# Research Contributions in System Security

Dr. Yueqi Chen
Assistant Professor
Dept. of Computer Science, CU Boulder
December 7, 2022

Computer Science
UNIVERSITY OF COLORADO **BOULDER**

# Agenda

- Review of Paper Presentation This Semester
  - Some statistics
- Seven Types of Research Contributions in System Security
  - 14 types in fact: { Empirical, Artifact, Methodological, Theoretical, Dataset, Survey, Opinion } x { Attack, Defense }
- Outcome of Research in Industry
  - Series of research works, rather than one single paper, that finally contribute to an industrial product
  - Industrial products are carried out by people

Outside jaunting car Ireland, c. 1890–1900

# Review of Paper Presentations this Semester

**Gaukas**

- CFIXX: Object Type Integrity for C++ Virtual Dispatch – NDSS'16
- You Can Run but You Can't Read: Preventing Disclosure Exploits in Executable Code – CCS'14
- CAIN: Silently Breaking ASLR in the Cloud – WOOT'15
- Weaponizing Middleboxes for TCP Reflected Amplification – SEC'21

**Raghuveer**

- Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications - S&P'15
- MARX: Uncovering Class Hierarchies in C++ Programs - NDSS'17
- Oxymoron: Making Fine-Grained Memory Randomization Practical by Allowing Code Sharing - SEC'14
- PAC it up: Towards Pointer Integrity using ARM Pointer Authentication - SEC'19

**Jackson**

- HexType: Efficient Detection of Type Confusion Errors for C++ - CCS'17
- Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization - S&P'12
- Meltdown: Reading Kernel Memory from User Space - SEC'18
- PointGuardTM: Protecting Pointers from Buffer Overflow Vulnerabilities – SEC'03

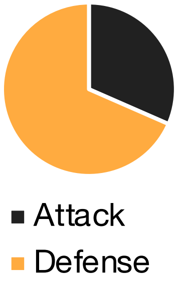
**Kidus**

- Jump over ASLR: Attacking Branch Predictors to Bypass ASLR – MICRO'16
- Shuffler: Fast and Deployable Continuous Code Re-Randomization – OSDI'16
- BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows – HPCA'21
- KASLR is Dead: Long Live KASLR – ESSoS'17

**Sylvia**

- Type Casting Verification: Stopping an Emerging Attack Vector – SEC'15
- Gadge Me If You Can: Secure and Efficient Ad-hoc Instruction-Level Randomization for x86 and ARM – ASIACCS'13
- SafeHidden: An Efficient and Secure Information Hiding Technique Using Re-randomization – SEC'19
- PUFs: Myth, Fact or Busted? A Secure Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon - CHES'12

# Some Statistics

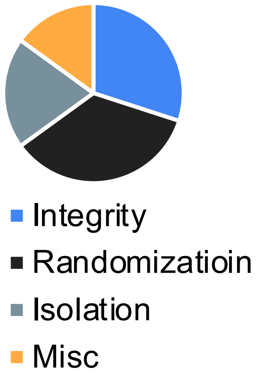


**Attack:**
1. CAIN
2. Weaponizing
3. COOP
4. MARX
5. Meltdown
6. Jump over ASLR

**Defense :**
1. CFIXX
2. You Can Run but You Can't Read
3. Oxymoron
4. PAC it up
5. HexType
6. Smashing the Gadgets
7. PointGuardTM
8. Shuffler
9. BlockHammer
10. KASLR is Dead
11. Type Casting Verification
12. Gadge Me if You Can
13. SafeHidden
14. PUFs

**Panel 3**: Quo vadis Cyber Security? Are we really building defense systems, or are we all just into attacks for fun and profit?
**Moderator**: Engin Kirda (Northeastern University)
**Panelists**: Yan Shoshitaishvili (Arizona State University)
XiaoFeng Wang (Indiana University)
Lujo Bauer (Carnegie Mellon University)
Dongyan Xu (Purdue University)

# Some Statistics (cont.)



**Integrity**:
1. COOP - 15
2. Type Casting Verification - 15
3. CFIXX - 16
4. HexType – 17
5. PointGuardTM - 03
6. PAC it up - 19

object type integrity

pointer integrity

**Randomization**:
1. Smashing the Gadgets - 12
2. Oxymoron - 14
3. Shuffler - 16
4. Gadge Me if You Can - 13
5. SafeHidden - 19
6. CAIN – 15
7. Jump over ASLR - 16

How to random?
Instruction-level, function level, program level;
random for one-time, continuous randomization

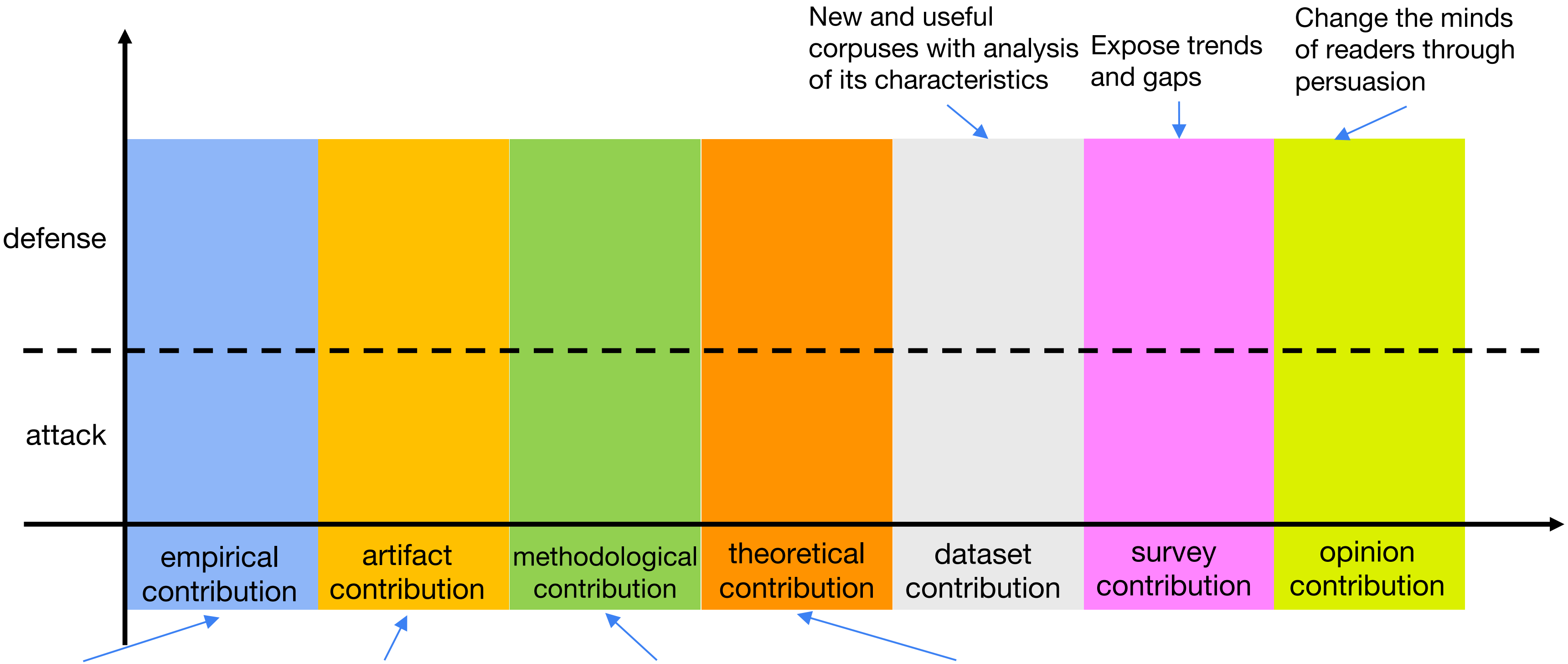
How to break randomization?
e.g., VM side channel, Branch Predictors side channel

**Isolation**:
1. You Can Run but You Can’t Read
2. Meltdown
3. KASLR is Dead

permission isolation
address space isolation

**Misc**:
1. BlockHammer
2. Weaponizing
3. MARX
4. PUFs

# Seven Types of Research Contribution in System Security



New and useful corpuses with analysis of its characteristics

Expose trends and gaps

Change the minds of readers through persuasion

defense

attack

empirical contribution

artifact contribution

methodological contribution

theoretical contribution

dataset contribution

survey contribution

opinion contribution

New findings based on observation and data gathering

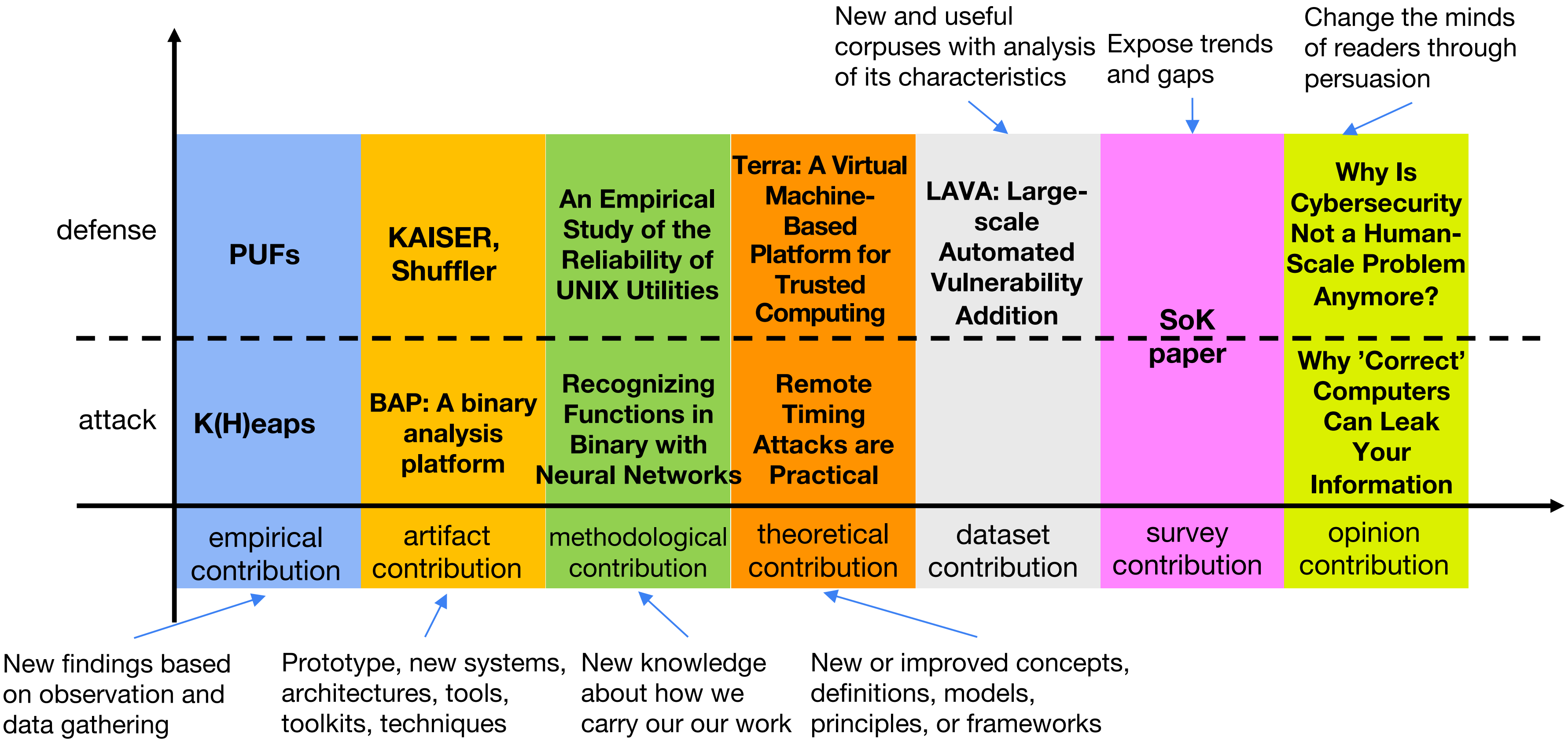
Prototype, new systems, architectures, tools, toolkits, techniques

New knowledge about how we carry our our work

New or improved concepts, definitions, models, principles, or frameworks

*Idea borrowed from "Research Contributions in Human-Computer Interaction", Jacob O. Wobbrock, Julie A. Kientz*

# Examples of the Seven Types of Contribution



New and useful corpuses with analysis of its characteristics

Expose trends and gaps

Change the minds of readers through persuasion

| | empirical contribution | artifact contribution | methodological contribution | theoretical contribution | dataset contribution | survey contribution | opinion contribution |
|---|---|---|---|---|---|---|---|
| defense | **PUFs** | **KAISER, Shuffler** | **An Empirical Study of the Reliability of UNIX Utilities** | **Terra: A Virtual Machine-Based Platform for Trusted Computing** | **LAVA: Large-scale Automated Vulnerability Addition** | **SoK paper** | **Why Is Cybersecurity Not a Human-Scale Problem Anymore?** |
| attack | **K(H)eaps** | **BAP: A binary analysis platform** | **Recognizing Functions in Binary with Neural Networks** | **Remote Timing Attacks are Practical** | | | **Why 'Correct' Computers Can Leak Your Information** |

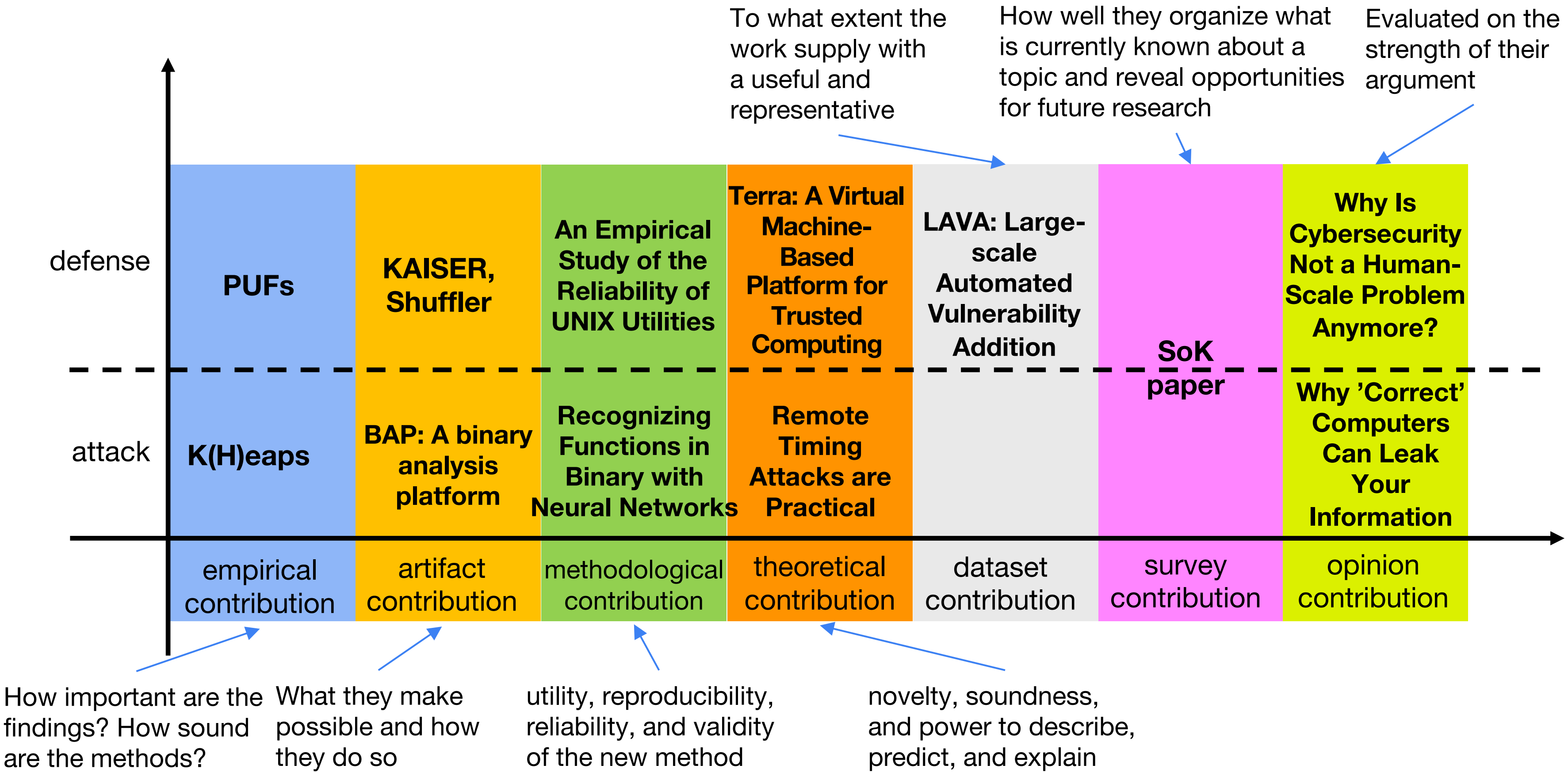
New findings based on observation and data gathering

Prototype, new systems, architectures, tools, toolkits, techniques

New knowledge about how we carry our our work

New or improved concepts, definitions, models, principles, or frameworks

# Evaluation Criteria for the Seven Types of Contribution



To what extent the
work supply with
a useful and
representative

How well they organize what
is currently known about a
topic and reveal opportunities
for future research

Evaluated on the
strength of their
argument

| | empirical contribution | artifact contribution | methodological contribution | theoretical contribution | dataset contribution | survey contribution | opinion contribution |
|---|---|---|---|---|---|---|---|
| defense | **PUFs** | **KAISER, Shuffler** | **An Empirical Study of the Reliability of UNIX Utilities** | **Terra: A Virtual Machine-Based Platform for Trusted Computing** | **LAVA: Large-scale Automated Vulnerability Addition** | **SoK paper** | **Why Is Cybersecurity Not a Human-Scale Problem Anymore?** |
| attack | **K(H)eaps** | **BAP: A binary analysis platform** | **Recognizing Functions in Binary with Neural Networks** | **Remote Timing Attacks are Practical** | | | **Why 'Correct' Computers Can Leak Your Information** |

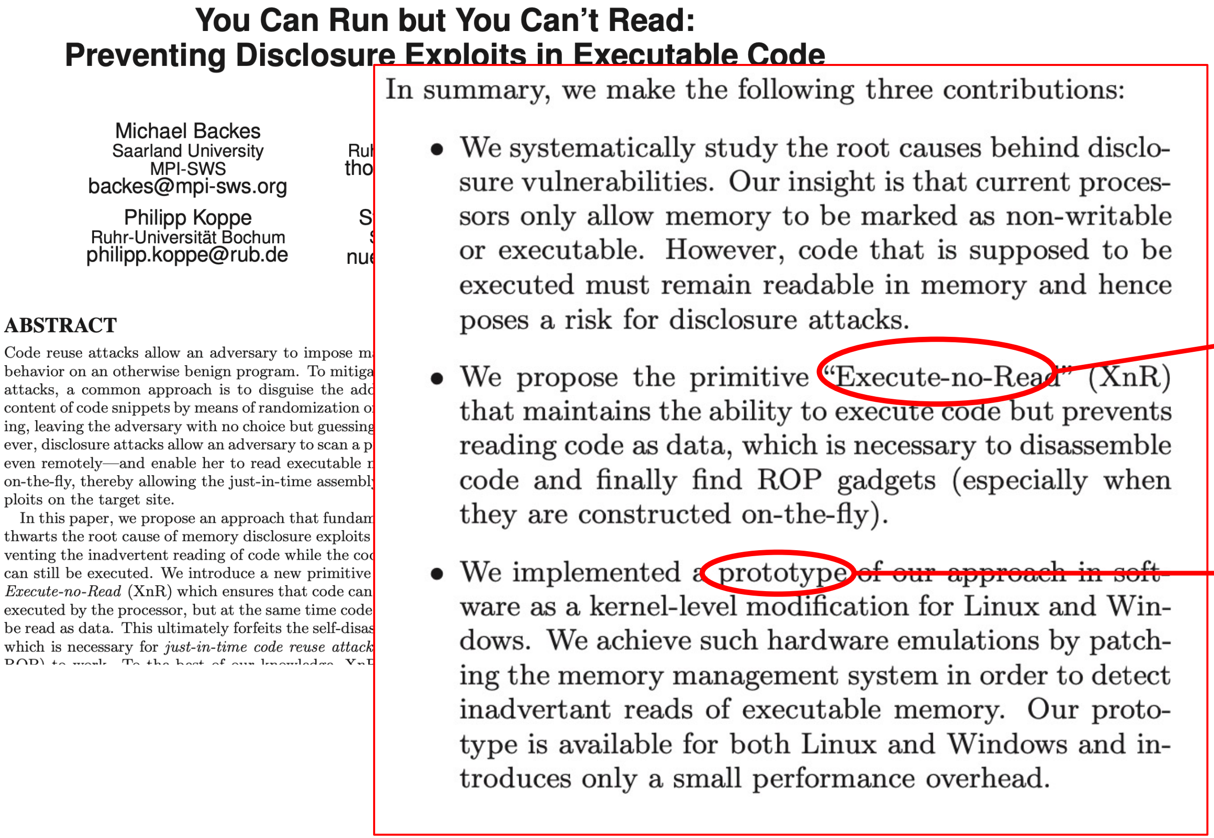
How important are the
findings? How sound
are the methods?

What they make
possible and how
they do so

utility, reproducibility,
reliability, and validity
of the new method

novelty, soundness,
and power to describe,
predict, and explain

# Case Study – I



**You Can Run but You Can't Read:**
**Preventing Disclosure Exploits in Executable Code**

Michael Backes
Saarland University
MPI-SWS
backes@mpi-sws.org

Philipp Koppe
Ruhr-Universität Bochum
philipp.koppe@rub.de

Ruh
tho

S
S
nue

**ABSTRACT**

Code reuse attacks allow an adversary to impose m
behavior on an otherwise benign program. To mitiga
attacks, a common approach is to disguise the add
content of code snippets by means of randomization o
ing, leaving the adversary with no choice but guessing
ever, disclosure attacks allow an adversary to scan a p
even remotely—and enable her to read executable r
on-the-fly, thereby allowing the just-in-time assembl
ploits on the target site.

In this paper, we propose an approach that fundam
thwarts the root cause of memory disclosure exploits
venting the inadvertent reading of code while the co
can still be executed. We introduce a new primitive
*Execute-no-Read* (XnR) which ensures that code can
executed by the processor, but at the same time code
be read as data. This ultimately forfeits the self-disas
which is necessary for *just-in-time code reuse attack*
ROP) to work. To the best of our knowledge, XnR

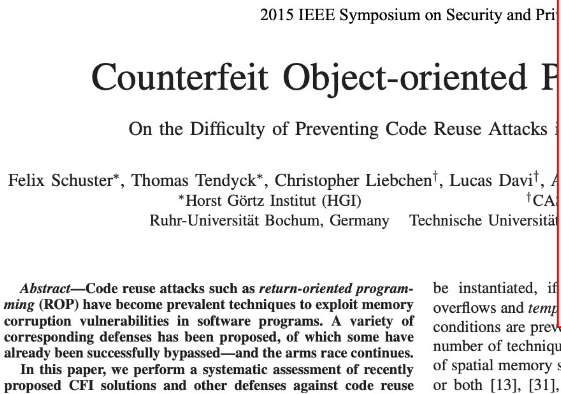
In summary, we make the following three contributions:

- We systematically study the root causes behind disclosure vulnerabilities. Our insight is that current processors only allow memory to be marked as non-writable or executable. However, code that is supposed to be executed must remain readable in memory and hence poses a risk for disclosure attacks.
- We propose the primitive "Execute-no-Read" (XnR) that maintains the ability to execute code but prevents reading code as data, which is necessary to disassemble code and finally find ROP gadgets (especially when they are constructed on-the-fly).
- We implemented a prototype of our approach in software as a kernel-level modification for Linux and Windows. We achieve such hardware emulations by patching the memory management system in order to detect inadvertant reads of executable memory. Our prototype is available for both Linux and Windows and introduces only a small performance overhead.

methodological contribution

artifact contribution

# Case Study – II



2015 IEEE Symposium on Security and Pri

## Counterfeit Object-oriented P

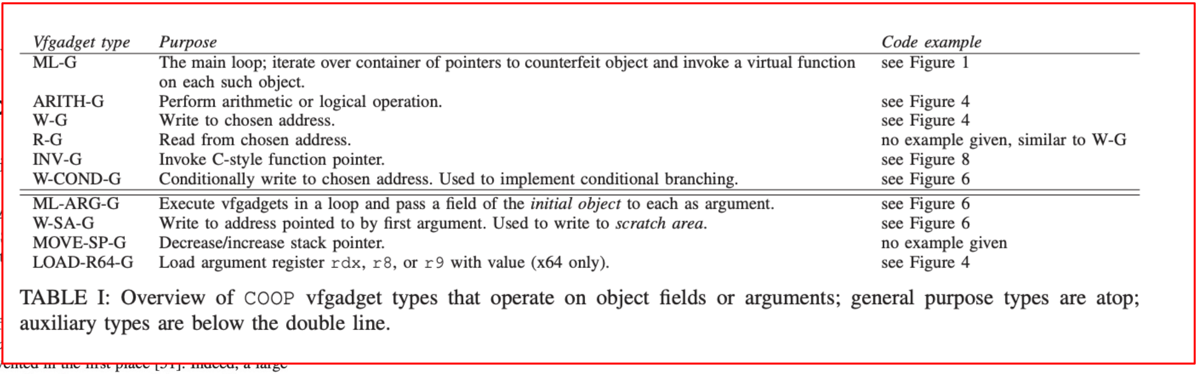
On the Difficulty of Preventing Code Reuse Attacks i

Felix Schuster[*], Thomas Tendyck[*], Christopher Liebchen[†], Lucas Davi[†], A
[*]Horst Görtz Institut (HGI)    [†]CA
Ruhr-Universität Bochum, Germany    Technische Universität

***Abstract*****—Code reuse attacks such as *****return-oriented programming***** (ROP) have become prevalent techniques to exploit memory corruption vulnerabilities in software programs. A variety of corresponding defenses has been proposed, of which some have already been successfully bypassed—and the arms race continues.**

**In this paper, we perform a systematic assessment of recently proposed CFI solutions and other defenses against code reuse**

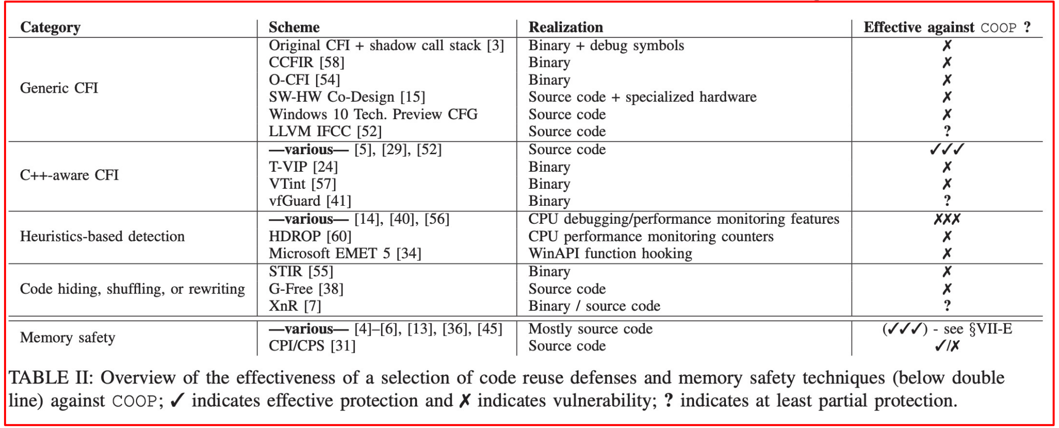
be instantiated, if
overflows and *temp*
conditions are prevented in the first place [51]. Indeed, a large number of techniques have been proposed that provide means of spatial memory safety [5], [6], temporal memory safety [4], or both [13], [31], [36], [45]. On the downside, for precise



| *Vfgadget type* | *Purpose* | *Code example* |
|---|---|---|
| ML-G | The main loop; iterate over container of pointers to counterfeit object and invoke a virtual function on each such object. | see Figure 1 |
| ARITH-G | Perform arithmetic or logical operation. | see Figure 4 |
| W-G | Write to chosen address. | see Figure 4 |
| R-G | Read from chosen address. | no example given, similar to W-G |
| INV-G | Invoke C-style function pointer. | see Figure 8 |
| W-COND-G | Conditionally write to chosen address. Used to implement conditional branching. | see Figure 6 |
| ML-ARG-G | Execute vfgadgets in a loop and pass a field of the *initial object* to each as argument. | see Figure 6 |
| W-SA-G | Write to address pointed to by first argument. Used to write to *scratch area*. | see Figure 6 |
| MOVE-SP-G | Decrease/increase stack pointer. | no example given |
| LOAD-R64-G | Load argument register `rdx`, `r8`, or `r9` with value (x64 only). | see Figure 4 |

TABLE I: Overview of `COOP` vfgadget types that operate on object fields or arguments; general purpose types are atop; auxiliary types are below the double line.

artifact contribution

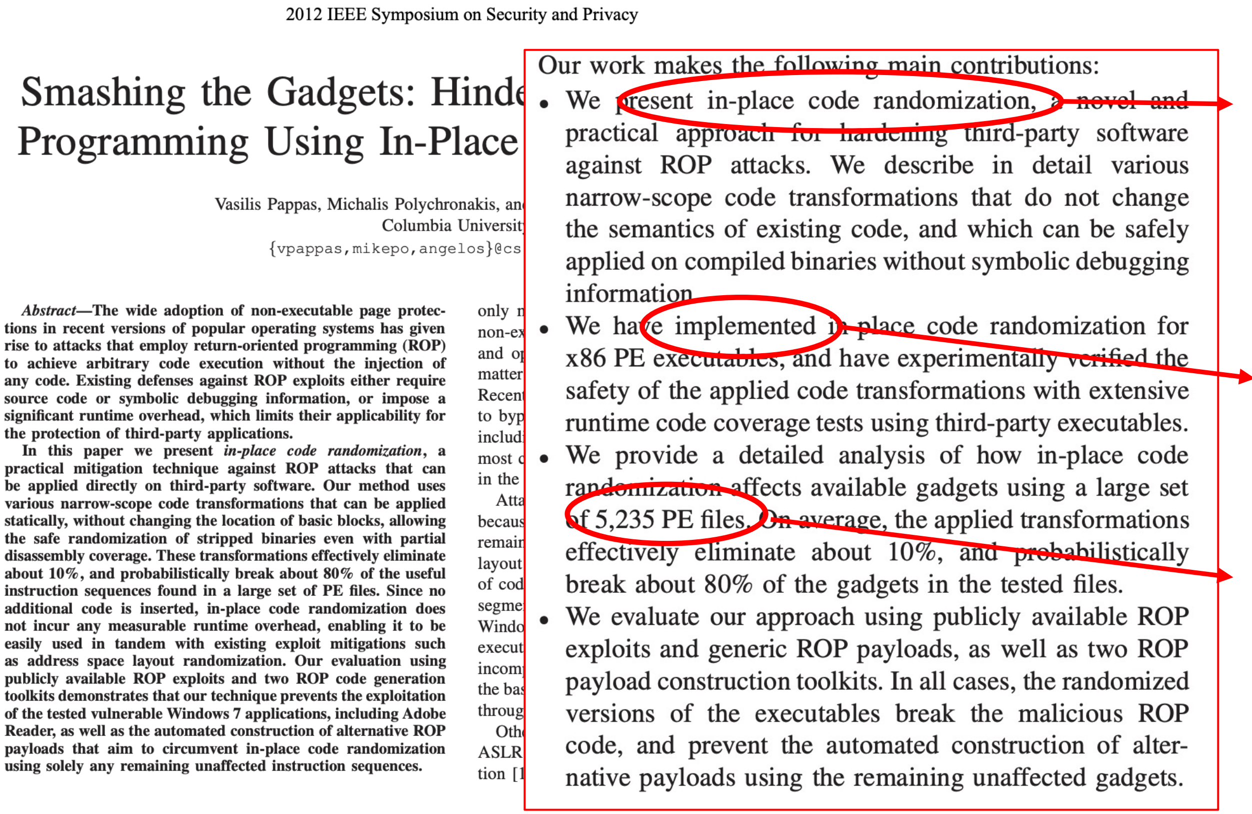


| **Category** | **Scheme** | **Realization** | **Effective against `COOP` ?** |
|---|---|---|---|
| Generic CFI | Original CFI + shadow call stack [3] | Binary + debug symbols | ✗ |
| | CCFIR [58] | Binary | ✗ |
| | O-CFI [54] | Binary | ✗ |
| | SW-HW Co-Design [15] | Source code + specialized hardware | ✗ |
| | Windows 10 Tech. Preview CFG | Source code | ✗ |
| | LLVM IFCC [52] | Source code | ? |
| C++-aware CFI | **—various—** [5], [29], [52] | Source code | ✓✓✓ |
| | T-VIP [24] | Binary | ✗ |
| | VTint [57] | Binary | ✗ |
| | vfGuard [41] | Binary | ? |
| Heuristics-based detection | **—various—** [14], [40], [56] | CPU debugging/performance monitoring features | ✗✗✗ |
| | HDROP [60] | CPU performance monitoring counters | ✗ |
| | Microsoft EMET 5 [34] | WinAPI function hooking | ✗ |
| Code hiding, shuffling, or rewriting | STIR [55] | Binary | ✗ |
| | G-Free [38] | Source code | ✗ |
| | XnR [7] | Binary / source code | ? |
| Memory safety | **—various—** [4]–[6], [13], [36], [45] | Mostly source code | (✓✓✓) - see §VII-E |
| | CPI/CPS [31] | Source code | ✓/✗ |

TABLE II: Overview of the effectiveness of a selection of code reuse defenses and memory safety techniques (below double line) against `COOP`; ✓ indicates effective protection and ✗ indicates vulnerability; ? indicates at least partial protection.

survey contribution

# Case Study – III



2012 IEEE Symposium on Security and Privacy

Smashing the Gadgets: Hinde
Programming Using In-Place

Vasilis Pappas, Michalis Polychronakis, an
Columbia Universit
`{vpappas,mikepo,angelos}@cs`

***Abstract*—The wide adoption of non-executable page protections in recent versions of popular operating systems has given rise to attacks that employ return-oriented programming (ROP) to achieve arbitrary code execution without the injection of any code. Existing defenses against ROP exploits either require source code or symbolic debugging information, or impose a significant runtime overhead, which limits their applicability for the protection of third-party applications.**

**In this paper we present *in-place code randomization*, a practical mitigation technique against ROP attacks that can be applied directly on third-party software. Our method uses various narrow-scope code transformations that can be applied statically, without changing the location of basic blocks, allowing the safe randomization of stripped binaries even with partial disassembly coverage. These transformations effectively eliminate about 10%, and probabilistically break about 80% of the useful instruction sequences found in a large set of PE files. Since no additional code is inserted, in-place code randomization does not incur any measurable runtime overhead, enabling it to be easily used in tandem with existing exploit mitigations such as address space layout randomization. Our evaluation using publicly available ROP exploits and two ROP code generation toolkits demonstrates that our technique prevents the exploitation of the tested vulnerable Windows 7 applications, including Adobe Reader, as well as the automated construction of alternative ROP payloads that aim to circumvent in-place code randomization using solely any remaining unaffected instruction sequences.**

only n
non-ex
and o
matter
Recent
to byp
includ
most o
in the
Atta
becaus
remain
layout
of cod
segme
Windo
execut
incom
the bas
throug
Othe
ASLR
tion [1

Our work makes the following main contributions:

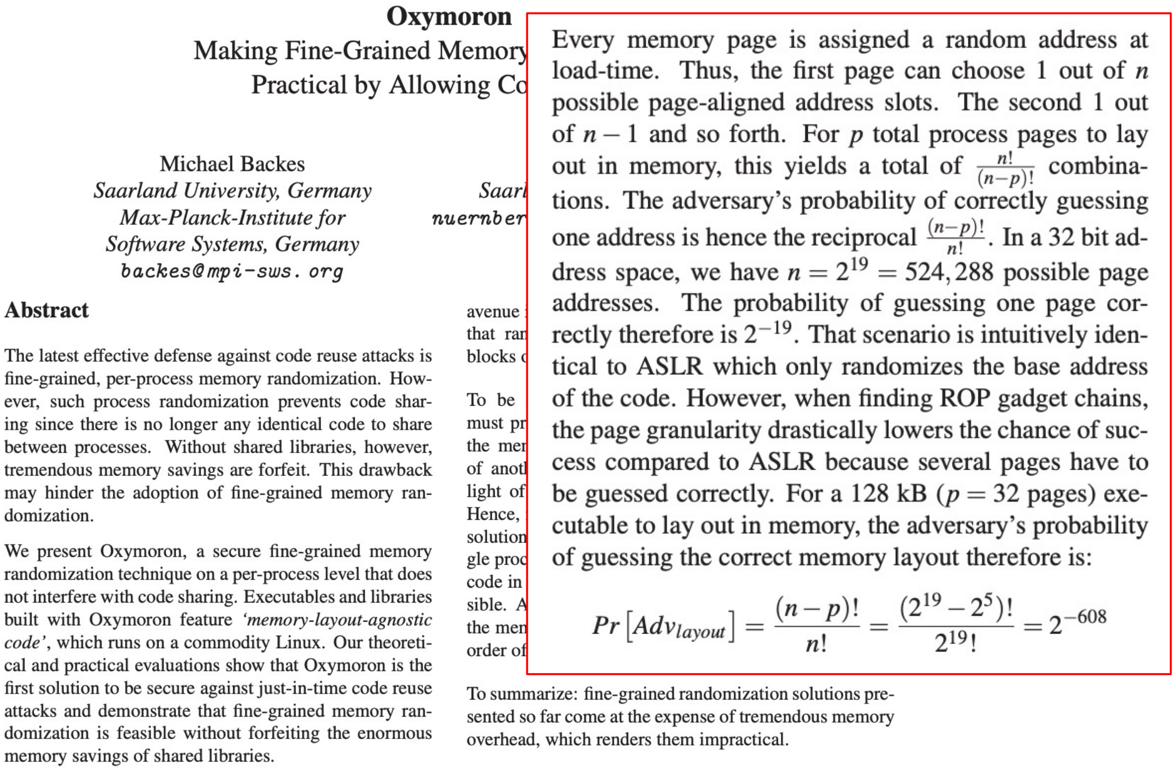
- We present in-place code randomization, a novel and practical approach for hardening third-party software against ROP attacks. We describe in detail various narrow-scope code transformations that do not change the semantics of existing code, and which can be safely applied on compiled binaries without symbolic debugging information.
- We have implemented in-place code randomization for x86 PE executables, and have experimentally verified the safety of the applied code transformations with extensive runtime code coverage tests using third-party executables.
- We provide a detailed analysis of how in-place code randomization affects available gadgets using a large set of 5,235 PE files. On average, the applied transformations effectively eliminate about 10%, and probabilistically break about 80% of the gadgets in the tested files.
- We evaluate our approach using publicly available ROP exploits and generic ROP payloads, as well as two ROP payload construction toolkits. In all cases, the randomized versions of the executables break the malicious ROP code, and prevent the automated construction of alternative payloads using the remaining unaffected gadgets.

methodological contribution

artifact contribution

empirical contribution

# Case Study – IV



**Oxymoron**
Making Fine-Grained Memory
Practical by Allowing Co

Michael Backes
*Saarland University, Germany*
*Max-Planck-Institute for*
*Software Systems, Germany*
`backes@mpi-sws.org`

*Saarl*
`nuernber`

**Abstract**

The latest effective defense against code reuse attacks is fine-grained, per-process memory randomization. However, such process randomization prevents code sharing since there is no longer any identical code to share between processes. Without shared libraries, however, tremendous memory savings are forfeit. This drawback may hinder the adoption of fine-grained memory randomization.

We present Oxymoron, a secure fine-grained memory randomization technique on a per-process level that does not interfere with code sharing. Executables and libraries built with Oxymoron feature *'memory-layout-agnostic code'*, which runs on a commodity Linux. Our theoretical and practical evaluations show that Oxymoron is the first solution to be secure against just-in-time code reuse attacks and demonstrate that fine-grained memory randomization is feasible without forfeiting the enormous memory savings of shared libraries.

avenue
that ran
blocks c

To be
must pr
the mem
of anot
light of
Hence,
solution
gle proc
code in
sible. A
the mem
order of

Every memory page is assigned a random address at load-time. Thus, the first page can choose 1 out of $n$ possible page-aligned address slots. The second 1 out of $n-1$ and so forth. For $p$ total process pages to lay out in memory, this yields a total of $\frac{n!}{(n-p)!}$ combinations. The adversary's probability of correctly guessing one address is hence the reciprocal $\frac{(n-p)!}{n!}$. In a 32 bit address space, we have $n = 2^{19} = 524,288$ possible page addresses. The probability of guessing one page correctly therefore is $2^{-19}$. That scenario is intuitively identical to ASLR which only randomizes the base address of the code. However, when finding ROP gadget chains, the page granularity drastically lowers the chance of success compared to ASLR because several pages have to be guessed correctly. For a 128 kB ($p = 32$ pages) executable to lay out in memory, the adversary's probability of guessing the correct memory layout therefore is:

$$Pr\left[Adv_{layout}\right] = \frac{(n-p)!}{n!} = \frac{(2^{19}-2^{5})!}{2^{19}!} = 2^{-608}$$

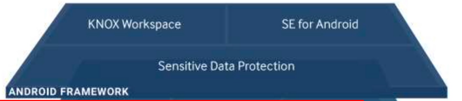
To summarize: fine-grained randomization solutions presented so far come at the expense of tremendous memory overhead, which renders them impractical.

theoretical contribution

# How Research Works Contribute to Industrial Products

Whitepaper: Samsung Knox Security Solution, Version 2.2 May, 2017

# Samsung’s Patented Real-time Kernel Protection (RKP)



- ❖ A security monitor in either the Secure World of ARM TrustZone or a thin hypervisor
  - ➢ TEE, first defined in 2009
  - ➢ The first hypervisor supporting full virtualization were SI
    40 produced in Jan 1967. Classified into two types in Ro
    PhD thesis
- ❖ Prevent modification of kernel code, injection of ur
  execution of userspace code in the privileged mo
  - ➢ IMA
  - ➢ kGuard: Lightweight kernel protection against return-to-
  - ➢ Many other previous works
- ❖ Prevent DMA Attacks, Control Flow Attacks, etc

**True IOMMU protection from DMA attacks: When copy is faster than zero copy**
A Markuze, A Morrison, D Tsafrir - Proceedings of the Twenty-First …, 2016 - dl.acm.org
… True **DMA attack** protection We propose an alternative intra-OS protection design … We use the **IOMMU** to restrict device access to a set of permanently-mapped shadow **DMA** buffers and …
☆ Save 99 Cite Cited by 51 Related articles All 7 versions

**IOMMU-resistant DMA attacks**
G Kupfer, D Tsafrir, N Amit - 2018 - cs.technion.ac.il
… presenting several concrete **attacks** that remain valid even when an **IOMMU** is present … **attacks** by presenting classic **DMA attacks**, the **IOMMU** protection against them, and recent **attacks** …
☆ Save 99 Cite Cited by 6 Related articles »

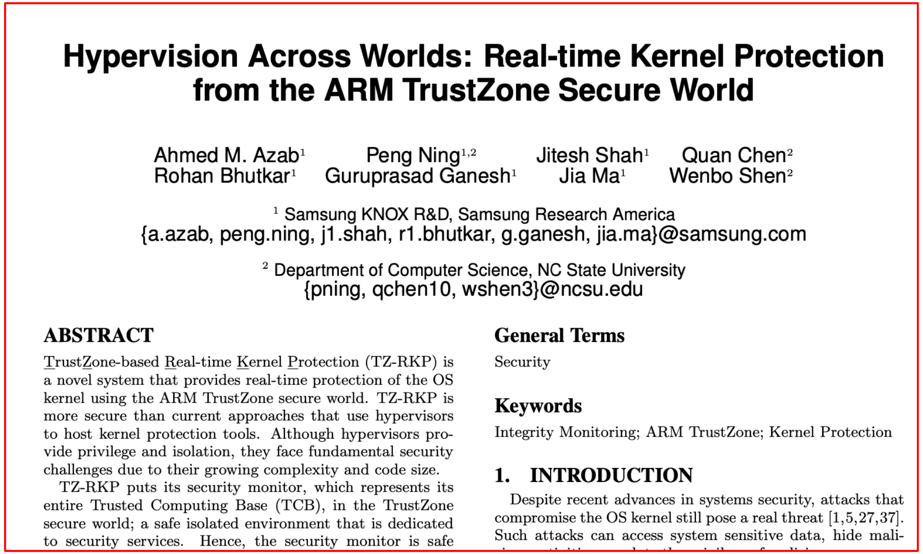
**Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals**
T Markettos, C Rothwell, BF Gutstein, A Pearce… - 2019 - repository.cam.ac.uk
… They are included here as a demonstration of an OS that makes poor use of the **IOMMU** to defend against **DMA attacks** and the use of our platform to reproduce state of the art …
☆ Save 99 Cite Cited by 69 Related articles All 13 versions »

**Characterizing, exploiting, and detecting DMA code injection vulnerabilities in the presence of an IOMMU**
M Alex, S Vargaftik, G Kupfer, B Pismeny… - Proceedings of the …, 2021 - dl.acm.org
… First, we describe classic **DMA attacks** and the **IOMMU** protection against them. Then, we discuss well-established protection practices to prevent privilege escalation (ie, code injection) …
☆ Save 99 Cite Cited by 4 Related articles All 6 versions

**Bypassing IOMMU protection against I/O attacks**
B Morgan, E Alata, V Nicomette… - 2016 Seventh Latin …, 2016 - ieeexplore.ieee.org
… In this paper, we focus on **DMA attacks**. These **attacks** were described in several studies. In … Fortunately, most of these vulnerabilities have been fixed with the integration of **IOMMU** in …
☆ Save 99 Cite Cited by 20 Related articles All 6 versions

# Industrial Products Are Carried Out by People

- It costs roughly 0.5 million to graduate one PhD student
  - Tuition, RA/TA, fringe benefits, double pay in summer, conference travelling, fees p/semester, IDC
  - Quit or dismissal halfway
- Your contribution is more than 0.5 million
  - High-tech needs people like Steven Jobs, Bill Gates, and You to make progress



## Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World

Ahmed M. Azab[1] Peng Ning[1,2] Jitesh Shah[1] Quan Chen[2]
Rohan Bhutkar[1] Guruprasad Ganesh[1] Jia Ma[1] Wenbo Shen[2]

[1] Samsung KNOX R&D, Samsung Research America
{a.azab, peng.ning, j1.shah, r1.bhutkar, g.ganesh, jia.ma}@samsung.com

[2] Department of Computer Science, NC State University
{pning, qchen10, wshen3}@ncsu.edu

### ABSTRACT

TrustZone-based Real-time Kernel Protection (TZ-RKP) is a novel system that provides real-time protection of the OS kernel using the ARM TrustZone secure world. TZ-RKP is more secure than current approaches that use hypervisors to host kernel protection tools. Although hypervisors provide privilege and isolation, they face fundamental security challenges due to their growing complexity and code size.

TZ-RKP puts its security monitor, which represents its entire Trusted Computing Base (TCB), in the TrustZone secure world; a safe isolated environment that is dedicated to security services. Hence, the security monitor is safe

### General Terms

Security

### Keywords

Integrity Monitoring; ARM TrustZone; Kernel Protection

### 1. INTRODUCTION

Despite recent advances in systems security, attacks that compromise the OS kernel still pose a real threat [1,5,27,37]. Such attacks can access system sensitive data, hide mali-