Diving Into Linux Kernel Security

Alexander Popov

positive technologies

December 14-15, 2024

Teaser

- Operating system security seems like a very complex topic

Teaser

- Operating system security seems like a very complex topic
- A lot of research articles are published about it

Teaser

- Operating system security seems like a very complex topic
- A lot of research articles are published about it
- You'd like to delve into it but:
  - You don't know where to start
  - You are afraid to get lost in huge amount of materials

Teaser

- Operating system security seems like a very complex topic
- A lot of research articles are published about it
- You'd like to delve into it but:
  - You don't know where to start
  - You are afraid to get lost in huge amount of materials
- You may also need to configure the security parameters of your Linux systems

Teaser

- Operating system security seems like a very complex topic
- A lot of research articles are published about it
- You'd like to delve into it but:
  - You don't know where to start
  - You are afraid to get lost in huge amount of materials
- You may also need to configure the security parameters of your Linux systems
- If so, then this talk is for you

Teaser

- Operating system security seems like a very complex topic
- A lot of research articles are published about it
- You'd like to delve into it but:
  - You don't know where to start
  - You are afraid to get lost in huge amount of materials

- You may also need to configure the security parameters of your Linux systems
- If so, then this talk is for you

Who Am I

- Alexander Popov
- Linux kernel developer since 2012
- Maintainer of some free software projects
- Principal security researcher and Head of Open Source Program Office at positive technologies

- Conference speaker:

  OffensiveCon, NullCon Goa, Linux Security Summit, Still Hacking Anyway, Zer0Con, HITB, Positive Hack Days, ZeroNights, HighLoad++, Open Source Summit, OS Day, Linux Plumbers

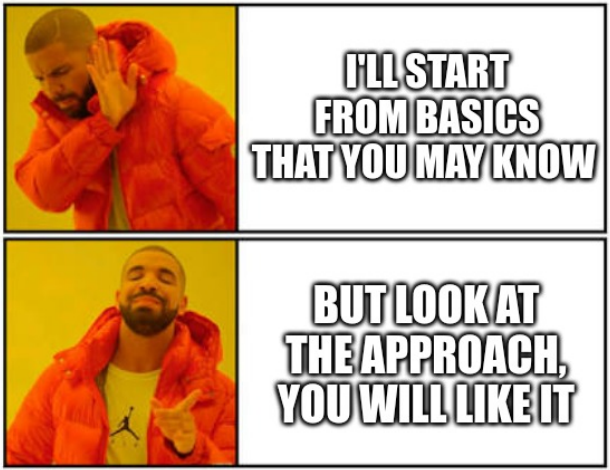
  a13xp0p0v.github.io/conference_talks

Agenda

1. Basic terminology
2. Approach to learning operating system security area
3. Overview of Linux kernel security
4. The tool for checking Linux security parameters
5. **Goal**: to interest you
6. **Bonus**: invite you to join open source projects

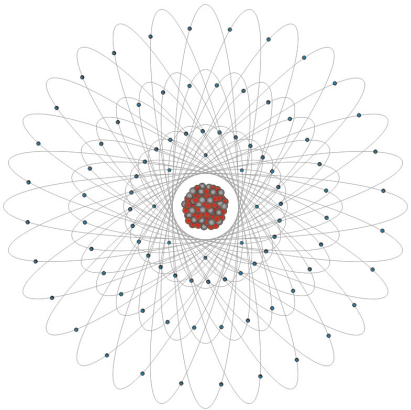
   (this talk is about my personal OSS projects)

Disclaimer


I'LL START
FROM BASICS
THAT YOU MAY KNOW
BUT LOOK AT
THE APPROACH,
YOU WILL LIKE IT
imgflip.com

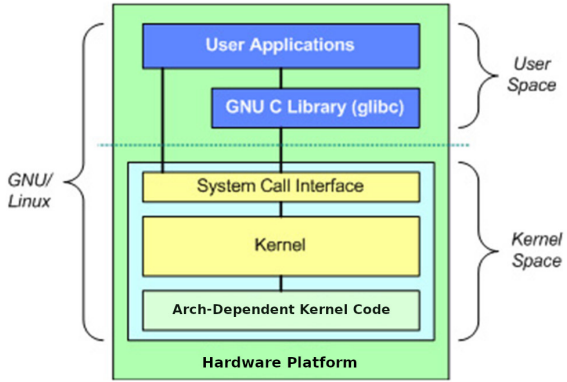
Set Up the Terminology: OS

- An **operating system** is software that
  1. Manages HW and SW resources
  2. Provides services for computer programs
- An **OS kernel** is a part of the OS that runs in the privileged CPU mode
- The kernel manages processes and their usage of
  - CPU time
  - Memory and storage
  - I/O devices
  - IPC



pediaa.com/difference-between-uranium-and-thorium

OS Architecture (GNU/Linux)



ibm.com/developerworks/linux/library/l-linux-kernel/

Set Up the Terminology: Security Model

- To build security, you first need a **threat model**
- Without a threat model, security measures are just a bunch of tricks for
  1. Weakening system performance
  2. Making users mad
- A threat model is required to create a **security model**

A security model describes

how security measures work together
to mitigate the relevant threats

Set Up the Terminology: OS Security Model

- An OS threat model may include
  1. Parsing and handling untrusted data
  2. Executing untrusted applications (userspace code)
  3. Communicating via untrusted networks

Yefim Deshalyt: The heroic defense of Old Ryazan

Set Up the Terminology: OS Security Model

- An OS threat model may include
  1. Parsing and handling untrusted data
  2. Executing untrusted applications (userspace code)
  3. Communicating via untrusted networks
- These attack vectors against OS may be used for
  1. Remote code execution (RCE)
  2. Local privilege escalation (LPE)
  3. Denial of service (DoS)
  4. Data leaks

Yefim Deshalyt: The heroic defense of Old Ryazan

Set Up the Terminology: OS Security Model

- An OS threat model may include
  1. Parsing and handling untrusted data
  2. Executing untrusted applications (userspace code)
  3. Communicating via untrusted networks
- These attack vectors against OS may be used for
  1. Remote code execution (RCE)
  2. Local privilege escalation (LPE)
  3. Denial of service (DoS)
  4. Data leaks

Yefim Deshalyt: The heroic defense of Old Ryazan

- An OS security model defines how OS security features mitigate these threats
- Excellent example: Android Platform Security Model

Main Term: Attack Surface

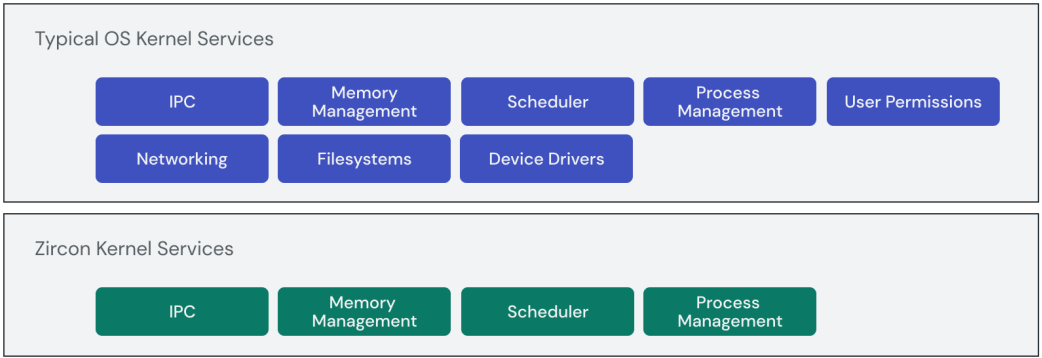
The **attack surface** covers all the system interfaces that an attacker can interact with

Main Term: Attack Surface

The **attack surface** covers all the system interfaces that an attacker can interact with

A bug reachable via the attack surface is a **vulnerability**

Comparing Linux Kernel and Zircon Microkernel



fuchsia.dev/fuchsia-src/get-started/learn/intro/zircon?hl=en

Linux Kernel: Huge Codebase

- The Linux kernel is being developed at **incredible speed**
- Development statistics:
  - More than 26 million lines of code (v6.11)
  - A new major release every 9 or 10 weeks
  - Around 2000 developers contribute to the kernel every release
  - Around 14000 changesets are merged into the mainline on every release

Fixing Bugs in the Linux Kernel

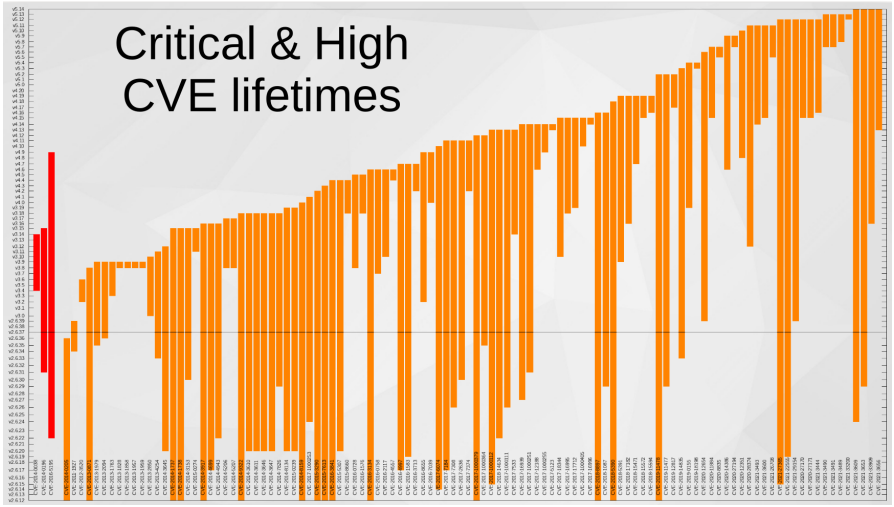
- We have a lot of bug detection tools:
  - Kernel sanitizers
  - Syzkaller kernel fuzzer
  - Various static analysis tools

Fixing Bugs in the Linux Kernel

- We have a lot of bug detection tools:
  - Kernel sanitizers
  - Syzkaller kernel fuzzer
  - Various static analysis tools

- However, kernel vulnerabilities appear faster than they are fixed
- Proof: A tale of a thousand kernel bugs by Dmitry Vyukov
- Moreover, kernel vulnerabilities have long lifetime

Upstream bug lifetime – 5,5 years (Kees Cook, 2021)



outflux.net/slides/2021/lss/

Another Approach

- To improve Linux kernel security, bug fixing alone is insufficient
- The OS kernel should handle errors safely (not giving a chance to attackers)
- **grsecurity and PaX** were the pioneers in this approach
- Their ideas inspired the **Kernel Self Protection Project (KSPP)**
- KSPP goal: kill whole bug classes and exploit methods in the mainline kernel
- KSPP overview by Kees Cook: outflux.net/slides/2021/lss/kspp.pdf

Linux Kernel Security

Linux kernel security is a very complex knowledge area.
Key concepts:

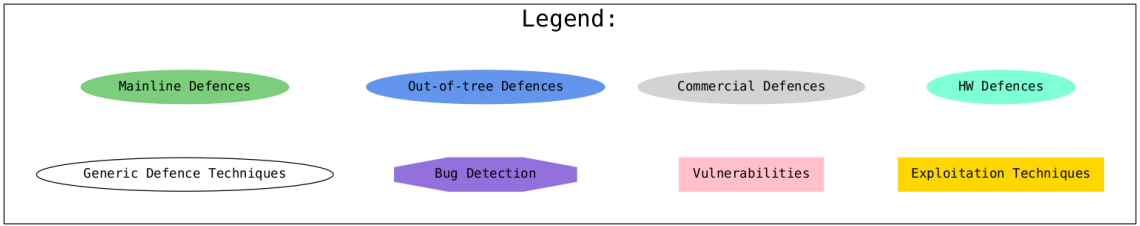
1. Vulnerability classes
2. Exploitation techniques
3. Bug detection mechanisms
4. Defence technologies
   - In the mainline
   - Shipped separately (commercial or under development)
   - Requiring special hardware features

All they have complicated relations with each other...
It would be nice to have a graphical representation of it.

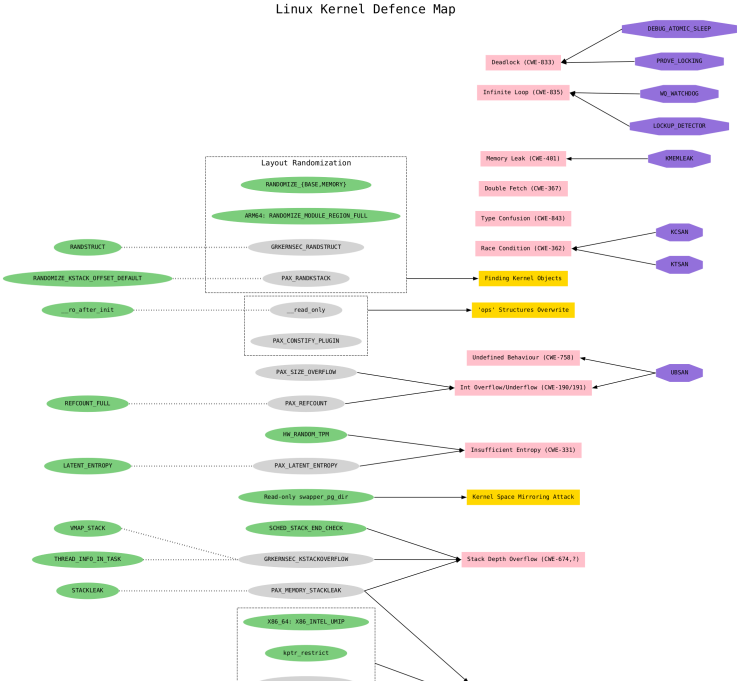
Drawn by Daniel Reeve, made by weta

Linux Kernel Defence Map

- So I developed the **Linux Kernel Defence Map**
  github.com/a13xp0p0v/linux-kernel-defence-map
- I started this project in 2018, and I'm continuing to improve and update it
- Map legend with key concepts:

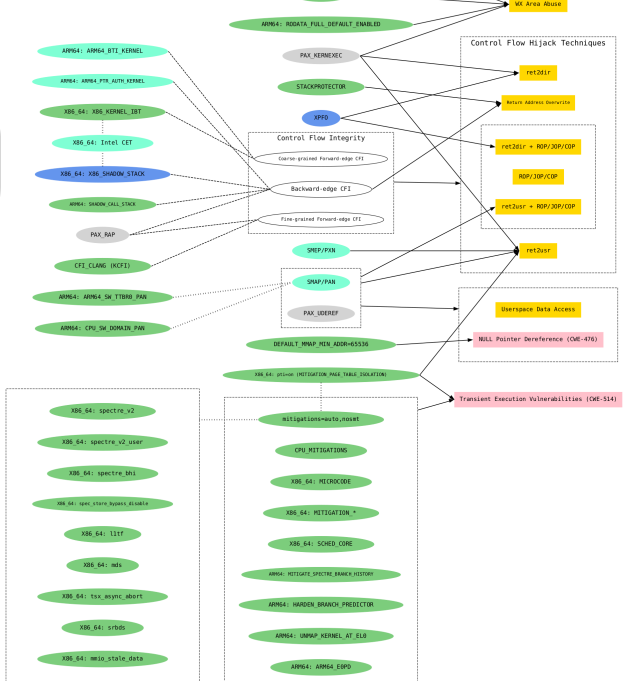


Linux Kernel Defence Map, whole picture (1/4)



Linux Kernel Defence Map, whole picture (2/4)

Linux Kernel Defence Map, whole picture (3/4)



Linux Kernel Defence Map, whole picture (4/4)

Linux Kernel Defence Map: How It Works

- Each connection between nodes represents some kind of relationship between them
- The node connections don't always mean "full mitigation"
- The map helps with navigating documentation and Linux kernel sources

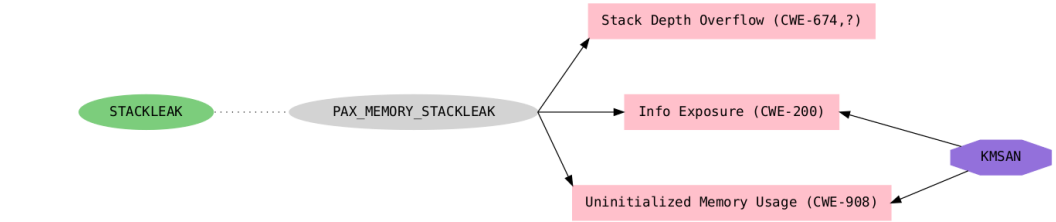
Linux Kernel Defence Map: How It Works

- Each connection between nodes represents some kind of relationship between them
- The node connections don't always mean "full mitigation"
- The map helps with navigating documentation and Linux kernel sources
- The map provides the Common Weakness Enumeration (CWE) numbers for vuln classes

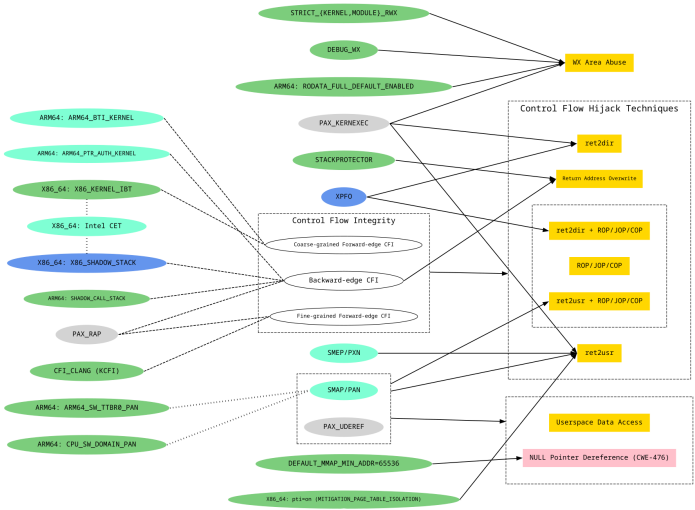
Linux Kernel Defence Map: How It Works

- Each connection between nodes represents some kind of relationship between them
- The node connections don't always mean "full mitigation"
- The map helps with navigating documentation and Linux kernel sources
- The map provides the Common Weakness Enumeration (CWE) numbers for vuln classes
- This map describes kernel security hardening
- [!] The map doesn't cover
  - Cutting the attack surface
  - Userspace security features
  - Security policies enforced by Linux Security Modules (LSM)

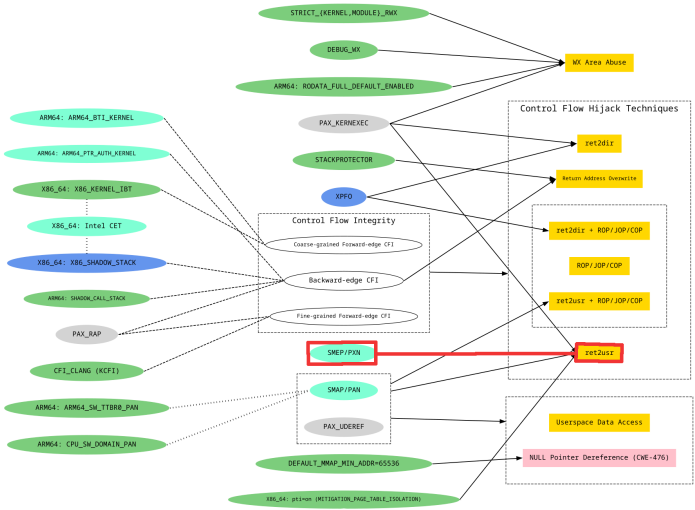
Example from the Map: STACKLEAK



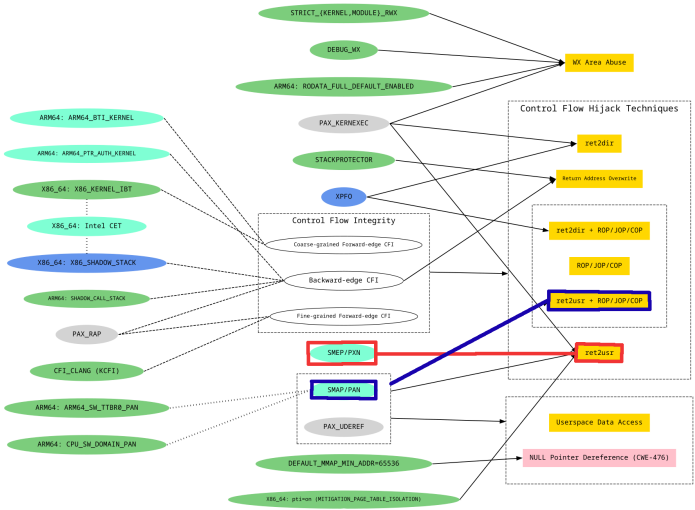
Example from the Map: Control-Flow Hijack



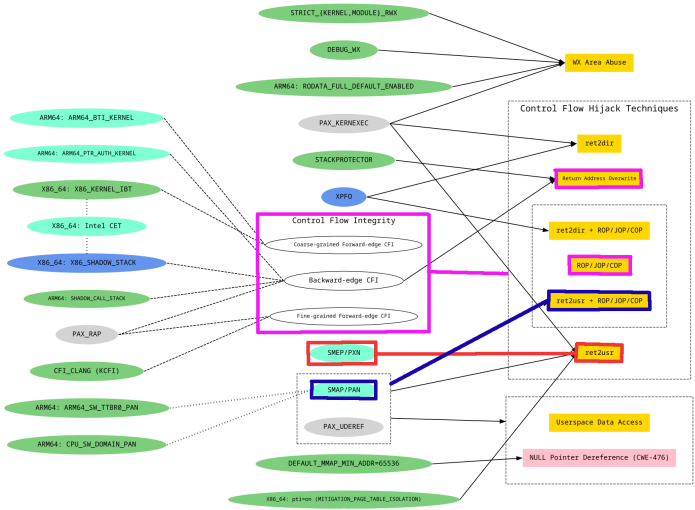
Example from the Map: Control-Flow Hijack



Example from the Map: Control-Flow Hijack

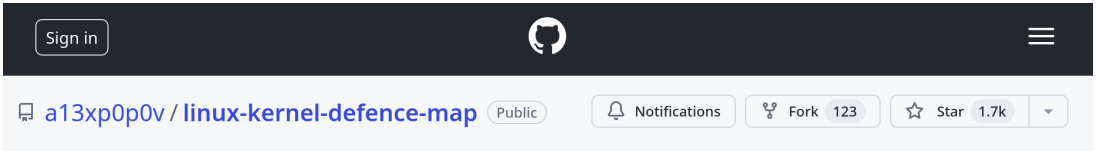


Example from the Map: Control-Flow Hijack



Linux Kernel Defence Map: Implementation

- The map needs to be updated (the Linux kernel is evolving)
- I **want** to develop it in text:
  1. To use Git
  2. To make it a free software project under GPL-3.0
- I **don't want** to place nodes and edges manually 🤯
- So I use the DOT language provided by Graphviz: `dot -Tsvg map.dot -o map.svg`
- The project is live and successful, you're welcome to join!



Linux Kernel Defence Map: Code Example

```
// Defences relations
edge [style=dotted, arrowhead=none, dir=none, headport=_, tailport=_];
"STACKLEAK":e -> "PAX_MEMORY_STACKLEAK":w;
// Defences vs Vulnerabilities and Exploitation Techniques
edge [style=solid, arrowhead=normal, dir=forward, headport=_, tailport=_];
"PAX_MEMORY_STACKLEAK":e -> "Stack Depth Overflow (CWE-674,?)":sw;
"PAX_MEMORY_STACKLEAK":e -> "Uninitialized Memory Usage (CWE-908)":nw;
"PAX_MEMORY_STACKLEAK":e -> "Info Exposure (CWE-200)":w;
// Bug Detection Mechanisms vs Vulnerabilities
edge [style=solid, arrowhead=normal, dir=back, headport=_, tailport=_];
"Uninitialized Memory Usage (CWE-908)":e -> "KMSAN";
"Info Exposure (CWE-200)":e -> "KMSAN";
```

Linux Kernel Defence Map: Knowledge Sources

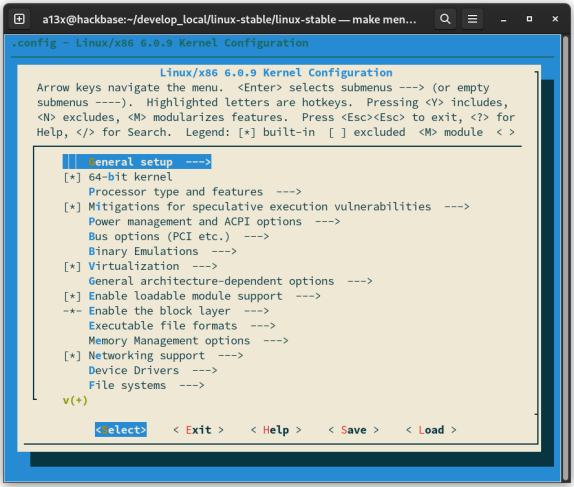
- Linux kernel security documentation
- grsecurity documentation
- Kernel Self Protection Project recommendations
- Microsoft Security Response Center (MSRC) publications
- And much more:

  github.com/a13xp0p0v/linux-kernel-defence-map#references

Nice Map! But What's in Practice?

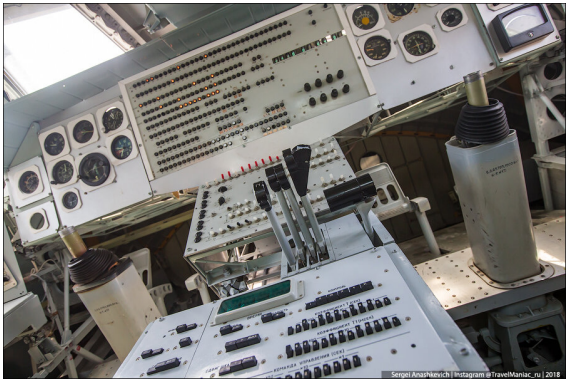
Victor Belov: Soviet scientists theorists (1972)

In Practice We Have This!



Linux Kernel Parameters

- Kconfig options (compile-time)
- Kernel cmdline arguments (boot-time)
- Sysctl parameters (runtime)



Buran spacecraft control panel

Linux Kernel Security Settings

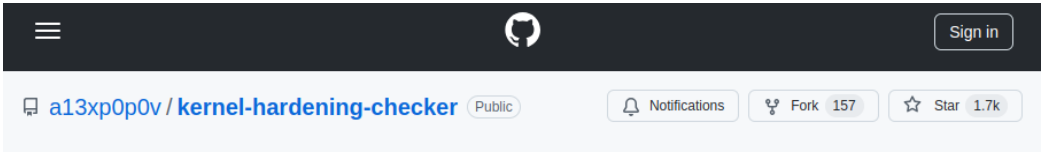
- There are **plenty** of Linux kernel security parameters
- A lot of them are **not enabled** by the major distros
- **Nobody** likes checking configs manually

Linux Kernel Security Settings

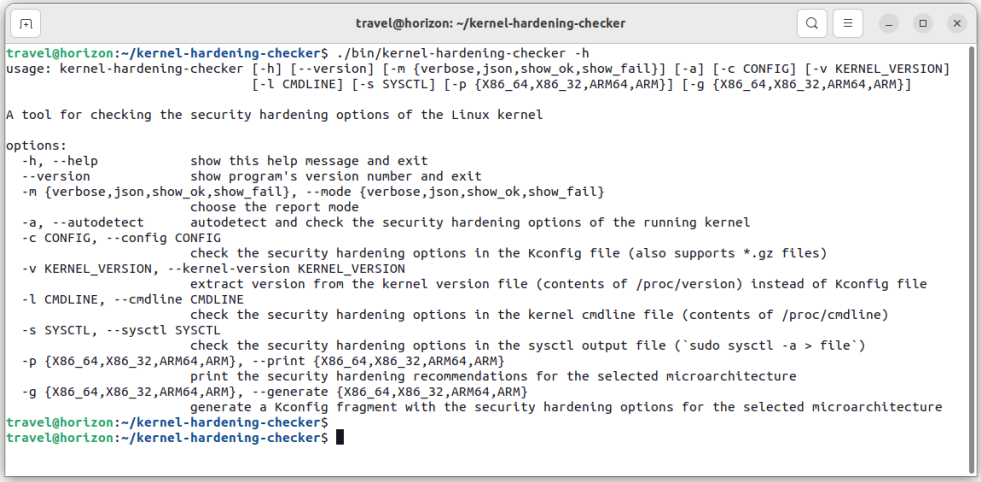
- There are **plenty** of Linux kernel security parameters
- A lot of them are **not enabled** by the major distros
- **Nobody** likes checking configs manually
- **So let computers do their job!**

Linux Kernel Security Settings

- There are **plenty** of Linux kernel security parameters
- A lot of them are **not enabled** by the major distros
- **Nobody** likes checking configs manually
- **So let computers do their job!**
- I created **kernel-hardening-checker** for checking the security-related parameters of the Linux kernel: github.com/a13xp0p0v/kernel-hardening-checker
- I started this work in 2018, the project is in continuous development



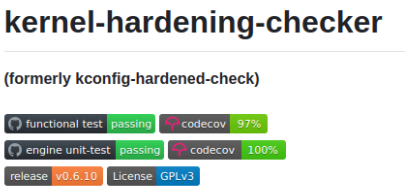
kernel-hardening-checker -h



kernel-hardening-checker: Demo Time

kernel-hardening-checker: Under the Hood

- **Free software project** under **GPL 3.0** license
- Written in **Python**
  - Please don't cry if my code looks like C code
  - I'm just a kernel developer :)
- Distribution via **pip/setuptools**
- Regular releases (linked to kernel releases)
- **CI**: automatic functional and unit tests, code coverage **>97%**
- Wonderful contributors from all over the world (**kudos**!)
- This tool is used by many GNU/Linux distributions (I'm glad!)



kernel-hardening-checker: Ideas and Plans

Vasily Tikhonenko: Highlanders. At the new construction site (1975)

- Allow redefining rules and expanding rule sets
- Add "with care" column to mark settings that
  - May break some kernel functionality
  - Or may introduce significant performance impact
- Evaluate the performance penalty of the recommended kernel settings (depends on the workload)
- Create documentation describing Linux kernel security settings
- Add RISC-V support
- And many more: github.com/a13xp0p0v/kernel-hardening-checker/issues

Conclusion

1. The **Linux Kernel Defence Map** helps to:
   - Get an **overview** of Linux kernel security
   - Develop a **threat model** for your GNU/Linux system
   - Learn about kernel defences that **can help** against these threats

Conclusion

1. The **Linux Kernel Defence Map** helps to:
   - Get an **overview** of Linux kernel security
   - Develop a **threat model** for your GNU/Linux system
   - Learn about kernel defences that **can help** against these threats

2. The **kernel-hardening-checker** tool helps to control the security-related parameters of your kernel

Conclusion

1. The **Linux Kernel Defence Map** helps to:
   - Get an **overview** of Linux kernel security
   - Develop a **threat model** for your GNU/Linux system
   - Learn about kernel defences that **can help** against these threats

2. The **kernel-hardening-checker** tool helps to control
   the security-related parameters of your kernel

3. Please **don't** change these settings **without** knowing your **threat model**

Thanks! Obrigado!

Enjoy the conference!

Contacts:

a13xp0p0v

a13xp0p0v@tuta.io

Blog: a13xp0p0v.github.io

Channel: t.me/linkersec