Dynamically Enforcing Security Properties Using Metadata

2019September24, RISC-V Security Standing Committee

Greg Sullivan, Dover Microsystems, gregs@dovermicrosystems.com

DOVER MICROSYSTEMS

© 2019 Dover Microsystems, Inc. All rights reserved.

OUTLINE

- The wonders of metadata. Probably obvious
  - Framing common security approaches as metadata-based dynamic checking
- Important classes:
  - Confidentiality, Integrity (Information Flow Control)
  - Compartmentalization
  - Memory Safety
  - Control Flow Integrity
- Defining Security by
  Policy Definition and Composition
- Using CWEs and CVEs for metrics
- Hardware implications
- Interaction between metadata and side channel mitigations

Not a sales talk

Will refer to ongoing research funded under DARPA SSITH program, specifically team led by Draper

© 2019 Dover Microsystems, Inc.

WHAT IF…

We have arbitrary metadata for every word in memory?

- Does this word contain confidential data?
  - Has the current value been influenced by confidential data?
- Trusted data?
  - If came from trusted source, has current value been modified by untrusted data?
- Is the word an instruction?
  - If so, is it in an executable region?
  - If a jump instruction, where is it allowed to jump?
- A pointer?
  - If a pointer, where is it allowed to point (dereference)?

© 2019 Dover Microsystems, Inc.

DYNAMIC INFORMATION FLOW TRACKING
Confidentiality

- If an ADD, and either of the operand values is labeled Confidential, then result is labeled Confidential
- If a STORE, and value-register value is labeled Confidential, then result is labeled Confidential
- If a LOAD, and memory value is labeled Confidential, then value-register value is labelled Confidential
- If a STORE, and target location is labeled Public, VIOLATION
- If return value from trusted encryption routine (simplified), Confidential label is removed

© 2019 Dover Microsystems, Inc.

FORMALIZATION

Work by Pierce, Tolmach, et al on SSITH HOPE project

- Create a language for policy definition, with formal semantics
- Define higher level security goal (in a formal language)
- Prove that policy definition implies security goal
- But wait! Relying on declassification via trusted encryption code requires code integrity

Extracted from generic taint policy at https://github.com/draperlaboratory/hope-policies

```
^ arithGrp(env == _, op1 == [+Taint],
    op2 == _ -> env = env, res = op1)

^ arithGrp(env == _, op1 == _,
    op2 == [+Taint] -> env = env, res = op2)

^ loadGrp(mem == [+Taint], addr == _,
    env == _ -> env = env, res = mem )

^ storeGrp(mem == _, addr == _, val == [+Taint],
    env == _ -> env = env, mem = val)
```

Guarantees of one policy may Assume properties provided by another policy.

Policy Composition

© 2019 Dover Microsystems, Inc.

HEAP SAFETY MICROPOLICY

Example of a CoreGuard micropolicy in action

SECURITY MICROPOLICY

GOAL

Provide spatial and temporal safety

Stop buffer overflow attacks

METHOD

Give each pointer a unique "color" or tag

Color memory slots with this tag on allocation

Recolor on Free

MICROPOLICY

```
storeGrp(addr == color, mem == color -> mem = color)
```

Metadata on pointer to: x y z

Metadata on address to STORE

Memory stays the same

x y z

```
x = malloc(2);

x[0] = 0x09;

y = malloc(5);

y[3] = 0x04;

z = malloc(3);

z[1] = 0x01;

x[2] = 0xbad;
//FAIL
```

Payload data in host core
0x09
0x08
0x07
0x06
0x05
0x04
0x03
0x02
0x01
0x00
Metadata in CoreGuard

© 2019 Dover Microsystems, Inc.

COMPARE TO

- LLVM Address Sanitizer
  - Entirely in software; too expensive for production
- Arm Memory Tagging Extensions (MTE)
  - Max 16 colors
  - Feature-specific instructions
  - Protection of metadata???
- Sparc ADI
  - Similar to Arm MTE
- CHERI
  - Feature-specific instructions

© 2019 Dover Microsystems, Inc.

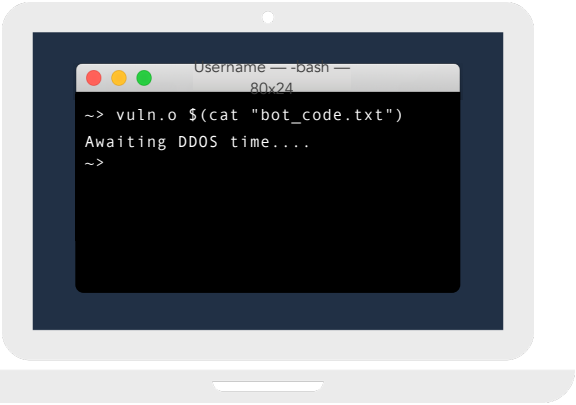
STACK SMASH BUFFER OVERFLOW

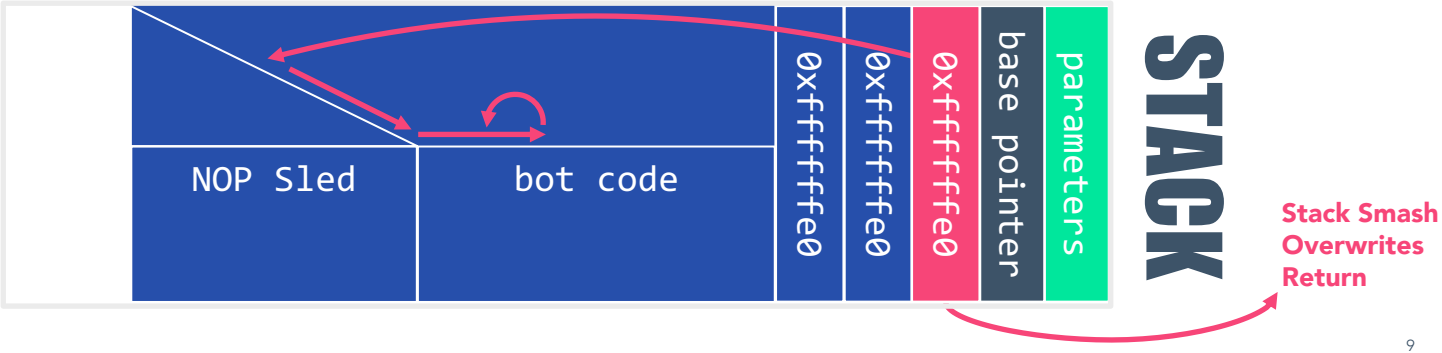
How a stack smash buffer overflow attack works

© 2019 Dover Microsystems, Inc.

STACK SMASH BUFFER OVERFLOW

How a stack smash buffer overflow attack works

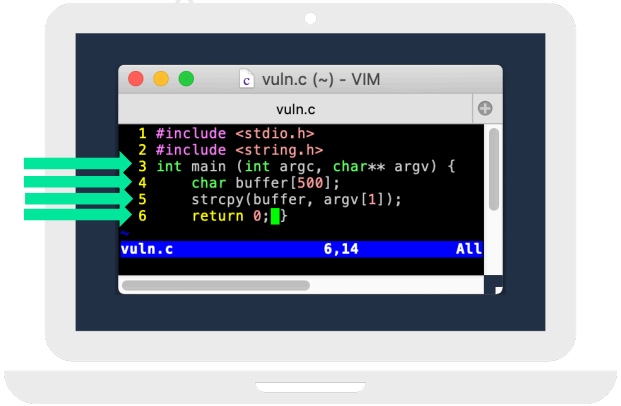


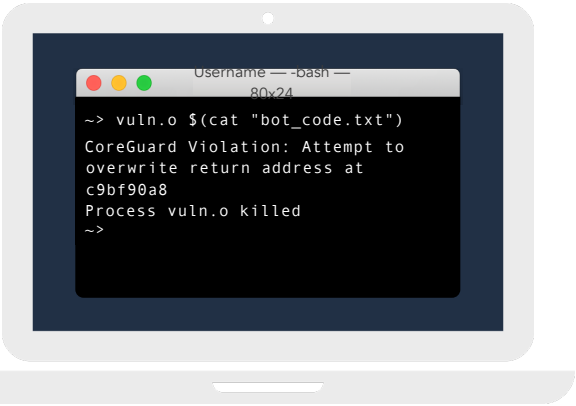


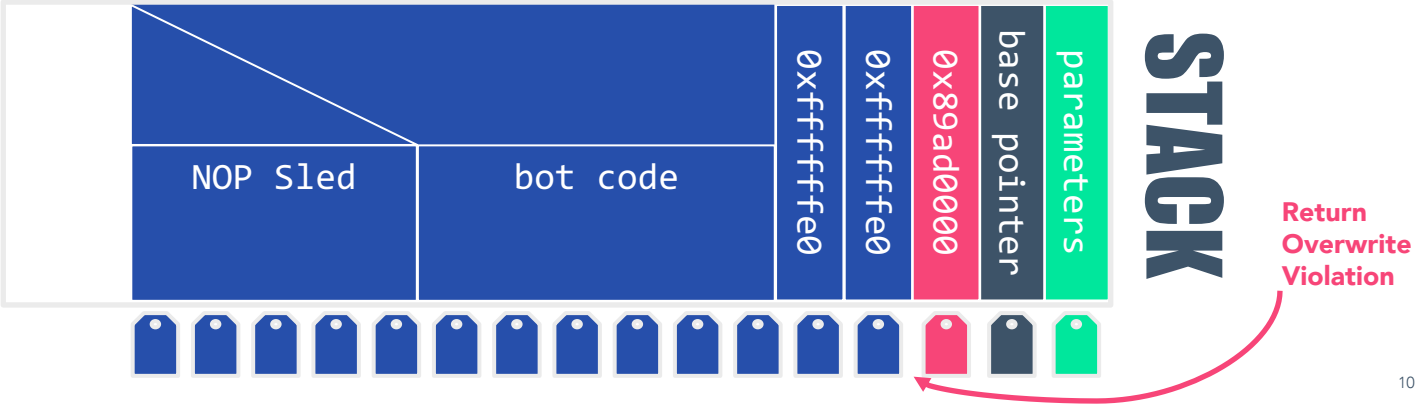
© 2019 Dover Microsystems, Inc.

STACK SMASH BUFFER OVERFLOW

How CoreGuard stops a stack smash buffer overflow attack







© 2019 Dover Microsystems, Inc.

COMPARE TO

- Arm Pointer Authentication (PAC)
  - Additional instructions, both in ISA and in binary

© 2019 Dover Microsystems, Inc.

READ-WRITE-EXECUTE MICROPOLICY

SECURITY MICROPOLICY

GOAL

Apply Read/Write/Execute permissions to code and data

Protect memory writes and execution

METHOD

Tag code and data with permissions

Initial labeling of ELF sections as R/W/X comes from compile-time toolchain

Only runtime loader instructions have ability to tag data as executable

MICROPOLICY

*(Load, G, —, —, —, M) ▸ ((M ~ G:*R*), —, M)*

*(Store, G, —, —, —, M) ▸ ((M ~ G:*W*), —, M)*

*(—, G, C, —, —, —) ▸ ((C ~ G:**X), —, —)*

*(—, G, —, R1, R2, —) ▸ (((R1 & R2) ~ G:R**), —, (R1 & R2))*

(—, Sys, NewG, R1, —, —) ▸ (TRUE, R1, —)

Current instruction word must be executable

Check that targets of STORE are writable

Check that sources of LOAD are readable

		G1:RW-	G1:R--	G1:RW-
G1:-W-	Add	X1,	X2,	X3

```
G1:-W-  jal  X1 // where X1=10000
        …

        10000:
G1:R--  .asciiz "Hello World\n"
        // Policy Violation when
        attempting to execute "Hell"
```

Expressed as (Opcode Type, PcTag, InstrTag, R1Tag, R2Tag, MemTag) ▸ (Allow Condition, New PcTag, New Rd/Mem Tag)

© 2019 Dover Microsystems, Inc.

COMPARE TO

- Classic NX bit (no-execute)
  - Under control of OS (so vulnerable to OS exploits)

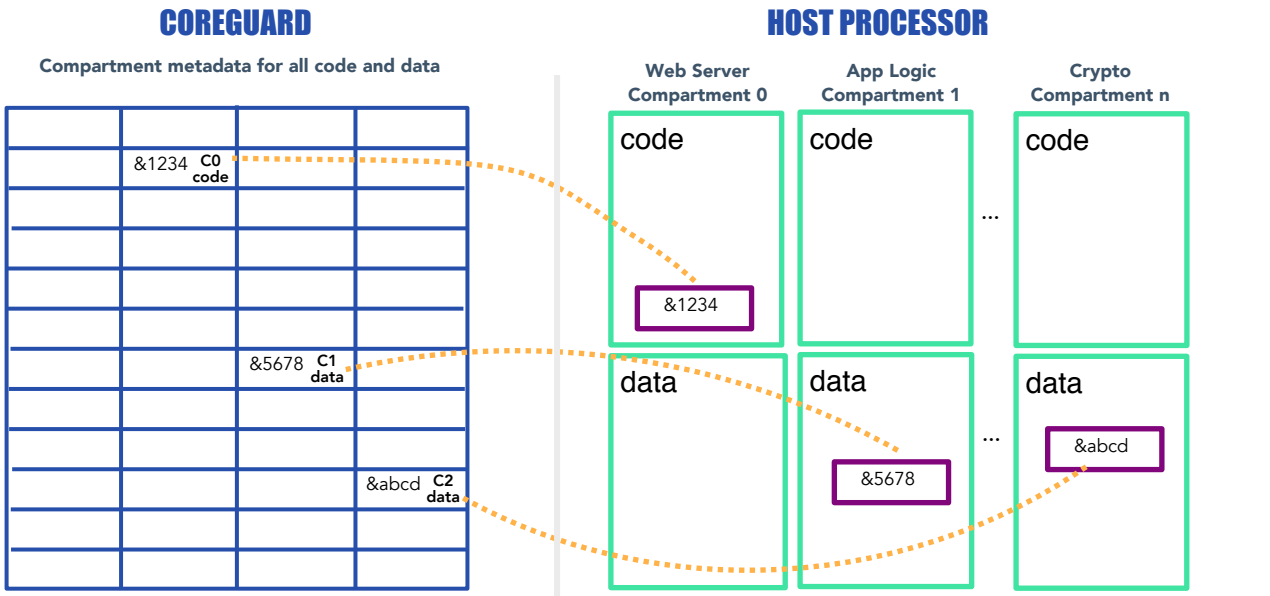
© 2019 Dover Microsystems, Inc.

COMPARTMENTALIZATION (C18N) MICROPOLICY

SECURITY MICROPOLICY

1 AT LOAD TIME

CoreGuard tags memory (code & data) with compartment metadata



2 AS THE APPLICATION EXECUTES

Metadata on the Program Counter (PC) tracks the current compartment

Compartment Metadata

Program Counter
Generalized to 'env' in policy language

© 2019 Dover Microsystems, Inc.

C18N FOR UNLIMITED COMPARTMENTS

SECURITY MICROPOLICY

3

AS THE APPLICATION EXECUTES

C18N micropolicy rules check whether an instruction is allowed

COREGUARD

```
jumpGrp (code == C, env == C -> env = C )
branchGrp (code == C, env == C -> env = C )
callGrp (code == C, env == C -> env = C )
```

Code can jump around within the same compartment.

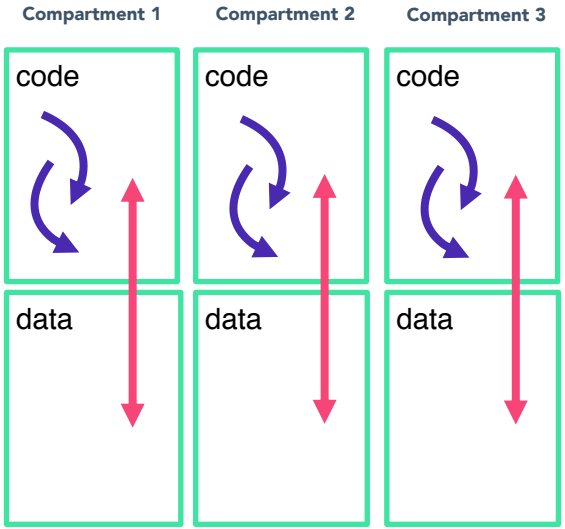
```
loadGrp(code == C, env == C, mem == C -> env = C, mem = C)

storeGrp(code == C, env == C, mem == C -> env = C, mem = C)
```

Code within a compartment can freely access data within that same compartment.

HOST PROCESSOR

JUMP | BRANCH | CALL



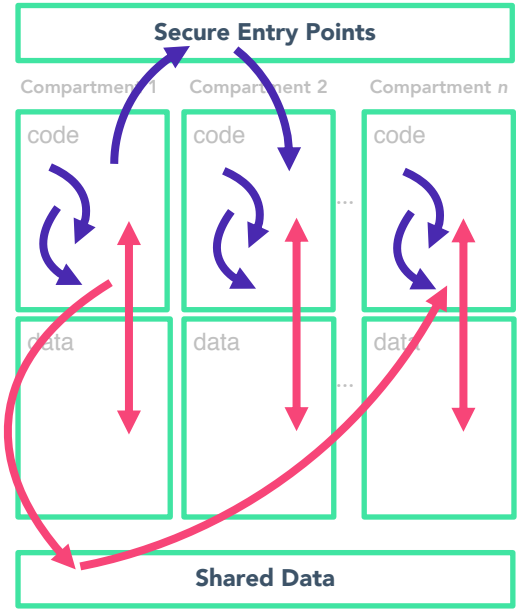
LOAD | STORE

© 2019 Dover Microsystems, Inc.

C18N: MULTIPLE APPROACHES

SECURITY MICROPOLICY

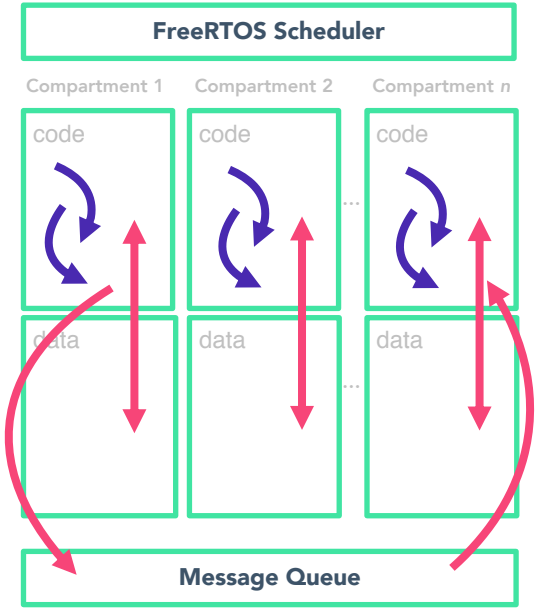
IMPLEMENT TRUSTZONE MODEL



Create an unlimited number of compartments

Support an unlimited number of entry points

ENFORCE BUILT-IN OS C18N

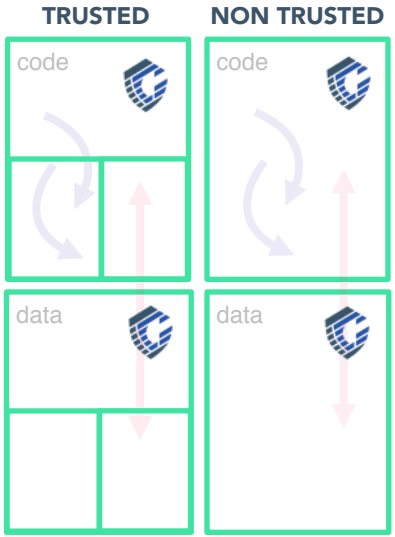


Support multiple C18N models

Task-based message passing (e.g., FreeRTOS)

Add security to the existing native model

EXTEND TRUSTZONE



Create sub-compartments for finer granularity

Protect code *within compartments* from attack

© 2019 Dover Microsystems, Inc.

COMPARE TO

- Arm TrustZone
  - Feature-specific instructions
  - Heavyweight context switches
  - Limited number of compartments
- HexFive MultiZone
  - Lighter weight context switches than TZ
  - More compartments
- Intel SGX
  - Feature-specific instructions, context switch

A note about:

Principle of Least Privilege, Context Switching, & Software Fault Isolation

- PoLP
  - Want more "compartments"
- SFI: Wahbe et al. '93
  - Motto: trade small increase in cost of every call/load/store in exchange for much faster context switches

μScope: Vasilakis et al. (Upenn)

© 2019 Dover Microsystems, Inc.

Feature versus General approach
Some thoughts for the RISC-V community

Feature approach

- Wait until someone invents a hardware mechanism that
  1. Mitigates a sufficient number of attacks,
  2. Has sufficiently low performance, power, area (PPA) costs
- Add instructions to take advantage of this new feature
- Update toolchains (compilers) and documentation to use/reflect new feature
- Design, Verify, Synthesize, Sell, etc.

General approach

- General mechanism for:
  - Secure metadata storage
  - Fast mapping from Host-accessible data to Metadata
  - Rule cache
- Build policies for everything
- If a policy is (a) always used, and (b) expensive, consider adding HW acceleration
- Do not change ISA; no new instructions

© 2019 Dover Microsystems, Inc.

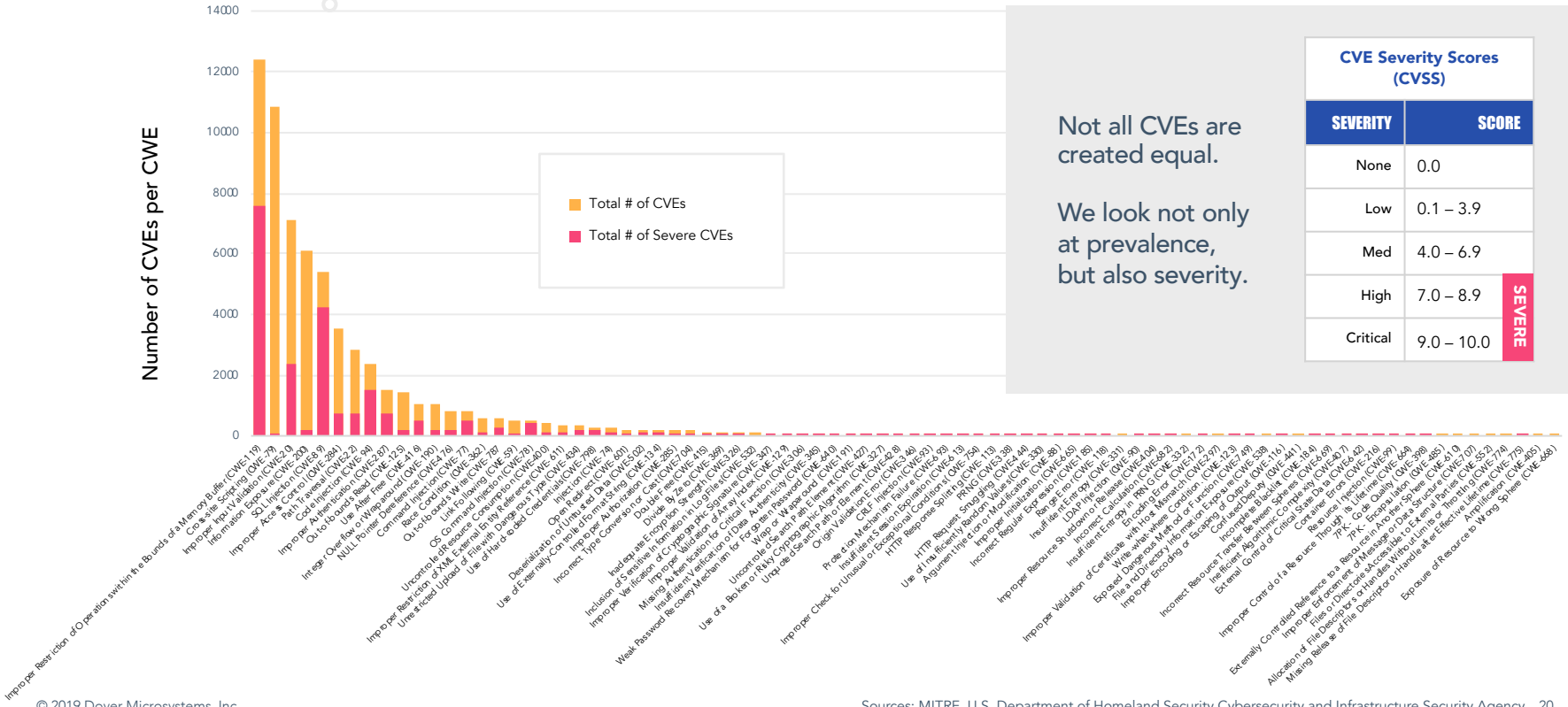
Measuring Security
Two approaches

- The policy definitions ARE the definition of security
  - No numbers. Just “this is allowed; this not”
  - Precise
- CWEs and CVEs

© 2019 Dover Microsystems, Inc.

UNIVERSE OF COMMON VULNERABILITIES

Over 63,000 vulnerabilities (CVEs), across 84 categories (CWEs)



Not all CVEs are created equal.

We look not only at prevalence, but also severity.

CVE Severity Scores (CVSS)

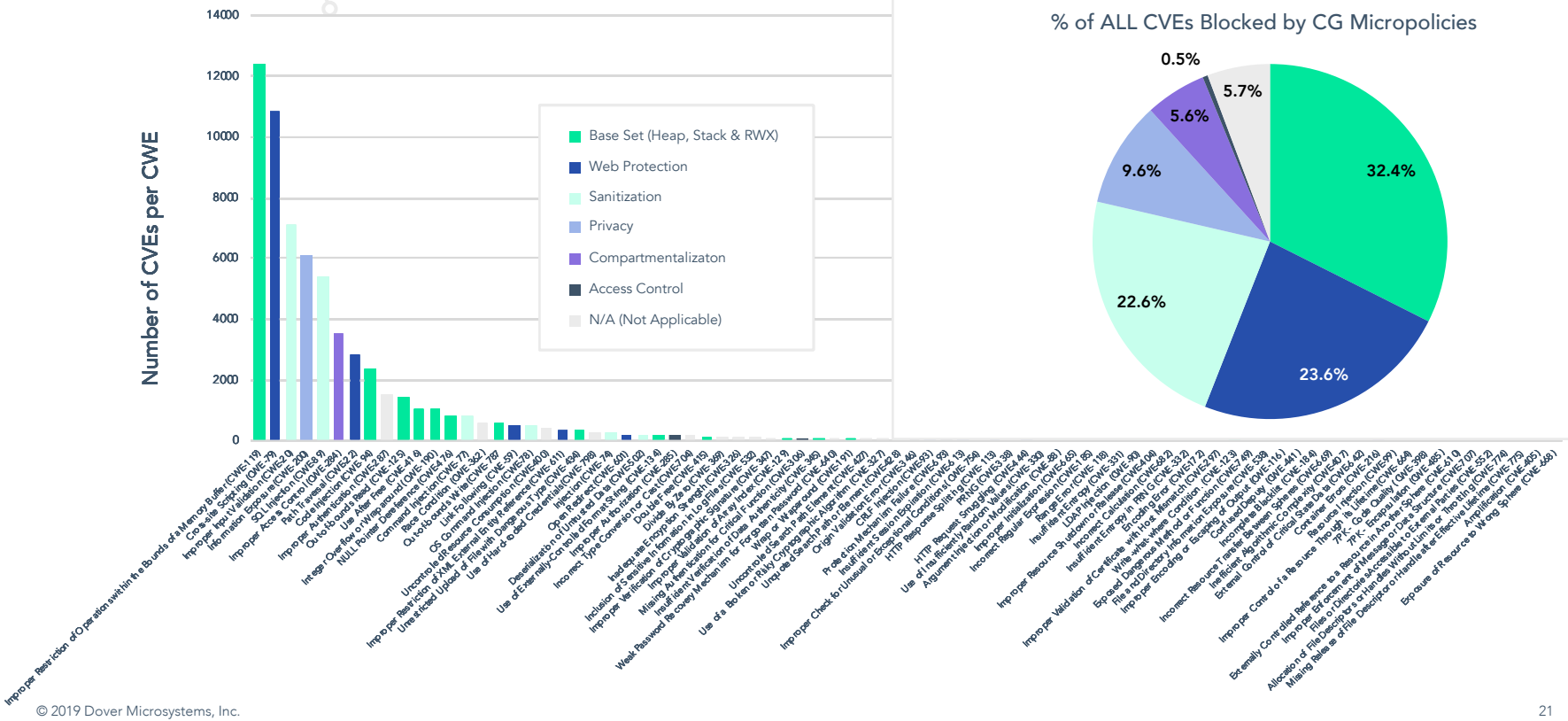
SEVERITY	SCORE	
None	0.0	
Low	0.1 – 3.9	
Med	4.0 – 6.9	
High	7.0 – 8.9	SEVERE
Critical	9.0 – 10.0	SEVERE

© 2019 Dover Microsystems, Inc.

Sources: MITRE, U.S. Department of Homeland Security Cybersecurity and Infrastructure Security Agency

COREGUARD PROTECTION BREAKDOWN

CWE coverage based on CoreGuard micropolicies



© 2019 Dover Microsystems, Inc.

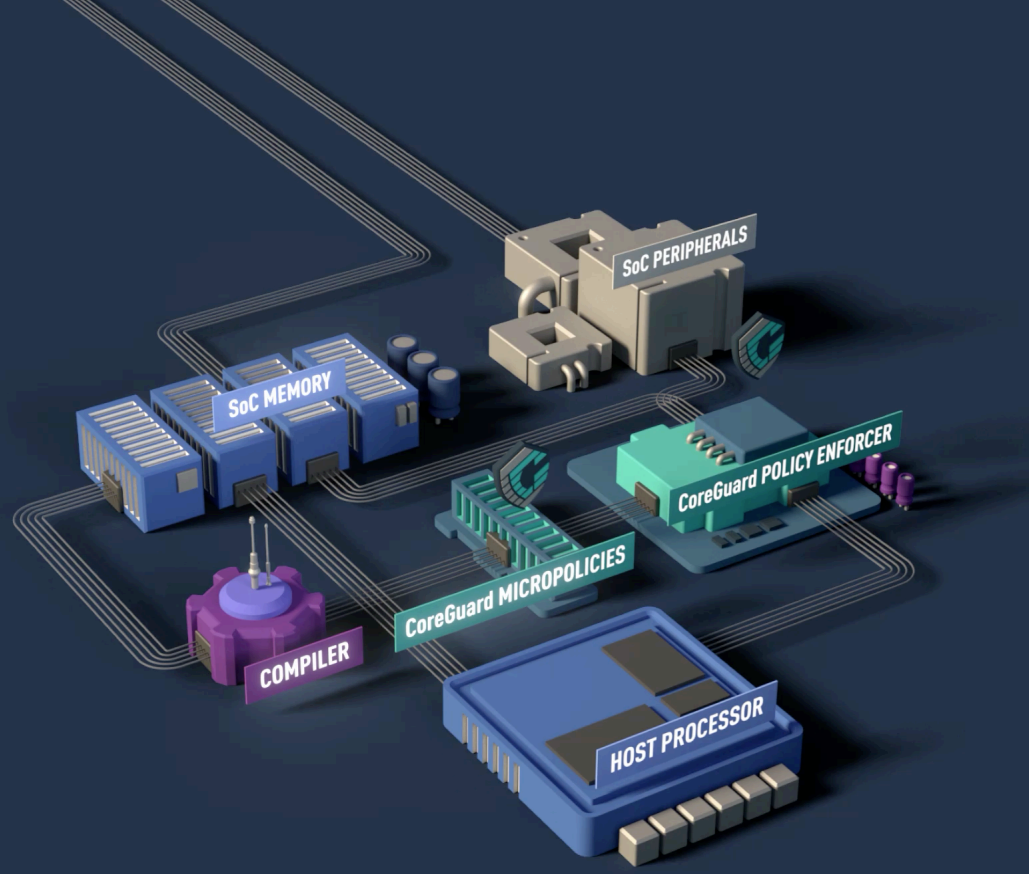
MAKING METADATA-BASED SECURITY PERFORMANT

- Arbitrary per-word metadata, subjected to arbitrary predicates, and updated according to arbitrary rules, at every instruction
  - Can represent pretty much any security policy
    - Compare to inline reference monitors, Valgrind, source or binary rewriting, etc.
  - What's not to like?

- What is the overhead?
  - "It depends"

© 2019 Dover Microsystems, Inc.

HOW DOES COREGUARD WORK?



© 2019 Dover Microsystems, Inc.

THANK YOU

DOVER
MICROSYSTEMS

gregs@dovermicrosystems.com
www.dovermicrosystems.com