SUMMARY OF INVENTION

Thus, there is a need for a flexible security architecture that can be quickly adapted to this ever-changing landscape. It would be desirable to have such an architecture provide support for software-defined metadata processing with minimal overhead. It is desirable that such an architecture is extensible to generally support and enforce any number and type of policies without placing a visible, hard bound on the number of bits allocated to metadata. Metadata may be propagated during execution to enforce policies and catch violation of such policies such as, for example, by malicious code or malware attacks.

In accordance with one aspect of the techniques described herein is a method of processing instructions comprising: receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction; determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, using a rule cache miss handler that is a hardware-implemented rule cache miss handler, in the metadata processing domain comprising: determining whether execution of the current instruction is allowed; responsive to determining the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction; writing to a control status register; and responsive to writing to the register, inserting the new rule into the rule cache. First metadata used to select the rule for the current instruction may be stored in a first portion of a plurality of control status registers used by the metadata processing, and wherein the first portion of the plurality of control status registers may be used to communicate a plurality of

metadata processing. The current instruction may be one of multiple instructions stored in a single word of memory associated with a single metadata tag, and said single metadata tag may be associated with the multiple instructions included in the single word. The plurality of control status registers may include a subinstruction control status register indicating which of the multiple instructions stored in the single word is the current instruction. The single metadata tag may be a first pointer to a first memory location including a different metadata tag for each of the multiple instructions in the single word. At least a first metadata tag stored in the first memory location for a first instruction of the multiple instructions may include a second pointer to a second memory location including metadata tag information for the first instruction. The metadata tag information for the first instruction may include a complex structure. The complex structure may include at least one scalar data field and at least one pointer field to a third memory location.

In accordance with another aspect of techniques herein is a non-transitory computer readable medium comprising code thereon that, when executed, perform a method of processing instructions comprising: receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction; determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, using a rule cache miss handler that is a hardware-implemented rule cache miss handler, in the metadata processing domain comprising: determining whether execution of the current instruction is allowed; responsive to determining the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction; writing to a control status register; and responsive to writing to the register, inserting the new rule into the rule cache.

In accordance with another aspect of techniques herein is a system comprising a processor; and a memory comprising code stored thereon that, when executed by the processor, performs a method of processing instructions comprising: receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction; determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, using a rule cache miss handler that is a hardware-implemented rule cache miss handler, in the metadata processing domain comprising: determining whether execution of the current instruction is allowed; responsive to determining the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction; writing to a control status register; and responsive to writing to the register, inserting the new rule into the rule cache. The processor may be a pipeline processor in a reduced instruction set computing architecture.

In accordance with another aspect of techniques herein is a method of processing instructions comprising: receiving a current instruction for metadata processing performed in a metadata processing domain that is isolated from a code execution domain including the current instruction; and determining, by the metadata processing domain in connection with metadata for the current instruction, whether to allow execution of the current instruction in accordance with a set of one or more policies, wherein the current instruction accesses a first location of a stack frame of a first routine, wherein the current instruction and locations of the stack frame have associated metadata tags, and the set of one or more policies includes a stack protection policy that provides stack protection and prevents improper access to stack storage

What is Claimed is

1. A method of processing instructions comprising:

receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction;

determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and

responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, *using a rule cache miss handler that is a hardware-implemented rule cache miss handler*, in the metadata processing domain comprising:

determining whether execution of the current instruction is allowed;

responsive to determining the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction;

writing to a *control status* register; and

responsive to writing to the register, inserting the new rule into the rule cache.

2. The method of Claim 1, wherein first metadata used to select the rule for the current instruction is stored in a first portion of a plurality of control status registers used by the metadata processing, and wherein the first portion of the plurality of control status registers are used to communicate a plurality of metadata tags for the current instruction to the metadata processing domain, wherein said plurality of metadata tags are used as data in the metadata processing domain.

3. The method of Claim 2, wherein the register is a first control status register of the plurality of control status registers used by the metadata processing, and wherein the first portion of the plurality of control status registers are used to communicate the plurality of metadata tags from the metadata processing domain to the rule cache.

4. The method of Claim 3, wherein the plurality of metadata tags are for the current instruction.

5. The method of Claim 4, wherein the new rule is inserted into the rule cache responsive to writing another metadata tag to the first control status register, wherein the another metadata tag is placed on a result of the current instruction, said result being any of a destination register or a memory location.

6. The method of Claim 2, wherein the plurality of control status registers includes any one or more of:

a bootstrap tag control status register including an initial metadata tag from which all other generated metadata tags are derived;

a default tag control status register specifying a default metadata tag;

a public untrusted control status register specifying a public untrusted metadata tag used to tag instruction and data classified as public and untrusted;

an opgroup value control status register including data written to a table including information on opgroups and care information for different opcodes;

an opgroup address control status register specifying a location in the table to which data of the opgroup value control status register is written; and

a pumpflush control status register wherein a write to the pumpflush control status register triggers flushing of the rule cache.

7. The method of Claim 2, wherein the plurality of control status registers includes a tag mode control status register denoting a current mode of metadata processing.

8. The method of Claim 7, wherein the tag mode control status register indicates when metadata processing is disengaged whereby rules of one or more defined policies are not enforced by metadata processing.

9. The method of Claim 8, wherein the tag mode control status register is set to one of a defined set of allowed states to denote the current mode of metadata processing, the allowed states including: an off state, a state whereby metadata processing writes a default tag on all results, and a state indicating that metadata processing is engaged and operational when instructions are executed in the code domain at one or more specified privilege levels.

10. The method of Claim 9, wherein the rule cache miss processing is performed in a first of the defined set of allowed states where metadata processing is disengaged.

11. The method of Claim 9, wherein the allowed states include a first state indicating that metadata processing is engaged only when instructions execute in the code domain at a user privilege level; a second state indicating that metadata processing is engaged only when instructions execute in the code domain at a user or supervisor privilege level; a third state indicating that metadata processing is engaged only when instructions execute in the code domain at a user, supervisor, or hypervisor privilege level; and a fourth state indicating that metadata processing is engaged when instructions execute in the code domain at a user, supervisor, hypervisor, or machine privilege level

12. The method of Claim 7, wherein whether the metadata processing is engaged or disengaged is determined in accordance with a current tag mode of the tag mode control status register in combination with a current privilege level of code executing in the code domain, wherein rules of one or more defined policies are not enforced when metadata processing is disengaged and wherein the rules are enforced when metadata processing is engaged.

13. The method of Claim 2 wherein a table includes information that maps an opcode of an instruction set to a corresponding opgroup and bit vector information, the opgroup denoting a group of associated opcodes treated similarly by the metadata processing domain, the bit vector information denoting whether particular inputs and outputs with respect to the metadata processing domain are used in connection with processing the opcode, wherein the table is indexed using a first portion of opcode bits less than a maximum number of allowable opcode bits, the maximum number denoting an upper bound on a number of bits of an opcode of the instruction set.

14. The method of Claim 13, wherein the first portion of the plurality of control status registers includes an extended opcode control status register including additional opcode bits, if any, for the current instruction, wherein the current instruction is included in the instruction set having variable length opcodes, wherein each opcode of the instruction set optionally includes the additional opcode bits and the extended opcode control status register includes the additional opcode bits, if any, for the current instruction.

15. The method of Claim 14, wherein, for each opcode mapped using the table there is a result bit vector corresponding to said each opcode, the result bit vector denoting what portion, if any, of the additional opcode bits in the extended opcode control status register are used with said each opcode for metadata processing.

16. The method of Claim 2, wherein the current instruction is one of multiple instructions stored in a single word of memory associated with a single metadata tag, said single metadata tag associated with the multiple instructions included in the single word.

17. The method of Claim 16, wherein the plurality of control status registers includes a subinstruction control status register indicating which of the multiple instructions stored in the single word is the current instruction.

18. The method of Claim 17, wherein the single metadata tag is a first pointer to a first memory location including a different metadata tag for each of the multiple instructions in the single word.

19. The method of Claim 18, wherein at least a first metadata tag stored in the first memory location for a first instruction of the multiple instructions includes a second pointer to a second memory location including metadata tag information for the first instruction.

20. The method of Claim 19, wherein the metadata tag information for the first instruction includes a complex structure, the complex structure comprising at least one scalar data field and at least one pointer field to a third memory location.

21. A non-transitory computer readable medium comprising code thereon that, when executed, perform a method of processing instructions comprising:

receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction;

determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and

responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, *using a rule cache miss handler that is a hardware-implemented rule cache miss handler,* in the metadata processing domain comprising:

determining whether execution of the current instruction is allowed; responsive to determining the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction;

writing to a *control status* register; and

responsive to writing to the register, inserting the new rule into the rule cache.

22. A system comprising:

a processor; and

a memory comprising code stored thereon that, when executed by the processor, performs a method of processing instructions comprising:

receiving, for metadata processing, a current instruction with an associated metadata tag, said metadata processing being performed in a metadata processing domain isolated from a code execution domain including the current instruction;

determining, in the metadata processing domain and in accordance with the metadata tag and the current instruction, whether a rule exists in a rule cache for the current instruction, said rule cache including rules on metadata used by said metadata processing to define allowed operations; and

responsive to determining no rule exists in the rule cache for the current instruction, performing rule cache miss processing, *using a rule cache miss handler that is a hardware-implemented rule cache miss handler*, in the metadata processing domain comprising:

determining whether execution of the current instruction is allowed;

responsive to determining whether execution of the current instruction is allowed to be executed in the code execution domain, generating a new rule for the current instruction;

writing to a *control status* register; and

responsive to writing to the register, inserting the new rule into the rule cache.

23. The system of Claim 22, wherein the processor is a pipeline processor in a reduced instruction set computing architecture.