Monthly Research

Understanding bypassing ASLR by a pointer at a fixed address

FFRI,Inc.

http://www.ffri.jp

Ver 2.00.01

MS13-063

- Security patch published by Microsoft in Aug 2013
- Includes a fix for ALSR bypassing vulnerability(CVE-2013-2556)
- This slide is about the detail of the vulnerability and the fix

A summary of the ASLR bypassing vulnerability(CVE-2013-2556)

- Published in CansecWest2013
  - This vulnerability alone does not allow an attacker to exploit an application (need another vulnerability for successful exploit)
  - This vulnerability allows an attacker to bypass ASLR if another specific kind of vulnerability can be found.

Details of the vulnerability

- This vulnerability was published with a title “DEP/ASLR bypass without ROP/JIT” in CanSecWest2013 by Yang Yu.
- Mainly 2 problems
  - In 32bit Windows, a pointer to KiFastSystemCall is at a fixed address
  - In a 32bit process on 64bit Windows, a pointer to LdrHotPatchRoutine is at a fixed address

Why these pointers at fixed addresses are problem?

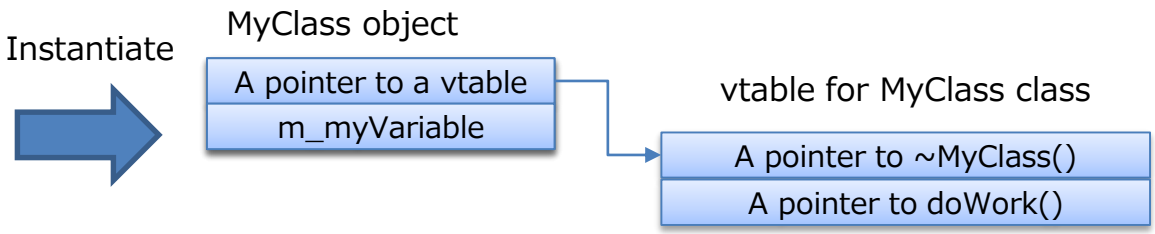
Can be used to bypass ASLR if there is use-after-free/heap overflow vulnerability which leads overwriting a pointer to a vtable of C++ objects.

What is “overwriting a pointer to a vtable”

Preliminary knowledge : C++ object layout

- C++ Object layout in general C++ implementation.

```
// Note that member functions are  "virtual"
class MyClass {
public:
    MyClass();
    virtual ~MyClass();
    virtual void doWork();
private:
    int m_myVariable;
};
```



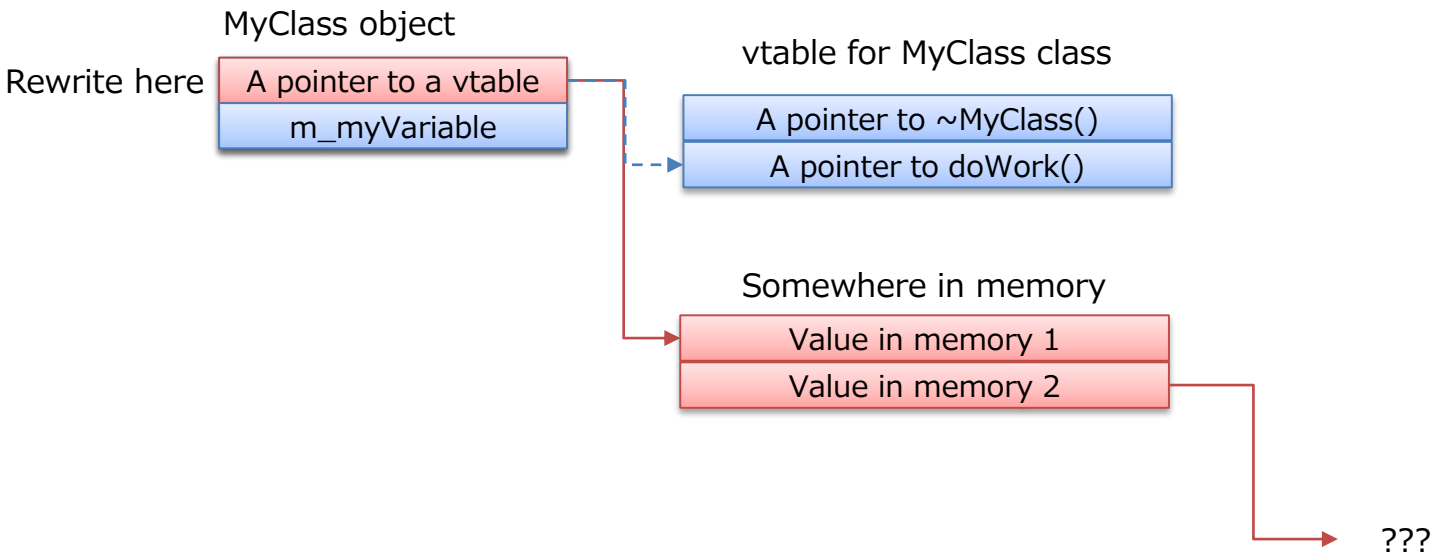
- How C++ calls member functions

```
// code to call doWork()
// ecx has the address of a MyWorker object
mov  eax,dword ptr [ecx]  // eax is the address of the vtable
push ecx                  // argument for the function call(*)
call dword ptr [eax+4]    // call doWork() (the address is obtained from the vtable)

* The first argument for cdecl calling convention is  "this"  pointer
```

A problem of rewriting a pointer to a vtable

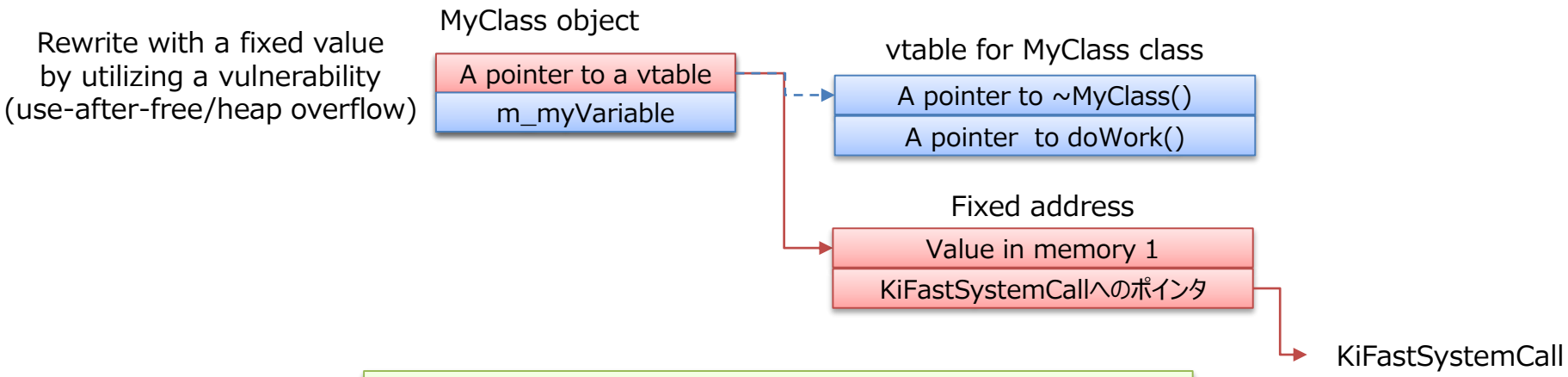
- What happens if a pointer to a vtable can be rewritten?




```
//The exactly same code from the previous slide.
//Rewriting a pointer to a vtable results in executing another function
mov  eax,dword ptr [ecx]  // eax is the address of vtable
push ecx                  // argument for the function call
call dword ptr [eax+4]    // Execute the memory "Value in memory 2" points to
```

In case of a pointer to KiFastSystemCall is at a fixed address

- Rewrite a pointer to a vtable in such a way that KiFastSystemCall is called
- KiFastSystemCall is a shared code to call system call in Windows
- ASLR is irrelevant in this scenario




```
mov  eax,dword ptr [ecx]  // eax is the address of vtable
push ecx                  // argument for the function call
call dword ptr [eax+4]    // call KiFastSystemCall
```

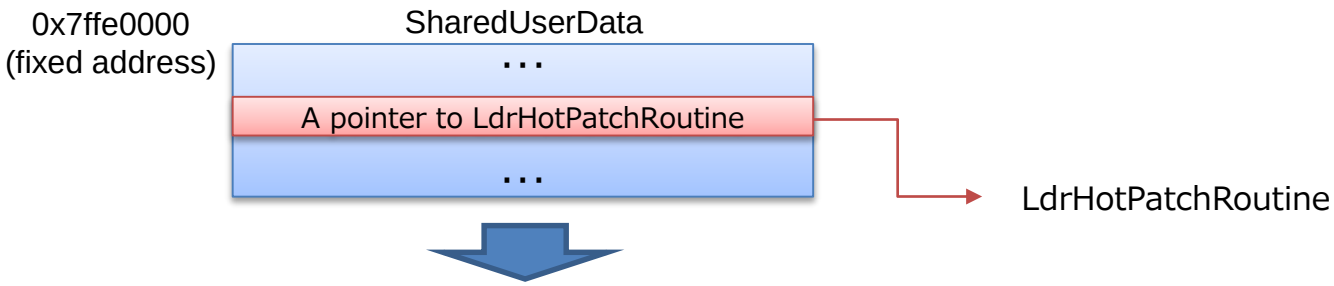
KiFastSystemCall is called.
However, calling system call with some arguments as an attacker intends to is difficult.

Using LdrHotPatchRoutine

- In 64bit Windows, a pointer KiFastSystemCall **does not exist** at a fixed address.
- But 32bit processes on 64bit Windows have a pointer to LdrHotPatchRoutine at a fixed address.
- LdrHotPatchRoutine internally loads a DLL which is specified via its argument.

```
struct HotPatchBuffer{
        ...
        USHORT PatcherNameOffset;  // An offset to a DLL name to load
        USHORT PatcherNameLen;     // The length of the DLL
        ...
};
void LdrHotPatchRoutine( struct *HotPatchBuffer);
```

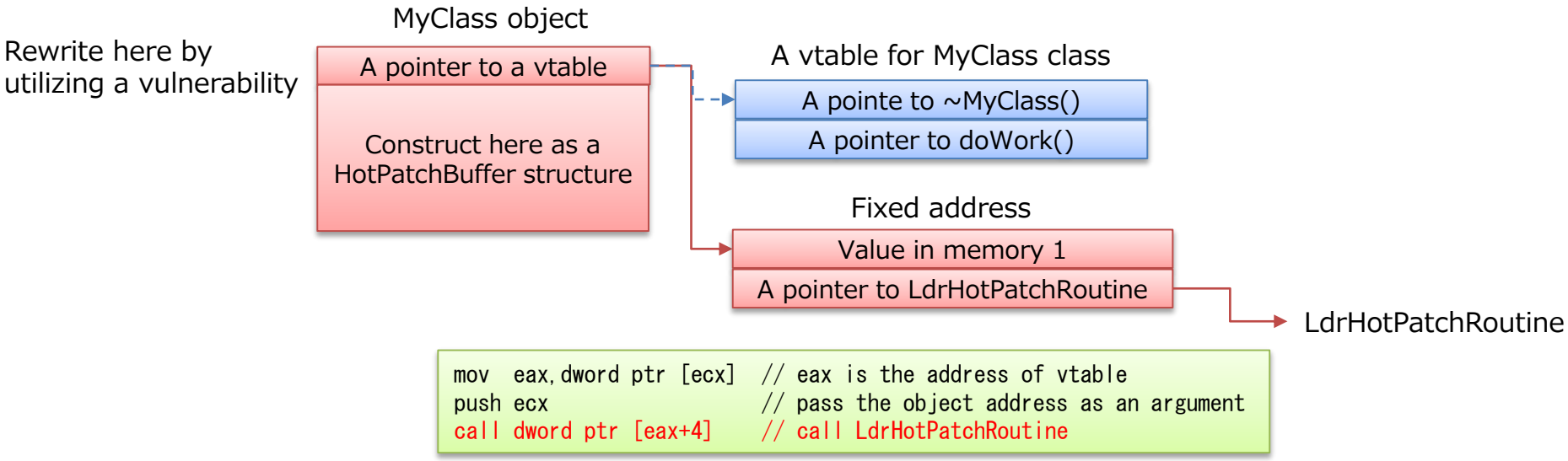
- A pointer to LdrHotPatchRoutine resides in SharedUserData in 32bit processes on 64bit Windows
- SharedUserData is at a fixed address(0x7ffe0000)



Overwrite C++ object in such a way that the vtable includes a pointer to LdrHotPatchRoutine.
DLL can be loaded.

In case of a pointer to LdrHotPatchRoutine is at a fixed address

- Rewrite a pointer to a vtable in such a way as calling LdrHotPatchRoutine

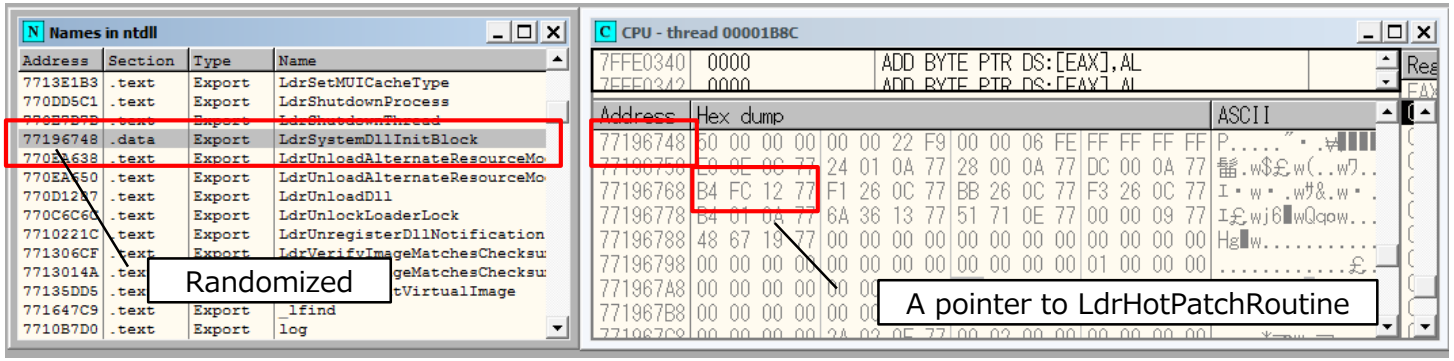



```
mov  eax,dword ptr [ecx]  // eax is the address of vtable
push ecx                  // pass the object address as an argument
call dword ptr [eax+4]    // call LdrHotPatchRoutine
```

- LdrHotPatchRoutine is called.
- Constructing an argument to LdrHotPatchRoutine (with a DLL name such as ¥¥192.168.1.100¥foo.dll) will results in loading a DLL on a server.
- Note that the object can be overwritten with arbitrary data when an attacker overwrites a pointer to a vtable

The fix in MS13-063

- MS13-063 fixes the vulnerability in such a way that a pointer to LdrHotPatchRoutine is not at a fixed address.
  - Eliminate a function table in SharedUserData
  - Move the function table to a data section in ntdll.dll and export it as LdrSystemDllInitBlock



- ASLR is enabled on ntdll.dll and it makes the address of the function table not fixed.

<u>Can not bypass ASLR to load a DLL by utilizing LdrHotPatchRoutine</u>

References

- http://technet.microsoft.com/ja-jp/security/bulletin/ms13-063
- http://cansecwest.com/slides/2013/DEP-ASLR%20bypass%20without%20ROP-JIT.pdf
- http://blogs.technet.com/b/srd/archive/2013/08/12/mitigating-the-ldrhotpatchroutine-dep-aslr-bypass-with-ms13-063.aspx
- http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2556

Contact Information

E-Mail : research-feedback@ffri.jp
Twitter : @FFRI_Research