SECURE BY DESIGN SERIES

Linux Kernel Security: Overview of Security Features and Hardening

Nathan Barrett-Morrison
Senior Embedded Systems Engineer
Timesys Corporation

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

What is a Kernel Configuration?

Security-related Configuration Settings

- Reasons for Modifying

Classes of Kernel Security

Cost-Benefit Analysis

Navigating The Complexity

Other Kernel Security Considerations

System Security Considerations

What is a Kernel Configuration?

Quick Overview

Kernel Configurations Overview

What is a kernel configuration?

- Commonly referred to as the default configuration or defconfig
- List of all the configuration options which can be enabled or disabled inside Linux kernel

Why is it relevant to security hardening?

- Most configuration options have security implications
- Example: i.MX 8 kernel config file currently has 8673 lines (Linux Kernel 5.4.3_2.0.0)
- Example: i.MX 8 kernel defconfig file currently has 1056 lines (Linux Kernel 5.4.3_2.0.0)
- Enormous amount of configuration options to consider

What does a kernel configuration look like?

```
#
# Automatically generated file; DO NOT EDIT.
# Linux/arm64 5.4.3 Kernel Configuration
#

#
# Compiler: aarch64-poky-linux-gcc (GCC) 8.3.0
#
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80300
CONFIG_CLANG_VERSION=0
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_ASM_INLINE=y
CONFIG_CC_HAS_WARN_MAYBE_UNINITIALIZED=y
```

```
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y

#
# General setup
#
CONFIG_INIT_ENV_ARG_LIMIT=32
# CONFIG_COMPILE_TEST is not set
# CONFIG_HEADER_TEST is not set
CONFIG_LOCALVERSION="-imx_5.4.3_2.0.0"
CONFIG_LOCALVERSION_AUTO=y
CONFIG_BUILD_SALT=""
CONFIG_DEFAULT_HOSTNAME="(none)"
CONFIG_SWAP=y
```

```
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SYSCTL=y
CONFIG_POSIX_MQUEUE=y
CONFIG_POSIX_MQUEUE_SYSCTL=y
CONFIG_CROSS_MEMORY_ATTACH=y
# CONFIG_USELIB is not set
CONFIG_AUDIT=y
CONFIG_HAVE_ARCH_AUDITSYSCALL=y
CONFIG_AUDITSYSCALL=y
#
# IRQ subsystem
#
CONFIG_GENERIC_IRQ_PROBE=y
CONFIG_GENERIC_IRQ_SHOW=y
CONFIG_GENERIC_IRQ_SHOW_LEVEL=y
```

Kernel Configurations Overview

What does a kernel default configuration look like (defconfig)? Minimal subset of the config

```
CONFIG_LOCALVERSION="-imx_5.4.3_2.0.0"
CONFIG_SYSVIPC=y
CONFIG_POSIX_MQUEUE=y
CONFIG_AUDIT=y
CONFIG_NO_HZ_IDLE=y
CONFIG_HIGH_RES_TIMERS=y
CONFIG_PREEMPT=y
CONFIG_IRQ_TIME_ACCOUNTING=y
CONFIG_BSD_PROCESS_ACCT=y
CONFIG_BSD_PROCESS_ACCT_V3=y
CONFIG_TASKSTATS=y
CONFIG_TASK_DELAY_ACCT=y
CONFIG_TASK_XACCT=y
CONFIG_TASK_IO_ACCOUNTING=y
CONFIG_IKCONFIG=y
```

```
CONFIG_IKCONFIG_PROC=y
CONFIG_NUMA_BALANCING=y
CONFIG_MEMCG=y
CONFIG_MEMCG_SWAP=y
CONFIG_BLK_CGROUP=y
CONFIG_CGROUP_PIDS=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_CPUSETS=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_CGROUP_PERF=y
CONFIG_NAMESPACES=y
CONFIG_USER_NS=y
CONFIG_SCHED_AUTOGROUP=y
CONFIG_RELAY=y
```

```
CONFIG_BLK_DEV_INITRD=y
CONFIG_EXPERT=y
CONFIG_KALLSYMS_ALL=y
# CONFIG_COMPAT_BRK is not set
CONFIG_PROFILING=y
CONFIG_JUMP_LABEL=y
CONFIG_MODULES=y
CONFIG_MODULE_UNLOAD=y
# CONFIG_IOSCHED_DEADLINE is not set
CONFIG_ARCH_SUNXI=y
CONFIG_ARCH_ALPINE=y
CONFIG_ARCH_BCM2835=y
CONFIG_ARCH_BCM_IPROC=y
CONFIG_ARCH_BERLIN=y
CONFIG_ARCH_BRCMSTB=y
```

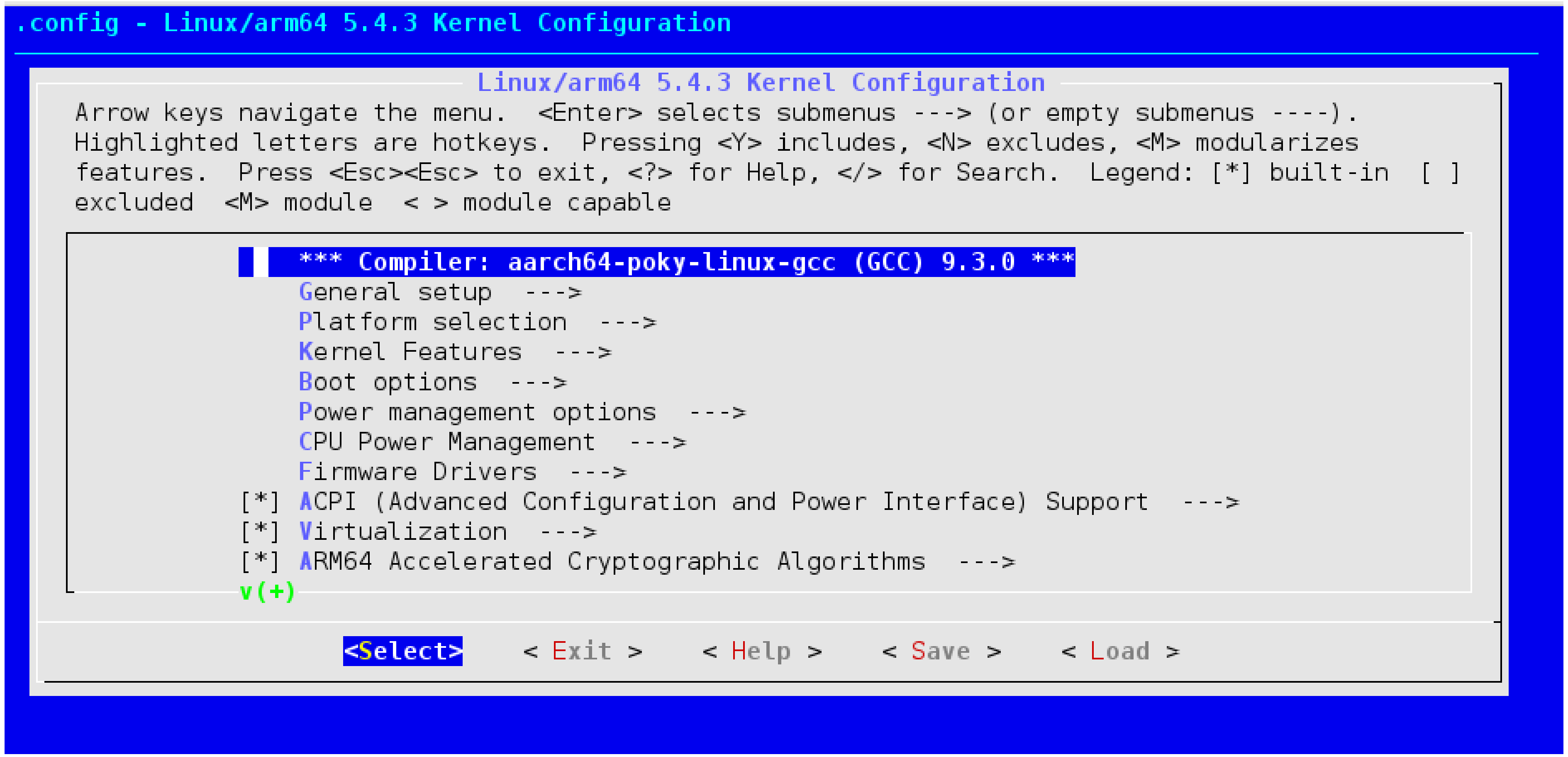
Kernel Configurations Overview

How do you understand all of these options?

- Menuconfig
  - Yocto: bitbake -c menuconfig virtual/kernel
- Linux Kernel Driver Database
  - https://cateee.net/lkddb/
- Bootlin / GitHub / CodeAurora if you need to delve into source
  - https://elixir.bootlin.com/linux/latest/source
  - https://github.com/torvalds/linux
  - https://source.codeaurora.org/external/imx/linux-imx/

Kernel Configurations Overview

Menuconfig:



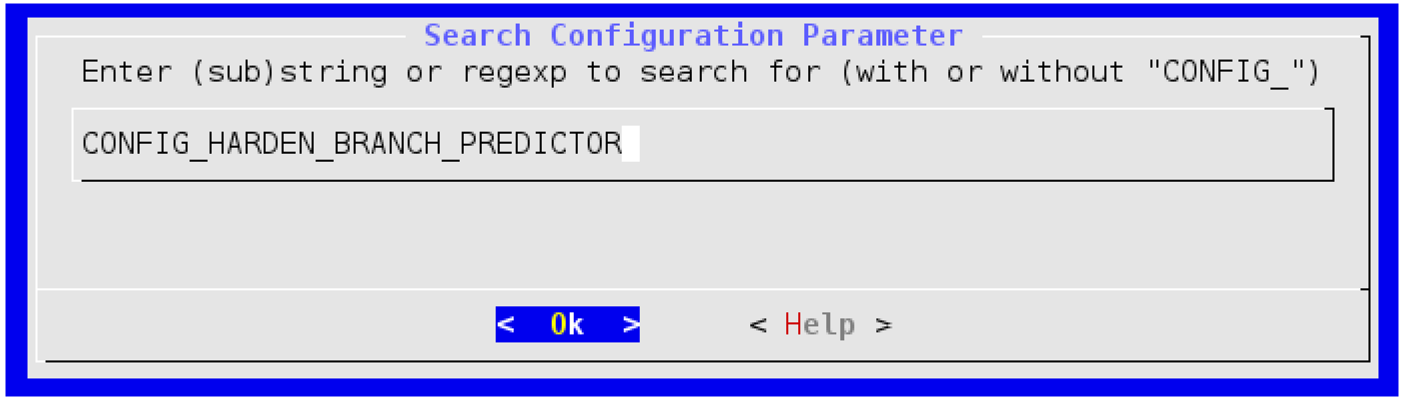
```
.config - Linux/arm64 5.4.3 Kernel Configuration

                    Linux/arm64 5.4.3 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

            *** Compiler: aarch64-poky-linux-gcc (GCC) 9.3.0 ***
            General setup  --->
            Platform selection  --->
            Kernel Features  --->
            Boot options  --->
            Power management options  --->
            CPU Power Management  --->
            Firmware Drivers  --->
        [*] ACPI (Advanced Configuration and Power Interface) Support  --->
        [*] Virtualization  --->
        [*] ARM64 Accelerated Cryptographic Algorithms  --->
        v(+)

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

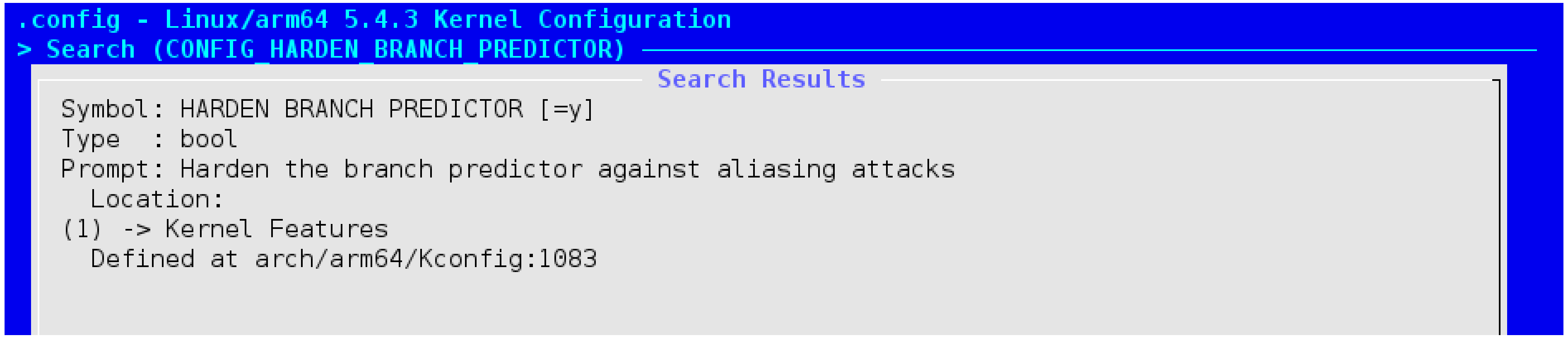
Menuconfig: use forward slash to search for a term



```
Search Configuration Parameter
Enter (sub)string or regexp to search for (with or without "CONFIG_")

CONFIG_HARDEN_BRANCH_PREDICTOR

            < Ok >          < Help >
```



```
.config - Linux/arm64 5.4.3 Kernel Configuration
> Search (CONFIG_HARDEN_BRANCH_PREDICTOR)
                          Search Results
Symbol: HARDEN_BRANCH_PREDICTOR [=y]
Type  : bool
Prompt: Harden the branch predictor against aliasing attacks
  Location:
(1) -> Kernel Features
  Defined at arch/arm64/Kconfig:1083
```

Security-related Configuration Settings:
Reasons for Modifying a Setting

Security-related Configuration Settings

- When hardening the kernel, there are generally four categories of reason for which a configuration item may be enabled or disabled:
  1) Enable configurations that protect against known exploits
  2) Disable configurations that are known to be exploitable
  3) Enable security features that make it harder to hack
  4) Reduce attack surface by disabling unused configurations

Security-related Configuration Settings

- 1) Adding an additional level of protection against a known exploit by enabling a configuration item.

  For example:
  - Enabling CONFIG_HARDEN_BRANCH_PREDICTOR to protect against Spectre-related speculation attacks
    - Speculation attacks against some high performance processors rely on being able to manipulate the branch predictor for a victim context by executing aliasing branches in the attacker context. Such attacks can be partially mitigated against by clearing the internal branch predictor state and limiting the prediction logic in some situations.

```
$ cat /proc/cpuinfo | grep bugs
bugs                                    : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
```

Security-related Configuration Settings

- 2) Disabling a configuration item or subsystem which is known to be exploitable.

  For example:
  - Disabling the USB networking subsystem (USB_USBNET=is not set), so that applications using network-based IPC(InterProcessor Communication) mechanisms may not be inadvertently exposed through a USB port.
  - Disabling /dev/mem access to physical memory (DEVMEM=is not set), so that physical memory cannot be easily modified.

Security-related Configuration Settings

- 3) Enabling general security strengthening features in the kernel (which may not necessarily protect against a presently known attack).
  - Reducing the risk of memory page leakage by enabling page poisoning (PAGE_POISONING=Y) to overwrite any potentially sensitive information upon freeing.
  - Enabling Security-Enhanced Linux

Security-related Configuration Settings

- 4) Disabling any unused kernel configuration options. If you don't need it, disable it.
  - This reduces the potential attack surface. If an option is not enabled, then it cannot be exploited!
  - Example
    - Disable Serial / UART console driver if not used in production
    - Disable eFuse driver
    - Disable ethernet driver if not used in production

Classes of Kernel Security

Quick Overview

Classes of Kernel Security

- Further breaking down the reasons for setting a configuration item:
    - 1) Memory Protections
        - Stack canaries
        - Usercopy Protections
    - 2) GCC plugins
    - 3) Module Loading Security
    - 4) Security Policy Management
    - 5) Attack Surface Reduction
    - 6) File System Hardening

Classes of Kernel Security

- Memory Protections
  - Items which protect against memory exploitation
    - CONFIG_BUG=y
    - CONFIG_PAGE_POISONING=y
    - CONFIG_INIT_STACK_ALL=Y
  - Stack Canaries
    - Canary in a coal mine…
    - Warn of an unexpected stack change by checking for bad canary value
      - CONFIG_STACKPROTECTOR=y
  - Usercopy Protections
    - CONFIG_HARDENED_USERCOPY=Y
  - Kernel Address Space Layout Randomization (KASLR)
    - CONFIG_RANDOMIZE_MEMORY=Y
  - Memory Information exposure
    - PROC_VMCORE=is not set
  - Heap Overflow
    - COMPAT_BRK=is not set

Classes of Kernel Security

- GCC plugins
  - Actions which GCC can perform at compile time to help protect against exploitation
  - Example:
    - CONFIG_GCC_PLUGIN_LATENT_ENTROPY=Y
    - CONFIG_GCC_PLUGIN_RANDSTRUCT=Y
    - CONFIG_GCC_PLUGIN_STRUCTLEAK=y
    - CONFIG_GCC_PLUGIN_STRUCTLEAK_BYREF_ALL=Y
    - CONFIG_GCC_PLUGIN_STACKLEAK=Y
    - CONFIG_GCC_PLUGIN_ARM_SSP_PER_TASK=Y

Classes of Kernel Security

- Module Loading Security
  - These help protect loadable kernel modules (or disables them altogether)
  - Example:
    - CONFIG_MODULES=is not set
      - If you don’t need MODULES, disable them altogether. Otherwise:
    - CONFIG_STRICT_MODULE_RWX=y
    - CONFIG_MODULE_SIG_ALL=y
    - CONFIG_MODULE_SIG_SHA512=Y
    - CONFIG_MODULE_SIG_FORCE=y
    - CONFIG_DEBUG_SET_MODULE_RONX=Y

Classes of Kernel Security

- Security Policy Management
  - Additional policy management actions which can be used to improve security
  - Some of these are very detailed / need a separate presentation
    - CONFIG_SECURITY_SELINUX=y
      - Security Enhanced Linux
  - Example:
    - CONFIG_SECURITY_YAMA=y
    - CONFIG_SECURITY_SELINUX_DISABLE=y
    - SECURITY_LOCKDOWN_LSM=Y
    - SECURITY_SAFESETID=Y
    - SECURITY_LOADPIN=Y

Classes of Kernel Security

- Attack Surface Reduction
  - General options which can be set to minimize the amount of attack vectors available
  - Example:
    - CONFIG_PANIC_ON_OOPS=is not set
    - CONFIG_PANIC_TIMEOUT=-1
    - DEVMEM=is not set
    - KEXEC=is not set
    - DEVKMEM=is not set
    - TRACING=is not set
    - FTRACE=is not set
    - DEBUG_FS=is not set
    - STAGING=is not set
    - KALLSYMS=is not set

Classes of Kernel Security

- File System Protections
  - File system encryption and/or verification
    - DM_VERITY=Y
      - read only
    - DM_CRYPT=Y
      - encryption
    - DM_INTEGRITY=Y
      - integrity checking which still permits writable file system

Cost-Benefit Analysis

SECURE CONNECTIONS
FOR A SMARTER WORLD

Cost-Benefit Analysis

- Changing configuration options is not without impact:
    - Compile Time
    - Kernel Binary Size
    - Boot Time
    - Processor Performance
- Up to you to decide which options are warranted for your system

Cost-Benefit Analysis

- Compilation Time
  - This is especially true for security features which are checked by the compiler during compile time.
  - For example, the GCC_PLUGIN options:
    - CONFIG_GCC_PLUGIN_STACKLEAK=Y
  - Added compilation time is well worth the added security.

Cost-Benefit Analysis

- Kernel Binary Size
  - This is usually not a concern. Adding and removing features may change the kernel size by a few megabytes, which is generally negligible on most modern systems
  - If you have extremely small storage requirements, then disabling features for space can help.

Cost-Benefit Analysis

- Boot time
    - For example:
        - DM_CRYPT=y
            - Adds drive encryption capabilities to your kernel
    - The more drivers that are enabled, the longer the kernel will take to load
    - Most drivers add a negligible amount of time by themselves (~10-100 milliseconds)
        - Cumulatively adds up…
        - More important if targeting <5-10 second total boot times

Cost-Benefit Analysis

- Processor Performance
  - This is perhaps the most concerning option and must be determined by trial and error.
  - For example:
    - Erasing memory with PAGE_POISONING_ZERO=Y will use CPU cycles.
    - Disabling/hardening speculative branching will degrade CPU performance
      - CONFIG_RETPOLINE (Intel)
      - HARDEN_BRANCH_PREDICTOR=Y (Arm)
    - INIT_ON_FREE_DEFAULT_ON=Y
    - REFCOUNT_FULL=Y
    - SLAB_FREELIST_HARDENED=Y

Navigating The Complexity

SECURE CONNECTIONS
FOR A SMARTER WORLD

Navigating the Complexity

- Analyzing, understanding, and modifying the kernel configuration with these tasks in mind is not trivial.
  - Plan accordingly! Security conscious company must allocate time or resources
- Many companies have two separate configurations (development and production)
- Maintenance burden once you have created your final configuration.
  - Kernel Updates / Patches→new configuration items
    - Consider: At the time you release your product the kernel may be secure. Within a few years, there may be multiple released zero-day exploits for that kernel version.
    - Security is not once and done

Navigating the Complexity

- NXP's Community Forum
  - https://community.nxp.com/t5/OSS-Security-Maintenance/bd-p/ossecmaib
- Are there any solutions which handle these configuration items?
  - Kernel Self Protection Project
    - https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
  - CLIP OS
    - https://clip-os.org/en/

Navigating the Complexity

- Timesys has a tool which will scan your kernel configuration and produce a list of recommendations

Navigating the Complexity

- Commands which were run in the previous video:

#Build basic imx8mq EVK yocto build with a 5.4.3 kernel

```
$ repo init - u https://source.codeaurora.org/external/imx/imx - manifest - b imx - linux - zeus - m imx - 5.4.3 -
2.0.0.xml

$ repo sync

$ DISTRO=fsl - imx - wayland MACHINE=imx8mqevk source imx - setup - release.sh - b build

$ bitbake core - image - minimal
```

#Add Timesys meta layer to bblayers.conf

```
$ cd ../sources/

$ git clone git@src.timesys.com:timesys/features/ meta - timesys - vigishield.git

$ cd ../build/

$ nano conf/bblayers.conf
```

Navigating the Complexity

#Add layers to bblayer.conf

- > Add:

${BSPDIR}/sources/meta-timesys-vigishield

${BSPDIR}/sources/meta-timesys

#Inherit timesys-kernel-check

```
$ nano conf/local.conf
 -> Add:
INHERIT += "timesys-kernel-check"
```

#Rebuild

```
$ bitbake core-image-minimal
```

#View report

```
$ libreoffice timesys_kernel_hardening_results.csv
```

Timesys VigiShield

SECURE CONNECTIONS
FOR A SMARTER WORLD

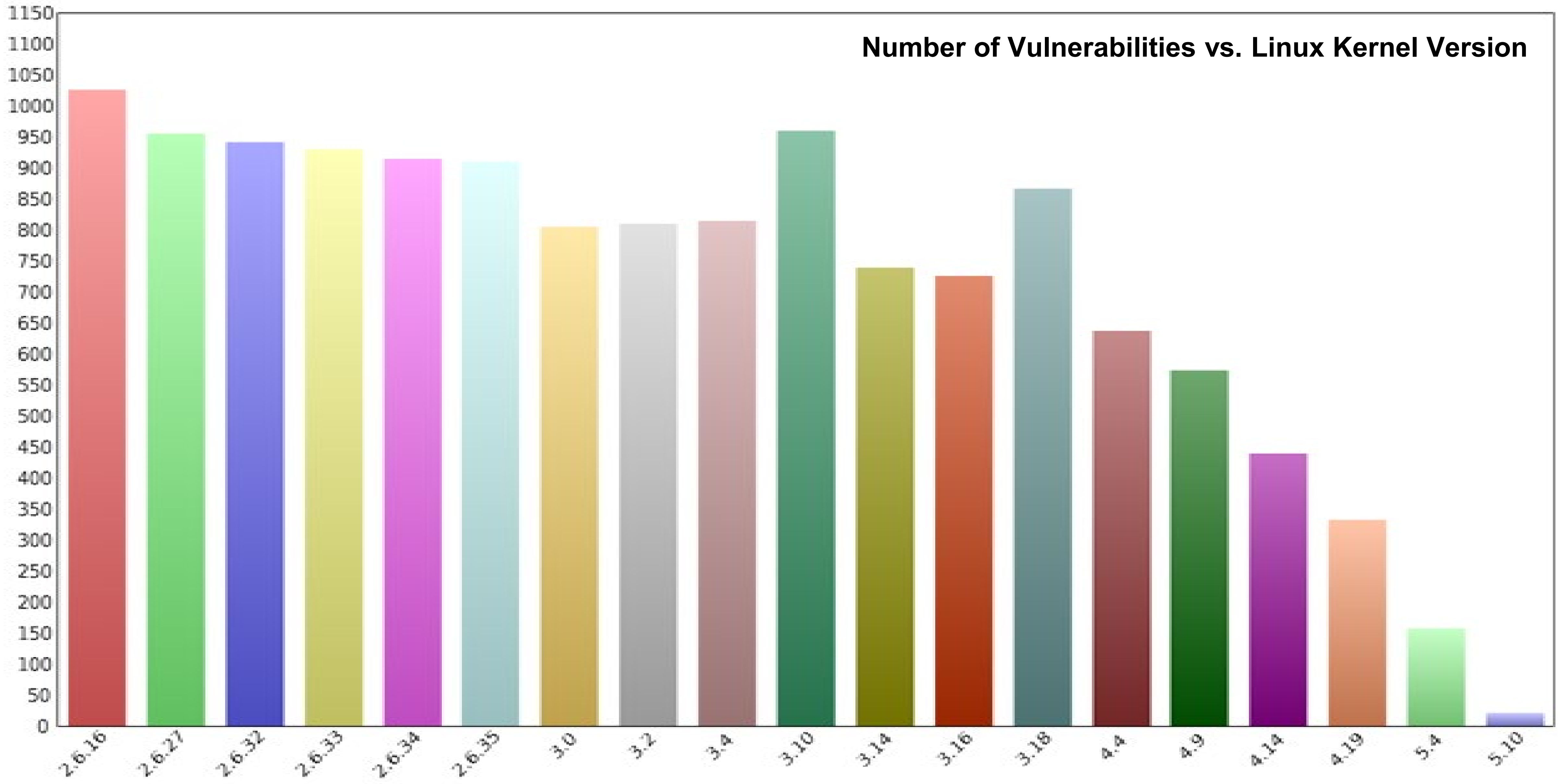
Timesys VigiShield

- Security Hardening Layer for i.MX processors
    - Secure Boot
        - High Assurance Boot (i.MX 6, i.MX 7, i.MX 8M Families) / Advanced High Assurance Boot (i.MX 8/8X Families)
    - Root File System Encryption
    - Root File System Verification
    - UUU Helpers (Universal Update Utility)
        - JTAG disablement
        - Secure fuse programming
    - Kernel Configuration Checking
    - Offline Firmware Update (encrypted and signed)

Other Kernel Security Considerations

Other Kernel Security Considerations

- Kernel Versions
  - Older kernel = More unpatched Common Vulnerabilities and Exposures (CVEs)
    - https://cve.mitre.org/index.html
  - Keep your kernel updated to the latest LTS kernel if possible
    - NXP releases are based on LTS kernels
- Check your included open source software packages for Common Vulnerabilities and Exploitations (CVEs) on the national vulnerability database
- A typical embedded system has hundreds or thousands of these packages which must be checked. There are tools which can help!
  - Vigiles Vulnerability Management
    - Can filter for vulnerabilities based on your kernel configuration as well
  - Timesys BSP maintenance team can help monitor and maintain security for you


Number of Vulnerabilities vs. Linux Kernel Version
1150
1100
1050
1000
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
2.6.16
2.6.27
2.6.32
2.6.33
2.6.34
2.6.35
3.0
3.2
3.4
3.10
3.14
3.16
3.18
4.4
4.9
4.14
4.19
5.4
5.10

Other Security Considerations

Other Kernel Security Considerations

- Commands which were run in the previous video:

Set up configuration:

```
$ nano conf/local.conf
```

Add:

```
    #Inherit Timesys Vigiles CVE checker
    INHERIT += "vigiles"
```

Build:

```
$ bitbake core        - image - minimal
```

Other Kernel Security Considerations

- System Control Settings (sysctl)
  - Kernel runtime configuration parameters
    - Example:
      - kernel.dmesg_restrict
- Discretionary Access Control
  - Check your file system permissions and controls
    - Minimize uses of SUID and GUID where possible

Other Kernel Security Considerations

- Commands which were run in the previous video:

Set up configuration:

```
$ nano conf/local.conf
```

Add:

```
    #Inherit Timesys DAC report generator
    INHERIT += "timesys    - discretionary    - access  - report"
```

Build:

```
$ bitbake core    - image - minimal
```

View results:

```
$ libreoffice /mnt/HDD2/Projects/NXP        - Webinar/build/timesys_dac.csv
```

Other Kernel Security Considerations

- Mandatory Access Control
  - Security Enhanced Linux (SELinux)
  - NSA’s answer to discretionary access controls (DAC) shortcomings
  - Adds more privilege control than typical DAC
    - User-customizable security policy on running processes and their actions
    - Operation permissions are checked after DAC

Other Kernel Security Considerations

- A very comprehensive list of exploits that have been performed in the past can be found here:
  - https://github.com/xairy/linux-kernel-exploitation
  - Also includes some links for defenses and other items
  - Tons to look at!

Takeaways

- There are many security related options configuration options in the Linux kernel
- You must tailor them to fit your requirements
- Make security requirements part of your product requirements from day 1
- Consider using Timesys VigiShield
  - Ready-to-use Yocto that provides security out of the box for i.MX processors
- If needed, leverage assistance of experienced security development teams from NXP and Timesys:
  - Product security design
  - Configuration and implementation of needed security features
  - Additional security documentation
  - Security verification
  - Compliance alignment
- Start with initial non-binding conversation

GOLD PARTNER

Upcoming Webinars

SECURE CONNECTIONS
FOR A SMARTER WORLD

In-depth Dive

- **Secure Software Updates:** Designing OTA Updates for secure embedded Linux systems

- **Linux System Hardening:** Securing your embedded device from the risk of being compromised

Previous Webinars

SECURE CONNECTIONS
FOR A SMARTER WORLD

Previous Webinars

Secure By Design Series

- Trusted Execution Environments: Getting Started with OP-TEE on i.MX Processors
- Software integrity and data confidentiality: Establishing secure boot and chain of trust on i.MX processors
- Securing Embedded Linux Devices: Pitfalls to Avoid

Stay Secure (Vigiles) Series

- Best practices for triaging Common Vulnerabilities & Exposures (CVEs) in embedded systems
- Full Life Cycle Security Maintenance of Embedded Linux BSPs
- BSP security maintenance: Best practices for vulnerability monitoring & remediation
- Software Security Management: Cutting through the vulnerability storm with NXP Vigiles

For More Information and to Become More Secure

Timesys is an embedded Linux security expert and NXP Gold Partner.
To discuss your project, please contact us at sales@timesys.com

Use this link to go to [Services for securing your device]()

Thank You!

Q&A

SECURE CONNECTIONS
FOR A SMARTER WORLD