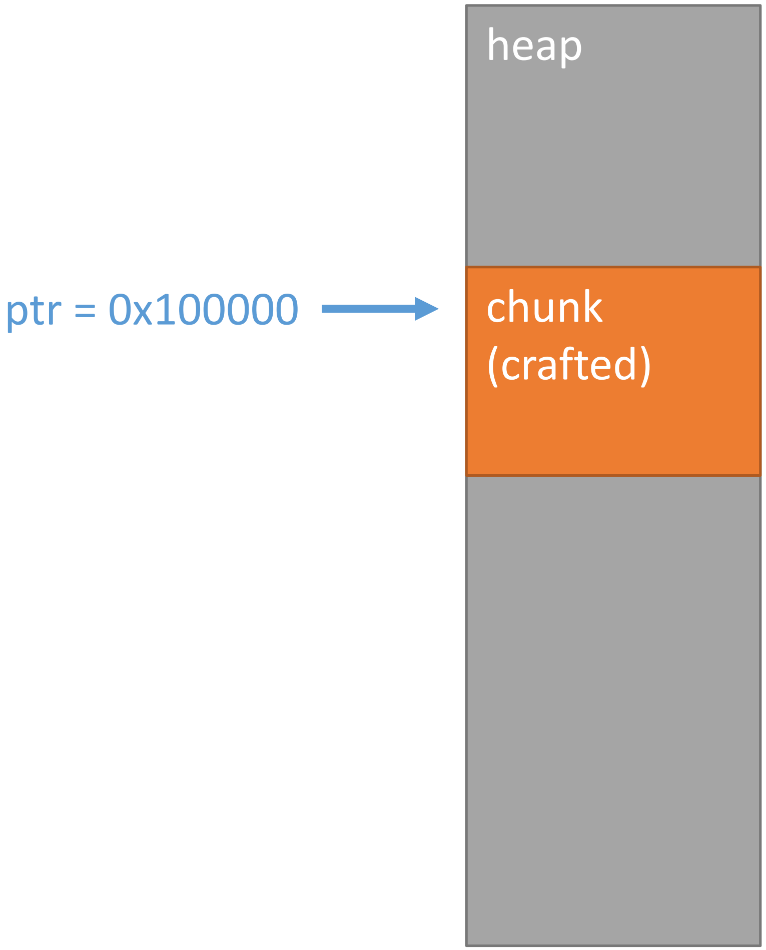


Free the chunk (or trigger garbage collection)

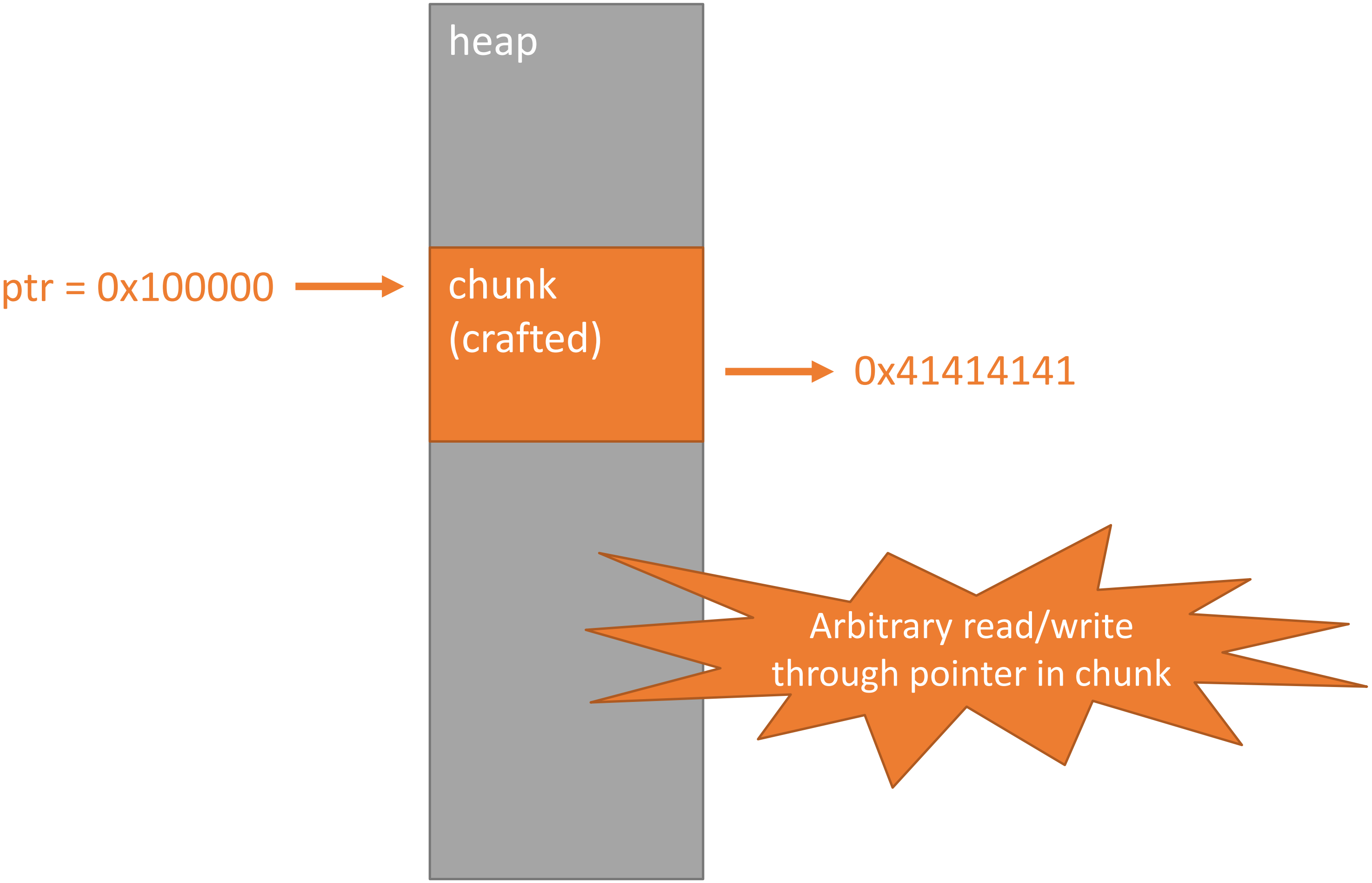


Now this points invalid address: *dangling pointer*

Reallocate the freed chunk with crafted data



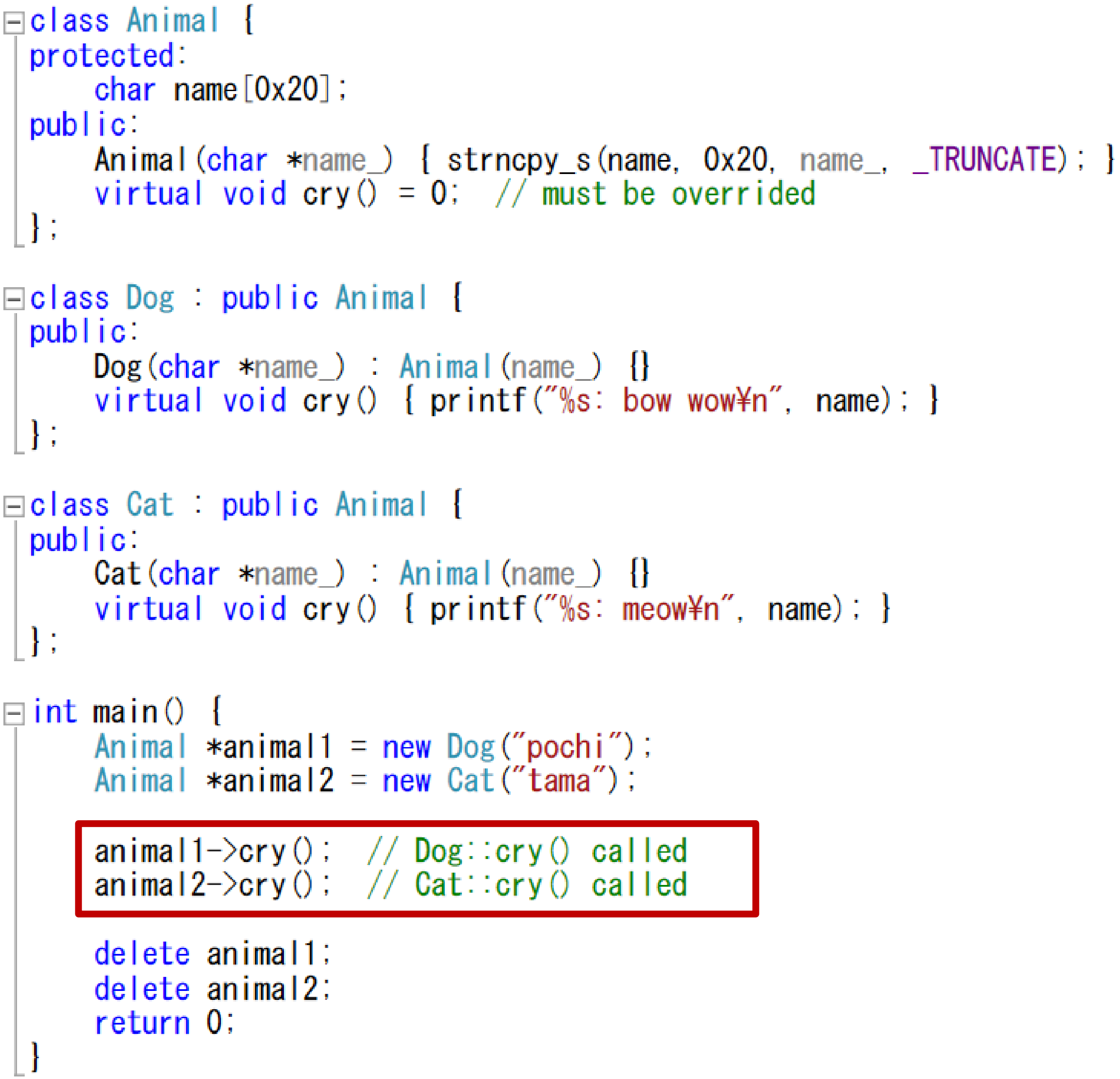
Access freed memory via dangling pointer



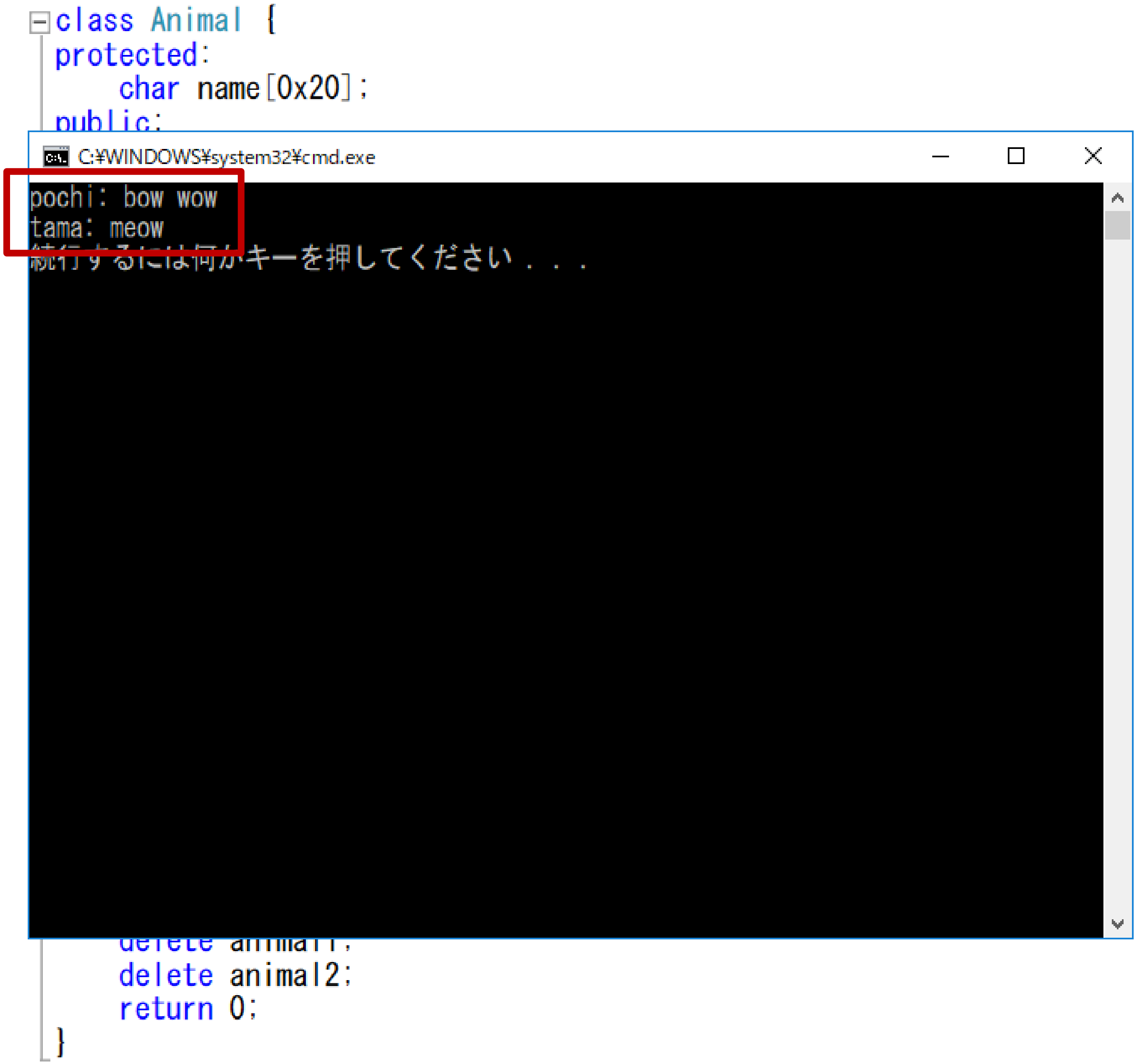
Virtual function table

Virtual function

- Handle derived classes as their base class and call functions of the former in the same manner
  - This characteristics is called “Polymorphism”



```
class Animal {
protected:
    char name[0x20];
public:
    Animal(char *name_) { strncpy_s(name, 0x20, name_, _TRUNCATE); }
    virtual void cry() = 0;  // must be overrided
};

class Dog : public Animal {
public:
    Dog(char *name_) : Animal(name_) {}
    virtual void cry() { printf("%s: bow wow¥n", name); }
};

class Cat : public Animal {
public:
    Cat(char *name_) : Animal(name_) {}
    virtual void cry() { printf("%s: meow¥n", name); }
};

int main() {
    Animal *animal1 = new Dog("pochi");
    Animal *animal2 = new Cat("tama");

    animal1->cry();  // Dog::cry() called
    animal2->cry();  // Cat::cry() called

    delete animal1;
    delete animal2;
    return 0;
}
```



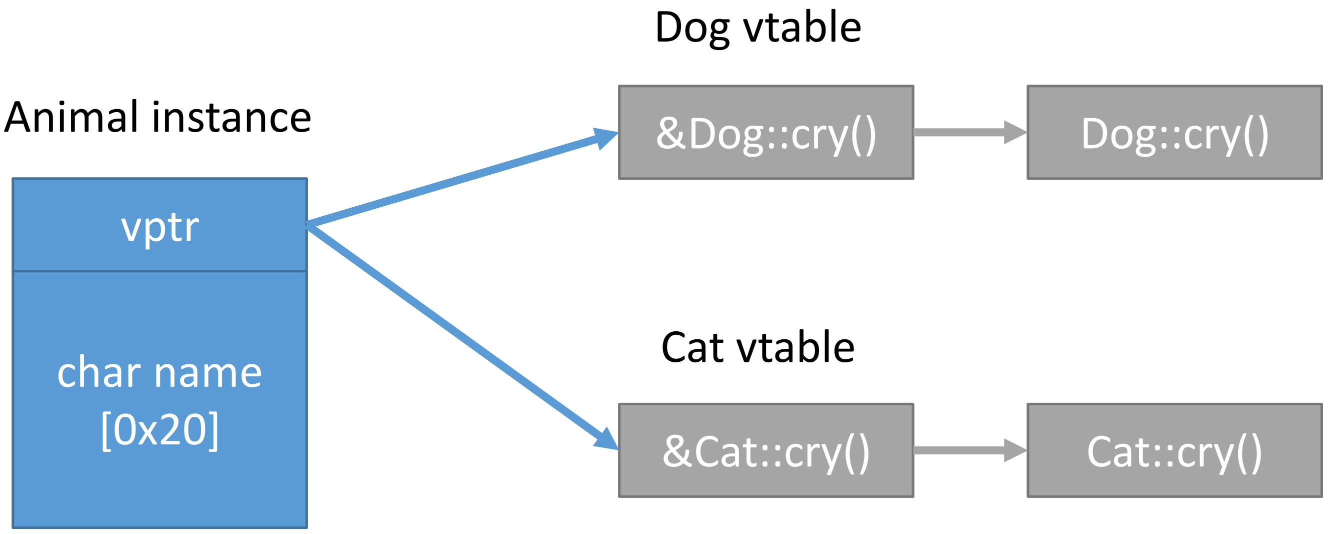
```
class Animal {
protected:
    char name[0x20];
public:
```

C:¥WINDOWS¥system32¥cmd.exe

```
pochi: bow wow
tama: meow
続行するには何かキーを押してください . . .
```

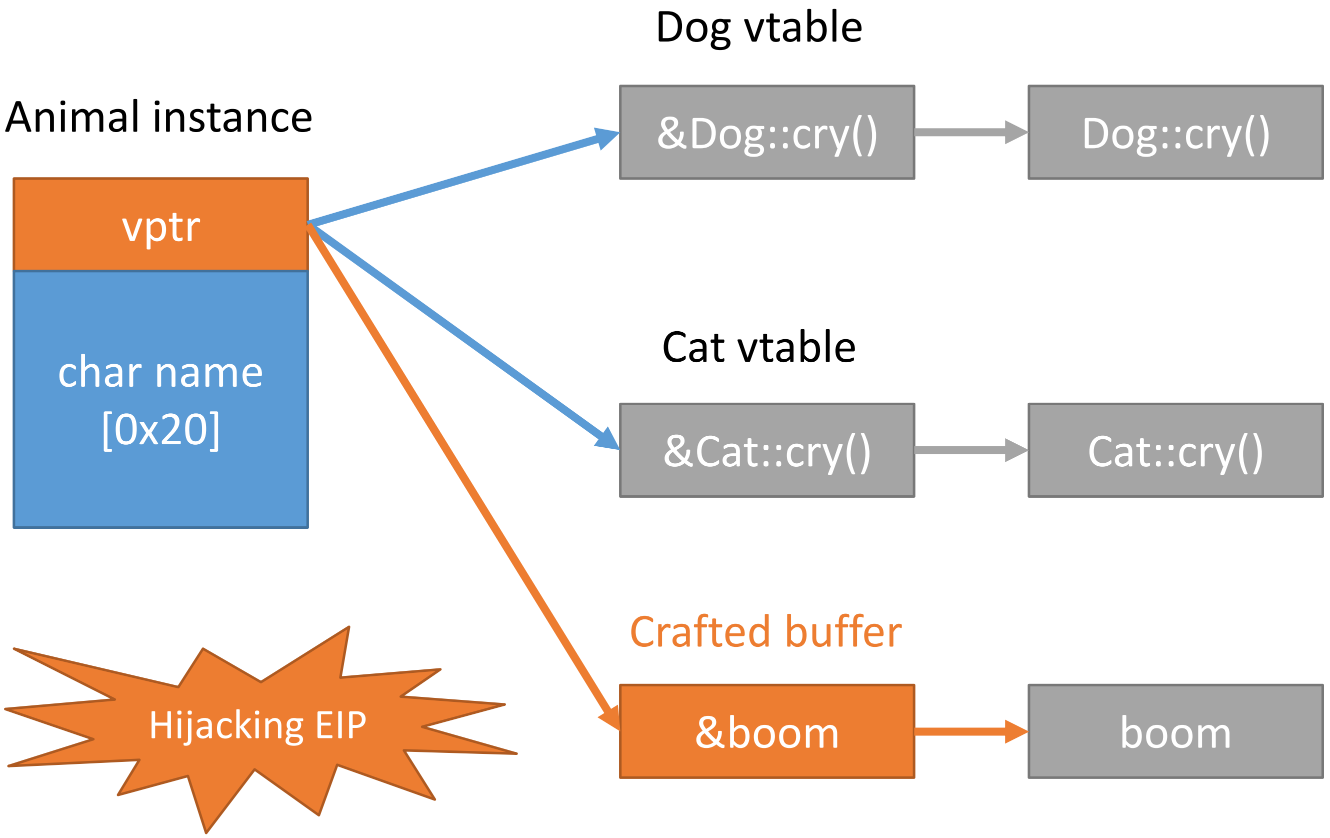
```
    delete animal1;
    delete animal2;
    return 0;
}
```

Virtual function table (vtable)



* gray indicates read only

Virtual function table (vtable)



* gray indicates read only

On Linux …

- Test program on glibc’s ptmalloc
  - https://gist.github.com/inaz2/f24f6d09cbf406d344debc649106fd89

Even on Windows 10 …

- Test program (unprotected) on low fragmentation heap
  - https://gist.github.com/inaz2/0c6edd5f485b95a114104fd48a533750

When will it be a problem?

- When user can call functions in arbitrary order
  - Parser / command interpreter
  - Script engine (especially JavaScript)
  - OS kernel / drivers

- Security Vulnerabilities Related To CWE-416 - CVE Details
  - http://www.cvedetails.com/vulnerability-list/cweid-416/vulnerabilities.html

- Published Advisories 2017 - Zero Day Initiative
  - http://www.zerodayinitiative.com/advisories/published/2017/

Zero-day vulnerabilities

- CVE-2014-1776 (Remote code execution)
  - Use-after-free vulnerability in MSHTML!CMarkup::IsConnectedToPrimaryMarkup() in Microsoft Internet Explorer 6 through 11
  - https://www.fireeye.com/blog/threat-research/2014/04/new-zero-day-exploit-targeting-internet-explorer-versions-9-through-11-identified-in-targeted-attacks.html

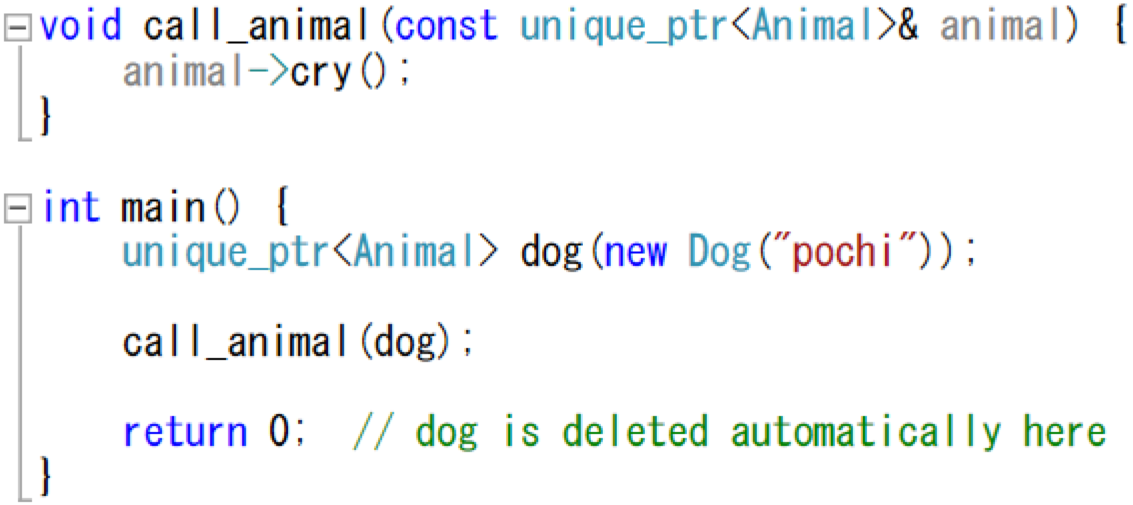
- CVE-2017-0263 (Local privilege escalation)
  - Use-after-free vulnerability in win32k!xxxDestroyWindow() in Microsoft Windows 7, 8.1, 10 and Windows Server 2008, 2012, 2016
  - https://www.fireeye.com/blog/threat-research/2017/05/eps-processing-zero-days.html

Preventing use-after-free attacks

C++11: Using smart pointers and references

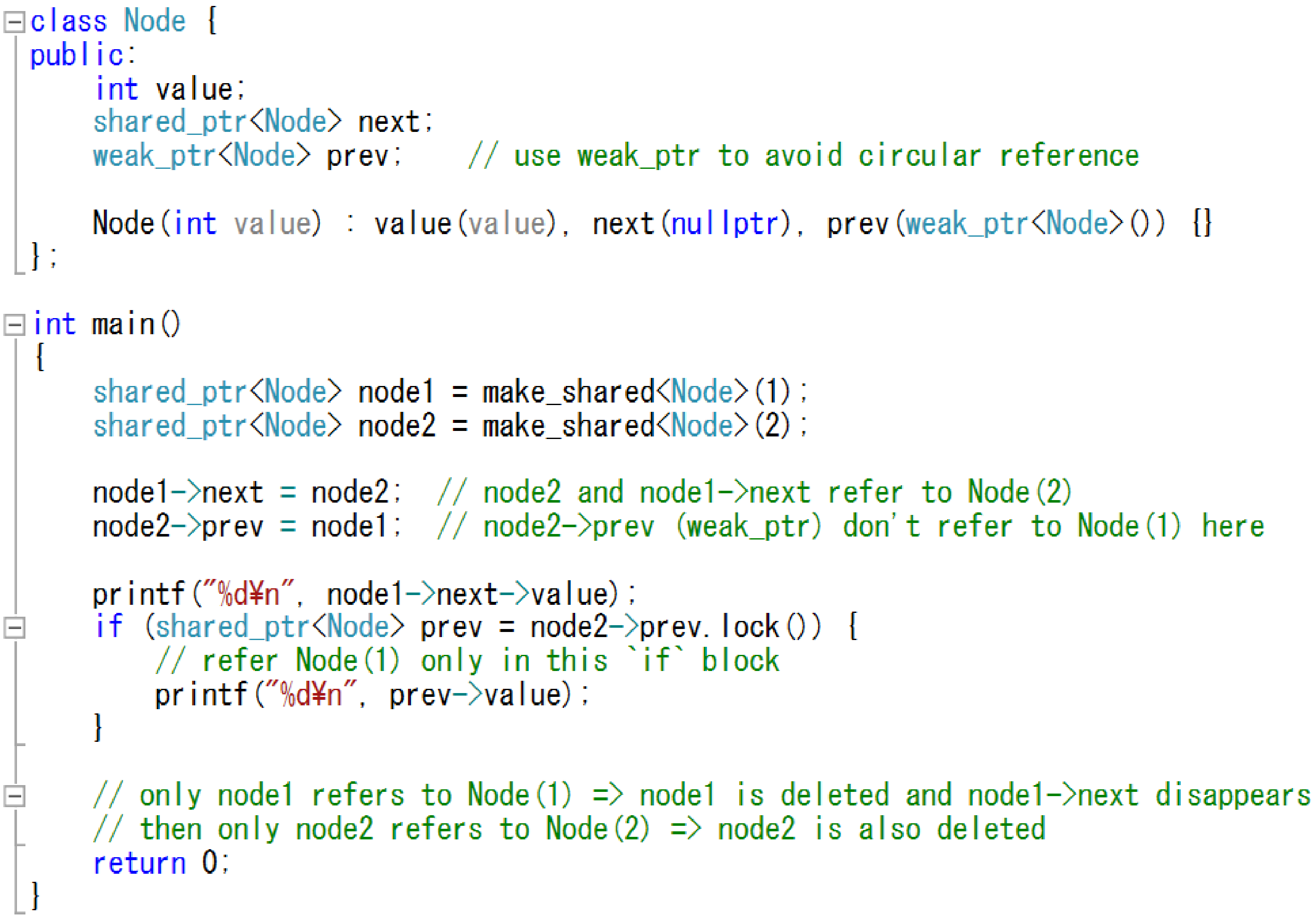
- Smart Pointers (Modern C++)
  - https://msdn.microsoft.com/en-us/library/hh279674.aspx
- #include <memory>
- unique_ptr<T> (pointer with ownership)
  - when the object is pointed by only one pointer
- shared_ptr<T> (pointer with reference count)
  - when the object is pointed by more than one pointers
  - To avoid circular reference, weak_ptr<T> is used together

The unique pointer across functions



```
void call_animal(const unique_ptr<Animal>& animal) {
    animal->cry();
}

int main() {
    unique_ptr<Animal> dog(new Dog("pochi"));

    call_animal(dog);

    return 0;  // dog is deleted automatically here
}
```

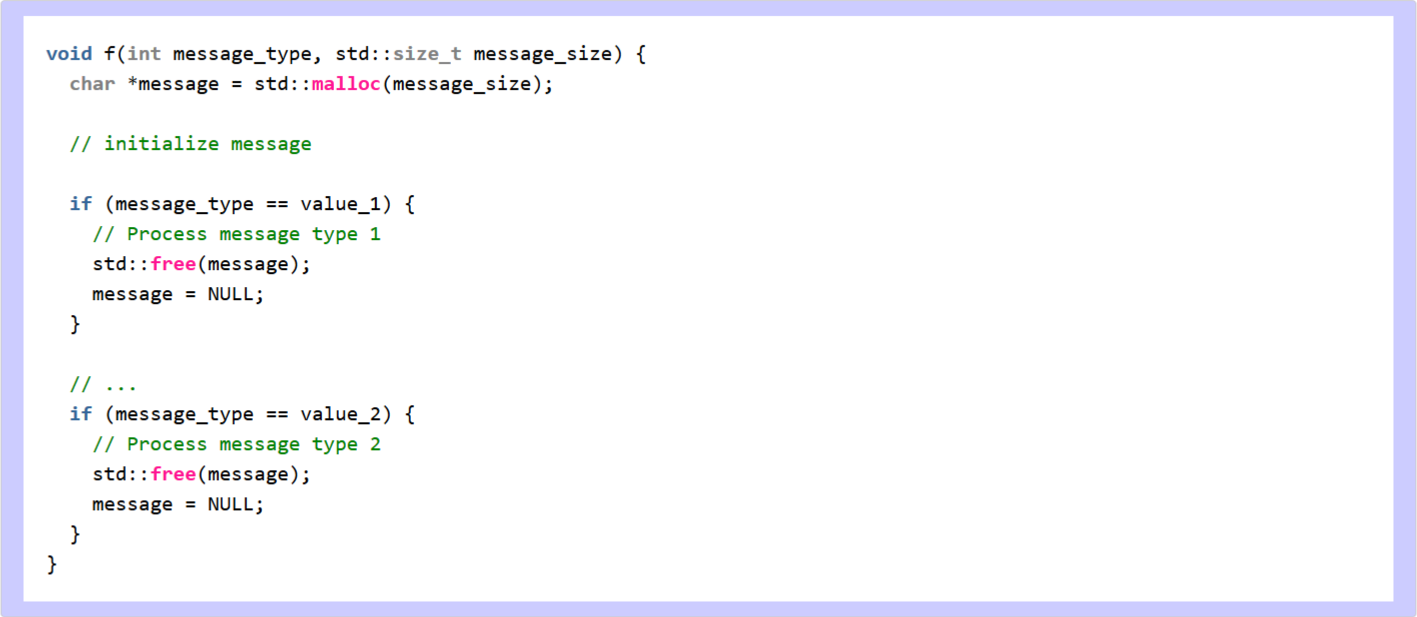
Bidirectional linked list by using shared_ptr



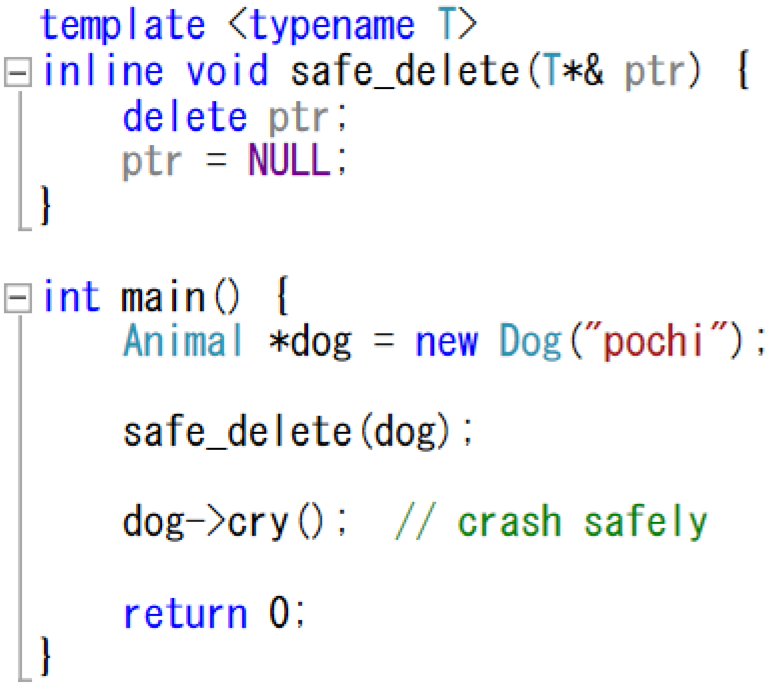
```
class Node {
public:
    int value;
    shared_ptr<Node> next;
    weak_ptr<Node> prev;    // use weak_ptr to avoid circular reference

    Node(int value) : value(value), next(nullptr), prev(weak_ptr<Node>())  {}
};

int main()
{
    shared_ptr<Node> node1 = make_shared<Node>(1);
    shared_ptr<Node> node2 = make_shared<Node>(2);

    node1->next = node2;  // node2 and node1->next refer to Node(2)
    node2->prev = node1;  // node2->prev (weak_ptr) don't refer to Node(1) here

    printf(”%d¥n”, node1->next->value);
    if (shared_ptr<Node> prev = node2->prev.lock()) {
        // refer Node(1) only in this `if` block
        printf(”%d¥n”, prev->value);
    }

    // only node1 refers to Node(1) => node1 is deleted and node1->next disappears
    // then only node2 refers to Node(2) => node2 is also deleted
    return 0;
}
```

Assigning NULL immediately after free

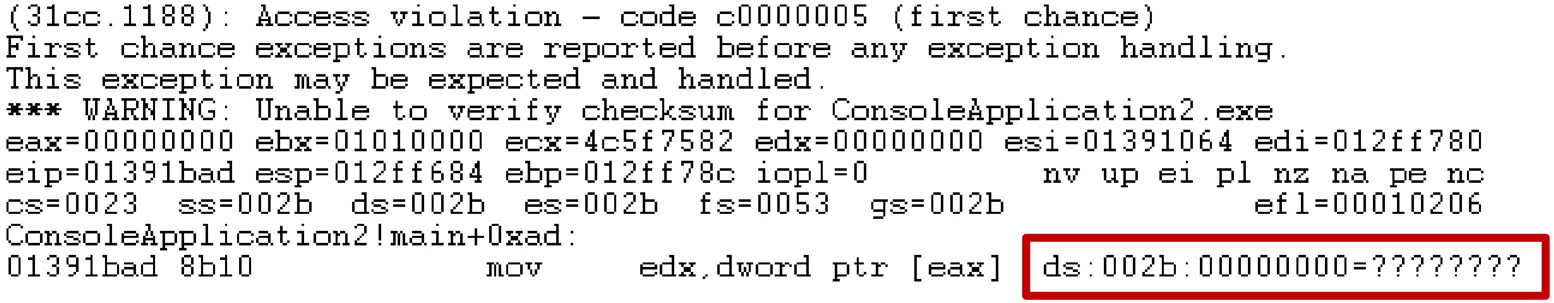
- MEM01-CPP. Store a valid value in pointers immediately after deallocation - SEI CERT C++ Coding Standard **[outdated]**
  - https://www.securecoding.cert.org/confluence/display/cplusplus/MEM01-CPP.+Store+a+valid+value+in+pointers+immediately+after+deallocation



```
void f(int message_type, std::size_t message_size) {
  char *message = std::malloc(message_size);

  // initialize message

  if (message_type == value_1) {
    // Process message type 1
    std::free(message);
    message = NULL;
  }

  // ...
  if (message_type == value_2) {
    // Process message type 2
    std::free(message);
    message = NULL;
  }
}
```



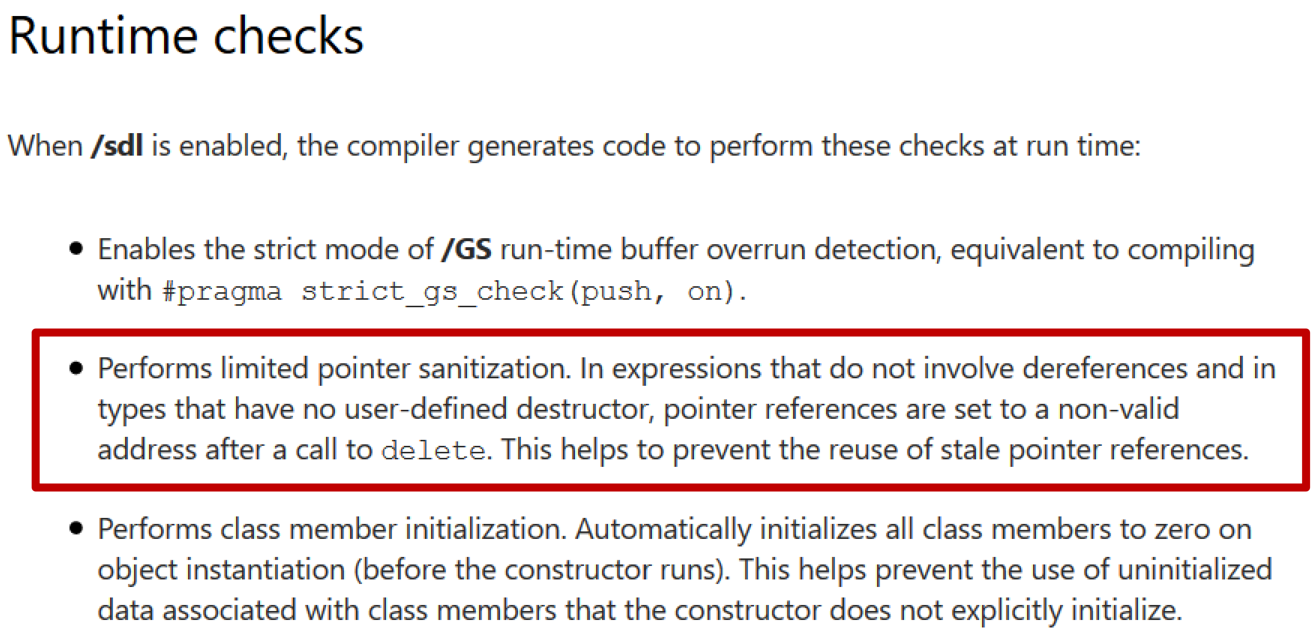
```cpp
template <typename T>
inline void safe_delete(T*& ptr) {
    delete ptr;
    ptr = NULL;
}

int main() {
    Animal *dog = new Dog("pochi");

    safe_delete(dog);

    dog->cry();  // crash safely

    return 0;
}
```



```
(31cc.1188): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** WARNING: Unable to verify checksum for ConsoleApplication2.exe
eax=00000000 ebx=01010000 ecx=4c5f7582 edx=00000000 esi=01391064 edi=012ff780
eip=01391bad esp=012ff684 ebp=012ff78c iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
ConsoleApplication2!main+0xad:
01391bad 8b10            mov     edx,dword ptr [eax]  ds:002b:00000000=????????
```

Microsoft Visual C++: enable SDL checks

- /sdl (Enable Additional Security Checks)
  - https://msdn.microsoft.com/en-us/library/jj161081.aspx



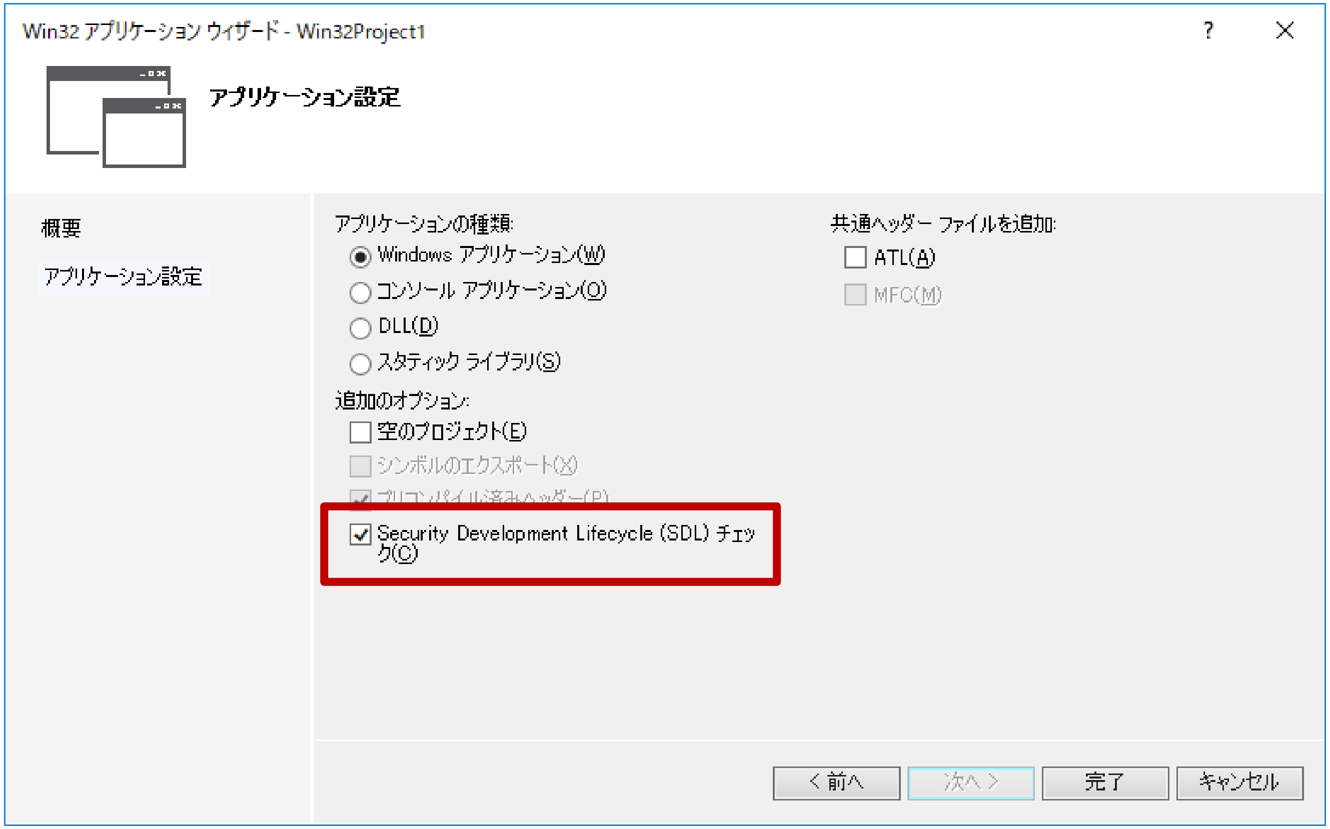
Runtime checks

When **/sdl** is enabled, the compiler generates code to perform these checks at run time:

- Enables the strict mode of **/GS** run-time buffer overrun detection, equivalent to compiling with `#pragma strict_gs_check(push, on)`.
- Performs limited pointer sanitization. In expressions that do not involve dereferences and in types that have no user-defined destructor, pointer references are set to a non-valid address after a call to `delete`. This helps to prevent the reuse of stale pointer references.
- Performs class member initialization. Automatically initializes all class members to zero on object instantiation (before the constructor runs). This helps prevent the use of uninitialized data associated with class members that the constructor does not explicitly initialize.

Microsoft Visual C++: enable SDL checks

- Enabled by default in Visual Studio 2015 “New Win32 Project”



Win32 アプリケーション ウィザード - Win32Project1

アプリケーション設定

アプリケーションの種類:

● Windows アプリケーション(W)

○ コンソール アプリケーション(O)

○ DLL(D)

○ スタティック ライブラリ(S)

追加のオプション:

□ 空のプロジェクト(E)

□ シンボルのエクスポート(X)

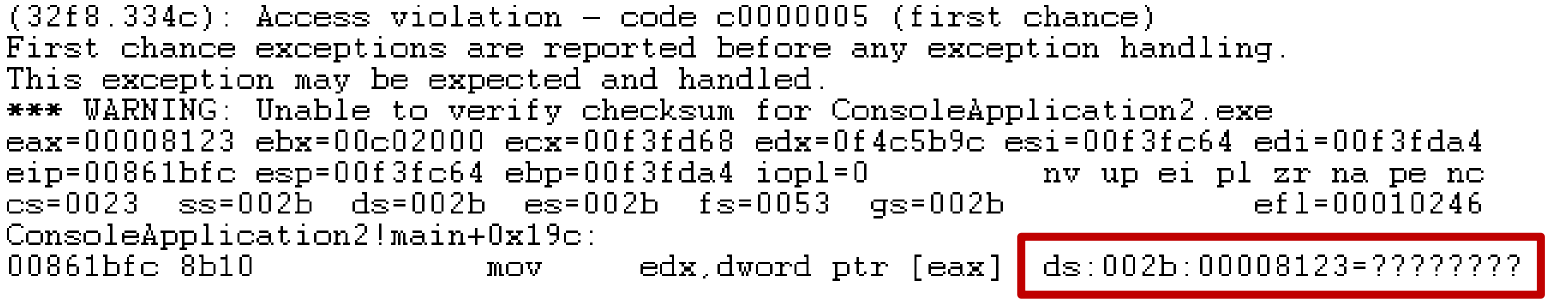
☑ プリコンパイル済みヘッダー(P)

☑ Security Development Lifecycle (SDL) チェック(C)

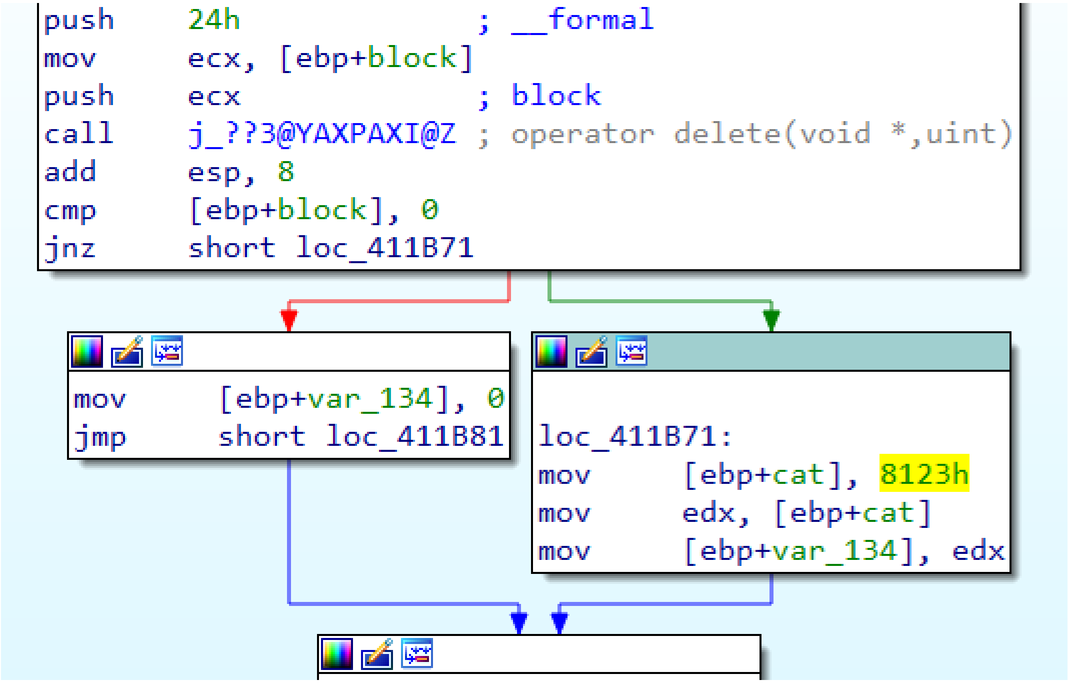
共通ヘッダー ファイルを追加:

□ ATL(A)

□ MFC(M)



```
(32f8.334c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** WARNING: Unable to verify checksum for ConsoleApplication2.exe
eax=00008123 ebx=00c02000 ecx=00f3fd68 edx=0f4c5b9c esi=00f3fc64 edi=00f3fda4
eip=00861bfc esp=00f3fc64 ebp=00f3fda4 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
ConsoleApplication2!main+0x19c:
00861bfc 8b10            mov     edx,dword ptr [eax]  ds:002b:00008123=????????
```



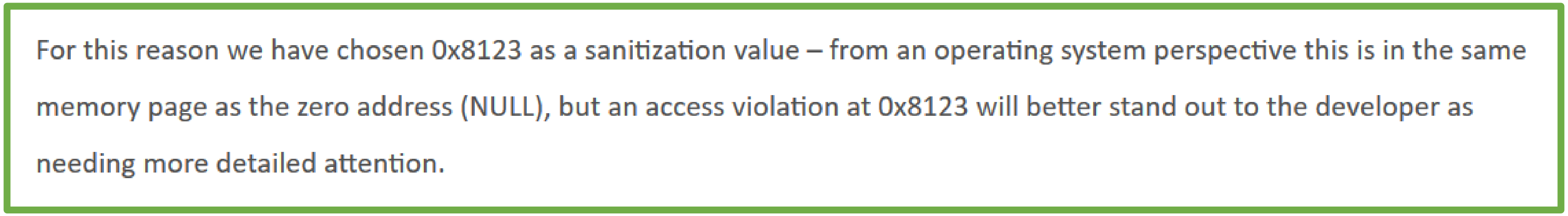
```
push    24h             ; __formal
mov     ecx, [ebp+block]
push    ecx             ; block
call    j_??3@YAXPAXI@Z ; operator delete(void *,uint)
add     esp, 8
cmp     [ebp+block], 0
jnz     short loc_411B71
```

```
mov     [ebp+var_134], 0
jmp     short loc_411B81
```

```
loc_411B71:
mov     [ebp+cat], 8123h
mov     edx, [ebp+cat]
mov     [ebp+var_134], edx
```

Magic address ds:002b:00008123

- Guarding against re-use of stale object references | Microsoft Secure Blog
  - https://blogs.microsoft.com/microsoftsecure/2012/04/24/guarding-against-re-use-of-stale-object-references/



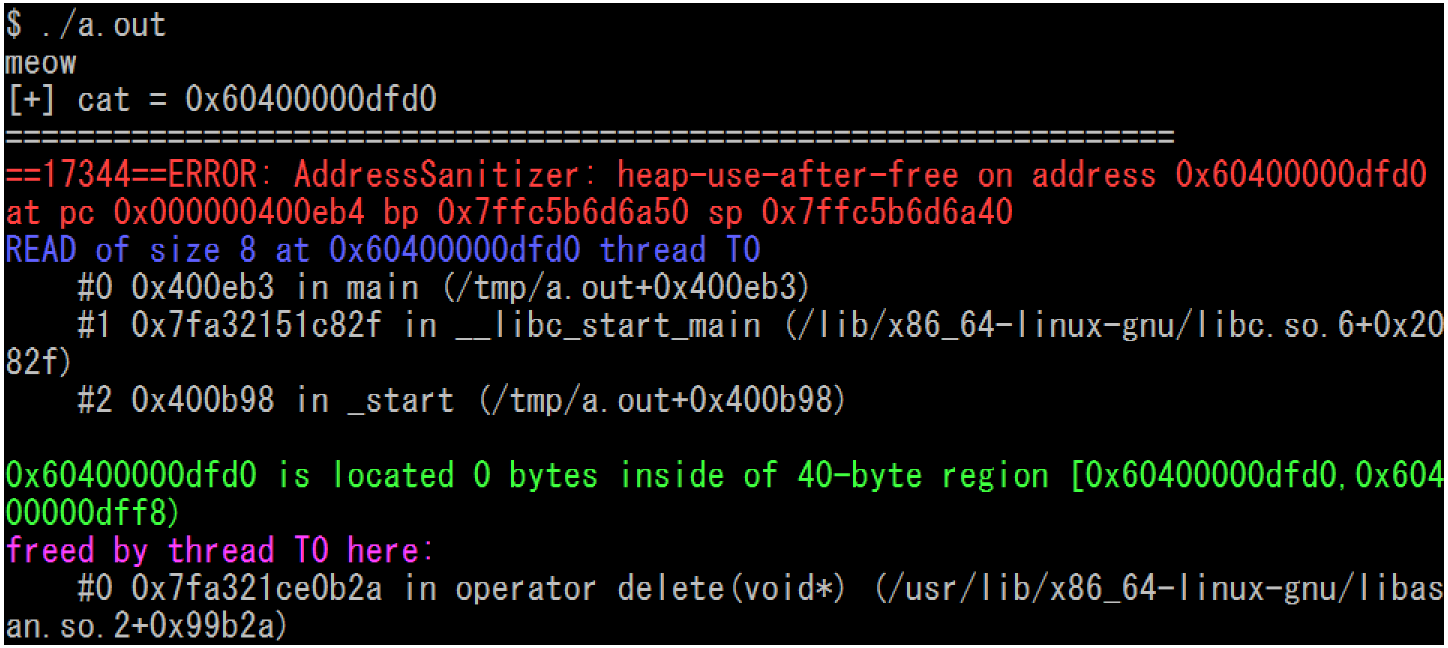
For this reason we have chosen 0x8123 as a sanitization value – from an operating system perspective this is in the same memory page as the zero address (NULL), but an access violation at 0x8123 will better stand out to the developer as needing more detailed attention.

But copied pointers remain dangling

- If more than one pointer pointing to the same address exist, others are not nullified
  - Need tracking copies to prevent completely
- FreeSentry: Protecting Against Use-After-Free Vulnerabilities Due to Dangling Pointers [NDSS 2015]
  - https://www.internetsociety.org/doc/freesentry-protecting-against-use-after-free-vulnerabilities-due-dangling-pointers
- Preventing Use-after-free with Dangling Pointers Nullification (DangNull) [NDSS 2015]
  - https://sslab.gtisc.gatech.edu/2015/dang-null.html
- DangSan: Scalable Use-after-free Detection [EUROSYS 2017]
  - http://www.cs.vu.nl/~giuffrida/papers/dangsan_eurosys17.pdf

GCC / Clang: AddressSanitizer

- -fsanitize=address
  - https://github.com/google/sanitizers/wiki/AddressSanitizer
  - Runtime memory error detection

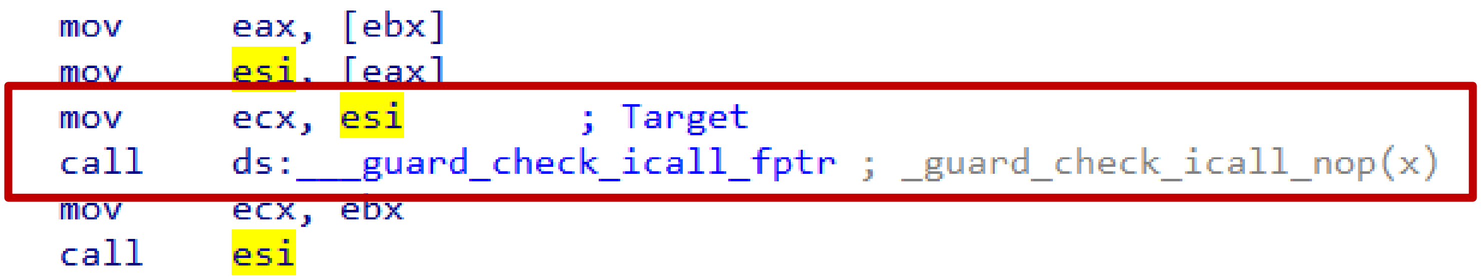


```
$ ./a.out
meow
[+] cat = 0x60400000dfd0
=================================================================
==17344==ERROR: AddressSanitizer: heap-use-after-free on address 0x60400000dfd0
at pc 0x000000400eb4 bp 0x7ffc5b6d6a50 sp 0x7ffc5b6d6a40
READ of size 8 at 0x60400000dfd0 thread T0
    #0 0x400eb3 in main (/tmp/a.out+0x400eb3)
    #1 0x7fa32151c82f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20
82f)
    #2 0x400b98 in _start (/tmp/a.out+0x400b98)

0x60400000dfd0 is located 0 bytes inside of 40-byte region [0x60400000dfd0,0x604
00000dff8)
freed by thread T0 here:
    #0 0x7fa321ce0b2a in operator delete(void*) (/usr/lib/x86_64-linux-gnu/libas
an.so.2+0x99b2a)
```

Eliminating all dangling pointers is hard …

Windows 8.1+: Control Flow Guard

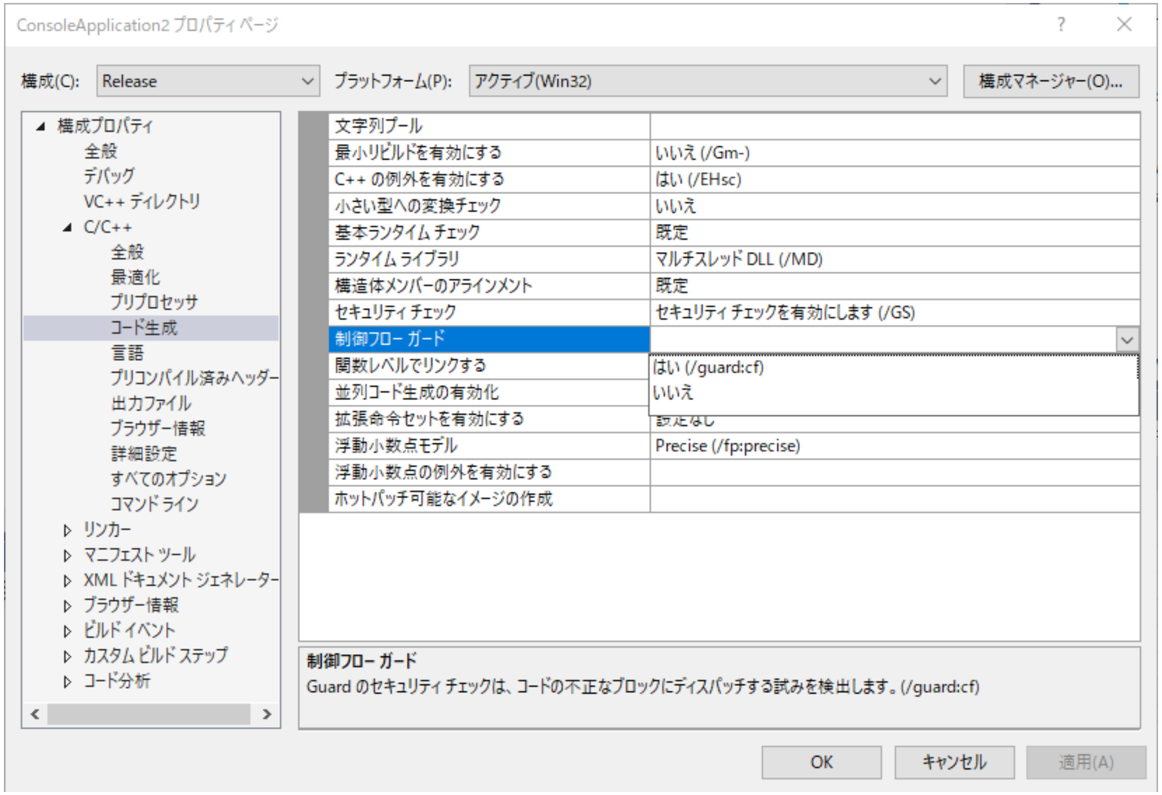
- /guard:cf (Enable Control Flow Guard)
  - https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx
  - Add check if the target of indirect call is the beginning of function



```
mov     eax, [ebx]
mov     esi, [eax]
mov     ecx, esi            ; Target
call    ds:___guard_check_icall_fptr ; _guard_check_icall_nop(x)
mov     ecx, ebx
call    esi
```

Windows 8.1+: Control Flow Guard

- Not enabled by default in Visual Studio 2015 “New Win32 Project”
- Incompatible with /ZI (Edit and Continue debug information)



構成(C): Release

プラットフォーム(P): アクティブ(Win32)

文字列プール	
最小リビルドを有効にする	いいえ (/Gm-)
C++ の例外を有効にする	はい (/EHsc)
小さい型への変換チェック	いいえ
基本ランタイム チェック	既定
ランタイム ライブラリ	マルチスレッド DLL (/MD)
構造体メンバーのアラインメント	既定
セキュリティ チェック	セキュリティ チェックを有効にします (/GS)
制御フロー ガード	はい (/guard:cf) いいえ
関数レベルでリンクする	
並列コード生成の有効化	
拡張命令セットを有効にする	設定なし
浮動小数点モデル	Precise (/fp:precise)
浮動小数点の例外を有効にする	
ホットパッチ可能なイメージの作成	

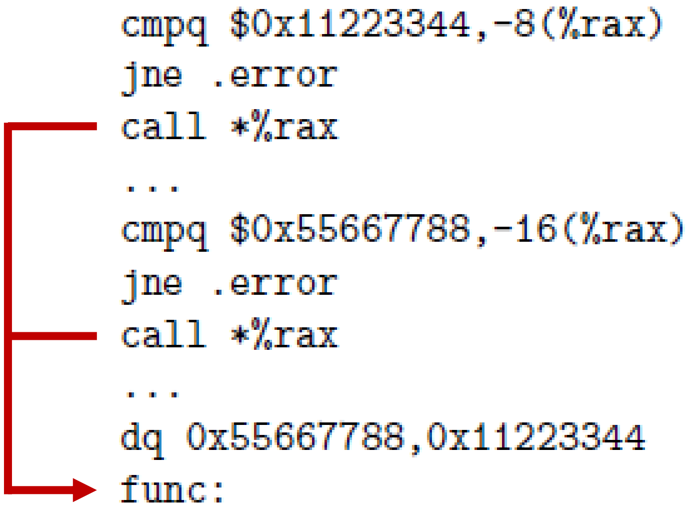
制御フロー ガード

Guard のセキュリティ チェックは、コードの不正なブロックにディスパッチする試みを検出します。(/guard:cf)

Linux grsecurity: RAP

- https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf

- Indirect Control Transfer Protection
  - Prepend type signature before each function and verify it before indirect call

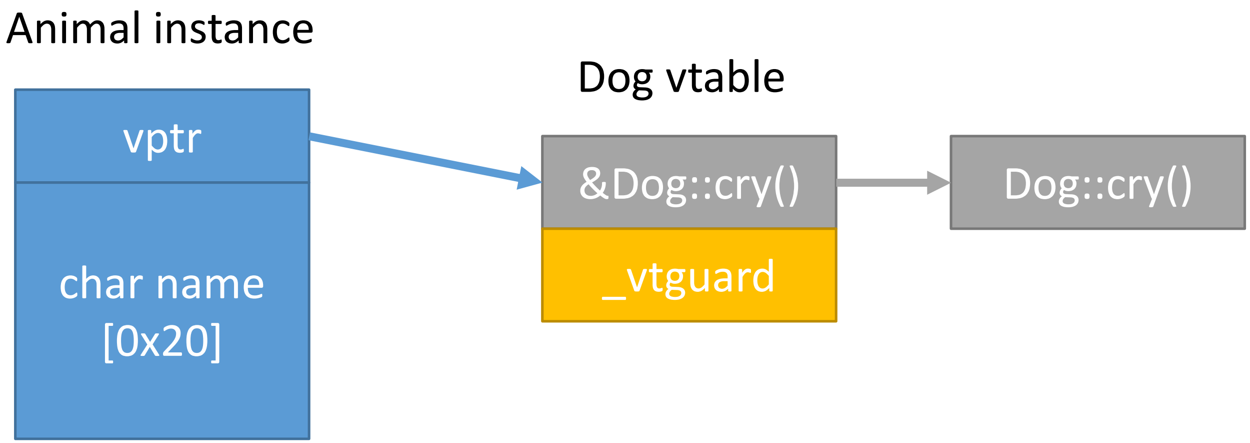


```
cmpq $0x11223344,-8(%rax)
jne .error
call *%rax
...
cmpq $0x55667788,-16(%rax)
jne .error
call *%rax
...
dq 0x55667788,0x11223344
func:
```

Windows 10: VTGuard (IE & Edge)

- Append canary to vtable and verify it before call

- Understanding the Attack Surface and Attack Resilience of Project Spartans New EdgeHTML Rendering Engine [Black Hat USA 2015]
  - https://www.blackhat.com/docs/us-15/materials/us-15-Yason-Understanding-The-Attack-Surface-And-Attack-Resilience-Of-Project-Spartans-New-EdgeHTML-Rendering-Engine.pdf



Windows 10: MemGC (IE & Edge)

- Managed heap and garbage collector which don’t free if there are dangling pointers
  - Enabled by default
- MemGC: Use-After-Free Exploit Mitigation in Edge and IE on Windows 10
  - https://securityintelligence.com/memgc-use-after-free-exploit-mitigation-in-edge-and-ie-on-windows-10/
- You can read it on GitHub!!1
  - https://github.com/Microsoft/ChakraCore/blob/master/lib/Common/Memory/Recycler.cpp

Comprehensive mitigations

- Mandatory access control
  - Mandatory Integrity Control (Windows Vista+) / SELinux (Linux)
  - Enforce the access policy between processes and resources (files etc.)
- Isolation
  - AppContainer (Windows 8+) / Namespaces (Linux)
  - Execute in separated context
- Endpoint security solution
  - Provided by many security venders including Microsoft
  - Monitoring host activities and detect critical accesses
- Patch update

Recap

- Use-after-free attacks are caused by dangling pointers to freed memory area
    - Recent vulnerabilities are not only buffer overflow
- There are ways to mitigate it
    - Safer language features, carefully programming, compiler's security option, smart GC, comprehensive mitigations
    - Enable /guard:cf in Release build of your new MSVC projects
- Think about what our problem is

References

- use-after-freeによるC++ vtable overwriteをやってみる - ももいろテクノロジー
  - http://inaz2.hatenablog.com/entry/2014/06/18/220735
- ROP検知手法RAPについてまとめてみる - ももいろテクノロジー
  - http://inaz2.hatenablog.com/entry/2015/10/30/024234
- Beyond Zero-day Attacks（4）: Use After Freeとヒープスプレー - ＠IT
  - http://www.atmarkit.co.jp/ait/articles/1409/22/news010.html
- Use After Free 脆弱性攻撃を試す
  - https://www.slideshare.net/monochrojazz/use-after-free
- Understanding the Low Fragmentation Heap (Black Hat USA 2010)
  - http://illmatics.com/Understanding_the_LFH_Slides.pdf
- Getting back determinism in the Low Fragmentation Heap - LSE Blog
  - https://blog.lse.epita.fr/articles/74-getting-back-determinism-in-the-lfh.html
- saaramar/Deterministic_LFH: Have fun with the LowFragmentationHeap
  - https://github.com/saaramar/Deterministic_LFH
- katagaitai CTF勉強会 #8 pwnables編
  - https://speakerdeck.com/bata_24/katagaitai-ctf-number-8

@inaz2

Thank You!