# Análise Transparente de *Malware* com Suporte por *Hardware*

Marcus Botacin[1], André Grégio[1,2], Paulo Lício de Geus[1]

[1]Instituto de Computação - UNICAMP
{marcus,paulo}@lasca.ic.unicamp.br

[2]Universidade Federal do Paraná (UFPR)
gregio@inf.ufpr.br

09 de Novembro de 2016

# Tópicos

# Tópicos

# Panorama

## *Malware*

- Ameaças persistentes.
- Propagação crescente de exemplares.
- Uso de técnicas de anti-análise.

## Cenário Atual

- *Packers* e ofuscação.
- Detecção de efeitos colaterais de emulação.
- Detecção de injeção de código em *runtime*.

## Objetivos

- Desenvolvimento de solução de análise transparente.

# Tópicos

# Trabalhos Relacionados

## *Hardware Virtual Machines* (HVM)

- Monitoramento Externo.
- Exige escrita de *hypervisor*.

## *System Management Mode* (SMM)

- Monitoramento a partir da BIOS.
- Exige reescrita da BIOS.

## Monitores de Performance

- Focados em Efeitos Colaterais.
- Limitações de implementação.

# Implementações Atuais

## Kbouncer, ROPecker, CFIMon, e outros

- Focados em ROP.
- Injeção de código.
- Base Estática.
- Implementação como extensão ou módulo.

## Proposta

- Solução Modular.
- Sem injeção de código.
- Reconstrução de fluxos.

# Tópicos

# Monitoração de *branch*

## Definição

- Recurso do processador.
- Armazenamento em registradores ou em memória.
- Dividido em LBR/BTS e PEBS.

## Funcionamento

- Interrupção ao atingir *threshold*.
- Filtragem de ações e por privilégio.
- Acesso via *kernel*.

# Monitoração de *branch*

## LBR/BTS

- `JNE,JMP,CALL,RET`.
- Tratamento de Exceções.
- Usado por soluções que tratam ROP.

## PEBS

- *Cache hit/miss, branches predicted*.
- Usado por soluções que tratam efeitos colaterais.

# Tópicos

# Proposta de solução.

## Funcionamento

- *Threshold* de 1 desvio.
- Interrupção para isolamento de processos.
- Introspecção de sistema para obtenção de contexto.
- *Dump* da memória de instruções para reconstruir fluxo de controle.

## Modelo de Ameaças

- Nível de usuário.
- *Single-Core*.
- Chamadas de API do sistema.
- Sistema Operacional moderno.

# Tópicos

# Implementação

## Captura de Dados

- *Model Specific Register* (MSR) exige *driver* de *kernel*.

## Tratamento de Interrupções

- *Local Vector Table* (LVT).
- *HalpPerfInterruptHandler hook*.
- *Non Maskable Interrupt* (NMI).

## Arquitetura

- Cliente-Servidor.
- Captura *system-wide*.
- Cliente implementa políticas e isolamento.

# Implementação

## Identificação de Processos

- `PsGetCurrentProcessId`

## Acesso à memória

- `ReadProcessMemory`.

## Bibliotecas Carregadas

- `GetModuleHandle`

## *Disassembly*

- `LibOpcodes`
- `Capstone`

# Introspecção.

Tabela : Exemplo do efeito do mecanismo de aleatorização de endereços (ASLR) sobre os módulos dinâmicos em duas inicializações consecutivas do SO.

| Módulo | Endereço | Módulo | Endereço |
|---|---|---|---|
| ntdll.dll | 0xBAF80000 | ntdll.dll | 0x987B0000 |
| KERNEL32.DLL | 0xB9610000 | KERNEL32.DLL | 0x98670000 |
| KERNELBASE.dll | 0xB8190000 | KERNELBASE.dll | 0x958C0000 |
| NETAPI32.dll | 0xB6030000 | NETAPI32.dll | 0x93890000 |

# Introspecção.

Tabela : Exemplos de *offsets* das funções de biblioteca `ntdll.dll`

| **Função** | ***Offset*** |
|---|---|
| NtCreateProcess | 0x3691 |
| NtCreateProcessEx | 0x30B0 |
| NtCreateProfile | 0x36A1 |
| NtCreateProfileEx | 0x36B1 |
| NtCreateResourceManager | 0x36C1 |
| NtCreateSemaphore | 0x36D1 |
| NtCreateSymbolicLinkObject | 0x36E1 |
| NtCreateThread | 0x30C0 |
| NtCreateThreadEx | 0x36F1 |

# Introspecção.

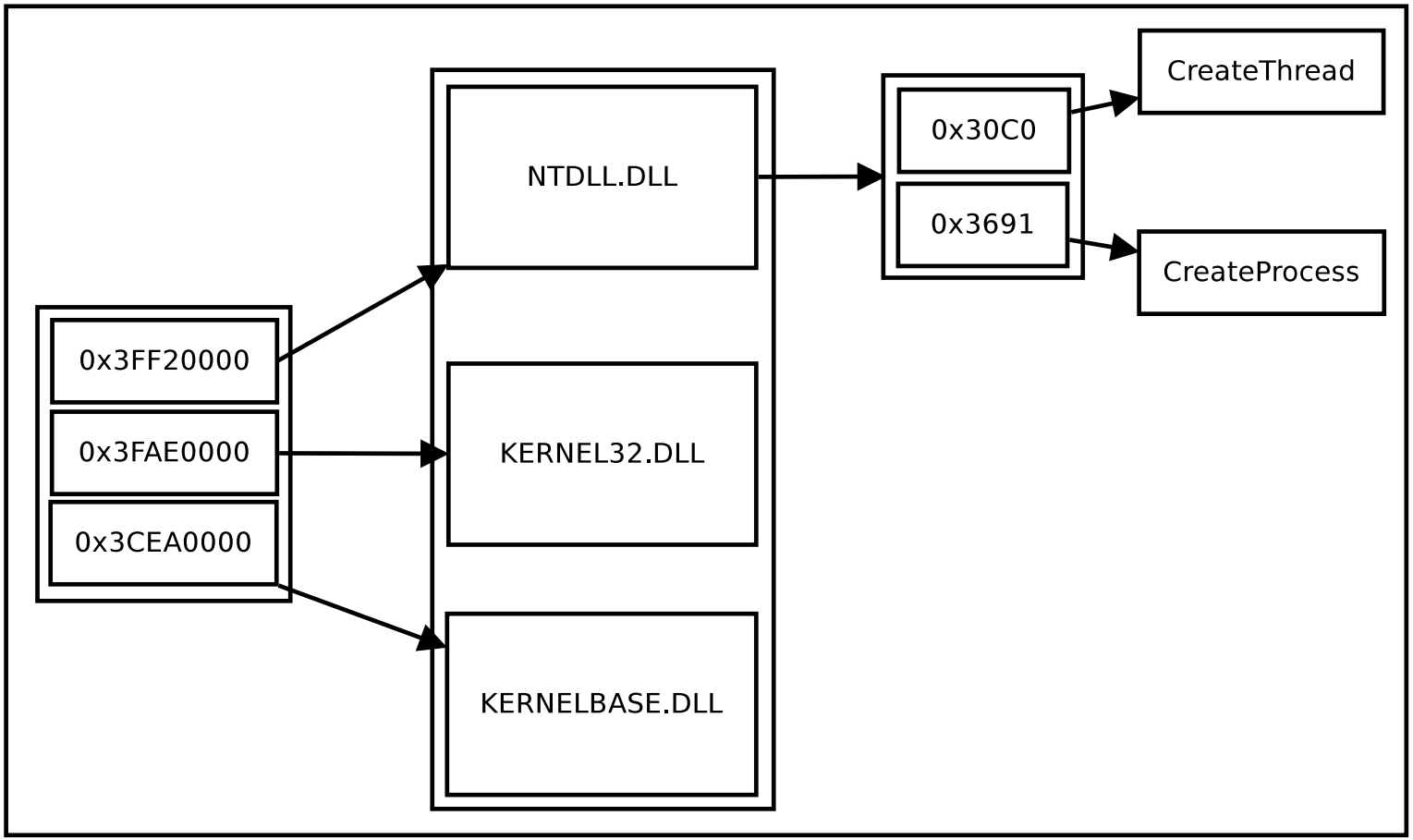


Figura : Diagrama de funcionamento do mecanismo de introspecção para associação de nomes de funções a endereços de módulos.

# *Disassembly*.

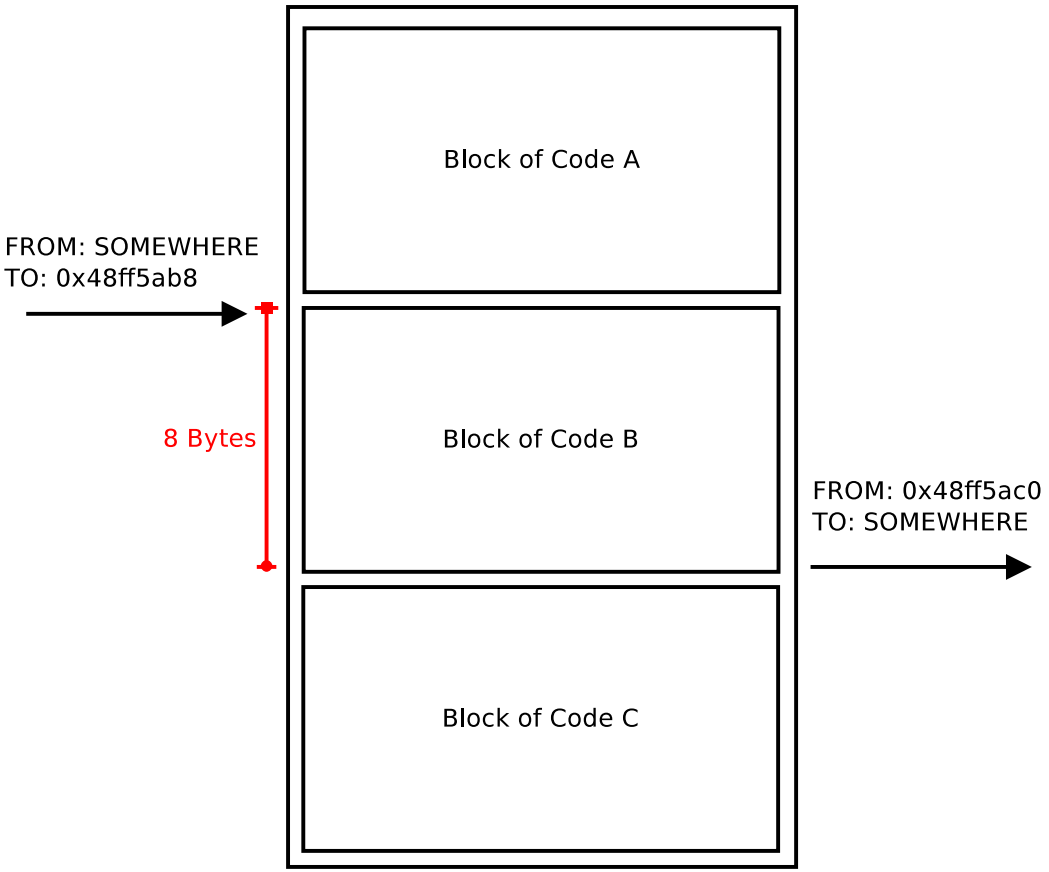


Figura : Identificação de um bloco de código através de duas instruções de desvio consecutivas.

# *Disassembly*.

Listagem 1 : Exemplo de buffer de instruções obtido a partir dos endereços fornecidos pelo mecanismo BTS.

```
1 \xff\x15\x0a\x11\x00\x00\x48\x8d\x0d\x9f\x11\x00\
      x00
```

Listagem 2 : Conversão das instruções do buffer para opcodes.

```
1 0x1000 (size=6)   call    QWORD PTR [rip+0x110a]
2 0x1006 (size=7)   lea     rcx,[rip+0x119f]
```

# Tópicos

# Testes - CG.

Listagem 3 : Código de exemplo para a reconstrução do CG.

```
1 scanf("%d",&n);
2 scanf("%s",val);
3 for(i=0;i<n;i++)
4       printf("%s\n",val);
```

# Testes - CG.

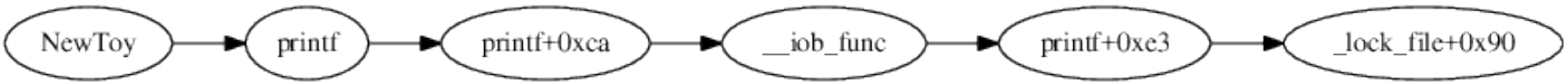


Figura : Visualização completa de trecho do CG.


Figura : Visualização do CG reduzido.

# Testes - CFG.

Listagem 4 : Código de exemplo para a reconstrução do CFG.

```
1  a=0;
2  scanf("%d",&n);
3  for(i=0;i<n;i++)
4        if(i%2==0)
5              a++
6        else
7              a--
8        printf("%d\n",a)
```

# Testes - CFG.

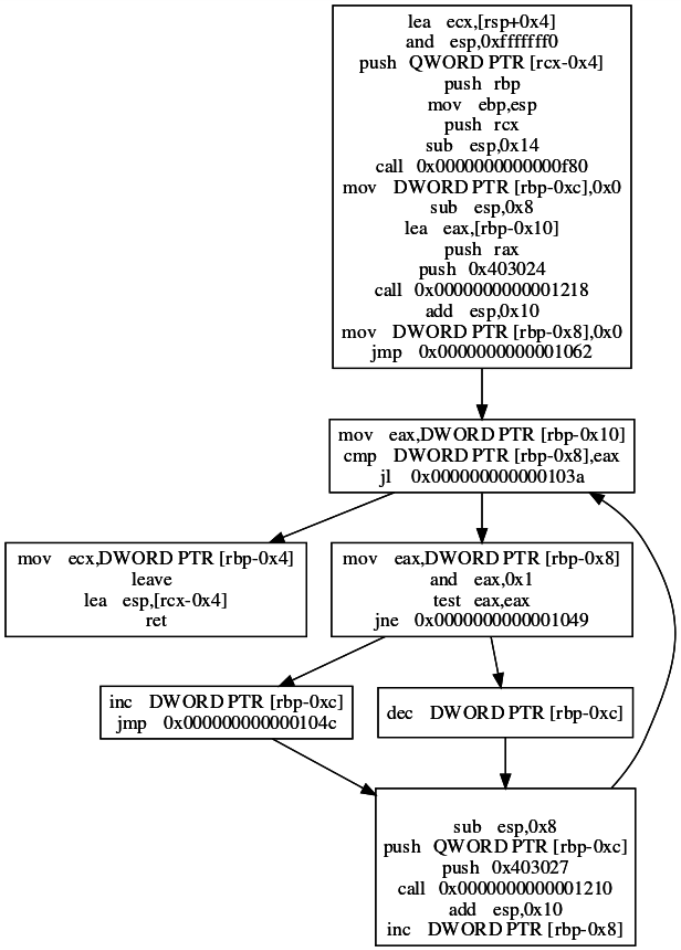


Figura : CFG reconstruído a partir da execução do código de exemplo.

# Exemplares Reais.

Tabela : Comparação entre exemplares evasivos reais em execução em nossa solução e em outra sandbox.

| Amostra (MD5) | BehEMOT | Esta Solução |
|---|---|---|
| f03c0df1f046197019e12f3b41ad8fb2 | ✗ | ✓ |
| 2b647bdf374a2d047561212c603f54ea | ✗ | ✓ |
| 7a4b29df077d16c1c186f57403a94356 | ✗ | ✓ |
| 340573dd85cf72cdce68c9ddf7abcce6 | ✗ | ✓ |

# Discussão.

## CFG

- *Branch* determina o bloco.
- Desvios reais - não é afetado por desalinhamento.
- *Online Disassembly* permite avaliar código gerado em *runtime*.

## Desafios

- Nível de Abstração (*semantic gap*).
- Interpretação de desvios não tomados.
- Perda do controle quando em *kernel*.

# Discussão.

## Portabilidade

- Solução Portável.
- Recurso do processador.
- Técnicas independentes de plataforma.
- Depende de Bibliotecas.

# *Overhead.*

## Habilitação do Mecanismo

- 1%.

## Coleta de dados

- 14%.

## Introspecção

- 26%.

## *Disassembly*

- 26%.

# *Overhead*.

## Comparação

- Esta Solução[a]: 43%.
- Ether: 72%.
- MAVMM: 100%.

[a] *disassembly offline*

## Reduzindo o *Overhead*

- *Disassembly offline*.
- Uso de múltiplos núcleos.
- Uso de banco de dados de código.

# Tópicos

# Análise dos Resultados

## Sumarização

- Exemplares aplicam técnicas de anti-análise.
- Soluções precisam ser transparentes.
- Mecanismos existentes tem grande custo de desenvolvimento.
- Mecanismo de monitoração de *Branch* pode ser utilizado.
- Análise é transparente.
- Custo de desenvolvimento é baixo (*kernel driver*).
- *Overhead* comparável ao estado-da-arte.
- Validação com exemplares reais.

# Limitações e Trabalhos Futuros

## Limitações

- Análise em *kernel*.
- Uso de APIs do sistema.
- Maior granularidade do que HVM e SMM.

## Trabalhos Futuros

- Aplicação a clusterização de exemplares de *malware* evasivos.

# Tópicos

# Conclusões

## Conclusões

- Mecanismos de análise transparente precisam ser desenvolvidos.
- Uso dos monitores de performance como meio transparente.
- Reconstrução de fluxos (CG e CFG) é viável.

# Agradecimentos


- CNPq, pelo financiamento via Proj. MCTI/CNPq/Universal-A edital 14/2014 (Processo 444487/2014-0)
- CAPES, pelo financiamento via Proj. FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69).
- Instituto de Computação/Unicamp
- Departamento de Informática/UFPR


## Contato:

marcus@lasca.ic.unicamp.br
paulo@lasca.ic.unicamp.br
gregio@inf.ufpr.br