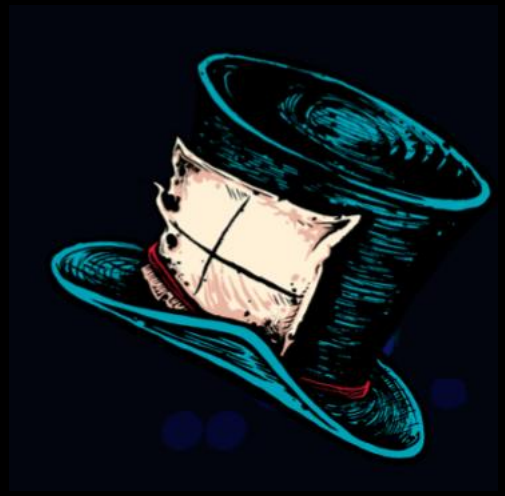
Tencent

You didnt see it’s coming?

“Dawn of hardened Windows Kernel”



KEEN security lab

$whoami

Peter

- @zer0mem
- Windows kernel research at KeenLab, Tencent
- pwn2own winner (2015 / 2016), pwnie nominee (2015)
- fuzzing focus : state
- wushu player

Tencent

KEEN security lab

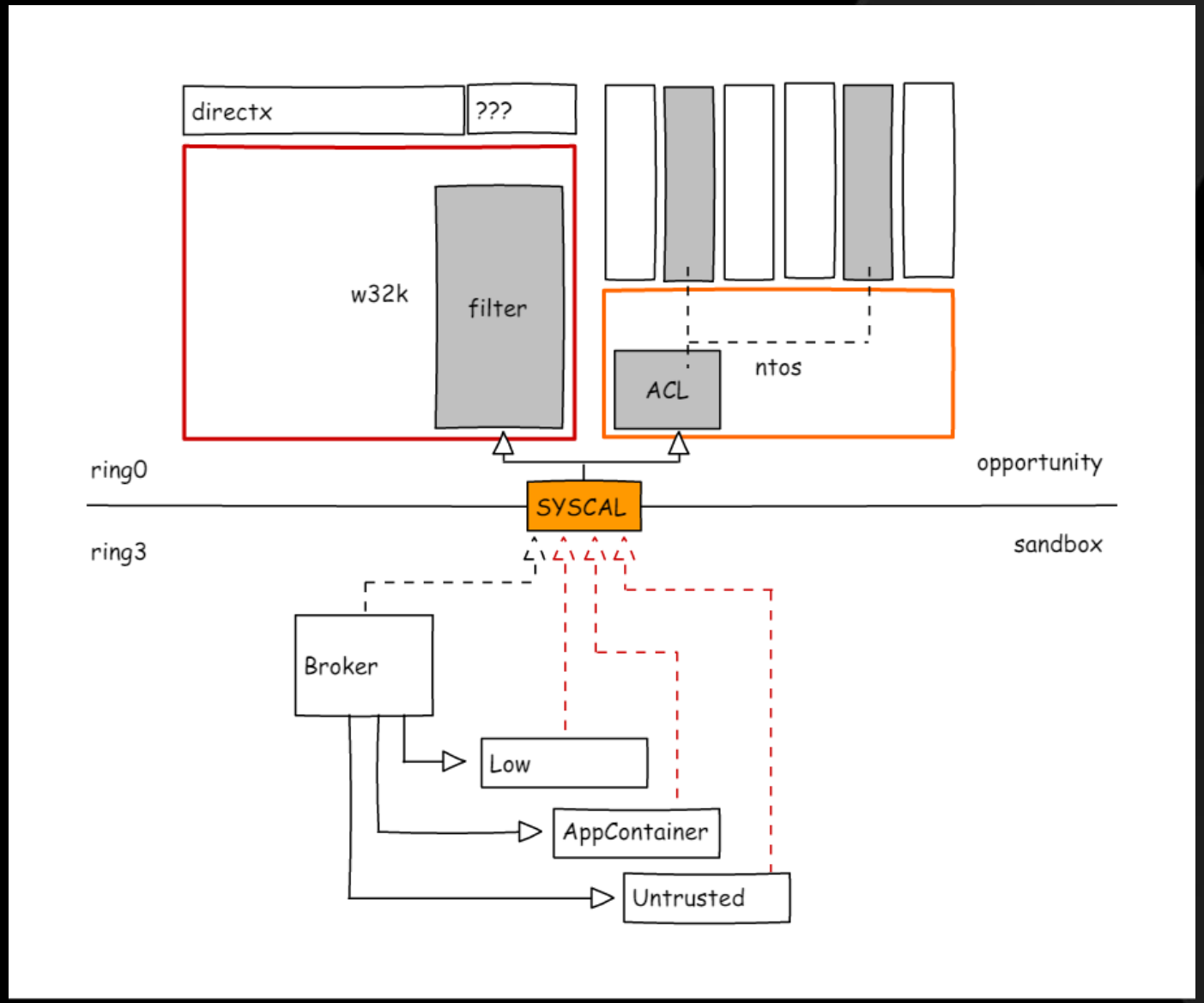
agenda

- Kernel attack surface
- Fuzzing
- Exploitation techniques
- Mitigations

Tencent

KEEN security lab

windows sandbox kernel attack surface



Tencent

KEEN security lab

path of less resistance - w32k*

- resides in ring 0
  - natural := (level < medium) -> SYSTEM (/kernel code exec) escape
- huge attack surface
  - huge in comparsion to ntoskrnl counterpart or in-ring3-sandbox interface
  - state logic, window callbacks, hidden syscalls, directx, format parsing, ...
- accessible from sandbox-es
  - nowdays more or less => big success!
- easy develepoing of exploitation techniques

w32k vs 2016

MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Information Disclosure Vulnerability	CVE-2016-017
MS16-062	Microsoft DirectX Graphics Kernel Subsystem Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-019
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-019

technet.microsoft.com/en-us/library/security/mt674627.aspx

win32k | 1 of 33 | Options

Security Advisories and Bulletins > Acknowledgments

2016

Acknowledgments – 2016

MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3249
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3250
MS16-090	GDI Component Information Disclosure Vulnerability	CVE-2016-3251
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3252
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3254
MS16-090	Microsoft win32k Elevation of Privilege Vulnerability	CVE-2016-3286

	Win32k Elevation of Privilege Vulnerability	CVE-2016-3308
	Win32k Elevation of Privilege Vulnerability	CVE-2016-3309
	Win32k Elevation of Privilege Vulnerability	CVE-2016-3310
	Win32k Elevation of Privilege Vulnerability	CVE-2016-3311

Tencent

KEEN security lab

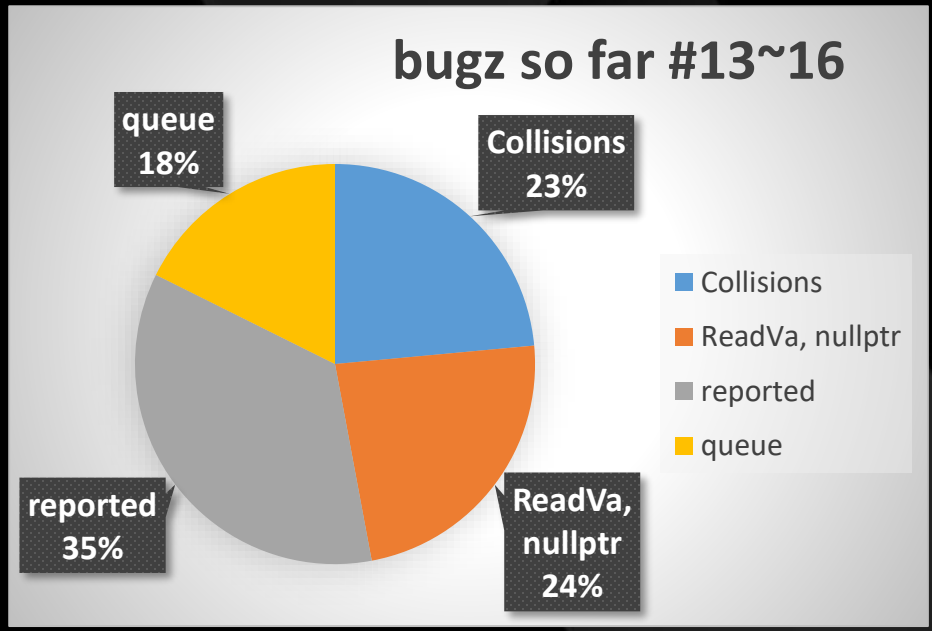
what is going on ?

- huge numbers of syscalls
- lot of objects
- lot of hardcore graphics stuffs
- lot of things i dunno

```
C:\>cat w32k@subsurface | grep "Nt" | wc -l
1042
```

w32k vs Qilin

- Qilin - internal multi-platform fuzzing framework
- Target – gdi - part of w32k
- Technique :
  - knowledge based
  - random driven
  - interconnection aware
- Results in graph
- technique ++ code coverage
  - feedback + knowledge based (ongoing)



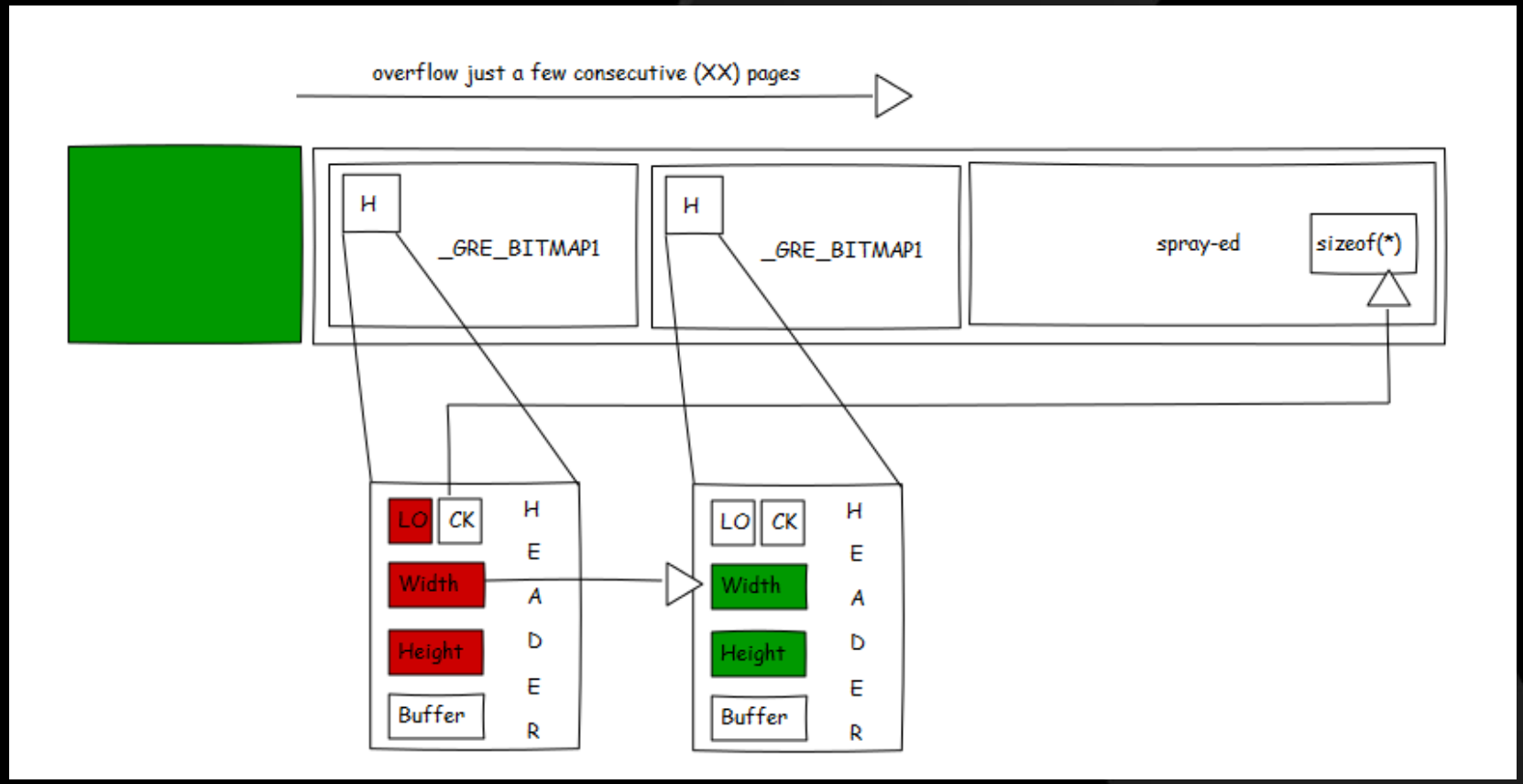
http://www.slideshare.net/PeterHlavaty/rainbow-over-the-windows-more-colors-than-you-could-expect

from bug to take over SYSTEM

- except huge code base with lot of space for bugs
- huge code base with lof of space for missusing existing mechanisms
- valuable arsenal
  - virtual tables and alikes
  - unprotected lengths
  - plain state members
  - buffer pointers
- prequisites :
  - call syscalls

W32k kernel io : _gre_bitmap

- Simple & *reliable* pool layout
- Direct syscalls for content manipulation
- Pivot -> worker technique



w32k out of scope of interest ?

- somehow :
  - going to be locked down & filtered out
  - not so cool anymore, many research done, many bugs out, ...
  - so complex so it even gets boring after time of researching there
  - if you trying to find something -> you will : if no, then try harder...

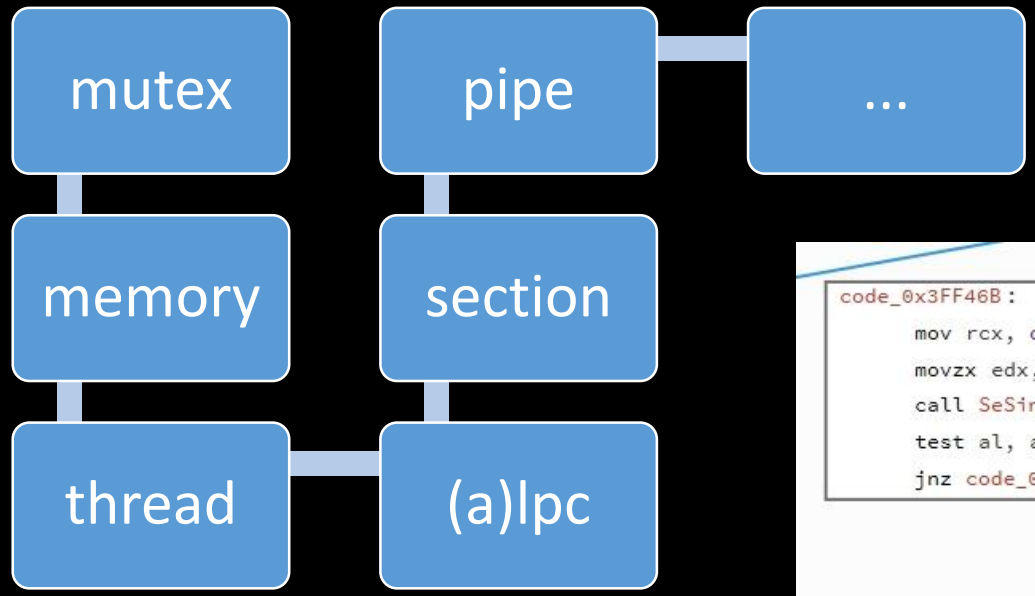
- however in nowadays still applies (and some time will)
  - weakest points of windows kernel
  - accessible from most sandboxes
  - best attack vector nowadays
  - attackers are lazy, why do something hard if no need ?

PERFECT GROUND FOR YOUR FUZZER

Tencent

KEEN security lab

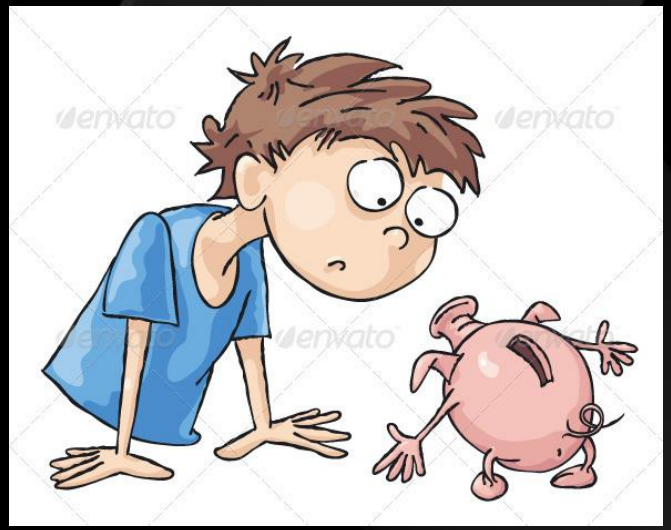
ntos : lets move from win32k




```
code_0x3FF46B:
        mov rcx, qword [data_0x742CE0]
        movzx edx, r12b
        call SeSinglePrivilegeCheck    ; unsigned char __cdecl SeSinglePrivilegeCheck( ur
        test al, al
        jnz code_0x3FF3C6
```

```
code_0x3FF483:
        mov edi, 0xC0000061
```

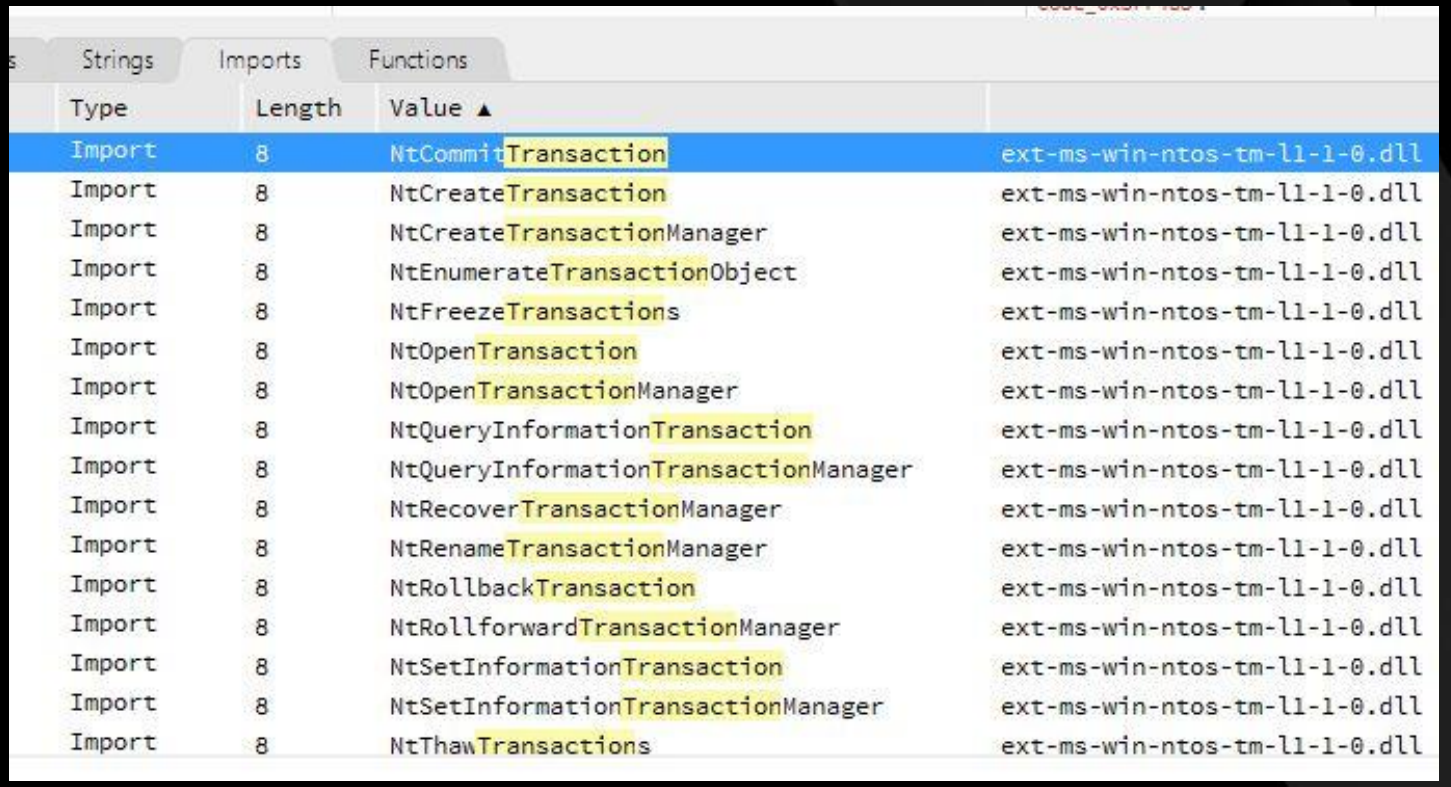
ntos - attack surface



- difference :
  - seemingly no data parsing +-
  - state only
  - from untrusted level / app container not much to touch
  - small number of state changing syscalls
  - state changes are minimalistic in most cases*

ntos under the microscope - extensions

- Nt*Transaction*
- Nt*Enlistment*
- Nt*Manager*



Type	Length	Value ▲	
Import	8	NtCommitTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtCreateTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtCreateTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtEnumerateTransactionObject	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtFreezeTransactions	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtOpenTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtOpenTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtQueryInformationTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtQueryInformationTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtRecoverTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtRenameTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtRollbackTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtRollforwardTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtSetInformationTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtSetInformationTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
Import	8	NtThawTransactions	ext-ms-win-ntos-tm-l1-1-0.dll

tm.sys

- ntos extension
- Kernel Transaction Manager
- however not much attack surface as well
- simple states & minimum syscalls
- however interconnections

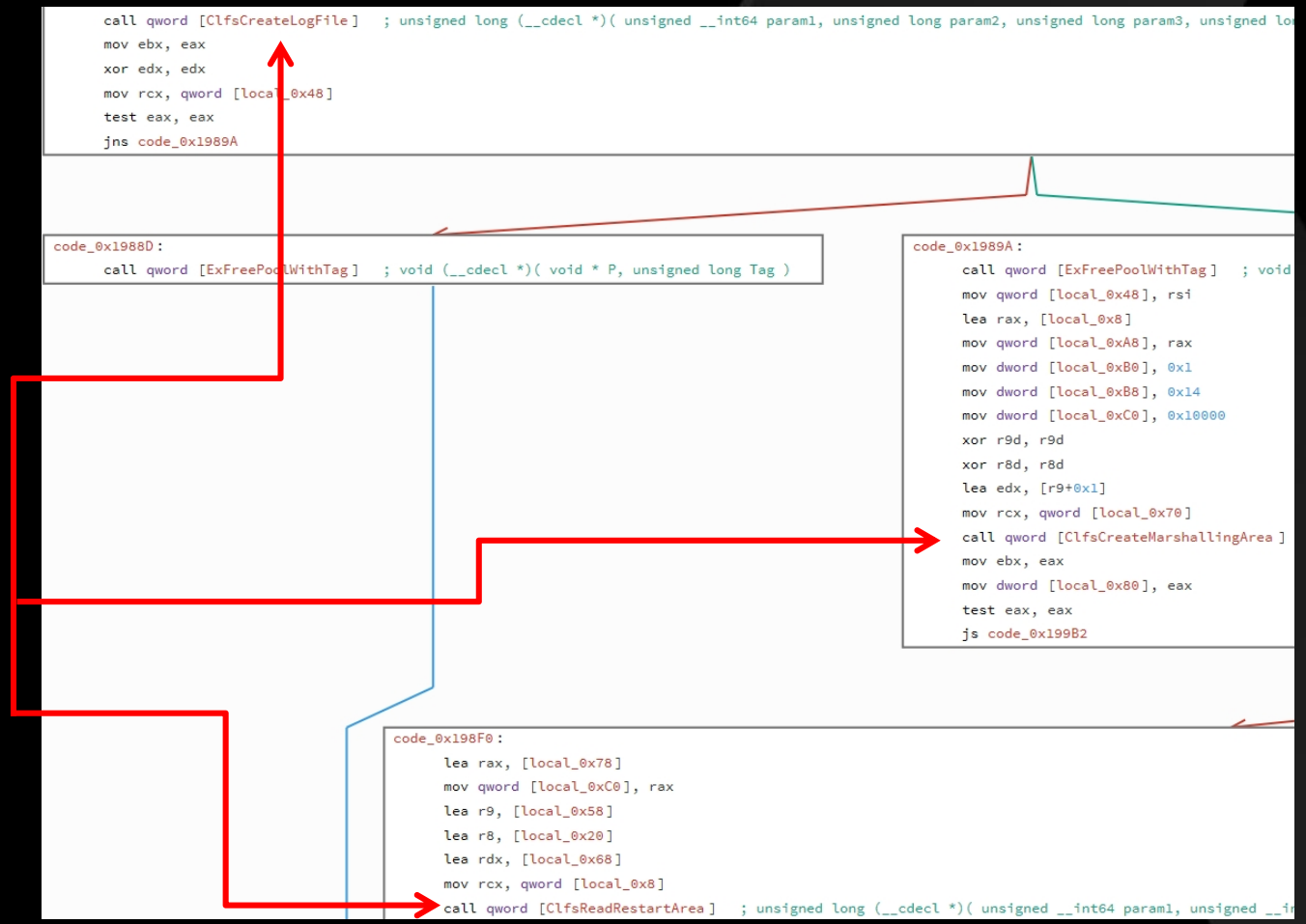
- Qilin vs tm, round1 :
  - 1 nullptr
  - 1 exploitable double ObDeref -> type confusion -> uaf

Tencent

KEEN security lab

explore unknown : indirections++

- tm.sys simple
  purpose driver
- but interesting
  module involved
  at backend
- CLFS.sys



CLFS.sys : unseen hand behind the scenes

- Common Log File System
- not everything need to be direct
- easy to get interest if you look at simplistic tm.sys
- clfs used at many different places in ntos itself as well

- clfs on other hand more complex
  - c++ code base
  - complex state
  - involve data parsing
- Qilin vs clfs : 3~5 state bugs, 6~8 data parsing
- Advanced *data* fuzzer : covers all first hand bugs, and extend 4+
  - by @long123king

Tencent

bug hunting

research, fuzzing, feedback, analysis, loop back

KEEN security lab

kick off

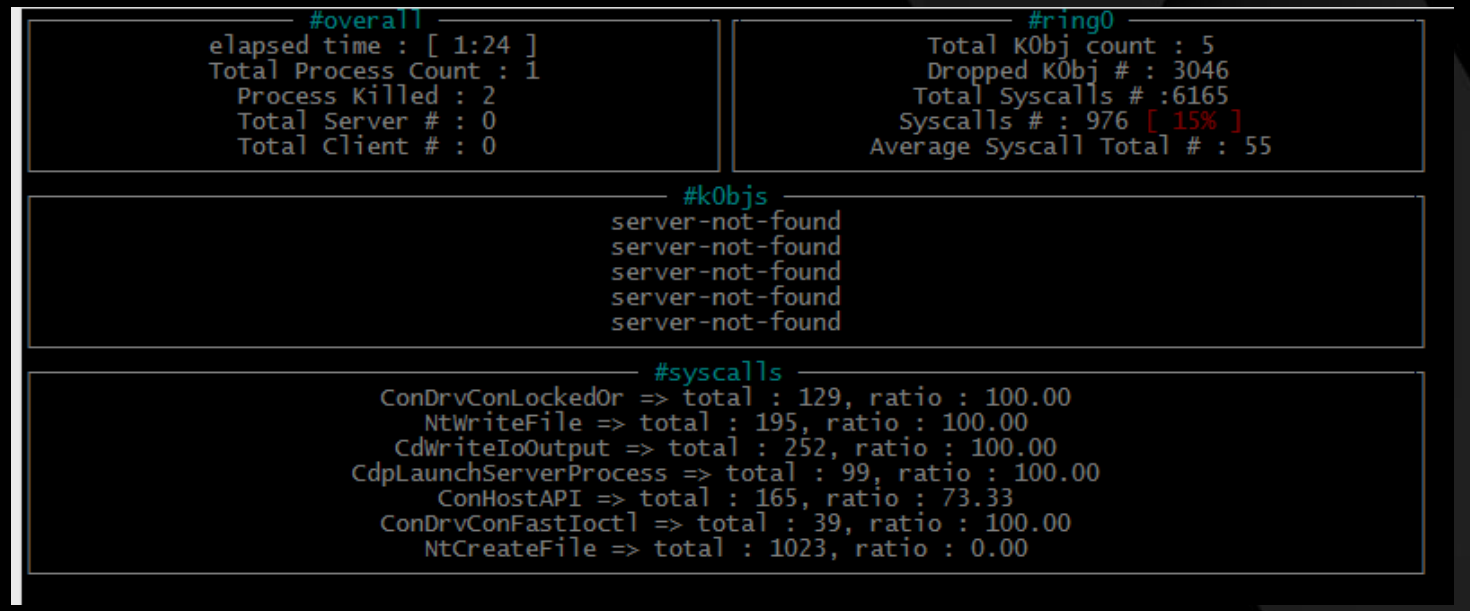
- select target
- collect related syscalls (apis)
- research attack surface
  - go trough related msdn part (if any)
  - understand abstracted functionality
    - what it is purpose
    - what is usual way of working with it
  - double check interesting points with kernel implementation

fuzzer

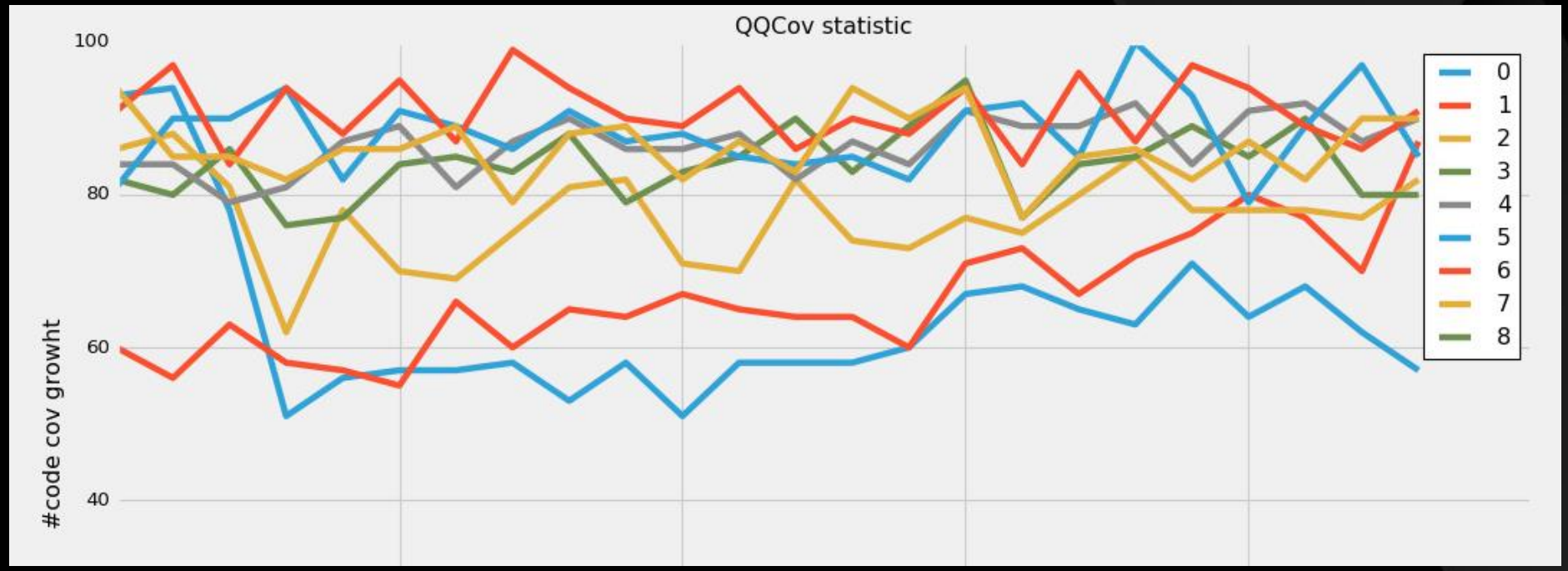
- make some code to automate working with target
  - optionally use existing framework, just add functionality per target
- make it work
  - most basic syscalls working (open, close, get)
- make use of previous reseach
  - implement basic schema from previous step
  - ensure that your fuzzer can sucessfully work on target based on documentation of target
  - introduce randomnes with preserving certain sucess ratio
  - introduce connections (syscall & handle dependencies)

feedback

- how good is your fuzzing ?
- sucess ratio % ?
- code coverage ?

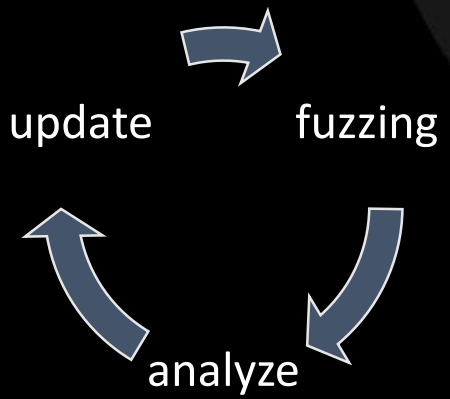


meassure your fuzzing



research

- fuzzing is one part
- you want more bugz ? you need know your target well
- not only target, but your fuzzer as well



- analyze less covered parts
- think of possible buggy scenarios
- implement back to fuzzer
- loop back (from documentation, trough fuzzing up to understanding)

additional thoughts

- fuzzing is not enough
  - you need to understand what you are fuzzing
  - better understanding helps to build more tricky logic
    - hower once you concrete too much, then you puting yourself into corner
- better to be backuped with code coverage corpus
  - corpus is essential to keep & update, but not recycle (scatter&replace) often
- fuzzer is not all you need
  - runtime tools [adress sanitizers, race tools, debugger plugins, ..]
  - static tools [helping your fuzzer, helping your analysis]

Tencent

lets move to exploitation techniques!

KEEN security lab

options

- w32k easy to go, however not so easy from locked down / filtered state anymore
  - can go trough chaining, but we can do better
- ntos, harder to go
  - not much available syscalls - harder to setup & control technique properly
  - not much used before - why to go harder way if no need ?
  - less objects, with less nasty states

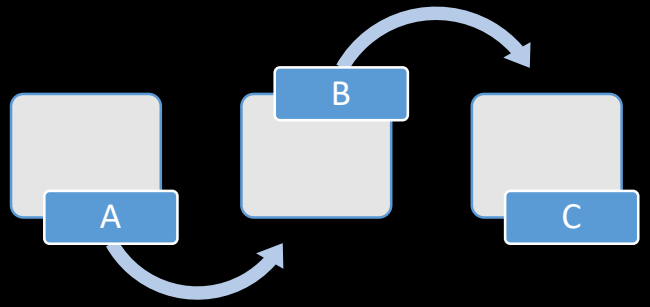
nt!KeWaitForSingleObject

- small object
- relativelly simple
- interesting logic behind usage
  - lock / unlock
  - wait
  - kernel pointers
  - ethreads
- offers various primitives :
  - arbitrary decrease
  - arbitrary write where - kernel pointer*
  - ...

```
nop     dword ptr [rax+rax]
inc     ebp
test    dword ptr [nt!HvlLongSpinCountMask (fffff802`ce0281c0)],ebp
je      nt! ?? ::FNODOBFM::`string'+0x39f0 (fffff802`cddddbb0)
pause
mov     eax,dword ptr [rdi]
test    al,al
js      nt!KeWaitForSingleObject+0x440 (fffff802`cdcbc610)
lock bts dword ptr [rdi],7
jb      nt!KeWaitForSingleObject+0x440 (fffff802`cdcbc610)
lea     rdx,[rbx+140h]
```

SafeLink write where primitive

- Introduced to deal
  meta data link corruptions
- previously lead to :
  write where - what
- now it leads to int 0x29
  ... not necessary, or ?

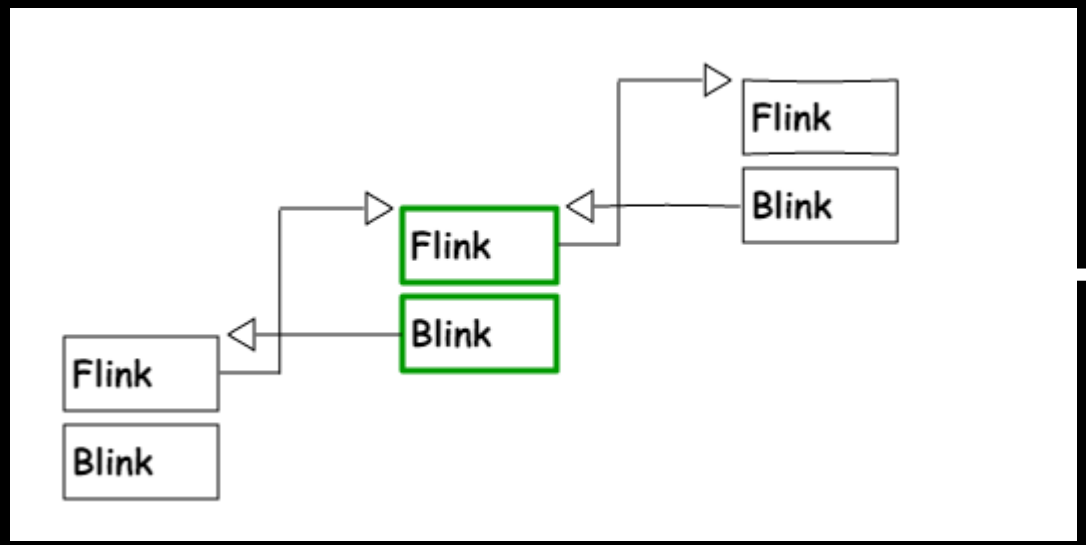


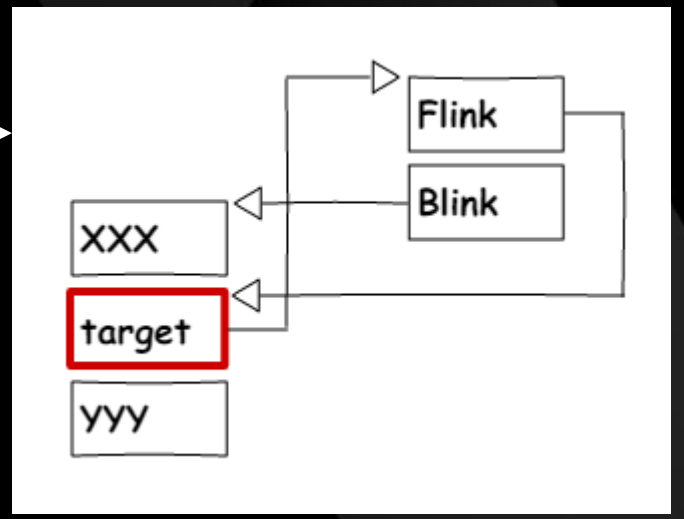

```
286:
ebp, 0x102
code_0x1D2B3

code_0x1D079:
        mov rcx, qword [r14+0x10]
        lea rax, [r14+0x8]
        cmp qword [rcx], rax
        jnz code_0x1D28D

code_0x1D28D:
        mov ecx, 0x3
        int 0x29
code_0x1D294:
        movzx ecx, byte
        movzx eax, cl
        and al, 0x7
        cmp al, 0x1

code_0x1D08A:
        mov qword [r10], rax
        mov qword [r10+0x8], rcx
        mov qword [rcx], r10
        mov qword [rax+0x8], r10
```

Arbitrary write – kernel pointer

- target needs to point to semaphore / event
- Problem ? .. Well not much :
  - You can predict kernel mode memory members
  - You can misalign

Flink –> Blink == Blink -> Blink





http://www.slideshare.net/PeterHlavaty/how-safe

ALPC

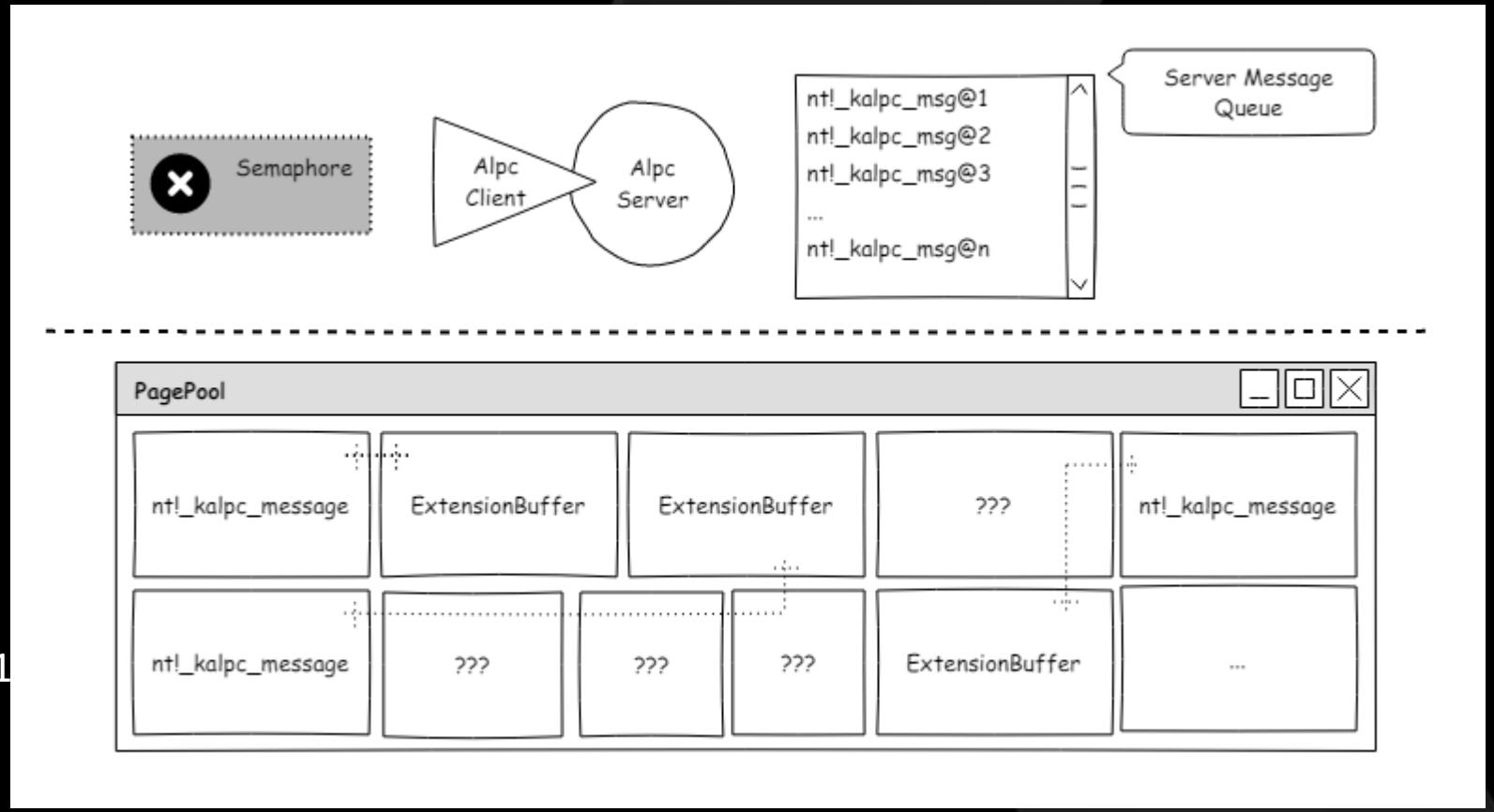
- not everything touchable in ntos is really minimalistic
- more complex mechanism
- more complex state
- good interconnections

- .. however well written code ..
  - lot of safe checks
  - safe user mode memory handling (via 'getters')
  - seems well designed

http://alex-ionescu.com/Publications/SyScan/syscan2014.pdf

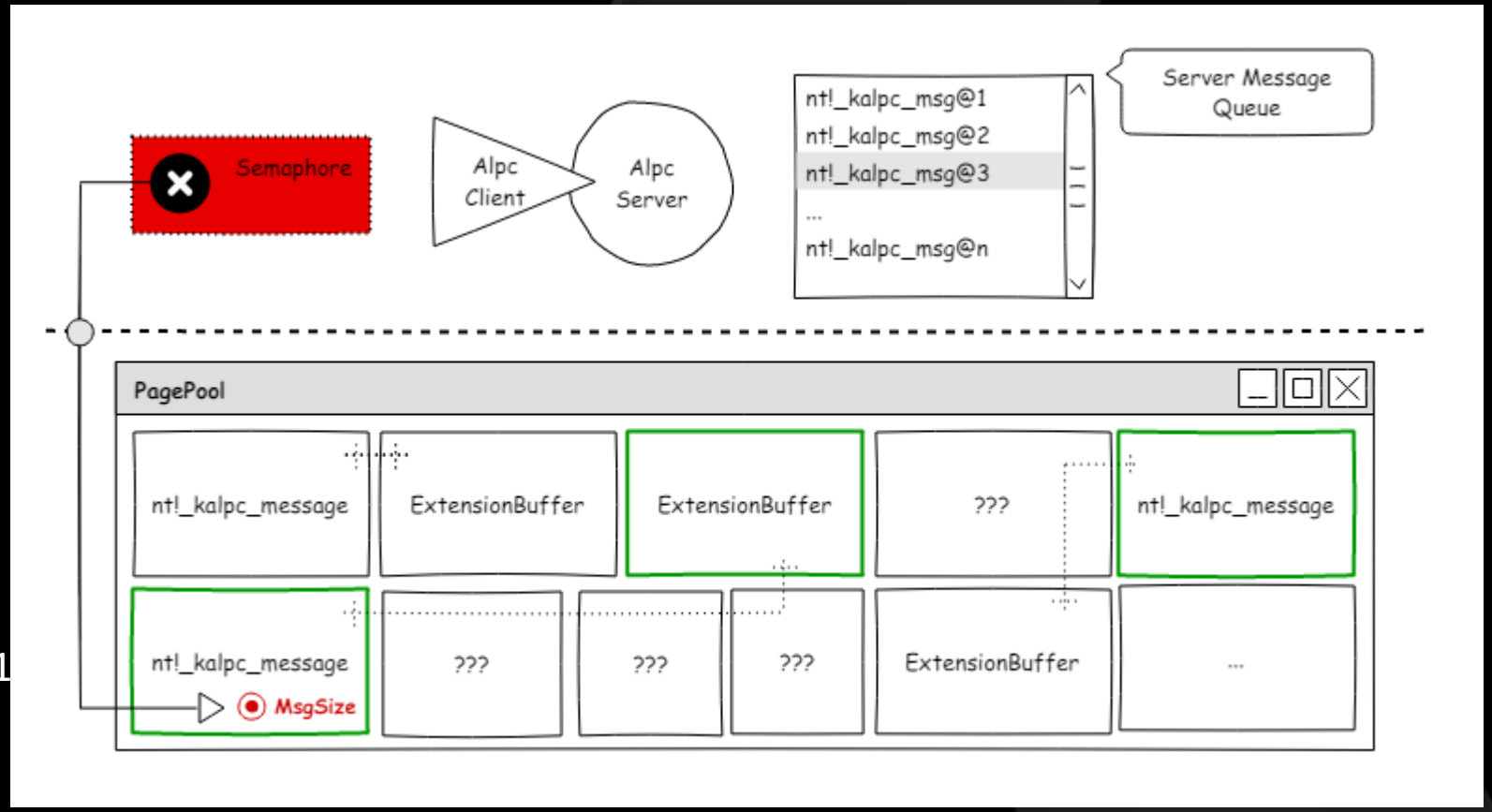
ALPC #spray

```
kd> dt nt!_kalpc_message
   …
   +0x010 PortQueue
   +0x018 OwnerPort
   …
   +0x068 MessageAttributes
   +0x0b0 DataUserVa       : Ptr64 Void
…
   +0x0d8 ExtensionBuffer  : Ptr64 Void
   +0x0e0 ExtensionBufferSize : Uint8B
   +0x0e8 PortMessage : _PORT_MESSAGE
kd> dt nt!_kalpc_message PortMessage->u1
   +0x0e8 PortMessage      :
      +0x000 u1            :
         +0x000 s1         : <unnamed-tag>
         +0x000 Length     : Uint4B
      +0x004 u2
```



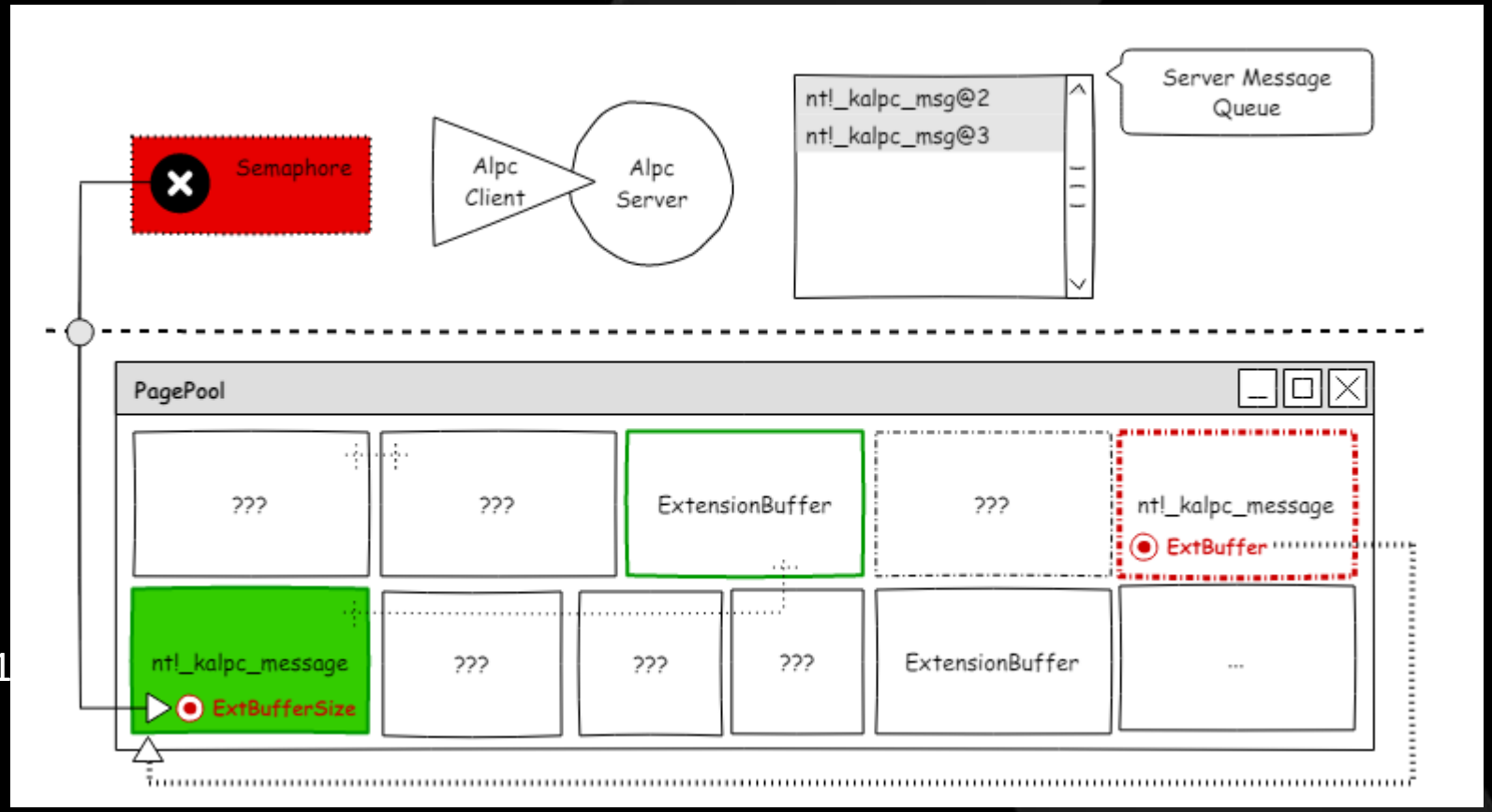
ALPC #detect

```
kd> dt nt!_kalpc_message
   …
   +0x010 PortQueue
   +0x018 OwnerPort
   …
   +0x068 MessageAttributes
   +0x0b0 DataUserVa       : Ptr64 Void
…
   +0x0d8 ExtensionBuffer  : Ptr64 Void
   +0x0e0 ExtensionBufferSize : Uint8B
   +0x0e8 PortMessage : _PORT_MESSAGE
kd> dt nt!_kalpc_message PortMessage->u1
   +0x0e8 PortMessage      :
      +0x000 u1            :
         +0x000 s1         : <unnamed-tag>
         +0x000 Length     : Uint4B
      +0x004 u2
```



ALPC #overflow

```
kd> dt nt!_kalpc_message
   …
   +0x010 PortQueue
   +0x018 OwnerPort
   …
   +0x068 MessageAttributes
   +0x0b0 DataUserVa       : Ptr64 Void
…
   +0x0d8 ExtensionBuffer  : Ptr64 Void
   +0x0e0 ExtensionBufferSize : Uint8B
   +0x0e8 PortMessage : _PORT_MESSAGE
kd> dt nt!_kalpc_message PortMessage->u1
   +0x0e8 PortMessage      :
      +0x000 u1            :
         +0x000 s1         : <unnamed-tag>
            +0x000 Length  : Uint4B
      +0x004 u2
```



ALPC #io

- #4 threads
  (gdi tech, only #1 thread)
  - io
  - pivot
  - worker
  - server

```
void
PushRequest(
    CAlpcMsg& msg
    )
{
    CAlpcClientWorker worker(m_client, msg);
    worker.SendTcp();//let server to handle re
}
```

- Performing additional write per io !
- synchronization
- Undocumented ALPC
  - Setup server – client
  - Keeping messages alive
- pool feng shui is problematic

```
void
KernelIoWrite(
    uint8_t* addr,
    const uint8_t* mem,
    size_t size
    )
{
    if (!size)
        return;
    if (size % sizeof(void*))
        return;

    m_ioAddr = nullptr;

    std::thread pivot(PushRequest, this, m_pivot);

    while (!m_ioAddr)
        Sleep(100);

    std::thread worker(PushRequest, this, m_worker);

    Sleep(200);//wait until worker will be in queue

    memcpy(m_worker.IoData(), mem, sizeof(void*));

    m_ioAddr = addr; // pivot start operate!

    pivot.join();
    worker.join();

    return KernelIoWrite(
        addr + sizeof(void*),
        mem + sizeof(void*),
        size - sizeof(void*));
}
```

Tencent

KEEN security lab

Tencent

Hardenings !?

SMEP, KASLR, Nx, CFG, RFG, ACL, SafeLink ...

KEEN security lab

from user to kernel, from kernel to jail : w32k

- Filtering
  - ? win32k – win32kfull – win32kbase ?
    - win32k -> wrapper to add stub* with access checks
  - restrict access only to necessary parts
    - maybe not 'clean' solution, but security++ (relatively)
  - app can have different sandboxed entities, and per entity different attack surfaces
    - find way from one to another, and you can find way out of filter
  - in optimal scenario limit potential bug landscape
    - first part of good effect
  - in optimal scenario limit exploitation techniques
    - this is the second good effect
- lockdown
  - wow, this sure solve something in crazy *absolute* way O_o

w32k is here for you if you need it!

- One can have arbitrary decrease ? [see previous semaphore slides]
  - Or any suitable primitive, just necessary be creative
- Would he choose alpc technique ?
- If he is lazy (smart) enough, then no …

```
ffff9784`e97a4800 1020 MicrosoftEdge.
ffff9784`e874c800 1764 browser_broker
ffff9784`cdbbc800 1824 MicrosoftEdgeC
0: kd> dt _eprocess ffff9784`cdbbc800  EnableFilteredWin32kAPIs DisallowWin32kSystemCalls AuditFilteredWin32kAPIs
nt!_EPROCESS
   +0x300 DisallowWin32kSystemCalls : 0y0
   +0x6c4 EnableFilteredWin32kAPIs  : 0y1
   +0x6c4 AuditFilteredWin32kAPIs   : 0y1
```

- Re-enable w32k instead!
  - even allow more proc in your job in case of need
- do one-bit kernel pwn via win32k!_gre_bitmap and enjoy life!

kernel code exec vs mittigations

- KASLR, code signing, Nx, CFG, SMEP

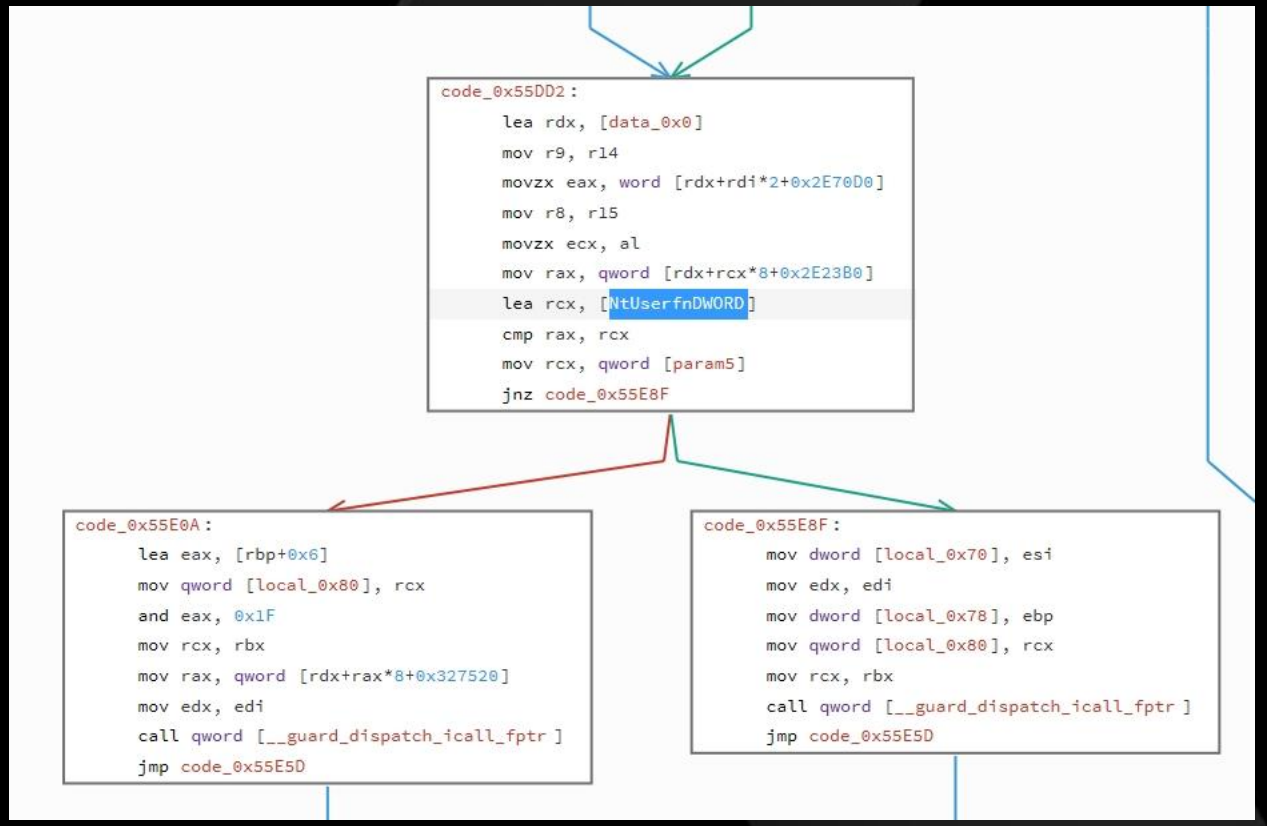
- set some good security boundaries
- however getting code exec no problem after all
  - kaslr - good bug or additional info leak
  - code signing : we will ship our code to kernel via kernel io, and exec it
  - Nx + SMEP : find rwe page, or create it
  - CFG : find good trampoline or target stack

easy to be said, harder to be done ?

- not actually
- kernel-io techniques already described
- still headache with kernel Nx / SMEP ?
  - NtUserMessageCall [following slide]
- CFG
  - prevents only from ROP to be kicked of
  - i dont like ROP neither, target stack [+ functions]
  - CFG alone is not enough (in terms of code exec) !

NtUserMessageCall

- Function of many faces
- 7 parameters
- Full control over them
- Indirect call invoked
- 7+ member of vtable <- dummy!
- vcall return is passed to user!

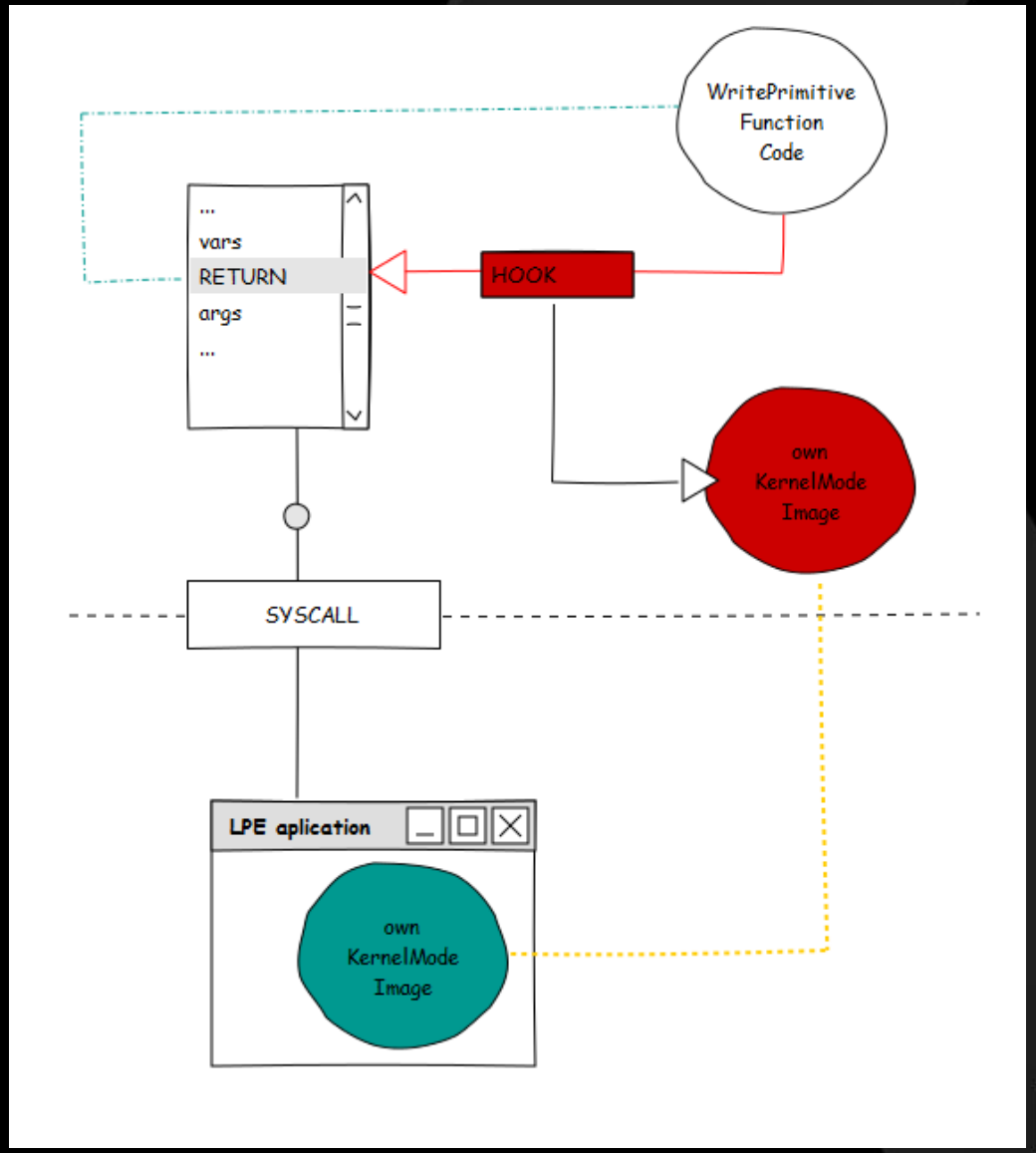


- Enough primitives for kernel code exec, with valid function (no ROP) !

Get kernel code exec

- Misuse existing functions
  - NtUserMessageCall
- Allocate RWE memory
- memcpy you kernel driver
- Stack hook

- Game over



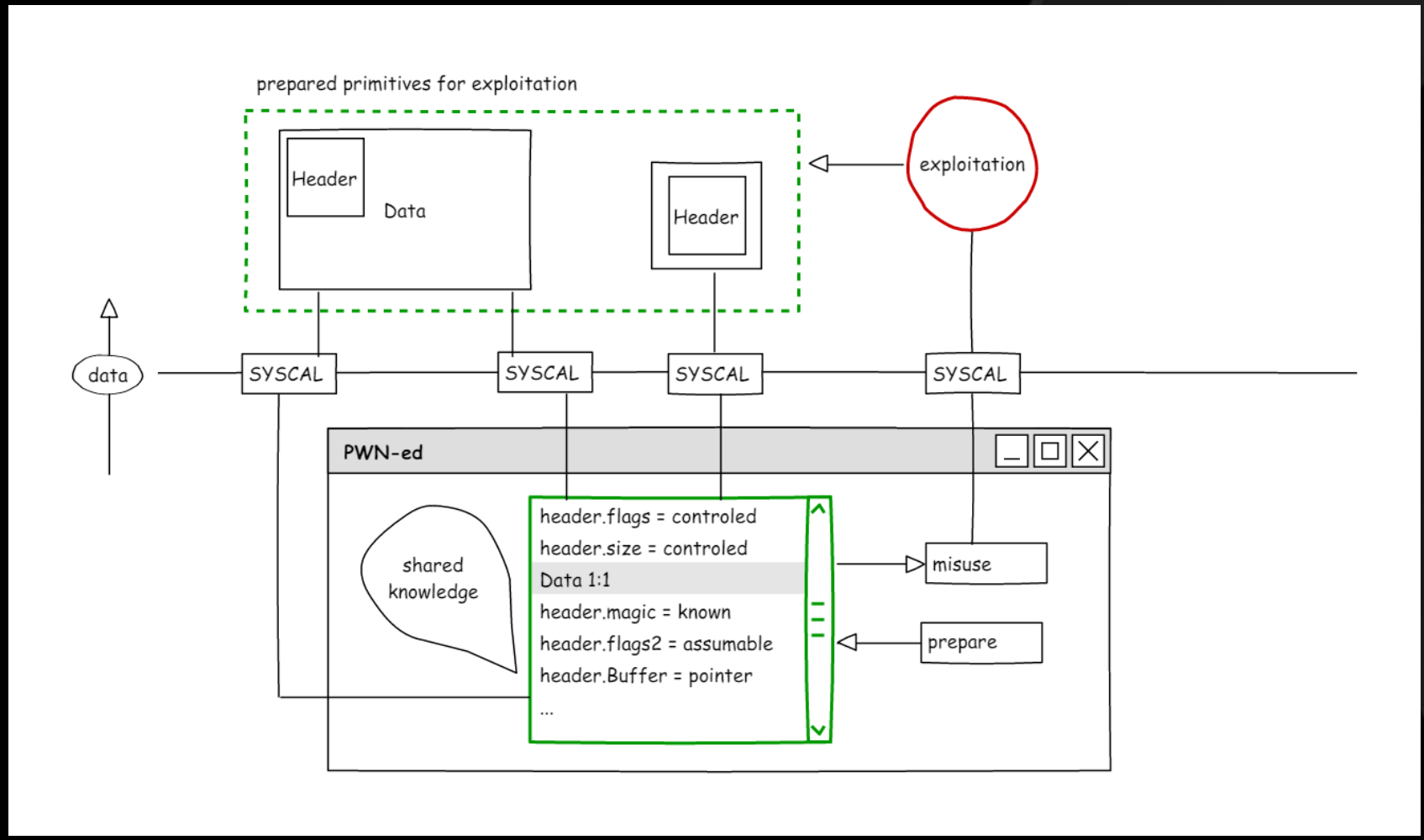
SMAP - tale of lacking feature

- prevents kernel unwanted access to user mode memory
- sets bare metal borders between data in user mode and kernel
- should serve as fine security feature … does it ?

Tencent

KEEN security lab

SMAP - tale of lacking feature



SMAP - tale of lacking feature

... however ...

- without proper security schema in underlying OS, it is just shinny ...
  - nullptr deref protection - anything bellow 0x10000
  - poisons should be *not-mappable* by user by default!

  - you need controlled data at exploitation ? why you need them in user mode ?
  - easy to put in kernel (pipes, direct mem mapping, physmap, kernel stack, ...)
  - in occasional cases one more bug to leak address layout
    - in most cases you can get them out from mem corruption bug (unprotected raw pointers, ...)

- security features is good to have, but better to use to max potential
  - which OS nowadays is using SMAP in way that it is really obstacle for attackers ?

Tencent

KEEN security lab

Control Flow – hardware!

- CET-IBT
  - Indirect jumps / calls + endbrXX
  - Similar idea with CFG indeed
  - Your jumps can not go wild, like ROP do!
- CET-Shadow Stack
  - nicely done!
  - You can not subvert control flow via stack hooking anymore!
- so seriously, what now ?
  - … wait for some time to adapt to mainstream O_o

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

Tencent

KEEN security lab

Return Flow Guard

- Rumors about RFG
  - Return Flow Guard
  - Preliminary documentation (nice!) :
    http://xlab.tencent.com/en/2016/11/02/return-flow-guard/
  - Simple & fast & effective
- Effective ?
  - Shadow Stack alike, therefore yes!
    - .. and we are back with FS on x64! fs:[rsp]
    - **arbitrary read / write will not touch it!**
  - in combination with CFG ? – hell, a lot!
    - Two simple & effective mitigations together are pretty solid!
    - Can be implemented with todays hardware!

3.4 RUNTIME INSERTED EPILOGUE BYTES (15 BYTES)

```
MiRfgInstrumentedEpilogueBytes
mov     r11, fs:[rsp]
cmp     r11, [rsp]
jnz     _guard_ss_verify_failure
```

https://people.eecs.berkeley.edu/~dawnsong/papers/osdi14-paper-kuznetsov.pdf

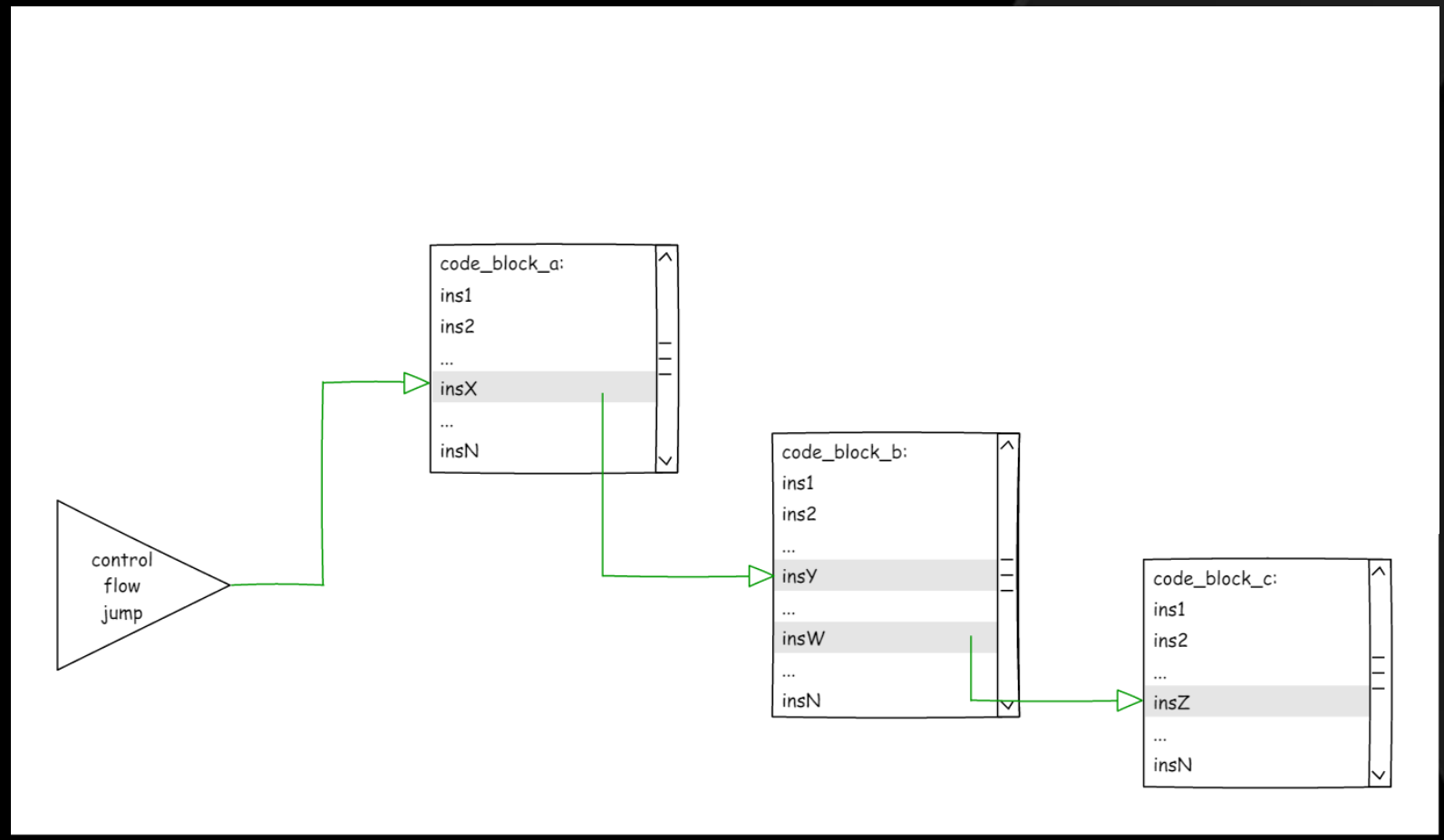
{C/R}FG vs CET vs CFI

- CET-IBT
  - some criticize that it lacks type integrity hash checks
  - It sets bare metal rules – how calls / ret must behave
  - CET is *not* about control flow *integrity* in particular
  - Implies : ROP is over but you can jump to *any* function instead!
- CFG + RFG
  - very close to CET
  - Added some integrity checks on top of it
    - black-list (not include into bitmap) “dangerous” functions
  - Implies : ROP is over but you can jump to large *subset* of functions instead!
    - depends on corner cases however
    - As you can see CFG improvements over time, cat – mouse game ?

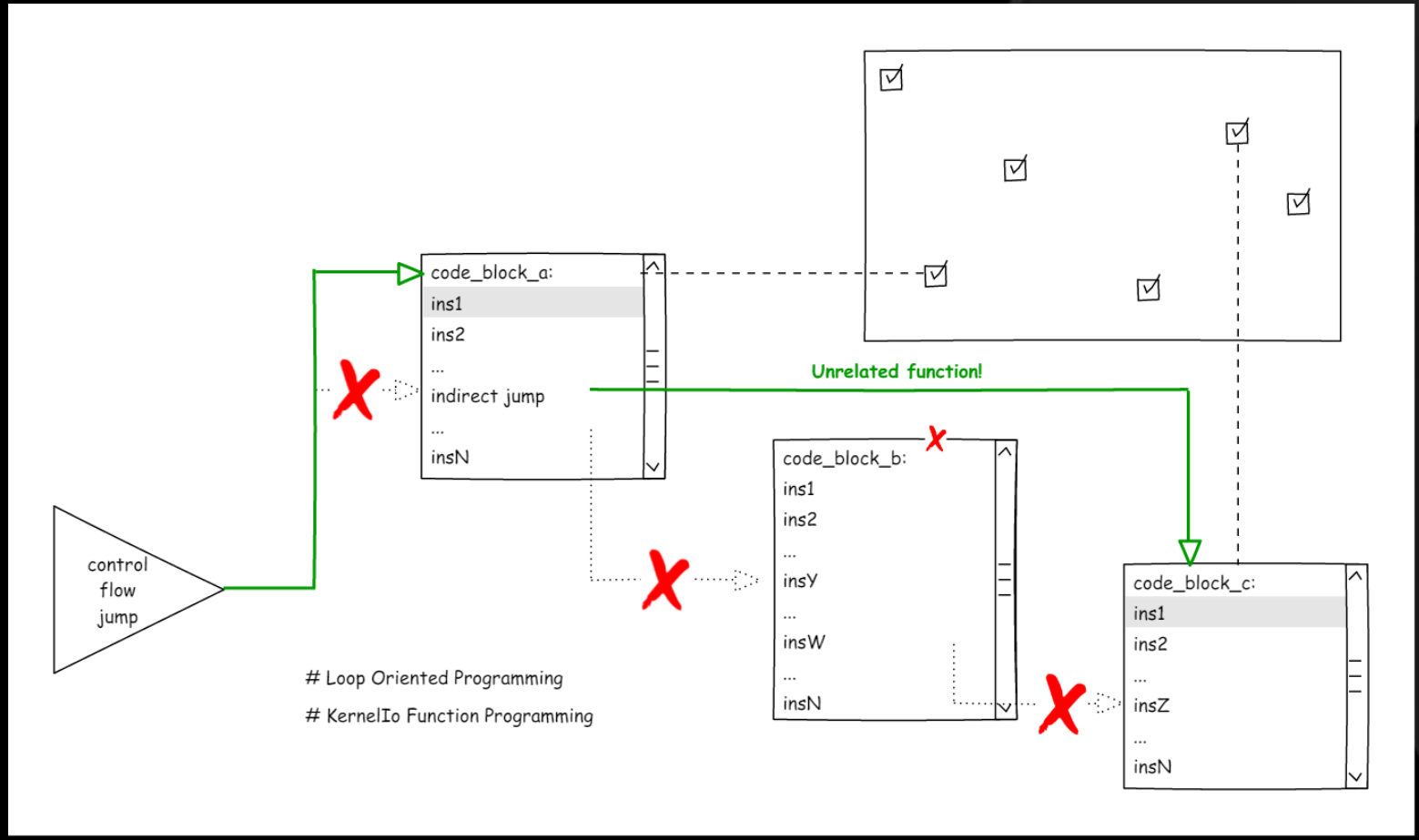
Tencent

KEEN security lab

ROP – wild technique



Function oriented - restricted technique

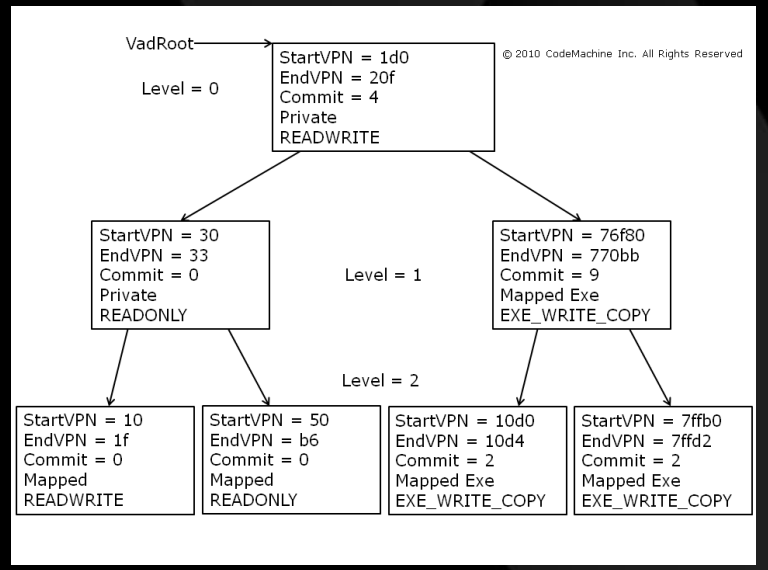


http://dl.acm.org/citation.cfm?id=2848479

CET-{IBT+ShadowStack} / {C/R}FG

- Gadgets == functions (Loop dispatcher / KernelIo)
  - Proper CFI needed (clang / pax)
- Kernel Code Exec, doable ?
  - Well, if you want install old school rootkit, doable
    - RWE on kernel component in your reach
    - Patch legitimate kernel
    - Jump
    - Patch back
  - Obviously bad code!
  - any reasonable benefits from kernel code exec ?
- Kernel Code Exec, needed ?
  - do you even need your own _eproc ?
  - why don't just use kernel io (legit non-harm) syscalls
  - force(trick) other _ethreads to do job for you
  - Patching their stacks (params / rets) should do the job

(VadRoot technique)



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

http://www.slideshare.net/PeterHlavaty/back-to-the-core

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

Data attacks! - TODO

What is your goal ?

DOP – 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.

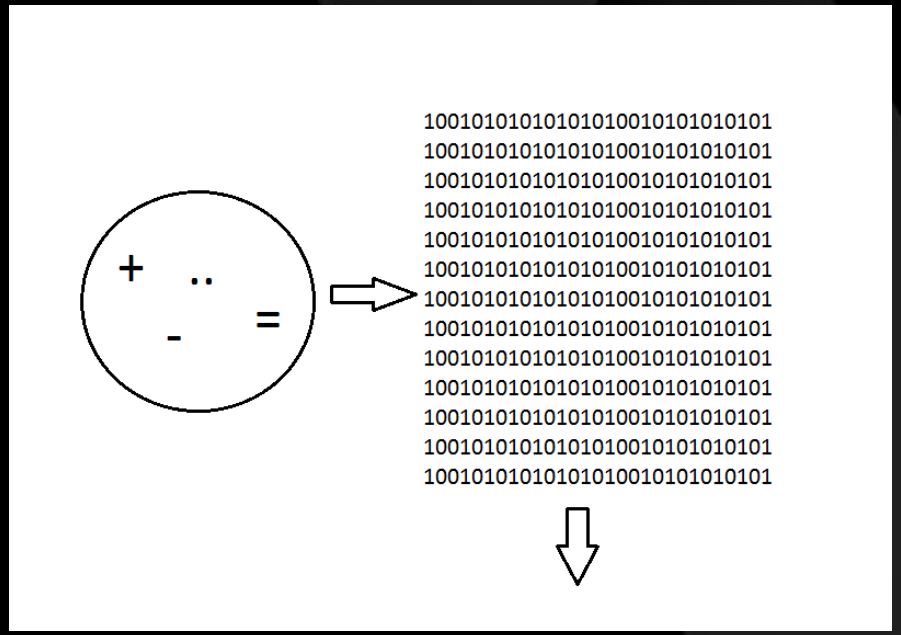
- *(Conditional) jump operation.* Code 10 shows the ProFTPD program logic to read the next command from an input buffer. `pbuf->current` is a pointer to the next command in the input, thus forming a virtual PC for the attacker's MINDOP program. By corrupting `pbuf->current`, the attacker can select a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

Kernel Io

- In the end data are all what matters!
- Code is just group of instructions working over those data

- Your ultimate goal is to access data
- No need to be SYSTEM
- No need to do Kernel Exec
  - Kernel io can ‘emulate’ apis
  - You can do math operations
  - …



Data attack prevention

https://taesoo.gtisc.gatech.edu/pubs/2016/song:kenali.pdf

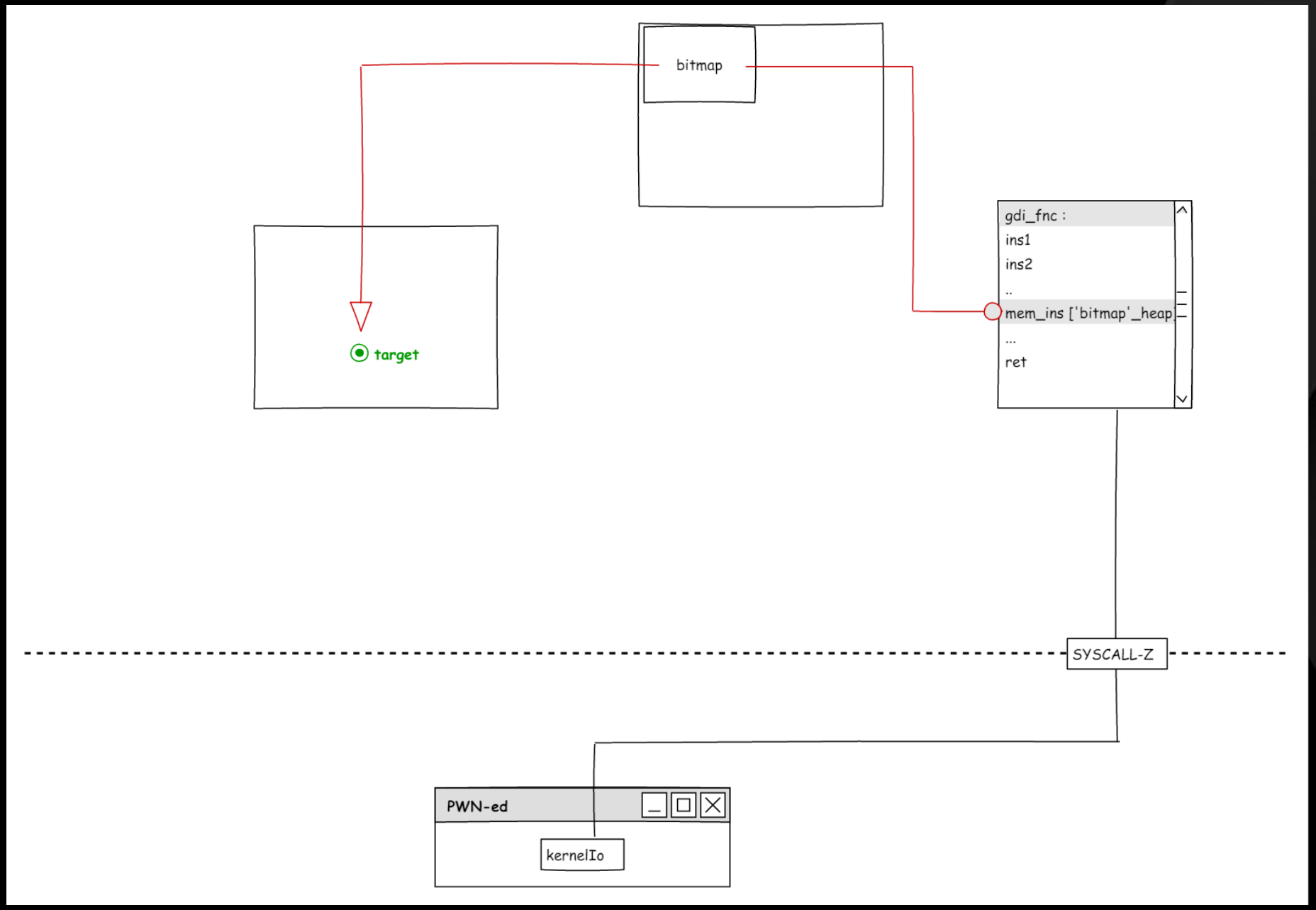
- Same piece of code should work on same memory type
- w32k gdi function will not touch _eprocess
- possible to simple set ranges (hardware) ?
- Pool isolation
- Isolated heap (object caches) ?
- local vars (thread) by default allowed

- Existing solutions ?
  - Intel mpx – seems not going to succeed ?
  - PAC – looks like game changer (CPI approach) ?
  - Isolation ? ☺

```
kernel_fnc_working_over_mem(
    __inout type_t* mem
    )
{
    ...
//setting mem access limits
    SETMB typeinfo<type_t>.base
    SETML typeinfo<type_t>.limit
//access memory
    mov [mem], reg
    mov reg, [mem]
    ...
}
```

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

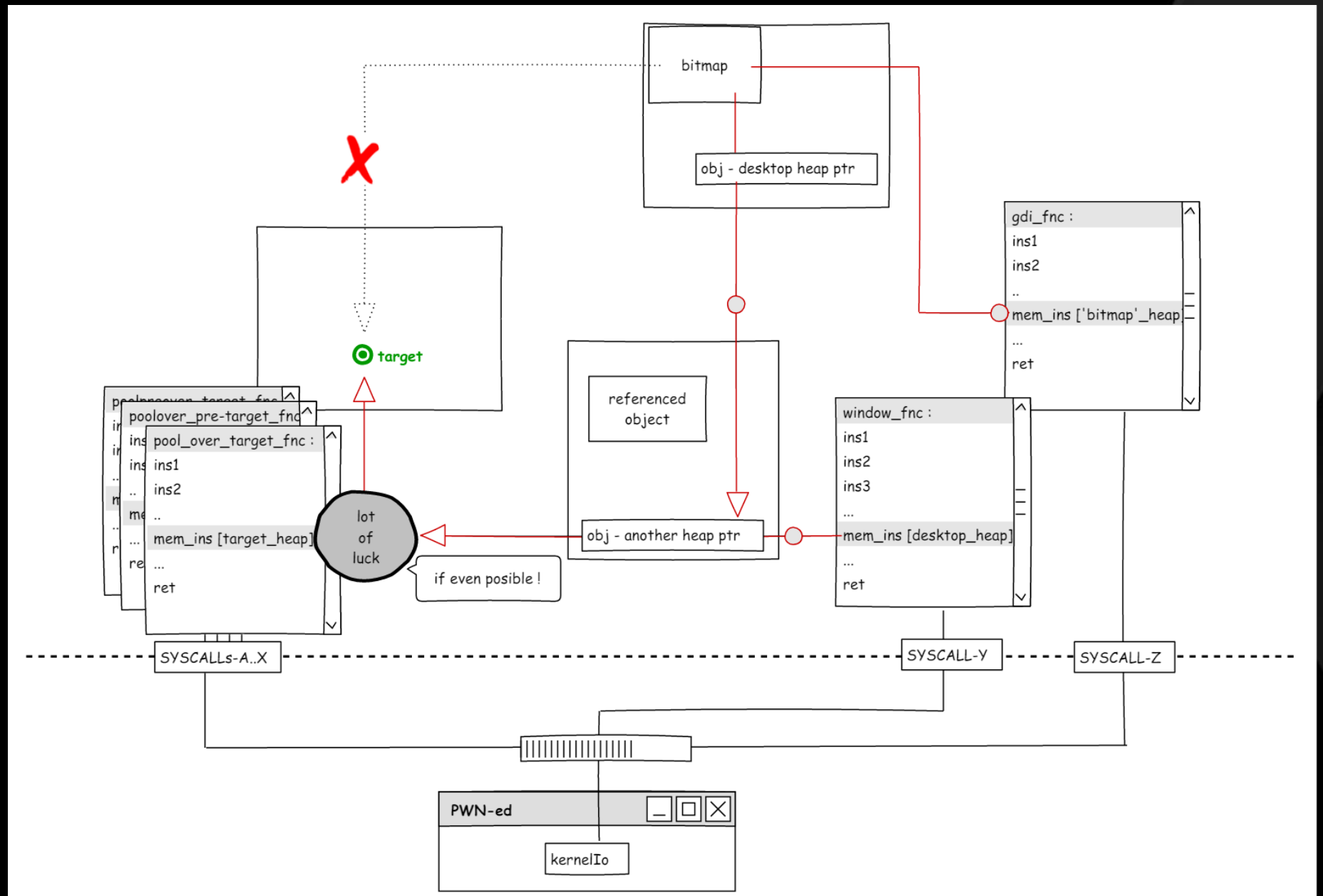
https://community.arm.com/groups/processors/blog/2016/10/27/armv8-a-architecture-2016-additions



Kernel Io vs cross heap checks

Tencent

KEEN security lab



Kernel Io vs cross heap checks

Tencent

KEEN security lab

Application Guard!

- well, this can change things lot!
- still kernel-Io is here real power ?
  - … well …
  - doomed within domain
- One does not simple do exec when ring -1 is watching!
  - no real RWE page [ept]
  - code signing [rw -> e]

Containers

- But, as always .. it is just another layer, another ring ...
  - bugs
  - logical issues
- with small big difference
  - less code
  - code quality ++

http://www.alex-ionescu.com/syscan2015.pdf

conclusions

- KASLR, nullptr-deref, Nx, SMEP
  - very well adapted!
- ACL for kernel object access
  - good security model!
- w32k re-designing { filtering & lockdown }
  - tackling biggest security problem in windows kernel!
- {C/R}FG
  - nice anti-exploitation approaches! { - step by step to adapt to kernel }
- CET - IBT & Shadow Stack
  - nice co-operation towards to new security model!
- HyperV based App Containers
  - moving operating system to another league in terms of security

Tencent

KEEN security lab

conclusions - todo#memory-corruptions

- Control flow integrity – to improve
  - fine grained CFG – fast, smart, compat! but can it be fine enough ?
  - adapt { forwarded-edge (clang) / type-hash (pax) } – existing, fast, does the job!
  - Make code reuse attacks to disappear ?
- Data attacks break down – to tackle
  - Memory access boundary checks
  - pointer load / function to heap memory access
  - Make kernel io disappear ?

- Will it move memory corruptions from attack vector to state of art ?
  - memory corruption == ddos only ?
  - Not yet, long way to go, but now we can see that possibility
  - However maybe that can even happen in foreseeable future ?

Tencent

Thank you!
Q & A

科恩实验室
KEEN
security lab

KEEN
security
lab

Tencent

KEEN
security
lab