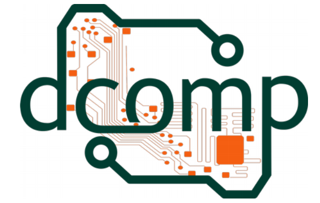
Universidade Federal do Espírito Santo – CCENS UFES

Universidade Federal do Espírito Santo
Centro de Ciências Agrárias – CCENS UFES
Departamento de Computação



Análise de Malware

Computação Forense
Site: http://jeiks.net
E-mail: jacsonrcsilva@gmail.com

Introdução

- *Malware* é proveniente do Inglês: ***Mal**icious Soft**ware**.*
- É um software criado:
  - Para se infiltrar em um sistema computacional alheio de forma ilícita, e
  - Para causar danos, alterações ou roubo de informações (confidenciais ou não).
- Exemplos:
  - Vírus de computador,
  - Worms,
  - Trojan (cavalos de troia) e
  - Spywares.

Análise

- Maneiras de estudar um programa:
  - Análise estática:
    - Estudar o programa sem executá-lo.
  - Análise dinâmica:
    - Estudar o programa a medida que ele executa.
  - Análise *post-mortem*:
    - Estudar os efeitos após a execução do programa.

Análise estática

- Para estudar um programa antes de executá-lo, é necessário utilizar ferramentas como:
  - Desasembladores;
  - Descompiladores;
  - Analisadores de código fonte;
  - Até mesmo utilitários básicos, como strings e grep.
- A vantagem é a inteireza:
  - Revelação de partes de um programa que geralmente não são executadas.
- É possível obter um cenário aproximado no melhor dos casos, prevendo até mesmo todo o comportamento do software.

Análise Dinâmica

- Para estudar um programa a medida que ele executa, é necessário utilizar ferramentas como:
  - Depuradores;
  - Rastreadores de chamada de função;
  - Emuladores de máquina;
  - Analisadores lógicos; e
  - *Sniffers* (Analisadores de rede).
- Tem a vantagem da velocidade de obter informações.
- Desvantagens/Dificuldades:
  - "o que você vê é tudo o que você tem".
  - É impossível prever o comportamento de um programa não trivial;
  - É impossível fazer com que o programa corra todos os caminhos do seu código.

Análise Dinâmica

- Uma opção é a análise “caixa preta”:
  - O programa é analisado sem um conhecimento de seus aspectos internos.
  - Os únicos itens observados são:
    - Entradas;
    - Saídas; e
    - Seus relacionamentos de tempo.
  - As entradas e saídas podem até mesmo ser *consumo de energia* e *radiação eletromagnética* gerada.

Análise Post-mortem

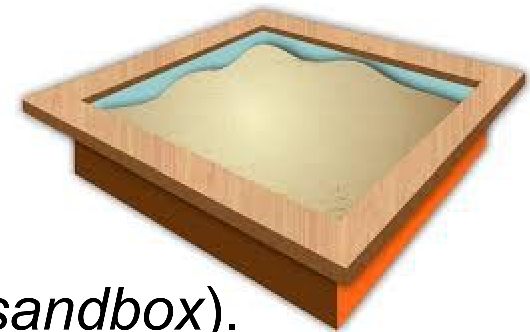
- A análise de um programa após sua execução pode ser feita com a análise de:
  - Inclusão de um login local ou remoto;
  - Alterações no conteúdo de um arquivo;
  - Alterações nos padrões de data/hora de acesso ao arquivo;
  - Informações sobre m arquivo excluído;
  - Informações de dados gravados na swap;
  - Informações de dados que ainda subsistem na memória;
  - E informações que foram registradas fora da máquina.

Análise Post-mortem

- Esta análise é na maioria das vezes a única ferramenta disponível depois de um incidente.
- A desvantagem é que as informações desaparecem ao longo do tempo.
  - Pois o comportamento normal do sistema apaga os vestígios.
- Porém:
  - Os efeitos residentes na memória podem durar horas ou dias; e
  - Os efeitos residentes em discos podem durar dias ou semanas.

Os perigos da análise dinâmica

- Ao executar um programa e ver o que ocorre, encontram-se os problemas:
  - O programa poderia destruir todas as informações na máquina; ou
  - Poderia enviar um e-mail ameaçador a pessoas com as quais você não quer se indispor;
  - Ou fazer qualquer outra atividade intrusiva.



- O que deve ser feito é:
  - Executar o programa em uma “caixa de areia” (*sandbox*).
- Um *sandbox* de software é um ambiente controlado para executar o software, podendo ser:
  - Uma máquina real exclusiva para isso na rede; ou
  - Uma máquina virtual.

Máquinas Virtuais

- Baseadas em Hardware
  - Máquinas sofisticadas com multiprocessadores, sendo capazes de dividir uma máquina em algumas pequenas partições no nível do hardware.
  - Possuem valor elevado.
- Baseadas em Software
  - Maneira flexível de compartilhar hardware entre múltiplos sistemas operacionais simultaneamente em execução;
  - O programa monitor da máquina virtual faz a mediação do acesso com o hardware real;
  - Podem sobrecarregar a máquina real e dar problemas de execução;
  - Permitem que um investigador, pause, retroceda ou avance um incidente.

Perigos de uma Máquina Virtual Baseada em Software

- Manter um software confinado em uma máquina virtual
  - Exige completo isolamento do processador e demais dispositivos.
- Porém, se o *malware* conhecer o ambiente virtal,
  - ele pode explorar *bugs* na implementação do monitor virtual.
  - Assim, ele pode sair da máquina virtual e afetar o sistema real.

Perigos de uma Máquina Virtual Baseada em Software

- Exemplo 1:
  - Uma trilha de setores pode estar contínua no disco da máquina virtual, porém em diferentes setores do disco.
  - Assim, se o convidado tiver o tempo preciso de acesso, ele pode notar tempos de acesso incomuns aos setores “sequenciais”.
- Exemplo 2:
  - Comando “dmesg”:

    …

    *acd0: CDROM <**VMware** Virtual IDE CDROM Drive>...*

    ...
  - Comando “ifconfig lnc0”:

    ...

    ifnet6 fe80::250:56ff:fe**10:bd03**%1e0 prefixlen 64 scopeid 0x1

    ...

Outros confinamentos

- Chroot
  - Isolam somente o sistema de arquivos;
  - O mesmo kernel é compartilhado entre o sistema real e os confinamentos;
- Prisões
  - Possui melhorias no *chroot* que restringem seu acesso às dispositivos e memória.

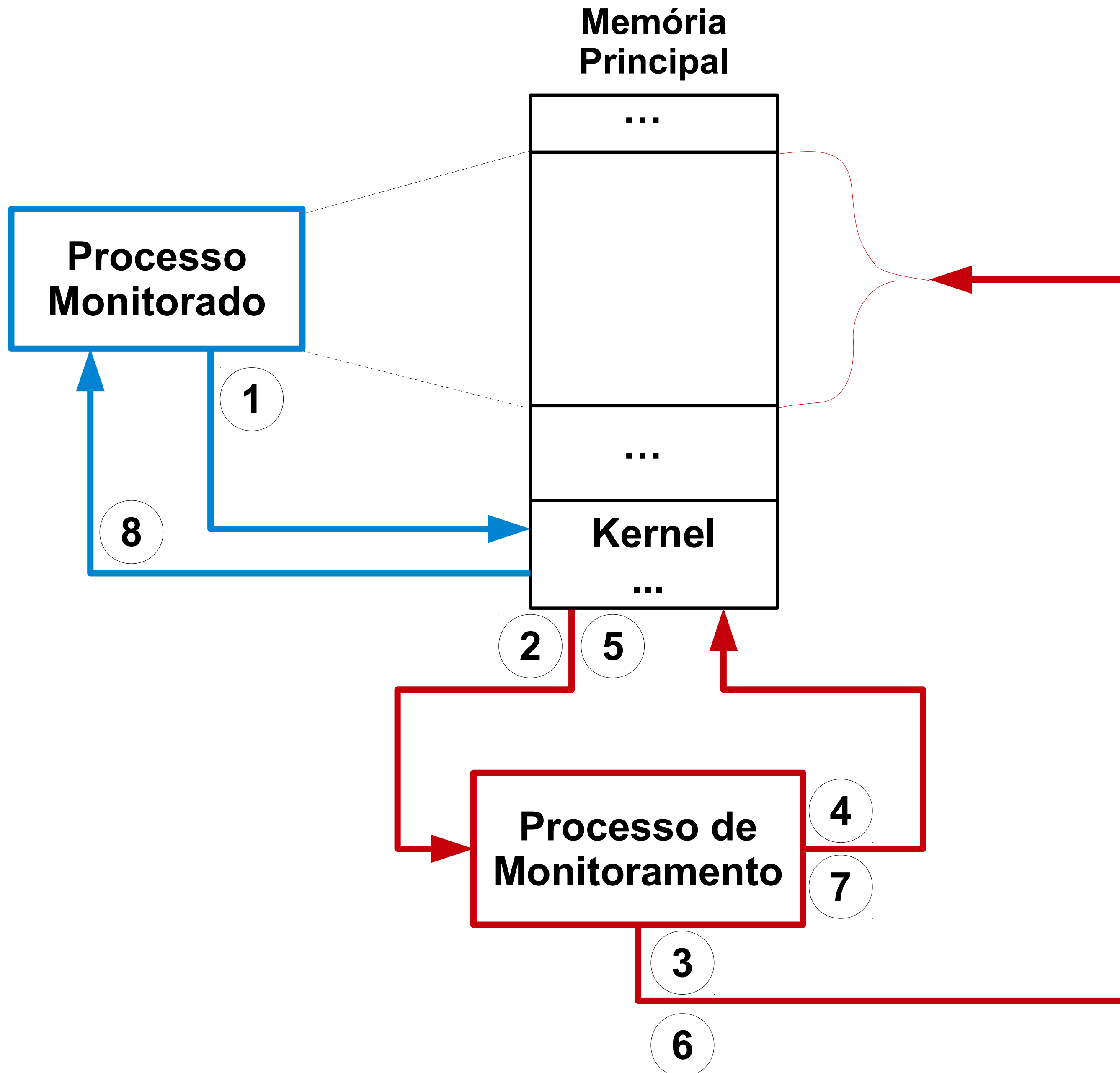
Análise dinâmica

- Podem ser utilizados **monitores de chamada de sistema**:
  - strace: para Linux, FreeBSD, Solaris, …
  - truss: para Solaris.

Funcionamento

1. O processo monitorado invoca uma chamada de sistema.
2. O kernel passa o controle ao processo de monitoramento. Ele monitora o processo monitorado, com acesso aos registradores, memória, etc.
3. O kernel executa a chamada de sistema.
4. Ao terminar a chamada de sistema, o processo de monitoramento pode inspecionar o processo monitorado novamente, verificando sua memória, registradores e os resultados da chamada de sistema.
5. O kernel passa o controle de volta para o processo monitorado.

Funcionamento



Trabalhando com strace

Crie o arquivo teste.py:

```
#!/usr/bin/python
f = open('/tmp/arquivo.txt', 'w')
f.write('Oi mundo, me entenda!\n')
f.close()
```

Execute-o e veja o que ele faz.

Agora, execute-o com o seguinte comando:

```
strace -e trace=open,write,close "./teste.py"
```

Recolha então suas chamadas de sistema e explique-as.

Agora, que tal trabalhar com outros arquivos do sistema?

```
strace -f -p PID -e trace=read,write -e write=3 -e read=5
./call_strace.sh [-p <pid> | -n <name>]
```

Confinamento com censores

- Outra forma de monitoramento é com a implantação de ganchos de monitoramento de chamada de sistema,
  - Permitindo restringir as ações de um processo monitorado.
  - Evitando assim danos ao sistema.
  - Podem trabalhar a nível do usuário ou a nível do kernel.
- Censores de chamada de sistema:
  - Nível do usuário: Janus;
  - Nível de kernel: Systrace.

Configurações...

- Janus:

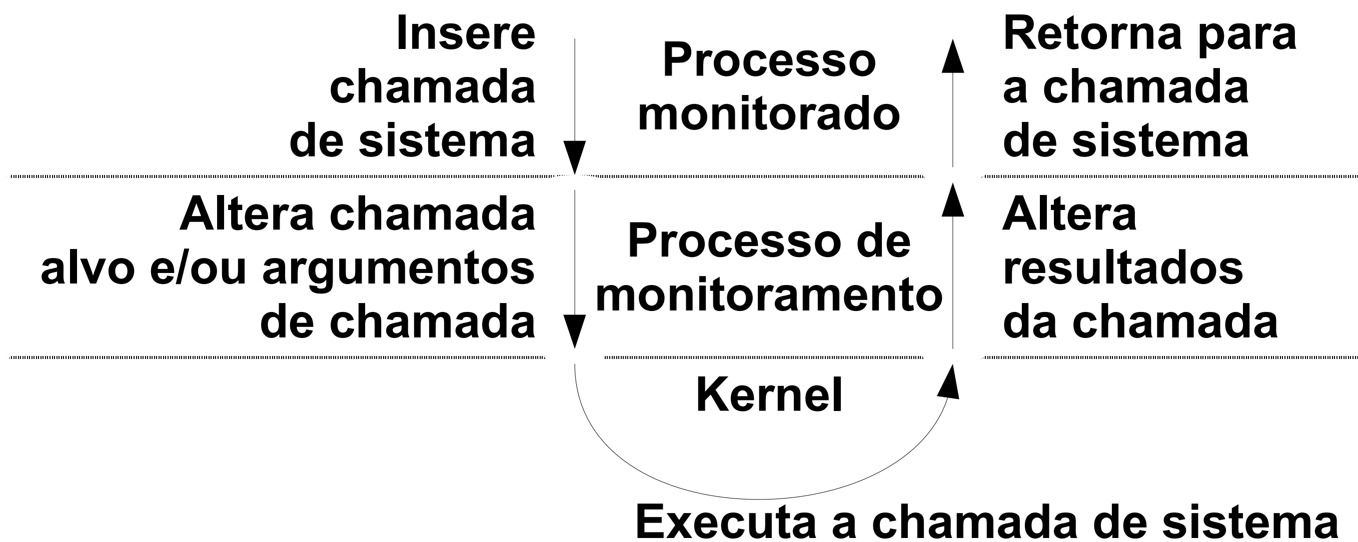
```
# diretório inicial
starting_dir /home/malware
# permite acesso de leitura ao "passwd"
path allow read /etc/passwd
# Restringe conexões
net allow connect tcp 192.168.0.1 80
```

- Systrace:

```
# Permite o stat(), lstat(), readlink(), access(), open() para leitura
native-fsread: filename eq "$HOME" then permit
# Permite conexões com qualquer servidor WWW
native-connect: sockaddr match "inet-*:80" then permit
```

Spoofing de chamada de sistema

- Evitar algumas chamadas de sistemas é interessante, porém isso evita saber qual foi o dano que o software tentou gerar.
- A alternativa é permitir que o dano aconteça, mas sem efeitos permanentes.
- Uma forma interessante é utilizar um spoofing, que falsifica a chamada de sistema.



Pseudocódigo de um spoofing

```
filho = produzir_filho( comando );

retorno_spoof = 0;

while (1)
{
    espera_pelo_filho( filho );
    if (retorno_spoof == 0)
    {
        numero_syscall = ler_registrador( filho, ORIG_EAX );
        if ( numero_syscall == SYS_fork )
        {
            escrever_registrador( filho, ORIG_EAX, SYS_getpid );
            retorno_spoof = 1;
        } else {
            escrever_registrador( filho, EAX, 0);
            retorno_spoof = 0;
        }
    }
}
```

Análise dinâmica de biblioteca

- Também pode-se utilizar a análise de bibliotecas.
- Aplicativo utilizado: ltrace;
- Exemplo:

```
ltrace date 2>&1 > /dev/null | less
```

Confinamento com chamada de biblioteca

Vamos trabalhar com o arquivo backdoor.c
Não olhe seu código agora, simplesmente o compile-o.

Agora, vamos ver o que conseguimos identificar com o gdb:

```
gdb backdoor.e
(gdb) disassemble main
```

Tente identificar/encontrar suas funções e as funções externas.

Agora, execute o seguinte comando:

```
objdump --dynamic-syms backdoor.e | grep UND
```

Depois, execute esses comando:

```
nm -op backdoor.e | grep ' U '
```

Qual saída esses comandos nos fornece?

Vamos trocar a função externa agora para analisar o programa?

Arquivo "strstr.c"

Perigos com monitoramento de chamadas de bibliotecas

- Se o programa não for programado seguindo as regras padrões, ele pode
  - Burlar a chamadas de bibliotecas externas;
  - Burlar o monitoramento;
  - Utilizar de estouro de buffer;

Outras medidas de análise

- Ainda existem técnicas para pessoas altamente motivadas.
- Utilizando:
  - Desassemblagem de programa;
  - Descompilação de programa;
  - Análise estática;
- Podem fazer engenharia reversa, ou seja, recuperar/reconstruir o código fonte através da análise de seu código assembly.

Últimas notas

- Alguns programas não seguem as regras, pois possuem intuito de complicar a análise do programa.
- Algumas dos casos comuns são:
  - Um programa não será descompilado em código de alto nível se não tiver sido gerado por um compilador de alto nível;
  - Um programa pode ter o código embolado por um ofuscador de código;
  - Um arquivo pode ter seu código malicioso criptografado dentro de si, sendo necessário conhecer:
    - O método utilizado para criptografar o código (ex.: Burneye);
    - A chave utilizada na criptografia.
  - Possuir código "*automodificável*". Sendo comum em vírus, código que salta para executar o conteúdo dos dados e estouro de buffer.