Using Unikernels to Enhance the Attack-Resistance of Spire, a Network-Attack-Resilient Intrusion-Tolerant SCADA for the Power Grid

Brad Whitehead
Mike Boby

CS3551 – Advanced Topics in Distributed Systems
Class Project

Original Project Goals

Convert Spire to self-contained unikernels and demonstrate that:

- They continue to operate correctly and
- They exhibit the increased performance and reduced resource utilization characteristics of unikernel technology
- If possible, demonstrate the increased compromise resistance of the Unikernel-based Spire.

Revised Project Goals

Convert Spire to self-contained unikernels and demonstrate that:

- They continue to operate correctly and
- They exhibit the increased performance and reduced resource utilization characteristics of unikernel technology
- Spire compromise resistance can be increased by combining polymorphic executables (Multicompiler) with unikernel
- If possible, demonstrate the increased compromise resistance of the unikernel-based Spire (both GCC- and multicompiler-based)

[Red = Changes from original]

Why?

- While randomization is an excellent approach, security and compromise resistance may be further enhanced by discarding the use of an operating system and converting the executables into unikernels, isolated from other applications through hardware-enforced virtual machine technology
- Not only will this increase the compromise resistance, it will significantly enhance portability, performance in the areas of initialization (“bootup”) and throughput, as well as decreasing resource utilization (memory)
- Considerable thought and effort has been applied to the problem of making the Spire code resistant to attack and compromise, includes using MultiCompiler to create polymorphic versions
- Combining the use of a unikernel approach with the Multicompiler may have a synergestic effect, as comprehensive security often involves a “layered” approach

What Do We Need For Spire?

- Run a single application per server (no scheduler required)
- Run as a single user
- Uses a known set of hardware drivers
- Uses 1 or 2 communications protocols
- Needs security (from unauthorized access - “hacking”)
- Needs reliability
- Nice to-have: Speed (low startup and processing latency)

Using A Unikernel Library Operating System: Keeping Only The OS Functions That Spire Requires

- Enhanced security:
  - Greatly reduced attack surface (99.92% reduction based on code size)
- Userland applications not required (removes 410 million lines of potentially flawed code)
  - No shell (/bin/sh)
- No ability to run malicious or hacking tools on the same VM
- Function calls instead of system calls (more secure)
- No time consuming context switches
- No re-configuration attacks

Using The Multicompiler: "Mixin' it up!"

- Security enhancement, already tested and used by the Spire researchers
- Randomizes the memory location of functions
  - Similar to Address Space Layout Randomization, but more effective and finer-grained
  - ASLR randomly sets the base address of each library in the process
  - Discovering the memory location of one function in ASLR completely defeats the protection of that library
  - Multicompiler changes the order of the functions in each library with each compilation
- Randomly inserts NOPs
  - Breaks Return-Oriented Programming (ROP) [a malicious attack technique] "gadgets"

Potential Challenges

1) Use of Unix socket interprocess communications (IPC)

2) Multicompiler and Unikernel build systems are not designed for sequential "pipelining"

Potential Challenge #1
Use Of Unix Socket Interprocess Communications (IPC)

- A review of Spire code identified the use of Unix socket IPC
- Unix IPC involves two or more processes within the same "system", sharing memory (in some fashion)
- Unikernels involve only one process per "system" – no memory to share
- **Mitigation Approach** - Spire code might need to be modified, converting Unix socket IPC to network-based TCP/UDP IPC

Potential Challenge #2
Multicompiler And Unikernel Build Conflicts

- Multicompiler and Unikernel build systems are not designed for sequential “pipelining”
- Both toolsets operate on source code and produce an executable
- **Mitigation Approach** – Research latest unikernel build systems and attempt to identify a system that can operate with an executable instead of source code

Actual Challenges and Chosen Mitigations

- Use of Unix socket interprocess communications (IPC)
  - Spire code modified to convert Unix socket IPC to network-based UDP sockets
- Multicompiler and Unikernel build systems are not designed for sequential “pipelining”
  - Identified two unikernel build systems that operate with executables instead of source code
    - Hermitux – (Systems Software Research Group – Virginia Polytechnic Institute)
    - NanoVMs/OPS – commercial unikernel system
- Dynamically linked Spire executables are not compatible with chosen unikernel (Hermitux)
  - Spire author (Dr. Babay) developed required modifications to build Spire as statically linked executables
  - Benefit – static executables are less vulnerable to injection attacks

Additional Challenge and Mitigation

- Lost network access to the development/test server due to error during virtual network setup
  - Original development server had sufficient compute resources to support the required 8 full-operating system VMs
  - Establishment and testing of normal Spire configuration was completed prior to loss of connectivity
- Re-established development and testing on smaller server
  - While (probably) not having sufficient compute resources to support 8 full operating system VMs, it is sufficient for 26 unikernel VMs (because of their significantly reduced resource requirements)

Project Steps

1) Familiarization with the Spire system (obtain and compile the code, and run the supplied benchmarks
2) Research available unikernel libraries and select the most appropriate one
3) Select an appropriate paper on unikernels and security to present in class
4) Compile the Spire executables into unikernels
5) Iteratively, make necessary code changes
6) Test and benchmark Spires unikernels using the included benchmark suite
7) Investigate the compromise resistance of the Spire unikernels (this step is dependent on the availability of any existing compromise/penetration tests or test tools)
8) Document the project
9) Prepare and deliver project presentation for class

Project Plan

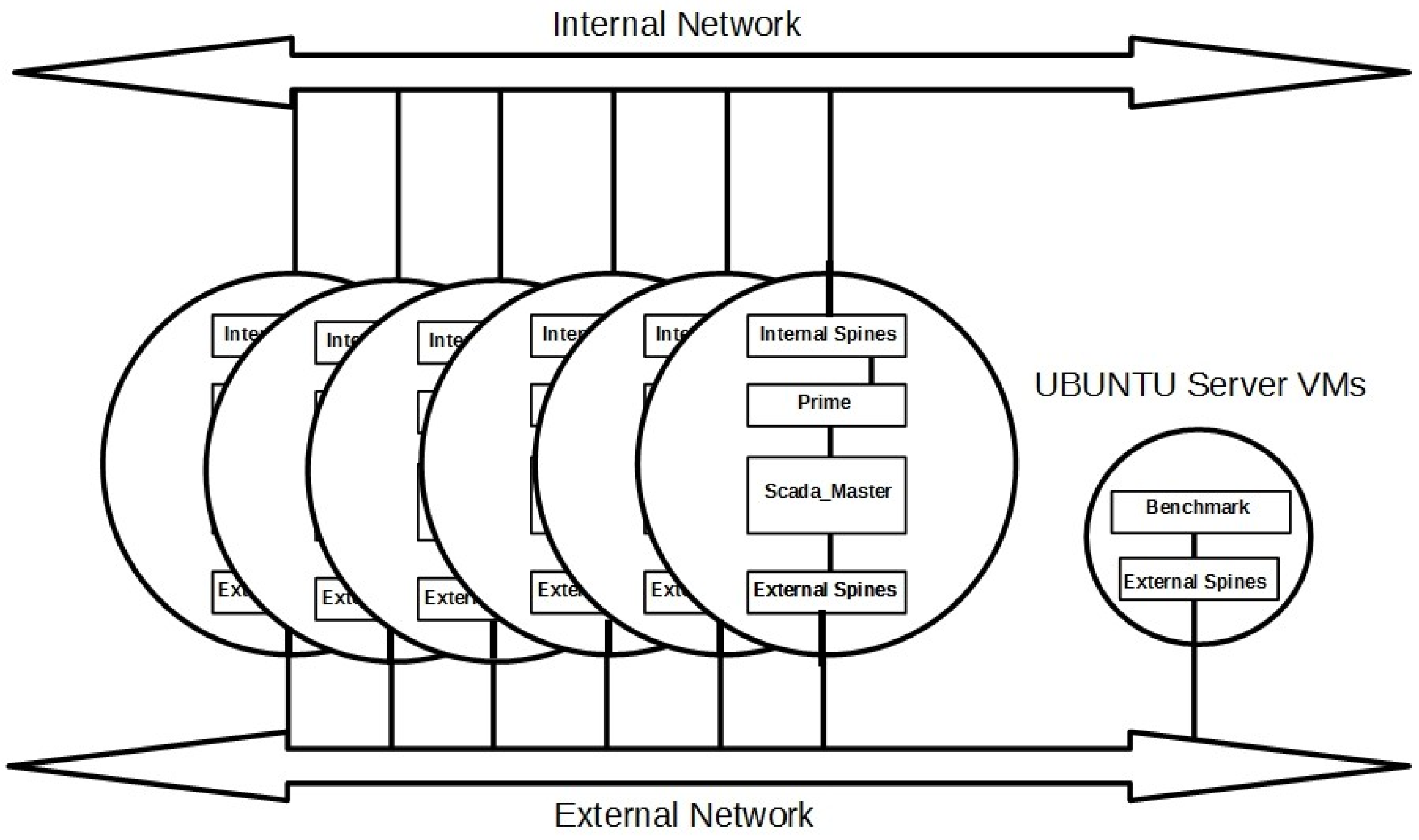
- **Class Week #11 (March 22 – 28)**
  - **Present Status Checkpoint to Class**
  - **Compile Spire system using GCC**
  - **Install KVM and create the VM configuration files for the ~~6~~ 8 VMs we need to test Spire**
  - **Run the Spire benchmark, using recommended configuration**
- **Class Week #12 (March 29 – April 4)**
  - **Compile Hermitux build tools**
  - **Link Hermitux and GCC-compiled Spire executables**
  - **Create the VM configuration files for the 26 VMs we need to test unikernel-Spire**
  - Re-run the benchmark, using these Hermitux unikernel executables
- **Class Week #13 (April 5 – 11)**
  - **Compile Multicompiler**
  - **Re-compile Spire using the Multicompiler**
  - Re-run the Spire benchmark, using the Multicompiler-compiled executables
- **Class Week #14 (April 12 – 18)**
  - **Link Hermitux and Multicompiler-compiled Spire executables into unikernel executables**
  - Re-run the benchmark using the Hermitux & Multicompiler unikernels
- **Class Week #15 (April 19 – 23)**
  - Present Finding to Class

[**Green**, **bolded** items have been completed]

Evaluation

- **Original Evaluation Server**
  - Dell PowerEdge T620
    - two 6-core Xeon processors, for 24 hyperthreads
    - 2.1 GHz
    - 32GB ECC memory
  - Create 2 virtual networks (Internal and External network)
  - Created 7 Ubuntu 18.04 Server VMs
    - 6 VMs ran the combination of scada_master, internal_spines, external_spines, and prime
    - 1 VM ran the combination of benchmark and external_spines
  - Established memory and boot-time metrics

Evaluation – Baseline Configuration

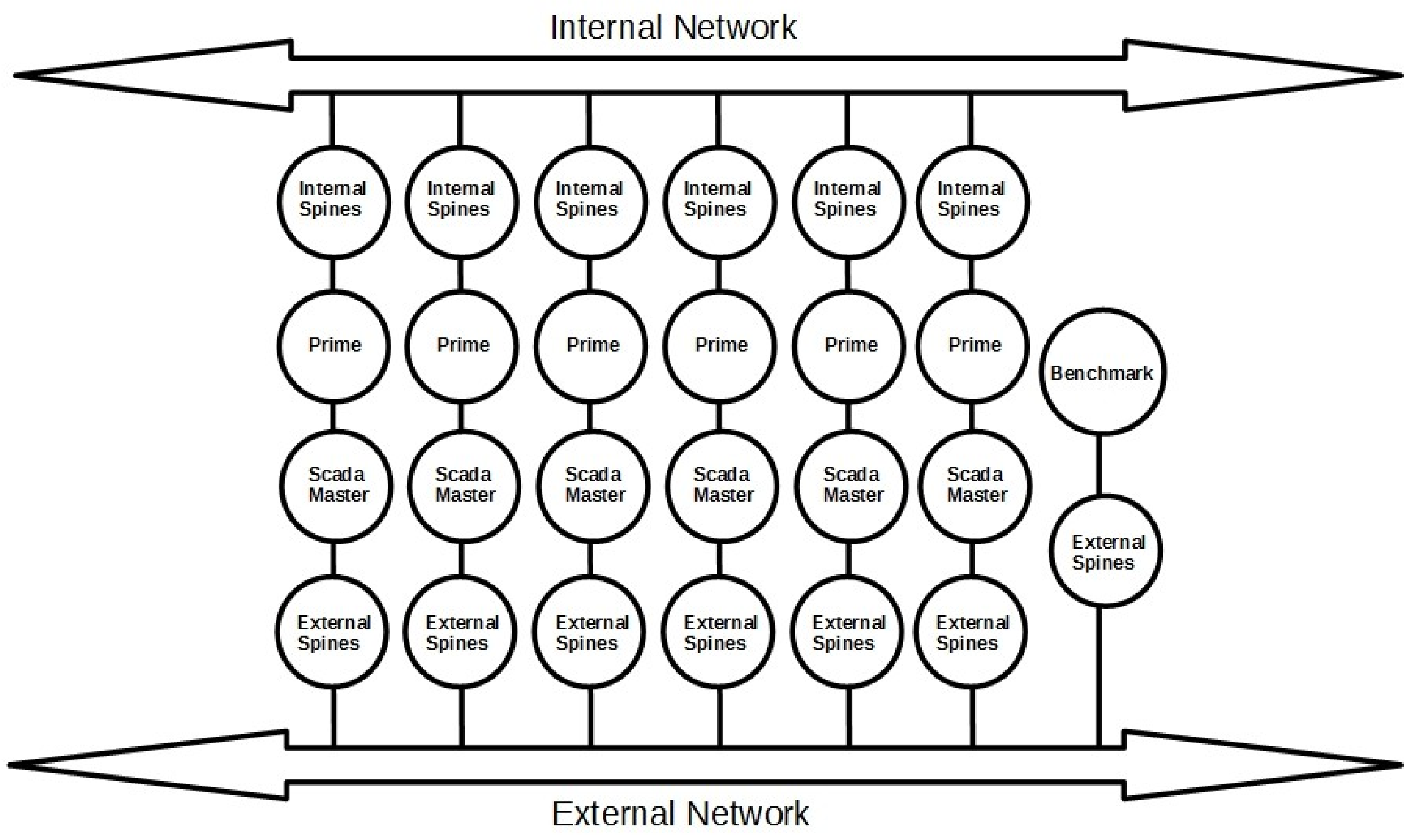


7 Ubuntu Virtual Machines – 35 GB of disk images

Evaluation

- **<u>Unikernel Evaluation Server</u>**
  - Dell Optiplex
    - One 4 core Intel Core i7 CPU with 8 hyperthreads
    - 2.2 GHz
    - 12GB DDR memory
  - Create a virtual network (Spire Internal and External networks defined by ports)
  - Compiled Spire to static executables and ran with Hermitux unikernel library (26 VMs)
  - Compiled Spire with Multicompiler to static executables and ran with Hermitux unikernel library
  - Conducted memory and boot time metrics

Evaluation – Unikernel Configuration



26 Unikernel Virtual Machines – 208 MB of disk images

Review Against Project Goals - #1

Convert Spire to self-contained unikernels and demonstrate that:

1) **They continue to operate correctly**

   The project partially met this goal. While there was an unresolved error with the Hermitux unikernel library when using dynamically linked spire executables, statically linked executables booted and operated without errors*

* (One code change was required to change a ‘gettimeofday’ call to use a NULL rather than a TZ struct)

Review Against Project Goals - #2

Convert Spire to self-contained unikernels and demonstrate that:

2) **They exhibit the increased performance and reduced resource utilization characteristics of unikernel technology**

The project met this goal:

Baseline Spire (Ubuntu 18.04 Server) VM	
Startup: 12 sec	Memory: 163 MB
Alpine Linux VM*	
Startup: 9 sec	Memory: 48 MB
Unikernel Spire (Hermitux library) VM	
Startup: 10 millisec	Memory: 9 MB

*(From literature, not measured as part of this project)

Review Against Project Goals - #3

Convert Spire to self-contained unikernels and demonstrate that:

3) **Spire compromise resistance can be increased by combining polymorphic executables (Multicompiler) with unikernel**

The project met this goal. By using a unikernel library that interoperates with executables, it was possible to sequentially 1) compile the Spire source code with the multicompiler; and 2) load the compiled executable with the Hermitux unikernel library into a lightweight VM

Review Against Project Goals - #4

Convert Spire to self-contained unikernels and demonstrate that:

4) **If possible, demonstrate the increased compromise resistance of the unikernel-based Spire (both GCC- and multicompiler-based)**

   The project was unable to determine this through empirical evaluation due to a lack of time. A logical evaluation of using the unikernel approach for indicates that the "machine" (virtual or bare-metal) that Spire operates on has 99.9% fewer instructions. The extension of this reasoning is that the removal of these instructions enhances Spire's compromise resistance

Lessons Learned

- Unikernel technology is still maturing
  - Most unikernels still require compiling source code and library OS together to produce a VM
  - Hermitux and OPS/nanovms using Linux ABI to allow unaltered executables to be combined with a library OS to produce the VM
  - Hermitux implements ~150 of close to 400 Linux susyem calls – 95% real-world coverage
- Unikernel technology is very promising – extremely fast boot times, much smaller memory
- It is possible to combine two compromise resistance technologies into a single executable (multicompiler randomization and unikernel surface reduction

Lessons Refreshed

- Things take longer than expected
- Always have a Plan B (and possibly C)
  - Spare servers
  - Redundant network access
  - Software incompatibilities
- Be flexible and goal-oriented

Areas For Future Research

- Complete revisions to Spire to permit TCP IPCs
  - Increase Spire deployment flexibility
  - Support Spire unikernels
- Investigate OPS/nanovms and other unikernel libraries that operate with executables
- Investigate self-randomization (selfrando – UCI) as an alternative or in conjunction with multicompiler
- Investigate the combination of self-randomization and a compile-time unikernel library

Prior Research

- While there are a number of publications on the unikernel concept and its applicability to security, since the seminal paper in 2013, only one paper was found that specifically addressed the use of unikernels in a SCADA environment:
  - Sakic E. et al. (2018) VirtuWind – An SDN- and NFV-Based Architecture for Softwarized Industrial Networks. In: German R., Hielscher KS., Krieger U. (eds) Measurement, Modelling and Evaluation of Computing Systems. MMB 2018. Lecture Notes in Computer Science, vol 10740. Springer, Cham

  In this paper, unikernels were selected not for their security properties but rather for their fast instantiation and low memory requirements
- There are two other papers that mention the possibility of using unikernels in industrial networks, but both authors felt that the unikernel orchestration systems were not mature enough
- Both papers chose to use containers instead. Containers have a number of well known security issues. Based on this project, we believe that unikernels are sufficiently mature and that enhanced compromise resistance for Spire is achievable.

References

Spire

- A. Babay, T. Tantillo, T. Aron, M. Platania and Y. Amir, "Network-Attack-Resilient Intrusion-Tolerant SCADA for the Power Grid," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg City, 2018, pp. 255-266.
- Spire: Intrusion-Tolerant SCADA for the Power Grid (http://www.dsn.jhu.edu/spire/)

Multicompiler

- M. Franz, “E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism”, NSPW’10, September 21–23, 2010, Concord, Massachusetts
- Optimize for Security (https://immunant.com/blog/2018/09/multicompiler/)
- https://github.com/securesystemslab/multicompiler

Hermitux

- P. Olivier, D. Chiba, S. Lankes, C. Min, B. Ravindran, “A Binary-Compatible Unikernel”, VEE ’19, April 13–14, 2019, Providence, RI, USA
- Hermitux (https://ssrg-vt.github.io/hermitux/)

BACKUP SLIDES

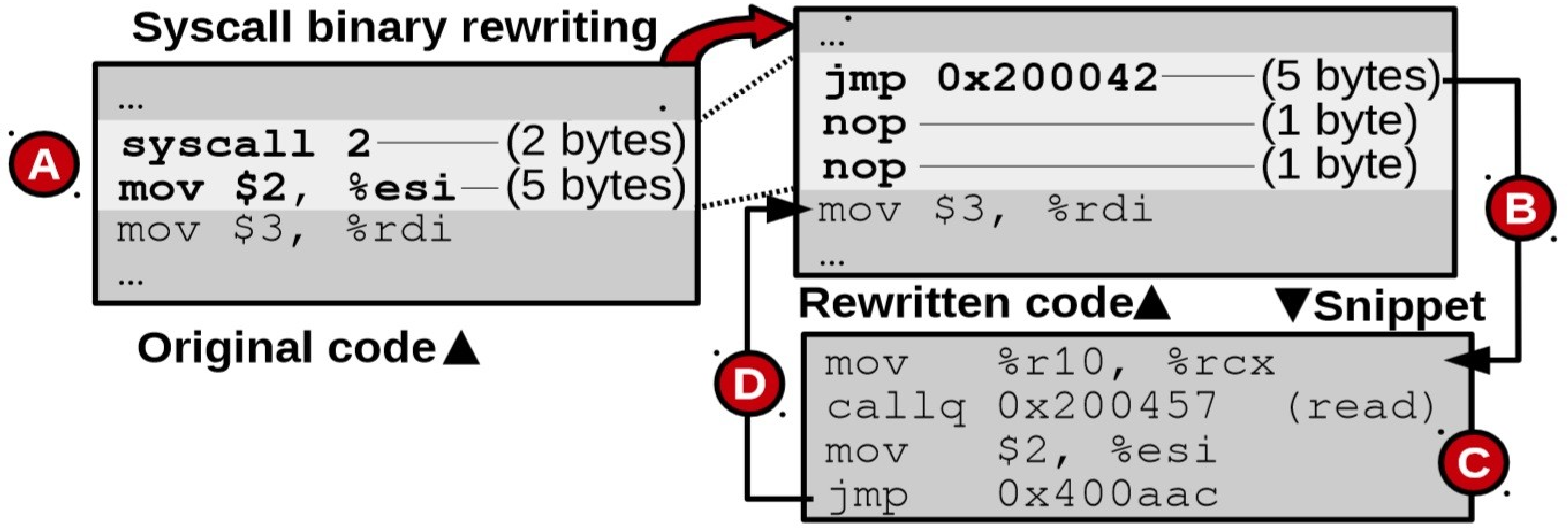
This slide intentionally left blank

- IBM

How Hermitux Works With Executables

- Existing executable is already linked to runtime library
  - Hermitux provides a special runtime library
  - Where the regular runtime library has system calls, Hermitux has regular functions that do the same work as the system call
- When the program explicitly makes a system call, Hermitux does a binary re-write as part of the loading process
  - Intel ‘syscall’ instruction takes fewer bytes than a ‘jmp’ instruction
  - Replacing a ‘syscall’ with a ‘jmp’ overwrites the next two bytes after the syscall – BAD :-(
  - Can’t push the instructions down – All subsequent jmp instructions would point to the wrong place – BAD :-(
  - You could change every subsequent jmp target, but it would take a lot of time and you would miss dynamically calculated targets – BAD :-(
  - Hermitux re-arranges instructions – CLEVER :-)

Clever Re-Write Example



```
...
syscall 2        (2 bytes)
mov $2, %esi     (5 bytes)
mov $3, %rdi
...
```

```
...
jmp 0x200042     (5 bytes)
nop              (1 byte)
nop              (1 byte)
mov $3, %rdi
...
```

```
mov    %r10, %rcx
callq  0x200457   (read)
mov    $2, %esi
jmp    0x400aac
```

System call rewriting example.