CONSTRUÇÃO DE ARTEFATOS CIBERNÉTICOS DE ALTA CAPACIDADE UTILIZANDO OS RECURSOS DE ALTO PRIVILÉGIO DE MICROPROCESSADORES

Building high capacity cyber artifacts using high privileged resources from microprocessors

William Augusto Rodrigues de Souza[1]

Resumo: Este trabalho discute como usar o modo de gerenciamento do sistema (SMM) dos processadores Intel a fim de estabelecer um conjunto de requerimentos e, a partir dele, propor uma arquitetura genérica que pode ser usada para construir ferramentas de segurança ou armas cibernéticas, as quais teriam o controle total do sistema computacional onde fossem instaladas. Uma prova de conceito foi construída e os experimentos realizados foram bem-sucedidos, demonstrando a viabilidade da proposta e a validade dos requerimentos.

Palavras-chave: Armas cibernéticas. Ferramentas de segurança. Modo de gerenciamento do sistema. Processadores Intel.

Abstract: This work discusses how to use the System Management Mode (SMM) on Intel processors to establish a set of requirements and from those requirements proposes a generic architecture that can be used to build high capacity security tools or cyber weapons, which can take full control of the host computational system. A proof-of-concept was built and successful experiments were conducted showing the feasibility of this proposal and the validity of the requirements.

Keywords: Cyber weapons. Security tools. System management mode. Intel processors.

1. INTRODUÇÃO

A segurança de sistemas computacionais exige um processo de melhoria contínua, incluindo a capacidade de executar verificações em um sistema-alvo para minimizar o risco de que ameaças em potencial se tornem ataques bem-sucedidos. Um exemplo desse tipo de verificação é o uso de *software* antivírus.

Nesse contexto, alguns conceitos são fundamentais, como: isolamento, privilégio e visão. Ferramentas de segurança devem operar de maneira isolada para minimizar o risco de sofrer os mesmos ataques que estão tentando evitar. Devem ter alto privilégio para entrar em ação no momento necessário, com prioridade sobre as demais tarefas em execução no sistema. Além disso, precisam ter visão ou alcance global do sistema e habilidade para lidar com o próprio ambiente em que está operando, aumentando as chances de sucesso ao realizar suas tarefas.

Uma tarefa de segurança comum, que demanda isolamento e alto privilégio, é medir a integridade de componentes do sistema, como arquivos essenciais de sistemas operacionais e de monitores de máquinas virtuais, atuando em ambientes virtualizados, como *data centers* e estruturas de computação em nuvem. A verificação de segurança pode ser feita de maneira estática (quando o item-alvo está apenas armazenado na memória, sem ser atualizado ou estar em execução) ou dinâmica (durante a execução do item-alvo).

Em geral, para medir a integridade de maneira estática, uma ferramenta de segurança pode executar um serviço criptográfico a fim de gerar um código de resumo (*hash*) do alvo. Para verificar a integridade, é necessário e suficiente gerar um novo código

1. Capitão de Fragata (T) do Centro de Análises de Sistemas Navais – Rio de Janeiro, RJ – Brasil. E-mail: william.augusto@marinha.mil.br

hash do alvo um momento à frente no tempo e comparar os dois códigos. Se os códigos forem iguais, nenhuma mudança terá ocorrido e a integridade estará preservada. Do contrário, pode-se dizer que houve violação da integridade, mas não maliciosa. Cabe ao administrador do sistema fazer esse julgamento, já que, em tese, ele tem o controle sobre as alterações não maliciosas no ambiente. Realizar a medição dinâmica é muito mais complicado, porque o alvo irá, naturalmente, variar o seu conteúdo durante o tempo, gerando diferentes códigos *hash*, o que não indica necessariamente violação de integridade. Em ambos os casos, a ferramenta de segurança deve ser instalada em um ambiente isolado para não ser atacada e violada, ter privilégios suficientes e boa visão (alcance) dos itens-alvo distribuídos pelo sistema computacional.

Assim, o problema que os analistas de segurança estão enfrentando é a instalação de uma ferramenta de segurança em um ambiente isolado no sistema computacional, com alto privilégio e boa visão dos alvos, a fim de minimizar o risco de ataques.

Os recursos disponíveis no sistema computacional quando o processador entra no modo de gerenciamento do sistema (*system management mode* — SMM) oferecem ambiente e características apropriados para serem usados por uma ferramenta de segurança, tais como: isolamento, alto privilégio e boa visão do sistema. Neste artigo, são considerados apenas os processadores Intel, entretanto muitas das características válidas para processadores Intel são válidas para processadores de outros fabricantes, como os da AMD.

Por seus recursos — área de memória isolada, alto privilégio e prioridade de execução, inclusive sobre o sistema operacional, além de visão e acesso completo a todo o sistema — , o SMM tem sido usado para outros propósitos além dos descritos nos manuais da Intel (INTEL, 2014a; 2014b; 2014c), contribuindo para a corrida armamentista entre atacantes e defensores de sistemas computacionais.

O SMM foi introduzido na arquitetura IA-32, no processador Intel 386SL, lançado em 1990 (INTEL, 2001; 2003; 2014c; 2015), com o objetivo de fornecer recursos para gerenciar o sistema computacional hospedeiro do processador. Desde então, o SMM tem sido um item padrão nas arquiteturas IA-32 e IA-64. Não foi por acaso que o SMM foi lançado no processador Intel 386SL, uma versão móvel do Intel 386 (INTEL, 2001; 2003; 2015). Computadores móveis precisam de uma maneira robusta para o gerenciamento de energia do sistema. Assim, o SMM foi desenvolvido para fornecer gerenciamento de energia e outras funções de gerenciamento do sistema, como controle de *hardware*. Antes do SMM, a maioria dessas funções era realizada por ferramentas *in-circuit emulation* (ICE).

Uma característica-chave do SMM é o seu alto privilégio; o *software* executivo do SMM, chamado *SMI handler*, tem privilégios mais altos do que os do sistema operacional ou dos monitores de máquinas virtuais. Ele é instalado pelo Sistema Básico de Entrada e Saída (BIOS) em uma área de memória isolada e protegida, permitindo-o gerenciar tarefas críticas do sistema com máxima prioridade. Esse é o ponto central deste trabalho, dado que, se sobrescrevermos o *SMI handler* por uma ferramenta de segurança ou por uma arma cibernética, esses artefatos terão, em tese, os mesmos recursos do *SMI handler*. Este trabalho discute como realizar essa substituição, estabelecendo os requisitos necessários e propondo a arquitetura que o artefato (de segurança ou arma cibernética) deverá usar para ser bem-sucedido.

Neste trabalho, é possível, sem perda de generalidade, substituir os termos "segurança" ou "ferramenta de segurança" por "código malicioso" ou "arma cibernética" e, assim, hospedar na memória do SMM um poderoso código malicioso que teria acesso irrestrito e protegido a todo o sistema computacional. Por questões práticas, a expressão "ferramenta de segurança" é usada com mais frequência do que "código malicioso" (ou "arma cibernética") neste trabalho.

Nos últimos anos, o SMM foi explorado com sucesso de diferentes maneiras, por exemplo: evitando os mecanismos de proteção do SMM com uso da técnica *cache poisoning* (DUFLOT; ETIEMBLE; GRUMELARD, 2006; WOJTCZUK; RUTKOWSKA, 2009b) ou explorando uma falha de implementação em um *SMI handler* específico, o que leva a uma brecha nos mecanismos do *Intel Trusted Execution Technology* (Intel TXT), abrindo caminho para a exploração bem-sucedida do monitor de máquinas virtuais Xen (um dos mais conhecidos e usados no mercado) (WOJTCZUK; RUTKOWSKA, 2009a). Por outro lado, pesquisadores em segurança têm apresentado novas ferramentas que tiram proveito dos recursos do SMM, tais como: HyperSentry (AZAB et al., 2010), usada para medir a integridade de monitores de máquinas virtuais; *Strongly Isolated Computing Environment* (SICE) (AZAB; NING; ZHANG, 2011), para fornecer um ambiente de execução isolado; e AppCheck (WANG; SUN; STAVROU, 2012), para proteger aplicações por meio da inspeção dos respectivos códigos armazenados na memória física.

2. OBJETIVO

Apesar do número considerável de trabalhos sobre os recursos do SMM, não há estudos que investiguem o próprio SMM e seus fundamentos. Os trabalhos encontrados reportam o uso dos recursos e as maneiras de tirar vantagem de algumas características do SMM, mas há carência de estudos dedicados a entender esse sistema, estabelecer requerimentos para o seu uso em tarefas de segurança e definir uma arquitetura para sua utilização como plataforma para ferramentas de segurança.

Em particular, as ferramentas de segurança baseadas nos recursos do SMM estudadas nesta pesquisa não tiram vantagens de todos esses atributos, falhando notadamente em explorar seu isolamento e sua transparência. Além disso, muitas dessas ferramentas não respeitam as restrições e limitações do SMM, como o tempo máximo de latência (tempo em que o sistema operacional fica "parado", esperando a execução do *SMI handler*). Em geral, a arquitetura dessas ferramentas é modular, e apenas um dos módulos é colocado na área de memória isolada e protegida do SMM, comunicando-se com outros módulos armazenados em áreas não protegidas do sistema, os quais carecem do isolamento oferecido pelo SMM e podem ser atacados, assim como outros itens do sistema.

Assim, esta pesquisa investiga o SMM no contexto dos processadores Intel para estabelecer um conjunto de requerimentos e, a partir dele, propor uma arquitetura genérica que pode ser usada para construir ferramentas de segurança ou armas cibernéticas de alta capacidade, as quais teriam o controle total do sistema computacional onde fossem instaladas. Por fim, uma prova de conceito é construída para medir a integridade de um arquivo do monitor de máquina virtual Xen. Tal medição é limitada ao mínimo necessário para provar o conceito da arquitetura.

3. METODOLOGIA DO TRABALHO

A metodologia empregada neste trabalho consistiu em pesquisa na literatura científica e nos manuais da Intel e em sondagens, por meio de programas nas linguagens C e *assembly*, a diversos computadores equipados com diferentes modelos de processadores Intel para identificar os detalhes expostos nos manuais desses produtos.

Um conjunto de requerimentos e um modelo de ameaças foram estabelecidos, e uma arquitetura genérica foi concebida para ser usada na construção dos artefatos. Uma prova de conceito foi construída usando a linguagem C.

4. RESULTADO DO TRABALHO

Os requerimentos estabelecidos para construir artefatos que utilizem os recursos do SMM foram os seguintes:

- Requerimento 1 – Pequeno. Existem 32512 *bytes* disponíveis para o código do *SMI handler* e seus dados na memória isolada e protegida do SMM, chamada SMRAM. A SMRAM pode ter o tamanho de até 4 *Gbytes*, mas é recomendado manter o tamanho mínimo para não afetar a transparência das ferramentas. Se forem observadas as ferramentas de segurança que usam recursos do SMM, nota-se que a maioria delas utiliza arquitetura semelhante, levando em conta uma máquina remota para analisar os dados recolhidos e para outras funções de gerência. Essa arquitetura resolve a questão do espaço limitado, modularizando as ferramentas. Embora amenize um problema funcional, isso cria um problema de segurança, pois abre novos locais de ataque, como o canal de comunicação, os *drivers* e os dispositivos utilizados para permitir a comunicação com a máquina remota. Além disso, tais ferramentas perdem a proteção de isolamento do SMM ao usar outros módulos fora da SMRAM. Assim, uma aplicação de segurança baseada em SMM deve ser pequena o suficiente para caber no menor tamanho disponível.
- Requerimento 2 – Rápido. A Intel especifica que a latência máxima, ou seja, o tempo máximo que o sistema computacional pode permanecer em execução no modo SMM, deve ser inferior a 150 microssegundos para minimizar o risco de travamento, entre outros problemas (INTEL, 2014d). Assim, o tempo de execução a cada chamada para a execução do artefato deve ser menor ou igual a 150 microssegundos.
- Requerimento 3 – Persistente. A SMRAM é volátil, e a reinicialização do sistema irá limpar todo o seu conteúdo. Assim, o código e os dados relacionados com SMM precisam ser recarregados a cada inicialização. Um artefato que utiliza os recursos do SMM, portanto, precisa ser incorporado na BIOS (ou entidade equivalente), uma vez

que a BIOS contém o código relacionado ao SMM e seus dados e carrega-os para a SMRAM durante o processo de inicialização do sistema computacional.

- Requerimento 4 – Cooperativo. As funções do *SMI handler* precisam ser preservadas, uma vez que têm importantes tarefas a executar. Qualquer *software* baseado no SMM deve preservar as funções originais do *SMI handler*, adicionando seu próprio código ao código desse software executivo sem substituir qualquer parte dele. Ao entrar no SMM, o processador procura a primeira instrução a ser executada no endereço indicado pelo conteúdo do registrador SMBASE + 0x8000 (por 0x38000 por padrão) na SMRAM, onde o *SMI handler* está localizado, o que implica que qualquer artefato baseado no SMM deve ser uma versão modificada do *SMI handler*.
- Requerimento 5 – Isolado. O *SMI handler* — e, consequentemente, qualquer *software* baseado no SMM — executa suas tarefas sem notificar ou ser reconhecido pelo sistema operacional (GRAWROCK, 2008). Além disso, uma vez que o sistema operacional fica parado durante todo o tempo em que o processador está no SMM, a execução do *software* baseado em SMM é transparente para qualquer outro *software* no sistema. Logo, produzir um artefato para usar o SMM e colocá-lo na área de memória fora da SMRAM é, no mínimo, contraintuitivo, uma vez que a principal motivação para usar o SMM é beneficiar-se de seus recursos, como isolamento e transparência.
- Requerimento 6 – Resistente. O *cache poisoning* é um ataque efetivo ao SMM, possível graças à manipulação dos registradores *Memory Type Range Registers* (MTRRs) para tornar a SMRAM "cacheável" (dados da SMRAM passam a ser copiados na memória *cache*). A fim de impedir esse ataque, o mecanismo Interface SMRR, disponibilizado pela Intel, deve ser usado para proteger os registradores MTRR relacionados. Assim, qualquer recurso disponível no sistema para reforçar a segurança do SMM deve ser empregado. Alerta-se que nem todos os processadores Intel possuem suporte para a Interface SMRR.
- Requerimento 7 – Ser chamado por qualquer interrupção de gerenciamento de sistema. Para iniciar qualquer *software* baseado no SMM, uma interrupção de gerenciamento de sistema (SMI) deve ser gerada. Uma abordagem comum para desencadear uma SMI e iniciar tal *software* é escrever na porta de entrada e saída "0xB2H". Como essa é uma abordagem bastante conhecida, um ataque pode ser usado para impedir tal ação pela identificação da assinatura do código e, em seguida, negação do uso dessa via. Assim, um artefato baseado no SMM deve tirar proveito de qualquer SMI gerada para começar a executar seu trabalho. Por outro lado, sempre que a ferramenta precisar começar, ela deverá ser capaz de iniciar uma SMI de diferentes maneiras.
- Requerimento 8 – Completo. Como discutido nos requerimentos 1 (pequeno, para que todas as partes fiquem dentro da SMRAM) e 5 (isolado, todas as partes dentro da SMRAM e, assim, sem uso de máquina remota), algumas ferramentas de segurança precisam de uma máquina remota para analisar os dados recolhidos e para outras funções de gerência. Tais ferramentas devem manter parte de seu código no sistema operacional ou no monitor de máquina virtual. A HyperSentry (AZAB et al., 2010), por exemplo, usa um agente implantado na base de código do monitor de máquina virtual. Enquanto resolvem alguns problemas, essas implementações perdem os benefícios de usar o SMM, como isolamento e transparência. Assim, um *software* baseado no SMM deve ter todas as funcionalidades para executar suas tarefas e todos os dados necessários completamente instalados na SMRAM.

Uma prova de conceito foi construída usando a linguagem C para medir a integridade de um arquivo essencial (xend-config.sxp) do monitor de máquinas virtuais Xen. Os experimentos foram realizados com sucesso em 13 máquinas, com detalhes dos processadores e *chipset* de cada máquina (Tabela 1).

5. CONCLUSÃO

Este trabalho apresenta uma pesquisa sobre o SMM em processadores Intel a fim de estabelecer um conjunto de requerimentos e, a partir dele, propor uma arquitetura genérica que pode ser usada para construir ferramentas de segurança ou armas cibernéticas de alta capacidade, as quais teriam controle total do sistema computacional onde fossem instaladas.

Uma prova de conceito foi construída e os experimentos realizados foram bem-sucedidos, demonstrando a viabilidade da proposta e a validade dos requerimentos.

Tabela 1. Máquinas utilizadas nos experimentos.

OEM	Processador	MCH	ICH
Gigabyte	Dual Core E2160	945P	ICH7
Gigabyte	Dual Core E2160	945P	ICH7
Gigabyte	Pentium 4	945P	ICH7
Gigabyte	Pentium 4	945P	ICH7
Asus Tek	Pentium 4	Não identificado	ICH7
Dell	i5-2500	Sandy Bridge	H67
Asus	Atom N270	82945GSE	ICH7-M
Foxconn	Atom N270	E7500	ICH3-M
HP	i5-650	Clarkdale	P55
Fujitsu	Pentium 4-M 548	82845	ICH3-M
HP	i7-2670QM	Sandy Bridge	HM65
Acer	i5-3450	Ivy Bridge	B75
Compaq	Pentium III-M	82830M	ICH3-M

REFERÊNCIAS

AZAB, A.M.; NING, P.; WANG, Z.; JIANG, X.; ZHANG, X.; SKALSKY, N.C. Hypersentry: Enabling stealthy in-context measurement of hypervisor integrity. *In*: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 17., 2010, Nova York. *Anais*... Nova York: ACM, 2010. p. 38-49.

AZAB, A.M.; NING, P.; ZHANG, X. SICE: A hardware-level strongly isolated computing environment for x86 multi-core platforms. *In*: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 18., 2011, Nova York. *Anais*... Nova York: ACM, 2011. p. 375-388. http://doi.org/10.1145/2046707.2046752

DUFLOT, L.; ETIEMBLE, D.; GRUMELARD. Using CPU system management mode to circumvent operating system security functions. *In*: CANSECWEST SECURITY CONFERENCE, 7., 2006. *Anais*... 2006.

GRAWROCK, D. *Dynamics of a trusted platform:* a building block approach. 2ª ed. Hillsboro, OR: Intel Press, 2008.

INTEL. *Intel IA-32 Architecture Software Developer's Manual*. Denver: Intel Corporation, 2001. v. 3.

_____. *Intel IA-32 Architecture Software Developer's Manual*. Denver: Intel Corporation, 2003. v. 3.

_____. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver: Intel Corporation, 2014a. v. 1.

_____. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver: Intel Corporation, 2014b. v. 2.

_____. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver: Intel Corporation, 2014c. v. 3.

_____. The Intel BIOS Implementation Test Suite (BITS), version 1090. Denver: Intel Corporation, 2014d.

_____. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver: Intel Corporation, 2015. v. 3.

WANG, J.; SUN, K.; STAVROU, A. Hardware-assisted application integrity monitor. *In*: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCE, 45., 2012. *Anais*... HICSS, IEEE Computer Society, 2012. p. 5375-5383. http://doi.org/10.1109/HICSS.2012.299

WOJTCZUK, R.; RUTKOWSKA, J. Attacking intel trusted execution technology. *In*: BLACK HAT CONFERENCE., 2009. *Anais*... Las Vegas, 2009a.

WOJTCZUK, R.; RUTKOWSKA, J. Attacking SMM memory via intel CPU cache poisoning. 2009b.