Paper Review – Bastion

Summary

Bastion is a **hardware-software architecture that protects security-critical components in an untrusted software stack** using extended hardware virtualization. The key contribution is to propose a system that can **provide both physical and software security to application modules** in presence untrusted operating system by **reducing the security perimeter to just the microprocessor chip** and assuming hardware virtualization support giving more control to application developers.

Bastion microprocessor **protects storage and runtime memory state of enhanced hypervisor** against both software and hardware attacks by providing both confidentiality and integrity. Hardware protected hypervisor **provides protection to an arbitrary number of trusted software modules** by providing a **secure compartmentalized execution environment** and storage area to each module. Hypervisor also handles **the secure invocation and preemption** of these modules. The domain is specified by associating **module id with the protected pages to enforce the access rules** with the help of shadow access control mechanism using Module state table. **Two new cryptographic engines and on-chip RNG** are added to provide integrity and confidentiality for the data. **Merkle tree** technique is used to provide integrity.

Instructions/hyper calls are added for launching the secure environment for the hypervisor/application modules. The **hypervisor and microprocessor need to be modified** to support this system. The authors have also implemented a **proof of concept on OpenSPARC platform** and also provide some analysis for complexity and performance impact.

Best Point

The best point is a **well thought and scalable design** from the security perspective of the application modules. This technique allows to c**ompletely bypass an untrusted and possibly vulnerable operating system** providing support for multiple mutually distrustful security domains (even within a VM) with the security solely based on hardware(and trusted hypervisor). This technique can be especially useful in cloud environment where VMs are provided to application developers as PaaS. It gives **power and flexibility to an application developer to manage the security of the critical modules in his application without relying on the underlying software stack as he can specify the access control rules in security segments**.

The authors have tried to cover all the intricacies from a hypervisor boot to the application level module invocation and preemption and have tried to provide security and scalability to arbitrary number of trusted software modules of arbitrary sizes in different trust domains. This well thought design and idea of giving security control at application level is the best point of this paper.

Worst Point

The **complexity and the overhead** introduced by this design are not practical. It requires a **new microprocessor with modified TLBs and caches, new registers, modified TLB lookup logic, internal**

microprocessor routines, new cryptographic engines and RNG. In addition, it requires a lot **of changes in the hypervisor** such as extension of shadow page table entries, extra step to TLB miss handling procedure, support for new **hypercalls**. This adds a lot of cost and complexity. The hypercalls are used frequently and thus, the **intermediation provided by hypervisor to provide this security adds a lot of performance overhead.**

Moreover, **Bastion still relies on the security of the modified hypervisor to accomplish security goals.** What if the deployed hypervisor is already compromised or the added complexity introduces some vulnerability which can be exploited in some way. As the system binds these enhanced microprocessor and hypervisor, it can compromise the security of the whole system and make it useless. This binding also makes the normal hypervisors incompatible and thus reduces the usability.

The **authors haven't demonstrated good practical implementation** of this and **the limited analysis** has been done with many assumptions which don't give a practical picture of added complexity.

<u>**Comparison with Related Schemes Studied**</u>

The paper is related to **the concept of hardware-enabled trusted computing** studied in the class. It tries to establish a sort of **chain of trust** as the processor authenticates the hypervisor and the hypervisor authenticates the application modules using the cryptographic hashes. It also uses the concept of hardware virtualization studied in the class.

This paper (published in 2010) is **related to the XOM paper** (published in 2000) discussed in the class which has been mentioned in the references. Both provide confidentiality and integrity of the data and also protect against hardware attacks. XOM is susceptible to memory replay attacks. XOM doesn't rely on hardware enabled virtualization.

The concept of **Merkle tree** studied in the class has been used in the paper to ensure integrity. The authors suggest the use of **caching of Merkle tree referencing the paper by Gassend et. al.** and also adopt some **ideas from the paper by Rogers et. al.** to provide protection for on-disk pages.

<u>**Improvement**</u>

The concept is really good and gives some control at the application level but the complexity added makes it impractical. The missing sufficient evaluation stops us from getting a clear probability of practical adaptation of this system. I would like to improve upon this work by **extending the design for multi-threaded and multi-core processor systems** which are commonly used in large data centers. In addition, I would like to implement **various techniques which can help to reduce the cost and performance overhead** such as caching of tree nodes, better hashing techniques and crypto engines.

Finally, I would like to do **a thorough evaluation of this improved prototype to get a better understanding of the overhead incurred** due to this security system. It will also motivate the practical adaptation of this system.