Research Contributions in System Security

Dr. Yueqi Chen
Assistant Professor
Dept. of Computer Science, CU Boulder
December 7, 2022
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
# Review of Paper Presentations this Semester

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaukas</td>
<td>CFIXX: Object Type Integrity for C++ Virtual Dispatch</td>
<td>NDSS’16</td>
</tr>
<tr>
<td></td>
<td>You Can Run but You Can’t Read: Preventing Disclosure Exploits in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Executable Code</td>
<td>CCS’14</td>
</tr>
<tr>
<td></td>
<td>CAIN: Silently Breaking ASLR in the Cloud</td>
<td>WOOT’15</td>
</tr>
<tr>
<td></td>
<td>Weaponizing Middleboxes for TCP Reflected Amplification</td>
<td>SEC’21</td>
</tr>
<tr>
<td>Raghuveer</td>
<td>Counterfeit Object-oriented Programming: On the Difficulty of</td>
<td>NDSS’17</td>
</tr>
<tr>
<td></td>
<td>Preventing Code Reuse Attacks in C++ Applications</td>
<td>S&amp;P’15</td>
</tr>
<tr>
<td></td>
<td>MARX: Uncovering Class Hierarchies in C++ Programs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oxymoron: Making Fine-Grained Memory Randomization Practical</td>
<td>SEC’14</td>
</tr>
<tr>
<td></td>
<td>by Allowing Code Sharing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAC it up: Towards Pointer Integrity using ARM Pointer Authentication</td>
<td>SEC’19</td>
</tr>
<tr>
<td>Jackson</td>
<td>HexType: Efficient Detection of Type Confusion Errors for C++</td>
<td>CCS’17</td>
</tr>
<tr>
<td></td>
<td>Smashing the Gadgets: Hindering Return-Oriented Programming</td>
<td>S&amp;P’12</td>
</tr>
<tr>
<td></td>
<td>In-place Code Randomization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Meltdown: Reading Kernel Memory from User Space</td>
<td>SEC’18</td>
</tr>
<tr>
<td></td>
<td>PointGuardTM: Protecting Pointers from Buffer Overflow Vulnerabilities</td>
<td>SEC’03</td>
</tr>
<tr>
<td>Kidus</td>
<td>Jump over ASLR: Attacking Branch Predictors to Bypass ASLR</td>
<td>MICRO’16</td>
</tr>
<tr>
<td></td>
<td>Shuffler: Fast and Deployable Continuous Code Re-randomization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RowHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-</td>
<td>HPCA’16</td>
</tr>
<tr>
<td></td>
<td>Accessed DRAM Rows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KASLR is Dead: Long Live KASLR – ESSoS’17</td>
<td></td>
</tr>
<tr>
<td>Sylvia</td>
<td>Type Casting Verification: Stopping an Emerging Attack Vector</td>
<td>SEC’15</td>
</tr>
<tr>
<td></td>
<td>Gadge Me If You Can: Secure and Efficient Ad-hoc Instruction-Level</td>
<td>ASIACCS’13</td>
</tr>
<tr>
<td></td>
<td>Randomization for x86 and ARM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SafeHidden: An Efficient and Secure Information Hiding Technique</td>
<td>SEC’19</td>
</tr>
<tr>
<td></td>
<td>Using Re-randomization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUFs: Myth, Fact or Busted? A Secure Evaluation of Physically</td>
<td>CHES’12</td>
</tr>
<tr>
<td></td>
<td>Unclonable Functions (PUFs) Cast in Silicon</td>
<td></td>
</tr>
</tbody>
</table>
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

**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

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

**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
Exposé trends and gaps
Change the minds of readers through persuasion

New findings based on observation and data gathering
Prototype, new systems, architectures, tools, toolkits, techniques
New knowledge about how we carry 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

- **PUFs**
  - **defense**
  - **KAISER, Shuffler**
  - **empirical contribution**
  - **lava:** Large-scale Automated Vulnerability Addition

- **K(H)eaps**
  - **attack**
  - **BAP: A binary analysis platform**
  - **artifact contribution**
  - **Remote Timing Attacks are Practical**

- **empirical contribution**
  - **methodological contribution**
  - **theoretical contribution**
  - **dataset contribution**

- **artifact contribution**
  - **methodological contribution**
  - **theoretical contribution**
  - **survey contribution**

- **empirical contribution**
  - **methodological contribution**
  - **theoretical contribution**
  - **opinion contribution**

**New and useful corpuses with analysis of its characteristics**

**Expose trends and gaps**

**Change the minds of readers through persuasion**

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

**Examples of the Seven Types of Contribution**

- **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?
  - Why ’Correct’ Computers Can Leak Your Information
Evaluation Criteria for the Seven Types of Contribution

- **PUFs**
  - KAISER, Shuffler
  - An Empirical Study of the Reliability of UNIX Utilities

- **K(H)eaps**
  - BAP: A binary analysis platform
  - Recognizing Functions in Binary with Neural Networks

- **Terra**
  - Terra: A Virtual Machine-Based Platform for Trusted Computing

- **LAVA**
  - LAVA: Large-scale Automated Vulnerability Addition

- **SoK paper**
  - KAISER, Shuffler
  - An Empirical Study of the Reliability of UNIX Utilities

- **Why Is Cybersecurity Not a Human-Scale Problem Anymore?**

  - Why ‘Correct’ Computers Can Leak Your Information

**Evaluation Criteria:**

- **Defense**
  - How important are the findings? How sound are the methods?
  - What they make possible and how they do so

- **Attack**
  - Utility, reproducibility, reliability, and validity of the new method
  - Novelty, soundness, and power to describe, predict, and explain

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

- **Opinion**
  - Evaluated on the strength of their argument
You Can Run but You Can’t Read: Preventing Disclosure Exploits in Executable Code

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 inadvertent reads of executable memory. Our prototype is available for both Linux and Windows and introduces only a small performance overhead.
Counterfeit Object-oriented Programs

On the Difficulty of Preventing Code Reuse Attacks

Felix Schuster*, Thomas Tendyke*, Christopher Liebchen1, Lucas Davi1, Andreas Kruegel, Dario Perino2, Andreas Zeller, Guido Vezzetti

1Horst Grotz Institut (HGI)
2CA

Ruhr-Universität Bochum, Germany
Technische Universität München

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.

Vgadget 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 vgadgets in a loop and pass a field of the initial object to each as argument. | see Figure 6
W-ARG-G | Write to data 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-64-G | Load argument register x, x, or x with value (x64 only). | see Figure 4

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

Category | Scheme | Realization | Effective against COOP ?
---|---|---|---
Generic CFI | Original CFI + shadow call stack [3] | Binary + debug symbols | 
CCFR [58] | Binary | ✔
O-CFI [54] | Binary | ✔
SW-HW Co-Design [15] | Source code + specialized hardware | ✔
Windows 10 Tech. Preview CFG | Source code | ✔
LLVM IPC [52] | Source code | ✔
C++ aware CFI | —various— [5], [29], [52] | Source code | ✔
T-VIP [24] | Binary | ✔
VTInst [57] | Binary | ✔
vGuard [41] | Binary | ✔
C++-aware CFI | —various— [14], [40], [56] | CPU debugging/performance monitoring features | ✔
HEDROP [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 | ✔
Xor [7] | Binary / source code | ✔
Memory safety | —various— [4]-[6], [13], [36], [45] | Mostly source code | ✔
CPS/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.

artifact contribution

survey contribution
Case Study – III

Smashing the Gadgets: Hiding Programming Using In-Place

Vasilis Pappas, Michalis Polychronakis, and Angelos N. Stavrou
Columbia University
\{vpappas, mikepol, angelo\}@cs.columbia.edu

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.

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 Randomization Practical by Allowing Coexistence of Congruent Code Blocks

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

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 forgoing the enormous memory savings of shared libraries.

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 \( \binom{n}{n-p} \) combinations. The adversary’s probability of correctly guessing one address is hence the reciprocal \( \frac{1}{\binom{n}{n-p}} \). 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:

\[
P_r[Ad_{layout}] = \frac{(n-p)!}{n!} = \frac{(2^{19} - 2^{5})!}{2^{19}!} = 2^{-608}
\]

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

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 SIMMON and IBM CP-40 produced in Jan 1967. Classified into two types in Robert P. Goldberg’s PhD thesis

- Prevent modification of kernel code, injection of unauthorized code, execution of userspace code in the privileged mode
  - IMA
  - kGuard: Lightweight kernel protection against return-to-user attacks
  - Many other previous works

- Prevent DMA Attacks, Control Flow Attacks, etc.
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, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, Wonbo Shen

1 Samsung KNOX R&D, Samsung Research America
2 Department of Computer Science, NC State University

[a.azab, peng ning, j1.shah, rf.bhutkar, g.ganesh, jia.maj]@samsung.com
(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 mail-