Сбор и анализ данных о событиях безопасности в крупных корпоративных сетях

Е.В. Чернова[1], П.Н. Полежаев[1], А.Е. Шухман[1], Ю.А. Ушаков[1], И.П. Болодурина[1], Н.Ф. Бахарева[2]

[1]Оренбургский государственный университет, проспект Победы 13, Оренбург, Россия, 460018
[2]Поволжский государственный университет телекоммуникаций и информатики, Льва Толстого 23, Самара, Россия, 443010

Аннотация. С каждым годом компьютерные сети становятся сложнее, что напрямую влияет на обеспечение высокого уровня защиты информации. Различные коммерческие сервисы, критические системы и информационные ресурсы, преобладающие в таких сетях, являются выгодными целями для террористов, кибершпионов и криминальных элементов. Последствия варьируются от кражи стратегической, высоко ценимой интеллектуальной собственности и прямых финансовых потерь до значительного ущерба бренду и доверию клиентов. Нарушители имеют преимущество – в сложных компьютерных сетях им легче скрыть свои следы. На сегодняшний день обнаружение и идентификация инцидентов безопасности является одной из самых важных и сложно реализуемых задач. Возникает необходимость обнаружить инциденты безопасности как можно скорее, анализировать и правильно среагировать на них, чтобы не затруднить работу компьютерной сети организации. Сложность заключается в том, что разные источники событий предлагают разный формат данных или могут дублировать событие. Также некоторые события сами по себе не указывают на какие-либо проблемы, однако их определенная последовательность может означать наличие инцидента безопасности. Все процессы сбора должны выполняться в режиме реального времени, что подразумевает потоковую обработку данных.

1. Введение

Компьютерные сети последнее время имеют тенденцию к быстрому развитию, они становятся больше и сложнее, но по-прежнему остаются выгодной целью для различных нарушителей – криминальных элементов, кибершпионов и даже террористов. Под удар попадают коммерческие сервисы, критические системы и информационные ресурсы. Последствия могут быть разные, начиная от кражи стратегически важной информации, высоко ценимой интеллектуальной собственности и прямых финансовых потерь до значительного ущерба бренду и доверию клиентов.

Традиционный подход к кибербезопасности основан на идее, что нужно определить особую доверительную среду для своих сетей и данных, то есть организовать их так, чтобы уменьшить доступ к ним извне, но не мешать им корректно выполнять свои функции. Это поможет обнаружить и устранить уязвимости прежде чем нарушитель сможет их найти. В современных компьютерных сетях при постоянно меняющихся сценариях угроз подобный подход уже не так эффективен. Атаки могут быть организованы в любое время и в любом месте, а из-за

сложности сетей нарушителям проще скрыть свои следы. В теории, специалисты должны быть готовыми ко всем возможным вариантам событий, но на практике это невозможно. Поэтому для защиты систем требуется собирать данные со всей сети, понимать, как она работает, обнаруживать и идентифицировать угрозы, принимать соответствующие меры так быстро, как только это возможно. Конечно, существует огромное количество решений для выполнения этих задач, однако не все из них бесплатны, способны взаимодействовать с большинством доступных источников событий безопасности и работать в распределенных сетях [1,2,3].
В данной статье производится создание концепции системы для сбора и первичного анализа событий и инцидентов безопасности в больших корпоративных сетях.

2. Выбор и обзор программных средств

Основными требованиями для программных средств являются открытый исходный код, возможность работы в распределенных системах и поддержка потоковой обработки данных. Поскольку в реальном времени формируется большое количество разнородных данных, которые нужно привести к общему виду, обработать и проанализировать, следует также учесть, что выбранные инструменты должны поддерживать возможность работать с потоковыми данными. Под такими данным подразумевают данные, непрерывно предоставляемые множеством источников, из которых формируются небольшие пакеты. Источниками таких данных являются системы аутентификации, активное сетевое оборудование, IDS/ISP, журналы событий серверов, антивирусы, сканеры уязвимостей и прочие системы защиты и контроля ИБ.
В качестве фреймворка для работы с потоковыми данными будет использоваться Apache Spark [4]. Он работает намного быстрее, чем Hadoop, поддерживает кластерный режим, совместим с другими продуктами Apache. Обладает хорошо структурированной документацией и достаточно популярен среди разработчиков, а значит имеет огромное количество статей и мануалов. Spark может работать в среде Hadoop под управлением YARN, предоставляет API для Scala, Java, Python, R и поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД, Cassandra, Amazon S3, Kudu, MapR-FS. Кроме того, в Spark входят следующие библиотеки: Spark SQL для работы с запросами SQL, Spark Streaming для обработки потоковых данных, MLib для машинного обучения, GraphX для работы с графами.
Apache Spark имеет расширение базового API Spark Streaming. Оно не обрабатывает потоки целиком, а разбивает их на небольшие пакеты временных интервалов. Это называется DStream (discretized stream, дискретизированный поток). DStream является микро-пакетом, содержащим несколько RDD (resilient distributed dataset, отказоустойчивый распределенный набор данных). Они могут обрабатываться параллельно с помощью произвольных функций и преобразований на основе скользящего окна (sliding window). RDD поддерживают операции двух типов: трансформации и действия. Результат трансформации – это новый RDD, а действия – какое-либо значение. Трансформации выполняются не сразу, то есть Spark запоминает какую именно трансформацию и над какими данными следует совершить, а результат возвращается, когда вызывается действие. Подобный подход повышает эффективность Spark. Кроме RDD существует DataSet. Его отличие в том, что перед выполнением трансформаций срабатывает оптимизатор, который определяет более эффективный способ получения результата. Работа с ним происходит преимущественно с помощью запросов SQL.
Приложения Spark выполняются как независимые наборы процессов в кластере, координируемые объектом SparkContext в основной программе (называемой драйвером). В частности, SparkContext может подключаться к нескольким типам диспетчеров кластеров, которые распределяют ресурсы между приложениями. После подключения Spark получает исполнителей на узлах кластера, которые являются процессами, выполняющими вычисления и хранящими данные для приложения. Затем он отправляет код приложения исполнителям. Наконец, SparkContext отправляет задачи исполнителям для выполнения.
Spark также имеет недостатки. В первую очередь это “наследство” пакетной обработки – непостоянные нагрузки на сеть при загрузке данных и обработке. Поэтому возникает необходимость устанавливать ограничения плотности входного потока, поскольку Spark не

имеет эффективного инструмента для его отслеживания. Также он не умеет восстанавливать работу кластеров после сбоев.

Для решения данных проблем дополнительно будет использоваться фреймворк Apache Kafka, способный создавать конвейеры данных в режиме реального времени и потоковые приложения [5]. Kafka позволяет спроектировать распределенный сервер для формирования очередей сообщений. Таким образом, поток данных распределяется на другие машины в кластере, обеспечивая большую масштабируемость. Благодаря такому реплицированному и постоянному хранилищу снижается риск потери данных.

Основной абстракцией Kafka является раздел (topic) – это категория или имя ленты, в которую публикуются записи. Разделы в Kafka всегда имеют несколько подписчиков, то есть может быть ноль, один или несколько потребителей, которые подписываются на записанные в нее данные. Секция представляет собой упорядоченную не изменяемую последовательность записей, которая постоянно добавляется в структурированный журнал логов. Каждой записи в секциях присваивается последовательный идентификационный номер, называемый смещением, который однозначно идентифицирует каждую запись в секции.

Секции журнала логов распределяются по серверам кластера Kafka, при этом каждый сервер обрабатывает данные и запросы на общий доступ к секциям. Каждая секция реплицируется на настраиваемое число серверов для обеспечения отказоустойчивости и имеет один сервер, который действует как "лидер" и ноль или более серверов, которые действуют как "последователи". Лидер обрабатывает все запросы на чтение и запись секции, а последователи пассивно его реплицируют. Если лидер потерпит неудачу, один из последователей автоматически станет новым лидером. Каждый сервер выступает в качестве лидера для некоторых своих секций и в качестве последователя для других, поэтому нагрузка в кластере хорошо сбалансирована.

Kafka Streams – клиентская библиотека обработки и анализа входящих и исходящих данных, находящихся в Kafka. Рассмотрим ее концепцию. Топология процессора – это граф потоковых процессоров (узлов), которые связаны потоками (ребрами). Поток (stream) представляет собой неограниченный, постоянно обновляемый набор данных. Это упорядоченная, воспроизводимая и отказоустойчивая последовательность неизменяемых записей данных. Под записью данных имеется в виду пара ключ-значение. Потоковый процессор (stream processor) – это шаг обработки для преобразования данных в потоках путем получения одной входной записи за раз от вышестоящих процессоров в топологии, применяя к ней свою операцию, и может впоследствии создать одну или несколько выходных записей для своих нижестоящих процессоров. Существует два типа специальных процессоров: исходный (source) и конечный (sink). Исходный процессор не имеет вышестоящих процессоров, поскольку сам создает входной поток в топологию и пересылает сообщения дальше нижестоящим процессорам. Конечный процессор наоборот не имеет нижестоящих процессоров. Любая программа, которая использует библиотеку Kafka Streams, называется приложением потоковой обработки. Оно определяет свою вычислительную логику через одну или несколько топологий процессора.

Топология процессора приложения масштабируется путем разбиения на несколько задач. Kafka Streams создает фиксированное количество задач на основе секцией входного потока для приложения, причем каждой задаче назначается список секцией из входных потоков (т.е. разделов Kafka). Назначение секций задачи не меняется, так что каждая задача является фиксированной единицей параллелизма приложения. Затем задачи могут создать собственную топологию процессора на основе назначенных секцией. Каждый поток может выполнять одну или несколько задач со своими топологиями процессоров независимо друг от друга. Запуск большего количества потоков или нескольких экземпляров приложения просто сводится к репликации топологии и ее обработке другого подмножества секцией Kafka, эффективно распараллеливая обработку. Это упрощает параллельное выполнение топологий в экземплярах приложений и потоках.

3. Организация передачи и хранения данных

После выполнения различных операций нужно где-то хранить полученные результаты. Apache Cassandra выбрана в качестве NoSQL системы управления базами данных [6]. Cassandra обладает высокой масштабируемостью и пропускной способностью для операций записи и чтения, поддерживает репликацию. Не может работать с SQL, однако имеет подобный ему язык CQL (Cassandra Query Language). Большим преимуществом является возможность создавать надежные и устойчивые кластеры. Подключение новых экземпляров Cassandra к кластеру происходит очень легко, а поскольку узлы кластеры равнозначны и данные согласованы, то пользователь может обращаться к любому из них как для записи, так и для чтения.

Необработанные данные поступают в систему от источников, а именно сетевых устройств, антивирусов, файерволов, сетевых экранов и т.п. Для получения данных из этих систем есть Kafka Connect – инструмент для масштабируемой и надежной потоковой передачи данных между Apache Kafka и другими системами. Это упрощает быстрое определение коннекторов, которые перемещают большие коллекции данных в Kafka и из нее. Kafka Connect может принимать целые базы данных или собирать метрики со всех серверов приложений в разделы Kafka, делая данные доступными для потоковой обработки с низкой задержкой. Задание экспорта может доставлять данные из разделов Kafka во вторичное хранилище и системы запросов или в пакетные системы для автономного анализа. Поскольку источники разные, то для каждого создается свой раздел Kafka.

Коннектор, разработанный DataMountaineer, упрощает запись данных в базу данных Cassandra [7]. Соединитель преобразует значения из Kafka Connect SinkRecords в JSON, а затем асинхронно вставляет записи в Cassandra. Коннектор имеет возможность создавать защищенные соединения через SSL и исключения RetriableException при ошибке, указывающее коннектору повторно доставить сообщение. Выбор поля и управление разделом обрабатывается языком запросов Kafka Connect (KCQL).

Для реализации соединения Spark и Cassandra будем использовать библиотеку Spark Cassandra Connector [8]. Для передачи данных необходимо создать StreamingContext и DStream, который будет соединяться с узлом. Для настройки StreamingContext нужно использовать объект SparkConf, в котором указаны “контактный“ узел, имя и пароль пользователя Cassandra. “Контактным” узлом может быть любой узел кластера. Из него драйвер извлекает топологию кластера и пытается подключиться к ближайшему узлу в том же центре обработки данных. Всякий раз, когда вызывается любой метод, требующий доступа к Cassandra, опции объекта SparkConf будут использоваться для создания нового соединения или заимствования уже открытого из глобального кэша соединений. Все соединения кэшируются. Если вызвать два метода, которым требуется быстрый доступ к одному кластеру Cassandra последовательно или параллельно из разных потоков, они будут совместно использовать одно и то же логическое соединение, представленное базовым объектом кластера драйверов Java. В конце концов, когда все задачи, требующие связи с Cassandra, завершатся, подключение к кластеру вскоре будет закрыто. Spark Cassandra Connector может подключаться к нескольким кластерам Cassandra.

4. Корреляционный анализ

События потребляются приложением потоковой передачи Spark Streaming. Они имеют свое место в иерархии, которое определяется способом получения события и его особенностями (рисунок 1). То есть имеется три больших класса событий: полученные с помощью SNMP, полученные с помощью Syslog и остальные. Класс может подразделяться на подклассы в зависимости от общих признаков событий, те в свою очередь так же могут делиться и так далее.

Приложение Spark Streaming берет каждый полученный раздел Kafka и выполняет простую фильтрацию, преобразует данные в классы иерархии и сохраняет их, после чего выполняется корреляционный анализ.

Возникает необходимость составить правила, по которым он будет проводиться. Существует множество способов их организации. В разрабатываемой системе планируется использовать только два типа: шаблоны и запросы. Шаблон представляет собой набор правил с некоторыми

полями, в которые подставляются данные из события. Можно сказать, что запрос – это тот же шаблон, только он работает с потоками данных. Он позволяют: объединять, группировать, фильтровать, дедуплицировать, сортировать, прогонять события через скользящие окна. Окно сохраняет события с целью агрегации, объединения, поиска соответствия распознаваемым шаблонам, подзапросам. Оно определяет, какое подмножество событий следует сохранить. Например, окно данных хранит последние N событий, а временное окно – последние N секунд событий. И в шаблоне, и в запросах также может быть указана реакция, т.е. какие действия должна выполнить система при соответствии данных шаблону. Самая простая реакция – оповещение администратора о подозрительной деятельности, например по почте. Однако есть и более радикальные типа закрытия портов, выполнения каких-либо программ или алгоритмов (например, шейпинг при DDoS-атаках).

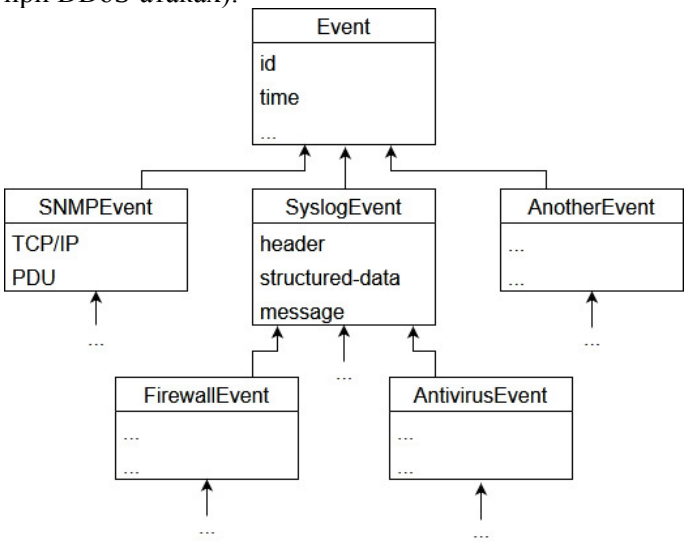

Рисунок 1. Иерархия событий.

Поля шаблонов могут быть разными, например, такими как категория события, IP-адрес источника событий, тип сообщившего устройства, номер порта, временные пороги и так далее. Выбор нужных для анализа полей в основном обусловлен доступными данными. Например, единица обмена SNMP включает в себя IP и UDP заголовки, версию протокола, пароль, тип единицы обмена, id запроса, статус, индекс ошибки и связанные переменные. А сообщение Syslog состоит из заголовка (header), структурированных данных (structured-data) и сообщения. Заголовок содержит информацию о приоритете, версии протокола, временной метке, имени хоста, названия приложения, id системного процесса и типе сообщения. Зная то, как формируются сообщения этих протоколов, можно вычленить нужные поля с помощью регулярных выражений. Для каждого типа событий используется специальное регулярное выражение, которое захватывает необходимые данные из сообщения источника событий. Следующий метод создает новое событие, в двойных кавычках указываются группы регулярного выражения.

```
create event ExampleEvent: Event {
severity:  " severitygroup",
datetime:  " datetimegroup",
hostname:  " hostnamegroup",
appname:  " appnamegroup",
type:  " typegroup",
message: " messagegroup",
//пользовательские поля
}
```

Кроме обычного метода создания событий будет еще несколько, предназначенных для определенного вида событий. Это сделано для удобства пользователя системы.

```
create event SyslogEvent: Event {
facility:  " facilitygroup",
severity:  " severitygroup",
datetime:  " datetimegroup",
hostname:  " hostnamegroup",
appname:  " appnamegroup",
type:  " typegroup",
message: " messagegroup"
}
```

```
create event SNMPEvent: Event {
destination_address:"destination_addressgroup",
source_address:  " source_addressgroup",
type_of_service:  " type_of_servicegroup",
sourse_port:  " sourse_portgroup",
destination_port:  " destination_portgroup",
pdu-type:  " pdu-typegroup",
error-status:  " error-statusgroup"
}
```

Сами правила внешне похожи на запросы SQL. Вначале пишется ключевое слово, обозначающее начало правила (select). Затем указывается выборка из какой группы событий нужна. По принципу конструкции SQL WHERE можно указать условия выборки. Поддерживаются логические операции (and, or, not). Также можно указать в течении какого времени события должны произойти, либо какая разница по времени между ними (timer). На этом этапе можно использовать группирование (groupby) или сортировку (orderby). Указывается реакция (then msg/block). В следующем примере производится сгруппированная по IP-адресу источника выборка времени оповещения об инциденте, IP-адреса хоста, степени тяжести и категории инцидента, полученных с помощью шаблона, кроме тех, у которых IP-адрес источника 192.168.100 или 192.168.1.101:

```
select alert_time, host_ip, severity, category from pattern
[pattern eventA=antivirus -> eventB=scanning_hosts (src_ip = host_ip) where timer within 60]
where src_ip not  “192.168.1.100” or  not “192.168.1.101”
groupby src_ip.
```

Шаблон указан в квадратных скобках. Он начинается с ключевого слова pattern, затем идут два события (eventA и eventB), через знак равенства указывается тип события, знак -> показывает, что события идут последовательно. Далее в скобках указаны поля этих событий. В примере поле второго события должно соответствовать свойству первого. Условие должно выполниться в течение 60 секунд (timer within 60), чтобы события были идентифицированы как инцидент. Также можно использовать сохраненные ранее шаблоны, для этого в квадратных скобках просто указывается его имя.

На основе подобных правил будет генерироваться особый код на Python, который выполнит анализ. В Web-интерфейсе системы есть возможность редактирования и добавления правил корреляции.

5. Концепция системы

Сначала необходимо собрать как можно больше данных о событиях безопасности в компьютерной сети. Как уже было написано выше, данные собираются с помощью протоколов Syslog и SNMP. Информацию о прочих событиях, которые они не охватывают, можно собирать специальными инструментами, например, с помощью IDS Suricata. Все данные передаются в формате JSON. Инструменты, используемые для транспортировки, поддерживают работу с этим форматом.

Итак, агенты SNMP собирают информацию на устройствах и передают их на сервер SNMP (рисунок 2). Причем существует два способа передачи: запрос-ответ и Trap. Во втором случае, агент в одностороннем порядке отправляет сообщения на сервер.

Данные, получаемые с помощью протокола Syslog, собираются серверами rsyslogd (рисунок 3). HAProxy используется для балансировки нагрузки и обеспечения доступности. Чтобы сервер rsyslog мог передавать данные на сервера Kafka необходимо установить специальный модуль вывода «omkafka» и изменить конфигурационный файл rsyslog.conf, добавив туда шаблон преобразования сообщений syslog в формат JSON [9].

Информация с источников, собирающаяся Suricata, с помощью Kafka поступает базу данных [10]. Для каждого типа источников событий на серверах Kafka должен быть свой раздел, который будет обрабатывать сообщения конкретного источника параллельно с остальными

разделами (в данном примере «sur-topic», «rsl-topic», «snmp-topic»). Все полученные “сырые” данные в формате JSON сохраняются в базе данных, откуда их забирает приложение Spark. Оно конвертирует, фильтрует и агрегирует события, после чего выполняется корреляционный анализ. Правила для корреляции берутся из отдельного хранилища. Все полученные результаты сохраняются в отдельную базу данных. В случае обнаружения инцидентов, отправляется сообщение в модуль реакции, который выполняет оповещение администратора. Web-интерфейс позволяет пользователям авторизоваться, сформировать и посмотреть отчет, а также редактировать и добавлять правила корреляции.

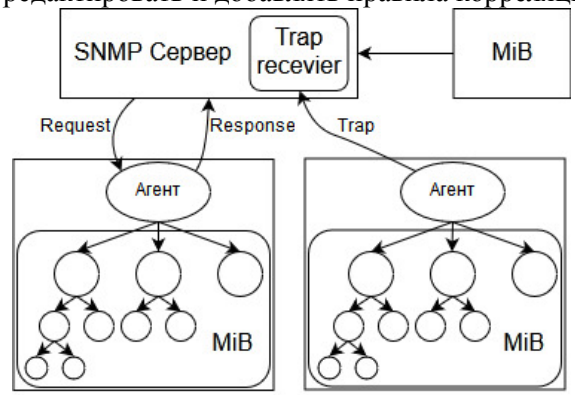

Рисунок 2. Взаимодействие менеджера и агентов SNMP.

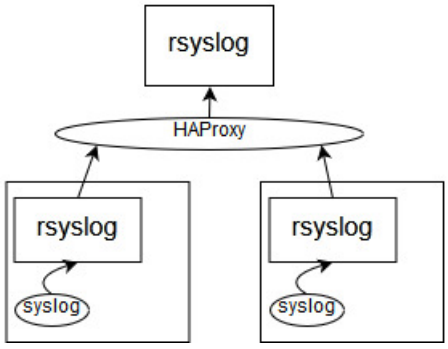

Рисунок 3. Сбор данных по протоколу Syslog с помощью серверов rsyslog.

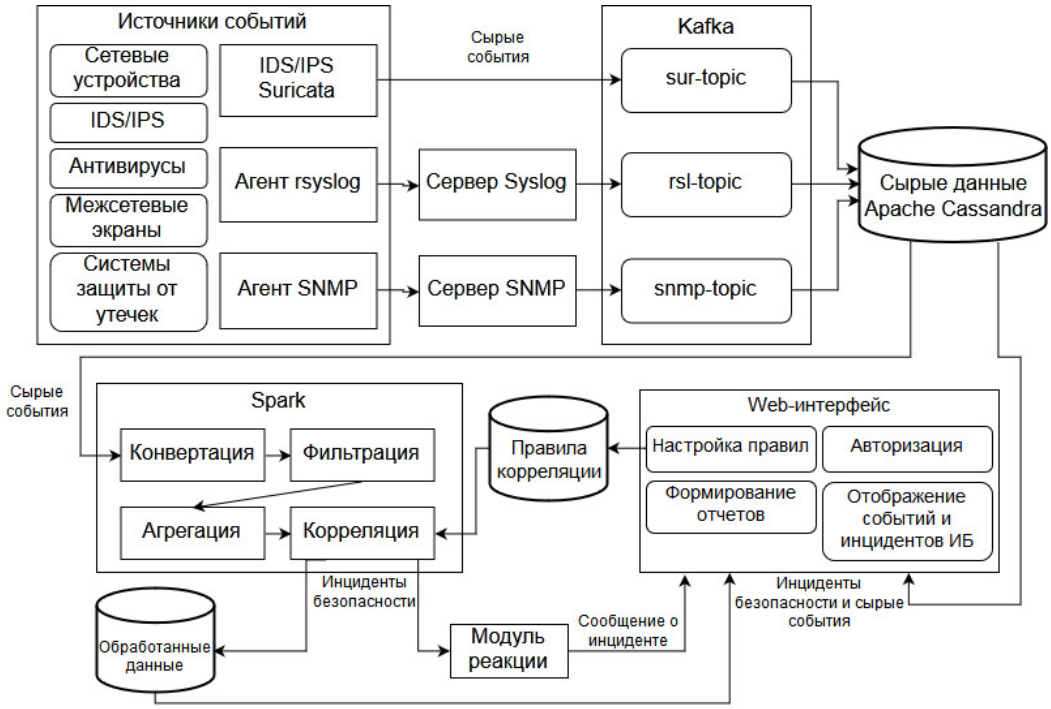

Рисунок 4. Концепция системы сбора, обобщения и корреляционного анализа событий безопасности.

6. Заключение

Таким образом, мы получаем дешевую распределенную систему сбора и обобщения событий и инцидентов безопасности, способную получать данные со всей сети из разных источников, обрабатывать их, проводить первичный анализ и выдавать результат в удобном формате. Вся система работает с потоковыми данными в режиме реального времени, что, несомненно,

является важным преимуществом. Подобное решение подходит для больших корпоративных сетей, а за счет высокой масштабируемости компонентов может работать и с большими данными (Big Data).

В будущем возможно расширение функционала системы за счет добавления поиска инцидентов с помощью нейронных сетей. А также подключение как можно большего числа источников событий и оптимизация системы. Библиотека Spark MLlib предоставляет высокоуровневое API, которое помогает создавать практические конвейеры машинного обучения.

7. Литература

[1] Jirsik, T. Toward real-time network-wide cyber situational awareness / T. Jirsik, P. Celeda // NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, 2018.

[2] Carvalho, V.S. OwlSight: platform for real-time detection and visualization of cyber threats / V.S. Carvalho, M.J. Polidoro, J.P. Magalhães // Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016. – P. 61-66.

[3] Marchal, S. A big data architecture for large scale security monitoring // IEEE International Congress on Big Data, 2014. – P. 56-63.

[4] Kafka 2.0 Documentation [Electronic resource]. – Access mode: http://kafka.apache.org/documentation/ (14.09.2018).

[5] Spark Overview [Electronic resource]. – Access mode: http://spark.apache.org/docs/latest/ (15.09.2018).

[6] Apache Cassandra Documentation v4.0 [Electronic resource]. – Access mode: http://cassandra.apache.org/doc/latest/ (15.09.2018).

[7] Kafka Connect Cassandra [Electronic resource]. – Access mode: https://www.confluent.io/connector/kafka-connect-cassandra/ (19.10.2018).

[8] Kindling: An Introduction to Spark with Cassandra (Part 1) [Electronic resource]. – Access mode: https://www.datastax.com/dev/blog/kindling-an-introduction-to-spark-with-cassandra-part-1 (22.10.2018).

[9] Rsyslog Documentation Electronic resource]. – Access mode: https://www.rsyslog.com/doc/v8-stable/ (25.09.2018).

[10] Suricata User Guide [Electronic resource] – Access mode: https://suricata.readthedocs.io/en/suricata-4.1.0/ (25.09.2018).

Благодарности

Исследование выполнено при финансовой поддержке Правительства Оренбургской области и РФФИ (проекты №18-47-560017, №18-07-01446 и №16-29-09639).

Security event data collection and analysis in large corporate networks

E.V. Chernova[1], P.N. Polezhaev[1], A.E. Shukhman[1], Yu.A. Ushakov[1], I. Bolodurina[1], N.Bakhareva[2]

[1]Orenburg State University, Pobedy av. 13, Orenburg, Russia, 460018
[2]Povolzhskiy State University of Telecommunications and Informatics, L. Tolstoy str. 23, Samara, Russia, 443010

Abstract. Every year computer networks become more complex, which directly affects the provision of a high level of information security. Different commercial services, critical systems and information resources prevailing in such networks are profitable targets for terrorists, cyber-spies and criminal elements. The consequences range from the theft of strategic, highly valued intellectual property and direct financial losses to significant damage to brand and customer trust. Attackers have the advantage in complex computer networks – it is easier to hide their tracks. The detection and identification of security incidents are one of the most important and difficult tasks. There is a need to detect security incidents as soon as possible, to analyze and respond to them correctly, so as not to complicate the work of the enterprise computer network. The difficulty is that different event sources offer different data formats or can duplicate events. Also, some events themselves do not indicate any problems, but their sequence may indicate the presence of a security incident. All collection processes must be performed in real time, which means streaming data processing.