Monthly Research

Windows New Security Features - Control Flow Guard

FFRI, Inc
http://www.ffri.jp

Ver 1.00.02

About Control flow guard(Guard CF)

- **Control flow guard** made its debut at Windows 8.1 Preview release
  - It disabled on Windows 8.1 RTM (Release To Manufacturing) and Windows 8.1 releases
  - Available on Windows 10 Technical Preview and Windows 8.1 Update Pack

- We call control flow guard “**Guard CF**” in this document
  - Because acronym of control flow guard(CFG) means control flow graph generally

Notes

- Guard CF is work-in-progress feature
- We tested Windows 10 Technical Preview and Visual Studio 2015 Preview

Threat Model

- Arbitrary code execution
  - Manipulating indirect call operand

- Typical example
  - vtable overwrite

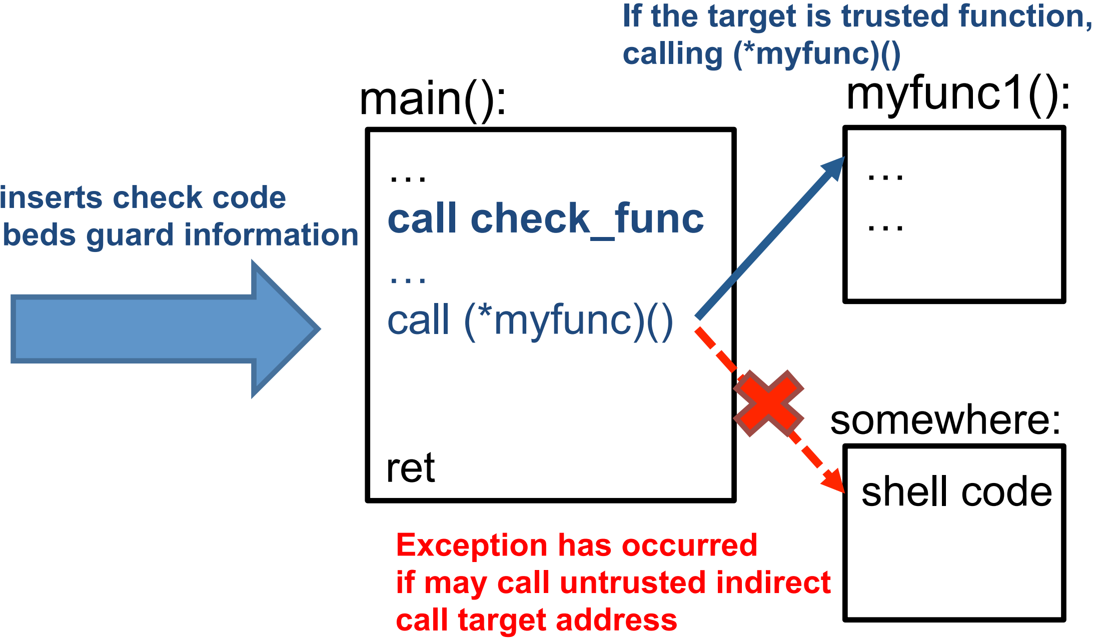
Protecting with Guard CF

- Insert check function called before indirect calls at compile time
- The check function validates indirect call target address
  - Raises violation if untrusted address’s called

```
void myfunc1() {
    printf("myfunc1\n");
}

int main(int argc, char* argv[])
{
    void(*myfunc)();
    myfunc = myfunc1;
    (*myfunc)();
    return 0;
}
```

Compiler inserts check code
Linker embeds guard information

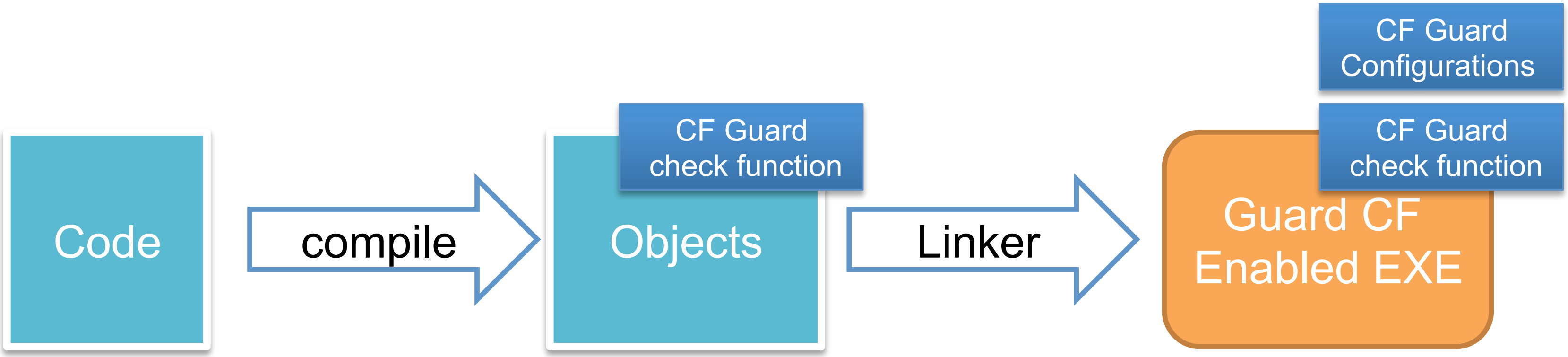


If the target is trusted function,
calling (*myfunc)()

Exception has occurred
if may call untrusted indirect
call target address

Protecting with Guard CF (cont.)

- Guard CF trusts registered address of guard CF function table
- Guard CF function table exists PE/COFF headers which made by linker
- Windows runtime (ntdll.dll) builds trusted function bitmap from Guard CF function table at loading time



Compiler Inserts check function before indirect calls

Linker makes guard CF configuration and appends executable file headers

Guard CF in Visual Studio 2015 Preview

- Using hidden option

```
cl /d2guard4  test.cpp  /link /guard:cf
```

See also：
http://blogs.msdn.com/b/vcblog/archive/2014/12/08/visual-studio-2015-preview-work-in-progress-security-feature.aspx

PE/COFF headers

- DLL Characteristics

```
OPTIONAL HEADER VALUES
             10B magic # (PE32)
                ...
         C140 DLL characteristics
                   Dynamic base
                   NX compatible
                   Guard
                   Terminal Server Aware
```

build with guard CF option

```
OPTIONAL HEADER VALUES
             10B magic # (PE32)
                ...
         8140 DLL characteristics
                   Dynamic base
                   NX compatible
                   Terminal Server Aware
```

build without guard CF option

PE/COFF headers (cont.d)

- Load config structure in PE/COFF headers

```
Section contains the following load config:

           0000005C size
           ...
           0041D108 Guard CF address of check-function pointer
           00000000 Reserved
           0041D150 Guard CF function table
                 2A Guard CF function count
           00003500 Guard Flags
                      CF Instrumented
                      FID table present
                      Protect delayload IAT
                      Delayload IAT in its own section
```

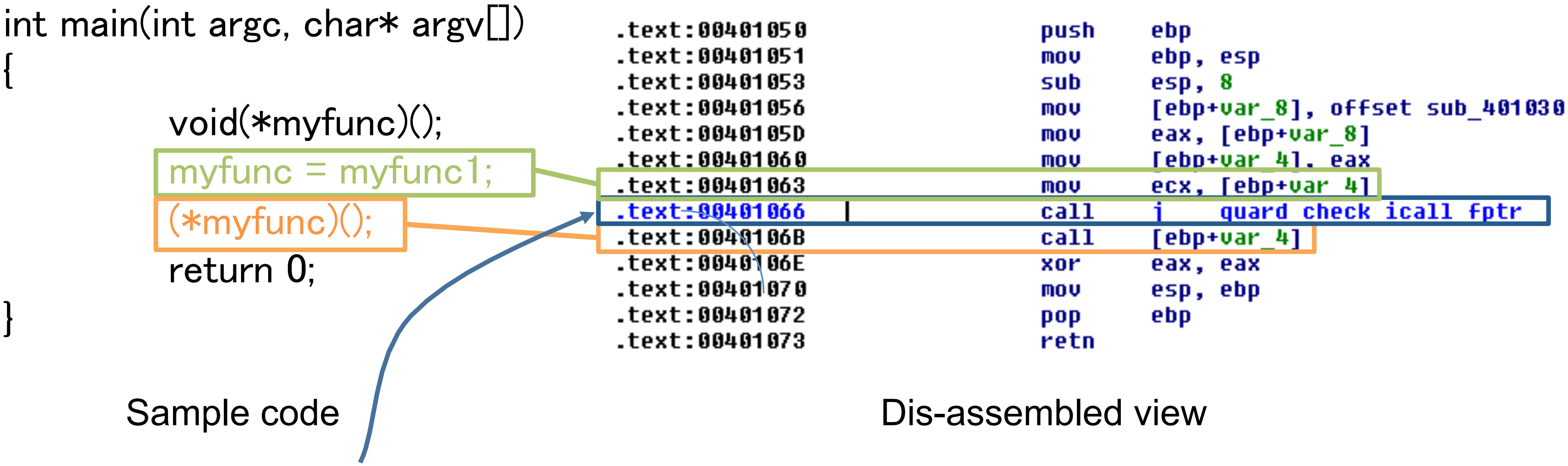
PE/COFF headers (cont.d)

- Guard CF function table in PE/COFF headers

```
…
Guard CF Function Table

          Address
          ---------
          00401000
          00401030
          004011E0
          00401270
          004013F0
         …
```

Guard CF Tutorial



```
int main(int argc, char* argv[])
{
        void(*myfunc)();
        myfunc = myfunc1;
        (*myfunc)();
        return 0;
}
```

Sample code

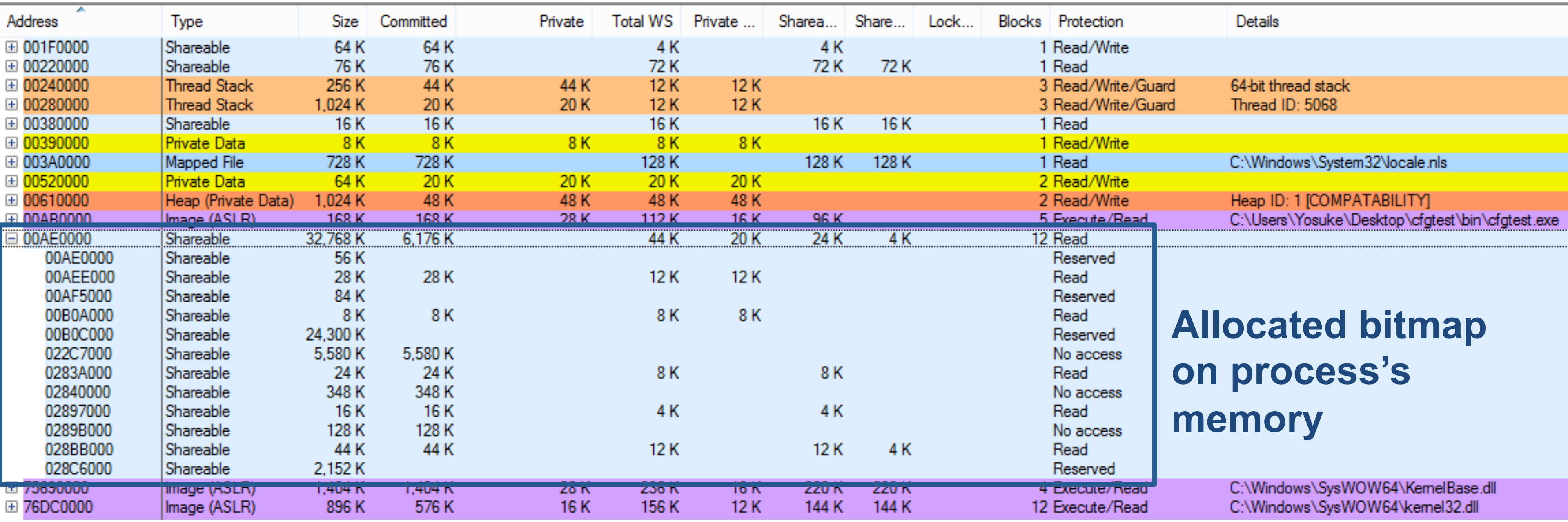
```
.text:00401050                 push    ebp
.text:00401051                 mov     ebp, esp
.text:00401053                 sub     esp, 8
.text:00401056                 mov     [ebp+var_8], offset sub_401030
.text:0040105D                 mov     eax, [ebp+var_8]
.text:00401060                 mov     [ebp+var_4], eax
.text:00401063                 mov     ecx, [ebp+var_4]
.text:00401066                 call    j___guard_check_icall_fptr
.text:0040106B                 call    [ebp+var_4]
.text:0040106E                 xor     eax, eax
.text:00401070                 mov     esp, ebp
.text:00401072                 pop     ebp
.text:00401073                 retn
```

Dis-assembled view

Inserted Guard CF check function

Guard CF Function Bitmap

- Guard CF check function validates target address using bitmap
  - Bitmap is created by loader
  - Raising security assertion exceptions(int 29h) if call target not exist in bitmap



Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
⊞ 001F0000	Shareable	64 K	64 K		4 K		4 K			1	Read/Write	
⊞ 00220000	Shareable	76 K	76 K		72 K		72 K	72 K		1	Read	
⊞ 00240000	Thread Stack	256 K	44 K	44 K	12 K	12 K				3	Read/Write/Guard	64-bit thread stack
⊞ 00280000	Thread Stack	1,024 K	20 K	20 K	12 K	12 K				3	Read/Write/Guard	Thread ID: 5068
⊞ 00380000	Shareable	16 K	16 K		16 K		16 K	16 K		1	Read	
⊞ 00390000	Private Data	8 K	8 K	8 K	8 K	8 K				1	Read/Write	
⊞ 003A0000	Mapped File	728 K	728 K		128 K		128 K	128 K		1	Read	C:\Windows\System32\locale.nls
⊞ 00520000	Private Data	64 K	20 K	20 K	20 K	20 K				2	Read/Write	
⊞ 00610000	Heap (Private Data)	1,024 K	48 K	48 K	48 K	48 K				2	Read/Write	Heap ID: 1 [COMPATABILITY]
⊞ 00AB0000	Image (ASLR)	168 K	168 K	28 K	112 K	16 K	96 K			5	Execute/Read	C:\Users\Yosuke\Desktop\cfgtest\bin\cfgtest.exe
⊟ 00AE0000	Shareable	32,768 K	6,176 K		44 K	20 K	24 K	4 K		12	Read	
00AE0000	Shareable	56 K									Reserved	
00AEE000	Shareable	28 K	28 K		12 K	12 K					Read	
00AF5000	Shareable	84 K									Reserved	
00B0A000	Shareable	8 K	8 K		8 K	8 K					Read	
00B0C000	Shareable	24,300 K									Reserved	
022C7000	Shareable	5,580 K	5,580 K								No access	
0283A000	Shareable	24 K	24 K		8 K		8 K				Read	
02840000	Shareable	348 K	348 K								No access	
02897000	Shareable	16 K	16 K		4 K		4 K				Read	
0289B000	Shareable	128 K	128 K								No access	
028BB000	Shareable	44 K	44 K		12 K		12 K	4 K			Read	
028C6000	Shareable	2,152 K									Reserved	
⊞ 75690000	Image (ASLR)	1,404 K	1,404 K	28 K	236 K	16 K	220 K	220 K		4	Execute/Read	C:\Windows\SysWOW64\KernelBase.dll
⊞ 76DC0000	Image (ASLR)	896 K	576 K	16 K	156 K	12 K	144 K	144 K		12	Execute/Read	C:\Windows\SysWOW64\kernel32.dll

Allocated bitmap on process’s memory

Memory usage (using vmmap)

Limitation

- CF Guard protects indirect call only
  - Indirect jump and return is not protected

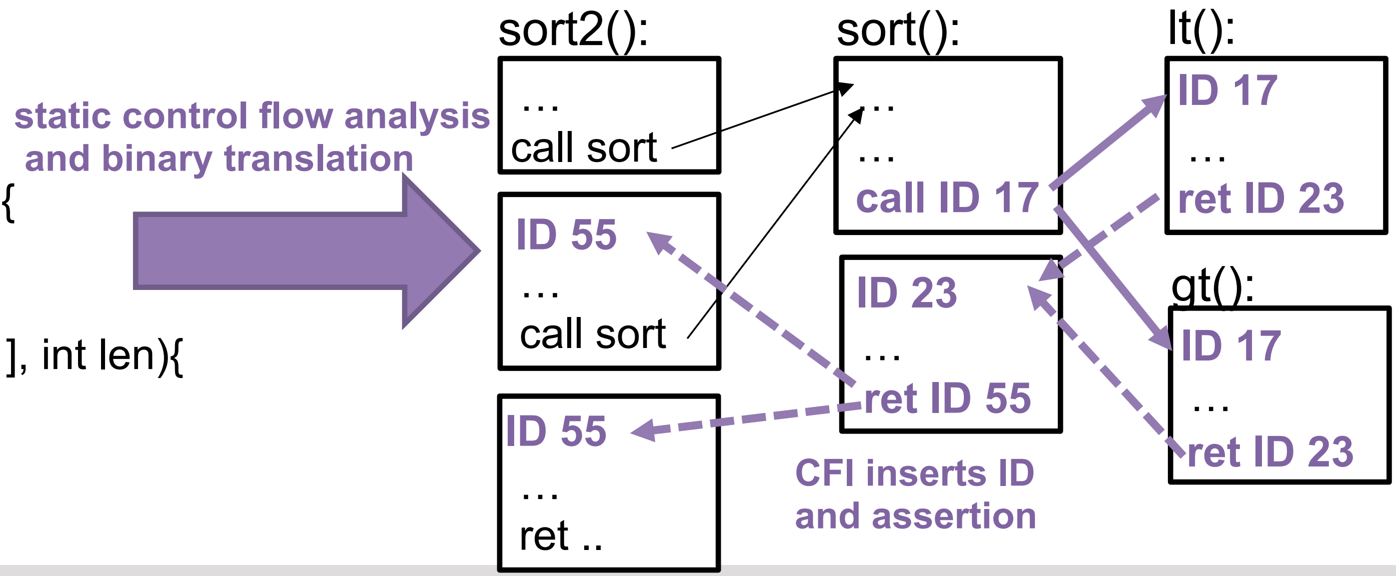
- Code reuse attack mitigation is limitedly
  - Guarded functions could be called by any indirect caller

Ref: Control flow integrity(CFI)

- Control flow integrity(CFI) restricts indirect branch(jmp, call, ret) source and destination
  - Microsoft researcher published this research in 2005

- CFI implementation uses binary translation and static control flow analysis

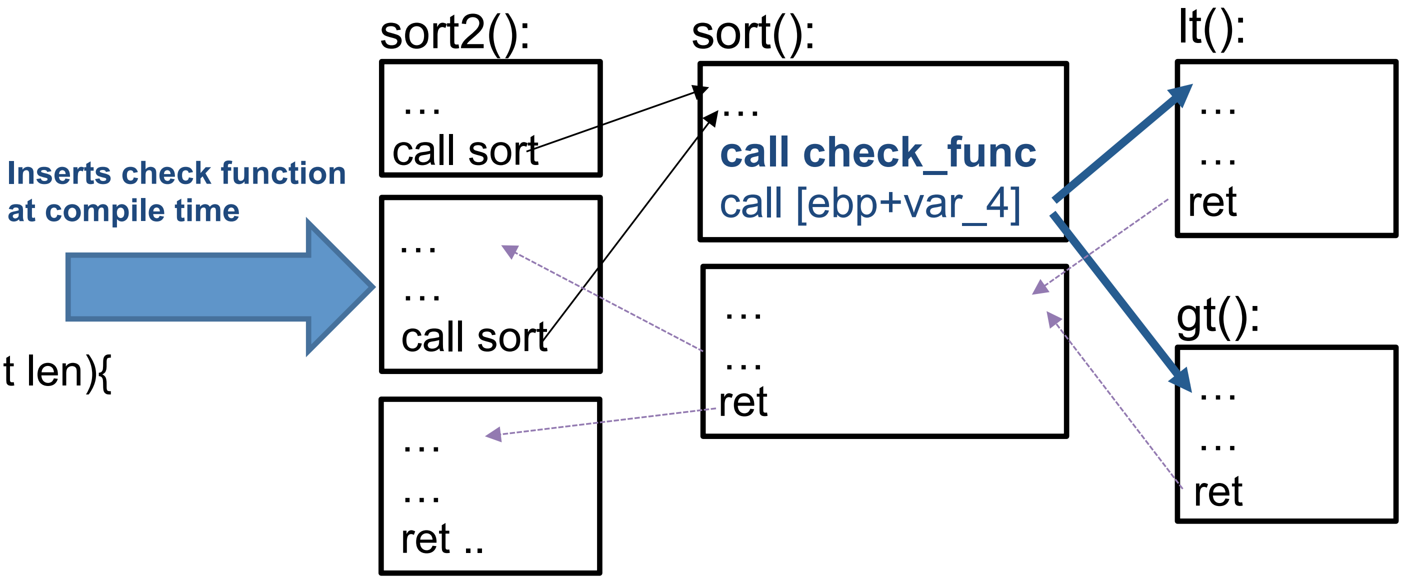
```
bool lt(int x, int y) {
  return x < y;
}
bool gt(int x, int y) {
  return x > y;
}
sort2(int a[ ], int b[ ], int len){
  sort( a, len, lt );
  sort( b, len, gt );
}
```



Relation between Guard CF and CFI

- CFI guaranteed stronger control flow integrity than Guard CF
- But, CFI needs binary translation and many function insertions
  - It has an impact on performance and binary compatibility
- Guard CF simplified CFI that checks trustworthiness of call target

```
bool lt(int x, int y) {
  return x < y;
}
bool gt(int x, int y) {
  return x > y;
}
sort2(int a[ ], int b[ ], int len){
  sort( a, len, lt );
  sort( b, len, gt );
}
```



Conclusion

- Introducing Control flow guard(Guard CF) design and implementation
  - To enable Guard CF for existing source code, application developers re-compile program using compiler option and linker option with Guard CF aware compiler

- Microsoft attempting to put Guard CF into practical use
  - It based on control flow integrity research over a decade

References

- “Visual Studio 2015 Preview: Work-in-Progress Security Feature”
  http://blogs.msdn.com/b/vcblog/archive/2014/12/08/visual-studio-2015-preview-work-in-progress-security-feature.aspx
  (2014/12/19 viewed)

- MJ0011, "Windows 10 Control Flow Guard Internals“, Power of Community 2014.

- Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti, “Control-Flow Integrity”, ACM CCS’05, November 2005
  http://research.microsoft.com/apps/pubs/default.aspx?id=64250

Contact Information

E-Mail : research—feedback@ffri.jp

Twitter : @FFRI_Research