электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.056.55:004.418:004.428

Разработка метода эффективного изменения модели безопасности сетей специального назначения при помощи модулей безопасностей Linux

Батуро А.Л.[1], Садовников Д.С.[2]

Научный руководитель: Старчак С.Л., д.т.н., доцент, профессор отдела № 1 УВЦ ВИ, МГТУ им. Н.Э. Баумана, г. Москва, Россия

Научный консультант:Морозов Ю.В., к.т.н., начальник отдела, НИЦ РКО 4 ЦНИИ МО РФ, г. Москва, Россия

МГТУ им. Н.Э. Баумана

Dima.Sadovnikov@gmail.com

Введение.

Linux Security Modules (LSM) — фреймворк, добавляющий в Linux поддержку различных моделей безопасности. LSM является частью ядра начиная с Linux версии 2.6. На данный момент в ядре находятся модули безопасности SELinux, AppArmor, Tomoyo и Smack. С версии ядра 3.4 был добавлен модуль безопасности Yama.

Модули работают параллельно со встроенной моделью безопасности Linux — избирательным управлением доступом (Discretionary Access Control, DAC). Проверки LSM вызываются на действия, разрешенные DAC. Применять механизм LSM можно по-разному. В большинстве случаев это добавление мандатного управления доступом (как, например, в случае с SELinux). Но главная возможность – добавление в систему поддержки произвольной модели безопасности. Для этого необходимо реализовать ее в виде модуля и внедрить, используя фреймворк.

До версии ядра Linux 2.6.24 была возможность внедрять новые модели безопасности в виде динамически подгружаемых модулей ядра (Linux Kernel Module, LKM). Это позволяло создать очень гибкую модель безопасности всей инфраструктуры, когда выбранные отдельные ЭВМ или даже целые подсети могли иметь различные модели безопасности, диктуемые необходимым уровнем защищенности данных на них.

Например, рядовая подсеть с избирательной моделью доступа, подсеть с усиленной безопасностью, обеспеченной принудительной моделью безопасности и секретная подсеть с разработанной моделью безопасностью, использующей аппаратные токены доступа.

Администратору подобного парка достаточно нескольких минут, чтобы поменять модель безопасности любого сегмента сети или любой отдельной ЭВМ.

Но, как уже было сказано, модули безопасности лишены динамической подгрузки в последних версиях ядра Linux. Это не позволяет создавать сети с гибкими гибридными моделями безопасности, так как теперь для внедрения нестандартной модели безопасности в систему администратор АСУ вынужден статически присоединять модуль к ядру. Для этого необходима компиляция всего ядра и перезагрузка системы, что уменьшает ценность модулей безопасности, разработанных под собственные нужды.

Таким образом, в качестве цели настоящих исследований выбран поиск способа придания модулям безопасности способности к динамической подгрузки в ядро без траты время на перекомпиляцию ядра и перезагрузку всей системы. Кроме этого, основной практической задачей исследований является создание макета сети на основе нестандартных модулей безопасности, ориентированных (заточенных) под собственные конкретные нужды (во время создания макета должен быть разработан хотя бы один модуль безопасности, реализующий модель безопасности отличную от моделей, входящих в стандартную комплектацию ядра Linux).

Постановка задачи разработки метода гибкого и эффективного внедрения произвольной модели безопасности в сеть специального назначения.

В процессе разработки и реализации метода внедрения модели безопасности должны быть пройдены следующие этапы:

1) Изучение принципов работы фреймворка модулей безопасности по исходным кодам ядра ОС Linux.
2) Анализ кодогенерации компилятора для компонентов ядра, отладка в нулевом кольце защиты и обратная инженерия для создания динамически подгружаемого модуля безопасности, реализующего произвольную модель безопасности (не входящую в ядро).
3) Автоматизация процесса изменения модели безопасности в сети, сегменте и отдельной ЭВМ.
4) Создание макета, демонстрирующего работу всех созданных компонентов.

Изучение принципов работы инфраструктуры модулей безопасности.

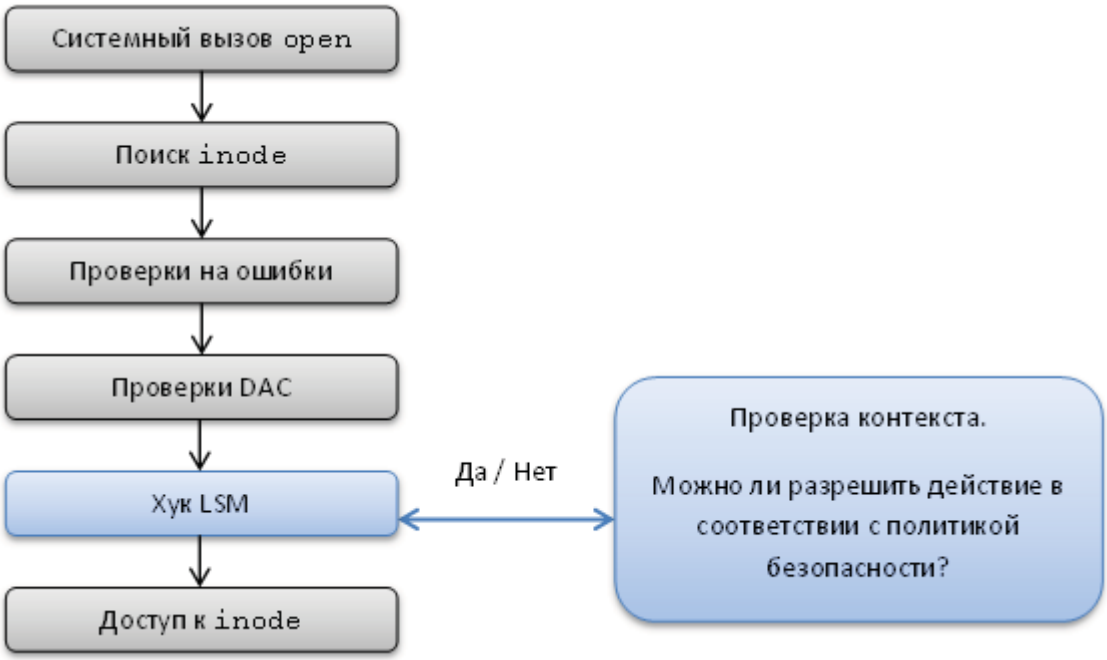

Рис. 1. Работа модуля безопасности на примере перехвата системного вызова open

Инфраструктура модулей безопасности дает возможность ответить на вопрос: «Может ли субъект S произвести действие OP над внутренним объектом ядра OBJ?».

Это достигается за счет помещения вызовов функций проверок непосредственно перед кодом обращения к объекту ядра. В результате модули безопасности могут установить нужные проверки на все действия в системе, которые будут реализовывать требуемую модель безопасности (рисунок).

Свои проверки модули хранят в структуре, содержащие указатели на функции обратного вызова (callback-функции).

Фрагмент описанной структуры:

```
struct security_operations
{
        char name[SECURITY_NAME_MAX + 1];

        int (*ptrace_access_check) (struct task_struct *child, unsigned int mode);
        int (*ptrace_traceme) (struct task_struct *parent);
        int (*capget) (struct task_struct *target,
                kernel_cap_t *effective,
                kernel_cap_t *inheritable, kernel_cap_t *permitted);
```

```
        …
};
```

Зарегистрировать собственный набор проверок модуль может с помощью специальной функции, предоставленной инфраструктурой модулей безопасности.

Исходный код функции регистрации модуля безопасности:

```
/**
 * register_security – регистрирует модуль безопасности в ядре.
 * @ops: указатель на структуру security_options, которая будет использоваться.
 * Если в ядре уже зарегистрирован модуль безопасности, то вернёт ошибку. При успехе вернёт 0.
 */
int __init register_security(struct security_operations *ops)
{
 if (verify(ops)) {
          printk(KERN_DEBUG "%s could not verify "
          "security_operations structure.\n", __func__);
          return -EINVAL;
 }
 if (security_ops != &default_security_ops) return -EAGAIN;
 security_ops = ops;
 return 0;
}
```

Создание динамически подгружаемого модуля безопасности, реализующего произвольную модель безопасности.

В настоящее время добавление новой модели безопасности занимает очень долгое время, так как помимо написания модуля, сборочных файлов (Makefile и Kconfig), правки сборочных конфигурационных файлов ядра необходимо также пересобрать все ядро (для включения кода модуля в код ядра) и перезапустить всю систему для вступления изменений в силу.

Такой подход может быть достаточным для единичных систем и отсутствием жестких сроков внедрения, но для крупных сетей смена модели безопасности в сжатые сроки становится трудновыполнимой задачей, так как необходимо выполнить

вышеописанные действия для всех машин сети. Сюда же можно включить время на настройку конкретных моделей безопасности.

В данном случае разумно сделать модули безопасности динамически подгружаемыми. Этот способ убирает все недостатки модулей безопасности: время на встраивание в ядро, полная пересборка ядра и его модулей, перезапуск системы.

Создание динамически подгружаемой модели было бы простой задачей при наличии нужных символов в экспорте ядра. Так как они были убраны разработчиками, то нужные адреса должен будет искать сам модуль.

После анализа компилятора и дизассемблирования полученного кода и отладки модуля в ядре получен следующий код, реализующий поиск адреса структуры, содержащий функции обратного вызова модулей безопасности:

```
static int address_calculator(struct file *file)
{
        return security_ops->file_alloc_security(file);
}
cp = (const u8*)addr_calculator;
for (i = 0; i < 128; i++)
{
        If (sizeof(security_ops) == sizeof(u32)) {
                if (*(u32*) cp == (u32)&security_ops)
                        break;
        } else if (sizeof(security_ops) == sizeof(u64)) {
                if (*(u64*) cp == (u64) & security_ops)
                        break;
}
        cp++;
}
if (i == 128) {
        printk(KERN_ERR "Can't resolve security_ops structure.\n");
        return -EINVAL;
}
cp = (const u8*)find_symbol(" security_file_alloc\n");
if (!cp)
{
```

```
        printk(KERN_ERR "Can't resolve security_file_alloc().\n");
        return -EINVAL; }
ptr = *(struct security_operations***) (cp + i);
if (!ptr)
{
        printk(KERN_ERR "Can't resolve security_ops structure.\n");
        return -EINVAL;
}
ops = *ptr;
if (!ops)
 {
        printk(KERN_ERR "No security_operations registered.\n");
        return -EINVAL;
}
```

Данный код работает со следующими предпосылками, полученными в результате анализа:

1) Компилятор генерирует одинаковый код для функции ядра security_file_alloc() и функции-двойника address_calculator() в модуле;
2) Структура security_ops модуля будет находится в 128 байтах после функции-двойника address_calculator() даже если будет добавлен дополнительный код (например, отладочные символы);
3) Чтение этих 128 байт не вызовет ошибки;
4) В этих 128 байтах будет, именно, security_ops данного модуля.

Автоматизация процесса внедрения новой модели безопасности в сеть.

Для автоматизации процесса смены были разработаны сценарии, отключающие в системе работающую модель безопасности и подгружающие модуль с новой, или подгружающие новую модель безопасности в качестве дополнительной (в данном случае новая модель дополняет основную модель и проверяет только действия, разрешенные основной).

Применять скрипты можно различными способами, например, через ssh-туннель, дающий администратору сети доступ к подопечным машинам.

Заключение.

В результате проведенных исследований реализован механизм гибкого и эффективного внедрения произвольной необходимой модели безопасности в сети специального назначения, основанный на модулях безопасности.

Метод универсален для всех существующих актуальных версий ядра ОС Linux и подходит для внедрения любой модели безопасности, разработанной под собственные нужды. Код, необходимый для работы метода будет собираться как под 32-х разрядные системы, так и под 64-х разрядные, что делает его применимым к сетям с ЭВМ, имеющим различные типы процессоров.

Разработанные средства автоматизации значительно упрощают работу системного администратора сети по смене модели безопасности всей сети, ее сегмента или ЭВМ, входящей в ее состав.

Список литературы

1. Mark Mitchell, Jeffrey Oldham, Alex Samuel. Advanced Linux Programming. — New Riders Publishing, 2001 – 368 c.
2. Daniel P. Bovet, Marco Cesati – Understanding the Linux Kernel – O'Reilly Media, 2002 – 784 c.
3. James Morris, Greg Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel.
4. Alfred V. Aho, Monica S. Lam, Ravi Sethi. Compilers: Principles, Techniques, and Tools. — Addison-Wesley, 2006.
5. Kris Kaspersky. Hacker Disassembling Uncovered. — A-List Publishing, 2003 – 600c.
6. Поисковые исследования методов обеспечения контроля за потоками информации в распределённых АСУ специального назначения. НТО о НИР «Скаут-АСУ» / Старчак С.Л. Батуро А.Л., Садовников Д.С. и др. – М.: ВИ МГТУ им. Н.Э. Баумана, 2011. – 103 с.
7. Исследование направлений развития, испытания и применения средств вычислительной техники для создания перспективного вооружения и военной техники. Промежуточный НТО по КНИР «Пролог-ВС» / Морозов Ю.В., Батуро А.Л., Садовников Д.С. и др. – М.: НИЦ РКО 4 ЦНИИ МО РФ, 2011. – 184 с.
8. Студенческий научный вестник. Сборник статей докладов общеуниверситетской научно-технической конференции "Студенческая научная весна-2012", посвященный 165-летию Н.Е. Жуковского. 02-09 апреля 2012г., МГТУ им. Н.Э. Баумана/ М.: НТА "АПФН", 2012. (Сер. Профессионал). Т. 12. Часть 5, 476 с., ил.