2025 IEEE 21st International Conference on Intelligent Computer Communication and Processing

Rootalyx: Stealthy Dynamic Analysis for Linux Malware using Kernel-Level Instrumentation and a Hybrid LSTM-Transformer Model

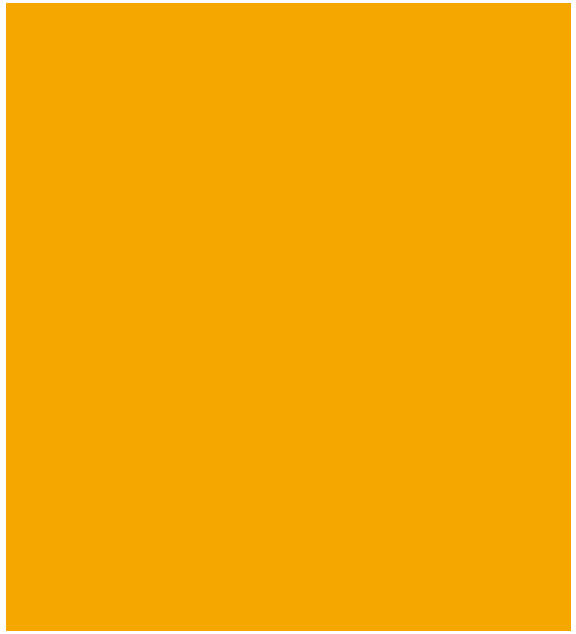
Doru Leșe, Andrei Sântoma, Ciprian Oprișa

Technical University of Cluj-Napoca & Bitdefender

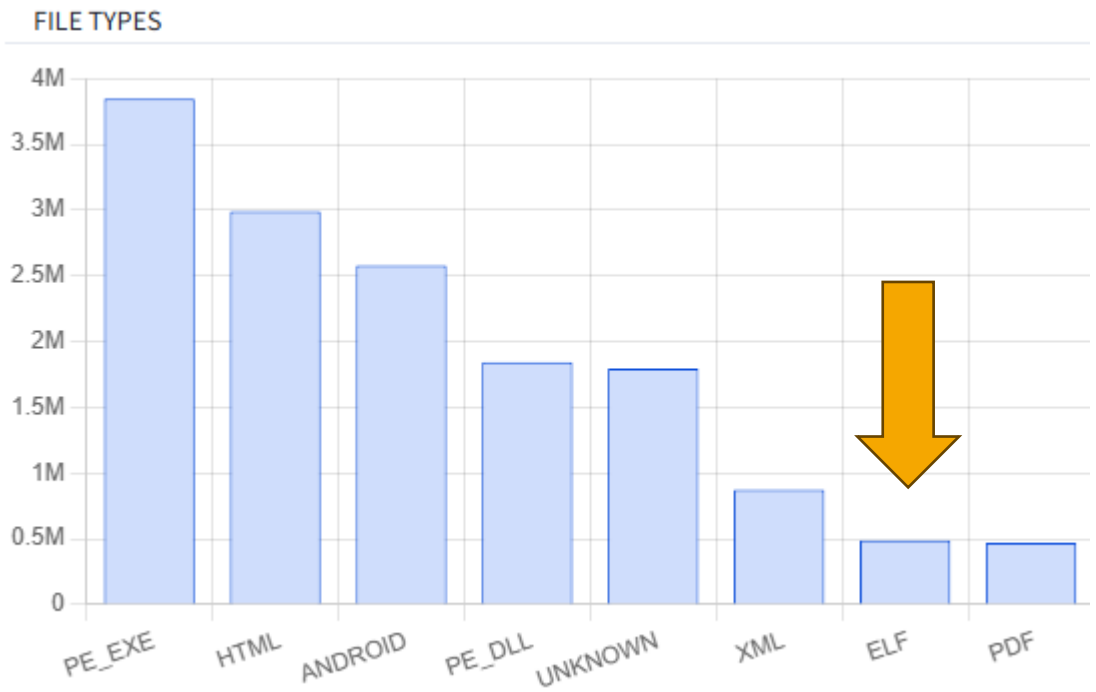
Table of contents

1. Introduction
2. Rootalyx architecture
3. Behavioral detection with Machine Learning
4. Experimental results
5. Conclusion

Introduction



Introduction



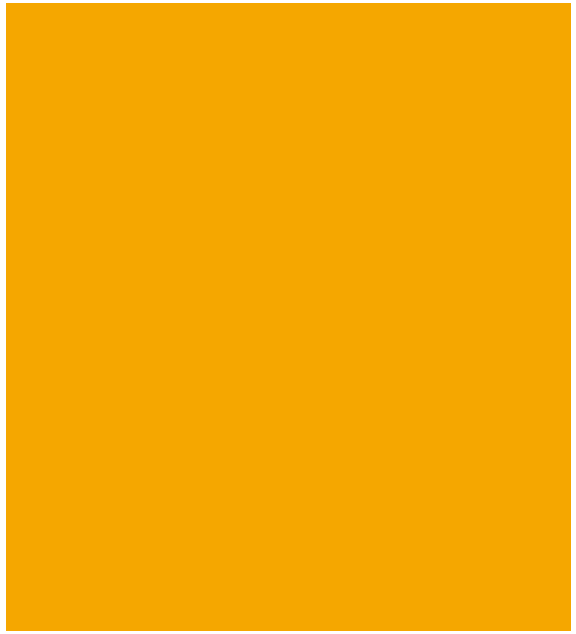
The file types uploaded to VirusTotal in the last 7 days [1]

- Linux has a strong reputation for security, yet it has become an increasingly attractive target for malicious actors due to its dominance in critical infrastructure.
- Dynamic analysis is a defense mechanism, where a sample is executed in an isolated environment (sandbox) to observe its behavior.
- Advanced malware can detect that it is monitored using common tools like *strace*.

Proposed approach

- **Central Idea:** If malware uses rootkits to hide, why can't we use the same techniques to hide our monitoring tools?
- **Case Study:** Our design is informed by an analysis of the *KoviD* rootkit, a modern Linux Kernel Module that implements comprehensive hiding strategies.
- **Our Contribution**: The Rootalyx system combines three components:
  - **Kernel-Level Monitoring:** Utilizes the *ftrace* subsystem for efficient, low-overhead system call instrumentation.
  - **Stealth Mechanisms:** Integrates rootkit-inspired hiding techniques to ensure a covert operational profile**.**
  - **Behavioral Classification:** A hybrid LSTM-Transformer model processes the collected data for automated malware detection.

Rootalyx architecture



Linux Kernel Modules (LKMs) and Tracing

- LKMs are objects of code that can be dynamically loaded to extend the kernel's functionality.
  - Device drivers, Filesystem drivers, Network protocols, Security Modules, etc.
- Kernel tracing is a powerful technique used to monitor and alter the behavior of the kernel.
  - kprobes, ftrace, eBPF, tracepoints, etc.



Official and unofficial icons for tracing frameworks [2]

System Architecture

- **Host:** Linux with QEMU/KVM
- **Manager:** A Python-based component orchestrates the analysis workflow, including VM lifecycle and sample execution.
- **Guest VM:** Ubuntu 24.04 LTS containing the Rootalyx Kernel Module.
- Analysis Workflow:
    - The Manager instantiates a VM from a clean base image.
    - A sample is transferred to the VM and executed.
    - The kernel module intercepts system calls from the sample's process tree.
    - An asynchronous kernel thread writes serialized call data to disk to minimize performance impact.
    - Logs are collected, and the VM is reverted.

Rootkit hiding techniques



KoviD hides the following:

- Module presence
- Processes
- Files and directories
- CPU usage
- Presence in kernel logs
- Kernel “taint” flag
- Intercepted functions
- Ftrace activation
- Network traffic
- Kernel symbols

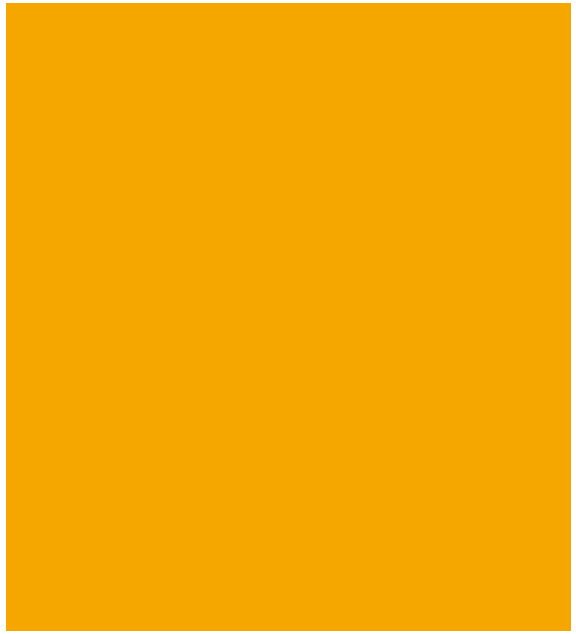
Rootalyx additionally hides:

- **Ftrace Trampolines:** The ftrace hook is detached from the global list post-registration to hide itself.
- **Module Initialization Function:** The module *__init* function, if discarded, will still have a reference in ftrace's internal tracking maps.

How to ensure module persistence?

Method	Description & Trade-offs	Boot Stage
User-space Mechanisms (e.g., Cron, Binary Injection)	Simple to implement but loads very late in the boot process and is easily disabled.	Late
/etc/modules (Chosen Method)	Standard Linux configuration. Simple, but loads after the core kernel is running.	
Initial RAM File System (initramfs)	Loads very early, before the root filesystem is mounted. Can be bypassed with boot parameters (e.g., noinitrd).	▼ Boot Sequence
Direct Kernel Modification	Provides deep integration and control. High complexity and maintenance overhead (requires recompilation).	
Firmware Injection (UEFI/BIOS)	Most persistent method; survives OS reinstallation. Extremely complex and hardware-dependent.	Early

Behavioral detection with Machine Learning



Data processing: From raw logs to model Input

1. System call serialization
   - The kernel module intercepts system calls and serializes their arguments.
   - This raw log data is structured but complex.

```
4994 access(AT_FDCWD, "/etc/ld.so.preload", R_OK) = -2
```

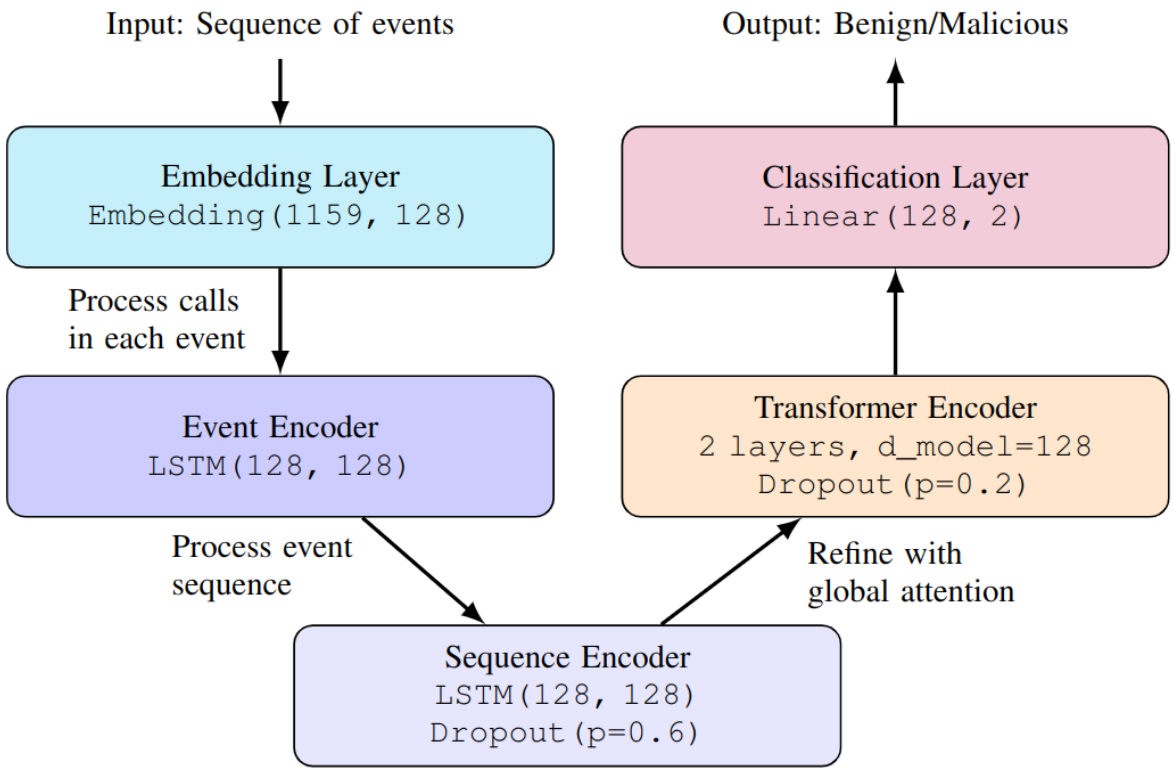
2. Normalization and sentence conversion
   - Raw log data is pre-processed and "flattened" into a sentence.
   - Special characters (e.g., {}, []) are replaced with spaces.

```
pid 4000 syscall access return_value 2 arg filename "/etc/ld.so.preload" arg mode 4
```

3. Byte Pair Encoding (BPE) tokenization
   - Sentences are tokenized using BPE with a 1000-token vocabulary.
   - BPE handles the dynamic, out-of-vocabulary nature of system call arguments (like file paths or IPs) by breaking them into sub-word units.

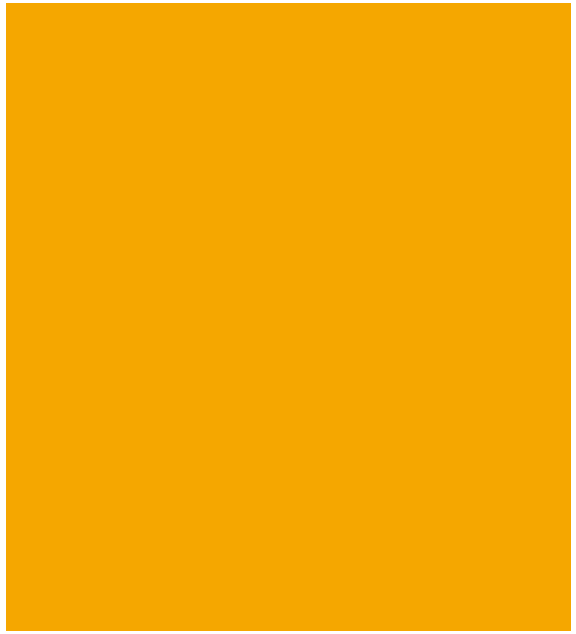
```
pid 4000 syscall access return_value 2 arg  filename "/etc/@@ l@@ d@@ .so.@@ pre@@ load@@ " arg mode 4
```

Model architecture



1. **Event-Level LSTM**: Processes the token sequence for each individual system call event
2. **Sequence-Level LSTM**: Models the temporal dependencies between consecutive system call events.
3. **Transformer Encoder**: Employs a global self-attention mechanism to capture long-range dependencies across the entire execution trace.

Experimental results



Evaluation criteria

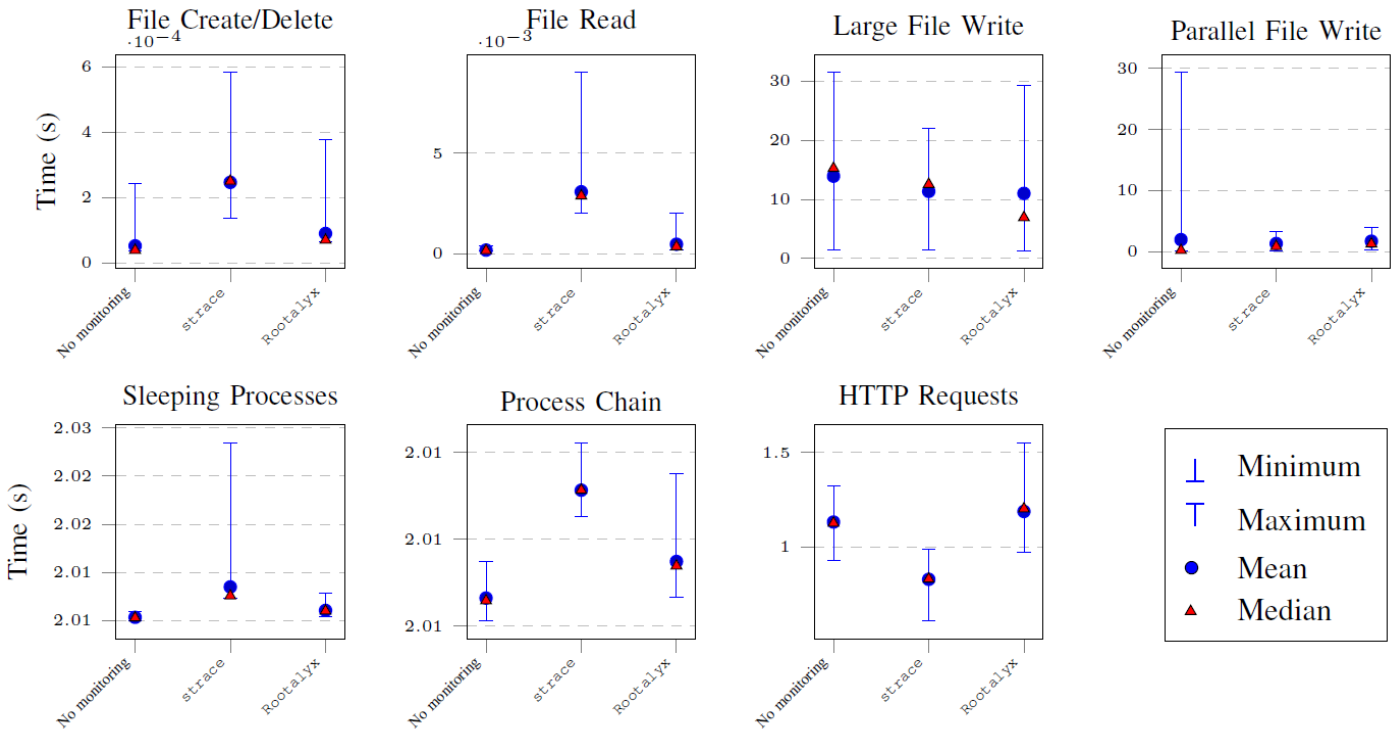
- **Stealth:** Can the module evade detection by common security tools?
- **Performance:** What is the overhead introduced by the monitoring system compared to strace and a baseline?
- **Efficacy:** How accurately does the machine learning model classify malware based on the collected data?

Stealth validation

- The system was evaluated against common anti-rootkit scanners, including Lynis, Chkrootkit, and Rkhunter. The Rootalyx module evaded detection by all of them.
- Manual inspection of kernel subsystems (eg. /proc, /sys/module) also confirmed the presence of the module and its helpers was concealed.



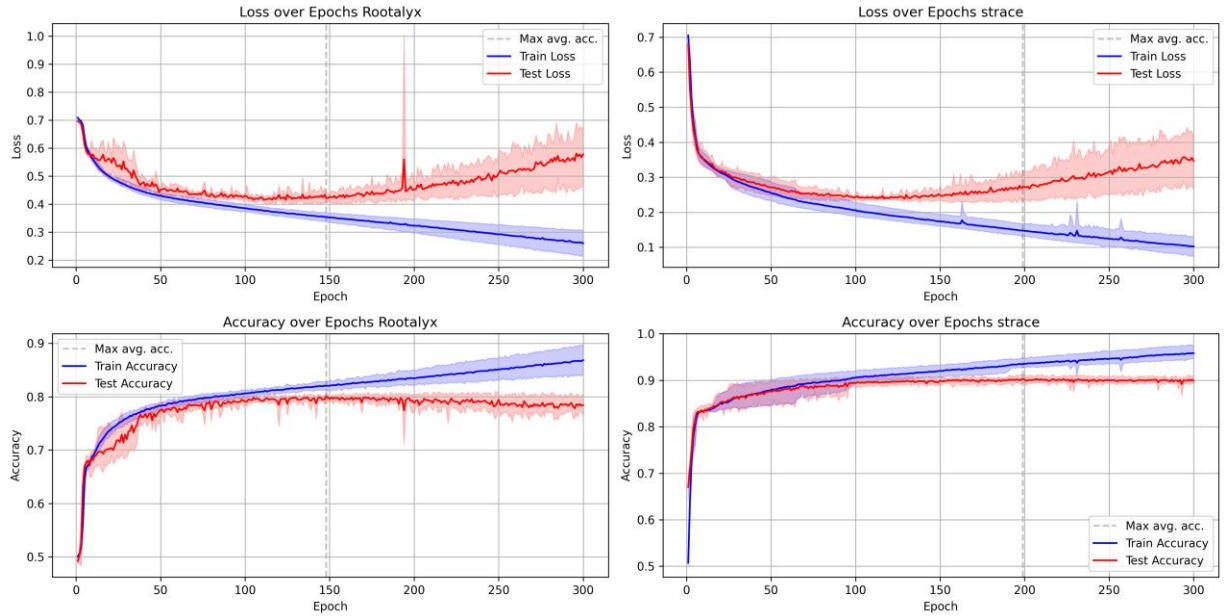
Performance overhead



Rootalyx demonstrated lower performance overhead compared to *strace*, particularly in I/O-intensive workloads. The impact on workloads with infrequent system calls was negligible.

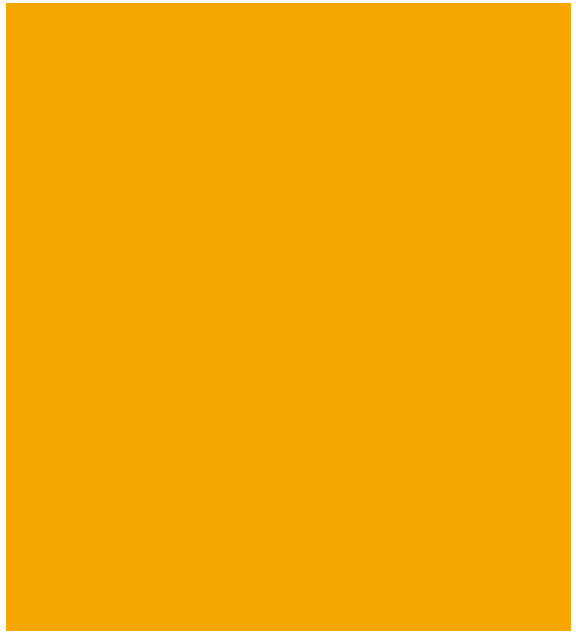
Detection Efficacy

- Dataset: 23,748 ELF executable files, with a balanced distribution of benign and malicious samples.
- Classification Accuracy:
  - Model trained on Rootalyx data: 80.34% Accuracy.
  - Model trained on strace data: 90.74% Accuracy.
- An analysis of misclassifications revealed 299 malware samples that were correctly identified by Rootalyx but missed by the strace model, from which at least one is strace aware.



Actual Class	Rootalyx	strace	No. Samples	% Samples
Benign	Correct	Correct	9326	42.80%
	Correct	Incorrect	542	2.48%
	Incorrect	Correct	1031	4.73%
	Incorrect	Incorrect	642	2.94%
Malware	Correct	Correct	7737	35.51%
	Correct	Incorrect	299	1.37%
	Incorrect	Correct	1588	7.28%
	Incorrect	Incorrect	622	2.85%

Conclusion



Conclusion and future work

- We presented Rootalyx, a stealthy dynamic analysis environment for Linux that integrates kernel-level monitoring with rootkit-inspired hiding techniques.
- The system successfully evades detection by common security tools while imposing lower performance overhead than strace.
- This research confirms that combining covert monitoring with advanced machine learning provides a robust framework for analyzing modern malware.
- The primary direction for future work is to improve the fidelity of system call argument collection to close the accuracy gap identified in our evaluation and to extend the dataset.

Bibliography

- [1] VirusTotal. "VirusTotal Statistics." VirusTotal. Accessed: Oct. 17, 2025. [Online]. Available: https://www.virustotal.com/gui/stats
- [2] B. Gregg. (2015) Choosing a linux tracer (2015). Accessed: Oct. 17, 2025. [Online]. Available: https://www.brendangregg.com/blog/2015-07-08/choosing-a-linux-tracer.html
- [3] i. carloslack, djolertrk. Kovid: Red-team linux kernel rootkit. Accessed: Oct. 17, 2025. [Online]. Available: https://github.com/carloslack/KoviD
- [4] Y. Guan and N. Ezzati-Jivan, “Malware system calls detection using hybrid system,” in 2021 IEEE International Systems Conference (SysCon), April 2021, pp. 1–8.
- [5] R. J. Maulana and G. P. Kusuma, “Malware Classification Based on System Call Sequences Using Deep Learning,” Advances in Science, Technology and Engineering Systems Journal, vol. 5, no. 4, pp. 207–216, 2020.

Thank you!

Any questions?