Гранулярный контроль безопасности поведения приложений со стороны ядра Linux

Денис Гамаюнов, Фёдор Сахаров

лаборатория вычислительных комплексов ВМК МГУ имени М. В. Ломоносова

Контроль поведения приложений на узлах

- Задача – минимизация ущерба от эксплуатации уязвимостей в сетевом ПО
  - Противодействие самораспространяющемуся ВПО
  - Противодействие прямым атакам

Принцип минимальных привилегий

- **В каждый момент времени приложение должно иметь ровно столько полномочий, сколько ему требуется для корректной работы**:
  - Шеллкод выполняется в виртуальном адресном пространстве атакованного приложения
  - Он выполняется с правами атакованного приложения
  - Ограничение прав приложения по доступу к ресурсам до необходимого минимума снижает возможный вред от успешного использования уязвимости
- Примеры: SELinux, AppArmor*, Tomoyo

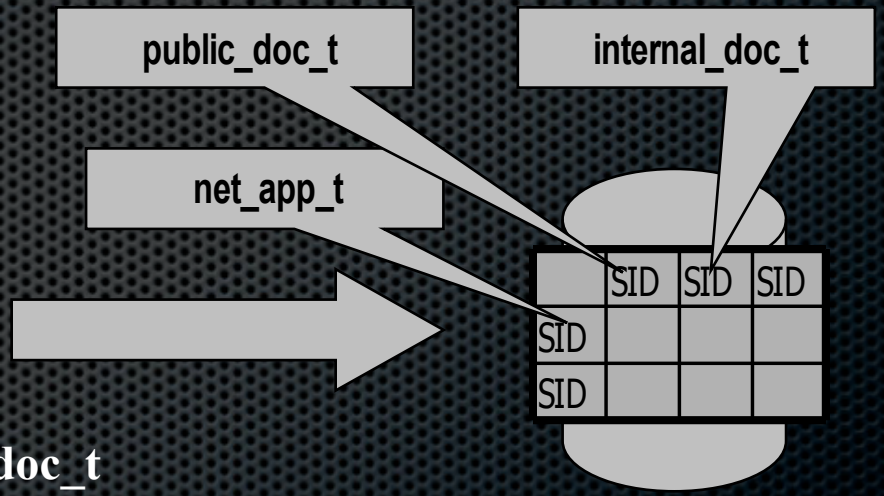
*AppArmor is dead

Как это работает?

- Расширение ядра SELinux контролирует все системные вызовы и проверяет контекст каждого вызова
- Для каждого приложения можно описать индивидуальный набор ограничений - политику
- Политика описывает набор разрешенных операций доступа к ресурсам узла
- Во время выполнения программы производится анализ соответствия реальных запросов приложения политике

Пример

- type net_app_t;
- /usr/sbin/net_app.* -- net_app_t
- type public_doc_t;
- /home/user/public/* -- public_doc_t
- type internal_doc_t;
- /home/user/work/tast.doc -- internal_doc_t
- /home/user/work/tast.doc -- internal_doc_t
- allow net_app_t public_doc_t {read write}

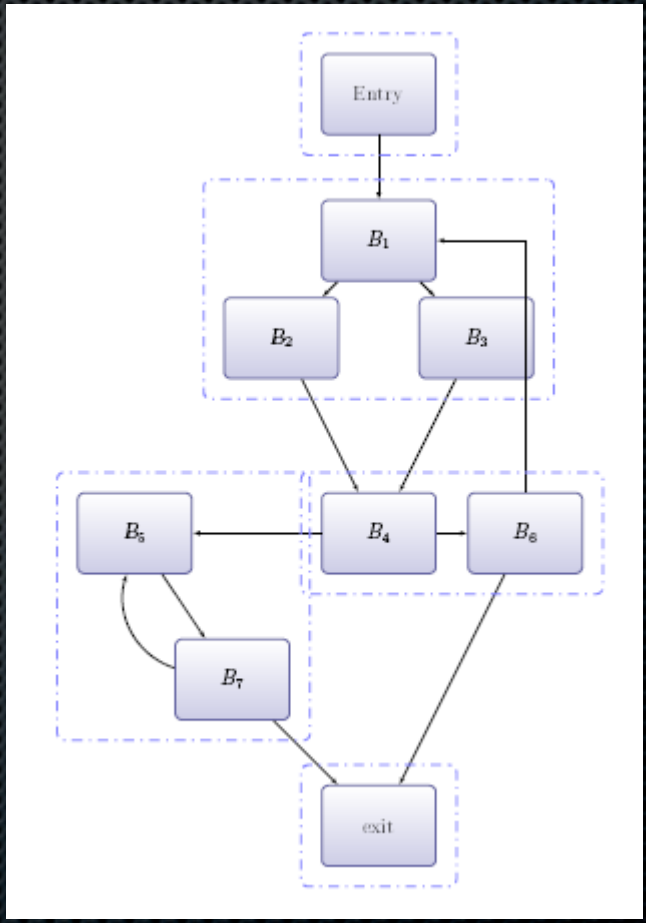


Почему же это не работает?

- Приложению один раз задается набор разрешенных операций доступа ко множеству ресурсов системы, который не меняется во время выполнения приложения
- Такой набор привилегий может являться избыточным большую часть времени выполнения программы – т.е. привилегии не являются минимально необходимыми для корректной работы приложения

Добавим гранулярности

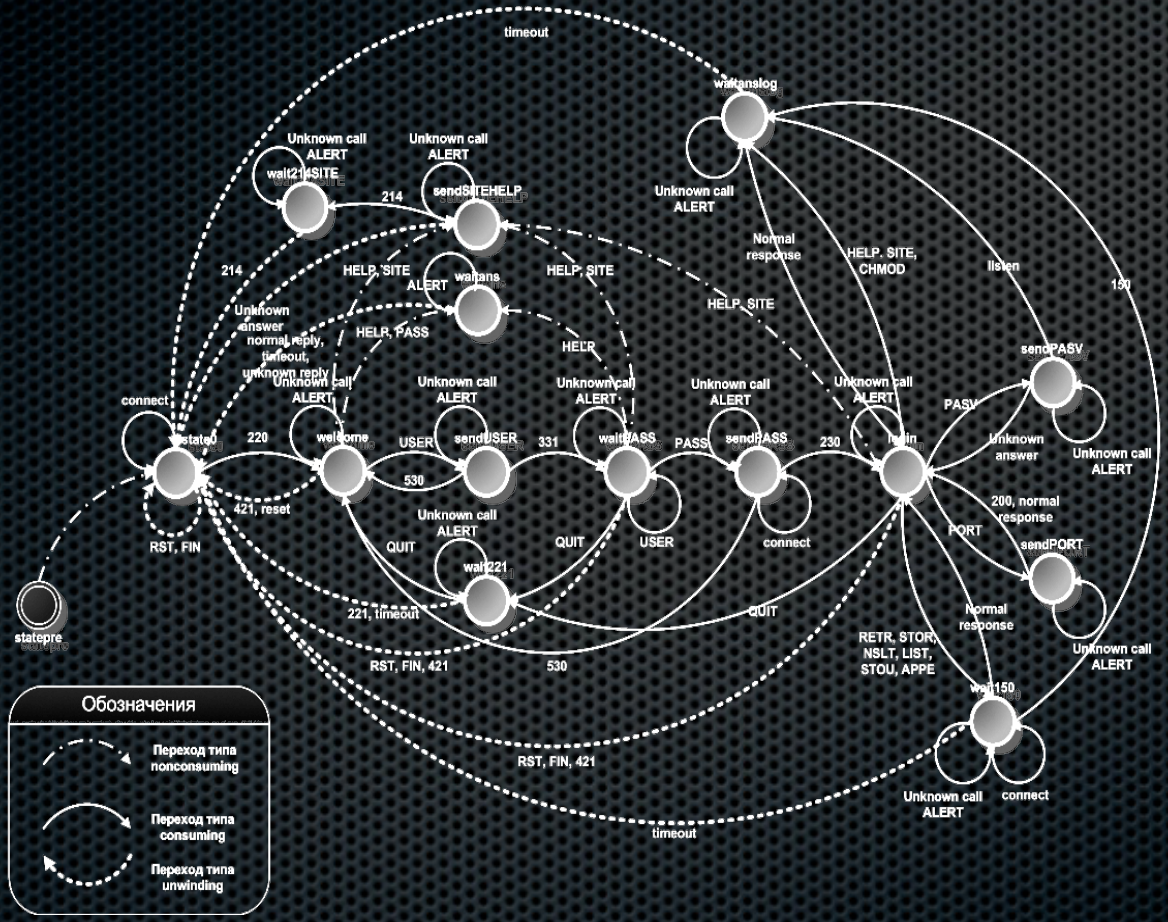
- Дано:
  - Набор трасс приложения
  - Исходный код приложения
  - Исходная политика SELinux для приложения
- Необходимо:
  - По множеству трасс и исходному коду приложения разбить CFG на множество блоков таким образом, чтобы множество порождаемых каждым блоком символов трассы было уже, чем множество разрешённых исходной политикой SELinux

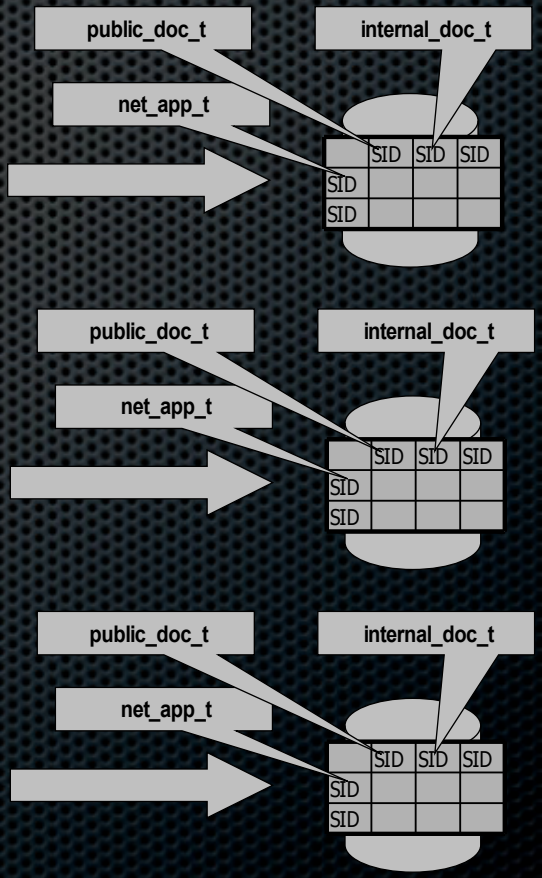


Решение

- Методом динамического анализа для каждого символа трассы находим соответствующую точку в программе (разметка символами трассы)
- Автоматизированно разбиваем текст программы на блоки, минимизируя количество различных символов в каждом блоке
- Устанавливаем метки на вход и выходы блока
- Для каждого блока строим уточнённую политику SELinux
- Генерируем новый набор трасс с двумя типами символов — системные вызовы и метки, строим распознающий ДКА
- **ДКА фукнционирует в userspace и динамически переключает политики для блоков при легитимных переходах между ними**

Желаемый результат





Детали реализации

- Установка и трассировка контрольных точек:
  - Uprobes
  - Utrace
- Монитор
  - Фильтр LSM — в ядре
  - AURA — в userspace
- PoC
  - SSHD с уязвимостью авторизации
  - Telnetd

Ближайшие перспективы

- Сложный PoC для SSHD
- Публикация полного набора инструментов и необходимых патчей к SELinux

Спасибо за внимание

- Контактная информация:
  - Денис Гамаюнов: gamajun@lvk.cs.msu.su
  - Тел. +7 (495) 939 46 71
  - Москва, 119899 Ленинские горы вл. 1/52, факультет ВМК МГУ имени М. В. Ломоносова, к. 764