Proteção de dados sensíveis através do isolamento de processos arbitrários no kernel Linux.

Otávio Augusto Araújo Silva

Universidade Estadual de Campinas

31 de Maio de 2016

Agenda

1. Introdução
   - Contexto
2. Trabalhos relacionados
   - PrivExec
   - Linux Security Modules
   - GrSecurity
3. Mecanismo Proposto
4. Perguntas

1 Introdução
- Contexto

2 Trabalhos relacionados

3 Mecanismo Proposto

4 Perguntas

Introdução

Parte dos *softwares* altamente utilizados possui, em alguma etapa do desenvolvimento, a busca e resolução de falhas de segurança.

Uso de *security frameworks* para fornecer integridade e determinismo durante a execução dos programas, provendo um ambiente mais seguro contra ataques.

Ambientes resistentes contra tentativas de violação à segurança, podem ser comprometidos mesmo sem vulnerabilidades nos *softwares*.

Problema

Vazamento de informações privilegiadas através de brechas na interação baseada na confiança entre os processos do S.O.

Contexto do problema

Necessidade de eficiência e desempenho dos sistemas operacionais modernos.

Grande visibilidade e confiança que várias instâncias do sistema operacional compartilham.

Violação não é, em geral, causada por bugs no sistema operacional, mas como consequência da otimização de recursos.

Contexto do problema

Compartilhamento de memória, permissões no sistema de arquivos persistente(ext4, fat32 ...) ou volátil(ramfs, procfs...) etc., envolvendo processos de um mesmo usuário.

Ex. vazamento

Um processo descobre quais áreas de memória um outro processo(mesmo usuário) está utilizando(procfs), assim reduzindo ou anulando, a complexidade do *ASLR(Address Layout Randomization)*.

1 Introdução

2 Trabalhos relacionados
- PrivExec
- Linux Security Modules
- GrSecurity

3 Mecanismo Proposto

4 Perguntas

Trabalhos relacionados

Métodos

1. PrivExec
2. LSM- AppArmor e SELinux
3. Grsecurity

Foram analisados métodos que poderiam resolver algumas instâncias do problema.
Eles representam os principais mecanismos para o isolamento de processos, e aos problemas de segurança que cada um daqueles métodos se propõem a evitar ou dificultar.

PrivExec

Overview

Utiliza uma distinção lógica entre os processos, os públicos e os privados, com uma chave secreta na *task struct*.

Os processos e os grupos privados estão sujeitos a restrições especiais para impedir a divulgação de dados sensíveis resultantes de sua execução.

Um grupo privado é aquele que explicitamente compartilha uma mesma chave na *task struct*.

- Em maioria para resolução de dependências entre processos.

PrivExec

Isolamento no Armazenamento

O isolamento ocorre pela cifragem dos arquivos assim que abertos para escrita pelo processo em execução.

Utiliza uma versão alterada do *eCryptfs*, destinado a cifrar partições inteiras como um único *secure container*.

Uso do *union file system*, com o *OverlayFS*, fazendo a junção do *secure container*, e sua árvore de diretórios, com o *F.S root*.

Isolamento no Swap

O *kernel* do Linux trata os *node* de *swap* diferentemente dos dispositivos de blocos.

As rotinas de *swap* foram alteradas para cifrar e decifrar as páginas de troca, usando a chave secreta de cada processo.

Caso haja o *swap*, as páginas irão cifradas e, no retorno ,decifradas para a memória.

PrivExec

Isolamento IPC

Chave do processo usada para controlar o ponto final da comunicação, antes da análise de permissões feita pelo *kernel*.

Livre comunicação no grupo privado, já outros processos recebem erro de permissão na leitura/escrita.

Modificações apenas nos protocolos de *IPC*:*UNIX SysV & POSIX shared memory, message queues* e *UNIX domain sockets*.

PrivExec

Restrições

- Provê isolamento apenas nos protocolos de *IPC* mais comuns.
- Não garante isolamento contra leitura direta de sua memória por outros processos.
- Arquitetura ainda não resistente a ataques e *malware*.
- Grandes grupos de processos para resolução de dependência. Por exemplo, várias instâncias do *X11 display*.

Overview

O kernel Linux fornece uma *framework* para segurança que permite a monitoração e controle do acesso aos objetos do *kernel* ou das *tasks*.

Esse controle é fornecido por diversos pontos de entrada, e *hooks*, ao longo do código do *kernel*. Permitindo que mecanismos de acesso sejam aplicados a objetos específicos.

Esses objetos podem variar de arquivos no disco até *kernel capabilities*, como a capacidade de interagir com *sockets*.

Linux Security Modules

Através desse *framework* é possível registrar um módulo no *kernel*, que utilizará a interface do LSM para aplicar a própria heurística nos objetos e subsistemas do *kernel*. Criando assim um *security framework*.

Exemplos de *security frameworks* que utilizam o LSM são o SELinux e AppArmor. Esses dois sistemas implementam um controle de acesso por restrição (MAC).

E assim incrementam o gerenciamento do acesso e as permissões já existentes no Linux, ao atribuir aos processos *labels* ou *tags* relacionadas às novas permissões.

Linux Security Modules

SELinux

O *SELinux* segue o modelo do menor privilégio: tudo é negado por padrão, e uma série de exceções nas políticas permitem o necessário;

As políticas são especificas para cada executável, que são identificados pelos seus respectivos *inodes*.

Estas são compiladas e carregadas pelo módulo de *kernel* para serem usadas, e aplicadas em dois regimes: *Enforcing* e *Permissive*.

No modo *Enforcing*: toda ação que infringe uma política de segurança é bloqueada e reportada como um evento logado.

No modo *Permissive*, os eventos são apenas reportados para serem logados, e nenhuma outra ação é tomada.

AppArmor

AppArmor utiliza *profiles* para gerenciar as regras de cada processo, que são válidas para quaisquer processos cujo executável corresponda ao *path* específico registrado.

Nos *profiles* estão contidas todas as políticas de acesso, objetos controlados e mecanismo de acesso disponível para o executável em questão.

Existem dois modos de operação, *Complain* e *Enforce*: o primeiro registra violações às regras, e o outro rejeita o acesso aos objetos controlados.

Linux Security Modules

Apesar das diferenças na configuração, aplicação dos *labels* ou identificação dos objetos e processos protegidos, os *frameworks* que usam o LSM são equivalentes em termos de segurança fornecida.

Limitações

- Proteção dependente das políticas; novas vulnerabilidades, novas políticas.
- Restrições aplicadas apenas a objetos críticos; *file system*, IPC(*sockets, queues etc.*), *kernel capabilities*.
- Não essencialmente privacidade dos dados do processo; pai-filho ou processos de um mesmo executável, mesmas permissões,
- Processos não controlados(sem *tags/labels* etc.) podem manipular dados de processos controlados(*race conditions*, *memory leaks*)

GrSecurity

Overview

Abordagem diferente da anterior, pois ele faz seus próprios *hooks* e pontos de entrada no *kernel*.

Grsecurity possui um conjunto de *patches* do *kernel*, fazendo mudanças em quase todos subsistemas.

Assim protegendo todo o sistema contra *exploits* ou ataques inerentes ao *kernel(jitspray)*.

Essa abordagem protege tanto o nível de usuário(restrição ao acesso), quanto o nível de *kernel*(escalação de privilégios, ataques *etc.*).

GrSecurity

Grsecurity aplica aprendizado de máquina com regras baseadas em RBAC(Role-Based Access Control), para realizar controle de acesso aos objetos.

O *PaX*, responsável pela gerência de segurança em nível de usuário, possui dois modos de operação: aprendizado e produção.

No modo aprendizado, o sistema é tido como fora de um ambiente de ataque, onde as operações dos processos são ditas confiáveis e farão parte da base de conhecimento.

No modo produção, todo o comportamento extraído e pertencente a base de conhecimento é usado para regrar o acesso aos objetos.

GrSecurity

O Grsecurity prove diversas *features* de segurança não presentes nos outros sistemas estudados.

- Highest performance and most secure ROP defense.
- Bounds checks on kernel copies to/from userland.
- Prevents direct userland access by kernel.
- Prevents kernel stack overflows on 64-bit architectures.
- Hardened userland memory permissions.
- Random padding between thread stacks.
- Hardened BPF JIT against spray attacks.
- Automatically responds to exploit bruteforcing.

GrSecurity

Limitações

- Necessita de um *kernel* especificamente compilado(diversos *patchs*).
- Processos de um mesmo executável compartilham permissões similares.
- O *security framework* foi construído para evitar e proteger contra ataques, não preservar privacidade dos dados.
- Dois processos de um mesmo usuário, tem iguais potenciais de vazarem dados entre si(*file system, IPC etc.*).

1 Introdução

2 Trabalhos relacionados

3 Mecanismo Proposto

4 Perguntas

OSPI

Overview

Objetivo: complementar a proteção provida pelo *Grsecurity*, ao resolver algumas de suas limitações, principalmente quato a base de confiança do sistema operacional.

O *OSPI(Operating system Security Process Isolator)*, é um módulo do *kernel Linux* para arquitetura *Intel* com suporte à extensão AES-NI.

Sua arquitetura permite aplicar a política segurança: um objeto manipulado por um processo, em nível de usuário ou não, é exclusivamente de sua propriedade ou compartilhado entre um grupo predefinido de processos.

OSPI

O controle de acesso não utiliza os locais dos objetos(*path, inode etc.*), mas as informações de posse presentes nas estruturas do módulo.

O OSPI decide como um recurso será compartilhado de acordo com a máscara de acesso do primeiro processo que obtiver a posse do objeto.

Posse é atribuída através da relação entre processo controlado e objeto a ser manipulado, dado que tal objeto não foi manipulado por nenhum outro processo ou grupo de processos até então.

OSPI

Estruturas de controle

- Lista de processos controlados.
- Lista global de estruturas do *kernel* controladas.
- Lista de símbolos do *kernel* por processo.
- Máscaras de bits por processo.

Mascara de bits

```
#define MASK_SIZE 128      // [control(2) task_pid(22) task_group_pid(22)    ]  [mask_perm(82)]
                           //    00 add    task_pid, task_group_pid ignored              ^
                           //    01 del    task_pid, ''             '' ignored           |
                           //    10 add    task_group_pid into task_pid group            |
                           //    11 del    task_group_pid from task_pid group            |
                           // [syscall_number(9) + kobj_list_pointer(64) + mask_access_controll(9)]
```

OSPI

O controle é feito no acesso da camada de *syscall(hookings)*, fornecendo uma interface simples para o controle sobre cada processo no sistema.

Um processo ao manipular qualquer objeto, passará primeiro pelo OSPI, já que este possui *hooks* em *syscalls* como: *open, read, write, exec, mmap, etc*.

Assim verificasse se há manipulação de objetos controlados, aplicando-se restrições se necessário, e ao final retornando para a *syscall* original.

Deste ponto em diante, o *Grsecurity* pode aplicar suas políticas de segurança, de modo que o OSPI fortalece aquele subsistema de segurança.

OSPI

Dentro do *hook*, o módulo pode atuar de duas maneiras distintas: *managed* ou *restricted*.

Managed: as políticas são aplicadas de acordo com um sistema *MAC* aos objetos(processo v.s propriedade).

Neste modo também é possível o uso de um objeto virtual, tratando-se de arquivos ou *buffers* geridos pelo *kernel*.

Esta execução efêmera permite que todas as alterações são confinadas na memória do *kernel*, até o fim da execução do processo.

OSPI

O modo *restricted* garante a privacidade e segurança de uma forma intrusiva: de/criptografa os dados ao ler/escrever, de todos os objetos tocados por um determinado processo.

Impede que outros processos, independentemente de sua permissão, consigam ler dados em texto claro de um processo executado no modo restrito.

A cifragem e decifragem dos dados ocorrem em *syscalls* como *read/write*, ou *msgget/msgsnd(IPC)*.

Essas operações utilizam o `criptossistema AES128` do *Linux CriptoAPI*, e diretamente as instruções *Intel AES-NI*.

OSPI

A chave no *criptossistema* é composta(XOR) de duas outras chaves:

- Chave do OSPI, nunca persistente na memória principal.
- Chave privada, única para cada processo controlado.

A geração da chave privada pode ocorrer de duas formas:

- *Hash MD5* do ELF relativo ao executável do processo.
- *Hash* das informações na *task_struct* do processo.

O hash obtido é capaz de representar todo o código disponível a ser executado pelo processo.

OSPI

Garante que um processo controlado consigo ler seus dados mesmo quando a máquina reinicie e todos os identificadores do processo mudem.

Caso a execução não tenha o bit de persistência; semente pseudo-aleatória é usada como *salt* no *hash*.

Tornando possível que processos de um mesmo executável mantenham instâncias privadas entre si; mesmo executável, mas *hashs* diferentes.

A chave do criptossistema é recalculada(XOR) a cada operação, e assim nunca reside a memória principal; apenas registradores.

OSPI

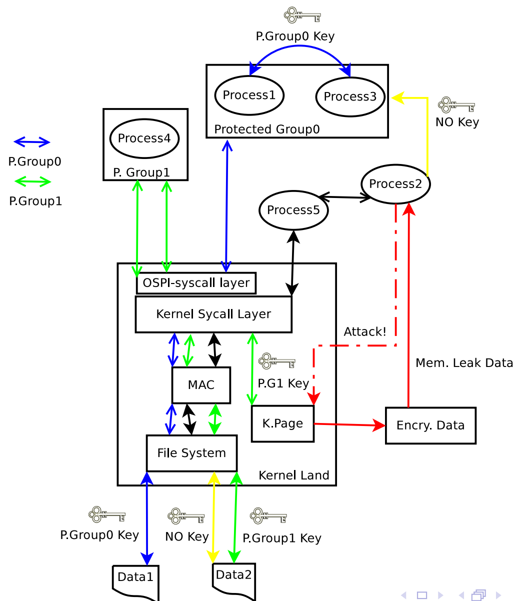
A chave do OSPI é utilizada para todos os processos, mas não é gerada pelo mesmo.

Ela é lida nas primeiras fases do *boot* por meio de dispositivos como: armazenamento de confiança, *hardware* gerador.

Armazenada nos registradores de *debug* do processador: DR0 ao DR3(desativados para uso do usuário).

Cada buffer usado para transporte da chave do OSPI até os registrados é preenchido com 0(zero).

OSPI - Interações



OSPI - Testes

O OSPI foi capaz de evitar ataques ignorados pelos *security frameworks* até então:

- Docker: docker container breakout.
- Sudo ($\leq$ 1.8.14) - Unauthorized Privilege (CVE-2015-5602).

Evitou que os seguintes ataques que causaram *memory leak*, mesmo sobe a proteção do GrSecurity, funcionassem:

- *Diamorphine rootkit signal 63 PID*: causa um *memory dump* para */tmp/PID.dmp*.
- *Cold Boot Attack*: congelamento das memórias principais, para *dump* em seguida.

OSPI - Testes

Overhead

Tabela : Tabela comparativa com o *turnaround time* em milissegundos das *syscalls* protegidas em comparação à chamada sem proteção (base). As medidas de *overhead* correspondem à porcentagem do *turnaround time* da chamada protegidaem relação à chamada desprotegida.

Syscall	Base	OSPI	Overhead	Grsecurity	Overhead	Apparmor	Overhead
null I/O	0,13	0,13	1%	0,13	5%	0,13	12%
open/close	2,05	2,05	4%	2,09	2%	4,28	109%
fork	131,17	131,43	2%	130,23	1%	134,71	27%
execve	386,52	575,91	49%	552,72	43%	587,51	52%
pipe	6,67	7,33	10%	7,67	15%	7,53	13%
read	8,38	11,98	43%	9,30	11%	9,14	9%
write	14,09	22,12	57%	15,49	10%	15,27	8%

Perguntas

Obrigado.