МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

На правах рукописи

Гамаюнов Денис Юрьевич

ОБНАРУЖЕНИЕ КОМПЬЮТЕРНЫХ АТАК НА ОСНОВЕ АНАЛИЗА ПОВЕДЕНИЯ СЕТЕВЫХ ОБЪЕКТОВ

Специальность 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
д.ф-м.н. Р.Л.Смелянский

МОСКВА
2007

ВВЕДЕНИЕ

1.1. Задача обнаружения компьютерных атак

Данная работа посвящена разработке метода обнаружения атак на распределенные информационные системы (РИС) на основе анализа поведения объектов защищаемой системы. Суть задачи состоит в создании модели компьютерной атаки и метода автоматического обнаружения атаки на основе данной модели, позволяющего обнаруживать компьютерные атаки при наблюдении за поведением объектов РИС и их взаимодействием.

Требуется разработать:

- модель функционирования РИС, пригодную для решения задачи обнаружения атак на РИС;
- метод обнаружения атак на основе предложенной модели.

1.2. Актуальность темы

Компьютерные сети за несколько последних десятилетий из чисто технического решения превратились в глобальное явление, развитие которого оказывает влияние на большинство сфер экономической деятельности. Одним из первых количественную оценку значимости сетей дал Роберт Меткалф, участвовавший в создании Ethernet: по его оценке «значимость» сети во всех смыслах пропорциональна квадрату числа узлов в ней. То есть, зависимость от нормальной работы сетей растёт быстрее, чем сами сети. Обеспечение работоспособности сети и функционирующих в ней информационных систем зависит не только от надёжности аппаратуры, но и, зачастую, от способности сети противостоять целенаправленным воздействиям, которые направлены на нарушение её работы.

Создание информационных систем, гарантированно устойчивых к вредоносным воздействиям и компьютерным атакам, сопряжено с существенными затратами как времени, так и материальных ресурсов. Кроме того, существует известная обратная зависимость между удобством пользования системой и её защищённостью: чем совершеннее системы защиты, тем сложнее пользоваться основным функционалом информационной системы. В 80-е годы XX века, в рамках оборонных проектов США, предпринимались попытки создания распределенных информационных систем специального назначения (MMS – Military Messaging System) [103], для которых формально доказывалась выполнимость основной теоремы безопасности – невыведение системы из безопасного состояния для любой последовательности действий взаимодействующих объектов. В этих системах использовалось специализированное программное обеспечение на всех уровнях, включая системный. Однако, на сегодняшний день подобные системы не получили развития, и для организации информационных систем используются операционные системы общего назначения, такие как ОС семейства Microsoft Windows, GNU/Linux, *BSD и различные клоны SysV UNIX (Solaris, HP-UX, etc).

Из-за высокой сложности и дороговизны разработки защищённых систем by design, тогда же в 80-е годы XX века появилось и начало активно развиваться направление информационной безопасности, связанное с обнаружением (и, возможно, последующим реагированием) нарушений безопасности информационных систем, в качестве эффективного временного решения, позволяющего закрывать «бреши» в безопасности систем до их исправления [5,32]. Данное направление получило название «обнаружение атак» (intrusion detection); и за прошедшие годы в рамках академических разработок были созданы сотни систем обнаружения атак для различных платформ: от систем класса mainframe до современных операционных систем общего назначения, СУБД и распространённых приложений [7,8,64,69].

Создание эффективных систем защиты информационных систем сталкивается также с нехваткой вычислительной мощности. С самого начала развития компьютеров

и компьютерных сетей наблюдаются две тенденции, называемые законом Мура и законом Гилдера. Закон Мура говорит о ежегодном удвоении производительности вычислителей, доступных за одну и ту же стоимость, а закон Гилдера – об утроении пропускной способности каналов связи за тот же период. Таким образом, рост вычислительной мощности узлов сети отстаёт от роста объёмов передаваемой по сети информации, что с каждым годом ужесточает требования к вычислительной сложности алгоритмов систем защиты информации.

Методы обнаружения атак в современных системах обнаружения атак (далее - СОА) недостаточно проработаны в части формальной модели атаки, и, следовательно, для них достаточно сложно строго оценить такие свойства как вычислительная сложность, корректность, завершимость и т.д. [6,7,8,64,69]. Принято разделять методы обнаружения атак на методы обнаружения аномалий и методы обнаружения злоупотреблений [6]. Ко второму типу методов относятся большинство современных коммерческих систем (Cisco IPS, ISS RealSecure, NFR) – они используют сигнатурные (экспертные) методы обнаружения [6,64]. Существует множество академических разработок в области обнаружения аномалий, но в промышленных системах они используются редко и с большой осторожностью, так как такие системы порождают большое количество ложных срабатываний. Для экспертных же систем основной проблемой является низкая, близкая к нулю, эффективность обнаружения неизвестных атак (адаптивность) [7,8,111]. Низкая адаптивность до сих пор остаётся проблемой, хотя такие достоинства как низкая вычислительная сложность и малая стоимость развёртывания определяют доминирование таких систем в данной области.

1.3. Цель работы

Целью данной работы является разработка метода и экспериментальной системы обнаружения атак на РИС на основе наблюдения за поведением объектов РИС, позволяющего объединить достоинства двух подходов – обнаружения аномалий и обнаружения злоупотреблений – при неухудшении показателей эффективности и сложности методов обнаружения злоупотреблений.

В рамках работы ставятся следующие задачи:

- построить модель функционирования РИС, в рамках которой определить такое явление как атака;
- выделить классы реальных атак, представимых в рамках предложенной модели функционирования РИС;
- разработать метод обнаружения атак на РИС на основе информации о поведении объектов РИС;
- разработать архитектуру системы обнаружения атак для РИС на базе ОС GNU/Linux, Windows 2000/XP и сетевого стека TCP/IP.

1.4. Методы решения

В данной работе предлагается модель функционирования РИС в условиях воздействия компьютерных атак в форме системы переходов, имеющая следующие особенности:

- функционирование РИС определяется через понятие состояния объекта РИС и переходы между состояниями, объекты типизированы;
- множество состояний разделяется на безопасные и опасные состояния;
- вводится понятие траектории и поведения объекта;
- понятие атаки вводится как траектория из безопасного состояния некоторого объекта в опасное;
- для каждого класса атак вводится понятия автомата первого рода, который принимает любую траекторию данного класса;

- для каждого класса объектов вводится понятие автомата второго рода, который принимает любую траекторию данного объекта и позволяет разделить множество траекторий на два класса – нормальных и аномальных.

Предложен язык описания поведения объектов РИС, позволяющий описывать состояния объектов РИС и переходы между ними. В основу языка положен формализм конечных автоматов, в которых переходы между состояниями типизированы, а состояние определяется логическим предикатом.

Описание предложенной экспериментальной системы обнаружения атак включает в себя следующие основные разделы:

- алгоритм обнаружения атак, реализующий метод обнаружения атак на основе описаний поведения объектов РИС и наблюдения за поведением объектов РИС в режиме реального времени;
- архитектура системы обнаружения атак;
- исследование эффективности системы и результаты экспериментов.

1.5. Структура работы

Глава 2 настоящей работы посвящена обзору близких по тематике методов обнаружения атак и ряда современных систем обнаружения атак.

Глава 3 настоящей работы посвящена формальной модели функционирования РИС, на основе которой обосновывается применимость метода обнаружения атак, основанного на анализе переходов состояний, а также производится оценка вычислительной сложности и доказательство корректности метода.

Глава 4 посвящена архитектуре экспериментальной системы обнаружения атак на основе предложенного метода.

Глава 5 посвящена исследованию эффективности экспериментальной системы обнаружения атак.

2. ОБЗОР МЕТОДОВ И СИСТЕМ ОБНАРУЖЕНИЯ КОМПЬЮТЕРНЫХ АТАК

В данном разделе приведен обзор основных методов обнаружения компьютерных атак, используемых в современных СОА, а также нескольких некоммерческих СОА.

Целью обзора является исследовать эффективность доступных в настоящее время СОА и определить основные недостатки используемых в них методов обнаружения атак. Основная сложность составления подобного обзора заключается в том, что множество доступных реализаций СОА представлено, в основном, коммерческими системами (такими как Cisco IPS, Juniper NetScreen, ISS RealSecure, NFR и т.д.), для которых отсутствует открытая информация о программной архитектуре и используемым формальным методам обнаружения атак [64]. Доступная информация по подобным системам носит маркетинговый характер, что затрудняет проведение сравнительного анализа по публикациям в литературе. По этой причине множество рассматриваемых в обзоре систем будет ограничено СОА с открытым исходным кодом, доступным публично.

В результате обзора будет показано, что:

1. Большинство современных СОА используют на базовом уровне ту или иную реализацию сигнатурного метода обнаружения (pattern matching, сравнение шаблонов). Реализации отличаются друг от друга уровнем рассмотрения системы, алфавитом сигнатур и используемым «движком» - от простого поиска подстрок до полноценной реализации регулярных выражений над заданным алфавитом.

2. Множество существующих методов обнаружения атак много шире, но их использование в системах имеет принципиальные ограничения, связанные с требованиями верифицируемости, устойчивости и воспроизводимости результата, а также большим числом ошибок второго рода (ложных срабатываний). Использование таких методов ограничено экспериментальными академическими разработками.

3. Доступные реализации СОА неустойчивы к модификациям атак и не могут автоматически адаптироваться к появлению новых атак. При этом использование методов обнаружения аномалий (например, в препроцессорах СОА Snort) ограничено по причинам, перечисленным в п.2.

2.1. Критерии сравнения

В обзоре используются две группы критериев: первая группа характеризует собственно методы обнаружения атак и специфичные для них качественные и количественные показатели эффективности, в то время как вторая группа критериев характеризует реализации этих методов в системах обнаружения атак.

2.1.1. Критерии сравнения методов обнаружения атак

Для сравнительного анализа методов обнаружения атак выбраны следующие критерии:

Уровень наблюдения за системой: Данный критерий определяет уровень абстракции анализируемых событий в защищаемой системе и определяет границы применимости метода для обнаружения атак в сетях. В рамках данного обзора рассматриваются следующие уровни:

- HIDS – наблюдение на уровне операционной системы отдельного узла сети;
- NIDS – наблюдение на уровне сетевого взаимодействия объектов на узлах сети;
- AIDS – наблюдение на уровне отдельных приложений узла сети;
- Hybrid – комбинация наблюдателей разных уровней.

Верифицируемость метода: Данный критерий позволяет оценить, может ли человек (например, квалифицированный оператор СОА или эксперт) воспроизвести

последовательность шагов по принятию решения о наличии атаки, сопоставляя входные и выходные данные СОА. Например, сигнатурные методы будем считать верифицируемыми, а кластерные – нет. Верифицируемость позволяет провести экспертную оценку корректности метода и его реализации в произвольный момент времени, в том числе в процессе эксплуатации системы обнаружения на его основе. Свойство верифицируемости метода важно при эксплуатации системы обнаружения атак в реальной обстановке в качестве средства сбора доказательной базы об атаках. Возможные значения: высокая (+), низкая (-).

Адаптивность метода: Оценка устойчивости метода к малым изменениям реализации атаки, которые не изменяют результат атаки. Адаптивность является единственным существенным преимуществом «альтернативных» методов обнаружения атак перед «сигнатурными». Отсутствие адаптивности не позволяет системе защиты оперативно реагировать на неизвестные атаки и требует организации системы регулярного обновления баз известных атак, по аналогии с антивирусными системами. Возможные значения: высокая (+), низкая (-).

Устойчивость: Данный критерий характеризует независимость выхода метода от защищаемой системы – для одного и того же входа метод должен давать один и тот же выход, независимо от защищаемой системы. Проблема устойчивости особенно остро стоит для статистических методов, анализирующих абсолютные значения параметров производительности и загруженности ресурсов сети и узлов, которые могут существенно отличаться на различных узлах и в различных сетях. Обученный в одной сети распознаватель может быть устойчивым в пределах данной сети и неустойчивым во всех остальных сетях. Такую устойчивость будем называть локальной. Так как процедура обучения обычно является «дорогой» - требует использования большого количества ресурсов и времени – число процедур обучения желательно минимизировать. Методы обнаружения атак, анализирующие семантику ввода, более устойчивы, чем статистические. Возможные значения: глобальная (+), локальная (-).

Вычислительная сложность: Теоретическая оценка сложности метода на основе информации из публикаций. В обзоре рассматривается только сложность метода в режиме обнаружения, без учёта возможных предварительных этапов настройки и обучения. Данный критерий является ключевым для задачи обнаружения атак в сетях и имеет гораздо большее значение, нежели сложность по памяти из-за опережающего роста пропускной способности каналов передачи данных и удешевления машинной памяти.

- Сублинейная – константа, логарифм;
- Линейная;
- Квадратичная и т.д.

В обзоре не рассмотрены такие важные критерии как полнота и точность метода, т.к. эти характеристики редко приводятся в публикациях.

2.1.2. Критерии сравнения систем обнаружения атак

Для сравнительного анализа СОА были выбраны следующие критерии:

Класс обнаруживаемых атак. Данный критерий определяет, какие классы атак способна обнаружить рассматриваемая система. Это один из ключевых критериев. В связи с тем, что на сегодняшний день ни одна система не способна обнаружить атаки всех классов, для более полного покрытия всего спектра атак необходимо комбинировать различные СОА. Здесь мы используем классификацию атак, основанную на разделении ресурсов защищаемой системы по типам.

Класс атаки – это четверка <**L**,**R**,**A**,**D**>, где **L** – расположение атакующего объекта, **R** – атакуемый ресурс, **A** – целевое воздействие на ресурс, **D** – признак распределенного характера атаки.

L: расположение атакующего объекта. Оно может быть либо внутренним по отношению к защищаемой системе (***li***), либо внешним (***le***).

R: атакуемый ресурс. Ресурсы разделяются по расположению и по типу.

По расположению: узловые (***rl***), сетевые (***rn***).

По типу: пользовательские ресурсы (***ru***), системные ресурсы (***rs***), ресурсы СУБД (***rd***), вычислительные ресурсы (***rc***), ресурсы защиты (***rp***).

A: целевое воздействие на ресурс: сбор информации (***as***), получение прав пользователя ресурса (***au***), получение прав администратора ресурса (***ar***), нарушение целостности ресурса (***ai***), нарушение работоспособности ресурса (***ad***).

D: признак распределенного характера атаки: распределенные (***dd***), нераспределенные (***dn***).

Следующий критерий характеризует источники и способы сбора информации о поведении объектов и состоянии ресурсов:

Уровень наблюдения за системой. Определяет, на каком уровне защищаемой системы собирают данные для обнаружения атаки. Различаются узловые и сетевые источники. В пределах узловых источников разделяются уровни ядра и приложения. От уровня наблюдения за системой зависит скорость сбора информации, влияние системы на собираемую информацию, вероятность получения искаженной информации. Следует отметить, что использование метода обнаружения, позволяющего анализировать поведение на всех уровнях абстракции, не означает, что эта возможность реализована в конкретной системе. Зачастую реализация обладает меньшими возможностями, чем теоретические возможности используемого ею метода.

- HIDS – наблюдение на уровне операционной системы отдельного узла сети;
- NIDS – наблюдение на уровне сетевого взаимодействия объектов на узлах сети;
- AIDS – наблюдение на уровне отдельных приложений узла сети;
- Hybrid – комбинация наблюдателей разных уровней.

Следующий критерий определяет эффективность обнаружения атаки на основе анализа полученной информации.

Используемый метод обнаружения. Метод обнаружения также является ключевым критерием сравнения. Существует два класса методов: методы обнаружения аномалий и методы обнаружения злоупотреблений. В приведенном ниже списке перечислены не отдельные методы, но, в основном, семейства методов, объединённых некоторым единым подходом или теоретической моделью.

- Обнаружение злоупотреблений
  - Анализ систем состояний [34,48,52]
  - Графы атак [93]
  - Нейронные сети [30,54,112]
  - Иммунные сети [50,108]
  - SVM [76]
  - Экспертные системы [90,106]
  - Методы, основанные на спецификациях [57]
  - MARS – Multivariate Adaptive Regression Splines [94]
  - Сигнатурные методы [60,61,102]
- Обнаружение аномалий
  - Статистический анализ [4,87]
  - Кластерный анализ (data mining) [55]
  - Нейронные сети [30,54,112]
  - Иммунные сети [50,108]
  - Экспертные системы [90,106]
  - Поведенческая биометрия [1]
  - SVM [76]

- Анализ систем состояний [34,52]

Адаптивность к неизвестным атакам. Определяет способность используемого метода обнаруживать ранее неизвестные атаки.

Следующие три критерия определяют такие архитектурные особенности СОА как управление и распределенность.

Масштабируемость. Определяет возможность добавления новых анализируемых ресурсов сети, новых узлов и каналов передачи данных, в том числе возможность управления единой распределенной системой обнаружения атак. Управление может быть централизованное и/или распределенное. Дополнительно может присутствовать возможность удаленного управления СОА. Сюда включаются задачи установки, настройки и администрирования системы. При полностью распределенном управлении необходимо управлять всеми компонентами СОА в отдельности. При полностью централизованном управлении все компоненты СОА могут управляться с одного узла. Оптимальной представляется организация управления по централизованной схеме, в которой может быть несколько центров, и они могут динамически меняться.

Открытость. Определяет насколько система является открытой для интеграции в нее других методов обнаружения атак, компонентов сторонних разработчиков и сопряжения ее с другими системами защиты информации. Это могут быть программные интерфейсы для встраивания дополнительных модулей и/или реализация стандартов взаимодействия сетевых компонентов.

Формирование ответной реакции на атаку. Определяет наличие в системе встроенных механизмов ответной реакции на атаку, кроме самого факта ее регистрации. Примерами реакции могут быть разрыв соединения с атакующим объектом, блокировка его на межсетевом экране, отслеживание пути проникновения атакующего объекта в защищаемую систему и т.д. Подробно это вопрос рассмотрен в разделе 2.3, который посвящён описанию рассмотренных в обзоре систем.

Защищенность. Определяет степень защищенности СОА от атак на ее компоненты, включая защиту передаваемой информации, устойчивость к частичному выходу компонентов из строя или их компрометации. Затрагиваются такие вопросы, как наличие уязвимостей в компонентах СОА, защищенность каналов передачи данных между ними, а также авторизация компонентов внутри СОА.

Таким образом, некая «идеальная» система обнаружения атак обладает следующими свойствами:

- покрывает все классы атак (система полна);
- позволяет анализировать поведение защищаемой РИС на всех уровнях: сетевом, узловом и уровне отдельных приложений;
- адаптивна к неизвестным атакам (использует адаптивный метод обнаружения атак);
- масштабируется для РИС различных классов: от небольших локальных сетей класса «домашний офис» до крупных многосегментных и коммутированных корпоративных сетей, обеспечивая возможность централизованного управления всеми компонентами СОА;
- является открытой;
- имеет встроенные механизмы реагирования на атаки;
- является защищённой от атак на компоненты СОА, в том числе от перехвата управления или атаки «отказ в обслуживании».

2.2. Методы обнаружения атак

Все методы обнаружения атак можно разделить на два класса: методы обнаружения аномалий и методы обнаружения злоупотреблений. Методы первого класса базируются на наличии готового описания нормального поведения наблюдаемых объектов РИС, и любое отклонение от нормального поведения считается

аномальным (нарушением). Методы обнаружения злоупотреблений основаны на описании известных нарушений или атак: если наблюдаемое поведение некоторого объекта РИС совпадает с описанием известной атаки, поведение объекта считается атакой.

2.2.1. Методы обнаружения злоупотреблений

Анализ систем состояний: В данной группе методов [34,48,52] функционирование защищаемой системы представляется через множество состояний и множество переходов между ними, т.е. в виде ориентированного графа (как правило, бесконечного). Суть метода обнаружения атак заключается в том, что часть путей в таком графе помечаются как недопустимые; конечное состояние каждого такого пути считается опасным для защищаемой системы. Процесс обнаружения атаки представляет собой построение части графа состояний системы и наблюдаемых переходов между ними, и поиск в полученном графе известных недопустимых путей. Обнаружение последовательности переходов, приводящей в опасное состояние, означает успешное обнаружение атаки. В соответствии с введёнными критериями, данный метод является гибридным с точки зрения уровня наблюдения за системой, верифицируемым, устойчивым, имеет низкую вычислительную сложность (линейную относительно длины трассы наблюдаемых переходов и числа состояний), но не является адаптивным.

Графы сценариев атак: В работе [93] предложен подход к обнаружению атак на основе использования методов формальной спецификации на моделях. На вход системе верификации подаётся конечная модель защищаемой системы и некоторое формальное свойство корректности, которое выполняется только для разрешённого поведения системы. Данное свойство корректности делит всё множество поведений на два класса – допустимого поведения, для которого свойство выполняется, и недопустимого, для которого оно не выполняется. Отличие данного метода от обычных систем верификации заключается в том, что их задача, обычно, найти один контрпример из множества недопустимого поведения, а в предложенном методе строится полный набор таких примеров для конкретной защищаемой системы, что даёт на выходе описание возможных путей атаки. Из-за высокой вычислительной сложности (NP) данный метод может быть использован для поиска уязвимостей проектирования систем и других сложных для обнаружения уязвимостей, но для задачи обнаружения атак в реальном времени он неприменим. По остальным критериям метод является гибридным, верифицируемым, устойчивым и адаптивным.

Нейронные сети: Так как задачу обнаружения атак можно рассматривать как задачу распознавания образов (или задачу классификации), то для её решения также применяются нейронные сети [30,54,112]. Для этого функционирование защищаемой системы и взаимодействующих с ней внешних объектов представляется в виде траекторий в некотором числовом пространстве признаков. В качестве метода обнаружения злоупотреблений, нейронные сети обучаются на примерах атак каждого класса и, в дальнейшем, используются для распознавания принадлежности наблюдаемого поведения одному из классов атак. Основная сложность в использовании нейросетей заключается в корректном построении такого пространства признаков, которое позволило бы разделить классы атак между собой и отделить их от нормального поведения. Кроме того, для классических нейронных сетей характерно долгое обучение, при этом время обучения зависит от размера обучающей выборки. В соответствии с введёнными критериями, нейронные сети используются на сетевом и узловом уровнях, являются адаптивными, имеют сравнительно низкую вычислительную сложность. При этом они не являются верифицируемыми и устойчивы, как правило, только в пределах той сети, в которой они обучались, что существенно ограничивает применимость метода (только локальная устойчивость).

Иммунные сети: Также как и нейронные сети, иммунные сети являются механизмом классификации и строятся по аналогии с иммунной системой живого организма. Основное достоинство иммунных сетей заключатся в возможности получения «антител» к неизвестным атакам [50,108]. В работе [113] предложена модель формального пептида, для которой заявлена возможность использования в системах обнаружения атак. Однако, позже было показано, что использование данного метода требует решения системы дифференциальных уравнений в режиме обнаружения, что даёт вычислительную сложность порядка $O(n^3)$ при использовании метода Рунге-Кутта. В соответствии с введёнными критериями, данная группа методов применима для сетевого и узлового уровней, не верифицируема, адаптивна, устойчива только локально, имеет высокую вычислительную сложность.

Support vector machines (SVM): SVM – это метод представления и распознания шаблонов, который позволяет формировать шаблоны в результате обучения [76]. Данный метод требует небольшого количества данных для обучения и позволяет обрабатывать векторы признаков большой размерности, что полезно для повышения точности систем обнаружения атак и снижения временных затрат на обучение и переобучение. Метод применим как для обнаружения злоупотреблений, так и для обнаружения аномалий. SVM имеет такие же достоинства и недостатки для решения нашей задачи, как и нейронные сети, т.е. является адаптивным, но неверифицируемым.

Экспертные системы: Использование экспертных систем для обнаружения атак основано на описании функционирования системы в виде множества фактов и правил вывода, в том числе для атак [90,106]. На вход экспертная система получает данные о наблюдаемых событиях в системе в виде фактов. На основании фактов и правил вывода система делает вывод о наличии или отсутствии атаки. Данная группа методов удовлетворяет практически всем критериям (верифицируема, адаптивна, устойчива), но в общем случае имеет очень большую вычислительную сложность, так как для нее может наблюдаться явление «комбинаторного взрыва» и полного перебора большого числа альтернатив.

Методы, основанные на спецификациях: В основе данного метода лежит описание ограничений на запрещенное поведение объектов в защищаемой системе в виде спецификаций атак [57]. В спецификацию может входить: ограничения на загрузку ресурсов, на список запрещенных операций и их последовательностей, на время суток, в течение которого применимы те или иные ограничения. Соответствие поведения спецификации считается атакой. Спецификации используются для сетевого уровня, является верифицируемым, локально устойчивым и имеет низкую вычислительную сложность. Данный подход близок к классу методов обнаружения аномалий. Основные недостатки – низкая адаптивность и сложность разработки спецификаций.

Multivariate Adaptive Regression Splines (MARS): Один из методов аппроксимации функций, основанный на сплайнах [94]. Аналогично нейронным сетям и кластерному анализу MARS оперирует в многомерном пространстве признаков. Поведение сетевых объектов отображается в последовательности векторов данного пространства. Задача процедуры MARS заключается в построении оптимальной аппроксимации поведения по заданной истории в виде обучающего множества векторов, при этом в качестве аппроксимирующей функции используются сплайны с переменным числом вершин. В ходе «обучения», с помощью переборного процесса, выбирается оптимальное число вершин для заданной выборки. Построенный сплайн является «шаблоном» атаки. В режиме распознавания наблюдаемое поведение отображается в параметрическое пространство и сравнивается с аппроксимирующей функцией. Достоинства и недостатки данного метода аналогичны SVM и нейронным сетям.

Сигнатурные методы: Наиболее часто используемая группа методов, суть которых заключается в составлении некоторого алфавита из наблюдаемых в системе событий и описании множества сигнатур атак в виде регулярных выражений (в общем случае) в

построенном алфавите [60,61,102]. Как правило, сигнатурные методы работают на самом низком уровне абстракции и анализируют непосредственно передаваемые по сети данные, параметры системных вызовов и записи файлов журналов. В наиболее развитом виде представляет собой реализацию регулярных выражений над различными трассами (сетевой трафик, системные вызовы, записи журналов приложений и т.п.). Сигнатурные методы примечательны тем, что для них хорошо применимы аппаратные ускорители, но при этом метод не является адаптивным. По остальным критериям данная группа методов является гибридной, глобально устойчивой, верифицируемой.

2.2.2. Методы обнаружения аномалий

Статистический анализ: Данная группа методов основана на построении статистического профиля поведения системы в течение некоторого периода «обучения», при котором поведение системы считается нормальным [4,87]. Для каждого параметра функционирования системы строится интервал допустимых значений, с использованием некоторого известного закона распределения. Далее, в режиме обнаружения, система оценивает отклонения наблюдаемых значений от значений, полученных во время обучения. Если отклонения превышают некоторые заданные значения, то фиксируется факт аномалии (атаки). Для статистического анализа характерен высокий уровень ложных срабатываний при использовании в локальных сетях, где поведение объектов не имеет гладкого, усреднённого характера. Кроме того, данный метод устойчив только в пределах конкретной системы, то есть построенные статистические профили нельзя использовать на других аналогичных системах.

Кластерный анализ: Суть данной группы методов состоит в разбиении множества наблюдаемых векторов-свойств системы на кластеры, среди которых выделяют кластеры нормального поведения [55]. В каждом конкретном методе кластерного анализа используется своя метрика, которая позволяет оценивать принадлежность наблюдаемого вектора свойств системы одному из кластеров или выход за границы известных кластеров. Кластерный анализ является адаптивным, но не верифицируемым и устойчивым в пределах конкретной системы, в которой собирались данные для построения кластеров.

Нейронные сети: Нейронные сети для обнаружения аномалий обучаются в течение некоторого периода времени, когда всё наблюдаемое поведение считается нормальным [30,54,112]. После обучения нейронная сеть запускается в режиме распознавания. В ситуации, когда во входном потоке не удается распознать нормальное поведение, фиксируется факт атаки. В случае использования репрезентативной обучающей выборки нейронные сети дают хорошую устойчивость в пределах заданной системы; но составление подобной выборки является серьёзной и сложной задачей. Классические нейронные сети имеют высокую вычислительную сложность обучения, что затрудняет их применение на больших потоках данных.

Иммунные сети: Обнаружение аномалий является одним из возможных приложений иммунных методов. Так как количество примеров нормального поведения обычно на порядки превышает число примеров атак, использование иммунных сетей для обнаружения анормалий имеет большую вычислительную сложность. [50,108].

Экспертные системы: Информация о нормальном поведении представляется в подобных системах в виде правил, а наблюдаемое поведение в виде фактов. На основании фактов и правил принимается решение о соответствии наблюдаемого поведения «нормальному», либо о наличии аномалии. Главный недостаток подобных систем – высокая вычислительная сложность (в общем случае). В том числе при обнаружении аномалий [90,106].

Поведенческая биометрия: Включает в себя методы, не требующие специального оборудования (сканеров сетчатки, отпечатков пальцев), т.е. методы обнаружения атак,

основанные на наблюдения клавиатурного почерка и использования мыши. В основе методов лежит гипотеза о различии «почерка» работы с интерфейсами ввода-вывода для различных пользователей. На базе построенного профиля нормального поведения для данного пользователя обнаруживаются отклонения от этого профиля, вызванные попытками других лиц работать с клавиатурой или другими физическими устройствами ввода. Поведенческая биометрия имеет строго локальную устойчивость (в пределах одной сети) и слабо верифицируема [1].

Support vector machines (SVM): SVM применим как для обнаружения злоупотреблений, так и для обнаружения аномалий, при этом метод имеет достоинства и недостатки, аналогичные нейронным сетям [76].

2.2.3. Результаты сравнительного анализа

Критерий / Метод	Уровень наблюдения	Аномалии/ Злоупот-ребления	Верифици-руемость	Адаптивность	Устойчивость	Вычислительная сложность
Системы переходов	Hybrid	-/+	+	-	+	O(n)
Графы атак	Hybrid	-/+	+	+	+	NP
Нейронные сети	NIDS, HIDS	+/+	-	+	-	O(n) и выше
Иммунные сети	NIDS, HIDS	+/+	-	+	-	O(n) и выше
SVM	NIDS, HIDS	+/+	-	+	-	ln(n)
Экспертные системы	NIDS, HIDS	+/+	+	+	+	В общем случае NP
Спецификации	NIDS	-/+	+	-	-	ln(n)
MARS	NIDS, HIDS	-/+	-	+	-	O(n) и выше
Сигнатурные методы	Hybrid	-/+	+	-	+	ln(n)
Статистические методы	NIDS, HIDS	+/-	-	+	-	O(n) и выше
Кластерный анализ	Hybrid	+/+	-	+	-	O(n) и выше
Поведенческая биометрия	HIDS	+/-	-	+	-	O(n) и выше

Таблица 1. Результаты сравнения методов обнаружения атак.

Таким образом, анализ публикаций показывает, что для большинства методов обнаружения аномалий характерна слабая верифицируемость и слабая глобальная устойчивость (либо её отсутствие). Основное достоинство методов обнаружения аномалий заключается в их адаптивности и способности обнаруживать ранее неизвестные атаки. Среди глобально устойчивых и верифицируемых методов, имеющих при этом низкую вычислительную сложность, можно отметить метод анализа системы переходов и простой сигнатурный метод. Ни один из рассмотренных методов не обладает одновременно адаптивностью, устойчивостью и верифицируемостью, имея при этом приемлемую вычислительную сложность.

2.3. Современные открытые системы обнаружения атак

В данном разделе рассмотрены доступные на сегодняшний день системы обнаружения атак с открытым исходным кодом.

2.3.1. Исследованные системы обнаружения атак

Всего рассмотрено 5 систем обнаружения атак. В табл. 2. приведена краткая информация по каждой из них.

Название системы	Производитель	Ссылки
Bro	University of California, Lawrence Berkeley National Laboratory	http://bro-ids.org/

OSSEC	Daniel B. Sid , OSSEC.net	http://www.ossec.net/
STAT	University of California at Santa Barbara	http://www.cs.ucsb.edu/~seclab/projects/stat/index.html
Prelude	Yoann Vandoorselaere yoann@mandrakesoft.com Laurent Oudot oudot.laurent@wanadoo.fr	http://www.prelude-ids.org/
Snort	Martin Roesch	http://www.snort.org/

Таблица 2. Открытые системы обнаружения компьютерных атак.

Часть рассмотренных систем (Bro, NetSTAT) разработаны в университетах и базируются на исследованиях в области обнаружения атак, проведенных в этих университетах.

2.3.2. Результаты сравнительного анализа

В данном разделе приводятся результаты сравнения рассмотренных СОА. Системы сравниваются отдельно по каждому из вышеуказанных критериев (см.раздел 2.1.2). Сводная таблица сравнения СОА по всем критериям дана в конце раздела.

Все рассмотренные системы используют в качестве основного метода обнаружения атак сигнатурный метод (сравнение строк, шаблонов).

Система Bro использует регулярные выражения над трассами, которые формируются сетевыми протоколами. Набор регулярных выражений создаётся экспертами. Кроме того, в состав системы входит транслятор сигнатур из формата системы Snort в сценарии Bro (хотя в настоящее время этот транслятор поддерживает не все конструкции языка Snort).

Система OSSEC является монолитной – в сенсоры и анализаторы «зашиты» знания разработчиков системы обнаружения атак о том, какие последовательности сообщений в журналах могут быть признаками атаки. Такая архитектура системы является трудно расширяемой с точки зрения базы знаний об атаках.

Система NetSTAT использует язык описания сценариев атак STATL, особенностью которого является возможность описания сценария атаки в виде последовательности действий над атакуемым ресурсом. Таким образом, эта система использует метод обнаружения, близкий к методу анализа переходов состояний.

Система Prelude использует различные анализирующие компоненты для сетевых данных и журналов регистрации. Для анализа сетевых данных можно использовать систему Snort. Также используется набор специализированных модулей для обнаружения специфических атак, таких как сканирование портов, некорректные ARP-пакеты и т.п. Специальные модули производят дефрагментацию IP, сборку TCP-потока, декодирование HTTP-запросов.

Система Snort использует базу сигнатур известных атак. В ней также используется набор специализированных модулей для обнаружения специфических атак, таких как сканирование портов или отправка большого числа фрагментированных пакетов. Специальные модули производят дефрагментацию IP, декодирование HTTP-запросов. Сторонние разработчики часто реализуют другие методы обнаружения атак в виде модулей (препроцессоров) Snort. Но в основную версию системы они не входят.

Класс обнаруживаемых атак

Все исследованные системы могут обнаруживать атаки нескольких классов. Поэтому для улучшения читаемости и сокращения объема текста вводятся понятия

объединения, пересечения и вложения классов атак. Для обозначения объединения и пересечения классов атак будем использовать символы “$\cup$” и “$\cap$” соответственно.

Системы, рассмотренные в данной работе, предназначены для обнаружения атак разных классов. Часть систем ориентирована на обнаружения узловых атак, и использует для анализа такие источники как журналы регистрации приложений, ОС, журналы систем аудита (OSSEC). Другие системы обнаруживают только внешние (сетевые) атаки и используют для анализа информацию, получаемую из каналов передачи данных в сети (Bro, Snort). Остальные системы являются гибридными и обнаруживают как локальные, так и внешние атаки (STAT, Prelude).

Система Bro является сетевой системой обнаружения атак. Она представляет собой набор модулей декомпозиции данных различных сетевых протоколов (от сетевого до прикладного уровня) и набор сигнатур над событиями соответствующих протоколов. Сигнатуры Bro фактически представляют собой регулярные выражения в алфавитах протоколов.

Данная система обнаруживет атаки следующих классов (L,R,A,D):

- L={***li*** $\cup$ ***le***} (внутренние и внешние атаки);
- R={***rn***} $\cap$ {***ru*** $\cup$ ***rs***} (атаки на сетевые пользовательские ресурсы и системные ресурсы);
- A={***as*** $\cup$ ***au*** $\cup$ ***ar*** $\cup$ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={***dn*** $\cup$ ***dd***} (нераспределенные и распределенные).

Система OSSEC, единственная из рассмотренных в данной работе, является изначально ориентированной на обнаружение атак уровня системы (узловых). Она наиболее «молодая» из рассмотренных систем; её последняя версия предназначена, в частности, для анализа журналов регистрации UNIX, типовых приложений (ftpd, apache, mail, etc), а также журналов межсетевых экранов и сетевых СОА. OSSEC включает в себя набор анализаторов для различных источников данных, контроль целостности файловой системы, сигнатуры известных троянских закладок (rootkits) и пр.

Обнаруживаются атаки следующих классов:

- L={***li*** } (атакующие объекты находятся внутри системы);
- R={***rl***} $\cap$ {***ru*** $\cup$ ***rs***} (узловые пользовательские и системные ресурсы);
- A={***au*** $\cup$ ***ar*** $\cup$ ***ai***} (попытки получения прав пользователя, попытки получения прав администратора, нарушение целостности ресурса);
- D={***dn*** $\cup$ ***dd***} (нераспределенные и распределенные).

Система STAT является экспериментальной университетской разработкой, и наиболее «старой» из рассматриваемых систем – первые публикации по STAT датируются 1992 годом. Система включает в себя набор компонентов обнаружения атак различных уровней – сетевой (NetSTAT), узловой (USTAT, WinSTAT), приложений (WebSTAT), т.е. является классической гибридной системой.

СОА обнаруживает атаки следующих классов: (L,R,A,D). Где:

- L={***li*** $\cup$ ***le***} (внутренние и внешние атаки);
- R={***rl*** $\cup$ ***rn*** } $\cap$ {***ru*** $\cup$ ***rs***} (атаки на узловые или сетевые пользовательские ресурсы и системные ресурсы);
- A={***as*** $\cup$ ***au*** $\cup$ ***ar*** $\cup$ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={***dn***} (нераспределенные).

Система Prelude, как и NetSTAT, является гибридной, т.е. способна обнаружить атаки как на уровне системы, так и на уровне сети. Данная система изначально

разрабатывалась в качестве самостоятельной СОА, но в настоящее время является высокоуровневой надстройкой над открытыми СОА и системами контроля целостности (AIDE, Osiris и т.п.). Узловая часть Prelude имеет достаточно широкий набор описаний атак и, в качестве источника информации, использует различные журналы регистрации:

- журналы регистрации межсетевого экрана IPFW;
- журналы регистрации NetFilter ОС Linux;
- журналы регистрации маршрутизаторов Cisco and Zyxel;
- журналы регистрации GRSecurity;
- журналы регистрации типовых сервисов ОС UNIX и другие.

СОА обнаруживает атаки следующих классов: (L,R,A,D). Где:

- L={***li*** ∪ ***le***} (внутренние и внешние атаки);
- R={***rl*** ∪ ***rn***} ∩ {***ru*** ∪ ***rs*** ∪ ***rp***} (атаки на локальные или сетевые пользовательские ресурсы, системные ресурсы и ресурсы защиты);
- A={***as*** ∪ ***au*** ∪ ***ar*** ∪ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={***dn***} (нераспределенные).

Система Snort это наиболее популярная на сегодняшний день некоммерческая СОА. Она активно и динамично развивается, обновления базы известных атак происходят с частотой, сравнимой с коммерческими аналогами (обычно обновления Snort опережают коммерческие). Snort является чисто сетевой СОА и, кроме основной базы описаний атак, имеет набор подключаемых модулей для обнаружения специфических атак или реализующих альтернативные методы обнаружения.

Система способна обнаружить атаки следующих классов (L,R,A,D):

- L=”внутренние” ∪ ”внешние”;
- R={***rl*** ∪ ***rn***} ∩ {***ru*** ∪ ***rs*** ∪ ***rp***} (атаки на локальные или сетевые пользовательские ресурсы, системные ресурсы и ресурсы защиты);
- A={***as*** ∪ ***au*** ∪ ***ar*** ∪ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={***dn*** ∪ ***dd***} (нераспределенные и распределенные).

Таким образом, ни одна из рассмотренных систем не покрывает всё множество классов атак. Следует также отметить, что эти системы используют неадаптивные методы обнаружения атак.

Уровень наблюдения за системой

Все рассмотренные выше системы работают с данными приложений и операционной системы на узловом уровне, а так же с сетевыми данными. То есть анализируемая информация получается из вторичных источников, таких как журналы регистрации приложений, ОС, либо из сетевого канала передачи данных.

Система OSSEC работает исключительно с журналами регистрации приложений и операционной системы. Системы Bro, Snort анализируют только сетевые данные. Системы NetSTAT и Prelude анализируют как данные из локальных системных источников, так и сетевые данные.

Из рассмотренных систем ни одна не покрывает все уровни наблюдения, и анализируемая каждой системой информация неполна с точки зрения возможности обнаружения атак всех классов. Для обнаружения атак всех классов необходимо анализировать информацию на всех трёх уровнях одновременно.

Адаптивность к неизвестным атакам

На данный момент эта возможность в рассмотренных СОА отсутствует. Возможно использование экспериментального модуля статистического анализа системы Snort, но его эффективность не изучена. За счет контроля целостности ресурсов узла в системе OSSEC присутствует условная адаптивность. Тем не менее, следует признать, что контроль целостности решает не задачу обнаружения атак, а лишь задачу обнаружения их последствий. Таким образом, адаптивность к неизвестным атакам в рассмотренных СОА, в целом, отсутствует.

Масштабируемость

Система Bro является нераспределённой и управляется централизованно на том узле, где она установлена, с помощью файлов конфигурации. При увеличении числа защищаемых узлов и каналов связи необходимо устанавливать дополнительные независимые экземпляры системы Bro, что означает фактическую немасштабируемость.

Система OSSEC является распределённой и управляется либо распределенно на узлах, где установлены агенты (при помощи файлов конфигурации), либо централизованно с помощью специализированной утилиты администрирования (manage_agents) с центрального сервера OSSEC. Система является хорошо масштабируемой.

Система NetSTAT также является распределённой и управляется распределенно через файлы конфигурации на всех узлах, где расположены компоненты системы. Индивидуальное управление компонентов системы делает процесс управления и настройки сложным и длительным, причём с ростом числа компонентов сложность настройки и внесения изменений в конфигурацию усложняется.

Система Prelude является распределённой и управляется централизованно при помощи управляющей консоли. Компоненты системы сами предоставляют управляющей консоли те параметры их функционирования, которые могут изменяться. Управление производится по защищенному каналу (SSL). Также управление может осуществляться через локальные конфигурационные файлы на тех узлах, где установлены компоненты СОА. Данная система является хорошо масштабируемой.

Система Snort управляется централизованно через файлы конфигурации, консольные команды и сигналы UNIX. Сама по себе система не является масштабируемой, но в случае использования Snort в качестве сенсора системы Prelude этот недостаток устаняется.

Открытость.

Три из рассматриваемых систем, за исключением Bro и OSSEC, имеют открытый интерфейс для добавления новых анализирующих модулей, а также используют стандартный для систем обнаружения атак формат обмена сообщениями (IDMEF).

Система Bro позволяет пользователю и сторонним разработчикам расширять набор сигнатур.

NetSTAT имеет открытый интерфейс для добавления новых агентов и фильтров.

Prelude имеет открытый интерфейс для добавления новых модулей анализа и реагирования, а так же ведения журналов регистрации. Обмен сообщениями между компонентами системы происходит по стандарту IDMEF (Intrusion Detection Message Exchange Format), оптимизированному для высокоскоростной обработки.

Snort имеет открытый интерфейс для добавления новых модулей анализа; имеется модуль, реализующий протокол SNMPv2.

По совокупности используемых стандартных интерфейсов, системы Prelude и Snort лучше остальных позволяют наращивать функциональность по обнаружению атак.

Формирование ответной реакции на атаку

Встроенную возможность реагирования на атаку имеют все рассматриваемые системы. В системе NetSTAT это реализовано лишь в тестовом варианте. Система Prelude имеет набор агентов ответной реакции, которые могут блокировать атакующего при помощи межсетевого экрана. Ведутся работы по агентам, способным либо полностью изолировать атакующего, либо уменьшить пропускную способность его канала. Система Snort имеет встроенную ограниченную возможность реагирования на атаку путем отправки TCP-пакетов, разрывающих соединение (с установленным флагом RST), а также ICMP-пакетов, сообщающих атакующему узлу о недоступности узла, сети или сервиса. Аналогичная функциональность по реагированию доступна в системе Bro. Система OSSEC позволяет использовать произвольные команды для реагирования – для этого необходимо статически задать соответствие между событием, командой и параметрами её вызова.

Защищенность

Все системы, которые пересылают какие-либо данные, используют для этого защищенные каналы. STAT и Prelude используют библиотеку OpenSSL для шифрования канала между компонентами. Snort реализует протокол SNMPv2, в котором присутствуют функции шифрования паролей при передаче данных.

СОА Prelude имеет дополнительные механизмы, обеспечивающие безопасность ее компонентов. В системе используется специализированная библиотека, которая делает безопасными такие библиотечные функции алгоритмического языка С как printf, strcpy, которые не проверяют размер передаваемых им данных. Библиотека предотвращает классические ошибки выхода за границы массивов и переполнения буферов.

Дополнительные модули анализа сетевых данных делают систему устойчивой к некорректным сетевым пакетам на разных уровнях стека и выходу ее компонентов из строя. Такие атаки, как отправка пакетов с неправильными контрольными суммами, обнуленными флагами TCP, ресинхронизация сессий, случайная отправка и «обрезание» сегментов системой игнорируются.

Из рассмотренных систем вопрос безопасности наиболее проработан в системе Prelude.

Итоговая таблица по критериям сравнения

Ниже в табл. 3. приведены сводные результаты сравнения рассмотренных систем по выбранным критериям.

	Bro	***OSSEC***	***STAT***	***Prelude***	***Snort***
Классы атак	*({**li** ∪ **le**}, {rl} ∩ {ru ∪rs ∪ rc}, {**as** ∪ **au** ∪ **ar** ∪ **ad**}, {**dn**})*	*({**li** }, {**rl**} ∩ {**ru** ∪**rs**}, {**au** ∪**ar** ∪**ai**}, {**dn**})*	*({**li** ∪**le**}, {**rl** ∪ rn) ∩ (**ru** ∪ **rs**}, {**as** ∪ **au** ∪ **ar** ∪ **ad**}, {**dn**})*	*({**li** ∪**le**}, {**rl** ∪ rn) ∩ {**ru** ∪ **rs** ∪**rp**}, {**as** ∪**au** ∪**ar** ∪**ad**}, {**dn**∪**dd**})*	*({**li** ∪**le**}, {**rl** ∪rn) ∩{**ru** ∪**rs**∪**rp**}, {**as** ∪**au** ∪**ar** ∪**ad**}, {**dn**})*
Уровень наблюдения за системой	*Системный*	*Системный*	*Системный, сетевой*	*Системный, сетевой*	*Сетевой*
Метод обнаружения	*Сигнатурный*	*Сигнатурный*	*Сигнатурный, анализ переходов состояний*	*Сигнатурный*	*Сигнатурный*
Адаптивность	-	+/-	-	-	-
Масштабируемость	-	+	+	+	-
Открытость	*Открытый API*	*Открытый API*	*Открытый API*	*Открытый API, IDMEF*	*Открытый API, SNMPv2*
Реакция	-	-	+	+	+

Защита	-	-	*SSL*	*SSL, Libsafe*	-

Таблица 3. Результаты сравнения открытых СОА.

На основании проведённого сравнительного анализа можно сделать вывод, что ни одна из рассмотренных выше открытых СОА, не соответствует в полной мере критериям «идеальной» СОА, изложенным в разделе 2.1. Основным недостатком является отсутствие адаптивности к неизвестным атакам и невозможность анализировать поведение объектов РИС на всех уровнях одновременно.

2.4. Описание исследованных систем

В данном разделе приводится подробное описание рассмотренных в данной работе СОА. Для каждой системы описывается ее архитектура, используемая платформа и некоторые индивидуальные особенности.

2.4.1. Bro

Система Bro [80,81] является разработкой Национальной лаборатории Лоуренса Беркли Калифорнийского университета, Беркли, США. Система является открытой и распространяется по собственной открытой лицензии в исходных текстах и бинарных пакетах для ряда UNIX-платформ. Система предназначена для пассивного мониторинга сетевого трафика и поиска подозрительной активности. Обнаружение атак выполняется на нескольких уровнях: входящий сетевой трафик разбирается для выявления семантики уровня приложений, после чего полученная трасса событий прикладного уровня анализируется набором событийно-ориентированных анализаторов и сравнивается с шаблонами атак.

Bro использует специализированный язык управления политиками, позволяющий подстраивать поведение системы под защищаемую систему, в соответствии с изменяющимися внешними условиями. При обнаружении атаки система может выполнить различные действия – записать сообщение в журнал, оповестить оператора, выполнить команды операционной системы.

Назначением системы является высокоскоростное обнаружение атак на сетевых каналах с высокой пропускной способностью (1Гбит).

Архитектурно Bro можно разделить на три основных компонента: библиотека libpcap для захвата пакетов, модуль генерации событий и интерпретатор сценариев. Интерпретатор сценариев выполняет анализаторы событий, написанные на языке Bro. Данный набор сценариев является политикой безопасности сети, который определяет реакцию системы на различные события. Сценарий может генерировать сообщения, а также выполнять произвольные команды операционной системы, т.е. реагировать на атаки.

В состав системы входит утилита snort2bro, которая транслирует сигнатуры Snort в сценарии Bro. Кроме трансляции, данная утилита выполняет оптимизацию сигнатур под анализатор Bro.

2.4.2. OSSEC

OSSEC HIDS является открытой узловой системой обнаружения атак [79]. В её задачи входит: анализ журналов, контроль целостности, обнаружение закладок, оповещение об атаках и активная реакция на атаки.
Система может быть установлена как в одиночной конфигурации на одном узле, так и в распределенной конфигурации на нескольких узлах – в таком случае одна из инсталляций становится сервером, а остальные – агентами системы. При этом управление агентами выполняется централизованно с сервера.

В состав системы входят несколько различных анализаторов. Анализатор журналов использует файлы журналов типовых приложений для UNIX-систем, системные журналы Windows и некоторых приложений (IIS).

Модуль обнаружения закладок сканирует файловую систему узла и ищет известные закладки по сигнатурам, а также неизвестные закладки и закладки на уровне ядра на основе обнаружения аномалий.

Модуль контроля целостности выполняет проверку наиболее критичных системных файлов (исполняемые, конфигурационные, файлы библиотек и т.п.). При первом запуске данный модуль создаёт базу данных критичных файлов, и сохраняет в нее, помимо самих файлов, информацию целостности: параметры доступа, размер, информацию о владельцах, контрольные суммы MD5 и SHA1. Затем модуль периодически производит полное сканирование системы и сравнивает системные файлы с копиями в базе данных. В том случае, если какой-либо файл изменился, генерируется сообщение администратору.

Система OSSEC также включает в себя модуль корреляции сообщений об атаках, который расширяет возможности по анализу сообщений системы Snort, удаляет ложные сообщения и позволяет инициировать реакцию на сложные события.

2.4.3. STAT

Система STAT (State Transition Analysis Tool - средство анализа систем переходов) является результатом проекта Калифорнийского Университета, Санта-Барбара, США [34,51,103,104]. Первые публикации по системе датированы 1992 г., последние – 2003 г. Основой используемого системой метода является описание исходной защищаемой системы в виде набора состояний ее компонентов и последующий анализ переходов из состояния в состояние в результате активных внешних воздействий.

Состояния защищаемой системы определяются при настройке и конфигурации системы обнаружения атак. Для каждого состояния определяется характеристика защищенности. Определяются *переходы* – изменения состояния защищаемой системы. Атаки описываются в виде последовательностей переходов. Данный подход также является эвристическим, относительно рассматриваемого в данной работе, и родственен подходу, используемому в экспертных системах, базирующихся на сигнатурах атак. Описание атак в виде последовательности переходов терминов состояний призвано избежать традиционных ограничений методов, основанных на сигнатурах и дать возможность описать шаблоны для целых классов типовых атак.

Основой системы является язык STATL – расширяемый язык, предназначенный для описания шаблонов атак в терминах STAT. Базовый язык оперирует наиболее абстрактными понятиями, не зависящими от конкретной системы и ее конфигурации. Язык позволяет достраивать себя, добавляя специфичные для конкретной системы *события*. Для каждого нового события описывается его *предикат*. К примеру, для расширения языка с целью определения событий, характерных для веб-сервера Apache, необходимо определить события, описывающие появляющиеся в журналах данного приложения записи. То есть, событие будет иметь поля *host, ident, authuser, request, status* и прочие, определенные в Apache's Common Log Format. После этого требуется описать предикаты интересующих событий. Например, предикат isCGIrequest() будет возвращать *true*, если имел место вызов CGI-сценария. Описания событий и предикатов группируются в Language Extension Module (модуль расширения языка) и в последствии их можно использовать в описании сценариев атак для STAT. Поток реальных событий сравнивается со сценариями ядром STAT. Все конструкции языка STATL и его расширения транслируются в язык C++.

Основной функциональной частью системы является *ядро* STAT. Этот компонент оперирует абстрактными объектами и событиями, не зависящими от

конкретной системы. Она сравнивает входящий поток событий с имеющимися сценариями атак и выполняет непосредственно функцию обнаружения. Для генерации

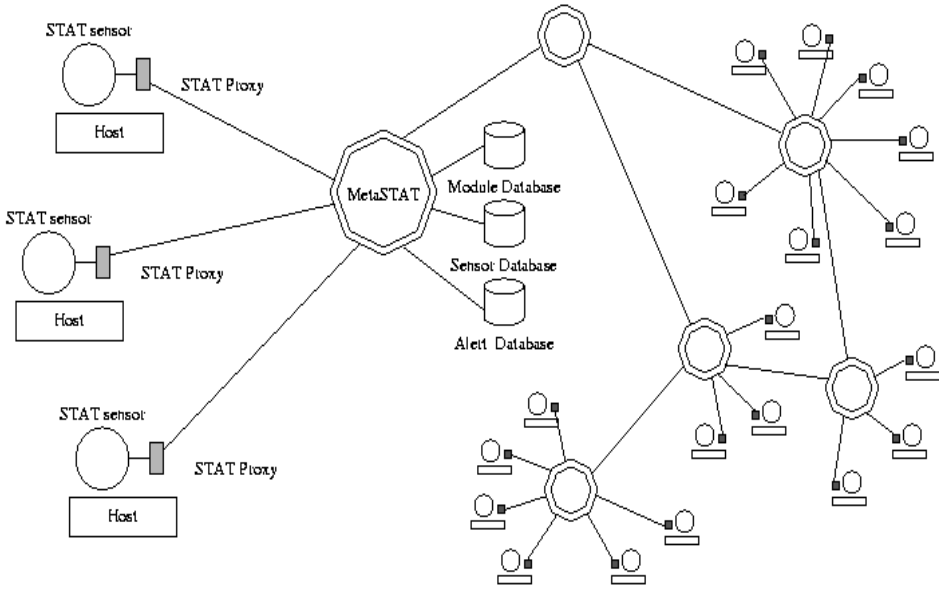

Рис. 1. Структура системы STAT.

потока событий используется *источник событий* – программный компонент системы обнаружения, осуществляющий преобразование информации из системных источников, таких как журналы регистрации, в формат, пригодный для функционирования ядра STAT.

В системе также могут использоваться *модули реакции* – связанные с ядром компоненты, осуществляющие реагирование на обнаруженную атаку.

Данная система является открытой и позволяет строить масштабируемые системы обнаружения. На основе STAT строятся агенты или сенсоры системы обнаружения, все остальное – вопросы архитектуры и взаимодействия агентов в распределенной системе. Этому и посвящены все работы xSTAT, которые применяют данную технологию для конструирования систем обнаружения атак различных типов – узловые, сетевые.

На рис. 1 представлена архитектура системы обнаружения атак, построенной на основе STAT.

Основные компоненты:

STAT sensor – модуль обнаружения атак на узле на основе ядра STAT.

STAT proxy – модуль, связывающий сенсоры с центральным модулем MetaSTAT.

MetaSTAT – модуль сбора информации об атаках, уведомления администратора, хранения информации об атака.

Языком реализации системы NetSTAT является язык C++. Она работает под управлением ОС Linux и Solaris.

2.4.4. Prelude

Система Prelude является системой с открытыми исходными текстами. Начало разработки – 1998 год. Она изначально задумывалась как гибридная СОА, которая могла бы помочь администратору сети отслеживать активность как на уровне сети, так и на уровне отдельных узлов. Система распределенная и состоит из следующих компонентов [103]:

сетевые сенсоры – различные сенсоры, анализирующие данные на уровне сети на основе сигнатурного анализа. Сенсоры генерируют сообщения об обнаружении атак и отправляют их модулям управления. Система Prelude использует в качестве сетевого сенсора систему Snort;

узловые сенсоры – различные сенсоры уровня системы, анализирующие журналы регистрации ОС, приложений. Сенсоры генерируют сообщения об обнаружении аномалий и отправляют их модулям управления. Существующий набор сенсоров позволяет анализировать данные журналов регистрации таких систем и приложений, как межсетевй экран IPFW, входящий в состав ОС FreeBSD, NetFilter ОС Linux 2.4.x, маршрутизаторы Cisco и Zyxel, GRSecurity, и типовые сервисы ОС UNIX.;

модули управления – процессы, которые получают и обрабатывают сообщения сенсоров. Различаются следующие виды модулей управления:

- модули журнализации – отвечают за регистрацию сообщений в журналах регистрации или базах данных. В настоящее время реализованы модули для MySQL, PostgreSQL;
- модули реагирования – анализируют сообщение и генерируют возможную ответную реакцию СОА на атаку. Возможны такие виды реакции как блокирование нарушителя на межсетевом экране (NetFilter, IPFilter). В дальнейшем возможны такие типы реакции как изоляция нарушителя и сужение пропускной способности канала нарушителя;

агенты реагирования – реализуют сгенерированную менеджером реакцию на атаку;

интерфейс – основан на протоколе http. Предоставляет возможность получать статистику и управлять системой при помощи web-браузера.

На рис. 2. представлена логическая схема соединения компонентов COA Prelude.

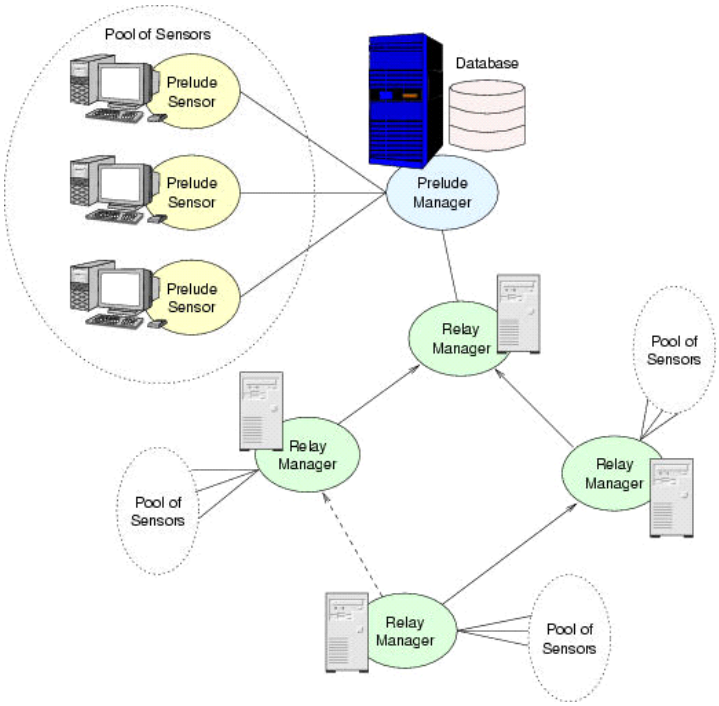

Рис. 2. Логическая структура COA Prelude.

У системы Prelude есть несколько особенностей, которые отличают ее от других современных открытых COA. Система везде, где возможно, построена на использовании открытых стандартов. Так, для обмена сообщениями используется формат IDMEF (Intrusion Detection Message Exchange Format), оптимизированный для высокоскоростной обработки. Это позволяет в дальнейшем интегрировать компоненты в системы сторонних производителей и наоборот.

При разработке системы особое внимание было уделено вопросам безопасности. Каналы передачи данных шифруются по протоколу SSL, кроме того, используется специализированная библиотека, которая предотвращает классические ошибки выхода за границы массивов и переполнения буферов.

Дополнительные модули анализа сетевых данных делают систему устойчивой к некорректным сетевым пакетам на разных уровнях стека и выходу ее компонентов из строя. Такие атаки как отправка пакетов с неправильными контрольными суммами, обнуленными флагами TCP, ресинхронизация сессий, случайная отправка и «обрезание» сегментов системой игнорируются и не приводят к отказу компонентов COA.

2.4.5. Snort

Система Snort является классическим продуктом с открытыми исходными текстами [89]. Разработку данной системы начал один автор, но благодаря открытой архитектуре и открытым исходным текстам, система стала быстро развиваться за счет других разработчиков, и, кроме того, интегрироваться с прочими программными продуктами, такими как базы данных для ведения журналов обнаружения, анализаторы журналов регистрации.

Snort обнаруживает атаки исключительно на основе анализа сетевого трафика. Основным методом обнаружения атак, используемым в системе, является обнаружение

злоупотреблений на основе описания сигнатур атак. В системе используется простой язык описания сигнатур атак, который полностью описан в документации и позволяет администраторам системы дополнять базу сигнатур своими сигнатурами. Каждое правило на этом языке состоит из двух частей: условие применения и действие.

Пример правила системы Snort:

alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf"; msg: "PHF probe!";)

Это правило определяет, что любой сегмент TCP, направленный на порт 80 на любой адрес в сети 10.1.1.0/24, и при этом имеющий в поле данных строку "/cgi-bin/phf", является подозрительным и необходимо послать уведомление администратору.

Кроме того, в последних версиях системы появилась специальная конструкция языка сигнатур, позволяющая классифицировать сетевой трафик по степени потенциальной опасности. Степень опасности определяется экспертом, который формирует сигнатуру атаки.

В настоящее время система находится в стадии активной разработки: каждые несколько месяцев появляются новые версии системы и новые функции.

Архитектура системы Snort целиком разрабатывалась из соображений эффективности и скорости работы. Поэтому она предельно проста и состоит из следующих подсистем: декодер пакетов, ядро обнаружения и подсистемы оповещения и реагирования. Декодер пакетов реализует набор процедур для последовательной декомпозиции пакетов в соответствии с уровнями сетевого стека, то есть принятый кадр последовательно преобразуется в пакет, сегмент и блок данных с применением специфичных для данного уровня сигнатур атак. В настоящее время поддерживаются протоколы канального уровня Ethernet, SLIP, PPP, ожидается поддержка ATM. Ядро выстраивает имеющиеся правила в т.н. *цепи правил* – двумерные последовательности правил, где правила с общей частью условий применения объединяются в одно звено цепи, а несовпадающие компоненты правил строятся цепью во втором измерении от полученного звена. Это сделано для ускорения анализа сетевого трафика. Каждый пакет проходит по цепочке от корня, первое подходящее правило выполняет свой блок действий и проход завершается. На рис. 3. представлен небольшой пример такой цепи правил.

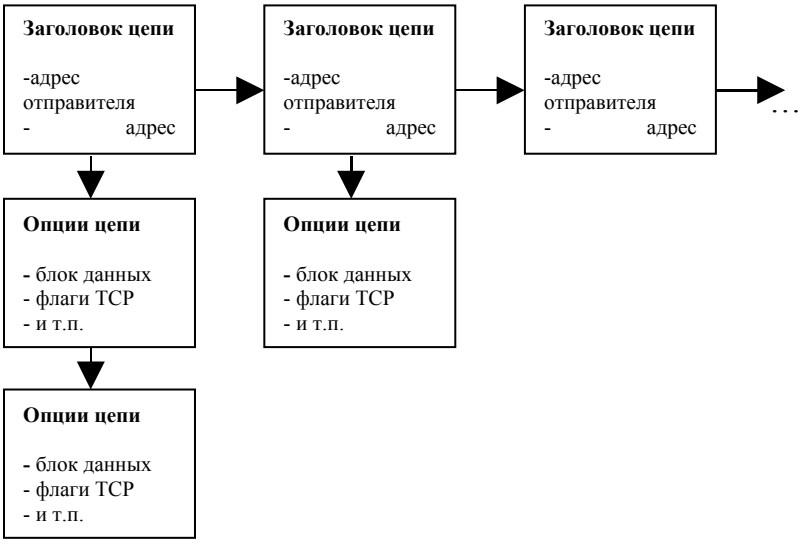

Рис. 3. Цепи правил системы Snort.

Кроме модуля анализа трафика на основе правил, к ядру обнаружения могут подключаться модули сторонних разработчиков (препроцессоры) и производить анализ на одном из уровней декомпозиции пакета. С помощью таких модулей можно добавлять функциональность ядру обнаружения атак и реализовывать различные методы обнаружения. Препроцессоры выполняют декомпозицию сетевого трафика и проверку соответствия спецификациям протоколов, дефрагментацию и т.п. В поставку системы входит несколько модулей, например, модуль обнаружения сканирования портов, модуль обнаружения Unicode-атаки на веб-сервер компании Microsoft и другие. Кроме того, в одной из версий в составе Snort был модуль статистического анализа, который предназначен для обнаружения аномалий в сетевом трафике.

Подсистема оповещения и реагирования отвечает за сохранение результатов анализа трафика в журналы регистрации самой системы Snort, либо вывод этой информации через системные службы регистрации событий ОС. Например, в UNIX-подобной ОС это может быть сервис регистрации событий *syslog*.

Система Snort реализована под множество UNIX платформ.

2.5. Заключение и выводы

Проведенный обзор и сравнительный анализ методов и систем обнаружения атак позволяют сделать вывод о том, что в настоящий момент не существует открытой общедоступной системы обнаружения, которая обладала бы адаптивностью к неизвестным атакам. Несмотря на наличие большого числа методов обнаружения аномалий, существенное количество ложных срабатываний, слабая устойчивость и неверифицируемость не позволяет их использовать в системах общего назначения. Кроме того, до сих пор использование методов обнаружения аномалий ограничено исследовательскими и узкоспециализированными системами. Например, делаются попытки использовать методы классификации данных, такие как кластерный анализ и нейронные/иммунные сети. Главный недостаток этих методов, в сравнении с экспертными методами и анализом систем переходов, в принципиальной невозможности верификации результата (выхода) – классы поведения строятся на основе обучения и не представляется возможным аналитически рассчитать уровень ошибок I и II рода для таких методов, а также проанализировать корректность принимаемых решений.

Для решения данной проблемы необходимо разработать адаптивный метод обнаружения атак, который при низкой (линейной) вычислительной сложности, устойчивости и верифицируемости будет иметь низкий уровень ложных срабатываний.

Также можно сделать вывод о том, что в области защиты от атак наблюдается переход от обнаружения атак к предотвращению атак: все рассмотренные открытые СОА уже включают в себя средства реагирования на атаки, которые в обязательном порядке включают в себя разрыв соединений и взаимодействие с внешними средствами

защиты – межсетевыми экранами и т.п. Данная тенденция, в сочетании с постоянным повышением пропускной способности каналов передачи данных, предъявляет повышенные требования к вычислительной сложности алгоритмов обнаружения атак. При этом большинство методов обнаружения аномалий имеет высокую вычислительную сложность по сравнению с наиболее распространенным сигнатурным методом.

В свете вышеизложенного представляется перспективным разработать гибридный метод обнаружения атак, который объединит сигнатурный метод и метод анализа систем переходов для обнаружения отклонений от нормального поведения. Объединение этих методов поможет сохранить верифицируемость, устойчивость и низкую вычислительную сложность при обнаружении злоупотреблений и дополнить их свойством адаптивности к неизвестным атакам. Наличие данных свойств позволит использовать метод в системах обнаружения атак общего назначения, а также в автономных системах, для которых все перечисленные выше свойства являются критичными.

3. МОДЕЛЬ ОБНАРУЖЕНИЯ АТАК

В данном разделе приведена модель функционирования РИС и формальная постановка задачи обнаружения атак в терминах данной модели.

Сегодня в области обнаружения и предотвращения компьютерных атак нет формальной модели, которая позволила бы оценивать эффективность и корректность предлагаемых решений. Большинство известных систем используют неформальные модели и методы [6,7,8,64,69]; например, сигнатурные методы, для которых нельзя получить теоретические оценки эффективности, показать корректность, завершаемость [6] и т.д. Модель, предлагаемая в данной работе, призвана устранить этот пробел. В основу этой модели легли идеи работ [34,93,104,110].

Следует упомянуть ряд работ, в которых решается похожая задача построения модели функционирования информационной системы для исследования свойств защищенности системы. В работе Шейнера [93] рассматриваются вопросы автоматического поиска уязвимостей и верификации защищенности приложений и протоколов с помощью графов атак и эмпирических критериев допустимого поведения системы. Построенная модель полна с точки зрения описания возможных отказов системы. Тем не менее сложность предложенного метода не позволяет использовать его для решения задачи обнаружения атак – лишь для аудита безопасности или расследования инцидентов.

Работы Энкмана, Вигны и Кеммерера [34,104] посвящены языку STATL и модели компьютерных атак, основанной на нем. Модель атак базируется на *состояниях* и *переходах*. *Состояния* соответствуют состояниям защищаемой системы. *Переходы* между состояниями соответствуют различным действиям, переводящим систему из состояния в состояние. Недостаток модели в том, что она не имеет критериев для разделения множеств атак и нормального поведения.

Работа Городецкого и Котенко [39] посвящена построению формальной модели атак. Авторы рассматривают атаку как стохастический процесс планирования действий, результат которого зависит от успешности предыдущих и ряда случайных факторов. Модель интересна тем, что атака моделируется с точки зрения атакующего и оперирует формализованным понятием цели атаки.

Описанная в данной работе модель создавалась как модель предметной области: функционирование распределенной информационной системы в условиях злонамеренных воздействий (атак), для обоснования корректности и оценки вычислительной сложности методов обнаружения атак.

3.1. Модель функционирования РИС

В данной работе термины «несанкционированный доступ» и «несанкционированное воздействие» мы будем понимать в соответствии с ГОСТ Р 50922-96 [109], а именно:

Защита информации от несанкционированного воздействия – деятельность по предотвращению воздействия на защищаемую информацию с нарушением установленных прав и/или правил на изменение информации, приводящего к искажению, уничтожению, копированию, блокированию доступа к информации, а также к утрате, уничтожению или сбою функционирования носителя информации.

Защита информации от несанкционированного доступа – деятельность по предотвращению получения защищаемой информации заинтересованным субъектом с нарушением установленных правовыми документами или собственником, владельцем информации прав или правил доступа к защищаемой информации.

Взаимодействие объектов – выполнение операций доступа одного объекта к другому. Например, чтение, запись, запуск на выполнение и т.д.

3.1.1. Основные понятия и определения

Представим распределённую информационную систему (далее РИС) как множество взаимодействующих экземпляров типов объектов (далее просто объектов). Для каждого типа объекта определены операции доступа (далее просто операции, где не возникает неоднозначности), например, по открытию, чтению, записи, запуску на исполнение и т.д. Доступ к объекту может быть как непосредственным, так и косвенным – через другие объекты. Доступ по некоторой операции к объекту подразумевает выполнение последовательности элементарных операций – открыть доступ по операции, выполнить операцию, закрыть доступ по операции. Предполагаем, что в ходе реализции доступа элементарные операции выполняются мгновенно, но могут отстоять друг от друга во времени.

Для каждого объекта также определен конечный набор прав доступа – правила, определяющие, к каким типам объектов он может иметь доступ. Любой доступ к объекту, не соответствующий этим правилам, будем называть *несанкционированным*. Задача систем защиты РИС – выявление несанкционированного доступа к объектам РИС и противодействие им, либо попыткам несанкционированного изменения прав доступа. Будем называть такие действия злоумышленными, а объект, инициирующий такие действия – злоумышленником или нарушителем. Выявление атаки может происходить в процессе ее реализации или после ее завершения.

Типы объектов РИС могут быть программными или аппаратными.

К аппаратным типам объектов относятся:

- локальные аппаратные средства узлов ЛВС;
- физические каналы связи между узлами ЛВС;
- каналообразующая аппаратура ЛВС.

К программным типам объектов относятся:

- узловые объекты – программные объекты на узлах РИС, взаимодействующие в рамках одного узла;
- сетевые объекты – программные объекты на узлах РИС и внешние программные объекты, взаимодействующие с узлами РИС;

Каждый объект в системе характеризуется *состоянием*. Состояние объекта – это множество объектов РИС, имеющих доступ к нему, а также характеристика его текущей загруженности. Далее мы строго определим понятие состояния объекта.

Объекты можно разделить на два подмножества: подмножество активных и подмножество пассивных объектов. Пассивный объект не может осуществлять доступ к другим объектам, тогда как активный может.

Обозначим:

Множество *Act* – множество типов активных объектов РИС;

Множество $\Re_C$ – множество экземпляров активных объектов РИС;

Множество *Psv* – множество типов пассивных объектов РИС;

Множество $\Re_P$ – множество экземпляров пассивных объектов РИС;

$\Re = \Re_C \cup \Re_P$;

Предполагаем, что $\Re_C \cap \Re_P = \varnothing$.

Как уже было сказано, над объектом $r \in \Re$ каждого типа определены операции доступа. Например, по чтению, записи, запуску на выполнение и т.п. Эти операции, как было отмечено выше, состоят из последовательности элементарных операций, так называемых примитивов. Будем предполагать, что каждая операция доступа состоит из двух примитивов: открыть и закрыть. Например, в случае доступа по чтению – открыть такой-то объект на чтение, закрыть такой-то объект на чтение. Выполнить примитив «закрыть» можно только в том случае, если был ранее выполнен примитив «открыть».

Обозначим a_r операцию доступа к объекту *r*, $\underline{a}_r$ - примитив «открыть доступ по операции *a*», а $\overline{a}_r$ - «закрыть доступ по операции *a*».

Обозначим *r* экземпляр объекта типа type. Тогда

$r = < name, type, \xi, A, D >$, где

name – идентификатор (имя экземпляра объекта),

type – тип объекта ($type \in Act \cup Psv$*),*

ξ - загрузка экземпляра объекта,

A – набор операций доступа, определенных для данного типа объектов,

D – порог загрузки объектов данного типа.

Определим ξ как функцию на $\Re$ со значениями в [0,1]; *D* как функцию на $\Re$ со значениями в [0,1]. Под значением $\xi(r)$ будем понимать текущую загрузку объекта $r \in \Re$.

Будем считать, что если $\xi(r) > D(r)$, то поведение объекта $r \in \Re$ не определено, а его состояние – опасное. Понятия поведения объекта и его состояния будут уточнены ниже.

Будем предполагать, что для каждой пары экземпляров объектов (r_i, r_j), где $r_i \in \Re_C, r_j \in \Re$, и операции доступа $a \in A_{r_j}$ определено понятие права объекта r_i иметь доступ к объекту r_j по операции *a* – предикат, который будем обозначать $a(r_i, r_j) \equiv 1 \Leftrightarrow a \in A_{r_j} u(a, r_i, r_j) \in w(r_i, r_j)$, где $w(r_i, r_j) = \{(a, r_i, r_j) \mid a \in A_{r_j}, r_i \in \Re_C, r_j \in \Re\}$ - отношение, которое мы будем называть «иметь право доступа». Отметим, что это отношение не симметрично по $r \in \Re$ и не транзитивно. Тройку $(a, r_i, r_j) \in w(r_i, r_j)$ будем обозначать $w_a(r_i, r_j)$, т.к. это отношение на $\Re_C \times \Re$.

С операциями доступа, кроме отношения «иметь право», связано ещё одно отношение – «иметь доступ». Это отношение возникает между экземплярами объектов $r_i \in \Re_C$ и $r_j \in \Re$, когда экземпляр объекта r_i выполнил операцию доступа $\underline{a}_{r_j}$ над экземпляром объектом r_j. Будем обозначать это отношение на $\Re_C \times \Re$ как $r_i \xrightarrow{a} r_j$, где $r_i \in \Re_C, r_j \in \Re$. Предположим, что $r_i \xrightarrow{a} r_j \Leftrightarrow \exists w_a(r_i, r_j) \in w(r_i, r_j), m.e. a(r_i, r_j) \equiv 1$.

Как мы уже упоминали, доступ может быть не только непосредственный, но и транзитивный. Определим транзитивный доступ по операции *a* через транзитивное замыкание отношения $\xrightarrow{a}$. Пусть существует множество экземпляров объектов $C = \{r_k \in \Re\}, 1 < k < N$ таких, что $r_1 \xrightarrow{a} r_2, ..., r_{N-1} \xrightarrow{a} r_N$. Транзитивным замыканием отношения $\xrightarrow{a}$ назовём отношение $r_i \xrightarrow{a^*} r_j$, которое возникает между экземплярами объектов $r_i \in \Re_C$ и $r_j \in \Re$, когда r_i выполнил операцию $\underline{a}_{r_1}$, r_N выполнил операцию $\underline{a}_{r_j}$ и r_i косвенно выполнил операцию $\underline{a}_{r_j}$. Множество *C* будем называть цепочкой транзитивного доступа от экземпляра объекта r_i к экземпляру объекта r_j .

Транзитивное замыкание описывает реально наблюдаемую ситуацию, когда один объект использует несколько других объектов через цепочку операций доступа. Например, удаленный клиент, обращаясь к терминальному серверу sshd, запускает на выполнение экземпляр такого сервера, который в свою очередь запускает на выполнение программную оболочку (bash), которая также по командам от пользователя может запускать новые процессы, и все они будут использовать пассивные объекты – процессор, память, исполняемые файлы на дисках. Набор последовательностей

объектов от клиента до конечного пассивного объекта по операции «выполнение» и будет транзитивным замыканием данной операции.

Определим подмножество множества активных объектов РИС, для которых отношение «иметь доступ» по какой-либо операции всегда транзитивно:

$$\dot{\Re}_C = \{r \in \Re \mid \exists a : \forall r_i, r_j \in \Re : r_i \xrightarrow{a} r, r \xrightarrow{a} r_j, a(r_i, r), a(r, r_j) \Rightarrow r_i \xrightarrow{a^*} r_j\}$$

Для рассмотренного выше примера такими объектами будут являться экземпляр терминального сервера sshd и программная оболочка bash. Любая цепочка транзитивного доступа по некоторой операции содержит хотя бы один экземпляр элемента множества $\dot{\Re}_C$.

Теперь введем понятие состояния экземпляра объекта РИС: *Состоянием экземпляра* пассивного объекта r_p будем называть тройку:

$S_{r_P} = (In(r), I(\xi(r_p)))$, где:

$In(r) = \{r \in \Re_C \mid \exists a \in A_{r_p} : r \xrightarrow{a} r_P\}$ - множество экземпляров объектов, имеющих непосредственный доступ к r_p,

$I(\xi(r_p)) = [0,1,2]$ - индикатор загрузки экземпляра объекта r_p. I принимает значение 0, если загрузка меньше пороговой, 1 – если загрузка больше пороговой, но меньше емкости объекта и 2 – если загрузка равна емкости объекта.

Состояние экземпляра активного объекта r_c:

$S_{r_c} = (In(r_c), Out(r_c), I(\xi(r_c)))$, где:

$In(r_c) = \{r \in \Re_C \mid \exists a \in A_{r_c} : r \xrightarrow{a} r_c\}$ - множество экземпляров активных объектов, осуществляющих непосредственный доступ к r_c ,

$Out(r_c) = \{r \in \Re \mid \exists a \in A_{r_c} : r_c \xrightarrow{a} r\}$ - множество экземпляров объектов, к которым r_c осуществляет непосредственный доступ,

$I(\xi(r_c))$ - индикатор загрузки экземпляра объекта r_c (аналогично экземпляру пассивного объекту).

Из данных определений состояния экземпляра объекта следует, что при выполнении операции доступа некоторого экземпляра объекта РИС к экземпляру другого объекта РИС изменяются состояния обоих экземпляров.

Обозначим $\Omega = \{S_r \mid r \in \Re\}$ множество всех возможных состояний экземпляров объектов РИС. Из вышесказанного следует, что данное множество конечно.

3.1.2. Модель поведения объекта и модель атаки

Теперь от определения отношений между экземплярами объектов РИС перейдём к описанию функционирования РИС. Назовём *траекторией* t_r *экземпляра объекта r* некоторую непустую конечную последовательность состояний экземпляра объекта *r*, замкнутую слева, т.е. если $t_r = S_1, S_2, ..., S_k \in \Omega$ - траектория, то

$$\forall\, i:\ i \le k \Rightarrow S_1, S_2, ..., S_i \text{ – траектория.}$$

Траекторию экземпляра активного объекта *r* можно представить как последовательность отрезков траекторий взаимодействующих с ним экземпляров объектов и собственных действий экземпляра объекта над экземплярами других объектов РИС, так как любая операция открытия или закрытия доступа вызывает изменение состояния взаимодействующих экземпляров объектов. Траекторию экземпляра пассивного объекта *r* можно представить как последовательность отрезков траекторий взаимодействующих с ним экземпляров активных объектов.

Определим поведение экземпляра объекта как множество всех возможных его траекторий:

$$Bh(r) = \{t_r\}.$$

Определим поведение типа объекта как множество всех возможных траекторий экземпляров объекта данного типа:

$$Bh(type) = \bigcup_{\Re_C} Bh(r).$$

Назовём состоянием РИС совокупность множеств экземпляров объектов РИС и их состояний:

$$\theta = < \Re, \{S_r \mid r \in \Re\} >$$

Определим траекторию РИС T_{sys} как непустую конечную последовательность состояний РИС, замкнутую слева, в которой любые два соседних состояния РИС отличны друг от друга.

Функционирование РИС представим как некоторую последовательность состояний РИС, такую, что каждое последующее состояние отличается от предыдущего состоянием хотя бы одного из объектов или объектов РИС. Для этого введем понятие полной траектории РИС.

Траекторию РИС будем называть полной, если для любых двух соседних состояний РИС θ_1 и θ_2 выполняются следующие условия:

1. Если $\theta_1 = < \Re_1, \bigcup_{r \in \Re_1} S_r >$ и $\theta_2 = < \Re_2, \bigcup_{r \in \Re_2} S_r >$, то либо $\Re_1 \neq \Re_2$, либо $\bigcup_{r \in \Re_1} S_r \neq \bigcup_{r \in \Re_2} S_r$, но не одновременно.
2. Если $\bigcup_{r \in \Re_1} S_r \neq \bigcup_{r \in \Re_2} S_r$, то $\forall r \in \Re_2 : S_r^1 \neq S_r^2 \Rightarrow \exists r' \in \Re_1, a \in A_r : r' \xrightarrow{a} r$, т.е. для любого экземпляра объекта РИС, такого что его состояния в соседних состояниях РИС различны, существует взаимодействующий с ним экземпляр объекта РИС в более раннем состоянии и его действие, вызвавшее смену состояний.

Эти условия означают, что если некоторая траектория РИС $\theta_1, \theta_2, ..., \theta_N$ полна, то она содержит все траектории экземпляров объектов, входящих в $\Re_1 \cup \Re_2 \cup ... \cup \Re_N$.

На множестве состояний отдельных экземпляров объектов, входящих в некоторую траекторию РИС, можно ввести время, как отношение частичного порядка:

$$\forall S_r^i, S_r^j \in T_{sys} : \exists r', a : S_r^i \xrightarrow{r', a} S_r^j \Rightarrow S_r^i \prec S_r^j, S_r^i \prec S_r^j$$

$$\forall S^i, S^j, S^k \in T_{sys} : S^i \prec S^j, S^j \prec S^k \Rightarrow S^i \prec S^k$$

При этом будем говорить, что одно состояние есть причина, а другое - следствие, если в траектории первое состояние предшествует второму.

Теперь определим понятия *опасного и безопасного* состояния экземпляра объекта РИС. Будем говорить, что РИС находится в информационно безопасном состоянии, если все экземпляры объектов РИС находятся в информационно безопасном состоянии. В соответствии со стандартом [8], информационно безопасное состояние РИС – это множество таких состояний РИС, в которых отсутствуют следующие нарушения: нарушения конфиденциальности экземпляров объектов РИС, нарушения целостности экземпляров объектов РИС, нарушения доступности экземпляров объектов РИС [2,5]. В терминах данной модели информационно безопасное состояние РИС – это множество состояний РИС, в которых все экземпляры объектов, имеющие доступ к другим экземплярам объектов РИС, имеют права доступа, и загрузка каждого объекта меньше его ёмкости.

Состояние экземпляра объекта будем считать *безопасным*, если:

$$\forall r', a \in A_r : r' \xrightarrow{a} r \Rightarrow a(r', r), 0 < \xi(r) < D(r)$$

Обозначим N_r множество безопасных состояний экземпляра объекта r.

Состояние экземпляра объекта будем считать *опасным*, если:

$$\xi(r) = 1, либо\ \exists r', a \in A_r : r' \xrightarrow{a^*} r, \neg a(r', r)\}$$

Обозначим $\aleph$ множество опасных состояний экземпляра объекта *r*.

Пример опасного состояния: при исчерпании максимального числа открытых TCP-соединений можно говорить о наличии атаки, нарушающей доступность объекта (атаки типа DoS). Определим нарушение информационной безопасности РИС как перевод РИС из некоторого безопасного состояния в любое опасное.

Множество опасных состояний РИС - это множество состояний, в которых хотя бы один экземпляр объекта находится в опасном состоянии. Переход любого экземпляра объекта в опасное состояние по определению означает переход всей РИС в опасное состояние. Такое определение опасного состояния РИС означает, что для его обнаружения нет необходимости собирать информацию о состоянии каждого объекта РИС в каждый момент времени, а достаточно наблюдать только за изменениями состояний критичных, с точки зрения защиты информации, объектов.

Следует отметить, что любому состоянию РИС можно сопоставить двудольный граф доступа $\Gamma = (V_A + V_P \mid A)$, где:

V_A – множество вершин, соответствующих экземплярам активных объектов РИС;

V_P – множество вершин, соответствующих экземплярам пассивных объектов РИС;

A – множество упорядоченных пар вершин:

$(v', v''), v', v'' \in \Re_A \vee v' \in \Re_A, v'' \in \Re_P$

Вершина (v', v'') принадлежит A, если $\exists a : v' \xrightarrow{a} v''$.

На рис. 4. показан граф доступа на множестве экземпляров объектов РИС в некотором состоянии РИС.

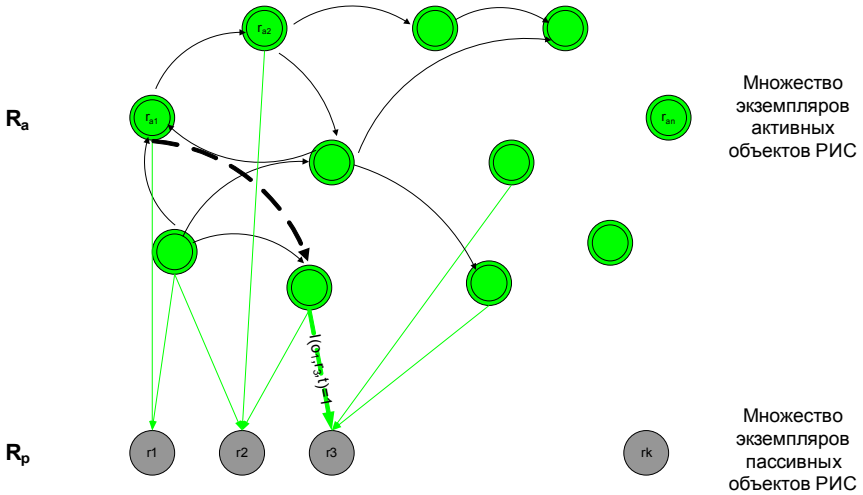

Рис. 4. Граф доступа в РИС в некотором состоянии РИС.

Если в состоянии *S* объект r' и объект *r* принадлежат некоторой цепочке транзитивного доступа по некоторой операции *a*, то в графе доступа существует путь от объекта r' к объекту *r (в силу определения графа доступа)*.

Атакой на РИС назовём такую последовательность действий, производимых некоторым экземпляром объекта, или набор последовательностей действий группы экземпляров объектов, что:

$$X = x_1, x_2, ..., x_K, \forall x_i \exists r \in \Re : x_i \in t_r, \exists 1 \le j \le K : \forall i \ge j S_j \in \aleph .$$

Таким образом, атака – это траектория некоторого экземпляра объекта или набор участков траекторий некоторой группы экземпляров объектов, выводящая РИС из информационно безопасного состояния. Атака переводит систему из безопасного состояния в опасное, либо оставляет систему в ранее достигнутом опасном состоянии.

Нормальное поведение объекта *r* определим как множество траекторий всех экземпляров объектов данного типа *type(r),* которые не выводят РИС из состояния информационной безопасности [1]. Будем считать, что ни одна траектория, входящая в нормальное поведение объекта, не может содержать опасных состояний, т.е. состояний, в которых ресурс перегружен или существует цепочка транзитивного доступа, нарушающая отношение прав доступа.

Предлагаемый метод обнаружения атак использует множество примеров траекторий атак и описаний нормального поведения для обнаружения злоупотреблений и аномалий поведения объектов на основе сопоставления наблюдаемого поведения экземпляров объектов с заданными примерами.

3.2. Формальная постановка задачи обнаружения атак

Теперь в терминах, упомянутых в разделе 1, подходов к обнаружению атак, задачу обнаружения атаки можно разделить на две подзадачи: обнаружение злоупотреблений и обнаружение аномалий как отклонения от нормального поведения.

1. Обнаружение злоупотреблений.

Пусть задано:

- множество примеров атак $X = \{X\}, X^i = x_1^i, x_2^i, ..., x_N^i$;
- последовательность наблюдаемых состояний защищаемой системы $T_{sys} = \theta_1, \theta_2, ..., \theta_N, ...$.

Требуется найти множество экземпляров объектов $O^* = \{o\} \subset \Re_C$ и соответствующее им множество траекторий $T^* = \{T_o \mid o \in O^*\}$, которые реализуют атаки из множества примеров атак.

2. Обнаружение аномалий.

Пусть задано:

- множество описаний нормального поведения объектов РИС, определенное в терминах типов объектов $B = \{Bh(Type) \mid Type \in Act \cup Psv, \forall r \in \Re : type(r) = Type, \forall S \in t_r \; S \in N_r\}$;
- последовательность наблюдаемых состояний защищаемой системы $T_{sys} = \theta_1, \theta_2, ..., \theta_N, ...$.

Требуется найти множество экземпляров объектов $O^* = \{o\} \subset \Re_C$ и соответствующее ему множество траекторий $T^* = \{T_o \mid o \in O^*\}$, которые не принадлежат описанию нормального поведения для соответствующих типов объектов РИС.

3.3. Распознавание нормальных и аномальных траекторий

Назовем *классом атак* множество атак, результатом которых является одно и то же опасное состояние любого экземпляра объекта *r* определенного типа. Отдельную атаку класса будем называть экземпляром класса. Ясно, что экземпляр атаки соответствует атаке на экземпляр объекта соответствующего типа. Таким образом, для каждого типа объектов РИС определено некоторое множество классов атак. Финальное

опасное состояние объекта типа *type(r)* для *i*-го класса атак на объект типа *type(r)* обозначим K_r^i.

Класс атак – это набор реализаций атак данного класса, где реализация атаки представляет собой траекторию в терминах типов объектов, удовлетворяющую признакам атаки заданного вида. Траектории представляют собой конечные последовательности действий любых объектов указанных типов. Будем рассматривать класс атак как язык, в котором траектории являются словами, а действия объектов – алфавитом.

Будем рассматривать задачу обнаружения атак как задачу обнаружения подцепочек в некоторой наблюдаемой последовательности символов.

Сопоставим каждому классу атак I конечный автомат *первого рода* таким образом, чтобы он принимал любую последовательность состояний экземпляров объектов из последовательности $\theta_1,\theta_2,...,\theta_{M-1},\theta_j \in \Theta, 1 \le j \le M$, которая оканчивается состоянием K_r^i и содержит все условно опасные состояния, которые предшествуют K_r^i. После построения автомата проводится процедура его минимизации.

Для решения задачи распознавания атак необходимо по полученному автомату построить новый автомат, который должен распознавать те же последовательности, но без последнего символа (K_r^i). Для этого удаляем терминальное состояние автомата и переходы, ведущие в него, а предшествующие состояния делаем терминальными и сворачиваем в одно терминальное состояние.

Формально автомат для каждого *i*-го класса атак представляет собой пятерку следующего вида:

$$K_r^i : (S, \Sigma, T, s_0, Q), \text{ где}$$

S – множество состояний;
$\sum$ - входной алфавит (множество действий);
T – функция переходов;
s_0 – начальное состояние;
Q – множество заключительных состояний.

Конструктивно автомат первого рода будем строить по следующему алгоритму:

1. Определить множество реализаций атак одного класса (заданное множество примеров атак);
2. Выделить последовательность состояний атакующего объекта и атакуемого объекта, которые входят в любую траекторию атаки из заданного множества реализаций;
3. Каждому состоянию полученной последовательности сопоставить состояние автомата первого рода. Множеству действий, которые ведут в некоторое состояние, сопоставить переходы в это состояние;
4. Заменить конечное (опасное) состояние на терминальное состояние, оставив все переходы в него.

Отметим, что автоматы первого рода позволяют распознавать принадлежность наблюдаемой траектории к некоторому классу атак. В то же время, можно использовать аналогичный механизм для распознавания принадлежности траектории к классу нормального поведения некоторого объекта. Соответственно, в силу конечности множества возможных действий в каждом состоянии объекта, автомат распознавания нормального поведения можно дополнить множеством конечных состояний таким образом, чтобы он принимал все возможные отклонения от нормальной траектории объекта.

Сопоставим каждому объекту РИС конечный автомат *второго рода* таким образом, чтобы он принимал любую последовательность состояний объекта

соответствующего типа, которая заканчивается опасным состоянием и содержит все состояния объекта, которые входят в любую нормальную траекторию для объектов данного типа.

Автомат второго рода строится на основе описания нормального поведения типа объектов. При этом описание нормального поведения объекта должно быть определено заранее в виде спецификации в текстовом виде и реализации в виде текста на языке программирования. Для построения автомата второго рода необходимо дополнить автомат нормального поведения заданного типа объектов дополнительными (опасными) состояниями и для каждого нормального состояния определить множество аномальных переходов, ведущих в опасное состояние. Данная операция всегда возможна в силу того, что множество действий над данным объектом всегда конечно и полностью определяется множеством операций $A = \bigcup_{r_c \in \Re} A_{r_c}$. Для каждого наблюдаемого экземпляра объектов заданного типа необходимо создавать отдельную копию автомата второго рода, где будет происходить анализ соответствия наблюдаемого поведения экземпляра нормальному поведению объектов этого типа.

Общее число конечных автоматов для распознавания N классов атак на M экземпляров объектов равно N*M. Вычислительная сложность распознавания атак конечным автоматом не зависит от числа состояний и определяется лишь от длиной входа. Максимальную длину входа можно оценить как максимальное число одновременно наблюдаемых траекторий (M+K)*l, где K – число типов объектов, т.е. максимальная сложность распознавания будет равна N*M*(M+K)*l. Следовательно, сложность обнаружения N классов атак на M объектов для K объектов с помощью конечных автоматов равна O(N*M^2*K*l), где l – максимальная длина наблюдаемой траектории. Таким образом, сложность распознавания атак конечными автоматами первого и второго рода имеет линейную зависимость от длины наблюдаемой траектории, т.е. от потока анализируемых событий. Следует отметить, что сложность распознающего автомата по памяти для автоматов второго рода будет существенно выше, чем для автоматов первого рода, так как зависит от общего числа возможных состояний объекта соответствующего типа, а не только от тех состояний, которые использует реализация атаки. При использовании подхода на основе обнаружения аномалий существенно возрастают требования по памяти для распознавания атак.

3.4. Язык описания автоматов первого и второго рода

Данный раздел посвящен некоторым аспектам реализации предложенной модели и метода обнаружения атак. Важной особенностью поведения сетевых объектов РИС является способ обслуживания запросов к сервисам по модели взаимодействия клиент-сервер. Для этого при появлении нового «клиентского» объекта порождается новый логический экземпляр «серверного» объекта. В одних РИС это реализуется порождением нового экземпляра объекта на каждый объект, в других – созданием и поддержанием независимых контекстов обслуживания нескольких объектов в одном объекте.

Эта особенность означает, что траектория поведения РИС может иметь сложную структуру: некоторые действия объектов РИС могут создавать экземпляры объектов так, что все последующие действия объектов будут порождать изменения состояний только в этих порождённых экземплярах объектов. Соответственно, существует множество действий, которые ведут к уничтожению экземпляров объектов. Как правило, к порождающим действиям относятся запросы на обслуживание к объекту, а к уничтожающим действиям – любые действия, вызывающие прекращение обслуживания объекта объектом. Кроме того, множеств объектов в РИС могут меняться с течением времени, что требует расширения множества событий автомата событиями таймера.

Таким образом, один «серверный» объект может иметь одновременно несколько траекторий, а «клиентский» объект всегда имеет лишь одну наблюдаемую траекторию. Для каждой РИС можно описать два типа автоматов: автоматы первого рода описывают атаки, а автоматы второго рода описывают нормальное поведение наблюдаемого объекта.

Для описания двух типов автоматов предложен язык описания сценариев атак, который имеет следующие особенности:

- переходы между состояниями типизированы;
- для состояния можно определить логическое условие возможности перехода в данное состояние.

Различаются следующие типы переходов, которые отражают специфику сетевых объектов:

- consuming (поглощающий) – не создается новой копии автомата (все переходы выполняются в текущем);
- nonconsuming (непоглощающий) – создается новая копия автомата, переход осуществляется в ней;
- unwinding (свёртка) – удаляются все автоматы, порождённые данным автоматом с помощью переходов типа nonconsuming.

Также алфавит событий каждого автомата искусственно расширен за счет событий таймера. События таймера – это события специального вида, которые также могут вызывать переход из состояния в состояние.

Для параметризации автомата используются переменные конфигурации, которые могут быть использованы в предикатах состояний и переходов. Множество таких внешних переменных является для автомата глобальным окружением, тогда как набор внутренних переменных и их значений является локальным окружением.

Формально автомат для каждого класса атак A_i представляет собой структуру следующего вида:

$$K_R^i : (S, P_S, T, P_T, s_0, I, g, q)$$, где

S – множество состояний;
P_S – множество предикатов состояний;
T – множество переходов;
P_T – множество предикатов переходов;
s_0 – начальное состояние;
I – множество экземпляров автомата;
g – глобальное окружение;
q – глобальная очередь таймера.

Множество экземпляров $I = \{i\}, i = \sum^+$ - *непустая последовательность срезов*. Каждый срез $\sigma \in \sum : N_K \times L \times Q \times E$ характеризует текущее состояние автомата и содержит имя текущего состояния (элемент пространства имен состояний), локальное окружение, локальную очередь таймера и локальную очередь событий (строка входного алфавита).

Переход выполняется в том случае, если предикаты перехода и целевого состояния принимают значение *true* одновременно.

Пример описания автомата первого класса:

```
/*
* Объявление имени сценария. В скобках указаны типы анализируемых событий -
* события только таких типов отправляются системой прогона автоматов на
* вход данному сценарию.
*/
   scenario MyScenario ( NetTCPEvent tcpEv, TimerEvent ) {
     timer    t;       // объявление локального таймера
     u_int_16 srcPort; // переменная для хранения номера порта

/*
* Объявление начального состояния, в данном случае оно пустое.
*/
     initial state state0 {}

/*
* Объявление рабочего состояния.
*/

     state state1 {
       ids_time  tm; // переменная для инициализации таймера
       tm.sec = 2;
       tm.msec = 0;
       SetTimer ( t, tm ); // устанавливаем таймер на значение переменной
     }
/*
* Переход из начального состояния в state1 по событию прихода TCP-пакета
* на порт 21.
*/
     nonconsuming transition state0->state1
       event NetTCPEvent ( tcpEv.tcpDestPort == 21 ) { // условие перехода
         DebugPrint ( «Соединение на порт FTP\n» );
         srcPort = tcpEv.tcpSrcPort; // сохранение номера порта источника
       }
/*
* Переход-свёртка из состояния state1 в начальное состояние по событию
* повторного прихода TCP-пакета на порт 21 в течение 2 секунл.
*/

     unwinding transition state1->state0
       event NetTCPEvent ( tcpEv.tcpDestPort == 21 &&
                           tcpEv.tcpSrcPort == srcPort ) {
         DebugPrint ( «Повторная попытка соединения в течение 2 сек.\n» );
       }
       event TimerEvent ( t ) {
         DebugPrint ( «В течение 2 сек. Нет данных.\n» );
       }
   };
```

Данный пример описывает автомат, обнаруживающий сетевое соединение на стандартный порт FTP. Ниже приведён фрагмент автомата второго рода, описывающего нормальное поведение сервера FTP proftpd (полностью данный автомат описан в разделе 5.2):

```
/*
*  Сценарий нормального поведения сервера ProFTPd версии 1.2.7+
*  Входные события – пакеты TCP, события таймера, события системных
*  вызовов fork(), vfork(), exec().
*/
   scenario proftpd ( NetTCPEvent te, TimerEvent, NetLinkEvent nl )
   {
```

```
            timer t1; // локальный таймер
            ids_time tm; // переменная для инициализации таймера
            TCPState ts; // переменная для хранения состояния соединения
            initial state statepre {} // начальное состояние
            state state0 {} // первое рабочее состояние
            state login {} // состояние после получения команды login
            state welcome {} // состояние после успешной отправки ответа
            state waitPASS {} // состояние ожидания ввода пароля
            state sendPASS {} // состояние после ввода пароля
            state sendUSER {} // состояние после получения команды USER
            state sendSITEHELP {}//состояние после получения команды SITEHELP
            state wait221 {} // состояние после отправки кода 221
            state wait214SITE {} // состояние после отправки кода 214
            state wait150 {} // состояние после отправки кода 150
            state sendPORT {} // состояние после получения команды PORT
            state sendPASV {} // состояние после получения команды PASV
   …
   consuming transition welcome->wait221//переход по получению команды QUIT
   event NetTCPEvent ( te.CheckState(ts) == -1 && bs(te.tcpPayload,
"QUIT"))
   {
     tm.sec = TIMEOUT;
     SetTimer ( t1, tm );
     d("welcome->wait221", te, 0);
   }
   unwinding transition welcome->state0 // переход по получению ответа 421
   event NetTCPEvent ( te.CheckState(ts) == 1 && bs(te.tcpPayload, "421") )
   {
     d("unw welcome->state0", te, 0);
   }
   …
}
```

Автомат построен по описанию протокола FTP в стандарте RFC 959 и его реализации в сервере proftpd. Для каждого состояния сервера FTP выделено отдельное состояние автомата второго рода. Переходы между состояниями соответствуют переходам сервера FTP в соответствии со спецификацией протокола. При этом в каждом состоянии обнаруживаются нелегальные переходы, которые вызывают порождение нового процесса и исполнение в нём произвольного кода (fork() и exec()). Это позволяет обнаруживать любые атаки класса «remote root» для сервера proftpd, включая ранее неизвестные.

3.5. Алгоритмы обнаружения атак

Алгоритм обнаружения атак сводится к формированию последовательности наблюдаемых состояний РИС и поиску в этой последовательности траекторий атак и отклонений от нормального поведения:

1. Для каждого защищаемого объекта *r* формируем множество автоматов первого рода $K_r = \{K_r^i\}$ (каждый автомат в этом множестве моделирует атаку некоторого класса *i*) и автомат второго рода N_r.
2. Пусть от наблюдателя поступил некоторый символ *a.*
   a. Определяем, к какому множеству действий принадлежит данный символ.
   b. Определяем подмножество автоматов, которые могут принять данный символ.
   c. Подаём символ *a* на вход каждому автомату из выбранного подмножества.
3. Для каждого автомата из выбранного подмножества вычисляем предикаты перехода из текущего состояния по символу *a* и предикат

итогового состояния. Если предикат некоторого итогого состояния «истина» и предикат перехода «истина», и тип перехода consuming, выполняем переход. Если переход имеет тип nonconsuming, порождаем новый экземпляр автомата и выполняем переход в новом экземпляре. При этом старый экземпляр автомата остаётся в предыдущем состоянии. Если тип перехода unwinding, то выполняется переход в итоговое состояние и уничтожаются все экземпляры автомата, порождённые в этом состоянии ранее.

4. Если итоговое состояние автомата является конечным, выполняем действия, определенные для конечного состояния (формируем сообщение об атаке), и уничтожаем данный экземпляр автомата, если он не единственный, иначе возвращаем его в начальное состояние.

Атака на защищаемый объект *r* будет обнаружена при одновременном выполнении следующих условий:

1. Траектория атаки принадлежит одному из классов, распознаваемых множеством автоматов первого рода $K_r = \{K_r^i\}$, либо не принадлежит нормальному поведению объектов данного типа *Bh(type(r))* – это является требованием корректности построения автомата второго рода N_r;

2. Все действия из траектории атаки были наблюдаемы системой прогона распознающих автоматов, при этом действия наблюдались строго в порядке выполнения.

Данные условия являются условиями, в которых предложенный алгоритм обнаружения атак корректен, т.е. для каждой атаки на входе алгоритма результатом выполнения алгоритма является хотя бы одно сообщение об атаке.

В следующем разделе приведено описание экспериментальной системы обнаружения атак, реализующей предложенный метод обнаружения для сетей TCP/IP/Ethernet и операционных систем семейства Linux с ядром версии 2.6.x и Windows 2000/XP.

4. ЭКСПЕРИМЕНТАЛЬНАЯ СИСТЕМА ОБНАРУЖЕНИЯ АТАК

В данном разделе описана экспериментальная система обнаружения атак, реализующая гибридный адаптивный метод обнаружения атак на основе сигнатур и анализа переходов состояний.

Напомним требования к «идеальной» СОА, изложенные в разделе 2:

- покрывает все классы атак (система полна);
- позволяет анализировать поведение защищаемой РИС на всех уровнях: сетевом, узловом и уровне отдельных приложений;
- адаптивна к неизвестным атакам (использует адаптивный метод обнаружения атак);
- масштабируется для РИС различных классов: от небольших локальных сетей класса «домашний офис» до крупных многосегментных и коммутированных корпоративных сетей, обеспечивая возможность централизованного управления всеми компонентами СОА;
- является открытой;
- имеет встроенные механизмы реагирования на атаки;
- является защищённой от атак на компоненты СОА, в том числе перехвата управления или атаки «отказ в обслуживании».

В результате сравнительного анализа доступных на сегодняшний день открытых систем было установлено, что их основными недостатками являются: отсутствие адаптивности и невозможность анализа данных поведения на всех трёх уровнях абстракции. Кроме того, наиболее эффективные по прочим критериям системы (Snort, прежде всего) плохо масштабируются, что не может быть решено простой доработкой.

Поэтому мы использовали предложенные критерии и результаты сравнительного анализа для обоснования архитектуры разработанной в данной работе системы.

4.1. Архитектура и алгоритмы работы системы обнаружения атак

На Рис. 5. представлена схема системы обнаружения атак с управляющими связями и связями по передаче сообщений.

В состав СОА входят следующие модули:

- сетевой сенсор;
- узловой сенсор;
- консоль управления;
- база данных;
- агенты реагирования.

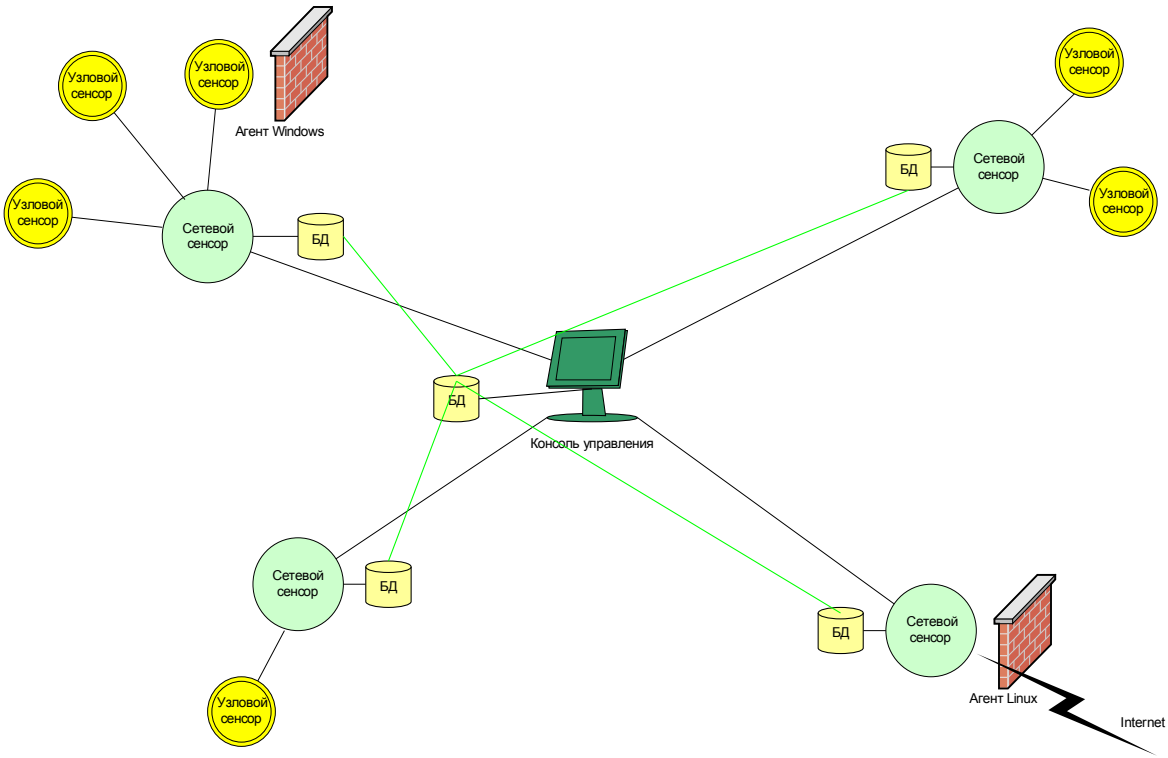

Рис. 5. Схема экспериментальной СОА.

Назначение и состав компонентов СОА:

Сетевой сенсор

Сетевой сенсор предназначен для анализа поведения сетевых объектов на основе данных сетевого трафика и сообщений от узловых сенсоров.

Сетевой сенсор состоит из наблюдателя, который формирует трассу событий на основе декомпозиции сетевого трафика, разбора информации различных протоколов, ядра анализа, которое получает поток событий от наблюдателя, подсистемы реагирования и служебной подсистемы, которая отвечает за управление и взаимодействие между компонентами СОА.

Ядро анализа представляет собой систему поддержки выполнения программ на языке СОА (СПВП) и набор автоматов первого и второго рода, описанных на данном языке. СПВП выполняет функции монитора анализирующих автоматов, формирует очередь выполнения тела переходов и состояний автоматов, планирует порядок выполнения автоматов, порождает и уничтожает экземпляры автоматов. Автоматы, по достижении конечного состояния, порождают атомарные сообщения об атаках в формате IDMEF.

Подсистема реагирования также состоит из наблюдателя, который получает сообщения об атаках и формирует трассу событий, ядра анализа (СПВП + автоматы) и служебной подсистемы. Автоматы реагирования формируют политику реагирования: в конечном состоянии каждого автомата выполняется заданная процедура реагирования. Это может быть разрыв соединения, настройка межсетевого экрана, корреляция сообщений об атаках и формирование более высокоуровневых сообщений.

Служебная подсистема сетевого сенсора отвечает за организацию шифрованного канала между компонентами СОА, сохранение сообщений об атаках и служебных сообщений в базе данных, реализацию функций удаленного управления и настройки сетевого сенсора с консоли управления.

Узловой сенсор

Узловой сенсор предназначен для анализа поведения сетевых объектов РИС на основе данных системных журналов и событий ОС: трассы действий приложений, пользователей, использования файловой системы и IPC контролируемой рабочей станции или сервера.

Узловой сенсор состоит из наблюдателя, который формирует трассу событий, ядра анализа, которое получает поток событий от наблюдателя, и служебной подсистемы, которая отвечает за управление и взаимодействие между компонентами СОА.

На выход узловой сенсор выдаёт сообщения об обнаруженных аномалиях или злоупотреблениях в формате IDMEF. Сообщения пересылаются сетевому сенсору, имеющему соединение с узловым сенсором.

База данных

База данных (БД) СОА представляет собой распределенное хранилище описаний нормального и аномального поведения объектов РИС, сообщений об атаках и журнала компонентов СОА. Данное хранилище используется сетевыми и узловыми сенсорами для централизованной загрузки автоматов в СПВП и хранения сообщений об атаках. База данных построена на основе открытой СУБД PostgreSQL.

Консоль управления

Консоль управления представляет собой графическое приложение управления СОА, в задачи которого входит:

- отображение физической и логической структуры СОА и защищаемой РИС;
- управление и настройка компонентов СОА;
- оповещение оператора о событиях безопасности в режиме рального времени (визуальные и звуковые эффекты);
- корреляция сообщений об атаках;
- централизованное реагирование;
- визуализация сообщений об атаках и журнала компонентов.

Агенты реагирования

Агенты реагирования СОА устанавливаются на узлы РИС, где установлены средства реагирования (межсетевые экраны), либо на контролируемые узлы РИС, и выполняют команды от подсистемы реагирования сетевого сенсора и консоли управления. В рамках экспериментальной СОА реализованы следующие агенты:

- агент IPTables для ОС Linux;
- агент для ОС Windows со встроенными возможностями пакетного фильтра и блокирования процессов на узле.

4.1.1. Структура и алгоритмы работы сетевого сенсора

В состав сетевого сенсора СОА входят следующие программные модули:

- модуль сбора сетевых данных;
- модуль анализа сетевых данных;
- лексический и синтаксический анализатор языка описания сценариев;
- система поддержки выполнения программ (СПВП);
- набор сценариев (автоматы первого и второго рода);
- модуль внешнего управления.

На рис. 6 представлена программная структура сетевого сенсора и связи между модулями, включая внешние.

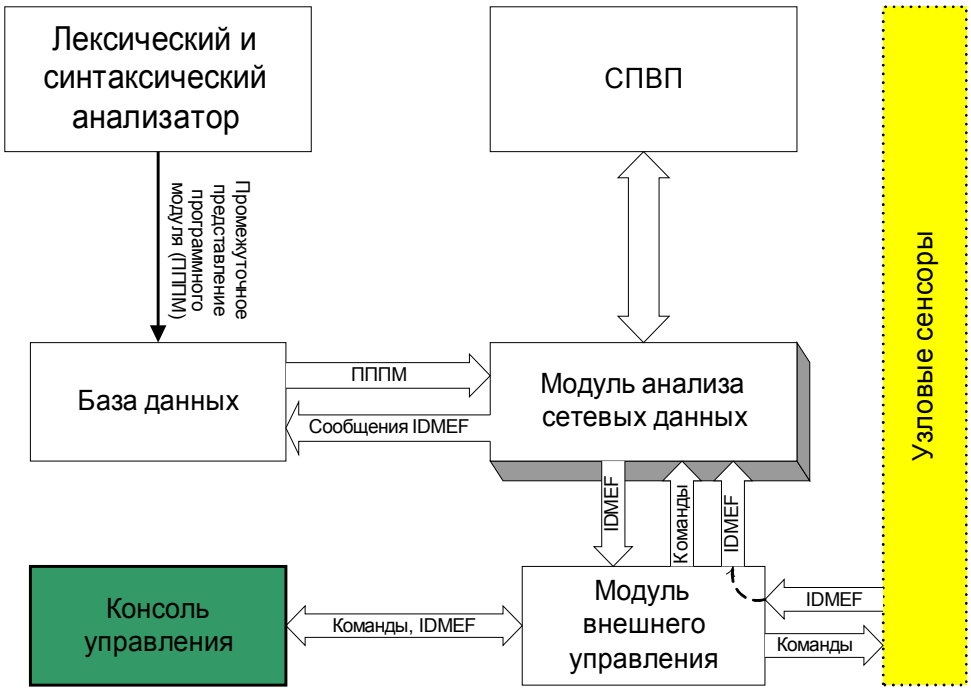

Рис. 6. Структура сетевого сенсора и связи с другими компонентами.

Модуль сбора сетевых данных

Модуль сбора сетевой информации осуществляет захват всех входящих и исходящих пакетов с сетевого интерфейса на уровне IP стека TCP/IP. Реализация модуля под ОС Linux выполняет функции захвата через механизм сокетов типа AF_PACKET.

Перед началом работы интерфейс (сетевой адаптер Ethernet) переводится в режим “promiscuous mode”, позволяющий просматривать все пакеты, видимые в данном сегменте сети, независимо от их получателя и отправителя.

Из всей реализации сетевого сенсора СОА текст программы для инициализации сетевого интерфейса наименее переносим. Однако, для ОС Linux платформенно-зависимая часть программы незначительна. Примерный код инициализации сетевого интерфейса следующий:

```
int        sock = socket ( AF_PACKET, SOCK_RAW, htons(ETH_P_ALL) );
if ( sock < 0 ) {
        perror ( “Cannot open raw socket” );
        exit(1);
}
struct ifreq ifr;
memset ( &ifr, 0, sizeof(ifr) );
strncpy ( ifr.ifr_name, ifName, sizeof(ifr.ifr_name) );

if ( ioctl ( sock, SIOCGIFINDEX, &ifr ) < 0 ) {
        perror ( “Cannot obtain interface id” );
        close ( sock );
        exit(1);
}

sockaddr_ll      saddr;
memset ( &saddr, 0, sizeof(saddr) );
saddr.sll_family = AF_PACKET;
saddr.sll_protocol = htons(ETH_P_ALL);
saddr.sll_ifindex = ifr.ifr_ifindex;
```

```
if ( bind ( sock, (struct sockaddr *)&saddr, sizeof(sockaddr_ll) ) < 0 ) {
        perror ( “Cannot bind to the interface” );
        close ( sock );
        exit(1);
}
struct packet_mreq      mreq;
memset ( &mreq, 0, sizeof(mreq) );
mreq.mr_ifindex = ifr.ifr_ifindex;
mreq.mr_type = PACKET_MR_PROMISC;
if ( setsockopt ( sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mreq, sizeof(mreq) ) <
0 )
                perror (  “Cannot change promiscuous mode” );
```

В целях повышения надежности системы сбора сетевой информации, то есть для избежания возможной потери пакетов при интенсивной загрузке процессора, применяется буферизация пакетов. После получения пакет либо передается модулю анализа сетевой информации, либо, если модуль анализа не готов принять пакет, буферизуется для дальнейшего анализа.

При передаче полученного пакета модулю анализа пакет анализируется на предмет соответствия его структуры пакетам известных протоколов стека TCP/IP; структура со списком полей передается в модуль анализа сетевых данных.

Модуль анализа сетевых данных

Модуль получает пакеты от модуля сбора сетевых данных и анализирует их с помощью сценариев, написанных на языке описания сценариев. Каждый сетевой пакет преобразуется в структуру типа NetEvent и передается соответствующему сценарию, либо множеству сценариев. Для выполнения сценариев используется система поддержки выполнения программ. Заключение об обнаружении атаки дается в результате выполнения сценария. В этом случае он вызывает функцию Alert(), которая реализована в библиотеке сетевого сенсора.

После получения пакета от модуля сбора сетевой информации производится первичная (оптимизационная) обработка данных с использованием хэш-таблиц от адресов, портов и идентификаторов протокола для определения предварительного списка сценариев, подходящих для обработки данного пакета. Для оптимизации просмотра сценариев модуль анализа сетевой информации просматривает синтаксическое дерево функций перехода для каждого из сценариев с целью получения эвристической информации о предикате перехода. В частности, если предикат является конъюнкцией нескольких логических выражений и некоторые из конъюнктов имеют один из видов (event – переменная типа NetEvent):

- IPMatch(event.src_ip,CONST1,CONST2)
- event.src_ip==CONST
- event.src_port == CONST
- IPMatch(event.dst_ip,CONST1,CONST2)
- event.dst_ip==CONST
- event.dst_port==CONST ,

то происходит предварительное отсечение сценариев, заведомо не действующих для данного события. Отсечение происходит с помощью двух хэш-таблиц для пары (src_ip,src_port) и пары (dst_ip,dst_port)..

Сообщения, пришедшие от узловых сенсоров, преобразуются в структуру NodeEvent и обрабатываются аналогичным образом.

Сценарии

Каждый сценарий из набора сценариев представляет собой реализацию автомата первого или второго рода в соотвествии с моделью. Подробная спецификация языка изложена в приложении 1. Здесь же приведём краткое описание сценария и переходов для понимания алгоритмов работы сенсоров СОА.

Определение сценария:

'scenario' <имя-сценария> '(' <список-аргументов> ')' '{'
[<декларация-переменной>|<декларация-состояния>|<декларация-перехода>]
'};'

Список аргументов в неявном виде задает класс событий, обрабатываемых сценарием. Типы структур, передаваемые в сценарий, сигнализируют о поддерживаемых сценарием событиях. Все аргументы, соответственно, должны быть структурами, унаследованными от структуры Event.

В теле сценария могут быть определены (как и в структуре) переменные и методы.

Каждому сценарию в процессе работы сопоставляется один или более контекстов выполнения. Контекст выполнения – это значения переменных, определенных внутри сценария плюс текущее состояние. Каждый контекст выполнения соответствует отдельному «экземпляру» данного сценария.

Декларация состояния:

['initial'] 'state' <имя-состояния> '{'
<логическое выражение>
<оператор>
'}'

Имя состояния является идентификатором, уникальным в пределах состояния. Логическое выражение определяет условие перехода в состояние (исключение: начальное состояние в момент создания копии сценария - в этом случае логическое выражение не вычисляется). Оператор выполняется при переходе в состояние.

Ключевое слово initial говорит о том, что состояние является начальным для сценария. В случае если ни для одного состояния не применено ключевое слово initial, начальным состоянием является первое состояние сценария.

Сценарий должен содержать хотя бы одно состояние.

Декларация перехода:

'transition' [<имя-перехода>] '(' <имя-состояния> '->' <имя-состояния> ')'
[‘consuming’|’nonconsuming’|’unwinding’] ‘{‘
<предикат>
<оператор>
'}'

Если предикат имеет значение «ложь» или не определён, дальнейших проверок не происходит. Осуществляется проверка предиката состояния. Если он истинен, выполняется оператор, осуществляется переход и выполняется оператор, определенный в теле состояния.

Различаются следующие типы переходов:

- consuming – не создается новой копии контекста выполнения (все выполняется в текущей копии);
- nonconsuming – создается новая копия контекста выполнения, переход осуществляется в ней;
- unwinding – удаляются все контексты выполнения, порожденные данным контекстом выполнения с помощью переходов типа nonconsuming.

Лексический и синтаксический анализатор

Лексический и синтаксический анализатор языка описания сценариев выполнен в виде отдельной программы. На вход подается блок текста (текст программной единицы), написанный на языке описания сценариев (ЯОС). Если данная программная единица содержит описания сценариев, то результатом выполнения является промежуточное представление кода сценариев. Программа может и не содержать сценариев, в этом случае она рассматривается как библиотечный модуль, содержащий функции (в том числе, возможно, и методы структур) для обеспечения корректной работы сценариев. Результат выполнения модуля с таким входным параметром – это промежуточное представление кода библиотеки.

Промежуточное представление программного модуля представляет собой набор следующих данных:

- список импортируемых функций; т.е. функций, внешних по отношению к данному программному модулю (ПМ). Для каждой функции указывается ее имя;
- список экспортируемых функций и методов; т.е. функций, определяемых внутри ПМ как доступных на вызов из других ПМ (глобальных функций). Формируется таблица имен функций;
- список сценариев, содержащихся в данном ПМ. Для каждого сценария дается список его состояний, список переходов, код для каждой из функций перехода, тип перехода;
- синтаксическое дерево модуля, переведенное в поток байт. Дается сокращенное дерево, его полнота достаточна для последующей компиляции в машинный код. Имена идентификаторов в таком дереве отсутствуют (исключение: идентификаторы, связывающие данный модуль с внешними модулями). На этапе конструирования дерева осуществляется первичная оптимизация программы (например, преобразование последовательности арифметических операций с константами в единую константу).

СПВП

Система поддержки выполнения программ обеспечивает подготовку набора сценариев к выполнению, а также взаимодействие сценариев между собой и с внешней средой. Она организована в виде библиотеки языка C++ и обеспечивает:

- компиляцию из временного представления в машинный код (для архитектуры IA32);
- сопоставление импортируемых и экспортируемых функций в нескольких ПМ;
- добавление внешних функций, написанных на C++ и доступных по вызову из ПМ;
- поддержку размещения в памяти контекста выполнения сценария;
- поддержку размещения в памяти контекста выполнения ПМ (глобальных переменных, определенных в ПМ);
- поддержку встроенных функций языка;
- конечный автомат для быстрого поиска множества строк или регулярных выражений (список строк и/или регулярных выражения должен быть заранее задан);
- поддержку перехода сценариев в различные состояния и, возможные в таких случаях, размножения и удаления контекста выполнения.

Сценарии или экспортируемые функции ПМ вызываются как обычные функции C++ (т.к. они откомпилированы в машинный код).

Внешние функции, написанные на языке программирования C++, также вызываются из сценариев способом, неотличимым от вызова из программы на C++. Их

можно использовать для расширения функциональности встроенной библиотеки языка представления сценариев.

Подготовка ПМ к выполнению состоит из нескольких этапов, на каждом из которых он либо переводится из одного представления в другое, либо производится итеративное преобразование его представления.

СПВП оперирует таблицей символов: итеративно наращиваемой в процессе обработки программных модулей ассоциативной таблицей - привязывающей имя объекта к его описанию или физическому адресу в памяти.

В начале работы таблица символов содержит только встроенные функции. Программный интерфейс СПВП предоставляет возможность ее обновления.

Конечный автомат для быстрого поиска подстрок и шаблонов в начале работы СПВП пуст. В процессе подготовки модулей к выполнению он расширяется для осуществления поиска подстрок и шаблонов, используемых данным модулем.

Этап 1 - подготовка: установление внешних связей с другими модулями.

Входным представлением первого этапа является сокращенное синтаксическое дерево, являющееся результатом работы модуля лексического и синтаксического анализа.

На данном этапе просматривается список внешних функций и методов, используемых в программном модуле, и осуществляется их поиск по имени в таблице символов. Если имя не найдено, фиксируется ошибка и работа с программным модулем прекращается. Найденные ассоциации для имен записываются в таблицу внешних имен программного модуля.

Далее осуществляется поиск неявных конструкторов для структур. Если они не найдены ни в таблице символов, ни в текущем модуле, они заносятся в таблицу символов со специальным флагом.

Этап 2 – примитивизация операций.

Этап является одним из подготовительных к этапу компиляции программы. Все выражения, содержащиеся в программе, разбиваются на примитивные операции, легко реализуемые в машинном коде.

Синтаксическое дерево программы на выходе данного этапа будет содержать не только типы операций, но и низкоуровневые инструкции для их исполнения. Возможно, при этом узлы синтаксического дерева будут разбиты на несколько более мелких.

На этом же этапе генерируются функции для работы с переходами и состояниями сценариев. Для каждого состояния, из которого возможен переход в другие состояния, генерируется функция-обработчик приходящих событий. Из результата синтаксического анализа выделяется оптимизационная информация о типе событий, на которые должен реагировать сценарий, и сохраняется в специальной таблице.

Этап 3 – выработка псевдокода.

Третий этап является основным в процессе подготовки программы к выполнению. Во время него синтаксическое дерево преобразуется в последовательность низкоуровневых команд с ветвлением на уровне if/goto.

Этап преобразует сложные структурные операторы if, for, while в форму, близкую к машинному коду. Логические выражения с детерминированным порядком вычисления (т.е. содержащие операции && или ||) также подлежат преобразованию в форму if/goto.

После завершения данного этапа синтаксическое дерево программы уже не участвует в работе алгоритмов и память, занимаемая им, может быть освобождена.

Имена идентификаторов, способных к экспорту во внешние модули, на данном этапе ассоциируются с соответствующими им низкоуровневыми командами; эта информация сохраняется в отдельной таблице.

Этап 4 – оптимизация псевдокода.

Выполняется итеративное преобразование последовательности примитивных операций и/или поиск типовых фрагментов для замены на наиболее оптимальные варианты. Ассоциации с экспортируемыми именами при необходимости обновляются.

На этом этапе также осуществляется специальная маркировка элементов последовательности с целью оптимизации выделения памяти и/или регистров процессора.

Этап 5 – компиляция в машинный код.

Псевдокод преобразуется в машинный код в два прохода.

На первом проходе определяются будущие адреса примитивных команд в оперативной памяти. Такой проход необходим, т.к. инструкции условного перехода в архитектуре IA32 имеют различный размер при различных дистанциях переходов.

На втором проходе генерируется машинный код. Если был пройден этап 4, используется маркировка, выработанная на этом этапе для распределения регистров; в противном случае регистры используются лишь для временного хранения промежуточных результатов примитивных операций.

В зависимости от операционной системы используются различные схемы аллокации временных регистров (т.к. различные операционные системы имеют различные соглашения о вызовах системных функций).

На данном этапе ассоциации экспортируемого имени с примитивным элементом заменяются ассоциациями имени с адресом в оперативной памяти.

Этап 6 – обновление таблицы символов.

На данном этапе просматривается таблица экспортируемых имен программного модуля: все обнаруженные имена и соответствующие им адреса в оперативной памяти добавляются в таблицу символов. Если в таблице символов уже присутствует символ с добавляемым именем, фиксируется ошибка и работа с программным модулем завершается (исключение: неявный конструктор структуры, помеченный как таковой в таблице символов).

В дальнейшем, другие модули могут использовать экспортируемые имена как внешние функции (методы, сценарии и т.п.).

Этап 7 – обновление конечного автомата быстрого поиска подстрок и шаблонов.

Чтобы обновить автомат поиска всех подстрок и шаблонов, требуется создать временный автомат, осуществляющий поиск конкретной строки или шаблона. Далее этот автомат подается на вход специализированному алгоритму, осуществляющему объединение двух конечных автоматов.

Строки и шаблоны для поиска берутся из выходных данных синтаксического анализатора языка. Для того, чтобы строка попала в таблицу для поиска, она должна присутствовать в вызове функций FastFind/FastMatch и быть константой.

Конечный автомат ищет в заданной ему строке все подстроки и шаблоны за один проход. Вычислительная сложность линейно зависит от длины строки, но практически не зависит от количества и структуры подстрок и шаблонов для поиска.

Этап 8 – создание неявных конструкторов для структур.

Этот этап выполняется непосредственно перед запуском любой из программ, то есть после обработки всех модулей. Если требуется создание одного или более неявных конструкторов для структур (их задачей является инициализация переменных типа string), на данном этапе создается фиктивный программный модуль, содержащий эти конструкторы, и компилируется.

Этап 9 – оптимизация конечного автомата поиска подстрок и шаблонов.

Этап 9, как и предыдущий, выполняется после обработки всех программных модулей. Он осуществляет оптимизацию размещения информационных структур, необходимых для работы конечного автомата, в оперативной памяти.

Таким образом, результатом подготовки программного модуля к выполнению являются:

- блок машинного кода, являющийся откомпилированным кодом текста программного модуля;
- обновленные таблица символов и конечный автомат поиска подстрок и шаблонов;
- таблица сценариев, которая содержит, в частности, размер контекста выполнения в байтах, список состояний, список адресов функций переходов. Для каждого состояния хранится адрес функции-обработчика события и служебная информация для оптимизации.

Модуль внешнего управления

В функции модуля внешнего управления входят аутентификация консоли управления, блокировка доступа к функциям управления сетевым анализатором и исполнение команд, поступающих от консоли управления.

Взаимодействовать с модулем по сети могут как и графическая, так и текстовая консоль управления. Переключение между режимами управления осуществляется средствами консоли управления или выбором управляющей программы локально на сетевом сенсоре.

4.1.2. Структура и алгоритмы работы узлового сенсора

Структура и алгоритмы работы узлового сенсора аналогичны структуре и алгоритмам работы сетевого сенсора с поправкой на драйверы сбора событий и иной способ формирования трассы событий для анализа.

В состав узлового сенсора СОА входят следующие программные модули:

- провайдеры узловых событий;
- модуль анализа событий;
- система поддержки выполнения программ (СПВП);
- набор сценариев (автоматы первого и второго рода);
- модуль внешнего управления.

На рис. 7 представлена программная структура сетевого сенсора и связи между модулями, включая внешние.

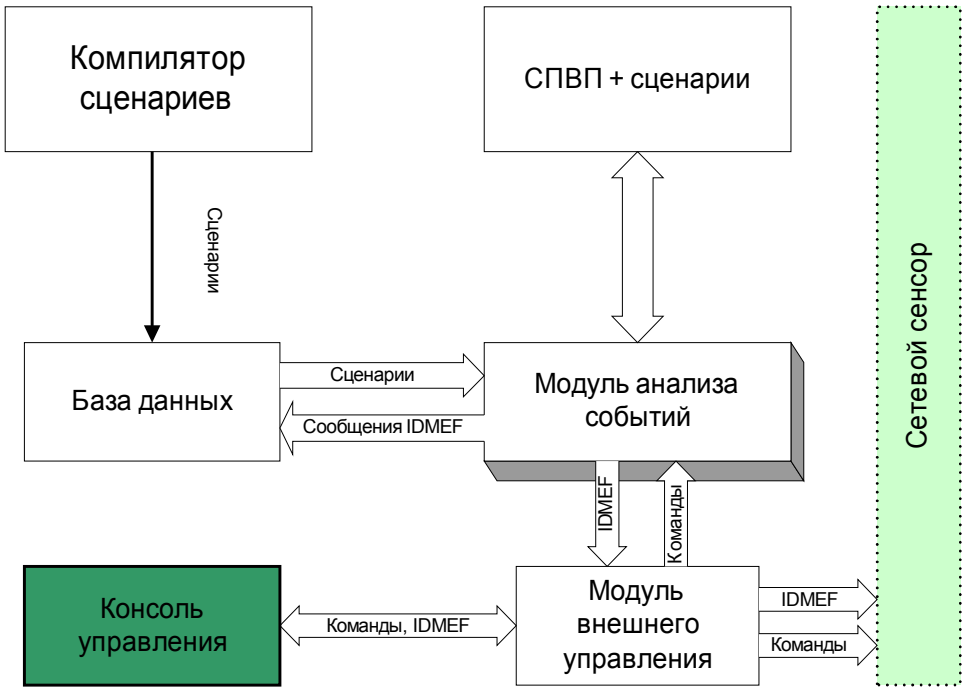

Рис. 7. Структура узлового сенсора и связи с другими компонентами.

Провайдеры узловых событий

В состав узлового сенсора экспериментальной СОА входят следующие провайдеры событий:

- подсистема сбора сетевых данных;
- драйвер узловых событий;
- драйвер событий аппаратуры.

Подсистема сбора сетевых данных организована для ОС Linux аналогично подсистеме сетевого сенсора. Для ОС Windows используется драйвер перехвата сетевых данных.

Драйвер узловых событий позволяет отслеживать следующие классы событий:

1. Файловая система (создание, открытие, закрытие, изменение файлов и т.д.);
2. Процессы (запуск и завершение процессов);
3. Реестр (только для ОС Windows: создание, удаление и изменение ключей реестра);
4. Сокеты (создание сокетов, прослушивание, чтение и отправка данных через сокет);
5. Стандартные журналы (только для ОС Windows: события системы и приложений).

Модуль анализа событий и прочие модули узлового сенсора реализованы аналогично сетевому сенсору.

4.1.3. Структура и алгоритмы работы подсистемы реагирования

Подсистема реагирования является иерархической и состоит из компонентов трёх типов: узловых менеджеров, центральных менеджеров и агентов. Узловой менеджер является подсистемой реагирования сетевого сенсора. Центральный менеджер реагирования является подсистемой реагирования консоли управления.

Агент реагирования установлен на каждом подключенном средстве реагирования и выполняет функции управления и настройки данного средства.

Каждый менеджер реагирования состоит из следующих модулей:

- провайдер событий;
- система анализа событий;
- модуль внешнего управления.

Структура менеджера реагирования представлена на рис. 8.

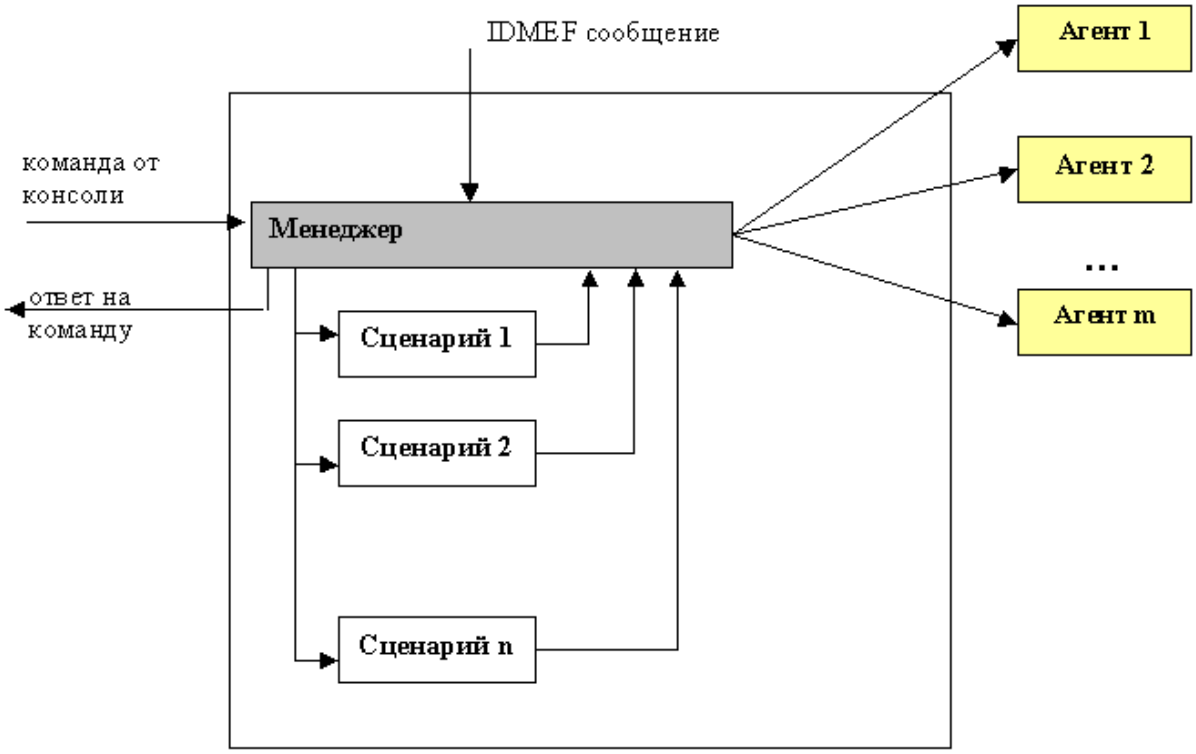

Рис. 8. Структура менеджера реагирования

Провайдер событий

От сетевых сенсоров менеджер получает сообщения в формате IDMEF. По полученному сообщению провайдер событий формирует событие и передаёт его системе анализа событий для анализа и формирования реакции.

Система анализа событий

Данный модуль состоит из СПВП и набора сценариев реагирования, реализованных на языке описания сценариев СОА. Как и в случае с подсистемой реагирования, алфавитом сценариев являются сообщения об атаках, а в конечном состоянии каждого сценария выполняется соответствующая ему функция реагирования.

Пример сценария реагирования:

```
scenario HIGHSeverityResponse ( NetworkAlertEvent netEv)
{
   initial state state0{}
   consuming transition state0->state0
            event NetworkAlertEvent (1)
            {
                        if (netEv.severity == NET_ATTACK_HIGH)
                        {
                                    sendFWRule(netEv.targetAddress, “REJECT –dir in –s “
+ Iptoa(netEv.sourceAddress) + “ –reject-type icmp-host-unreachable”);
                        }
```

```
        }
};
```

В данном примере в ответ на каждое сообщение об атаке с уровнем опасности high (высший по стандарту IDMEF) формируется блокирующее правило для межсетевого экрана IPTables и отправляется соответствующему агенту реагирования.

Модуль внешнего управления

Данный модуль отвечает за выполнение команд консоли управления, а также за регистрацию и проверку доступности агентов реагирования.

4.1.4. Структура и алгоритмы работы консоли управления

В данном разделе приведено описание графической консоли управления экспериментальной СОА. Консоль управления является инструментом администратора системы обнаружения атак.

Консоль управления решает следующие задачи:

- отображение физической и логической структуры СОА и защищаемой РИС;
- управление и настройка компонентов СОА;
- оповещение оператора о событиях безопасности в режиме реального времени (визуальные и звуковые эффекты);
- корреляция сообщений об атаках;
- централизованное реагирование;
- визуализация сообщений об атаках и журнала компонентов.

Графический интерфейс консоли управления состоит из нескольких окон. Большинство функций консоли управления доступны через следующие окна:

- основное окно;
- окно журнала;
- окно настройки компонента сети;
- окно настройки и управления компонентом СОА;
- окно настройки консоли управления.

При запуске консоли управления запускается основное окно. Основное окно всегда содержит меню, панель инструментов и строку состояния.

В зависимости от режимов работы и настроек в основном окне дополнительно могут быть:

- дерево доступных объектов сети;
- дерево доступных компонентов СОА;
- индикатор текущего объекта сети;
- индикатор текущего компонента СОА;
- редактор схем;
- текстовые консоли управления.

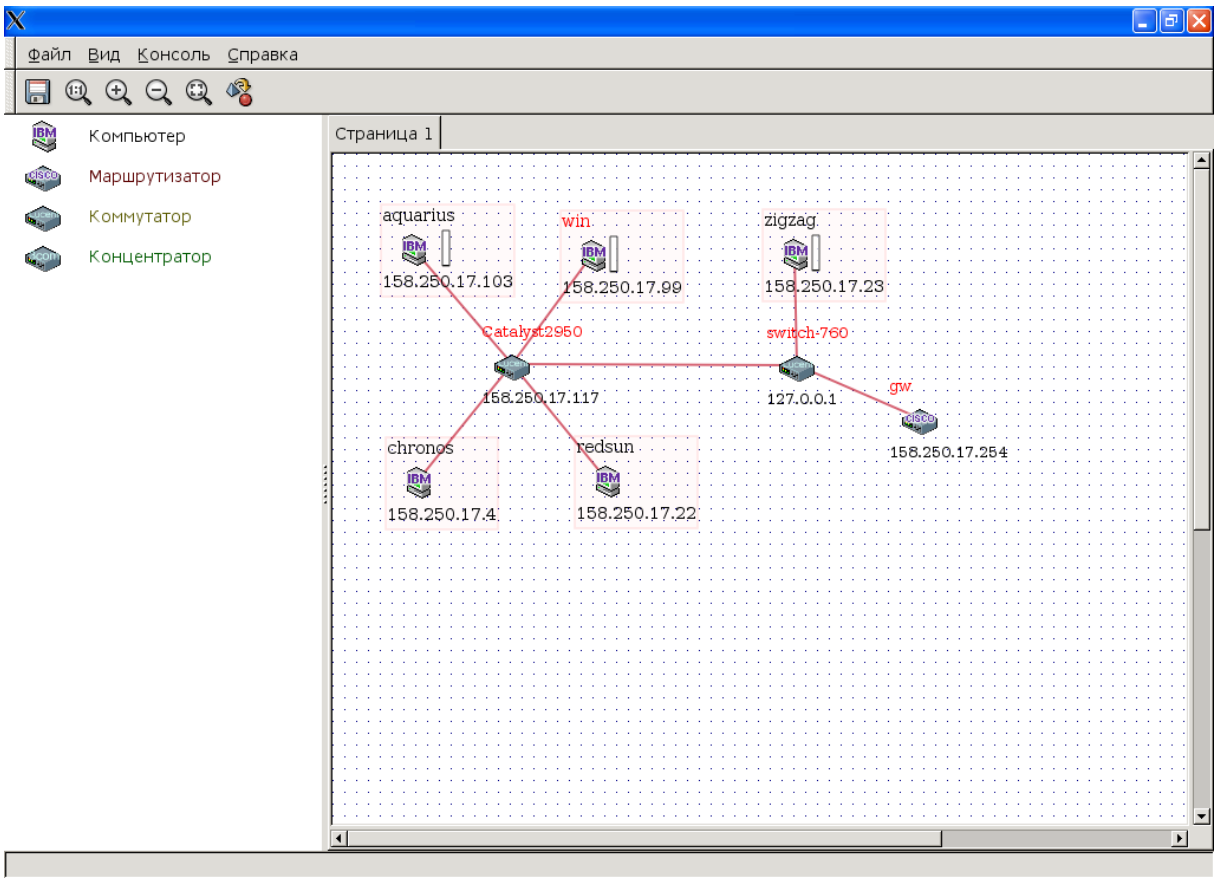

Рис. 9. Общий вид консоли управления

Основными компонентами главного окна можно назвать *редактор схем* и *текстовую консоль управления*.

Редактор схем позволяет редактировать и визуализировать структуру СОА и сети, в которой она установлена, а так же управлять компонентами СОА. Редактор визуализирует критические для безопасности события, происходящие в сети. Все это редактор делает при помощи *графического представления структуры СОА*.

Графическое представление структуры СОА

Графическое представление структуры СОА – схема, на которой изображены объекты сети и СОА и связи между ними. Есть два режима графического представления: режим физических соединений и режим логических соединений.

В *режиме физических соединений* сеть представляется графом с объектами сети в вершинах. Ребра этого графа соответствуют физическим соединениям компонентов (в большинстве случаев, проводам). Объектом сети может быть узел или какое-то сетевое устройство. Например, узел «корень иерархического хранилища данных». Каждый объект сети принадлежит одному из предопределенных типов. Возможные типы объекта сети: узел, концентратор, маршрутизатор, коммутатор.

С объектами сети типа «узел» могут быть связаны компоненты системы обнаружения атак. У каждого компонента сети есть «уточняющий тип». Пример уточняющего типа объекта сети: «Рабочее место администратора».

У объекта сети есть два изображения: краткое и полное. Краткое изображение служит для графического обозначения класса объекта в дереве компонентов и индикаторе объекта сети и состоит из маленькой иконки. Полное изображение представляет собой изображение объекта на схеме сети и состоит из большой иконки и дополнительных деталей. Дополнительные детали – это подписи и индикаторы. Подписи отображают значения параметров объекта и его компонентов.

Большую иконку объекта сети можно изменить для каждого конкретного объекта сети. Индикаторы используются для визуализации работы СОА. Каждому компоненту может соответствовать несколько индикаторов. Индикаторы отображают состояние компонента. Индикатор изображается как «светодиод».

С точки зрения консоли управления, объект сети – это множество компонентов, множество параметров, иконка для изображения на схеме сети, иконка для изображения на дереве объектов, тип, уточняющий тип, описание возможных соединений.

В *режиме логических связей* графическое представление содержит граф логических связей компонентов СОА. По аналогии с объектом в режиме физических соединений, в режиме логических соединений компонент СОА является базовым элементом графического представления.

У компонента СОА есть тип. Возможные типы компонентов:

- Сетевой сенсор;
- Узловой сенсор;
- Хранилище данных;
- Менеджер реагирования;
- Агент реагирования;
- Консоль управления.

Компонент изображается иконкой, возможно окруженной подписями со значениями параметров.

Сеть изображается практически так же, как в режиме физических соединений. Однако в этом режиме также изображаются компоненты объектов. Компоненты объектов изображаются поверх объектов их содержащих. Ребра физических соединений изображаются другим цветом, чтобы их не путали с логическими соединениями. В режиме логических соединений можно только добавлять и удалять компоненты из узлов и соединять их между собой.

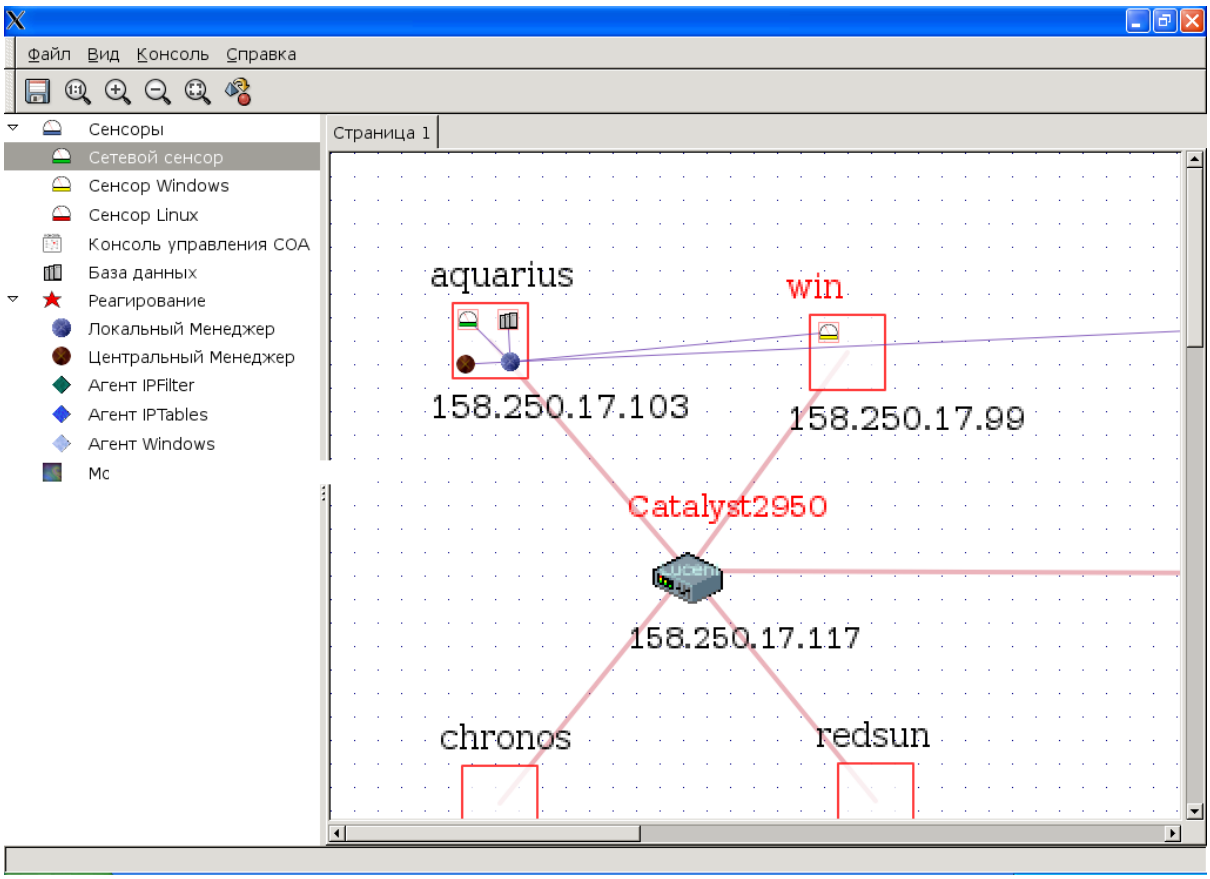

Рис. 10 Режим логических соединений

Редактор схем

Графическое представление РИС можно редактировать и просматривать при помощи редактора схем. Самая простая функция редактора схем – поставить на схему новый объект сети. Другая функция – соединить два объекта ребром. Соединить две вершины можно с помощью правой кнопки мыши.

Редактор схем используется для настройки и управления компонентами СОА. Если в контекстном меню выбрать пункт «Компонент ...», то появится окно настройки и управления компонентом СОА. Из этого окна, нажимая на соответствующие кнопки, можно запускать и останавливать компонент, можно менять его настройки и режимы работы.

Текстовая консоль

Другим средством управления СОА является текстовая консоль. Текстовая консоль позволяет в командном режиме настраивать и управлять компонентами системы.

Текстовая консоль может быть подключена к какому-либо компоненту системы. В течение сеанса связи все команды текстовой консоли направляются компоненту.

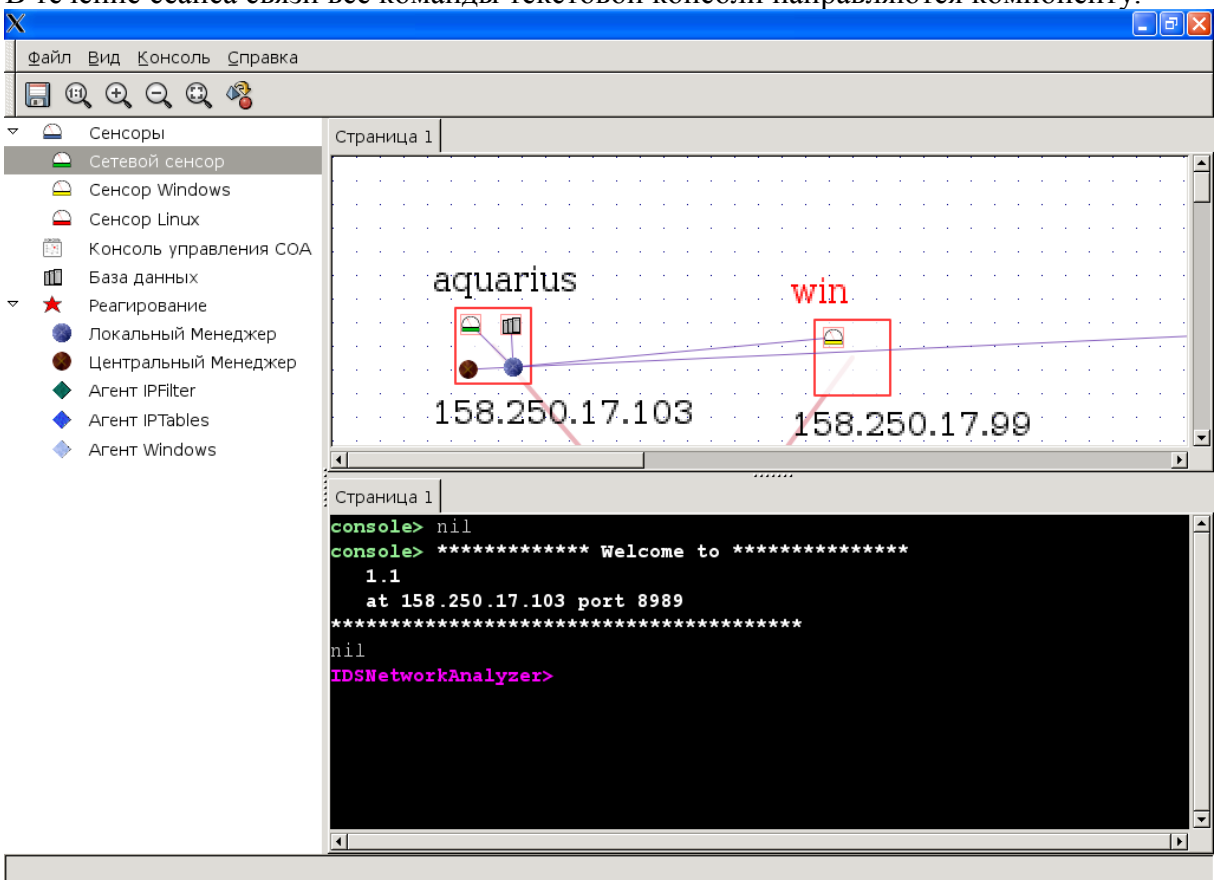

Рис. 11. Текстовая консоль.

Окно журнала

Информация о событиях, происходящих в системе обнаружения атак, о зарегистрированных аномалиях в исследуемой компьютерной сети и о событиях самой консоли управления сохраняются в журнале. Кроме того, в журнале можно настроить функцию оповещения администратора по электронной почте о критичных событиях. Список критичных событий и адреса электронной почты администратора можно настроить в окне настройки консоли управления.

В окне журнала можно просматривать информацию о произошедших событиях. Набор показываемых сообщений можно ограничить, задав характеризующие признаки.

Такой сокращенный журнал может быть сохранен в файл. После создания такого файла администратор увидит окно с сообщением, подтверждающим создание файла.

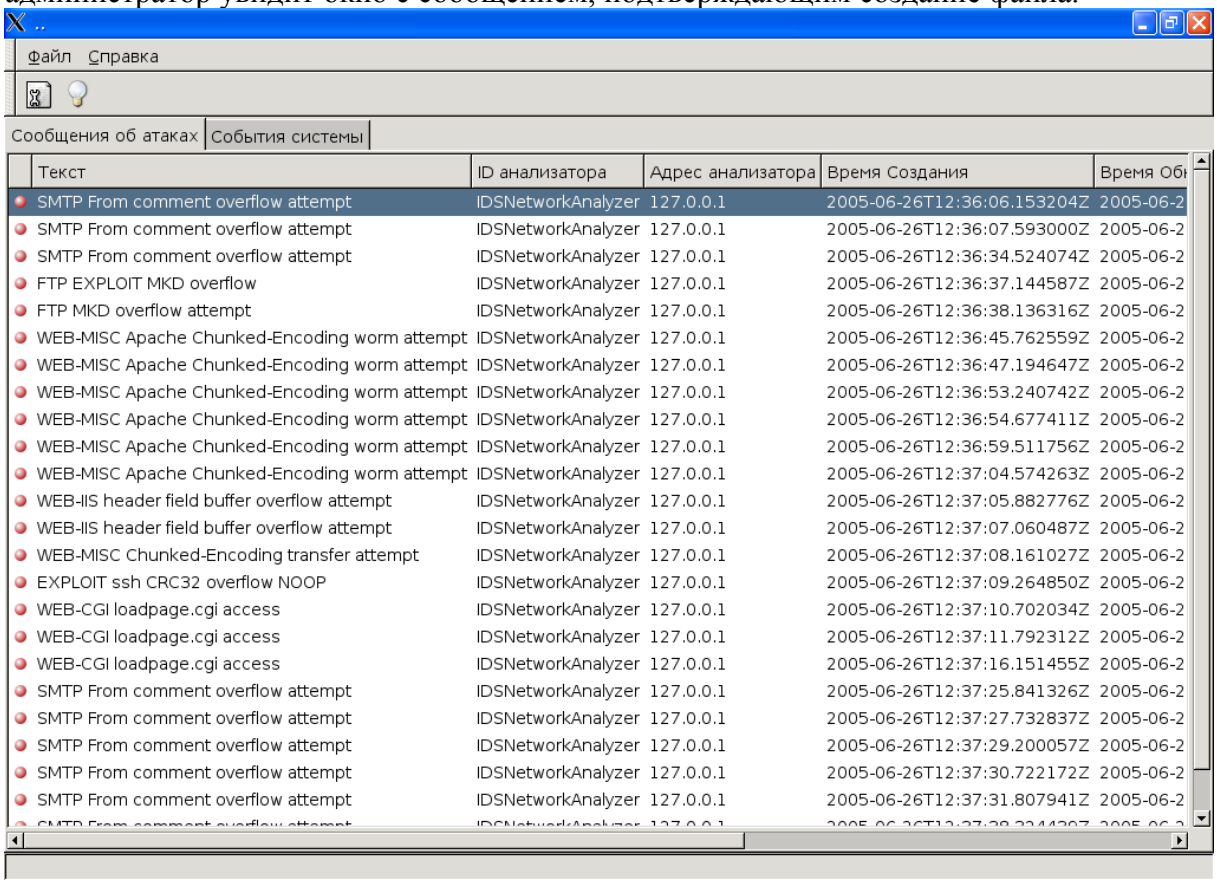

Рис. 12. Окно журнала консоли управления.

Язык текстовой консоли

В текстовой консоли языка управления используется алгоритмический язык Ruby. Текстовая консоль представляет собой интерпретатор этого языка. Для удобства работы администратора, все команды управления реализованы как функции языка Ruby, что упрощает синтаксис команд.

4.1.5. Организация иерархического хранилища данных

База данных экспериментальной СОА представляет собой двухзвенную иерархическую структуру. Каждый узел БД может быть либо листовым, либо корневым, при этом допускается только один корневой узел. Выполняется следующее свойство: все записи листовых узлов БД содержатся в корневом узле во все моменты времени, задаваемые при настройке БД. Таким образом, иерархия узлов хранилища обеспечивает двукратную избыточность информации для всех вершин, кроме корневой вершины.

БД хранит два типа записей: сценарии описания атак для сенсоров и сообщения от компонентов системы.

Записи сценариев имеют следующие поля:

- name – имя (например, название файла сценария);
- IP – ip адрес узла, сенсор которого работает с данным сценарием;
- data – двоичный код сценария, готовый к загрузке в сенсор.

Ключом записи является пара (name, IP).

Записи сообщений от компонентов имеют следующие поля:

- time – время наступления события;

- component_id – имя компонента, от которого пришло событие;
- IP – ip адрес, от которого пришло событие;
- event_type - тип наступившего события (сообщение об атаке, либо сообщение о работе компонента);
- severity – важность события;
- xml – описание события в xml-формате.

Ключом записи является тройка (time, component_id, IP).

Возможные запросы для таблиц сценариев имеют следующий вид:

- вставить сценарий (name, IP, data);
- удалить все сценарии с данным именем name;
- удалить все сценарии для данного узла IP;
- удалить конкретный сценарий (name, IP);
- получить все сценарии с данным именем name;
- получить все сценарии для данного узла IP;
- получить конкретный сценарий (name, IP).

Возможные запросы для таблиц сообщений от компонентов имеют следующий вид:

- вставить сообщение (time, component_id, IP, event_type, severity, xml);
- удалить все сообщения от данного компонента component_id;
- удалить все сообщения с данного узла IP;
- удалить все сообщения, наступившие до time;
- получить все сообщения от данного компонента component_id;
- получить все сообщения с данного узла IP;
- получить все сообщения, наступившие до time;
- получить все сообщения, наступившие в период (time1, time2).

В качестве СУБД используется PostgreSQL версии 7.4 и выше.

4.2. Выводы по архитектуре

На основе приведенного описания архитектуры и алгоритмов работы экспериментальной СОА можно провести её сравнение с рассмотренными в обзоре открытыми СОА по заданным критериям.

Класс обнаруживаемых атак

Экспериментальная СОА, как системы Prelude и NetSTAT, является гибридной, т.е. способна обнаруживать атаки на уровне системы, сети и приложений.

СОА обнаруживает атаки следующих классов: (L,R,A,D), где:

- L={***li*** ∪ ***le***} (внутренние и внешние атаки);
- R={***rl*** ∪ ***rn***} ∩ {***ru*** ∪ ***rs*** ∪ ***rp***} (атаки на локальные или сетевые пользовательские ресурсы, системные ресурсы и ресурсы защиты);
- A={***as*** ∪ ***au*** ∪ ***ar*** ∪ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={***dn***∪ ***au*** } (нераспределенные и распределенные).

По данному критерию возможности экспериментальной системы шире, чем у рассмотренных в обзоре систем.

Уровень наблюдения за системой

По данному критерию экспериментальная СОА наблюдает за поведением сетевых объектов на всех уровнях: сетевом, системном и прикладном - что являлось одной из целей разработки данной системы.

Адаптивность

Экспериментальная СОА использует гибридный метод обнаружения атак, основанный на использовании метода анализа переходов состояний как для обнаружения реализаций атак, так и для обнаружения аномалий поведения объектов РИС. Данный метод обладает свойством адаптивности, что позволяет использовать систему для обнаружения модифицированных и неизвестных атак на защищаемые объекты РИС. При этом вычислительная сложность метода находится в линейной зависимости от длины трассы анализируемых событий: состояний объектов РИС и их действий. В ней так же сохраняются свойства верифицируемости и устойчивости, присущие сигнатурным методам. Таким образом, достигнута одна из основных целей данной работы, которая и заключалась в создании адаптивного метода обнаружения атак с вычислительной сложностью, сравнимой со сложностью сигнатурных методов. Это позволяет использовать построенную систему в высокоскоростных системах, использующих каналы передачи данных с высокой пропускной способностью.

Масштабируемость

Разработанная СОА является линейно масштабируемой: добавление нового анализируемого узла или канала передачи данных требует включения в конфигурацию системы одного соответствующего компонента – сетевого или узлового сенсора. Масштабирование на уровне приложений осуществляется добавлением сценариев, реализующих автоматы первого и второго рода для соответствующих приложений в рамках узлового сенсора.

Экспериментальная СОА является открытой системой, имеет открытый интерфейс расширения набора сценариев и API системы прогона, который позволяет добавлять новые типы событий стандартным образом. Обмен сообщениями происходит по стандарту IDMEF (Intrusion Detection Message Exchange Format), что позволяет встраивать компоненты системы или систему целиком в сторонние системы защиты, а также интегрировать модули сторонних разработчиков, поддерживающие данный формат.

По данному критерию экспериментальная СОА соответствует возможностям системы Prelude.

Реагирование

В состав экспериментальной СОА входит распределенная система реагирования с возможностью задания политики реагирования в виде набора сценариев. Открытый интерфейс агентов реагирования, набор реализованных агентов реагирования для распространённых межсетевых экранов. Кроме того, модули реагирования системы имеют встроенную систему прогона сценариев и позволяют анализировать сообщения об атаках в качестве обычных событий сценариев СОА, что, в свою очередь, позволяет выполнять в данных модулях дополнительный сложный анализ событий. Например, устанавливать взаимосвязь между событиями и атаками на разных уровнях: сетевом, узловом и уровне приложений. Такой возможностью не обладает ни одна из систем, рассмотренных в разделе 2.

Защищенность

Экспериментальная СОА использует шифрованные каналы передачи данных и управления на основе модифицированного протокола SSHv2, интегрированного с инфраструктурой публичных ключей (PKI) по стандарту X.509.

Таким образом, разработанная система соответствует оптимальным параметрам по всем критериям сравнительного анализа, включая целевую адаптивность к неизвестным и модифицированным атакам, а также низкую вычислительную сложность метода обнаружения атак.

5. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ЭКСПЕРИМЕНТАЛЬНОЙ СОА

В данном разделе приведены результаты испытаний экспериментальной СОА на инструментальном стенде с использованием тестовых наборов компьютерных атак. В начале данного раздела приводится описание инструментального стенда и методики испытаний, в заключении приводятся результаты испытаний.

Так как цель тестирования – проверка эффективности обнаружения атак с помощью гибридного метода анализа переходов состояний, то для тестирования выбран ограниченный набор атакуемых ресурсов и реализаций атак. В качестве атакуемых ресурсов были использованы несколько версий сервера FTP ProFTPd, веб-сервер Apache и сервер SSH openssh.

Автомат описания нормального поведения был построен для семейства FTP-серверов ProFTPd. Описание данного автомата приведено ниже в описании тестовых сценариев обнаружения. Для остальных ресурсов были построены описания атак для конкретных реализаций атак из тестового набора. Их описание также см. ниже.

5.1. Набор тестовых примеров

В набор тестовых примеров включены реализации атак из открытых источников, которые были выбраны из набора примерно в 30 различных реализаций по принципу работоспособности «из коробки» - данные атаки успешно выполнялись для целевого сервиса без дополнительной настройки или доработки. Для реализации атаки класса «сканирование» был использован распространённый сканер безопасности nmap. Ниже приведен список использованных в испытаниях реализаций атак:

1. Атака на уязвимость переполнения буфера в FTP-сервере ProFTPd версий proftpd 1.2.7 и 1.2.8 с двумя разными наборами вредоносных команд (shellcode): proftp_put_down.c, proftp_put_down2.c;
2. Атака на уязвимость переполнения буфера в FTP-сервере ProFTPd версий proftpd 1.2.0pre3, propro, babcia;
3. Атака на web-сервер Apache версий 1.3.x, использующая уязвимость в обработке кодированных цепочками (chunk-encoded) HTTP-запросов;
4. Атака blackjack.c, позволяющая получить привилегированный доступ (remote root) через SSH-сервер ОС RedHat Linux 8.0 с настройками по-умолчанию;
5. Атака на Microsoft Windows DCOM, позволяющая получить удалённый cmd-сеанс с правами администратора на ОС Windows XP без SP2;
6. Атака Apache squ1rt на web-сервер Apache версий 2.0.x, использующая утечку памяти для переполнения памяти сервера и реализации атаки «отказ в обслуживании»;
7. Сканирование привилегированных портов с помощью утилиты nmap.

5.2. Тестовые сценарии обнаружения

В данном разделе описан набор тестовых сценариев для исследования эффективности экспериментальной СОА. Наибольший интерес представляет сценарий, реализующий автомат второго рода для FTP-сервера ProFTPd, т.к. он является примером использования предложенного метода для обнаружения неизвестных атак.

Алфавитом сценария является объединение множества команд FTP в соответствии с RFC 959 и множества системных вызовов ядра ОС Linux версии 2.4. Из множества всех системных вызовов в целях оптимизации рассматриваются лишь четыре, которые могут быть потенциально опасны в контексте выполнения proftpd: *fork()*, *vfork()*, *execve()* и *socketcall()*.

На основе анализа исходных текстов и трасс системных вызовов сервера proftpd был сделан вывод, что данные системные вызовы в нём используются в следующих ситуациях:

- после создания управляющего канала используется *fork*, создаётся поток для работы с этим каналом;
- после выполнения пользователем команды PASV для передачи данных в пассивном режиме сервер создаёт слушающий сокет, используя системный вызов *socketcall* с параметром SYS_LISTEN;
- также есть 6 команд (STOR, STOU, RETR, LIST, NLST, APPE), после которых, согласно протоколу, устанавливается канал данных. При этом используются системные вызовы socketcall с параметром SYS_CONNECT.

На основе спецификации протокола FTP и полученной информации о реализации строится сценарий нормального поведения сервера proftpd.

Напомним, что автомат второго рода представляет собой структуру следующего вида:

$$(S, P_S, T, P_T, s_0, I, g, q)\text{, где}$$

S – множество состояний;
P_S – множество предикатов состояний;
T – функция переходов;
P_T – множество предикатов переходов;
s_0 – начальное состояние;
I – множество экземпляров автомата;
g – глобальное окружение;
q – глобальная очередь таймера.

I, g и q отражают контекст выполнения сценария в заданном окружении в каждый момент времени.

Для нашей модели сервера proftpd множество состояний S = {statepre, state0, login, welcome, waitPASS, sendPASS, sendUSER, sendSITEHELP, wait221, wait214SITE, wait150, sendPORT, sendPASV, waitans, waitanslog}, начальное состояние s0=statepre.

Входной алфавит – множество команд FTP, ответы сервера FTP, состояния TCP-соединения и перечисленный выше набор системных вызовов: {"CWD", "EPRT", "XRMD", "APPE", "XCWD", "EPSV", "MKD", "REST", "CDUP", "ALLO", "XMKD", "TYPE", "ABOR", "XCUP", "RNFR", "PWD", "STRU", "SMNT", "RNTO", "XPWD", "MODE", "DELE", "SIZE", "RETR", "PORT", "MDTM", "STOR", "PASV", "RMD", "STOU", "LIST", "SITE", "NOOP", "SYST", "OPTS", "SIZE", "MDTM", "SYST", "PWD", "TYPE", "CDUP", "FEAT", "OPTS", "CWD", "REST", "MODE", "PASS", "USER", "MKD", "RMD", TCP_CONNECT, TCP_RST, TCP_FIN, TCP_ESTABLISHED, SYS_FORK, SYS_CONNECT, SYS_LISTEN, SYS_EXEC, SYS_VFORK}.

На рис. 13 схематически представлена статическая диаграмма состояний данного сценария.

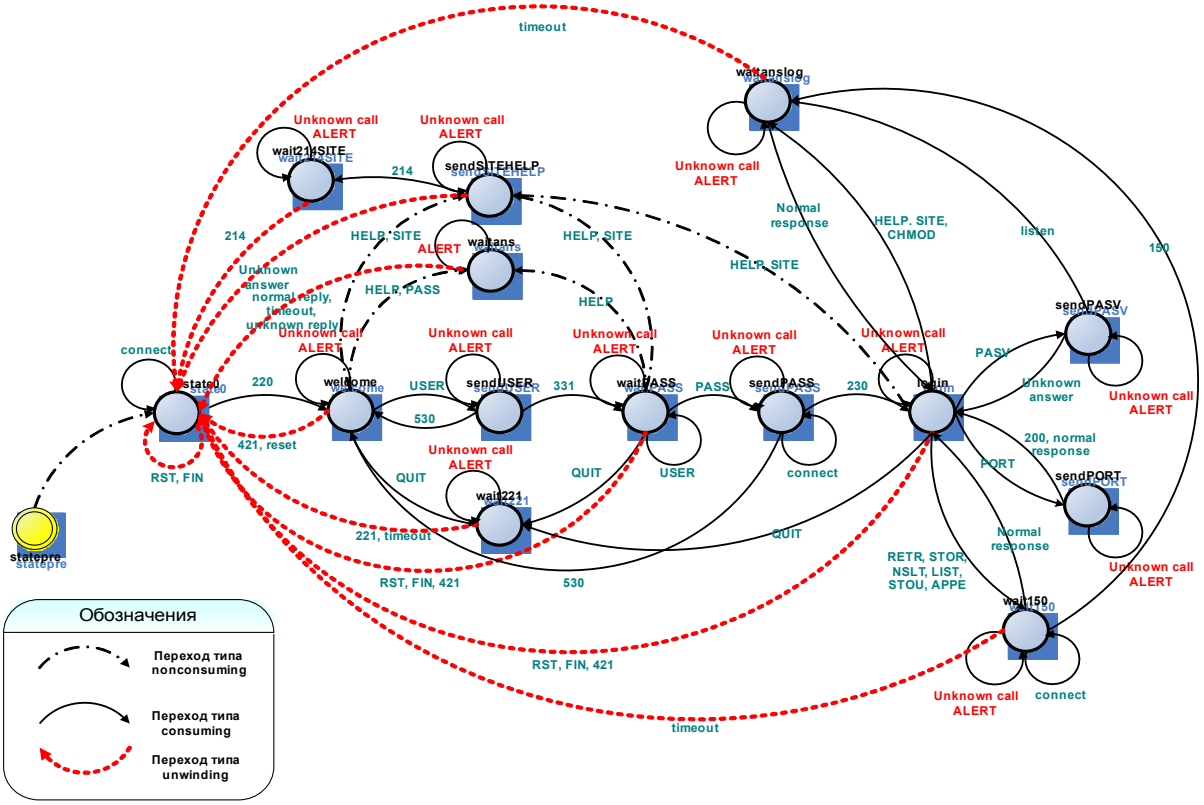

Рис. 13. Диаграмма состояний автомата второго рода для ProFTPd.

Сценарий нормального поведения ProFTPd в ходе испытаний функционировал в составе сетевого сенсора экспериментальной СОА, установленного на один узел с защищаемым сервисом.

Кроме данного сценария были построены сценарии, реализующие автоматы первого рода для каждой из используемых атак. Данные сценарии функционировали в составе сетевых и узловых сенсоров на всех узлах испытательного стенда.

В следующем разделе приведено описание стенда и схема размещения компонентов экспериментальной СОА на его узлах.

5.3. Состав и структура инструментального стенда

Состав стенда (см. рис. 13):

- коммутатор Cisco Catalyst 2950 – 24x100Mbit;
- sensor1, sensor2 – сетевой сенсор:
  Процессор: Pentium D 3000 Мгц (двухъядерный);
  Дисковое пространство: 80Гб HDD;
  Оперативная память: 2Гб;
  Операционная система: Debian GNU/Linux 3.1 Sarge.
- win1, win2 – клиентская рабочая станция:
  Процессор: Pentium III 800 Мгц (и выше);
  Дисковое пространство: 50Мб;
  Оперативная память: 256 Мб;
  Операционная система: Windows 2000.
- www, ftp – серверы:
  Процессор: Pentium IV 2400 Мгц;
  Дисковое пространство: 50Гб;

Оперативная память: 1000 Мб;
Операционная система: Debian GNULinux 3.1 Sarge и Stackware Linux.

- attacker1, attacker2 – рабочие станции нарушителей:
  Процессор: Pentium III 800 Мгц;
  Дисковое пространство: 50Мб;
  Оперативная память: 256 Мб;
  Операционная система: Debian GNULinux 3.1 Sarge.
- console – консоль управления:
  Процессор: Pentium IV 2800 Мгц;
  Дисковое пространство: 200Гб;
  Оперативная память: 1000 Мб;
  Операционная система: ОС Debian GNU/Linux 3.1 Sarge.

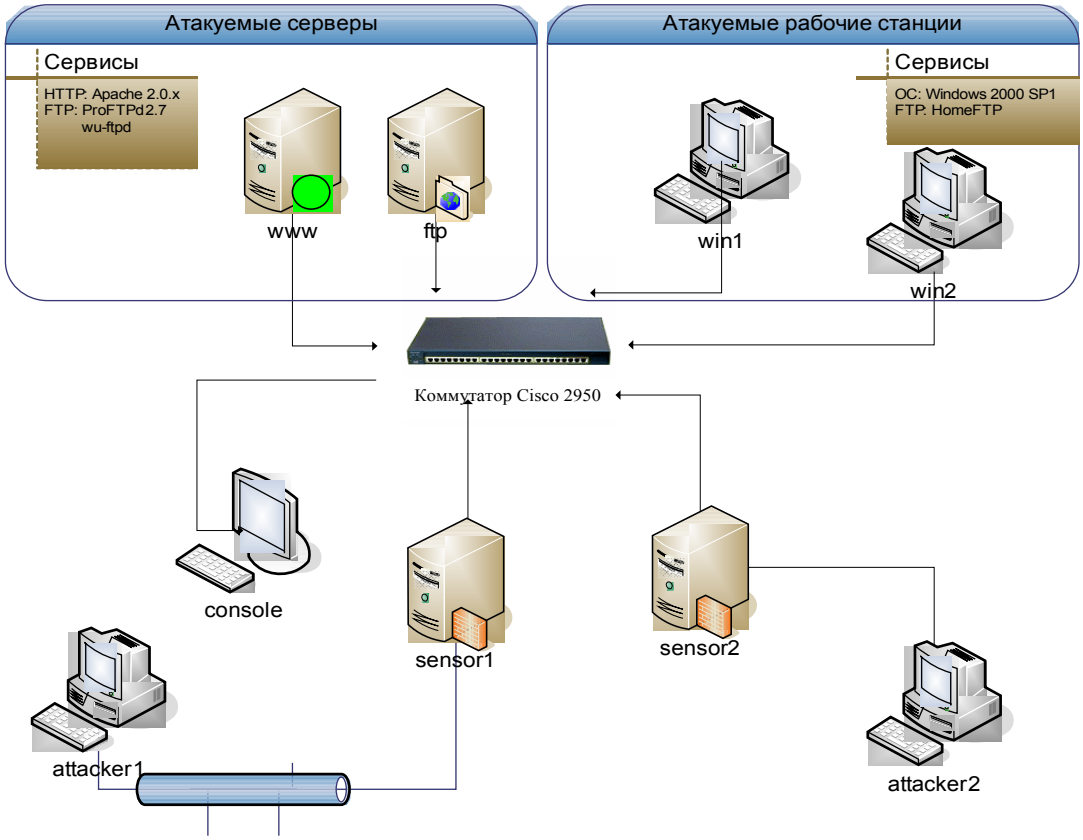

Рис. 14. Структура инструментального стенда.

Размещение компонентов экспериментальной СОА на узлах стенда:

- узел *console*: консоль управления СОА и база данных;
- узел *sensor1*: сетевой сенсор и база данных;
- узел *sensor2*: сетевой сенсор и база данных;
- узел *www*: сетевой сенсор и база данных;
- узел *ftp*: сетевой сенсор и база данных;
- узел *win1*: узловой сенсор;
- узел *win2*: узловой сенсор.

Подсистема реагирования на сетевых сенсорах и консоли управления настроена на автоматическую блокировку IP-адреса нарушителя на 10 минут.

Архитектура стенда, состав программных ресурсов защищаемых узлов стенда и набор тестовых примеров на атакующих узлах стенда выбраны исходя из необходимости обеспечить тестовое покрытие примерами компьютерных атак для

каждого из рассматриваемых классов: внешние и внутренние атаки, атаки на пользовательские, системные и сетевые ресурсы, распределенные и нераспределенные атаки.

5.3.1. Сетевая инфраструктура

При построении стенда использован коммутатор Cisco Catalyst 2950. Пропускная способность каждого канала составляет 100Мбит/с.

5.3.2. Серверные узлы

Для имитации служебной части защищаемой сети использованы серверы HTTP (www) и FTP (ftp), подключенные к коммутатору с помощью 100-мегабитного сетевого интерфейса.

На серверах установлены различные версии ОС Linux и набор прикладных сервисов, которые содержат программные уязвимости. Использование данных уязвимостей в тестовых примерах атак позволяет успешно реализовать атаки на защищаемые узлы стенда для каждого из рассматриваемых классов компьютерных атак.

Использованы следующие прикладные сервисы:

- HTTP: Apache 2.0.x, Apache 1.2.x;
- Proftpd 1.2.x.

5.3.3. Рабочие станции

Для имитации рабочих станций защищаемой части стенда использованы узлы с уязвимыми версиями ОС Windows 2000/XP. Использованы следующие уязвимые сервисы:

- Встроенный сервис Windows RPC.

5.3.4. Атакующие узлы

Атакующие узлы attacker1 и attacker2 функционируют под управлением ОС Debian Linux. Каждый атакующий узел подключен к сети с помощью 100Мбитного сетевого интерфейса. На атакующих узлах используется одинаковый набор тестовых примеров.

5.4. Порядок испытаний

Испытания проводились в два этапа:

1. Имитация нормального сетевого взаимодействия между узлами attacker1,2 и сетевыми сервисами на защищаемых узлах www, ftp, win1,2. Имитация выполняется с помощью следующих стандартных клиентов:
   - *wget* для HTTP-сервера и FTP-сервера;
   - *rdesktop* для Windows-машин (с включённой службой удаленного доступа);
   - *scp, sftp* для сервера SSH.
2. Проведение атак на фоне имитации нормального сетевого взаимодействия с узлов attacker1,2 на узлы www, ftp, win1,2.

На первом этапе проводилась передача файлов со случайными бинарными данными с узлов www, ftp на узлы attacker1,2 по соответствующим протоколам в цикле. Было использовано 10 файлов со случайными данными суммарным объёмом 10Гб. Данный этап предназначен для оценки уровня ложных срабатываний экспериментальной СОА.

На втором этапе атаки проводились на фоне тех же примеров нормального поведения. Атаки проводились с паузой между последовательными тестами в 10 минут (время, необходимое на разблокирование узла на межсетевом экране).

В ходе проведения атак замерялись (вычислялись) следующие параметры:

- время атаки;
- класс атаки;
- число проведенных атак;
- число сообщений об атаках;
- успех/блокирование атаки.

Результаты испытаний приведены в следующем разделе.

5.5. Результаты испытаний

После проведения двух этапов испытаний получены следующие результаты (таблица 4):

№	Название атаки	N тестов	Ресурс	Тип	% успешных	N сообщений об атаке	% блокировано	% ложных срабатываний
1	proftp_put_down1	8	ftp	remote root	100%	24	100%	0%
2	proftp_put_down2	8	ftp	remote root	100%	24	100%	0%
3	propro	4	ftp	remote root	0%	8	100%	0%
4	babcia	4	ftp	remote root	0%	8	100%	0%
5	Squ1rt	4	http	DoS	100%	20	100%	0%
6	apache-nosejob	4	http	remote root	100%	8	100%	0%
7	blackjack	4	ssh	remote root	100%	8	100%	0%
8	winrpc1	4	dcom	remote root	100%	8	100%	0%
9	winrpc2	4	dcom	remote root	100%	8	100%	0%
10	nmap	10	0-1024	port scan	100%	51	100%	2%

Таблица 4. Результаты испытаний экспериментальной СОА.

5.6. Выводы

Число сообщений об атаках во всех тестах превышает число проведённых атак, что в общем случае понижает информативность СОА для оператора, хотя и не влияет на эффективность обнаружения атак. В данном случае повышенное количество сообщений об атаках объясняется избыточностью схемы размещения компонентов СОА: узлы sensor1 и sensor2 являлись «транзитными» для всего сетевого трафика между атакующими узлами и защищаемыми узлами сети.

Доля ложных срабатываний для обнаружения сканирования ресурсов составила 2% - из-за недостаточно точно определенного порога срабатывания сценария обнаружения сканирования. В реальных сетях с открытыми в Интернет сервисами уровень ложных срабатываний в 2% является неприемлемо высоким, поэтом для таких сетей необходима настройка порога срабатывания сценария. Ложные срабатывания наблюдались для сессий нормального трафика, что в условиях реальной РИС недопустимо, если все сообщения об атаках вызывают автоматическую блокировку узла-нарушителя. Таким образом, настройка политики реагирования для защищаемой сети является существенной задачей, но подробное рассмотрение данной проблемы выходит за рамки данной работы.

Также испытания показали, что сценарий, реализующий автомат второго рода, обнаруживал только успешные атаки на сервис FTP. В то время как сценарий, реализующий автомат первого рода для соответствующей атаки, обнаруживает все попытки реализации атаки, а не только успешные. Данная особенность метода также важна для разработки политики реагирования защищаемой РИС, так как реагирование может быть ресурсоёмкой процедурой в условиях конкретной РИС и для таких систем важно минимизировать число реакций.

Свойство адаптивности к неизвестным атакам проверено на примере проведения успешных атак на сервис ProFTPd без загрузки сценариев, реализующих автоматы первого рода для соответствующих атак. В этих тестах успешные атаки были обнаружены сценарием, реализующим автомат второго рода (без идентификации версии атаки).

В описанных тестах загрузка центрального процессора на сетевых сенсорах, на которые приходилась максимальная вычислительная нагрузка по анализу сетевого трафика, не превышала 30%. При этом пиковая пропускная способность составляла 12Мб/сек при получении файлов по протоколу HTTP. Следует отметить, что вычислительная сложность анализатора с увеличением числа сценариев растёт пропорционально и, при большом числе сценариев, время на обработку сетевого трафика может превысить возможности вычислителя. Данная проблема может быть решена разными способами: распараллеливание, построение обобщённого сценария и его минимизация, либо простое уменьшение количества сценариев за счёт выбора только тех, которые анализируют поведение реально существующих объектов сети. Представляется перспективным исследовать возможность распараллеливания рассматриваемой задачи, при котором для каждой наблюдаемой траектории защищаемого объекта строится локальная очередь событий, каждая из которых анализируется независимо. При такой схеме можно использовать линейное распределение анализа траекторий по заданному числу процессоров. Если большинство траекторий независимы друг от друга и обмен данными между локальными очередями событий не требуется, прирост скорости обработки будет линейно зависим от числа процессоров.

В целом, экспериментальная СОА показала высокую эффективность обнаружения, низкий уровень ложных срабатываний (единицы сообщений на 10Гб сетевого трафика) и адаптивность к неизвестным атакам, что и являлось основной целью данной работы.

6. ЗАКЛЮЧЕНИЕ

Основные результаты работы заключаются в следующем:

- Построена модель функционирования РИС в условиях воздействия компьютерных атак, в рамках которой задача обнаружения атак сведена к задаче поиска подцепочек в цепочке символов. Данная модель позволила формально оценить вычислительную сложность предложенного в работе метода и показать его корректность;
- Предложен гибридный метод обнаружения атак на основе метода анализа систем переходов, позволяющий обнаруживать неизвестные атаки как отклонения наблюдаемого поведения сетевых объектов от нормального;
- На основе предложенных методов реализована система обнаружения атак для ОС Linux и Windows 2000/XP. Данная экспериментальня система показала высокую эффективность обнаружения на испытательном стенде. Экспериментально показана адаптивность системы к неизвестным атакам.

Предложенный метод обнаружения атак может быть использован для построения систем защиты распределенных вычислительных систем в условиях функционирования в сетях общего доступа, где высока вероятность появления новых

реализаций атак. Наибольшая эффективность метода достижима в тех системах, где множество классов объектов (используемых сервисов и программного обеспечения) ограничено и не меняется со временем существенным образом, что позволяет использовать модели нормального поведения для обнаружения атак.

Перспективой развития предложенного метода является исследование всех классов современных атак и оценка возможности построения грамматики для каждого класса в терминах операций доступа класса с ограниченным контекстом фиксированной длины (например, LR(k)-грамматику). Что позволит построить автомат, не только обнаруживающий произвольные атаки соответствующего класса, но и предсказывающий принадлежность наблюдаемой траектории классу атак до завершения атаки. Отдельный интерес представляет задача автоматизации построения автоматов первого и второго рода по заданным примерам атак и реальных приложений (включая приложения, доступные лишь в бинарном виде), а так же создание соответствующего инструментального средства.

7. ЛИТЕРАТУРА

1. Ahmed Awad E. Ahmed, Issa Traore, “Anomaly Intrusion Detection based on Biometrics.” // Proceedings of the 2005 IEEE, Workshop on Information Assurance, United States Military Academy, West Point, NY June 2005
2. Abraham A. and Thomas J., Distributed Intrusion Detection Systems: A Computational Intelligence Approach. // Applications of Information Systems to Homeland Security and Defense, Abbass H.A. and Essam D. (Eds.), Idea Group Inc. Publishers, USA, 2005
3. D Anderson, T Frivold, and A Valdes, “Next-generation intrusion-detection +ert system (NIDES)”. // Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, May 1995
4. Debra Anderson, Teresa F. Lunt, Harold Javitz, Ann Tamaru, and Alfonso Valdes, “Detecting unusual program behavior using the statistical component of the next generation intrusion detection system (NIDES)”. // Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, May 1995
5. J.P. Anderson, “Computer Security Threat Monitoring and Surveillance.” // J.P. Anderson Co, Fort Washington, PA, April 1980
6. Amoroso, Edward, G., Intrusion Detection // 1st ed., Intrusion.Net Books, Sparta, New Jersey, USA, 1999
7. Stefan Axelsson, “Research in Intrusion-Detection Systems: A Survey” // Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1999
8. Stefan Axelsson, “Intrusion detection systems: A survey and taxonomy.” // Technical Report 99-15, Chalmers Univ., March 2000
9. Bace R. An Introduction to Intrusion Detection Assessment for System and Network Security Management. // 1999
10. Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, E. H. Spafford, Diego Zamboni, “An Architecture for Intrusion Detection using Autonomous Agents”. // Department of Computer Sciences, Purdue University; Coast TR 98-05; 1998
11. A. Baur & W. Weiss, “Audit Analysis Tool for Systems with High Demands Regarding Security and Access Control.” // Research Report, ZFE F2 SOF 42, Siemens Nixdorf Software, München, November 1988
12. M. Blanc, L. Oudot, and V. Glaume, "Global Intrusion Detection: Prelude Hybrid IDS." // Technical Report, 2003
13. Damiano Bolzoni, Sandro Etalle, “APHRODITE: an Anomaly-based Architecture for False Positive Reduction” // University of Twente, The Netherlands, Arxiv preprint cs.CR/0604026, 2006
14. Damiano Bolzoni, Emmanuele Zambon, Sandro Etalle, Pieter Hartel, “Poseidon: A 2-tier anomaly-based intrusion detection system.” // Technical report, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands. November 2005
15. J.M. Bradshaw, An introduction to software agents // J.M. Bradshaw (Ed.), Software Agents, AAA1 Press/MIT Press, Cambridge, MA, 1997, pp. 3-46 (Chapter 1).
16. Kalle Burbeck, “Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks.” // Department of Computer and Information Science, Linköpings universitet, Linköping, Sweden, Linköping, 2006
17. Gabriela F. Cretu, Janak J. Parekh, Ke Wang, Salvatore J. Stolfo, “Intrusion and Anomaly Detection Model Exchange for Mobile Ad-Hoc Networks.” //

Department of Computer Science, Columbia University, New York, US, 2005-2006
18. Herve Debar, Marc Dacier, and Andreas Wespi, “Towards a Taxonomy of Intrusion Detection Systems.” // Computer Networks, vol. 31, pp. 805-822, 1999
19. Christian Charras, Thierry Lecroq, Exact String Matching Algorithms, [http://www-igm.univ-mlv.fr/~lecroq/string/index.html], 1997
20. S. Stani ford Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K Levitt, C. Wee, R. Yip, and D. Zerkle, “GrIDS—A graph based intrusion detection system for large networks.” // In Proceedings of the 19th National Information Systems Security Conference, 1996
21. S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, D. Zerkle, The design of GrIDS: a graph-based intrusion detection system. // Technical Report CSE-99-2, Department of Computer Science, University of California at Davis, Davis, CA, January 1999.
22. Chung, M., Puketza, N., Olsson, R.A., & Mukherjee, B. (1995) Simulating Concurrent Intrusions for Testing Intrusion Detection Systems: Parallelizing. // NISSC. pp. 173-183.
23. M. Crosbie, E. Spafford, Defending a computer system using autonomous agents. // Technical Report 95-022, COAST Laboratory, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398, March 1994.
24. M. Crosbie, E. Spafford, Defending a computer system using autonomous agents. // Proceedings of the 18th National Information Systems Security Conference, October 1995.
25. M. Crosbie, G. Spafford, Active defense of a computer system using autonomous agents. // Technical Report 95-008, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398, February 1995.
26. Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford, “IDIOT— Users Guide.” // The COAST Project, Dept. of Computer Science, Purdue University, West Lafayette, IN, USA, 4 September 1996
27. W Cui, R. Katz, and W. Tan. “BINDER: An Extrusion- Based Break-In Detector for Personal Computers.” // Proc. USENIX Annual Technical Conference, 2005
28. Dasgupta D., Gonzalez F., K. Yallapu, Gomez, J., Yarramsettii, R., Dunlap, G. and Greveas, M., “CIDS: An Agent-based Intrusion Detection System.” // CS Technical Report No. CS-02-001., Feb, 2002
29. Vaibhav Gowadia, Csilla Farkas, Marco Valtorta, “PAID: A Probabilistic Agent-Based Intrusion Detection system.” // Computers & Security, 2005
30. Herve Debar, Monique Becker, and Didier Siboni, “A neural network component for an intrusion detection system.” // Proceedings of the 1992 IEEE Computer Sociecty Symposium on Research in Security and Privacy, pages 240–250, Oakland, CA, USA, May 1992
31. Denault, M., Gritzalis, D., Karagiannis, D., and Spirakis, P. (1994). Intrusion Detection: Approach and Performance Issues of the SECURENET System. // Computers and Security Vol. 13, No. 6, pp. 495-507.
32. Denning, Dorothy. An Intrusion-Detection Model. // IEEE Transactions on Software Engineering, Vol. SE-13, No. 2., 1987

33. Cheri Dowel and Paul Ramstedt, “The computer watch data reduction tool.” // Proceedings of the 13th National Computer Security Conference, pages 99–108, Washington DC, USA, October 1990
34. S.T. Eckmann, G. Vigna, and R. A. Kemmerer. “STATL: An Attack Language for State-based Intrusion Detection”. // Dept. of Computer Science, University of California, Santa Barbara, 2000.
35. W.M. Farmer, J.D. Guttman, V. Swarup, Security for mobile agents: issues and requirements. // Proceedings of the 19th National Information Sytems Security Conference, vol. 2, National Institute of Standards and Technology, October 1996.
36. Debora Frincke, Don Tobin, Jesse McConell, A framework for cooperative intrusion detection. // Dept. of Computer Science, University of Idaho, Moscow, USA.
37. Ian Goldberg, David Wagner, Randi Thomans, and Eric Brewer, “A secure environment for untrusted helper applications (confining the wily hacker).” // Proceedings of the Sixth USENIX UNIX Security Symposium, San Jose, California, USA, July 1996
38. R. Gopalakrishna, E. H. Spafford, and J. Vitek, “Efficient Intrusion Detection Automaton Inlining.” // Proc. IEEE Symp. on Security & Privacy, Oakland, CA, May 2005
39. Vladimir I. Gorodetski, Igor V. Kotenko. “Attacks Against Computer Network: Formal Grammar-Based Framework and Simulation Tool”. // St. Petersburg Institute for Informatics and Automation, RAID 2002: 219-238
40. Noria Foukia, “IDReAM: Intrusion Detection and Response executed with Agent Mobility Architecture and Implementation.” // Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, the Netherlands, 2005
41. Jani Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu, “ASAX: Software architecture and rule-based language for universal audit trail analysis.” // In Yves Deswarte et al., editors, Computer Security – Proceedings of ESORICS 92, volume 648 of LNCS, pages 435–450, Toulouse, France, 23–25 November 1992
42. J. Habra, B. Le Charlier, A. Mounji & I. Mathieu, “Preliminary Report on Advanced Security Audit Trail Analysis on UniX” // Research Report 1/92, Institut d’Informatique, University of Namur, January 1992.
43. N. Habra, B. Le Charlier, A. Mounji, Advanced Security Audit Trail Analysis on uniX. Implementation Design of the NADF Evaluator. // Research Report, Computer Science Institute, University of Namur, Belgium, March 1993.
44. Hatch, Brian, LIDS overview, 2001, [http://www.lids.org/]
45. R. Heady, G. Luger, A. Maccabe, M. Servilla, The architecture of a network level intrusion detection system. // Technical Report, University of New Mexico, Department of Computer Science, August 1990.
46. Todd Heberlein, Gihan Dias, Karl Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber, “A network security monitor”. // Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, pages 296–304. IEEE, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1990
47. Helman, P., Liepins, G., and Richards, W. Foundations of Intrusion Detection. // Proceedings of the Fifth Computer Security Foundations Workshop pp. 114-120. 1992
48. Y. Ho, Partial order state transition analysis for an intrusion detection system. // Master’s thesis, University of Idaho, 1997.

49. Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josehpine Ford. NADIR: An automated system for detecting network intrusion and misuse. // Computers & Security, 12(3):235–248, 1993
50. S.A. Hofmeyr, An immunological model of distributed detection and its application to computer security. // Ph.D. thesis, University of New Mexico, May 1999.
51. Koral Ilgun, "USTAT: A real-time intrusion detection system for UNIX". // In Proceedings of the 1993 IEEE Symposium on Security and Privacy, pages 16–28, Oakland, California, 24–26 May 1993,
52. Koral Ilgun, Richard A Kemmerer, and Phillip A Porras, "State transition analysis: A rule-based intrusion detection approach". // IEEE Transactions on Software Engineering, 21(3):181–199, March 1995
53. Kathleen A Jackson, David H DuBois, and Cathy A Stallings, "An +ert system application for network intrusion detection." // Proceedings of the 14th National Computer Security Conference, pages 215–225,Washington, D.C., 1–4 October 1991
54. Rasool Jalili, Fatemeh Imani-Mehr, Morteza Amini, Hamid Reza Shahriari, "Detection of Distributed Denial of Service Attacks Using Statistical Pre-Processor and Unsupervised Neural Networks." // Lecture notes in computer science, 2005
55. Y. Frank Jou, Fengmin Gong, Chandru Sargor, Shyhtsun FelixWu, and CleavelandW Rance, "Architecture design of a scalable intrusion detection system for the emerging network infrastructure." // Technical Report CDRL A005, Dept. of Computer Science, North Carolina State University, Releigh, N.C, USA, April 1997
56. Minna Kangasluoma, Policy Specification Languages, Department of Computer Science, Helsinki University of Technology, 1999, [http://www.nixu.fi/~minna/draft2.html]
57. Calvin Ko, "Execution Monitoring of Security-critical Programs in a Distributed System: A Specification-based Approach." // PhD thesis, Department of Computer Science, University of California at Davis, USA, 1996
58. Calvin Ko, George Fink, and Karl Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring." // Proceedings of the 10th Annual Computer Security Applications Conference, volume xiii, pages 134–144. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994
59. Christopher Kruegel, Giovanni Vigna, William Robertson, "A multi-model approach to the detection of web-based attacks." // Computer Networks 48, 717–738, 2005
60. Sandeep Kumar and Eugene H. Spafford, "A pattern matching model for misuse intrusion detection." // Proceedings of the 17th National Computer Security Conference, pages 11–21, Baltimore MD, USA, 1994
61. Sandeep Kumar and Eugene H. Spafford, "An application of pattern matching in intrusion detection." // Technical Report CSD-TR-94-013, The COAST Project, Dept. of Computer Sciences, Purdue University, West Lafayette, IN, USA, 17 June 1994
62. Sandeep Kumar and Eugene H. Spafford, "A software architecture to support misuse intrusion detection." // Technical report, The COAST Project, Dept. of Comp. Sciences, Purdue Univ.,West Lafayette, IN, 47907–1398, USA, 17 March 1995

63. Sandeep Kumar, “Classification and Detection of Computer Intrusions.” // PhD thesis, Purdue University, West Lafayette, Indiana, August 1995.
64. Håkan Kvarnström, “A survey of commercial tools for intrusion detection”. // Technical Report 99-8, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1999
65. Tao Li, Xiaojie Liu, and Hongbin Li “A New Model for Dynamic Intrusion Detection.” // lecture notes in computer science, 2005
66. Lionel Litty, “Hypervisor-based Intrusion Detection.” // Graduate Department of Computer Science, University of Toronto, 2005
67. W Lu, I Traore, “A new unsupervised anomaly detection framework for detecting network attacks in real-time.” // Department of Electrical and Computer Engineering, University of Victoria, Lecture notes in computer science, 2005
68. Lunt, T.F. Real-Time Intrusion Detection. // Computer Security Journal Vol. VI, Number 1. pp. 9-14., 1989
69. T.F. Lunt, “Automated Audit Trail Analysis and Intrusion Detection: A Survey.” // Proceedings of the 11th National Security Conference, Baltimore, MD, October 1988.
70. Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Caveh Jalali, and Peter G. Neuman, “A real-time intrusion-detection +ert system (IDES).” // Technical Report Project 6784, CSL, SRI International, Computer Science Laboratory, SRI Intl. 333 Ravenswood Ave., Menlo Park, CA 94925-3493, USA, February 1992
71. Bharath Madhusudan, John W. Lockwood, “A HARDWARE-ACCELERATED SYSTEM FOR REAL-TIME WORM DETECTION.” // Micro, IEEE, 2005
72. Masayoshi Mizutani, Shin Shirahata, Masaki Minami, Jun Murai, “The Design and Implementation of Session Based NIDS.” // IEICO. pages 551-562, Mar 2005
73. A. Mounji, Languages and Tools for Rule-Based Distributed Intrusion Detection. // PhD Thesis, Computer Science Institute, University of Namur, Belgium, Sept 1997.
74. A. Mounji, B. Le Charlier, D. Zampuniéris, N. Habra, Preliminary Report on Distributed ASAX. // Research Report, Computer Science Institute, University of Namur, Belgium, May 1994
75. Mukherjee, B., Heberlein, L.T., Levitt, K.N. (May/June, 1994). Network Intrusion Detection. // IEEE Network. pp. 28-42.
76. Srinivas Mukkamala, Andrew H. Sung, Ajith Abraham, “Intrusion detection using an ensemble of intelligent paradigms.” // Journal of Network and Computer Applications, 2005
77. Daniel C. Nash , An Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computers. // Blacksburg, Virginia, 2005
78. Daniel C. Nash, Thomas L. Martin, Dong S. Ha, and Michael S. Hsiao, “Towards an Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computing Devices.” // Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, 2005
79. Ozturk Ahmet, OSSEC-HIDS Capabilities, Architecture and plans. // Presentation at the 5th Linux and Free Software Festival, Ankara, Turkey, 2006.
80. Vern Paxon, “Bro: A system for detecting network intruders in real-time.” // In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 1988

81. V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time. // Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec. 1999
82. Phillip A Porras and Alfonso Valdes, “Live traffic analysis of TCP/IP gateways.” // Proceedings of the 1998 ISOC Symposium on Network and Distributed Systems Security, San Diego, California, 11–13 March 1998
83. Philip A Porras and Peter G Neumann, “EMERALD: Event monitoring enabling responses to anomalous live disturbances.” // Proceedings of the 20th National Information Systems Security Conference, pages 353–365, Baltimore, Maryland, USA, 7–10 October 1997
84. Porras, P. A., Ilgun, K., and Kemmerer, R. A. (1995). State transition analysis: A rule-based intrusion detection approach. // IEEE Transactions on Software Engineering,SE-21: 181–199.
85. T.H. Ptacek, T.N. Newsham, Insertion, evasion, and denial of service: eluding network intrusion detection. // Technical Report, Secure Networks, January 1998.
86. Puketza, N., Chung, M., Olsson, R.A. & Mukherjee, B. (September/October, 1997). A Software Platform for Testing Intrusion Detection Systems. // IEEE Software, Vol. 14, No. 5
87. Guangzhi Qu, Salim Hariri, Mazin Yousif “Multivariate Statistical Analysis for Network Attacks Detection.” // Computer Systems and Applications, 2005
88. Marcus J. Ranum, Experiences Benchmarking Intrusion Detection Systems, [http://www.snort.org/docs/Benchmarking-IDS-NFR.pdf]
89. Roesch, Martin, Snort Users Manual, Snort Release: 2.4, 2007, [http://www.snort.org/]
90. Sebring, M., Shellhouse, E., Hanna, M. & Whitehurst, R. Expert Systems in Intrusion Detection: A Case Study. // Proceedings of the 11th National Computer Security Conference, 1988
91. Mohammed Shahidul Alam “APHIDS++: Evolution of A Programmable Hybrid Intrusion Detection System.” // The University Of British Columbia, Vancouver, Canada, 2005
92. S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. Lin Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal, D. Mansur, DIDS (distributed intrusion detection system) ± motivation, architecture, and an early prototype. // Proceedings of the 14th National Computer Security Conference, Washington, DC, October 1991.
93. Sheyner, Oleg “Scenario Graphs and Attack Graphs.” // PhD thesis, SCS, Carnegie Mellon University, 2004.
94. S. Smaha, “Haystack: an intrusion detection system” // 4th Aerospace Computer Security Applications Conf., pp. 37–44. October 1988
95. Eugene H. Spafford, Diego Zamboni, Intrusion detection using autonomous agents. // Computer Networks, 34(4):547-570, October 2000.
96. Steven R Snapp, Stephen E Smaha, Daniel M Teal, and Tim Grance, “DIDS (Distributed Intrusion Detection System) Motivation, Architecture, and An Early Prototype.” // Proceedings of the Summer USENIX Conference, pages 227–233, San Antonio, Texas, 8–12 June 1992
97. Eugene H. Spafford, Diego Zamboni, Intrusion detection using autonomous agents. // Computer Networks, 34(4):547-570, October 2000.
98. Mukkamala Srinivas, Sung Andrew H, Abraham Ajith, Ramos Vitorino. “Intrusion detection systems using adaptive regression splines.” // Seruca I, Filipe J, Hammoudi S, Cordeiro J, editors. Sixth international conference on enterprise information systems. ICEIS’04, Portugal, vol. 3. 2004

99. T.Srinivasan, Jayesh Seshadri, J.B.Siddharth Jonathan, Arvind Chandrasekhar, “A System for Power-aware Agent-based Intrusion Detection (SPAID) in Wireless Ad Hoc Network.” // Lecture notes in computer science, 2005
100. Staniford-Chen, S. Using Thumbprints to Trace Intruders. // UC Davis, 1995
101. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, GrIDS: a graph based intrusion detection system for large networks. // Proceedings of the 19th National Information Systems Security Conference, vol. 1, National Institute of Standards and Technology, October 1996.
102. Teng, H. S., Chen, K., and Lu, S. C. Adaptive real-time anomaly detection using inductively generated sequential patterns. // Proceedings of the 1990 IEEE Symposium on Research in Computer Security and Privacy, 278–284, 1990
103. “Trusted Computer System Evaluation Criteria”, The Orange Book, Department of Defense, NCSC, National Computer Security Centre, DoD 5200.28-STD, December 1985.
104. G. Vigna, R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection Approach." // Proceedings of the 14th Annual Computer Security Application Conference, Scottsdale, Arizona, December 1998.
105. G. Vigna, R.A. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System." // Journal of Computer Security, 7(1), IOS Press, 1999.
106. R.A. Whitehurst, “Expert Systems in Intrusion Detection: A Case Study.” // Computer Science Laboratory, SRI International, Menlo Park, CA, November 1987.
107. Yoann Vandoorselaere, Laurent Oudot, “Prelude, an Hybrid Open Source Intrusion Detection System”, [http://www.prelude-ids.org/]
108. M.P.Zielinski, “Applying Mobile Agents in an Immune-system-based intrusion detection system.” // University of South Africa, 2004
109. ГОСТ Р 50922-96: Защита информации. Основные термины и определения. // Госстандарт России, Москва, 1996г.
110. Гамаюнов Д. Ю., Смелянский Р. Л., Модель поведения сетевых объектов в распределенных вычислительных системах. // Журнал «Программирование», 2007, №4.
111. Р.Л. Смелянский, Д. Ю. Гамаюнов. “Современные некоммерческие средства обнаружения атак”. // Факультет Вычислительной Математики и Кибернетики, МГУ им. М. В. Ломоносова, Москва, 2002 г.
112. Р.Л. Смелянский, А.И. Качалин. “Применения нейросетей для обнаружения аномального поведения объектов в компьютерных сетях”. // Факультет Вычислительной Математики и Кибернетики, МГУ им. М. В. Ломоносова, Москва, 2004.
113. Тараканов А.О. Математические модели обработки информации на основе результатов самосборки. // Диссертация д.ф.-м.н., С.-П., 1999.

ПРИЛОЖЕНИЕ: ЯЗЫК ОПИСАНИЯ ПОВЕДЕНИЯ СЕТЕВЫХ ОБЪЕКТОВ

7.1. Язык описания поведения объектов РИС

Язык описания поведения (ЯОП) синтаксически похож на язык STATL, основой которого, в свою очередь, является язык программирования C++. По сравнению с языком STATL, данный язык расширяется за счет логических предикатов состояний и переходов.

7.2. Препроцессор языка

Перед разбором текстовый файл, содержащий сценарии атак или описание нормального поведения, проходит обработку препроцессором.

Из препроцессора языка "C" взяты следующие конструкции.

Директивы включения файлов:
#include "filename"
#include <filename>

Директивы семантически не отличаются. Файл с именем filename включается в текущий компилируемый исходный файл.
Директива создания символических констант:
#define A B

Далее по тексту идентификатор A является синонимом строки B.
Директивы компиляции по условию:
#ifdef A/#else/#endif
#ifndef A/#else/#endif

Условием включения фрагмента файла в компилируемый исходный файл является наличие или отсутствие определения идентификатора A директивой #define.

Задавать дополнительные определения макроподстановок можно из командной строки исполняемого файла, реализующего модуль СОА.

7.3. Лексическая структура языка

Лексический анализатор языка получает в качестве входных данных результат работы препроцессора и, на выходе, формирует список лексем, поступающий в синтаксический анализатор. Лексическая структура языка определяет правила преобразования текста программы в последовательность лексем.

7.3.1. Комментарии

Комментарии выделяют либо парой '/*'..'*/' (вложенные комментарии не допускаются), либо парой '//'..<конец файла или конец строки>. Разрешено вложение двух вышеозначенных типов комментариев друг в друга, т.е. конструкции типа:

```
// комментарий /* комментарий 2 */
/* комментарий // комментарий 2 */
```

7.3.2. Константы

7.3.2.1. Целочисленные константы

Целочисленные константы не могут занимать более четырех байт в двоичном представлении. Представление идентично принятому в языке "C".

Десятичная форма: возможно задание знака '+' или '-' (можно опустить), далее число. Число не может начинаться с нуля, иначе оно трактуется как восьмеричное.

Шестнадцатеричная форма: '0x' или '0X', далее не более восьми шестнадцатеричных цифр (из набора 0-9,a-f,A-F).

Восьмеричная форма: символ '0', далее число в восьмеричном представлении.

Примеры:

123	Число 123
-123	Число –123
-0123	Неверно, т.к. восьмеричное представление не может иметь знака
0123	Число 83 (123 в восьмеричной системе счисления)
0x123	Число 291
0	0 (формально является восьмеричным представлением)
0xa	Число 10

Таблица. 5. Примеры целочисленных констант языка.

7.3.2.2. Символьные константы

Символьные константы заключаются в одинарные кавычки. Они всегда имеют тип u_int_8, поэтому, в отличие от "C", в кавычках может быть только один символ.

Кодирование специальных символов:

\n,\r,\t - символы перевода строки (код 10), возврата каретки (код 13), табуляции (код 9);

\<восьмеричная константа> - код символа в восьмеричном виде. В частности, \0 - нулевой символ;

\x<шестнадцатеричное число> - код символа в шестнадцатеричном виде;

\<символ, отличный от {n,r,t,x,0}> - символ, расположенный после слеша (символа \). В частности, полезны комбинации \',\",\\.

7.3.2.3. Строковые константы

Строковые константы заключаются в двойные кавычки. Символы задаются тем же образом, что и для символьных констант. Две идущие подряд строковые константы объединяются в одну.

Примеры:

“”	Пустая строка
“abc”	Строка abc
“\\a\n”	Строка \a<символ новой строки>
“\0\012\\\a”	<нулевой символ><символ с кодом 10>\a
“\”\x12”	Строка “<символ с кодом 18>
“\”	Синтаксическая ошибка: ничего не следует после символа “\”

Таблица 6. Примеры строковых констант.

7.3.2.4. Булевы константы

Тип данных bool определяет только два значения: true и false.

7.3.3. Символы операций

Символы операций представлены в таблице 7 в порядке убывания приоритета. Операции, не разделенные линией, имеют одинаковый приоритет. Ассоциативность определяет трактовку выражений вида “a op b op c”. Если в третьей колонке указано «слева направо», такие выражения следует вычислять как “(a op b) op c”. «Справа налево» изменяет порядок вычисления на “a op (b op c)”.

Название операции	Символ операции	Ассоциативность
Взятие элемента массива	[]	Слева направо
Вызов функции	()	
Получение поля структуры	.	Слева направо
Инкремент (постфиксная форма) Декремент (постфиксная форма)	++ -	
Инкремент (префиксная форма) Декремент (префиксная форма)	++ -	
Унарный плюс Унарный минус	+ -	
Логическое "НЕ"	!	
Двоичное дополнение	~	
Конвертация типов	(type)	Справа налево
Умножение	*	Слева направо

Деление	/	Слева направо
Взятие остатка	%	Слева направо
Сложение	+	Слева направо
Вычитание	-	Слева направо
Сдвиг влево	<<	Слева направо
Сдвиг вправо	>>	Слева направо
Больше	>	Слева направо
Меньше	<	Слева направо
Больше или равно	>=	Слева направо
Меньше или равно	<=	Слева направо
Проверка на равенство	==	Слева направо
Проверка на неравенство	!=	Слева направо
Побитовое И	&	Слева направо
Побитовое ИЛИ	|	Слева направо
Побитовое исключающее ИЛИ	^	Слева направо
Логическое И	&&	Слева направо
Логическое ИЛИ	||	Слева направо
Одно из двух	C?a:b	Справа налево
Присваивание	=	Справа налево
Умножение и присваивание	*=	Справа налево
Деление и присваивание	/=	Справа налево
Взятие остатка и присваивание	%=	Справа налево
Сложение и присваивание	+=	Справа налево
Вычитание и присваивание	-=	Справа налево
Сдвиг влево и присваивание	<<=	Справа налево
Сдвиг вправо и присваивание	>>=	Справа налево
Логическое И и присваивание	&=	Справа налево
Логическое ИЛИ и присваивание	|=	Справа налево

Логическое исключающее ИЛИ и присваивание	^=	Справа налево
Последовательное выполнение	,	Слева направо

Таблица 7. Символы операций языка.

7.3.4. Ключевые слова

Нижеследующие последовательности символов являются ключевыми словами языка. Каждое ключевое слово преобразуется из набора символов во внутренний идентификатор, понятный синтаксическому анализатору. Этот идентификатор подается на выход вместе с кодом лексемы.

Список ключевых слов языка описания сигнатур атак:

char
u_char
int
u_int
long
u_long
int_8
u_int_8
int_16
u_int_16
int_32
u_int_32
string
timer
if
for
while
do
break
continue
return
void
struct
true
false
scenario
state
transition
consuming
nonconsuming
unwinding

7.3.5. Идентификаторы

Идентификаторы служат для обозначения имен макроподстановок, функций, переменных, сценариев, состояний, переходов, структур и методов. Первый символ идентификатора должен быть символом английского алфавита (строчным или заглавным) или символом подчеркивания (_). Последующие символы могут состоять из тех же символов, что и первый, плюс символы цифр (0-9). Длина идентификатора не ограничена. Строчные и заглавные буквы различаются (например, идентификаторы test и Test считаются различными). В качестве идентификатора запрещается использовать ключевые слова (см. предыдущий пункт).

7.4. Синтаксис и семантика языка

На вход синтаксического анализатора языка поступает последовательность лексем. В результате работы синтаксического анализатора получается так называемое синтаксическое дерево программы. Синтаксическое дерево, преобразованное в поток байт, является промежуточным представлением программы на языке, которое и направляется в БД СОА для хранения.

7.4.1. Типы данных

7.4.1.1. Атомарные типы данных

Атомарные типы данных – это простейшие встроенные типы данных, которые можно использовать непосредственно или создавать из них другие (агрегатные) типы.

Имя типа(ов)	Множество значений
char = int_8	-128..127
u_int_8 = u_char = bool	0..255 (false=0, true=1)
int_16 = short	-32768..32767
u_int_16 = u_short	0..65535
int_32 = int = long	-2^{31}.. 2^{31}-1
u_int_32 = u_int = u_long	0.. 2^{32}-1
string	Строка произвольной длины. Состоит из символов типа u_char. Может содержать любые символы (в том числе и нулевой)
timer	Таймер
void	У данного типа нет значений. Нельзя определить переменную типа void. Тип используется лишь для инидикации того факта, что функция не возвращает значения.

Таблица 8. Атомарные типы данных языка.

Выражение для декларации (определения) переменной атомарного типа имеет следующий вид:

type var;

7.4.1.2. Агрегатные типы данных

Агрегатные типы данных создаются из атомарных типов данных.

7.4.1.2.1 Массивы

Разрешается определять массивы атомарных арифметических типов фиксированной длины. Массивы не могут служить аргументами для вызова функций.

Определение массива:

type var[N];

, где N - целочисленная положительная константа (больше нуля);

[] - операция индексации. В случае выхода индекса за пределы интервала 0..N-1 фиксируется ошибка времени выполнения.

Операция присваивания над массивами не определена.

7.4.1.2.2 Структуры

Определение типа структуры:

struct <struct-name> [: <derived-from>] {
<декларации полей и/или методов>
};

Внутри структуры могут быть поля-структуры, но запрещается определение типа структуры внутри (т.е. структуры, используемые внутри структуры, должны быть определены ранее).

Декларации полей идентичны декларациям атомарных переменных и массивов. Декларации методов идентичны декларациям функций (исключение делается для конструкторов - они определяются без параметров и возвращаемого значения).

Тела методов определяются позднее в тексте программы с именами вида <struct-name>::<method-name>. В методе можно использовать поля структуры напрямую, без применения операции '.'.

Следует отметить, что, в отличие от языка C, все структуры передаются в функции по ссылке. Это означает, что функция может изменить переданную в нее структуру.

Конструктор определяется так же, как метод. Конструктор, имя которого совпадает с именем структуры, не может иметь аргументов и возвращаемого значения. Конструктор вызывается каждый раз для инициализации структуры.

7.4.2. Операции над типами данных

7.4.2.1. Явное преобразование типов

Явное преобразование типов имеет вид (type)expr. Семантика операций преобразования типов представлена в таблице:

Тип данных	Преобразовывается к типу	Действия по преобразованию
Int	char, u_char, short, u_short	Отсечение старших разрядов
short	char, u_char	Отсечение старших разрядов
int, u_int	u_int, u_short, u_char	Возможное отсечение старших разрядов и последующее побитовое копирование
short, u_short	u_short, u_char	Возможное отсечение старших разрядов и последующее побитовое копирование
char, u_char	u_char, char	Побитовое копирование
char	short, int	Знаковое расширение до необходимого количества бит
u_char	short, int, u_short, u_int	Беззнаковое расширение до необходимого количества бит
short	int	Знаковое расширение до необходимого количества бит
u_short	int	Беззнаковое расширение до необходимого количества бит
char	u_short, u_int	Знаковое расширение до необходимого количества байт и дальнейшее побитовое копирование
short	u_int	Знаковое расширение до необходимого количества байт и дальнейшее побитовое копирование
timer	char, u_char, short, u_short, int, u_int	Результатом является значение 1, если произошло событие от таймера, и 0 в противном случае
Любой тип	void	Никакого действия не происходит

Таблица 9. Виды явного преобразования типов в языке.

Явное преобразование типов для структур допустимо. Проверок на корректность преобразования не производится. Поведение программы в случае некорректного преобразования типа не определено.

7.4.2.2. Неявное преобразование типа

Неявное преобразование типа происходит в том случае, если указано значение одного типа, в то время как в текущем контексте требуется другой тип. В этом случае

производится преобразование по правилам, описанным в предыдущем пункте. Исключение составляет преобразование структур. Единственным допустимым неявным преобразованием структуры является преобразование к типу, являющимся родительским.

7.4.2.3. Операции над арифметическими типами

Операции над арифметическими типами данных подчиняются следующим правилам:

- если операнды имеют различный тип, меньший по занимаемому размеру операнд преобразуется к типу большего;
- тип результата операции тот же, что и у большего по размеру операнда;
- операции сравнения выдают результат типа bool (он же u_char). Если два аргумента операции сравнения имеют различные знаки, используется знаковое сравнение и выдается предупреждение;
- арифметические константы, участвующие в бинарных операциях, имеют тип второго операнда, если они находятся в рамках значений, которые может принимать второй операнд. В противном случае константа имеет тип u_int, если она превышает значение 2^{31}-1, и тип int в противном случае;
- порядок вычисления операндов в бинарных операциях определен только для логических операций (&&,||) – слева направо. Причем если результат операции становится известен по первому операнду, второй операнд не вычисляется.

7.4.2.4. Операции над строковым типом

Строки могу содержать произвольные символы (в т.ч. и нулевой). Тип элементов строк – u_char.

Над данными строкового типа определены следующие операции:

- присваивание (=);
- конкатенация (+): типом второго аргумента может быть string либо u_char;
- конкатенация и присваивание (+=),(+): типом второго аргумента может быть string или u_char;
- индексация ([]): возвращается символ с заданным индексом. Если произошел выход за пределы строки, возвращается 0;
- сравнение (==,!=,>,<,>=,<=).

Остальные операции над строками возможны через использование встроенных функций.

7.4.2.5. Операции над таймером

Преобразование (явное или неявное) типа timer к bool выдает 'true', если событие от таймера произошло, и 'false' , если нет.

Установка (сброс) таймера происходит путем вызова встроенных функций.

7.4.2.6. Операции со структурами

Допустимы всего две операции над структурами: получение доступа к полю или методу (.) и присваивание. В последнем случае два операнда операции присваивания должны иметь идентичный тип.

7.4.3. Функции

Синтаксис операции декларации функции имеет вид:

[<return-type>] <func-name> '(' [<arg-type> <arg-name> [',' <arg-type> <arg-name> [...]] ')'

Если return-type не указан, функция возвращает тип int (исключение: конструкторы, которые никогда не возвращают значение). В качестве return-type допускается указывать тип 'void' (это единственное место, где этот тип можно указать). В этом случае функция не возвращает ничего.

Чтобы объявить прототип функции, необходимо после декларации функции поставить точку с запятой (';'). Для определения тела функции вместо точки с запятой следует определить блок операторов. Определение функции не может быть вложенным в другой блок или в определение структуры.

Если в программе присутствуют как прототип(ы) функции, так и определение тела, декларации должны быть идентичны с точностью до переименования переменных.

Все переменные, имеющие структурный тип, передаются по ссылке, остальные – по значению. Передача массивов в качестве аргументов запрещена.

Вызов функции имеет формат:

<func-name> ‘(‘ [arg1 [arg2 …]] ‘)’ // для функции, не являющейся методом

<struct-var>.<func-name> ‘(‘ [arg1 [arg2 …]] ‘)’ // для методов

Количество передаваемых аргументов должно в точности соответствовать количеству объявленных аргументов. Типы фактических аргументов должны допускать неявное преобразование к типу формального (описанного в декларации функции) аргумента. Допускается явный вызов конструктора (для переинициализации).

Вызов функции с синтаксической точки зрения имеет тип, возвращаемый функцией. Вызов конструктора имеет тип ‘void’.

7.4.4. Выражения

Выражение является композицией атомарных операндов (констант, переменных, вызовов функций и выражений в скобках), связанных между собой символами операций (символы операций были определены выше в данном документе). Каждое выражение имеет детерминированный тип (возможно, void).

7.4.5. Операторы

В ЯОСА поддерживаются следующие типы операторов:

- <выражение> ‘;’ – простой оператор (любое выражение с последующим за ним символом «точка с запятой»);

- if ‘(‘ <выражение> ‘)’ <оператор> – условное выполнение. Оператор выполняется, если <выражение>, будучи преобразовано к типу int, получает ненулевое значение. Если выражение не допускает преобразования к типу int, фиксируется ошибка синтаксического анализа;

- if ‘(‘ <выражение> ‘)’ <оператор> else <оператор> – условное выполнение одной из двух ветвей;

- while ‘(‘ <выражение> ‘)’ <оператор> – вычисляется выражение и, если оно не равно нулю, выполняется оператор и осуществляется повторное вычисление выражения и т.д. (цикл с предусловием);

- do <оператор> while ‘(‘ <выражение> ‘)’ – цикл с постусловием, отличающийся от предыдущего порядком вычисления выражения и выполнения оператора;

- for ‘(‘ <выражение1> ‘;’ <выражение2> ‘;’ <выражение3>) <оператор> – цикл, аналогичный циклу for языка “C”. До первой итерации происходит вычисление выражения 1 (результат игнорируется). Далее проверяется значение выражения 2. Если оно равно нулю, происходит выход из цикла. В противном случае выполняется оператор и вычисляется значение выражения 3. Все повторяется с момента вычисления выражения 2;

- ‘{‘ [<оператор1> [<оператор2> …]] ‘}’ – составной оператор (блок).

7.4.6. Сценарии

Определение сценария (описания сигнатуры атаки) имеет формат:

'scenario' <имя-сценария> '(' <список-аргументов> ')' '{'
[<декларация-переменной>|<декларация-состояния>|<декларация-перехода>]
'};'

Список аргументов в неявном виде задает класс событий, обрабатываемых сценарием. Типы структур, передаваемые в сценарий, сигнализируют о поддерживаемых сценарием событиях. Все аргументы, соответственно, должны быть структурами и должны быть унаследованы от структуры Event.

В теле сценария могут быть определены (как и в структуре) переменные и методы.

Каждому сценарию, в процессе работы, сопоставляется один или более контекстов выполнения. Контекст выполнения – это значения переменных, определенных внутри сценария, плюс текущее состояние.

7.4.6.1. Декларация состояния

Синтаксис операции:

['initial'] 'state' <имя-состояния> '{'
<логическое выражение>
<оператор>
'}'

Имя состояния является идентификатором, уникальным в пределах состояния. Логическое выражение определяет условие перехода в состояние (исключение: начальное состояние в момент создания копии сценария - в этом случае логическое выражение не вычисляется). Оператор выполняется при переходе в состояние.

Ключевое слово initial говорит о том, что состояние является начальным для сценария. В случае, когда ни для одного состояния не применено ключевое слово initial, начальным состоянием является первое состояние сценария.

Сценарий должен содержать хотя бы одно состояние.

7.4.6.2. Декларация перехода

Синтаксис:

'transition' [<имя-перехода>] '(' <имя-состояния> '->' <имя-состояния> ')'
['consuming'|'nonconsuming'|'unwinding'] '{'
<логическое выражение>
<оператор>
'}'

Осуществляется проверка логического выражения для состояния. Если логическое выражение не выполнено, дальнейших проверок не производится. Если оно выполнено, выполняется оператор, осуществляется переход и выполняется оператор, определенный в теле состояния.

Различаются следующие типы переходов:

- consuming – не создается новой копии контекста выполнения (все делается в текущей);
- nonconsuming – создается новая копия контекста выполнения, переход осуществляется в ней;
- unwinding – удаляются все контексты выполнения, порожденные данным контекстом выполнения с помощью переходов типа nonconsuming.

7.5. Встроенная библиотека языка

Встроенная библиотека языка определяет набор типов и функций, предоставляемый программам, написанным на языке описания сигнатур атак. С помощью языкового модуля предоставляется возможность расширять набор функций как на языке описания сигнатур, так и на C++.

7.5.1. Встроенные структуры

Структура Event:

```
struct Event {
	int	type;
};
```

Единственным полем данной структуры является поле, идентифицирующее тип произошедшего события.

Структура FastFindState:

```
struct FastFindState {
        int     state;
};
```

Это структура состояния автомата, просматривающего строку в поисках заданного шаблона. Должна рассматриваться как абстрактная структура (т.е. она может передаваться в качестве аргумента библиотечным функциям, но ее поля не должны использоваться напрямую).

7.5.2. Встроенные функции

bool IPMatch (u_int ipAddr, u_int ipNetwork, u_int ipMask) –

возвращает true в том и только в том случае, когда адрес ipAddr находится в сети ipNetwork с маской ipMask. Здесь и далее: упаковка ip-адресов в значение типа u_int происходит, начиная со старшего байта. Пример: адрес 192.168.0.1 будет представлен как 0xC0A00001.

string IPToString (u_int ipAddr) – преобразует ip-адрес в строку.

int strlen (string str) – возвращает длину строки.

int strchr (string str, u_char c)
int strrchr (string str, u_char c)

Возвращают индекс первого (или последнего для strrchr) символа в строке с кодом c. Если не найдено такого символа, возвращается -1.

int strstr (string str, string substr) –

возвращает первое вхождение подстроки substr в строку str или -1, если не найдено подстроки. Для критичного по времени нахождения подстрок вместо данной функции стоит пользоваться функциями нахождения с помощью конечного автомата (см. ниже).

string substr (string str, int pos, int size) –

возвращает подстроку строки str, начинающуюся с позиции pos и имеющую длину size (если size равно -1, возвращается остаток строки str до конца).

string itoa (int n)
string utoa (u_int n, int radix)

Преобразуют число, знаковое или беззнаковое, в строку. radix – система счисления (число от 2 до 36).

bool FastFind (string source, string substr)
bool FastMatch (string source, string pattern)
bool FastFindStateful (string source, string substr, FastFindState state)
bool FastMatchStateful (string source, string pattern, FastFindState state, int block_position)

Данные функции осуществляют поиск подстроки или шаблона в строке. Функции …Find… находят подстроку, функции …Match… находят шаблон.

Шаблон состоит из последовательности атомов. Атомом может являться:

- символ (в этом случае совпадение фиксируется при совпадении символа шаблона с символом в тексте);
- множество символов. Задается с помощью конструкций вида [abcd0-9] (это означает символы a,b,c,d и все символы, обозначающие цифры). Можно префиксовать выражение, стоящее в квадратных скобках, символом ‘^’; в этом случае произойдет «инверсия» выбора, т.е. [^a0-9] означает «все, кроме символа a и числовых символов»;
- символ «точка» (.). Совпадение с таким символом фиксируется в любом случае;
- символ ‘^’. Совпадение фиксируется только в начале строки;
- символ ‘$’. Совпадение фиксируется только в конце строки;
- произвольный шаблон, заключенный в круглые скобки.

Непосредственно после атома может находиться модификатор повторения:

- символ ‘*’. Это означает ноль или более раз;
- символ ‘+’ – один или более раз;
- {N} – ровно N раз;
- {N,M} – от N до M (включительно) раз.

Если после атома не стоит ни одной из вышеперечисленных конструкций, совпадение фиксируется в случае появления атома ровно один раз.

Любой из специальных символов в шаблоне ([]()\{}^$-.,) может быть префиксован с помощью обратного слэша (\) для утраты своего специального назначения.

Примеры:

Шаблон	Описание	Подходящие строки (примеры)
abc	Строка abc	abc
a.*b	Все строки, начинающиеся на ‘a’ и заканчивающиеся на ‘b’	ab, acb, acccccccdfb
a.+b	То же, что и в предыдущей строке, только между a и b должен быть хотя бы один символ	acb, acccccccdfb
[a-zA-Z_][a-zA-Z_0-9]*	Синтаксис идентификатора языка описания сигнатур атак	ab, _a09, Foo_007_ab
(ab[0-9]{0,1})*	Набор из последовательностей ‘ab’ с возможной цифрой после b	ab, abab, ab1ab, abab ab2
\(.*\)	Любая строка, заключенная	(), (a), (12345)

	в скобки	

Таблица 10. Примеры поддерживаемых шаблонов для быстрого поиска.

Для осуществления быстрого поиска системой поддержки выполнения программ строится конечный автомат для поиска всех возможных подстрок и шаблонов. Для построения автомата необходимо a priori знать все шаблоны, которые будет необходимо найти. Таким образом, для использования функций быстрого поиска возникает ограничение на параметр substr/pattern: он должен быть константой. При неконстантом аргументе все функции вернут false.

Функции поиска с учетом состояния (с окончанием Stateful) работают при поиске во фрагментах данных. Если сохранять промежуточное состояние, то такие функции, например, способны найти подстроку ‘attack’ в последовательно принятых фрагментах данных ‘xxxxatt’, ‘ac’, ‘kxxxx’. Для инициализации состояния следует использовать функцию FastFindInitState (см. ниже) и передавать параметр состояния поиска в функции. Функции поиска с учетом состояния не работают с аргументом substr/pattern, не являющимся константой.

Функция FastMatchStateful в качестве последнего аргумента берет позицию блока, что необходимо для корректной обработки символов $ и ^ в шаблоне. Позиция блока может быть одной из следующих констант:

- BLOCK_FIRST – это первый блок (т.е. ^ на первом символе блока выдаст совпадение);
- BLOCK_LAST – последний блок ($ на конце блока выдаст совпадение);
- 0 – ни первый, ни последний блок;
- BLOCK_FIRST | BLOCK_LAST – и первый, и последний блок.

Все функции возвращают true, если поиск успешен, и false в противном случае.

void FastFindInitState (FastFindState state) –

инициализирует состояние для поиска в начальное значение. Вызов обязателен перед вызовом FastFind…Stateful для первого фрагмента блока.