Detecting BluePill

Edgar Barbosa
COSEINC Advanced Malware Labs

Hackers to Hackers Conference – 2007
Brasília - Brazil

Autor

- Edgar Barbosa
- Pesquisador de segurança atualmente trabalhando para a COSEINC, empresa situada em Cingapura.
- Tenho experiência com engenharia reversa de binários, especialmente do kernel do Windows e com projetos envolvendo detalhes da micro-arquitetura de processadores x86/x64
- Vários artigos publicados no site www.rootkit.com
- Participei da criação do BluePill, um rootkit baseado em suporte de virtualização por hardware

Conteúdo

- Parte I
  - Como rootkits baseado em virtualização por hardware funcionam?
- Parte II
  - Como detectá-los ?

Hardware virtualization rootkits

Detecting Bluepill

Hardware virtualization rootkits

- Intel e AMD desenvolveram extensões de hardware nos seus processadores para facilitar a implementação de máquinas virtuais.
- Intel VM-x e Amd SVM.
- Os mais conhecidos rootkits baseados em virtualização por hardware são:
  - Vitriol, criado por Dino Dai Zovi – usa Intel VT-x
  - Bluepill, idealizado pela Joanna Rutkowska – usa AMD SVM
- Joanna publicou o código fonte de uma variação do Bluepill no site http://bluepillproject.org/
- Bluepill foi implementando em processadores AMD mas os conceitos de funcionamento são os mesmos de processadores Intel.

Bluepill

- Idealizado por Joanna Rutkowska
- Propriedade intelectual da COSEINC
- Utiliza as extensões SVM dos processadores AMD
- Roda em 64-bits
- Suporta sistemas multicore
- É implementado como um device driver no Windows (.sys)

AMD SVM

- SVM significa “Secure Virtual Machine”
- É uma extensão de CPU para suportar Virtual Machine Monitors (VMM), também conhecidos como hypervisors.
- 8 novas instruções:
  - VMRUN
  - VMSAVE
  - VMLOAD
  - VMMCALL
  - CLGI
  - STGI
  - SKINIT
  - INVLPGA

Inicialização de um rootkit SVM

- Para poder executar as novas instruções SVM, o bit SVME do registrador MSR 'EFER' precisa estar setado como 1
  - Tentar executar uma instrução SVM enquanto o valor SVME é zero, resulta em uma exceção do processador (#UD – Invalid Opcode)
- Alocar (alinhado em 4kB) e inicializar a estrutura VMCB.
- A estrutura VMCB (Virtual Machine Control Block) descreve uma máquina virtual a ser executada
- Ela contém:
  - Instruções ou eventos que devem ser interceptados na máquina virtual
  - Bits de controle (paginação, TSC offset, …)
  - Estado do processador em Guest mode(Registradores,DRx, CRx)

Inicialização de um rootkit SVM

- Após a inicialização da estrutura VMCB, é necessário ajustar o registrador MSR VM_HSAVE_PA. Este registrador armazena o endereço físico onde deve ser salvo as informações de contexto do hypervisor quando transferindo a execução para uma máquina virtual.
- Executar a instrução VMRUN com o registrador RAX igual ao endereço físico onde foi alocado a estrutura VMCB correspondente à máquina virtual a ser executada.

Inicialização de um rootkit SVM

- AJUSTAR EFER.SVME = 1
- ALOCAR UMA PÁGINA ALINHADA EM 4KB PARA O VMCB
- INICIALIZAR O VMCB
- CONFIGURAR O REGISTRADOR VM_HSAVE_PA
- EXECUTAR VMRUN (RAX -> VMCB)

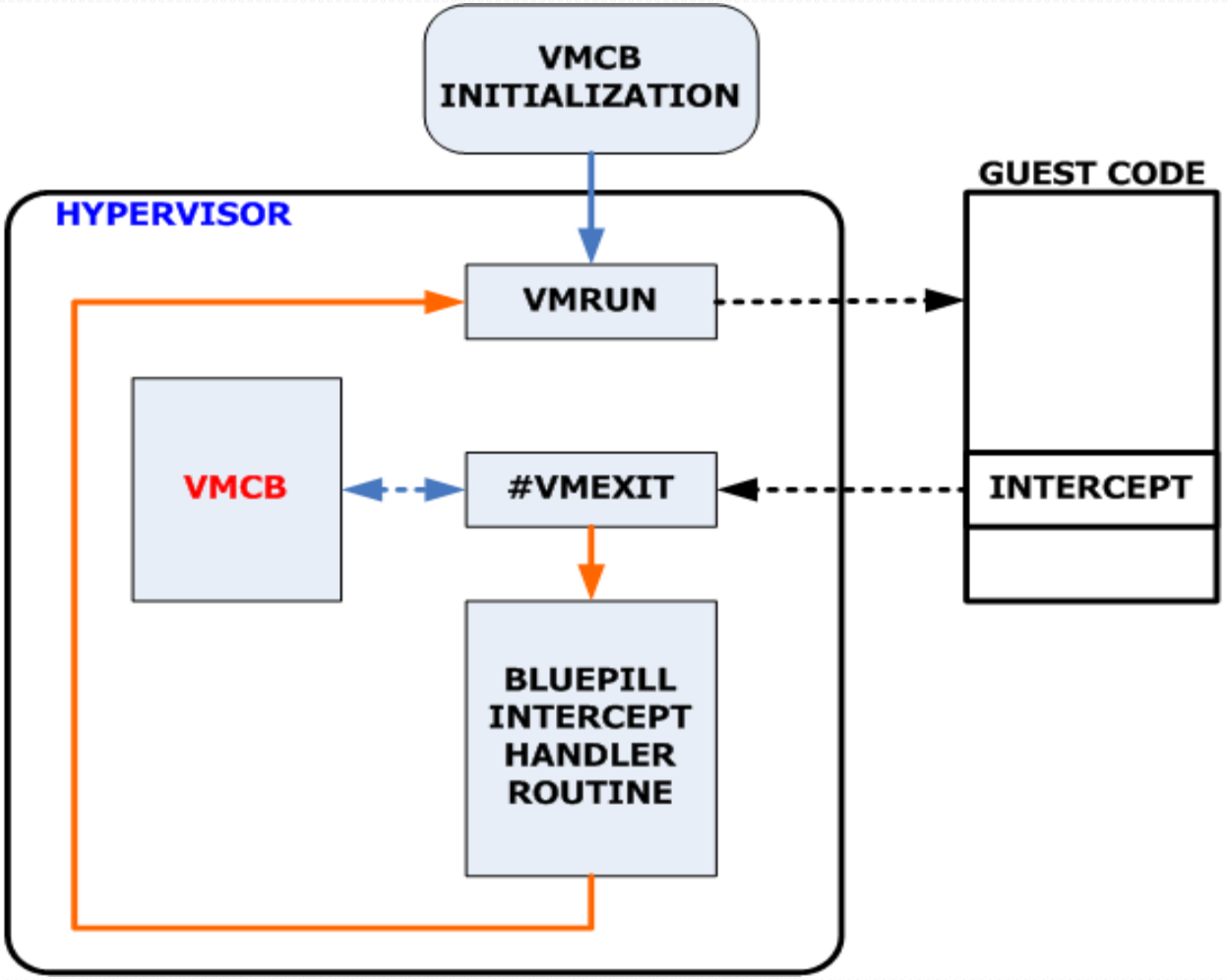
Instrução VMRUN

- Disponível apenas para código em Ring o
- A CPU entra em um novo modo de processamento: Guest Mode
- Em “guest mode” o comportamento de várias instruções mudam para facilitar a virtualização
- O processador executa checagens de consistência no estado dos modos Host e Guest
- Salva o estado do hypervisor na área VM_HSAVE_PA
- Carrega a máquina virtual descrita pelo VMCB
- A máquina virtual executa até que ocorra um #VMEXIT

#VMEXIT

- Quando um evento ou instrução de interesse do hypervisor ocorre na máquina virtual, o processador realiza um #VMEXIT
- Em um #VMEXIT o processor:
  - Desativa as interrupções
  - Desativa todos os breakpoints
  - Novamente checa o estado do hypervisor antes de recarregá-lo.
- O motivo pelo qual um #VMEXIT ocorre é salvo no campo EXITINFO da estrutura VMCB.
- Transfere o controle de execução para o hypervisor, neste caso, o rootkit Bluepill.

Bluepill hypervisor



Métodos de detecção

Detecting Bluepill

Rootkits “indetectáveis”

- Segundo Popek e Goldberg, as propriedades de todo gerenciador de máquinas virtuais devem ser:
  - Eficiência
  - Controle de recursos
  - Equivalência
- Equivalência implica que qualquer programa sendo executado em uma máquina virtual deve se comportar do mesmo modo como se estivesse sendo executada diretamente na máquina nativa [1]
- Rootkits baseados em SVM/VT-x são apenas teoricamente indetectáveis devido ao fato de que nestas arquiteturas o princípio de equivalência não é completamente respeitado.
- Os recursos que o hypervisor não consegue virtualizar completamente são:
  - TLB (parcial)
  - Branch prediction
  - Processamento SMP

Timming attacks

- O mais óbvio ataque contra rootkits como BluePill é o ataque baseado na medição do tempo.
- Mede-se o tempo de execução de alguma instrução interceptada pelo hypervisor e compara-se o resultado com o tempo medido em uma máquina que não esteja virtualizada.
- Entretanto, AMD e Intel criaram extensões para facilitar a virtualização de tentativas de leituras de fontes internas de tempo no processador:
  - Instrução RDTSC
  - Instrução RDMSR
  - Portas de entrada/saída

Métodos de detecção:

- Os métodos de detecção serão baseados em:
  - TLB
  - Branch prediction
  - Medição de tempo baseado em contadores
- Ataques baseados na controladora DMA não serão apresentados devido aos novos processadores com suporte a unidades IOMMU.

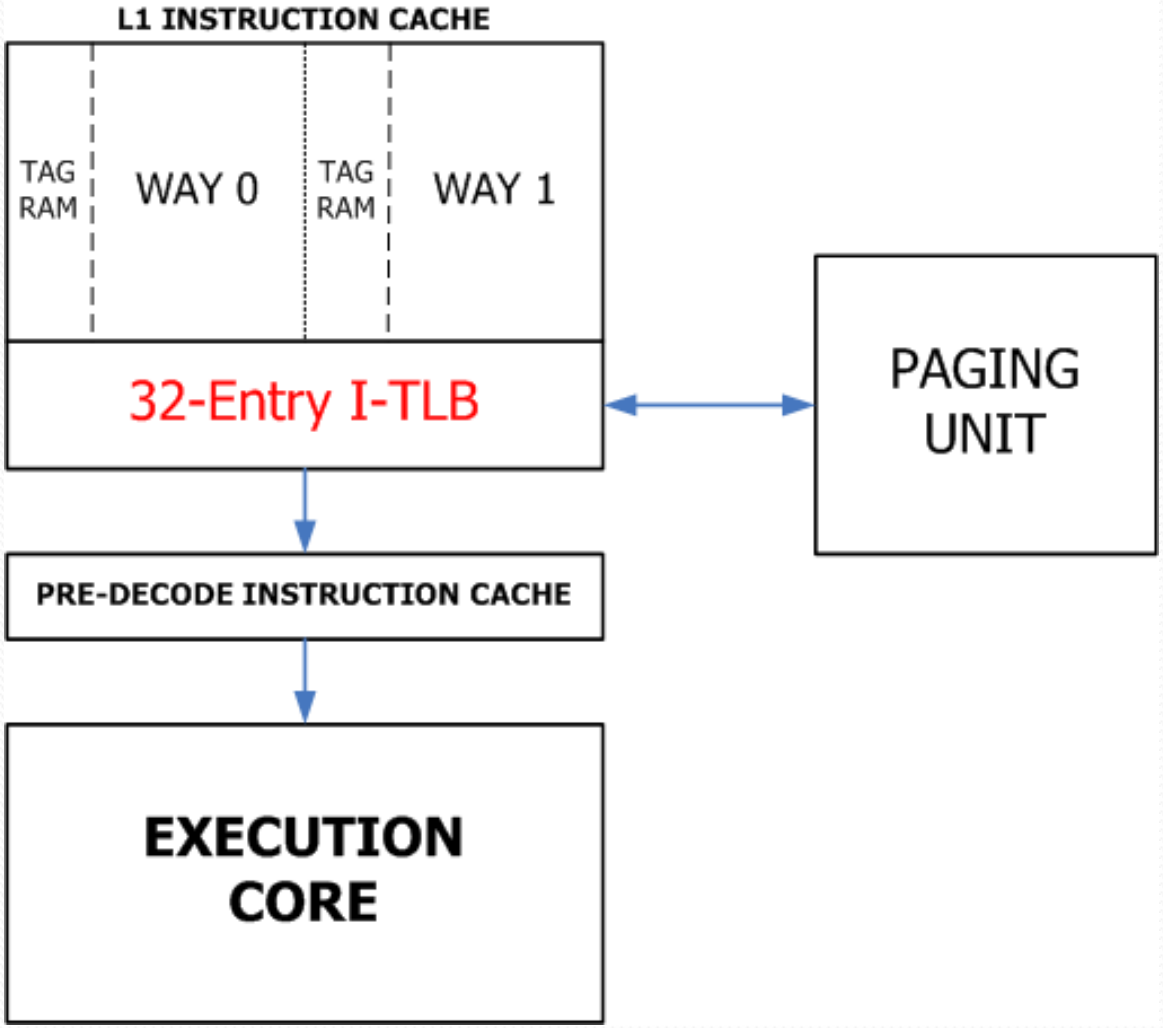
TLB

- Translation Lookaside Buffer (TLB) é um cache de CPU extremamente veloz que é usada para acelerar a tradução de endereços virtuais. Existem TLBs para acesso de instruções e para acesso de dados.
- Informações detalhadas sobre a TLB podem ser obtidas através da instrução CPUID, como o número de entradas de cada TLB, o tipo e associatividade.

TLB

- Cada linha de entrada na TLB armazena dados como:
  - Tag, usado para comparar o endereço virtual
  - ASID
  - Endereço físico, resultado da tradução do endereço virtual
- Se a tradução do endereço virtual está presente no cache (TLB hit), o TLB imediatamente retorna o endereço físico armazenado, caso contrário (TLB miss), ele envia o endereço virtual para a unidade de paginação da CPU para ser traduzido.
- TLB hit = 1 ciclo de tempo
- TLB miss = 30 ciclos de tempo

I-TLB


L1 INSTRUCTION CACHE
TAG RAM
WAY 0
TAG RAM
WAY 1
32-Entry I-TLB
PAGING UNIT
PRE-DECODE INSTRUCTION CACHE
EXECUTION CORE

TLB

- O cacheTLB em processador AMD usam ASIDs para marcar entrada na TLB.
- ASID = Address Space Identifier
- O hypervisor pode atribuir um ASID para cada máquina virtual. Assim, as traduções de endereço de um máquina virtual não interferem nas entradas no cache armazenadas pelas traduções de outra máquina virtual.
- Quando o TLB recebe um endereço virtual, ele retorna o endereço físico, se e somente se existe um entrada para o endereço virtual com o tag ASID equal ao ASID da máquina virtual.
- ASID = 0 somente para o hypervisor (VMM)
- ASID = 1 a 63 disponíveis para as máquinas virtuais.

TLB

- O cache TLB não pode ser desativado. Está sempre funcionando após a ativação de paginação no sistema.
- Não existe uma forma de acesso direto às entradas armazenadas na TLB.
- As instruções INVLPG e INVLPGA são as únicas que possuem um efeito direto sobre a TLB. Elas servem para invalidar uma entrada na TLB e ambas intruções são interceptáveis pelo hypervisor

TLB

- O pesquisador Peter Ferrie (Symantec) foi o primeiro a sugerir a detecção de um hypervisor através do uso da TLB[2]. Entretanto, devido a impletação de ASID nas CPUs AMD, ele posteriormente declarou que seu método não seria mais útil nestes processadores[3]
- O método sugerido é simples:
  - Preencha a TLB como valores conhecidos. Isto é realizado através do acesso a diversas páginas presentes na memória. A cpu irá então inserir estas traduções de endereço no cache TLB.
  - Faça a medida do tempo necessário para acessar estas páginas de memória.

TLB

- Execute uma instrução privilegiada que necessariamente precisa ser interceptada por um hypervisor, caso haja algum sendo executado no sistema. O método original sugere usar a instrução CPUID, mas o BluePill não necessita interceptar esta instrução. O código do hypervisor que será executado após a interceptação da instrução privilegiada vai interferir nas entradas do cache TLB.
- Faça agora a medida novamente do tempo necessário para acessar as mesmas páginas de memória e compare a diferença entre entre as medições anteriores. Se houver diferença significativa, significa que a instrução foi interceptada e portanto existe um hypervisor ativo no sistema.

TLB

- O fato é que o uso de ASID não impede que usemos este método de detecção.
- O cache TLB é um recurso finito do sistema. O código de detecção pode solicitar tantas traduções ao cache, que o mesmo ficará lotado de traduções deste código, preenchendo assim todo o cache.
- Após o preenchimento, o método anterior é executado.
- Após a medida inicial de tempo para acessar as páginas, uma possível instrução que obrigatoriamente deve ser interceptada por um rootkit como o Bluepill é a RDMSR(EFER). Após esta instrução, acessa-se as mesmas páginas novamente e compara-se o tempo gasto. Caso haja diferença o hypervisor foi detectado.

Branch prediction

- Estudos mostraram que o comportamento de instruções de desvio condicionais nó código são previsíveis. [5]
- O histórico das do resultado das últimas instruções de desvio pode ser usada para a previsão do comportamento futuro destas instruções.
- As CPUs modernas implementam uma unidade para tentar prever o resultado deste tipo de instruções, aumentando consideravelmente a performance de execução. Essa unidade é chamada de BPU (Branch Prediction Unit)
- Se o destino de uma instrução de desvio foi prevista como sendo a próxima instrução e esta previsão se mostra falsa, existe uma grande penalidade de performance nos processadores devido ao fato de que todo o pipeline de instruções da CPU precisa ser esvaziado.

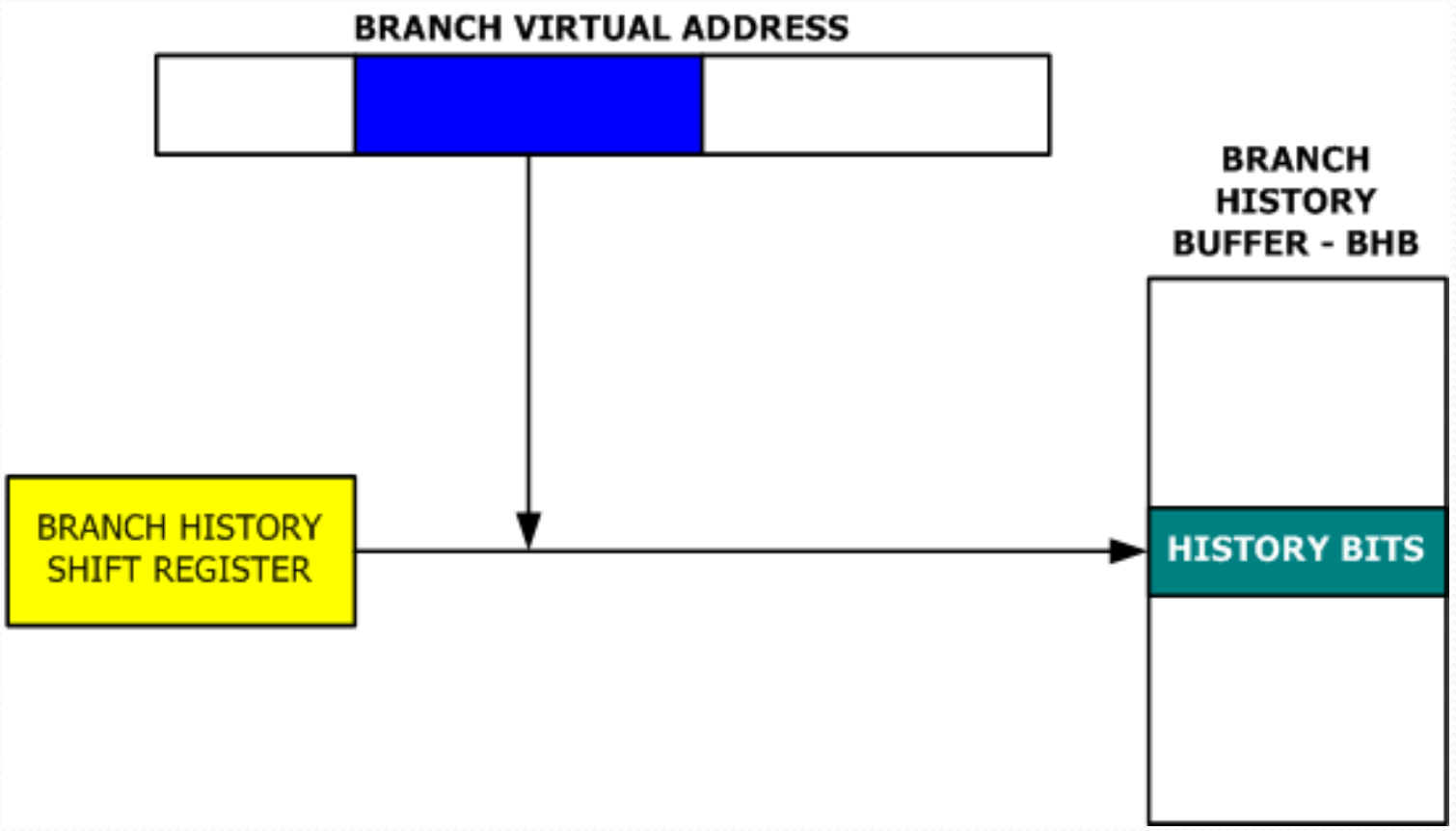
Branch prediction

- A BPU (Branch prediction unit) utiliza um pequeno cache para armazenar o histórico de instruções de desvio executadas e outro buffer para armazenar o histórico do alvo das intruções de desvio.
- Este segundo buffer é chamado de BTB (Branch Target Buffer)
- Como usar a BPU para detectar um hypervisor?
  - Usando as regras de previsão da BPU, podemos criar códigos que usam instruções de desvio cujos resultados serão armazenados no buffer BTU. A idéia é executar este código e medir seu tempo de execução. Após a execução do código, executamos uma instrução que um possível hypervisor teria de interceptar caso esteja ativo. A execução do código do hypervisor responsável pelo gerenciamento de interceptações irá interferir nos valores armazenos na BTU. O detector agora mede novamente o tempo gasto para executar o código com as instruções de desvio. Em caso de diferença, o hypervisor é detectado.

Branch prediction

- A BPU já foi usada anteriormente para efetivamente obter uma chave de criptografia de 512-bits usando um método chamado “Branch Prediction Analysis (BPA)”[6]. Este método é baseado em algumas característica interessantes da BPU:
  - Os dados armazenados nos buffers da BPU são acessados usando-se apenas alguns bits do endereço virtual da instrução de desvio. Portanto, duas instruções de desvio que estão localizadas em endereços diferentes mas compartilham estes mesmos bits, terão a mesma entrada no buffer sendo usada pela BPU!!! Este efeito é chamado de Branch Aliasing ou Branch Interference.
  - Os caches da BPU são compartilhados entre todas as threads.

Branch prediction



Branch prediction

- O método consiste portanto em criar uma thread espiã que é executada simultaneamente com a thread responsável pela descriptografia. Como as duas threads irão usar as mesmas entradas no cache da BPU, a thread espiã consegue determinar quais instruções de desvio foram executadas na outra thread e usando métodos matemáticos complexos, determinar a chave.
- Entretanto, não é possível usar este efeito de “Branch Aliasing” para detectarmos hypervisor devido ao fato de que não sabemos o endereço virtual que o mesmo usa quando está sendo executado.

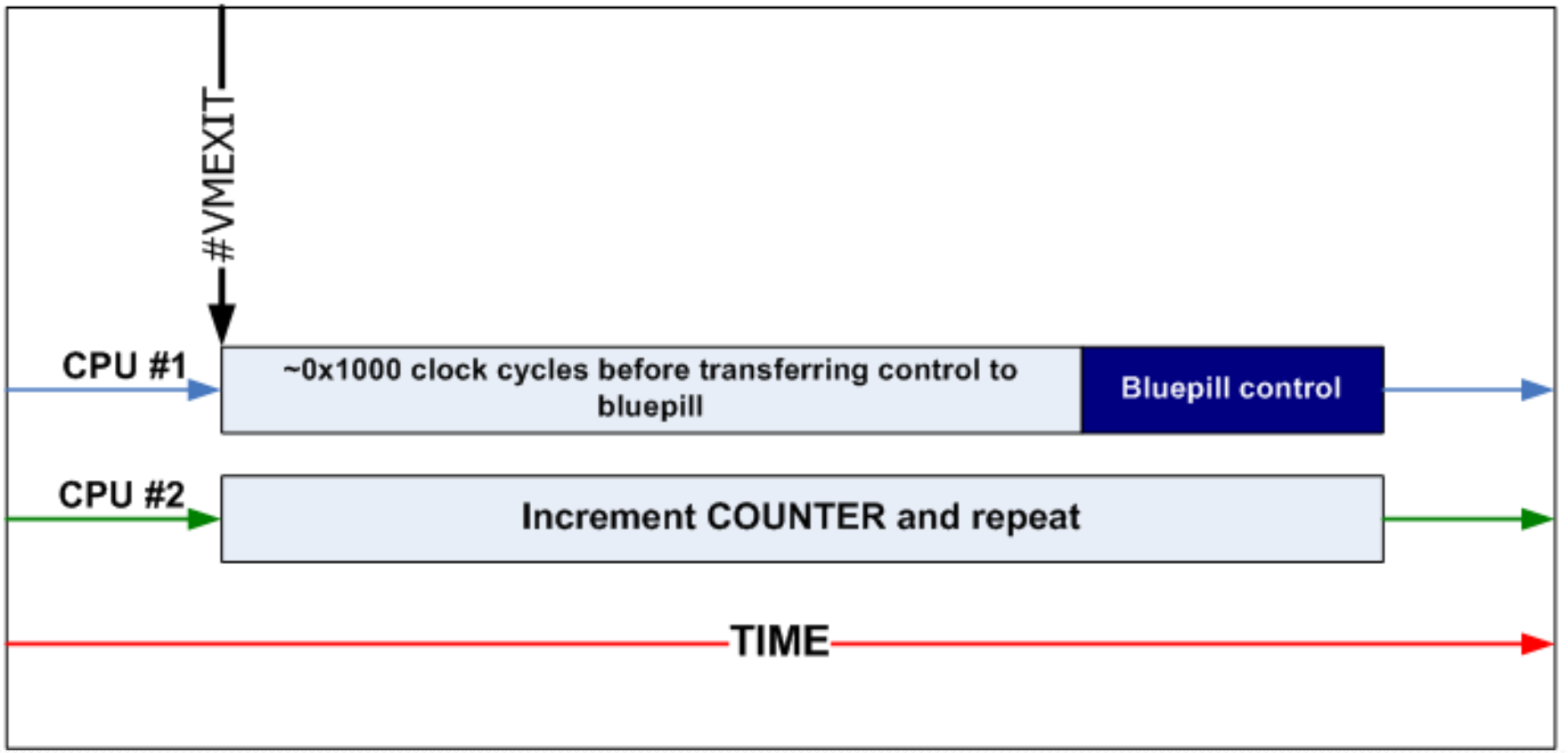
Counter based detection

- O rootkit Bluepill pode interceptar e virtualizar tentativas de leitura de qualquer fonte interna de medição de tempo.
- Entretanto, é possível criar uma forma de medir tempo através de um simples algoritmo de contagem.
- Usando sistema multicore, podemos criar 2 threads, cada uma sendo executada em um núcleo diferente e simultaneamente.
- Uma thread pode medir o tempo que outra thread gasta para executar alguma rotina ou instrução, mas para isso é necessária a criação de um protocolo de sincronização entre as duas threads para sinalizar o início e o termíno do processo de contagem de tempo.

Counter based detection

- Como este método realmente funciona?
- Idéia básica:
  - A Thread #1 irá disparar um sinal de sincronização inicial (variável global) e aguardar por uma resposta da thread #2.
  - Após a resposta da thread #2 a thread #1 irá executar uma instrução que um hypervisor precisa interceptar (rdmsr EFER), enquanto a thread #2 começa a incrementar o valor de um contador ao mesmo tempo. Se tiver um hypervisor presente no sistema, ele irá gastar milhares de ciclos de tempo para executar seu próprio código e retornar o controle para a thread #1. Caso não tenha um hypervisor ativo, a instrução privilegiada será executada em curtíssimo tempo.
  - Após a execução da instrução privilegiada, a thread #1 sinaliza para a thread #2 parar o processo de contagem.
  - Finalmente, o valor do contador é checado.

Counter based detection



Counter based detection

- Usando um simples contador como

  while(stopSignal == FALSE)

  counter++;

  temos os seguintes resultados após a execução de RDMSR(EFER):

  - Counter = 016h em um sistema sem hypervisor..
  - Counter = 02000h em um sistema com hypervisor ativo.
- O código do contador acima pode ser otimizado para aprensentar resultados ainda melhores através das regras de otimização de códigos dos processadores.

Counter based detection

- Para evitar detecção, bluepill precisa parar o contador da therad #2 assim que ela intercepta qualquer evento.
- Entretanto, somente a transferência de um #VMEXIT para o hypervisor gasta em média 1000h ciclos de tempo!!!
- Cada CPU possui seu próprio hypervisor. Para o bluepill esperando um #VMEXIT na CPU#2 avisar o bluepill na CPU #1, ele poderia usar uma interrupção IPI (Inter processor interrupt). Entretanto isto iria demorar ainda mais.
- O bluepill na CPU#1 não tem acesso direto ao contexto dos registradores da CPU #2.
- Não adianta alterar qualquer parâmetro do scheduler do sistema operacional.

Counter based detection

- O código de contagem do detector precisa ser executado da forma mais uniforme possível.
- Entretanto, quais as consequências se o código começar a ser interrompido por sinais externos, como as interrupções do clock da CPU?
- É umaboa idéia evitar interrupções, mas como um provável hypervisor tem como controlar completamente as fontes de interrupções do sistema, isso não tem muito efeito nos testes.
- A solução é simplesmente executar o código de detecção várias vezes.
- Entretanto, de acordo com vários testes, as interrupções não tiveram efeito perceptível no detector, bastando executar o método apenas uma vez para detectar o bluepill.

Counter based detection

- Existe outra forma para o bluepill evitar esta forma de detecção?
- É improvável. Inclusive podemos evitar que o bluepill ataque implementações específicas do detector pois é fácil criar diversas formas de algoritmos de contagem, assim como também determinar se duas threads estão realmente sendo executadas ao mesmo tempo.
- Se o bluepill tentar se descarregar ao detectar tal ataque, isso não impede sua detecção.
- A própria Joanna admitiu que não conseguiu criar um método para evitar esta forma de detecção e que está ainda pensando no assunto ☺

BP in hibernation-mode

- Joanna apresentou uma idéia interessante que foi inicialmente discutida no desenvolvimento do bluepill. A idéia seria a capacidade do rootkit se descarregar da memória após a detecção de inicialização de algum ataque e o posterior carregamento automático após o término do ataque. [8]
- Esta idéia foi reapresentada na última BlackHat como uma defesa aos ataques explicados nesta demonstração. Entretanto isso não é uma solução devido ao fato de que o detector pode ‘ capturar ‘ o controle do hypervisor podendo assim detectar posteriormente qualquer código que queira ter poderes de hypervisor. Outro ponto é que se um rootkit se descarrega, por definição os objetos que ele deveria estar escondendo serão revelados!

Failed attacks

- Alguns ataques propostos por outros pesquisadores não irão funcionar devido ao fato do hypervisor poder virtualizar os recursos necessários para sua detecção.[2]
- Algumas propostas inclusas:
  - Usar o mecanismo de Last Branch Record para detecção. Inútil devido ao suporte nativo em hardware para virtualizar o acesso ao LBR.
  - Usar os caches L1 e L2 para detecção. O hypervisor pode desativar estes caches através da programação dos bits CD e NW nos registradores de controle e através do bit PCD nas tabelas de paginação.

CPU bugs

- É possível usar bugs na CPU para detectar hypervisors?
  - Sim, mas não é uma forma muito confiável de se detectar rootkits.
  - Eu descobri que a execução da instrução VMSAVE juntamente com o opcode prefixo Address-Size (0x67) é capaz de congelar o processador AMD caso exista um hypervisor ativo no sistema ! ☺

Questions?

Credits

- Os fantásticos papers sobre ataques contra sistemas de criptografia baseados em detalhes da microarquitetura de cpus.

References

- [1] J. Smith and R. Nair. *Virtual Machines. Versatile platforms for systems and processes*. Morgan Kaufmann, 2005.
- [2]http://pferrie.tripod.com/papers/attacks.pdf
- [3]http://pferrie.tripod.com/papers/attacks2.pdf
- [4]http://www.chip-architect.com/news/2003_09_21_Detailed_Architecture_of_AMDs_64bit_Core.html
- [5]J. Shen and M. Lipasti. *Modern Processor Design. Fundamentals of Superscalar processors.* McGraw-Hill , 2005.
- [6]O. Acuçmez, Ç. Koç and J. Seifert. *On the power of simple branch prediction analysis.* http://eprint.iacr.org/2006/351.pdf
- [7] M. Milenkovic, A. Milenkovic and J. Kulick. *Demystifying Intel Branch Predictors.* http://www.ece.wisc.edu/~wddd/2002/final/milenkovic.pdf
- [8]http://blogs.zdnet.com/Ou/?p=297

Obrigado pela atenção!

edgar@research.coseinc.com