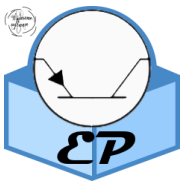


Централизованный сбор журналов событий Windows Operating System

EVENTS PROCESSOR

Оглавление:

Преамбула

Задача системы – централизованная сборка, хранения и обработка журналов событий (Event logs) операционной системы Microsoft Windows ™.

В своем «базовом» виде она содержит необходимый минимум (basic building block) для самостоятельного построения системы мониторинга интересующих вас событий в реальном времени с использованием языка запросов XPath (для выборки интересующих событий) и языка СУБД *Microsoft SQL Server T-SQL* (для последующей обработки событий).

Система состоит из 3х компонентов:

1. *Клиентского агента* (Events Processor emitter) реализованного в виде сервиса, который устанавливается на системы, с которых нужно обеспечить сборку событий. Он подписывается на события используя встроенные механизмы Eventlogs XPath Query language
2. *Центральной серверной компоненты* (Events Processor collector) реализованного в виде сервиса, который устанавливается на серверную ОС и обеспечивает приём и обработку всех событий с клиентских агентов
3. *Microsoft SQL Server ™* - используется для хранения и обработки дальнейшей событий с помощью процедур и функций на языке *T-SQL*

Вся система обеспечивает безопасную, надежную и высокоскоростную передачу событий журналов на центральный сервер для их консолидации, обработки и хранения.

Для получения данных используется язык построения запросов *Windows Eventlog XPath Query* Language. Поэтому это не потребует изучения дополнительных языков.

В случае невозможности быстрой доставки (к примеру сеть недоступна), клиентский агент складывает на диск в очередь данные и доставит сразу, как только появится такая возможность.

Серверная часть, при недоступности СУБД, продолжает сбор данных складывая их на диск в очередь. При появлении доступа к СУБД – они мгновенно добавятся туда и произойдет их дальнейшая обработка.

Языком для написания сценариев обработки данных выступает *Microsoft T-SQL*. Это обеспечивает быстроту разработки, модификации и поддержки сценариев.

Системные требования

Сервис не критичен к системным ресурсам и может функционировать на совсем маленьких ресурсах при необходимости. Естественно с увеличением нагрузки (количество данных, коэффициент одновременного использования) потребуется «добавить ресурсов».

- Операционная система >= Microsoft Windows 2012 Server R2
- CPU с двумя логическими ядрами
- 1Гб оперативной памяти
- СУБД >= Microsoft SQL Server 2012. Обьем БД десяток мегабайт на старте

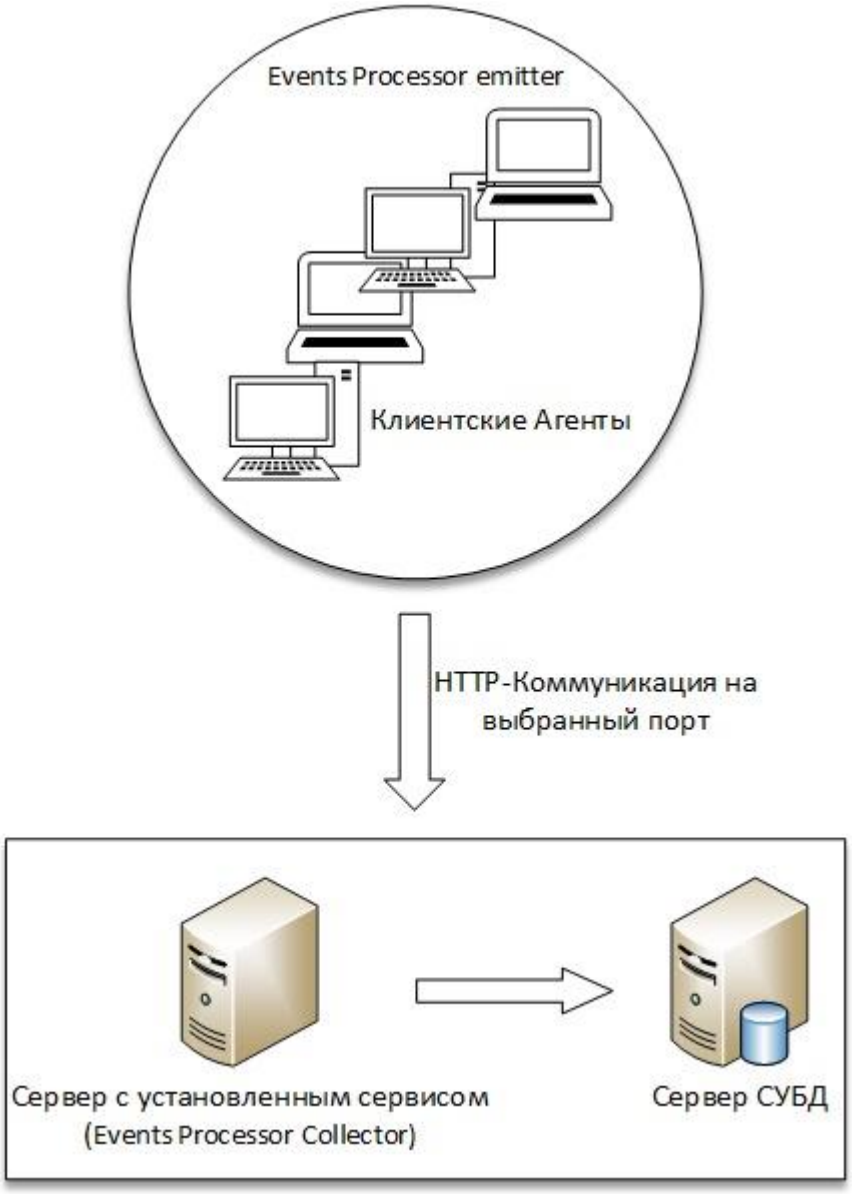
Клиентский агент тоже не требователен к ресурсам. Потребляет минимальное кол-во оперативной памяти и процессора.

- Операционная система >= Windows Vista (или её серверный аналог Windows Server 2008)
- CPU с одним логическим ядром. Тактовая частота не играет роли
- 16 мегабайт оперативной памяти под сам агент
- 6 мегабайт свободного места на диске

В зависимости от нагрузки (потока событий, кол-ва клиентских агентов онлайн) – серверной части может понадобится больше ресурсов.

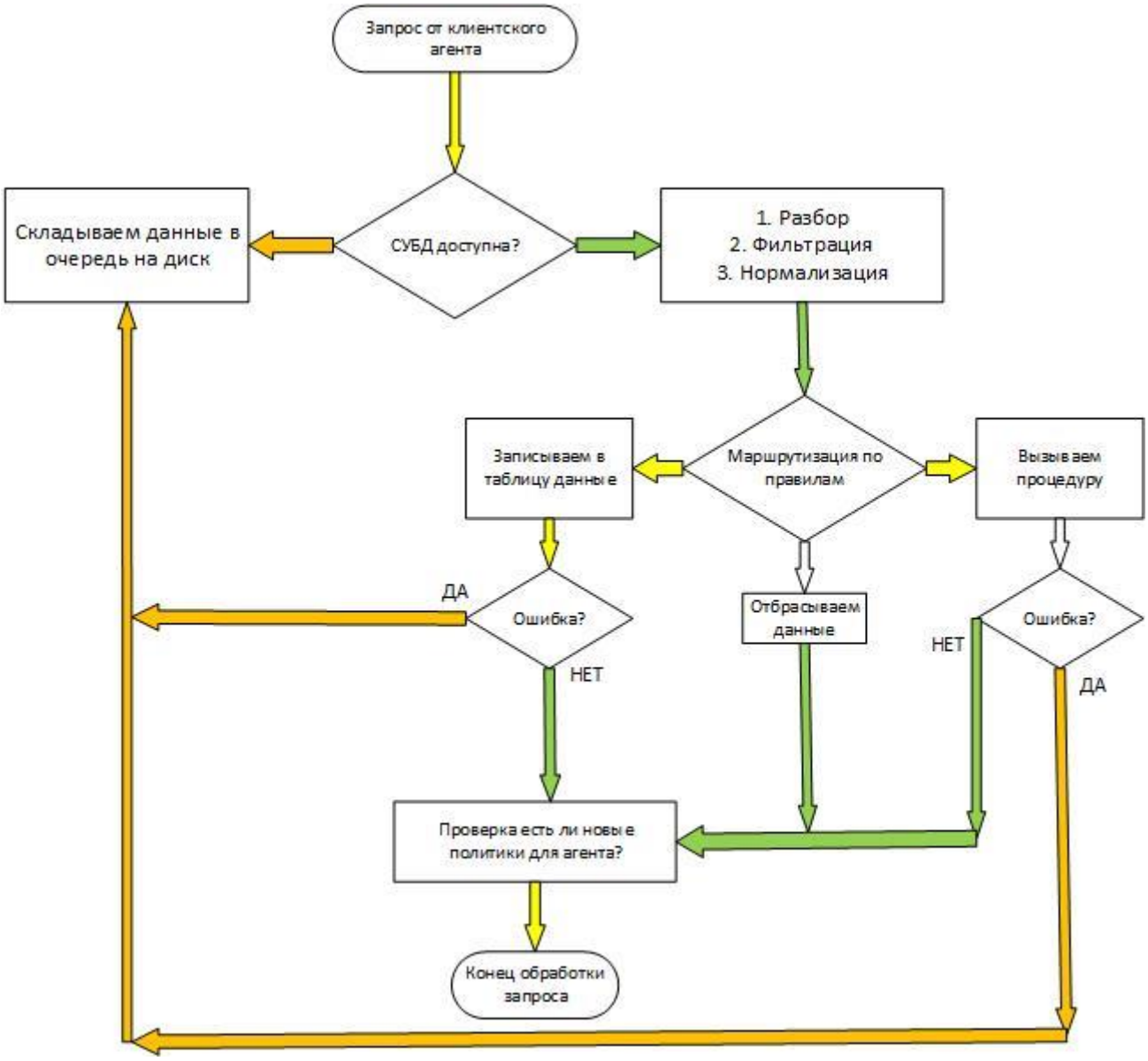
Общий принцип работы системы

На диаграмме ниже представлен общий принцип взавимодействия компонентов системы



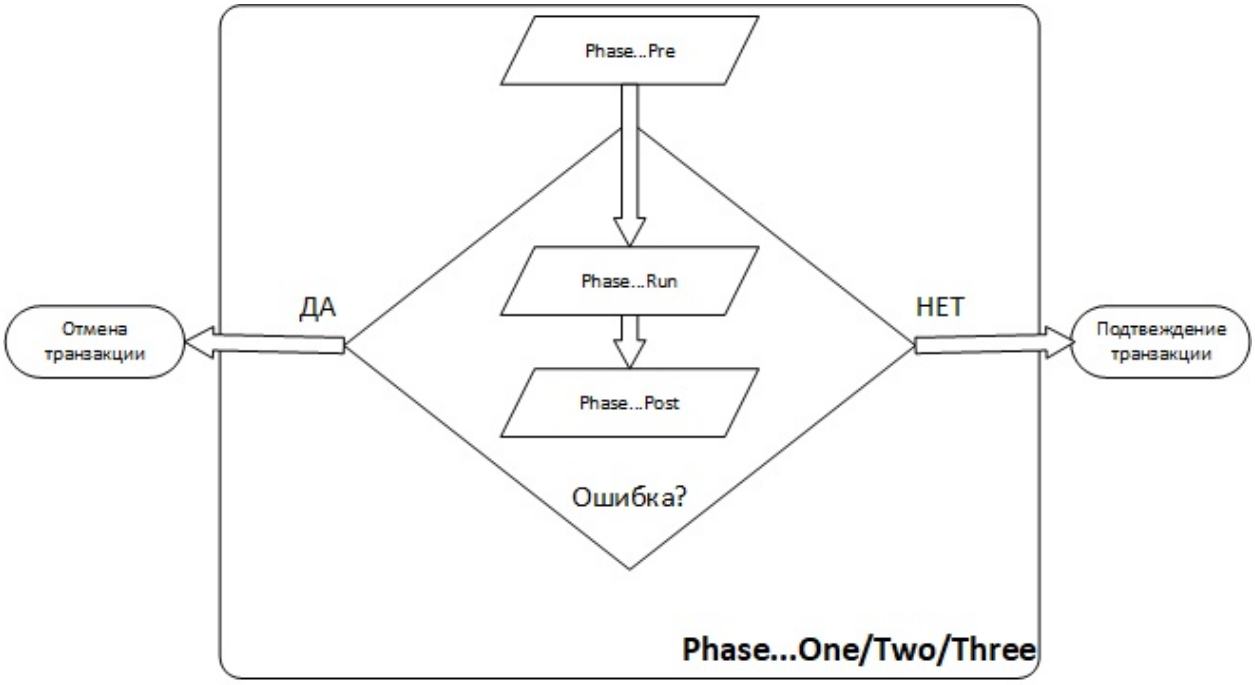
Процесс обработки принятых данных

На диаграмме ниже представлен процесс обработки входящих потоков данных



Последовательность фаз обработки

После настройки запуска 3х фаз обработки, сервер будет запускать их раз за разом через указанные интервалы времени. В каждой из фаз диаграмма исполнения будет выглядеть как на рисунке ниже



Вы можете отключить какую-либо из фаз обработки выставив значение параметров *phaseonesec* / *phasetwomin* / *phasethreedays* в 0.

Установка сервера

Для установки серверной части понадобится вначале установить Microsoft SQL Server ™.

При «первичной» установки сервиса подразумевается, что включен режим *mixed-security* и допустимо использовать локальные учетные записи SQL-сервера.

Для начала установки запустите *epserversetup.exe* и следуйте его указаниям на экранах.

В дальнейшем вы можете самостоятельно провести security hardening и запустить как сам сервис так и его доступ в СУБД под любой учетной записью.

При установке сервера будет установлена базовая консоль администрирования. Она состоит из двух файлов epui-settings.xml и *EP_fulladmin.exe* . Вы можете переносить их на любую другую машину где будете использовать.

Рекомендуется не сохранять пароли и использовать встроенную аутентификацию Windows SSO SSPI.

Серверная компонента присутствует в стандартном списке «*установка и удаление программ*» откуда его можно удалить используя административную учетную запись. Исключение – база данных и учетная запись в СУБД. Их нужно будет вручную удалять.

Установка клиента

Для установки клиентской части следует запустить инсталлятор *epagentsetup.exe*.
Поддерживаются «тихая установка» и полноценная с GUI-интерфейсом.
По-умолчанию имя сервера *evprocessor* порт *8999*.
Поддерживаются ключи командной строки
Epagentsetup.exe [/s] [/h servername] [/p port]:

- **/s** – запуск в «скрытом режиме»
- **/h** – указание имени сервера
- **/p** – указание порта сервера

Установщик поддерживает возврат *errorlevel* переменной:

0 – успешно, сервис стартовал
10 – неудача, ОС неподдерживается
11 – неудача, нет *Microsoft Visual C 2017 (32/64) redistributable* компонента

При запуске клиента он автоматически создаст скрытый каталог очереди на локальном диске в каталоге своей установки.

Клиент не ведет никаких журналов своей работы локально. Все события, ошибки и статусы он отсылает на сконфигурированный сервер.
Исключение из этого правила составляют совсем критические события. Пример невозможность создания/чтения файлов/каталога.

Клиент присутствует в стандартном списке «*установка и удаление программ*» откуда его можно удалить используя административную учетную запись.

Процесс запуска серверной части

В процессе запуска серверной компоненты происходит:

- Регистрация HTTP пространства имен servername:port/`EPEndpoint`
- Загрузка последнего состояния клиентов (для ускорения)
- Загрузка очереди всех необработанных данных в СУБД (если они есть)
- Исполнения триггеров обработки *PhaseOne* и *PhaseTwo* (если включены)

После окончания всей инициализации в лог-файл сервиса на диске будет записано сообщение `All fine. start processing network events`
Это свидетельствует об успешном старте сервиса и его полной работоспособности. Именно с этого момента начинается прием входящих данных от клиентских агентов.

При возникновении ошибок – они будут размещены в лог файле. Каждая фаза инициализации отражается в этом же файле.

Настройка сервера: захват событий

Для того чтобы события, передаваемые клиентским агентом, обрабатывались коллектором, нужно их описать правильным образом. Для этого используется конфигурационный файл *epweb.xml* находящийся в директории с установленным серверным компонентом *ep-server.exe*

```
<server sqldsn="" port="8999" phaseonesec="0" phasetwomin="5" phasethreedays="1"
workthreads="1" maxpostsize="10mb"  >

        <ldapcache>
          <domain domns="domain" query="LDAP://DC=domain,DC=com" />
          <domain domdns="another.dom" query="LDAP://DC=another,DC=domain"
        account="another.domain\user " passwd="hackme" />
        </ldapcache>

        <traps>
          <event provguid="{54849625-5478-4994-a5ba-3e3b0328c30d}" id="4624"
        target="rawlogons" fireproc="0" description="Success account logon" >
                  <map cols="SubjectLogonId" cold="srclogonid" type="10" />
                  <map cols="ProcessName" cold="procname" sizemax="1024" />
          </event>
        </traps>

</server>
```

Узел server.

Атрибут	Назначение
sqldsn	Указывает строку соединения с SQL Server
port	Порт HTTP который будет слушать сервер
maxpostsize	50Kb – 200Mb. Задает максимальный размер одного пакета принимаемых данных от клиентов и служит для ограничения жесткого со стороны сервера обьемов для недопущения исчерпания ресурсов. Параметр позволяет указывать как ввиде конкретного числа байт так и человеческом виде 5Mb, 100kb и т.п.
phaseonesec	10-300. *В секундах*. Указывает как часто должен происходить разбор «быстрых сырых данных». Это влечет за собой последовательный запуск SQL процедур: PhaseOnePre -> PhaseOneRun -> PhaseOnePost
phasetwomin	1-300. *В минутах*. Указывает как часто должен происходить разбор «длинных сырых данных». Это влечет за собой запуск последовательный запуск SQL процедур PhaseTwoPre -> PhaseTwoRun -> PhaseTwoPost. Помимо этого, если настроен (указан) запускается построение кэша с LDAP-каталогов указанных опционально в узле ldapcache
phasethreedays	1-7. *В днях*. Указывает как часто должен происходить «суммаризационный анализ» данных прошедших первых две фазы. Это влечет за собой запуск последовательный SQL процедур PhaseThreePre -> PhaseThreeRun -> PhaseThreePost
workthreads	1-255. опционально! Если не указано, то расчитывается по числу процессоров в системе. Указывает число рабочих потоков приема данных. На высоконагруженном коллекторе требуется увеличивать его ручным способом убеждаясь в отсутствии переполнения HTTP-очереди

Весь рабочий цикл сервиса состоит в запуске 3х стадий анализа данных (*PhaseOne* -> *PhaseTwo* -> *PhaseThree*) через сконфигурированные интервалы времени (секунды, минуты, дни).
В свою очередь каждая из них состоит из 3х SQL-процедур, вызываемых последовательно *PhaseOnePre* -> *PhaseOneRun* -> *PhaseOnePost*. Если какая-то из процедур будет завершена с ошибкой – её исполнение прервется. Если это Фаза *Pre* то сделанные изменения процедурой *HE* будут откачены назад (rollback). Откат изменений в случае ошибки предусмотрен для связки процедур *Run + Post*.

В *Pre*-фазу удобно помещать нормализационную и фильтрующую логику. Удаление не интересующих данных, наполнение словарей.
В *Run*-фазу удобно помещать непосредственный анализ данных.
В *Post*-фазу удобно размещать логику очистки обработанных данных.

Каждой из SQL-процедур передается параметр – штамп времени UTC в момент запуска стадии.

Если какая-либо из стадий будет выполняться дольше заданного интервала – в лог файл будет добавлено соответствующее оповещение:
`Phase... data analyzer is busy processing previous request!`
Очевидно, что это либо следствие слишком большого числа событий (перегрузка) или невозможность аппаратного обеспечения сервера обеспечить требуемую производительность (Диск, Процессор, ОЗУ). Конкретную причину следует устанавливать используя метрики производительности и мониторинг (Performance monitor counter и SQL Server counters).

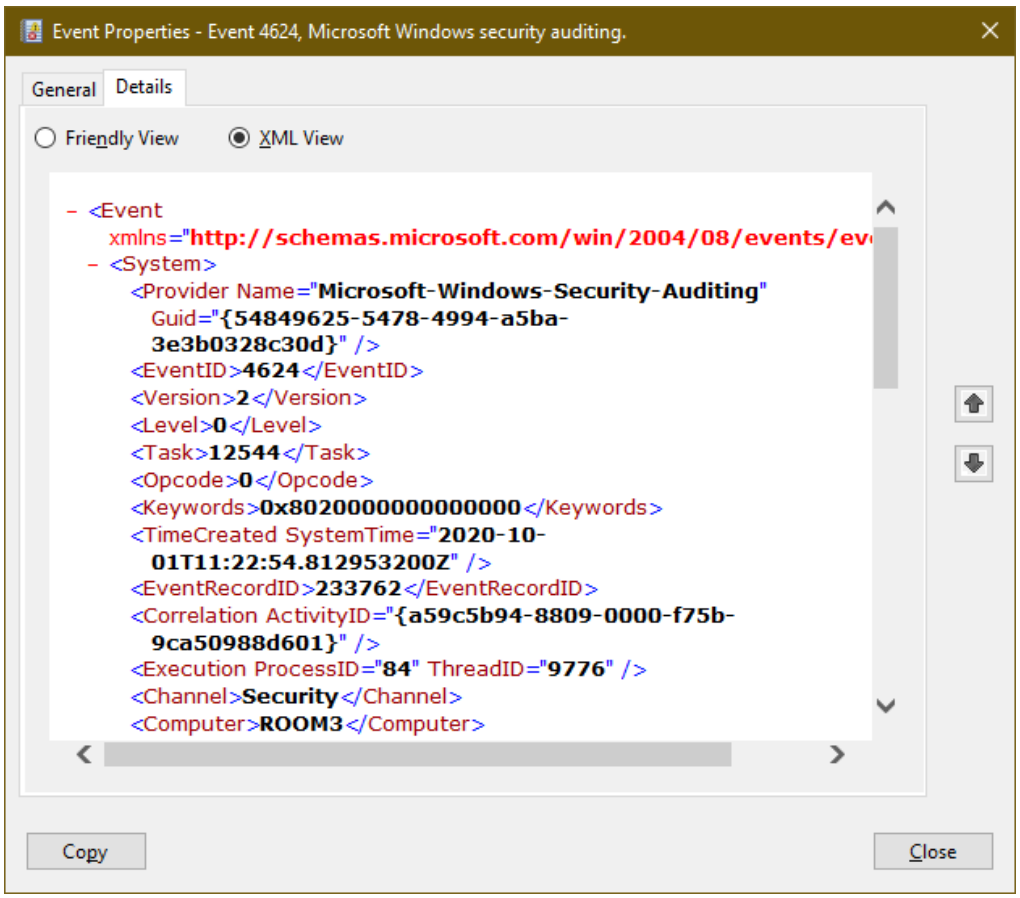
Узел ldapcache (опциональный).

Атрибут	Назначение
domnb	Опционально. NETBIOS имя домена
domdns	Опционально. DNS имя домена
query	LDAP-запрос конкретного OU либо всего домена. Без фильтров! Только путь
account	Опционально. Аккаунт под кем соединяться с LDAP-каталогом
passwd	Опционально. Пароль (если он требуется) от аккаунта

Включение этого узла (и его конфигурирование) включает механизм периодического кэширования состояния некоторых типов обьектов в Active Directory указанного домена. Кэш используется для обеспечения быстрой трансляции связи *SID<->Account+Domain* в приходящих событиях. Без его активации система будет не оптимально работать, потребуется некоторое время пока она соберет достаточно данных из приходящих событий для построения взаимосвязей. После окончания итоговая таблица `sidlookup` будет в консистентном состоянии.

Узел traps.

Для описания, нового события которое требуется обрабатывать следует воспользоваться Event Viewer встроенным в операционную систему (*eventvwr.msc*) и открыть интересующее вас событие в XML-виде. На рисунке ниже проиллюстрирован пример



Здесь можно посмотреть на интересующие графы события и записать их в правило обработки на серверной стороне (коллекторе). *Внимание: имена регистрозависимы!*

Узел event как раз описывает каждое событие, в отдельности которое требуется «поймать» в потоке.

Атрибут	Назначение
provguid	Соответствует значению Guid из скриншота выше
id	Соответствует значению EventID из скриншота выше
target	Имя таблички, либо процедуры которую вызывать при появлении данного события в потоке
description	*Опционально*. Нужно только для комментария человеку чтобы не потеряться в конфиге. Не используется никак
fireproc	Флаг булев указывающий вызывать ли на это событие SQL процедуру, указанную в target или просто вставлять данные в табличку с именем target

Дальнейший подузел maps описывает конкретные поля исходного события: EventData/Data Name.

При нахождении соответствия в исходном событии атрибута Name равному указанному в cols значению происходит «захват» значения данного поля в имя, указанное cold. *Если у события стоит флаг вызова SQL процедуры, то атрибут cold не требуется.*
Опциональный атрибут sizemax у текстовых данных указывает максимальный размер поля. Если длинна текста будет больше указанного числа – произойдет «обрезание».

Процедура сопоставления полей в исходном событии и в SQL БД может конвертировать сразу некоторые типы входящих данных (так эффективнее складировать и обрабатывать чем просто строковые данные).

У узла map для этой цели есть опциональный атрибут type. Его можно и не указывать (он опционален). Возможные значения в таблице ниже

Значение	Описание	Тип данных уходящий в SQL Server
1, 2	Преобразуется к типу текст (*по-умолчанию*)	Varchar
3	Пример: 125, 230, 4	Tinyint
4	Пример: 32768, 6, 1	Smallint
5	Пример: 3521819	Int
6	Пример: 613521819	Bigint
7	Преобразуется строковой вид IPv4/IPv6 к бинарному	Varbinary(16)
8	Пример: 0xAA, 0x9	Tinyint
9	Пример: 0x7E4D	Smallint
10	Пример: 0xAABBCCDD	Int
11	Пример: 0x88441122AADDEEFF	BigInt
12	Пример 0xAABBCCDD, AABBCCDD, AA BB CC	Varbinary

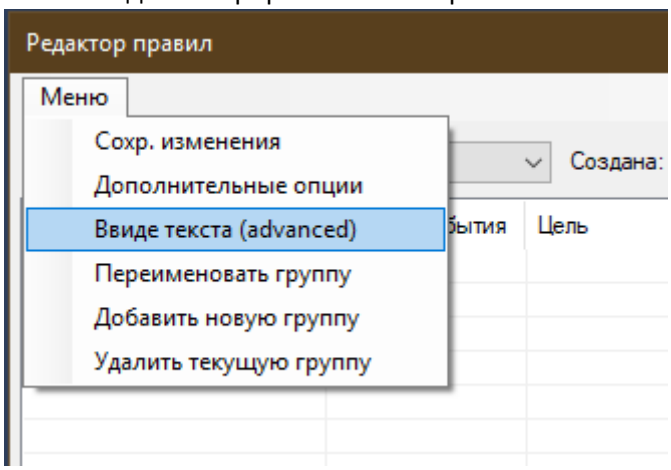
В ходе своей работы серверная компонента пишет собственный лог в подкаталог *Logs* каталога где установлен. Ежемесячно создается новый файл лога.

При возникновении ошибок взаимодействия с SQL-сервером, принятые данные кэшируются в скрытый подкаталог *Queue* где установлен сервис. Периодически каталог проверяется и данные оттуда повторно вливаются в SQL.

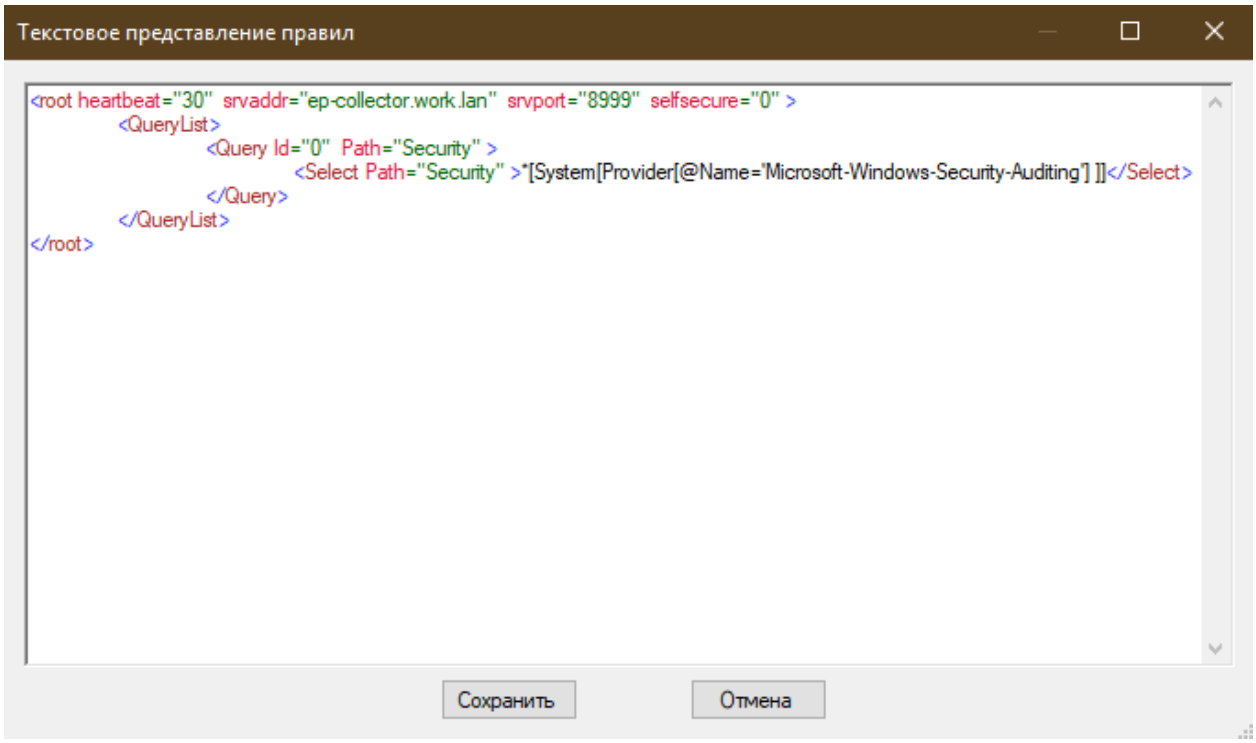
Помимо этого сама операционная система пишет лог ошибок HTTP API который располагается в системном подкаталоге \WINDOWS\System32\LogFiles\HTTPERR\ .
Смотрите за местом потребляемым этими журналами, возможно потребуется проводить их периодическую очистку!

Настройка клиента: сбор событий журналов

Для формирования клиентского запроса на получение событий от журнала ОС следует запустить консоль администрирования и выбрать в главном меню пункт «Редактор правил»



В появившемся окне выбираем необходимую группу (по-умолчанию это *default*). Далее выбираем пункт меню «в виде текста (advanced)». Откроется окно, содержащее все настройки конфигурации клиента



В нём нас интересует узел QueryList.

Язык для формирования запросов *Windows eventlog query* подробно описывается в официальных статьях Microsoft и выходит за рамки этого краткого руководства. См.:

- Описание схемы узлов, которые могут присутствовать в событии: Event Schema Elements https://learn.microsoft.com/en-us/windows/win32/wes/eventschema-elements

- Описание схемы узлов, описывающих запросы: Query Schema Elements

https://learn.microsoft.com/en-us/windows/win32/wes/queryschema-elements

- Примеры составления XPath запросов: Consuming Events (Windows Event Log)

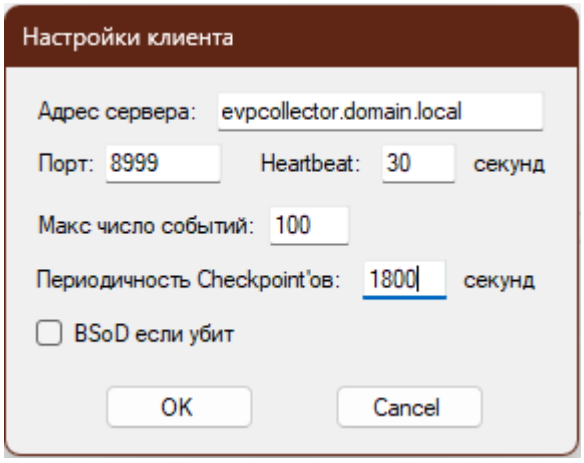
https://learn.microsoft.com/en-us/windows/win32/wes/consuming-events?redirectedfrom=MSDN#limitations

https://learn.microsoft.com/en-us/previous-versions/bb399427(v=vs.90)

Кратко суммаризируя: запрос состоит из единственного узла <QueryList> в котором располагается один или множество узлов <Query> (с обязательным указанием инкрементирующегося атрибута Id=”x”). Каждый из них должен содержать один или множество узлов <Select> описывающих интересующий журнал (атрибут Path) и сам запрос полей. Если из журнала надо передавать все события, то ставится просто звездочка (*).
Пример запроса, собирающего абсолютно все события в журнале Security (Безопасность):

```
<QueryList>
        <Query Id="0">
                <Select Path="Security">*</Select>
        </Query>
</QueryList>
```

Дополнительно клиент имеет опции позволяющие более точно настроить некоторые аспекты его функционирования. Настроить их можно выбрав в *меню* пункт «*Дополнительные опции*»



Здесь можно указать хост и порт для связи с сервером.

Помимо этого настроить интервал пульса (30 - 1800).

Так-же указать максимально агрегируемое число событий в одном пакете данных (10 - 1000).

Коммуникация с сервером будет происходит по любому из двух тригеров (пульс либо число событий).

И последний параметр - интервал контрольных точек (подробнее далее).

Работа с курсорами чтения журналов

Сервис имеет механизм работы с позициями журналов из которых получает данные. В случае остановки клиентского агента он сохраняет текущую позицию чтения всех настроеных журналов на локальном диске и на центральном сервере. Так-же есть интервал периодического (snapshot) сохранения данной позиции в ходе работы сервиса. Это настраивается параметром клиентского агента `bmcpsec=”seconds”` . Величина указывается в секундах и находится в интервале от *1800* до *86400* секунд. По-умолчанию она равна *36000*. Этот механизм нужен т.к. в случае резкого повисания ОС, проблем с железом или банального отключения электричества, журналы событий начнут вычитываться с самого начала. Для предотвращения этого служит данный механизм.

При своем старте клиентский агент пытается загрузить текущие позиции чтения журналов с диска и в случае отсутствия их – загружает с сервера.

В случае обнаружения неожиданных изменений значения последовательности курсоров – сервер будет оповещен и в журнале работы появится соответствующая запись. Это может быть индикатором злонамеренного действия и нуждается в дальнейшем расследовании.

Пример

В изначальной установке сервиса содержится пример захвата всех событий журнала Security (правило) клиентским агентом. А с серверной стороны обработка двух типов событий – вход в систему и выход из неё. Результатом работы станет таблица в СУБД *logons* с двумя событиями.