PAPER I – Minos

Summary

Control flow attacks exploiting vulnerabilities such as stack and heap overflows are very common and a major concern. Minos is **a micro-architectural mechanism** that tries to **prevent control flow attacks by protecting the integrity of the control data**. The key contribution of this paper is to show how the control data attacks can be prevented by tracking the information flow dynamically with the mechanism implemented in hardware. This is the **first paper to support DIFT in hardware** (Suh et. al. did a similar work in parallel).

An **integrity bit is added for every 32-bit word** of memory and general-purpose registers. This bit is set to either low or high depending upon the trust level for the corresponding data. **Biba's low-water-mark integrity policy** is implemented on each data word to facilitate integrity bit propagation as per the data flow. **An exception is raised when a low integrity data is used for the purpose of control flow transfer**. This allows Minos to cover low level control flow hijacking but doesn't provide protection against high level attacks such as SQLi and XSS.

Minos **requires changes in the processor architecture** introducing some memory overheads and adds complexity in the processor core. **Modifications are also required to the operating system** to allow this system to work. Authors have implemented Minos on an emulator for Linux and Windows, and have shown that the **scheme is capable of protecting against real world attacks** with small overhead.

Best Point

This work shows how hardware based information flow tracking can be implemented to provide security features. The authors have **demonstrated how security can be provided at hardware level with negligible performance overhead (as faced by software based schemes**). This **provides motivation and guidance to the researchers opening a new area of research in the domain of micro-architecture and security.** This is one of the first papers to implement hardware based information flow tracking system, addressing the issues involved and showing its usefulness to prevent control flow attacks. This **novelty is the best point** of this paper as it inspires new research work.

Worst Point

The mechanism proposed is **not at all flexible**. This is a hardware-only feature and thus, **doesn't allow having dynamic control on the propagation rules and tainted bits based on the policy requirements for certain application**. There are **much overhead and modifications** required in the processor architecture. This could have been accepted in case the proposed technique was generic and allowed its use for different purposes. If designed once, it can't be changed to adapt the future findings (such as the issues with JIT found by the authors) or evolving threats. Thus, it may result in a lot of false positives or false negatives. If the customers don't want to use this feature (in case they have other better protection techniques against control flow attacks), it is a pure wastage. Allowing multiple policies and making it general would have helped to attract more users as they could have used it for different purposes. So**, this much overhead and efforts are not acceptable for such a stringent scheme**.

Comparison with Related Schemes Studied

This is related to **the concept and the schemes based on dynamic information flow tracking** studied in the class. This paper is much **similar to the paper by Suh. et. al**. Both were developed independently and in parallel. Minos doesn't try to reduce the memory overhead as done by the later using multi-granularity security tags. Minos covers only control-data attacks while the later covers both control and non-control data attacks. Both can't protect against high level semantic attacks.

The **FlexiTaint scheme is also based on the similar concept** of dynamic information flow tracking. The authors have mentioned Minos in the related work and thus, probably got inspired from this work. FlexiTaint allows the policies and propagation rules to be defined programmatically and also provides support for multiple tags. FlexiTaint also avoids modification in pipeline stages by performing taint related operations at back-end. So, it provides more flexibility and less overhead as compared to Minos.

Another related paper is by **Clause et. al.** as it also uses similar taint propagation technique to provide protection against illegal memory accesses. **Capability based systems** studied in the class also try to provide some protection and access control. But, the main objective is not specifically to defend against control data attacks. **MMP and MemTracker can also help to protect against control flow attacks** up to some level as we can mark return locations as non-writable. **XOM** also provides security but focuses on confidentiality mainly.

Improvement

I would have tried to work in the area of **making this mechanism as flexible and programmable as possible**. This can be achieved by **implementing a hardware-software based technique**. It would allow my scheme to be generic so that it **can be used for different purposes** by defining policies and propagation rules in software. This **avoids making the scheme to be too stringent** and thus, resulting a **flexible approach which can be useful for different applications/policies of information flow tracking.** It won't let the modifications and overhead introduced to be a waste and would allow accepting any future findings/fixes. This should justify and make the changes in the processor architecture acceptable.

PAPER II – Raksha

Summary

The inflexibility in hardware-only DIFT schemes, overhead in software-only DIFT schemes and the rising interest of hackers in high level semantic vulnerabilities call for the need of a flexible DIFT solution which can be customized depending upon the context**. Raksha is a flexible hardware-software based DIFT architecture** to provide **security against low level attacks such as buffer overflows as well as high level attacks such as SQLi and XSS.** Though the flexibility provided is somewhat limited, the key contribution is to demonstrate **how a flexible approach with general hardware mechanism (getting the**

performance and fine grain protection benefits) and more customizable software control (allowing it to be flexible and robust) can be deployed to protect against a wide variety of attacks. Raksha allows the **security policies to be defined in software**, supports **enabling of multiple policies at a time** to facilitate protection against coordinated/concurrent attacks and finally, implements and protects the **exception handlers at user level** to facilitate software based analysis of an exception without incurring the cost of border crossing. It also allows the application of DIFT to protect OS code.

The proposed scheme associates **4 extra bits with each word to allow maximum four schemes** to be active at a time for which all storage locations are extended. **A pair of Tag Propagation Register and Tag Check Register is used for each policy to allow software control**. **One of the policies is used to protect the exception handling code** implemented at the same level of the vulnerable (protected) code. Registers, instructions and kernel code are added / modified to allow this. All the **pipeline stages are changed** to allow tag propagation and check at primitive operation level. Finally, the authors also present **some security and performance evaluation** on an FPGA-based prototype along with the lessons learnt from the experiments.

Best Point

The best point is the **flexibility and extensibility provided by this scheme** to implement DIFT for providing security. This is **the first paper to demonstrate how we can get the best of both hardware and software benefits to protect against low level as well as high level semantic attacks**. As the high level attacks are getting more popular, I think this **provides a really good direction to the researchers showing how software controlled hardware mechanism can help to defend** against such attacks. In addition, the flexibility provided **allows reducing the false positives and false negatives** by covering the emerging threats and the corner cases. It also allows defining **policies based on the context** and **enabling multiple policies** facilitating wider coverage of attacks.

Worst Point

The worst point is the **overhead** because of **the modifications required in the processor architecture and the operating system** as well as the due to **memory storage**. The proposed scheme requires all the **storage locations to be extended by tag bits** (4 bits per word). All the **ISA instructions are extended** to allow tag propagation and verification. It also **needs extra registers and instructions** to be added and the processor is supposed to have **trusted and non-trusted modes**. Moreover, **all the stages in the processor pipeline are modified** to allow tag propagation and check. Although one of the tag bits is used only for security handler protection, they mention the addition of this bit for all the storage locations. **OS needs to be modified** to allow proper functionality of exception handling and tag propagation. Finally, they **don't implement any intelligent tag management technique** and thus, doesn't try to reduce the overhead. So, all this requires a lot of efforts and modifications and is not efficient also.

Comparison with the Related Schemes Studied

The **basic concept is of dynamic information flow tracking** (which we studied in the class) to provide protection against attacks. The **general model is similar to the one proposed by Suh et. al. (mentioned**

by Raksha in related work) that we studied in the class. But, this is an improvement in terms of flexibility provided by the scheme as it introduces software controlled policies. Suh. et. al. used multi-granularity security tags to reduce memory overhead while Raksha doesn't do anything in this regard. Raksha can protect against high level attacks while the other scheme can't.

This scheme is also **similar to FlexiTaint (which mentions Raksha in references)** that we studied in the class. Both try to improve the flexibility and programmability of the DIFT system and both can detect high level attacks and support multiple concurrent attacks. FlexiTaint is more flexible as compared to Raksha as the later allows only to control if the tag should be propagated or not, and if yes, whether OR or AND operation should be performed on the operands' tags. FlexiTaint allows defining and using any set of propagation rules. Raksha stores the taint data along with the normal one and thus, requires many modifications. FlexiTaint also avoid modification of pipeline stages while Raksha doesn't.

Another related paper is by **Clause et. al.** as it also uses similar taint propagation technique to provide protection against illegal memory accesses. **MMP and MemTracker** can also help to protect against control flow attacks up to some level as we can mark return locations as non-writable. **XOM** also provides security but focuses on confidentiality mainly.

Improvement

I would have tried to extend this work **to reduce the overhead involved**. First task would have been to come up with **some efficient tag storage, manipulation and management scheme** so that we can have reduced memory overhead and also, can reduce the modifications required in the processor, instruction set and the operating system. Another area is to modify or extend this scheme in a way so that we don't have to make many changes in processor core. Proposed scheme needs each of the pipeline stages to be modified. It would be beneficial if we can avoid these changes as much as possible.

I would have chosen this direction as Raksha provides flexibility but with much overhead and modifications which makes its adoption for practical purposes difficult. Fixing this flaw would have made it a practical scheme.

Relation/Comparison between Minos and Raksha

Minos has been mentioned in the reference section of Raksha. The authors of Raksha acknowledge that Minos is one of the first papers to show how hardware based DIFT can be used to provide security and that it addresses the intricacies involved in addition and management of taint bits in modern processors.

So, we can say that **Minos is an inspiration for Raksha** as it **provides the general model on which Raksha has been designed. Raksha is surely an improvement** over Minos as it provides **flexibility by introducing the software control** to define the policies and propagation rules to be followed by the hardware mechanism. **Raksha also enables the protection against high level attacks** which was out of the scope for the authors of Minos. Raksha authors have also **explored more in the area of tag propagation and tainting/untainting** to reduce false positives and false negatives.