Ice Age melting down!

Intel features considered usefull!

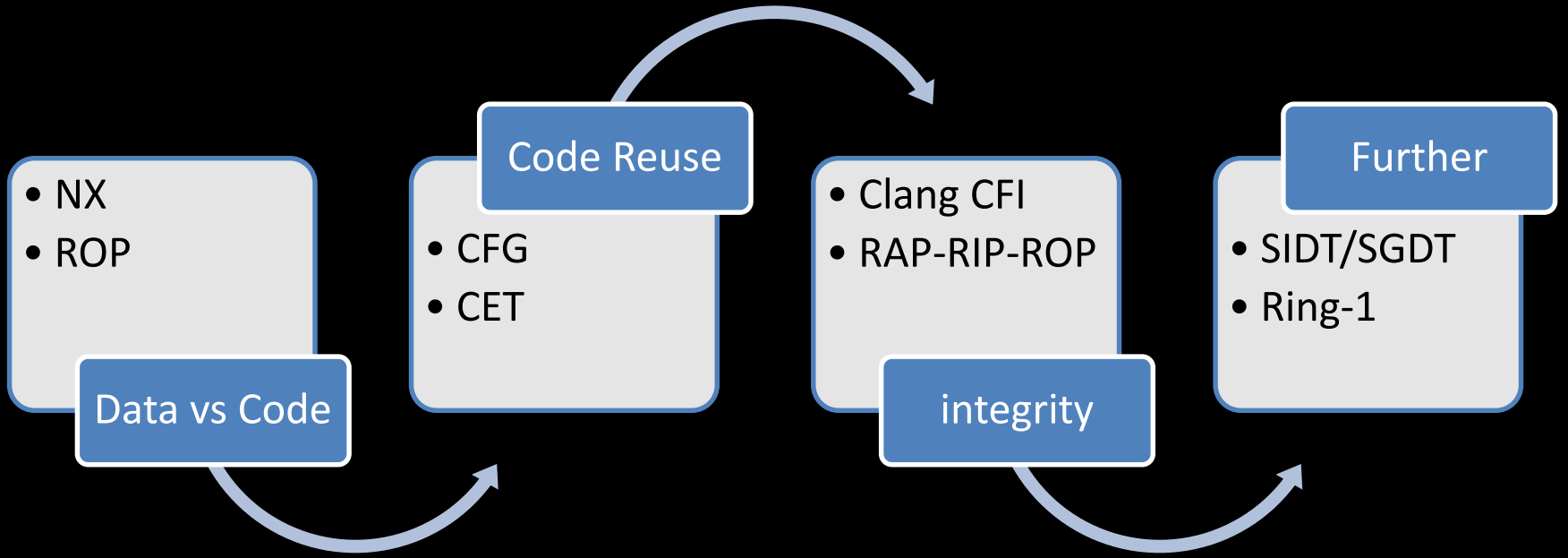
whoami

- @zer0mem, Peter Hlavaty
- Lead of Windows kernel security at KeenLab, Tencent

- 2015, 2016 pwn2own winner { team work! }
  - 2015 2x TTF
  - 2016 Master Of Pwn (Edge -> SYSTEM)
- Top 100 MSRC (Ranked 36 at 2016)
- Kernel CVEs and advanced exploitation techniques

- Delivered talks at Recon, SyScan, ZeroNights and others
- Wushu player

agenda


• NX
• ROP
Data vs Code
Code Reuse
• CFG
• CET
• Clang CFI
• RAP-RIP-ROP
integrity
Further
• SIDT/SGDT
• Ring-1

What is this talk about

- Elevation of Privilege
- Kernel code exec
- Software Mitigations
- New Bare metal principles
- Future of software security

At the very begining

```
48 83 EC 28 E8 7F BB 00  00 48 83 C4 28 E9 02 00  Hâ8(F.+..Hâ-(T..
00 00 CC CC 48 89 5C 24  10 48 89 74 24 18 57 48  ..¦¦Hë\$.Hët$.WH
```

:)

you got code exec

```
; Attributes: library function

public start
start proc near
sub     rsp, 28h
call    sub_14003BB0C
add     rsp, 28h
jmp     __tmainCRTStartup
start endp
```

Custom data shipped to domain in order to exec

#1 NonExec

- ExAllocatePool(...Nx)

ExAllocatePool allocates pool memory of the specified type and returns a pointer to the allocated block.

Syntax

C++

```
PVOID ExAllocatePool(
  _In_ POOL_TYPE PoolType,
  _In_ SIZE_T    NumberOfBytes
);
```

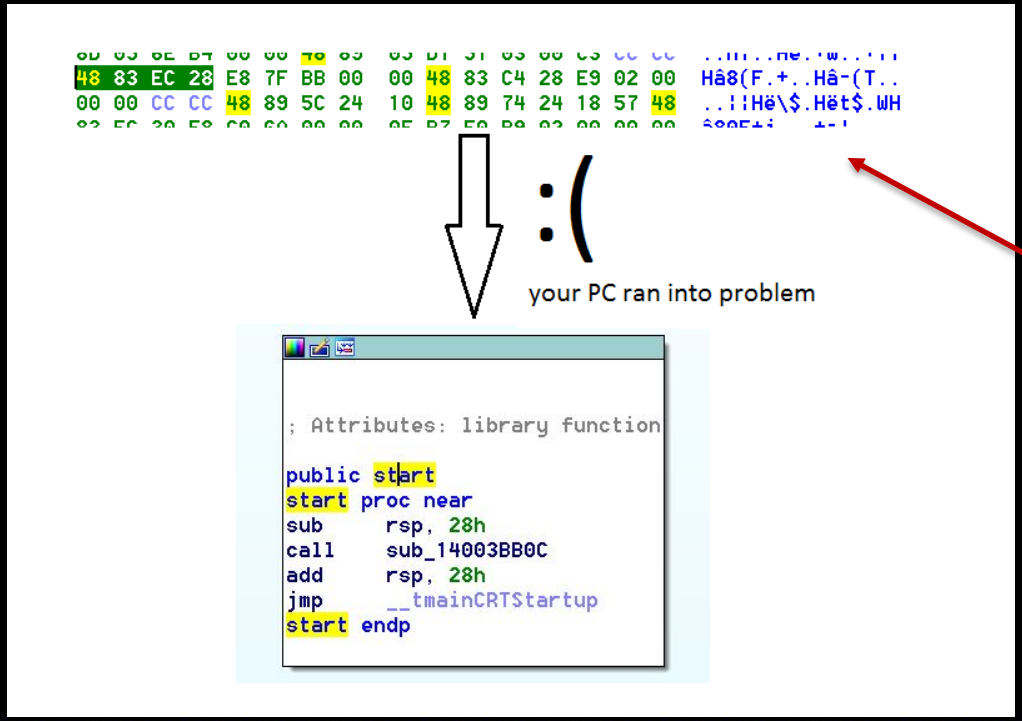
Parameters

PoolType [in]

Specifies the type of pool memory to allocate. For a description of the available pool memory types, see POOL_TYPE.

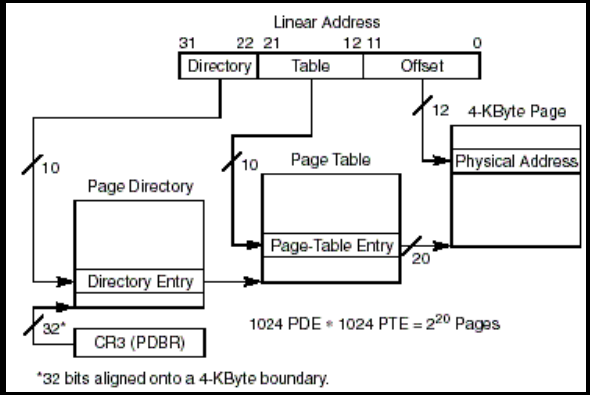
```
typedef enum _POOL_TYPE {
  NonPagedPool,
  NonPagedPoolExecute                 = NonPagedPool,
  PagedPool,
  NonPagedPoolMustSucceed             = NonPagedPool + 2,
  DontUseThisType,
  NonPagedPoolCacheAligned            = NonPagedPool + 4,
  PagedPoolCacheAligned,
  NonPagedPoolCacheAlignedMustS       = NonPagedPool + 6,
  MaxPoolType,
  NonPagedPoolBase                    = 0,
  NonPagedPoolBaseMustSucceed         = NonPagedPoolBase + 
  NonPagedPoolBaseCacheAligned        = NonPagedPoolBase + 
  NonPagedPoolBaseCacheAlignedMustS   = NonPagedPoolBase + 
  NonPagedPoolSession                 = 32,
  PagedPoolSession                    = NonPagedPoolSession
  NonPagedPoolMustSucceedSession      = PagedPoolSession + 
  DontUseThisTypeSession              = NonPagedPoolMustSuc
  NonPagedPoolCacheAlignedSession     = DontUseThisTypeSess
  PagedPoolCacheAlignedSession        = NonPagedPoolCacheAl
  NonPagedPoolCacheAlignedMustSSession = PagedPoolCacheAlign
  NonPagedPoolNx                      = 512,
  NonPagedPoolNxCacheAligned          = NonPagedPoolNx + 4,
  NonPagedPoolSessionNx               = NonPagedPoolNx + 32
} POOL_TYPE;
```

Security awerness++

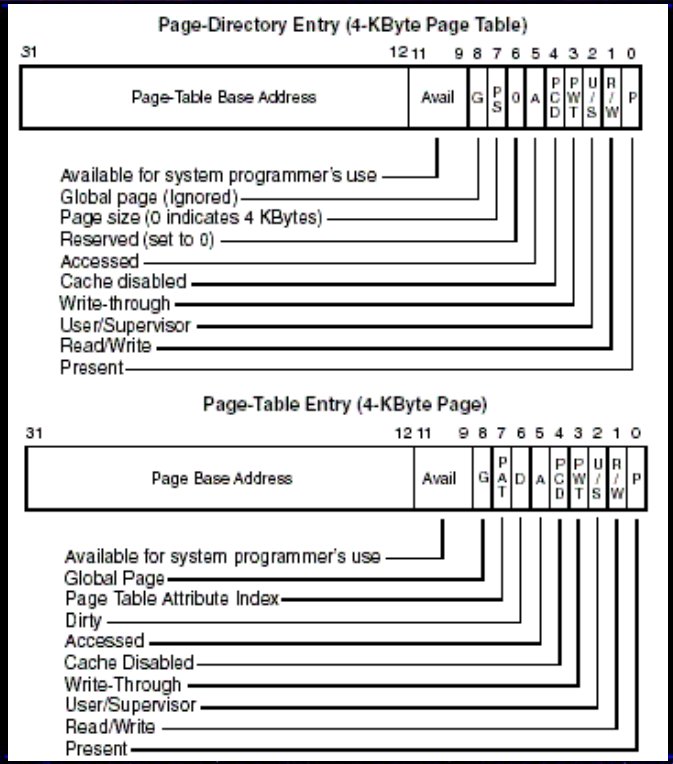


Page Tables

- Exec bit on / off
- Accessible from ring0*



- * if no ring-1 in place

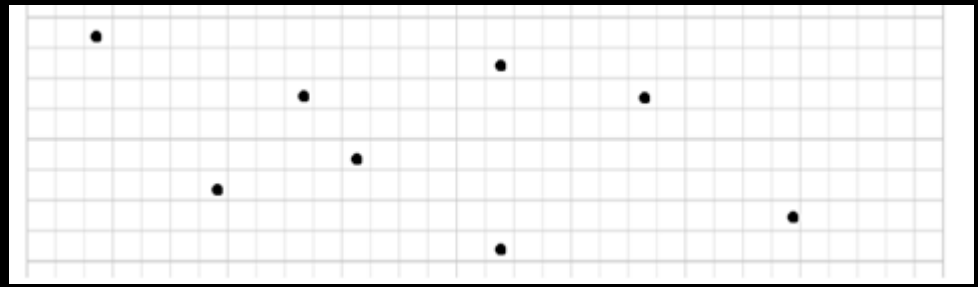


ROP

- Reusing of existing code
- Returning/Jumping into different (even misaligned) instructions
- Chaining those gadgets together
- Reliable control flow achieved

CFG

- Microsoft introduced mitigation
- Bitmap of valid indirect jump locations



- Simple
- Effective

http://blog.trendmicro.com/trendlabs-security-intelligence/exploring-control-flow-guard-in-windows-10/

CFG

- Check before indirect jump/call
  - LdrpValidateUserCallTarget

```
                mov     edx, dword ptr ds:GuardCFBitMapAddress
                mov     eax, ecx
                shr     eax, 8
                mov     edx, [edx+eax*4]
                mov     eax, ecx
                shr     eax, 3
                test    cl, 0Fh
                jnz     short not_aligned_adress
                bt      edx, eax
                jnb     short invalid_target
                retn
not_aligned_adress
                or      eax, 1
                bt      edx, eax
                jnb     short invalid_target
                retn
```

http://powerofcommunity.net/poc2014/mj0011.pdf

CFG

- Do not solve ROP itself, just focus on preventing kicking it off
  - fair enough approximation

Use function instead!

- pwn2own 2015
- Function driven attack
- Instead of gadget we use functions
  - Use it until we got custom kernel code exec

p2o 2015

- We got Read/Write primitive
  - ReCon , GdiBitmap technique
- Problems
  - #1 I dont like ROP and shellcode neither
  - #2 RWE page
  - #3 custom kernel code trigger

- NtUserMessageCall

```
extern "C"
void*
NtUserMessageCall(
    HWND hwnd,//window handle
    size_t fnSelector,//second arg - have to be bigger than 0x400
    size_t,
    size_t,
    size_t,
    size_t fnId//index-6 of function from MpFnidPfn table
    );
```

```
.data:FFFFF97FFF3C0DB0 mpFnidPfn       dq ?
.data:FFFFF97FFF3C0DB0
.data:FFFFF97FFF3C0DB8 qword_FFFFF97FFF3C0DB8 dq ?
.data:FFFFF97FFF3C0DC0 qword_FFFFF97FFF3C0DC0 dq ?
.data:FFFFF97FFF3C0DC8 qword_FFFFF97FFF3C0DC8 dq ?
.data:FFFFF97FFF3C0DD0 qword_FFFFF97FFF3C0DD0 dq ?
.data:FFFFF97FFF3C0DD8 qword_FFFFF97FFF3C0DD8 dq ?
.data:FFFFF97FFF3C0DE0 qword_FFFFF97FFF3C0DE0 dq ?
.data:FFFFF97FFF3C0DE8                 db    ? ;
.data:FFFFF97FFF3C0DE9                 db    ? ;
.data:FFFFF97FFF3C0DEA                 db    ? ;
.data:FFFFF97FFF3C0DEB                 db    ? ;
.data:FFFFF97FFF3C0DEC                 db    ? ;
.data:FFFFF97FFF3C0DED                 db    ? ;
.data:FFFFF97FFF3C0DEE                 db    ? ;
.data:FFFFF97FFF3C0DEF                 db    ? ;
.data:FFFFF97FFF3C0DF0                 db    ? ;
.data:FFFFF97FFF3C0DF1                 db    ? ;
.data:FFFFF97FFF3C0DF2                 db    ? ;
```

http://www.k33nteam.org/noks.html

p2o 2015

p2o 2015

- Use gdi.lo() to write our kernel mode driver in place
  - cc_shellcode framework with some kernel mode features
- Need to deal with relocations, and import resolving
  - cc_shellcode framework do it automatically

p2o 2015

- Kernel code exec switch
  - gdi.lo() _EPROCESS list walking
  - Find stack of our thread
  - With gdi.lo() rewrite own stack return

CET - IBT

- At compile time mark source and destination of call
  - Indirect call/jmp
  - ENDBRXX
- If you jump somewhere what is not marked as destination, boom
- Similar to CFG
  - roots better into the code
  - this time from bare-metal setting distinctions
- Direct and clean solving of ROP
  - Solve ROP comprehensively not just at very beginning!
  - Basically shaping architecture forward to secure oriented one
- Does *not* intend to solve integrity!

https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

Code Exec

- anti-ROP is no problem
- No direct page-table altering is no problem
- Type-hash integrity + stack, ouch…
- EPT ? …

EPT & Code Signing

- Running at ring-1
- No W+X pages at once, no ever
- EPT vs ring0 page tables, shadowing
- Signing check once it tries to exec
- Game over for custom code exec if no logical bug ?

Control Flow control

- CodeSigning or PageTable lockdown no problem
  - We dont intend to use custom code

- anti-ROP becomes a problem +-
- Type-hash integrity is essential problem

Control Flow Integrity

Another technologies
[not applied at windows kernel]

RAP-RIP-ROP

- Return address protection
- Reserved register for cookie
- Cookie per :
  - Task
  - Syscall
  - Special loops scenarios

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

RAP-RIP-ROP

```
push %rbx
mov 8(%rsp),%rbx
xor %r12,%rbx
...
xor %r12,%rbx
cmp %rbx,8(%rsp)
jnz .error
pop %rbx
retn
.error:
ud2
```

stack

- cookie ^ ret
- Kernel.Io() from another thread can be possibly used to leak & rewrite
  - However unread-able kernel stacks can harden approach

Self-modify

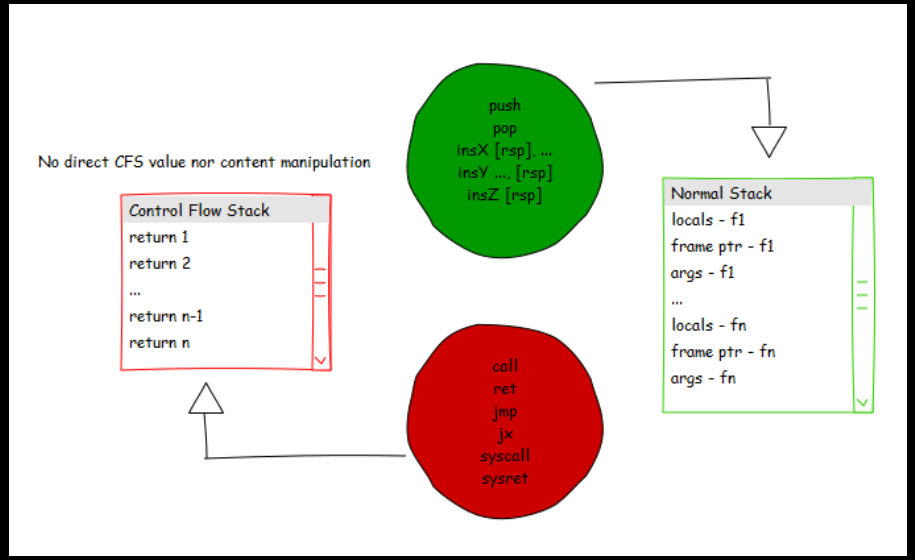
- Limited / jailed kernel-A.Io() to create better version of kernel-X.Io()
- kernel-X.Io() need :
  - Contains loop in which do read/write
  - Loop counter should depend on writable memory
- iovec alikes are good candidates (splice, ...)

Self-modify

- kernel-X.Io() modify itself
- Corrupt loop counter
- To create prolonged loop
- Read out own cookie^ret
- Rewrite own cookie^ret
  - In another thread resolve and modify
  - Readed value in previous step must be accessible somehow (user mode / kernel.Io()) to second thread
  - Patched value should be available in further read/write operations of self-looped kernel-X.Io()

Intel - Shadow stack

- Normal instruction can not touch it directly
- Kernel.Io() is out of game



Clang

- Forward-Edge CFI
- bit vector for static types
  - Read-only
  - Per static type
- Compiler + linker
  - Depends on LLVM's type metadata

http://www.pcc.me.uk/~peter/acad/usenix14.pdf

Clang

```
struct A {
  virtual void f1();
  virtual void f2();
  virtual void f3();
};

struct B : A {
  virtual void f1();
  virtual void f2();
  virtual void f3();
};

struct C : A {
  virtual void f1();
  virtual void f2();
  virtual void f3();
};
```

The scheme will cause the virtual tables for A, B and C to be laid out consecutively:

Virtual Table Layout for A, B, C

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A::offset-to-top	&A::rtti	&A::f1	&A::f2	&A::f3	B::offset-to-top	&B::rtti	&B::f1	&B::f2	&B::f3	C::offset-to-top	&C::rtti	&C::f1	&C::f2	&C::f3

The bit vector for static types A, B and C will look like this:

Bit Vectors for A, B, C

Class	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
B	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

http://clang.llvm.org/docs/ControlFlowIntegrityDesign.html

RAP-RIP-ROP

- Indirect Control Transfer protection
- Type hash for indirect 'calls'
  - Return type
  - Function name
  - Function parameters

RAP-RIP-ROP

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

Perfect ?

- Still imperfect, however infinite times better!
- Exponentially raise cost (time) of exploitation
- Security becoming real deal

Intel – errata

- Some may argue that intel technology is weak because dont do type-hash etc
- However goal of this is not provide CFI, but do strict rule how to jump (kill ROP)
- Simple & effective
- CFI need to be handled compiler specific

Bit of c++

- Is this correct ?
- Is it even important ?
- Could it ends up with control flow control ?

```
class A
{
    virtual void foo(x,y,z);
}
class B : A
{
    virtual void foo(x,y,z);
}
..
B b;
...//meanwhile is arbitrary rewriten b.foo = &A.foo
b.foo(x,y,z);
```

Bit of c++

- Well … yes and no …
  - depends on quality of code
    - Inheriting vs limiting / extending, etc
  - concrete object context
  - Must be used with (limited) Kernel.Io()
  - Could allows to reach data gadget dispatcher
  - … what ?

Data Oriented Prg

- Turing complete

```
1 //dispatcher & jump :
2 void cmd_loop(server_rec *server,conn_t *c) {
3   while (TRUE) {
4     pr_netio_telnet_gets(buf, ..);
5     cmd = make_ftp_cmd(buf, ...);
6     pr_cmd_dispatch(cmd);  //calls functions
7           // with memory errors and gadgets
8   }
9 }
10 char *pr_netio_telnet_gets(char *buf,...) {
11   while(*pbuf->current != '\n' && toread>0)
12     //reads through virtual PC
13     *buf++ = *pbuf->current++;
14 }
```

Code 10. Gadget dispatcher and simulated jump gadget. `pbuf->current` is the virtual PC pointing to the malicious input.

- *(Conditional) jump operation.* Code 10 shows the ProFTPD program logic to read the next command from an input buffer. `pbuf->current` is a pointer to the next command in the input, thus forming a virtual PC for the attacker's MINDOP program. By corrupting `pbuf->current`, the attacker can select a particular input that invokes a specific MINDOP operation. We use the assignment operation to conditionally update the virtual PC, thus simulating a conditional jump operation.

https://www.comp.nus.edu.sg/~shweta24/publications/dop_oakland16.pdf

Data Oriented Prg

- However type-hash approaches ..
- #1 You have luck to hit vulnerability reachable at control flow of some gadget dispatcher
- #2 you need to find type-hash friendly candidate and prepare for redirection to it before you proceed to DOP

Bit of objective-C

```
void foo(x,y,z);
void fxx(x,y,z);

struct A
{
  decltype(&foo)* AFoo;
}
..
A a;
a->AFoo = foo;
...//meanwhile is arbitrary rewriten a.AFoo = &fxx
a.AFoo(x,y,z);
```

- This is more powerful to abuse
- some languages are build to be dynamic .. some of them not
  - Therefore harder to protect

OOP problems

- Data – Code mix
  - Objective-C
    - Hard to impossible to distinguish
  - C++
    - Can be distinguished
    - Built for dynamic code in mind

Joe Armstrong (2011)

Why OO Sucks

"Objects bind functions and data structures together in indivisible units. I think this is a fundamental error since functions and data structures belong in totally different worlds."

Data flow integrity

- DFI – data flow integrity

- Different set of instructions are usually responsible for some set of data
- Likely to work on same type of data

DFI - implementation

- Static data flow graph at compilation
- Checking per memory access ?
- Costly overhead

- Approximations ?

DFI - approximation

- Limit only to subset of functions
- Limit only to subset of data

- Leaves lot of attack surface
- Approximate more ?

DFI - approximation

- Maybe help of hardware ?
- Special instruction :
  - change *accessible range* mem-access instructions
    - Default should be current stack
  - Whoaa, somehow close to intel's mpx! ☺
- Fine grained pools & inside isolated heaps alikes
- Compilers + linkers with type scope information
- Data attacks will be solved 一步一步

https://software.intel.com/en-us/blogs/2013/07/22/intel-memory-protection-extensions-intel-mpx-support-in-the-gnu-toolchain

DFI - approximation

- However .. Based on current (pretty badass) implementation of DOP and its early results
- Seems to be fair enough to just protect potential gadget dispatchers
  - Numbers are tend to be small
- But this is more like state-of-art, and subsets of this approach still remains (kernel.Io()!)

ROP & CodeExec

- Those are results of non - security thinking (limitations) at the very beginning

- Seems to be solved
  - Unless logic bugs in solutions
  - However idea of solutions is pretty solid

Code Reuse & Data attack

- Can we say with all those potential mitigations (hardware & software) will be solved ?

Code Reuse & Data attack

- Not likely
- As those exploiting very principles of current architectures & dynamic code
  - Proper protection is hard (but looks like possible) for built-in dynamic languages
  - To what extend is it possible to languages dynamic by possibility not be design ?

Security

- However … with ongoing research (clang, pax, intel, ..)
- It will be far more harder to do successful attack
  - in many (many) cases will be even impossible
- Goal of success are however not solely mitigations, nor bugs killing itself
  - Both aspects is important and has its own role
  - Final outcome could be very secure environments

Back to the future

Still way to go..

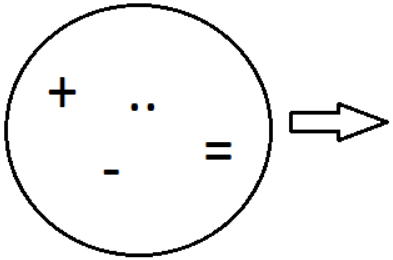
XCon安全焦点信息安全技术峰会

XCon XFocus Information Security Conference

Time to think

- What is SYSTEM / Kernel Code Exec ?
- Why we do need it ?
- How is it used ?
- What is real goal we want to achieve ?

DATA



1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101
1001010101010101001010101010101

DATA

- DATA are all what matters
- Code is just group of targeted instructions to work over those data
- You want read / write to data
- SYSTEM / kernel code exec did it for you in easy way

Kernel.Io()

- We have it from the time being
- Kernel.Io() is subset of DOP implementation
- We used it for making shortcuts

- Once you can read / write to targeted domain you won
  - no need to direct control flow control

Kernel.Io()

- Kernel.Io()
  - Vulnerability
    - Considering mem-corruption here
  - Technique
    - Leaking domain addresses
    - Reliable re-use multiple times
    - Deep domain knowledge
    - Code quality++

Kernel.Io()

```
bool
Write(
    __in_bcount(size) void* addr,
    __in_bcount(size) const void* buff,
    __in size_t size
    ) override
{
    return Io<true>(addr, const_cast<void*>(buff), size);
}
bool
Read(
    __in_bcount(size) const void* addr,
    __inout_bcount(size) void* buff,
    __in size_t size
    ) override
{
    return Io<false>(const_cast<void*>(addr), buff, size);
}
```

Kernel.Io()

- with Kernel.Io() you can :
  - do any mathematic operation on arbitrary data
  - emulate known 'api'
    - With no protection of stack-return it is pretty cheap

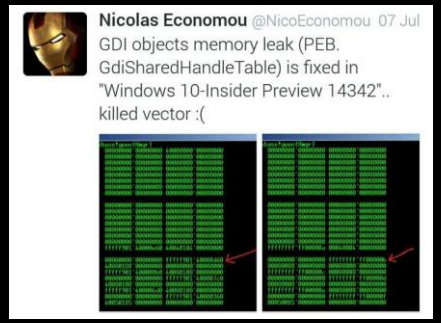
security

- SIDT / SGDT
- CR4.UMIP (documented at revision 58)

- Solves (trap to block) :
  - Leaks for user mode
  - Virtualization leak issue

Security [WINDOWS]

- gdi info leakage
  locked



kernel debug (x86)

```
kd> dq poi (win32kbase!gpentHmgr)
ffff8322'80210000  00000000'00000000 00000000'00000000
ffff8322'80210010  00000000'00000000 00000000'00000000
ffff8322'80210020  00000000'00000000 00000000'00000000
ffff8322'80210030  00000000'00000000 00000000'00000000
ffff8322'80210040  00000000'00000000 00000000'00000000
ffff8322'80210050  00000000'00000000 00000000'00000000
ffff8322'80210060  00000000'00000000 00000000'00000000
ffff8322'80210070  00000000'00000000 00000000'00000000
kd> d
ffff8322'80210080  00000000'00000000 00000000'00000000
ffff8322'80210090  00000000'00000000 00000000'00000000
ffff8322'802100a0  00000000'00000000 00000000'00000000
ffff8322'802100b0  00000000'00000000 00000000'00000000
ffff8322'802100c0  00000000'00000000 00000000'00000000
ffff8322'802100d0  00000000'00000000 00000000'00000000
ffff8322'802100e0  00000000'00000000 00000000'00000000
ffff8322'802100f0  ffffffff'ff00000a 00040004'00000000
kd>
ffff8322'80210100  00000000'00000000 ffffffff'ff00000b
ffff8322'80210110  00080088'00000000 00000000'00000000
ffff8322'80210120  ffffffff'ff00000c 00080008'00000000
ffff8322'80210130  00000000'00000000 ffffffff'ff00000d
ffff8322'80210140  00080008'00000000 00000000'00000000
ffff8322'80210150  ffffffff'ff00000e 00080008'00000000
ffff8322'80210160  00000000'00000000 ffffffff'ff00000f
ffff8322'80210170  00050085'00000000 00000000'00000000
kd>
```

Security [WINDOWS]

- w32k lockdown
  - dont touch what you dont need
- ntos restricted access
  - Low attack surface
  - Code quality good

Conclusions

- Hardware setting security boundaries at possibility level
- Software continues to tackling attackers techniques to the edge
- Vulnerabilities space is shrinking
  - Notable by security research from china

Way to go

- Technologies benefits one from another, making software more secure!

Thank you!

Q & A

科恩实验室

KEEN security lab