Санкт-Петербургское государственное бюджетное
профессиональное образовательное учреждение
«Академия управления городской средой, градостроительства и печати»

УТВЕРЖДАЮ
Заместитель директора
по учебно-производственной работе
_____ О.В. Фомичева
«26» декабря 2023 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
по выполнению практических работ
по МДК.05.03 Основы форензики
ПМ.05 УПРАВЛЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТЬЮ

для специальности
10.02.05 Обеспечение информационной безопасности автоматизированных систем

Санкт-Петербург
2023 г.

Методические рекомендации рассмотрены на заседании методического совета СПб ГБПОУ «АУГСГиП»

Протокол № 2 от «29» ноября 2023 г.

Методические рекомендации одобрены на заседании цикловой комиссии общетехнических дисциплин и компьютерных технологий

Протокол № 4 от «21» ноября 2023 г.

Председатель цикловой комиссии: Караченцева М.С. _____

Разработчики: преподаватели СПб ГБПОУ «АУГСГиП»

С О Д Е Р Ж А Н И Е

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Рабочая тетрадь для выполнения практических работ предназначена для организации работы на практических занятиях по МДК 05.03 Основы форензики, которая является важной составной частью в системе подготовки специалистов среднего профессионального образования по специальности 10.02.05 «Обеспечение информационной безопасности автоматизированных систем».

Практические занятия являются неотъемлемым этапом изучения МДК и проводятся с целью:

− формирования практических умений в соответствии с требованиями к уровню подготовки обучающихся, установленными рабочей программой учебной дисциплины;

− обобщения, систематизации, углубления, закрепления полученных теоретических знаний;

− готовности использовать теоретические знания на практике.

Практические занятия по МДК 05.03 «Основы форензики» способствуют формированию в дальнейшем при изучении профессиональных модулей, следующих общих и профессиональных компетенций:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей.

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержание необходимого уровня физической подготовленности.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.

ОК 11. Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере.

ПК 5.5. Принимать участие в управлении проектами.

В Рабочей тетради предлагаются к выполнению практические работы, предусмотренные учебной рабочей программой МДК 05.03 Основы форензики.

При разработке содержания практических работ учитывался уровень сложности освоения студентами соответствующей темы, общих и профессиональных компетенций, на формирование которых направлена дисциплина.

Выполнение практических работ в рамках учебной дисциплины позволяет освоить комплекс работ по выполнению практических заданий по всем темам МДК 05.03 «Основы форензики»

Рабочая тетрадь по учебной дисциплине МДК 05.03 «Основы форензики» имеет практическую направленность и значимость. Формируемые в процессе практических занятий умения могут быть использованы студентами в будущей профессиональной деятельности.

Рабочая тетрадь предназначена для студентов, изучающих МДК 05.03 «Основы форензики».

Оценки за выполнение практических работ выставляются по пятибалльной системе. Оценки за практические работы являются обязательными текущими оценками по учебной дисциплине и выставляются в журнале теоретического обучения.

1 ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ ПО ТЕМАМ МДК 05.03 «ОСНОВЫ ФОРЕНЗИКИ»

№ раздела, темы	Формируемые ОК и ПК	Тема практического занятия	Кол-во часов
Тема 3.1. Основы компьютерной криминалистики	ПК 5.3 ОК 1-11	Практическое занятие 1 Сбор артефактов ОС	2
Тема 3.2. Объекты компьютерной криминалистики	ПК 5.3 ОК 1-11	Практическое занятие 2 Настройка аудита в Windows	2
		Практическое занятие 3 Журналирование в Linux	2
Тема 3.3 Программное и аппаратное обеспечение экспертных исследований	ПК 5.3 ОК 1-11	Практическое занятие 4 Установка средств анализа сетевого трафика	2
		Практическое занятие 5 Установка Сканер ВС	2
		Практическое занятие 6 Установка LogonTracer	2
		Практическое занятие 7 Настройка Skadi	2
		Практическое занятие 8 Установка analyzeMFT	2
Тема 3.4. Компьютерная форензика	ПК 5.3 ОК 1-11	Практическое занятие 9 Сбор логов в ОС Windows	2
		Практическое занятие 10 Проверка файлов реестра	2
		Практическое занятие 11 Сбор логов в Linux	2
		Практическое занятие 12 Обнаружение инцидентов в ОС	2
		Практическое занятие 13 Проведение расследования инцидента	4
Тема 3.5 Сетевая форензика	ПК 5.3 ОК 1-11	Практическое занятие 14 Анализ сетевого трафика	2
		Практическое занятие 15 Обнаружение действий нарушителя	2
		Практическое занятие 16 Поиск подключений по SSH	2
		Практическое занятие 17 Настройка Zabbix для сбора данных	2
		Практическое занятие 18 Проведение расследования инцидента	4

2 ОПИСАНИЕ ПОРЯДКА ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

Практическая работа № 1 Сбор артефактов ОС

Задание:

Известно о произошедшей атаке на инфраструктуру компании. Используя предложенные средства извлеките артефакты, которые помогут в дальнейшем расследовании.

Используемые средства:

1. R-Studio — утилита для восстановления данных.
2. FTK Imager — инструмент для анализа и получения образов диска.
3. Утилиты NirSoft для Windows.
4. Утилиты Eric Zimmerman для анализа артефактов Windows.

Зафиксируйте выполнение скриншотами.

Практическая работа № 2 Настройка аудита в Windows

Задание:

Настройте журналы аудита в ОС Windows, которые помогут в дальнейшем контролировать процессы в ОС и расследовать инциденты.

Настраиваемые журналы:

1. Журнал входа в систему.
2. Журнал выхода из системы.
3. Аудит входа в систему
4. Аудит изменения политики
5. Аудит системных событий
6. Аудит событий входа в систему
7. Аудит управления учетными записями

Чтобы настроить политики аудита на устройстве:

1. Откройте окно Выполнить, нажав комбинацию клавиш Win+R.
2. В открывшемся окне введите запрос secpol.msc и нажмите ОК.
3. Откроется окно Локальная политика безопасности.
4. Перейдите в раздел Параметры безопасности → Локальные политики → Политика аудита.
5. В панели справа двойным щелчком мыши откройте свойства политики, для которой вы хотите включить аудит успешных и неуспешных попыток.
6. В окне Свойства <Имя политики> на вкладке Параметр локальной безопасности установите флажки Успех и Отказ, чтобы отслеживать успешные и прерванные попытки.

Зафиксируйте выполнение скриншотами.

Практическая работа № 3 Журналирование в Linux

Задание:

Настройте основные возможности службы управления журналами в ОС Астра Линукс, которые помогут в дальнейшем контролировать процессы в ОС и расследовать инциденты.

Настраиваемые службы:

1. yslog-ng - служба управления журналами syslog-ng по умолчанию используется в Astra Linux Special Edition x.7, доступна в репозиториях более ранних очередных обновлений;
2. rsyslog - служба управления журналами rsyslog по умолчанию используется в Astra Linux Common Edition и Astra Linux Special Edition ранее очередного обновления x.7, доступна в основном репозитории Astra Linux Special Edition x.7;
3. syslog (sysklogd) - служба управления журналами syslog в настоящее время в Astra Linux не используется;
4. logrotate - служба архивирования и очистки (ротации) журналов.

Зафиксируйте выполнение скриншотами.

Практическая работа № 4 Установка средств анализа сетевого трафика

Задание:

Установите и настройте средство анализа сетевого трафика Wireshark.
Проведите пробное сканирование сетевого трафи ка.

Зафиксируйте выполнение этапов скриншотами.

Ссылка для скачивания: https://www.wireshark.org/download.html

Практическая работа № 5 Установка Сканер ВС

Установите и настройте средство анализа Сканер ВС
Проведите пробное сканирование и дайте описание полученного результата.

Зафиксируйте выполнение этапов скриншотами.

Практическая работа № 6 Установка LogonTracer

Установите и настройте LogonTracer.
Проведите пробное сканирование.

Зафиксируйте выполнение этапов скриншотами.

Ссылка для скачивания: https://github.com/JPCERTCC/LogonTracer/wiki/for-linux

Практическая работа № 7 Настройка Skadi

Установите и настройте средство анализа Skadi.
Проведите пробное сканирование.

Зафиксируйте выполнение этапов скриншотами.

Ссылка для скачивания: https://github.com/orlikoski/Skadi

Практическая работа № 8 Установка analyzeMFT

Установите и настройте средство analyzeMFT.
Проведите пробное сканирование.

Зафиксируйте выполнение этапов скриншотами.

Ссылка для скачивания: https://github.com/rowingdude/analyzeMFT

Практическая работа № 9 Сбор логов в ОС Windows

Задание:

Воспользуйтесь официальной бесплатной утилитой Sysmon из пакета Microsoft Windows Sysinternals, которая существенно расширяет и дополняет возможности мониторинга ОС. Данная утилита дает возможность проводить аудит создания файлов, ключей реестра, процессов и потоков, а также осуществлять мониторинг загрузки драйверов и библиотек, сетевых подключений, WMI-событий и именованных каналов. Из особо полезных функций отметим возможность утилиты показывать родительский процесс и командную строку процесса, отображать значение хэш-сумм при событиях создания процесса и загрузки драйверов и библиотек с указанием наличия и действительности цифровой подписи. Несложным путем можно автоматизировать сравнение полученных хэш-сумм с индикаторами компрометации (IoCs, Indicator of Compromise) из данных фидов CyberThreat Intelligence, а также использовать приложение QVTI для IBM QRadar, с помощью которого хэши запускаемых файлов автоматически проверяются через сервис VirusTotal. Еще одной приятной опцией является возможность создания XML-конфигураций, в которых можно предельно четко указать объекты контроля и настройки работы Sysmon. Одними из наиболее продвинутых и детальных вариантов XML-конфигураций, с нашей точки зрения, являются конфиги https://github.com/ion-storm/sysmon-config и https://github.com/SwiftOnSecurity/sysmon-config .
Установка Sysmon предельно проста и также может быть легко автоматизирована:
1. Дистрибутив скачивается с https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
Все исполняемые файлы подписаны.

2. Создается или скачивается по приведенным выше ссылкам xml-файл с конфигурацией Sysmon.
3. Установка sysmon для x64 производится командой:
C:\folder\sysmon64.exe -accepteula -i C:\folder\sysmonconfig-export.xml , где sysmonconfig-export.xml – файл конфигурации, sysmon64.exe – файл-установщик.
Поддерживается запуск установки из сетевой папки.
4. После установки создается журнал Microsoft-Windows-Sysmon/Operational , размер которого мы сразу рекомендуем увеличить как минимум до 100 Мб.
Перезапуск устройства не требуется, Sysmon работает в виде сервиса, его исполняемый файл находится в C:\Windows\sysmon64.exe . По нашим подсчетам, footprint на конечной системе даже при использовании максимально детального конфига Sysmon не превышает 5-10% ЦПУ и около 100 Мб ОЗУ.
Осуществите:
1. Поиск по имени учетной записи в журнале Security - возьмем для примера имя Username:

```
<QueryList>
<Query Id="0" Path="Security">
<Select Path="Security">*[EventData[Data[@Name='TargetUserName']='Username']]
</Select>
</Query>
</QueryList>
```

2. Поиск по значению конкретного свойства события в журнале Sysmon - возьмем для примера поиск событий, в которых фигурировал целевой порт 443:

```
<QueryList>
  <Query Id="0" Path="Microsoft-Windows-Sysmon/Operational">
    <Select Path="Microsoft-Windows-Sysmon/Operational">*[EventData[Data[@Name='DestinationPort'] =
'443']]</Select>
  </Query>
</QueryList>
```

3. Произведем поиск сразу по двум условиям - возьмем для примера событие входа с EventID=4624 и имя пользователя Username:

```
<QueryList>
  <Query Id="0" Path="Security">
    <Select Path="Security">
*[System[(EventID=4624)]]
and
*[EventData[Data[@Name='TargetUserName']='Username']]
</Select>
  </Query>
</QueryList>
```

4. Поиск по трем условиям - дополнительно укажем Logon Type = 2, что соответствует интерактивному входу в ОС:

```
<QueryList>
  <Query Id="0" Path="Security">
    <Select Path="Security">
*[System[(EventID=4624)]]
and
*[EventData[Data[@Name='TargetUserName']='Username']]
and
*[EventData[Data[@Name='LogonType']='2']]
</Select>
  </Query>
</QueryList>
```

5. Рассмотрим функционал исключения из выборки данных по определенным критериям - это осуществляется указанием оператора Suppress с условиями исключения. В данном примере мы исключим из результатов поиска по фактам успешного входа (EventID=4624) все события, которые имеют отношения к системным учетным записям (SID S-1-5-18/19/20) с нерелевантным для нас типам входа (Logon Type = 4/5), а также применим функционал задания условий поиска с логическим оператором «ИЛИ», указав не интересующие нас имя процесса входа (Advapi) и методы аутентификации (Negotiate и NTLM):

```
<QueryList>
  <Query Id="0" Path="Security">
```

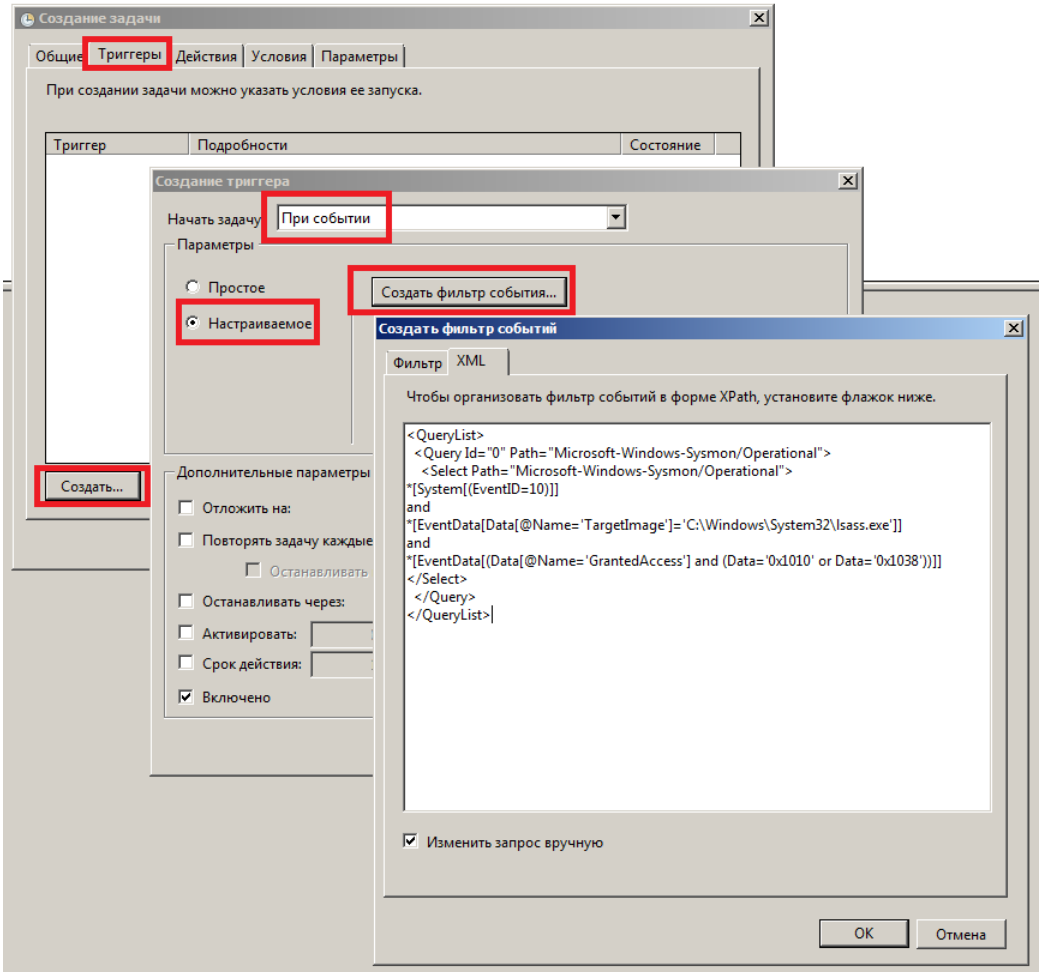
<Select Path="Security">*[System[(EventID=4624)]]</Select>
<Suppress Path="Security">*[EventData[(Data[@Name='TargetUserSid'] and (Data='S-1-5-18' or Data='S-1-5-19' or Data='S-1-5-20') and Data[@Name='LogonType'] and (Data='4' or Data='5'))]]
or
*[EventData[(Data[@Name='LogonProcessName'] and (Data='Advapi') and Data[@Name='AuthenticationPackageName'] and (Data='Negotiate' or Data='NTLM'))]]
</Suppress>
</Query>
</QueryList>

IRP-система штатными средствами Windows

Как мы увидели, встроенный функционал подсистемы журналирования Windows позволяет весьма гибко осуществлять поиск по зафиксированным событиям аудита ИБ, комбинируя различные условия поиска. Однако, у Windows есть еще одна интересная «фишка», которая позволяет использовать сформированные описанным выше образом правила поиска событий - мы говорим про создание задач с определенным триггером в «Планировщике заданий» Windows, что также является штатным функционалом ОС.

Как мы знаем, задачи в ОС Windows могут выполнять совершенно разные функции, от запуска диагностических и системных утилит до обновления компонент прикладного ПО. В задаче можно не только указать исполняемый файл, который будет запущен при наступлении определенных условий и триггеров, но и задать пользовательский PowerShell/VBS/Batch-скрипт, который также будет передан на обработку.

В контексте применения подсистемы журналирования интерес для нас представляет функционал гибкой настройки триггеров выполнения задач. Открыв «Планировщик заданий» (*taskschd.msc*), мы можем создать новую задачу, в свойствах которой на вкладке «Триггеры» мы увидим возможность создать свой триггер. При нажатии на кнопку «Создать» откроется новое окно, в котором в drop-down списке следует выбрать вариант «При событии», а в открывшейся форме отображения установить radio-button «Настраиваемое». После этих действий появится кнопка «Создать фильтр события», нажав на которую, мы увидим знакомое меню фильтрации событий, на вкладке XML в котором мы сможем задать произвольное поисковое условие в синтаксисе XPath-запроса.



Например, если мы хотим выполнять некоторую команду или скрипт при каждом интерактивном входе в систему пользователя Username, мы можем задать в качестве триггера задачи следующее поисковое выражение, уже знакомое нам по примеру выше:

```
<QueryList>
  <Query Id="0" Path="Security">
    <Select Path="Security">
*[System[(EventID=4624)]]
and
*[EventData[Data[@Name='TargetUserName']='Username']]
and
*[EventData[Data[@Name='LogonType']='2']]
</Select>
  </Query>
</QueryList>
```

Другой пример: оповещение администратора при подозрительном обращении к системному процессу *lsass.exe*, который хранит в своей памяти NTLM-хэши и Керберос-билеты пользователей Windows, что может говорить об использовании утилиты Mimikatz или аналогичных ей:

```
<QueryList>
  <Query Id="0" Path="Microsoft-Windows-Sysmon/Operational">
    <Select Path="Microsoft-Windows-Sysmon/Operational">
*[System[(EventID=10)]]
and
*[EventData[Data[@Name='TargetImage']='C:\Windows\System32\lsass.exe']]
and
*[EventData[(Data[@Name='GrantedAccess'] and (Data='0x1010' or Data='0x1038'))]]
</Select>
  </Query>
</QueryList>
```

Практическая работа № 10 Проверка файлов реестра

Проверка целостности системных файлов Windows 10 может пригодиться в том случае, если у вас есть основания полагать, что такие файлы были повреждены или же возникли подозрения о том, что какая-либо программа могла изменить системные файлы операционной системы.

В Windows 10 присутствует два инструмента для проверки целостности защищенных системных файлов и их автоматического восстановления при обнаружении повреждений — SFC.exe и DISM.exe, а также команда Repair-WindowsImage для Windows PowerShell (использующая DISM для работы). Вторая утилита служит дополнением первой, в случае, если SFC не удается восстановить поврежденные файлы.

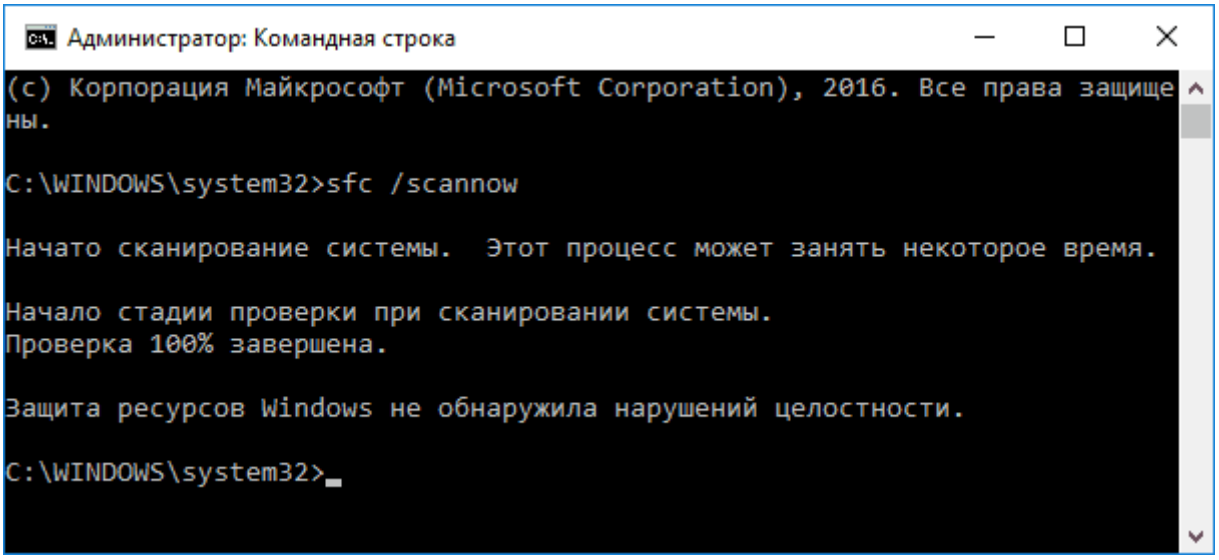
Примечание: описываемые в инструкции действия безопасны, однако, в том случае, если до этого вы проделывали какие-либо операции, связанные с заменой или изменением системных файлов (например, для возможности установки сторонних тем и т.п.), в результате восстановления системных файлов, эти изменения будут отменены.

Использование SFC для проверки целостности и исправления системных файлов Windows 10

Многим пользователям знакома команда проверки целостности системных файлов sfc /scannow которая автоматически проверяет и исправляет защищенные системные файлы Windows 10.

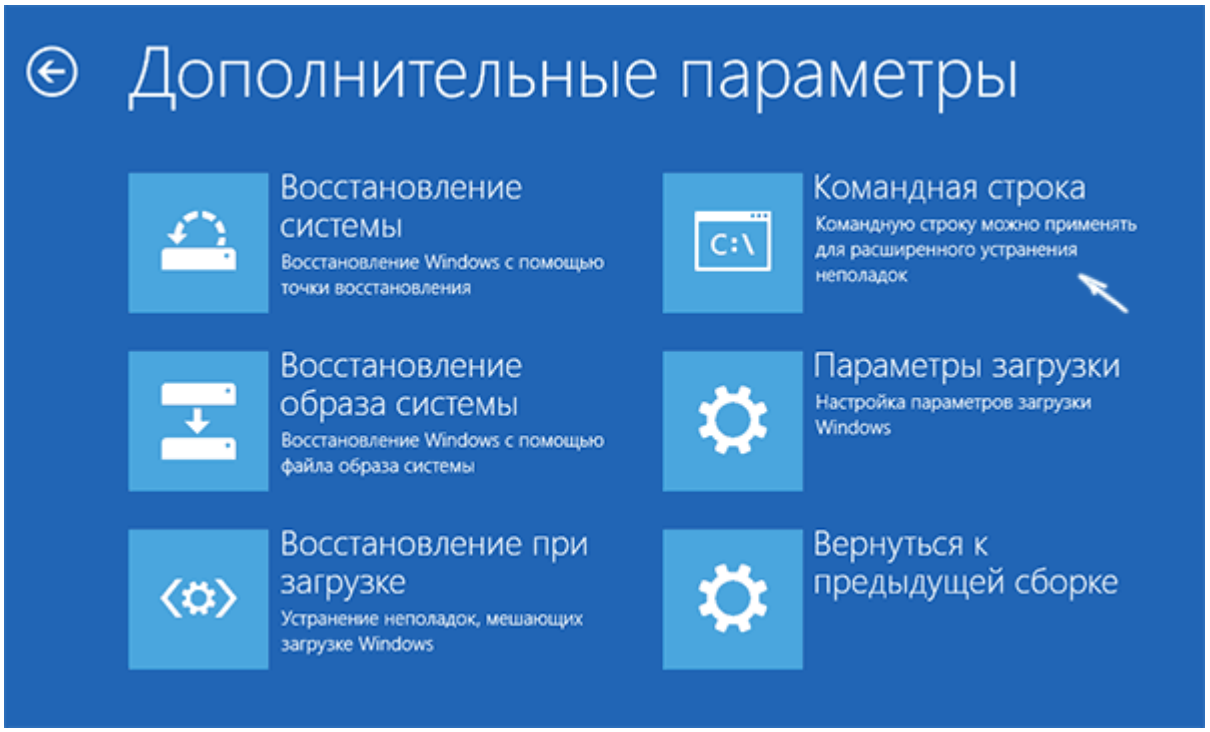
Для запуска команды стандартно используется командная строка, запущенная от имени администратора (запустить командную строку от администратора в Windows 10 можно, введя «Командная строка» в поиске в панели задач, затем — правый клик по найденному результату — Запуск от имени администратора), вводим в нее sfc /scannow и нажимаем Enter.

После ввода команды, начнется проверка системы, по результатам которой найденные ошибки целостности, которые можно исправить (о том, какие нельзя — далее) будут автоматически исправлены с сообщением «Программа защиты ресурсов Windows обнаружила поврежденные файлы и успешно их восстановила», а в случае их отсутствия вы получите сообщение о том, что «Защита ресурсов Windows не обнаружила нарушений целостности».



Также имеется возможность проверить целостность конкретного системного файла, для этого можно использовать команду
sfc /scanfile="путь_к_файлу"
Однако при использовании команды есть один нюанс: SFC не может исправить ошибки целостности для тех системных файлов, которые используются в настоящий момент времени. Чтобы решить проблему, можно запустить SFC через командную строку в среде восстановления Windows 10.
Запуск проверки целостности Windows 10 с помощью SFC в среде восстановления
Для того, чтобы загрузиться в среде восстановления Windows 10, вы можете использовать следующие способы:

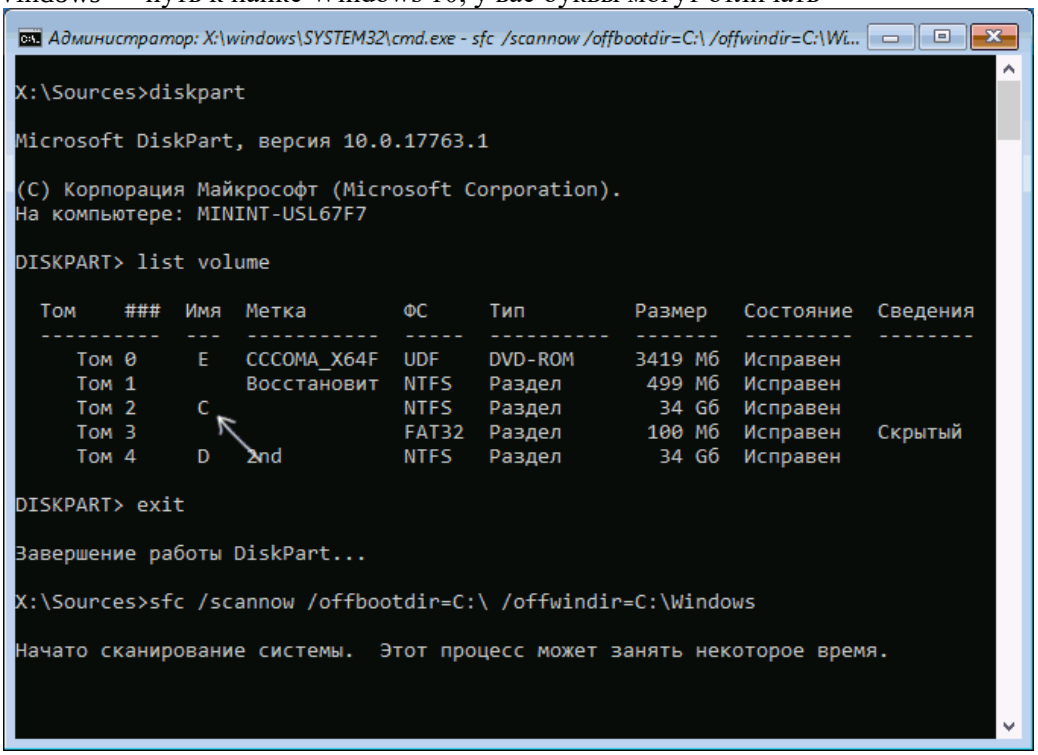
1. Зайти в Параметры — Обновление и безопасность — Восстановление — Особые варианты загрузки — Перезагрузить сейчас. (Если пункт отсутствует, то вы можете также использовать такой метод: на экране входа в систему, кликните по значку «вкл» справа внизу, а затем, удерживая Shift, нажмите «Перезагрузка»).
2. Загрузиться с заранее созданного диска восстановления Windows.
3. Загрузиться с установочного диска или загрузочной флешки с дистрибутивом Windows 10, а в программе установки, на экране после выбора языка, слева внизу выбрать «Восстановление системы».



4. После этого, зайдите в «Поиск и устранение неисправностей» — «Дополнительные параметры» — «Командная строка» (в случае если вы использовали первый из указанных выше способов, вам

также потребуется ввести пароль администратора Windows 10). В командной строке по порядку используйте следующие команды:

5. diskpart
6. list volume
7. exit
8. sfc /scannow /offbootdir=C:\ /offwindir=C:\Windows (где C — раздел с установленной системой, а C:\Windows — путь к папке Windows 10, у вас буквы могут отличаться).



9. Запустится сканирование целостности системных файлов операционной системы, при этом в этот раз команде SFC будет доступно восстановление всех файлов, при условии, что не повреждено хранилище ресурсов Windows.

Сканирование может продолжаться в течение значительного времени — пока мигает указатель подчеркивания, ваш компьютер или ноутбук не завис. По завершении закройте командную строку и перезагрузите компьютер в обычном режиме.

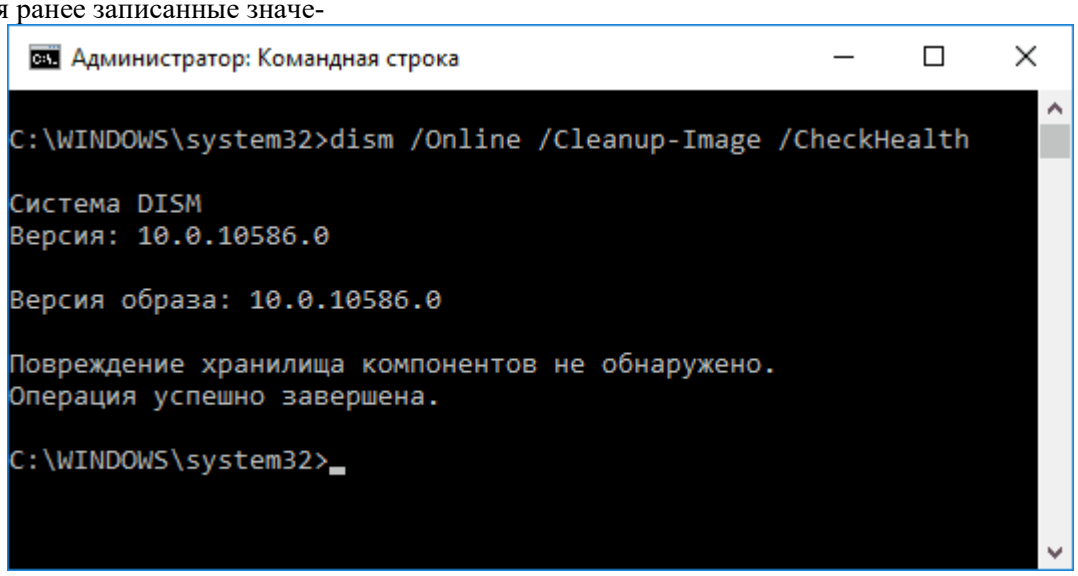
Восстановление хранилища компонентов Windows 10 с помощью DISM.exe

Утилита для развертывания и обслуживания образов Windows DISM.exe позволяет выявить и исправить те проблемы с хранилищем системных компонентов Windows 10, откуда при проверке и исправлении целостности системных файлов копируются оригинальные их версии. Это может быть полезным в тех ситуациях, когда защита ресурсов Windows не может выполнить восстановление файлов, несмотря на найденные повреждения. В этом случае сценарий будет следующим: восстанавливаем хранилище компонентов, а после этого снова прибегаем к использованию sfc /scannow.

Для использования DISM.exe, запустите командную строку от имени администратора. После чего можно использовать следующие команды:

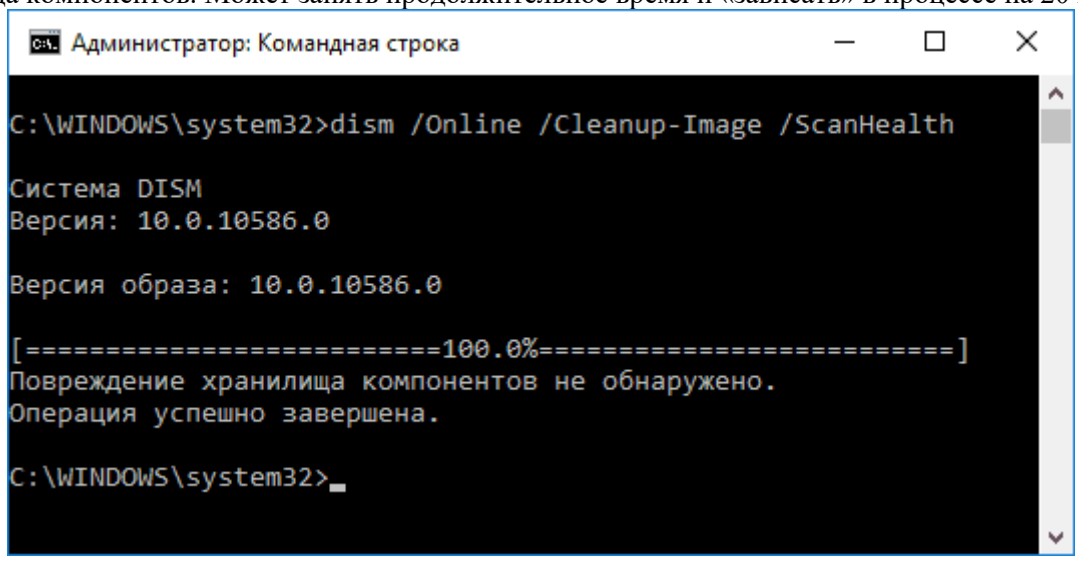
- dism /Online /Cleanup-Image /CheckHealth — для получения информации о состоянии и наличии повреждений компонентов Windows. При этом сама проверка не производится, а лишь проверя-

ются ранее записанные значе-



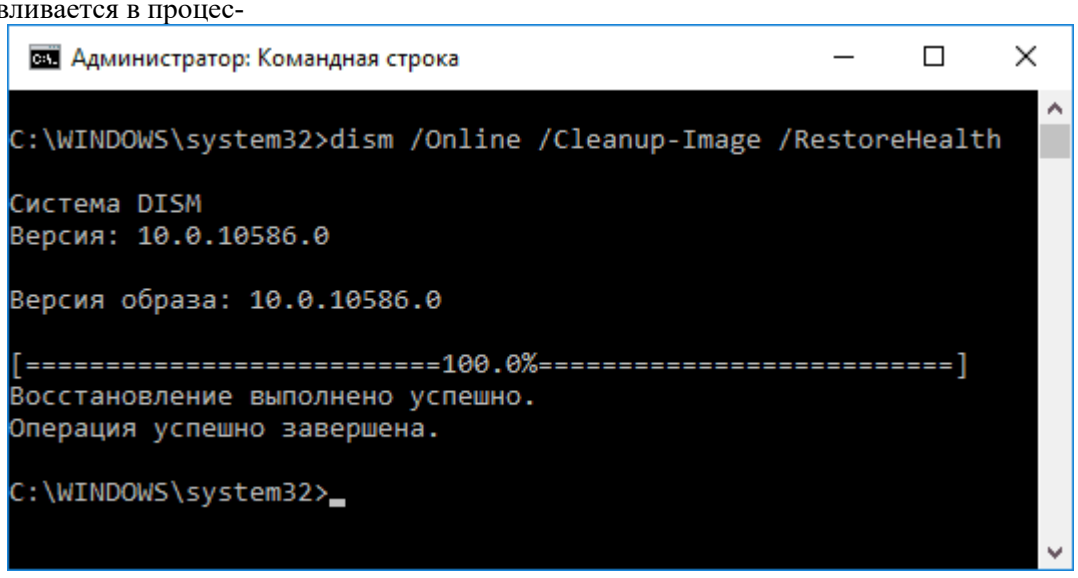
ния.

- dism /Online /Cleanup-Image /ScanHealth — проверка целостности и наличия повреждений хранилища компонентов. Может занять продолжительное время и «зависать» в процессе на 20 процен-



тах.

- dism /Online /Cleanup-Image /RestoreHealth — производит и проверку и автоматическое восстановление системных файлов Windows, также как и в предыдущем случае, занимает время и останавливается в процес-



се.

Примечание: в случае, если команда восстановления хранилища компонентов не работает по той или иной причине, вы можете использовать файл install.wim (или esd) со смонтированного ISO образа Windows 10 (Как скачать Windows 10 ISO с сайта Microsoft) в качестве источника файлов, требующих

восстановления (содержимое образа должно соответствовать установленной системе). Сделать это можно с помощью команды:
dism /Online /Cleanup-Image /RestoreHealth /Source:wim:путь_к_файлу_wim:1 /limitaccess
Вместо .wim можно использовать файл .esd тем же образом, заменив в команде все wim на esd.
При использовании указанных команд, журнал выполненных действий сохраняется
в Windows\Logs\CBS\CBS.log и Windows\Logs\DISM\dism.log.
DISM.exe также можно использовать в Windows PowerShell, запущенном от имени администратора (запустить можно из меню правого клика по кнопке Пуск) с помощью команды Repair-WindowsImage. Примеры команд:

- Repair-WindowsImage -Online -ScanHealth — проверка наличия повреждений системных файлов.
- Repair-WindowsImage -Online -RestoreHealth — проверка и исправление повреждений.

Дополнительные методы восстановления хранилища компонентов, если описанные выше не срабатывают: Восстановление хранилища компонентов Windows 10.
Как видите, проверка целостности файлов в Windows 10 — не такая уж и сложная задача, которая порой может помочь исправить самые разные проблемы с ОС. Если не смогла, возможно, вам поможет что-то из вариантов в инструкции Восстановление Windows 10

Практическая работа № 11 Сбор логов в Linux

Задание:
Настройте сбор логов в Линукс.
Уровни логов или приоритеты определяет ядро Linux. В зависимости от важности события, ему присваивается один из приоритетов, представленных ниже:

- KERN_EMERG - система неработоспособна;
- KERN_ALERT - нужно немедленно принять меры;
- KERN_CRIT - критическая ошибка;
- KERN_ERR - обычная ошибка;
- KERN_WARNING - предупреждение;
- KERN_NOTICE - замечание;
- KERN_INFO - информационное сообщение;
- KERN_DEBUG - сообщения отладки.

Соответственно, при логировании можно указать в настройках работы Syslog, события с каким приоритетом сохранять в каком файле.
Еще одним важным значением в сообщения является категория (Facility). Категории могут принимать значения от 0 до 23, им соответствуют различные категории системных служб: 0.
— kernel, 2 — mail, 7 — news и т. д. Последние 8 категорий — от local0 до local7 — определены
для служб, не попадающих в предопределённые категории.
Локальные недостатки
По умолчанию все журналы событий хранятся на машине локально. Однако даже в небольшой сети такой способ хранения логов не является оптимальным. Для чего нужны журналы событий? Для решения возможных проблем с производительностью, для мониторинга состояния системы и для своевременного выявления подозрительных и вредоносных активностей. И для всех этих активностей лучше использовать централизованное хранение логов на отдельном сервере или хранилище. Дело в том,
что на локальных узлах в целях экономии места логи циклически удаляются (ротируются) то есть,
при достижении максимального объема файла журнала, самые старые события удаляются. В результате мы можем лишиться важных событий просто потому что они уже затерлись более новыми. Также,
не слишком удобно проверять логи локально на каждом узле, централизованно анализировать гораздо проще. Большой поток событий может создавать дополнительную нагрузку на дисковую подсистему.
Отдельная история про безопасность. В случае, если злоумышленник проник в систему и захватил права root он сможет без труда изменить файлы журналов так, чтобы скрыть следы своих действий. В процессе взлома такие события, как подбор пароля и вход в систему обязательно отражаются в логах, и если их сразу передать на центральный сервер и проанализировать, то можно предотвратить атаку еще до того, как злоумышленнику удалось захватить систему.
Таким образом, необходимо сохранять основные события со всех узлов на отдельном сервере. Изначально в ОС Linux для этого использовался протокол Syslog, по которому события могли передаваться между серверами. Но этот протокол имеет существенные недостатки. В качестве транспортного протокола Syslog использует UDP, то есть отправка пакетов ведется без подтверждения получения. Таким образом нет никакой гарантии, что отправленное с одного узла событие на сервер Syslog будет доставлено.
В случае с критически важными событиями это не очень хорошо.

Кроме того, протокол Syslog не предусматривает никаких механизмов защиты. События передаются в открытом виде и могут быть легко перехвачены.

Rsyslog

В качестве альтернативы можно использовать протокол RSyslog (Rocket-fast System for log processing). Преимуществами этого протокола является наличие многопоточности (то есть возможность обрабатывать большой объем событий), использование протокола TCP на транспортном уровне, наличие шифрования SSL, а также возможность сохранения готовых событий в базы данных (MySQL, PostgreSQL, Oracle). Отдельного внимания заслуживает возможность фильтрации по любой части лога и полностью настраиваемый формат вывода событий.

Ниже приводится фрагмент конфига /etc/rsyslog.conf с клиентской машины.

```
# /etc/rsyslog.conf configuration file for rsyslog
#
# For more information install rsyslog-doc and see
# /usr/share/doc/rsyslog-doc/html/configuration/index.html
#
# Default logging rules can be found in /etc/rsyslog.d/50-default.conf


#################
#### MODULES ####
#################

module(load="imuxsock") # provides support for local system logging
#module(load="immark")  # provides --MARK-- message capability

# provides UDP syslog reception
#module(load="imudp")
#input(type="imudp" port="514")

# provides TCP syslog reception
#module(load="imtcp")
#input(type="imtcp" port="514")

# provides kernel logging support and enable non-kernel klog messages
module(load="imklog" permitnonkernelfacility="on")

###########################
#### GLOBAL DIRECTIVES ####
###########################

#
# Use traditional timestamp format.
# To enable high precision timestamps, comment out the following line.
#
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

# Filter duplicated messages
$RepeatedMsgReduction on

#
# Set the default permissions for all log files.
#
$FileOwner syslog
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
```

В качестве примера настроим пересылку логов с клиента на сервер. Для этого на стороне сервера необходимо в файле /etc/rsyslog.conf добавить следующие строки:

$ModLoad imtcp

$InputTCPServerRun 514

Для большей надежности мы используем TCP в качестве транспортного протокола и стандартный порт 514, на котором сервер будет слушать трафик.

Для того, чтобы не запутаться в собираемых журналах событий лучше всего сразу сохранять пришедши е события в отдельных файлах. Идея складывать все события в один файл очень плоха, так как файлы логов должны, либо подвергаться циклической чистке, когда старые события удаляются, либо архивироваться. И то, и другое ведет к тому, что вы можете потерять часть важных событий. Поэтому лучше использовать следующую структуру /var/log/rsyslog/ИМЯ_УЗЛА_источника/приложение_источник.log.

Для этого надо добавить в файл конфигураций следующие строки:

$template RemoteLogs,"/var/log/rsyslog/%HOSTNAME%/%PROGRAMNAME%.log"

. ?RemoteLogs

Так как Rsyslog может содержать набор правил для обработки приходящих событий, нам необходимо явно сообщить ему, что пришедшие события не надо дополнительно обрабатывать.

Для этого добавим:

& ~

Далее перезапускаем службу Rsyslog:
systemctl restart rsyslog
На клиенте необходимо настроить пересылку событий на сервер. Для этого нужно создать отдельный файл конфигураций:
nano /etc/rsyslog.d/log.conf
В этот файл добавляем строку, указывающую какие журналы, и куда необходимо пересылать
. @@192.168.222.135:514
В примере мы пересылаем все события с клиента на порт 514 сервера 192.168.222.135.
Но если нас не интересуют все события, а нужны только логи из конкретной категории, например аутентификации, то нужно указать следующее:
auth.* @@192.168.222.135:514
Далее также перезапускаем Rsyslog
systemctl restart rsyslog
В журнале событий на сервере можно убедиться, что события с клиента успешно приходят:

```
Feb  1 06:01:50 192.168.222.133 sshd[1379]: Accepted password for otus from 192.168.222.1 port 59153 ssh2
Feb  1 06:01:50 192.168.222.133 sshd[1379]: pam_unix(sshd:session): session opened for user otus(uid=1000) by (uid=0)
Feb  1 06:01:50 192.168.222.133 systemd-logind[908]: New session 3 of user otus.
```

Модули Rsyslog
Иногда возникает необходимость в более интеллектуальной обработке журналов событий, поступающих с источников. Для этого в Rsyslog имеются модули. Модули ввода — начинаются с im, собирают информацию из различных источников. Модули вывода — начинаются на om, и они отправляют сообщения. Могут отправлять сообщения как в файл так и по сети или складывать в базу. Модули фильтрации — начинаются с fm. Фильтруют сообщения по разным параметрам. Модули парсинга — начинаются с pm. Позволяют проводить синтаксический анализ. Модули модификации сообщений — начинаются с mm. Меняют содержимое обрабатываемых сообщений. Модули генерации строк — начинаются с sm. Позволяют генерировать строки на основе обрабатываемых сообщений.
Посмотрим пример для того, чтобы было понятно, о чем идет речь. В следующем примере мы будем на клиенте отслеживать изменения в файле audit.log и отправлять на сервер события с уровнем warning и выше и категорией local0.
$ModLoad imfile
$InputFileName /var/log/audit/audit.log
$InputFileTag tag_audit_log:
$InputFileStateFile audit_log
$InputFileSeverity warning
$InputFileFacility local0
$InputRunFileMonitor
. @@192.168.222.135:514
На сервере также необходимо модифицировать файл конфигураций для того, чтобы данные события сохранялись в отдельном файле:
$template HostAudit, "/var/log/rsyslog/%HOSTNAME%/audit.log"
Local0.* ?HostAudit
Сохранение в БД
Собранные события удобно хранить в текстовых файлах лишь когда их не более десятка. При большем объем лучше использовать СУБД для централизованного хранения и обработки событий. В качестве примера будем отправлять все события в MySQL. Вот общий формат таких настроек:
$ModLoad ommysql
. :ommysql:адрес_сервера,имя_базы,имя_пользователя,пароль
Например:
mail.* :ommysql:127.0.0.1,syslog,syslogwriter,topsecret
В случае использования PostgreSQL формат будет следующий:
$ModLoad ompgsql
. :ompgsql:адрес_сервера,имя_базы,имя_пользователя,пароль
Использование СУБД позволяет не только собирать и хранить события, но и делать запросы по наличию тех или иных событий, строить отчеты и реагировать на появление определенных событий. Хотя, когда количество источников начинает измеряться сотнями, для работы с событиями уже лучше использовать специализированные решения. Например, для обработки событий ИБ лучше использовать решения класса SIEM (Security Information Event Management).

Практическая работа № 12 Обнаружение инцидентов в ОС

Задание:
Настройте auditd, обнаружьте и опишите произошедший инцидент.
auditd (сокращение от Linux Audit Daemon) — нативный инструмент предназначенный для мониторинга событий операционной системы и записи их в журналы событий, разрабатываемый и поддерживаемый компанией RedHat. Был создан для тесного взаимодействия с ядром операционной системы — во время своей работы наблюдает за системными вызовами и может записывать события — чтение, запись, выполнение, изменение прав - связанные с файлами ОС. Таким образом, с его помощью можно отслеживать практически любые события, происходящие в операционной системе.
Плюсы auditd:
- работает на низком уровне мониторинга — отслеживает системные вызовы и действия с файлами;
- имеет неплохой набор утилит в комплекте для удобства работы;
- постоянно развивается и обновляется;
- бесплатен и легко устанавливается.
Минусы auditd:
- большинство событий, возникающих при атаках характерных для конкретного приложения, практически невозможно отслеживать поскольку на уровне системных вызовов и работе с файлами трудно отличить взлом от нормальной работы приложения. Такие события лучше отслеживать на уровне самих приложений;
- auditd может замедлять работу ОС. Это связанно с тем, что подсистеме аудита необходимо проводить анализ системных вызовов;
- не слишком гибок в настройке правил;
- на данный момент это не лучший инструмент для работы с контейнерами.
Файлы конфигурации и синтаксис
Рассмотрим основные примеры настроек и параметров, которые будут использоваться далее. Более полное руководство по auditd вы можете найти здесь. Основную информацию можно найти в мануалах к auditd и его инструментам.
Файлы конфигурации хранятся в /etc/audit/. Правила желательно хранить в /etc/audit/rules.d/*.rules, по умолчанию доступ к этой директории только у root'a. Обратите внимание на то, что файл с правилами в этой директории должен иметь название *.rules, иначе auditd не прочитает его без явного указания. Если вы решили хранить правила в другом месте, то владелец файла должен быть root. Помимо этого рекомендую выставить группу файла root и права 600, чтобы никто кроме root'a не мог работать с файлом конфигурации auditd, т.к. зная что логируется, атакующий может избежать обнаружения. То же самое касается и файлов с правилами для других инструментов.
Правила для логирования можно добавлять следующими способами:
- записать его в файл(ы) /etc/audit/rules.d/<имя файла>.rules и перезапустить сервис;
- записать в файл по произвольному пути и указать его явно: auditctl -R <путь к файлу>;
- добавить правило утилитой auditctl [-A,-a] <правило>.
Синтаксис
Подробное описание синтаксиса на русском языке можно посмотреть здесь.
-D - удалить все правила. Обычно используется в начале файла, чтобы избежать неожиданностей;
-a [list,action],[action,list] - добавляет правило в конец списка правил. Списки и действия рассмотрим далее.
Основные варианты списков:
exit - Добавить правило к списку, отвечающему за точки выхода из системных вызовов. Этот список применяется, когда необходимо создать событие для аудита, привязанное к точкам выхода из системных вызовов.
exclude - Добавить правило к списку, отвечающего за фильтрацию событий определенного типа. Этот список используется, чтобы отфильтровывать ненужные события. Например, если вы не хотите видеть avc сообщения, вы должны использовать этот список. Тип сообщения задается в поле *msgtype*.
Варианты действий:
always - установить контекст аудита. Всегда заполнять его во время входа в системный вызов и всегда генерировать запись во время выхода из системного вызова;
never - аудит не будет генерировать никаких записей. Это может быть использовано для подавления генерации событий. Обычно необходимо подавлять генерацию вверху списка, а не внизу, т.к. событие инициируется на первом совпавшем правиле.

-A list,action - добавить правило в начало списка. Например, для удобство чтения правило находится ниже, чем должно быть по логике настроек, тогда можно использовать этот параметр.

-F [n=v | n!=v | n<v | n>v | n<=v | n>=v | n&v | n&=v] - задать поле сравнения для правила. Атрибуты поля следующие: объект, операция, значение. Вы можете задать до 64 полей сравнения в одной команде. Каждое новое поле должно начинаться с -F. Аудит будет генерировать запись, если произошло совпадение по всем полями сравнения. Допустимо использование одного из следующих 8 операторов: равно, не равно, меньше, больше, меньше либо равно, больше либо равно, битовая маска (n&v) и битовая проверка (n&=v). Битовая проверка выполняет операцию «and» над значениями и проверяет, равны ли они. Битовая маска просто выполняет операцию «and». Поля, оперирующие с идентификатором пользователя, могут также работать с именем пользователя — программа автоматически получит идентификатор пользователя из его имени. То же самое можно сказать и про имя группы.
Поля сравнения и их описание:

- **a0, a1, a2, a3** - первые 4 аргумента системного вызова соответственно;
- **arch** - так как система ориентируется на номера (не названия) системных вызовов, а для многих системных вызовов номера отличаются для 32 и 64 разрядных систем, то необходимо указывать для какой архитектуры мы пишем правило;
- **auid** - ID пользователя, с которым он вошёл в систему. Системные сервисы, как правило, имеют auid=-1 (или 4294967295);
- **dir** - директория, за которой необходимо наблюдать. Будут залогированы и все события связанные с файлами и поддиректориями в указанной директории рекурсивно;
- **euid** - действительный идентификатор пользователя;
- **exe** - полный путь к исполняемому файлу. Может использоваться только с exit;
- **exit** - значение, возвращаемое системным вызовом при выходе;
- **key** - установка поля для фильтра. Добавляет поле с заданным именем в событие, что облегчает поиск событий в журналах;
- **msgtype** - тип события. Весь список событий можно посмотреть здесь;
- **path** - полный путь к файлу, за которым необходимо следить, может использоваться только с exit;
- **perm** - то же, что и параметр -p, будет рассмотрен ниже;
- **success** - если значение, возвращаемое системным вызовом, больше либо равно 0, данный объект будет равен «true/yes», иначе «false/no». При создании правила используйте 1 вместо «true/yes» и 0 вместо «false/no»;
- **uid** - идентификатор пользователя;

Вместо числовых идентификаторов пользователей можно указывать имена (www-data, mail, irc), таким образом вам не придется учитывать их числовые различия на разных серверах.
-p - [r|w|x|a] - описывает разрешение доступа файла за которым нужно следить: чтение, запись, выполнение или изменение прав доступа соответственно;
-w <path> - устанавливает наблюдение за директорией (рекурсивно) или файлом;
-W <path> - исключает наблюдение за указанной директорией или файлом.
Стоит упомянуть про встроенные инструменты анализа полученных событий: ausearch, aureport. С их помощью удобно тестировать правила "на месте". Много интересных примеров правил можно посмотреть здесь.
Общие принципы написания правил
Некоторые специалисты предпочитают писать правила максимально широкими, а логику фильтрации и обработки событий уже настраивать в подсистеме анализа событий, чтобы атакующий не знал какие его действия собираются и анализируются. На мой взгляд, у такого подхода есть недостаток: порождается большое количество событий из-за чего могут возникнуть проблемы с производительностью на серверах, где ведется сбор событий, и далее по цепочке архитектуры SIEM, особенно если источников логов много. Помимо этого в большой инфраструктуре сильно возрастает нагрузка по выявлению полезных событий среди общего потока в конечной точке.
Чтобы избежать случаев, когда атакующий может узнать, какие его действия анализируются, нужно установить мониторинг за файлом с правилами логирования таким образом, чтобы при чтении этого файла любым пользователем возникало событие, а также перечислить команды, позволяющие узнать правила без чтения файлов конфигурации такие как audictl -l.
На основе этих событий можно сделать алерты. Если логи передаются по сети, то, помимо установки наблюдения за доступом к правилам, необходимо настроить шифрование данных.
В независимости от того, какой подход будет выбран вами между ними есть много общего и, не смотря на то что в примерах далее почти вся логика будет настраиваться на источнике событий, фильтры/правила можно писать как на стороне источника событий, так и в подсистеме анализа событий (или даже на промежуточном этапе).
При написании правил auditd необходимо учитывать следующее:

1. Для каждого события отрабатывает лишь то подходящее правило, которое встретилось первым. Поэтому сначала пишутся фильтры и только потом правила. То же самое касается и выбора между несколькими правилами - выше размещать стоит то правило, которое важнее учитывать.
2. Писать правила лучше от частного к общему. Допустим, вы хотите журналировать действия в директории /etc/. Чтобы потом в логах не искать прикладными утилитами (grep, sed или средствами SIEM) все события, связанные, например, с ssh, sudoers, passwd и т.д., сначала указываете правила для мониторинга конкретных директорий/файлов в /etc/ и только после этого размещаете правило для самой директории /etc/.
3. В auditd есть преднастроенные правила и иногда возникают ситуации, когда они срабатывают и наши правила не учитываются. Поэтому каждое правило лучше предварительно тестировать отдельно.
4. Если вы хотите написать правила с целью выявления конкретного случая (атаки, ситуации), то лучше определить общее звено и написать правило для него. Так, например, для обнаружения запуска интерактивных шеллов можно использовать обращения к /dev/tty, /dev/pts/. Чем лучше вы понимаете как работает ваша операционная система, тем лучше. Используя такой подход, злоумышленнику станет тяжелее избежать обнаружения.
5. Для одного и того же действия с точки зрения пользователя может существовать несколько системных вызовов. Так для открытия файла могут использоваться: open, openat, creat, open_by_handle_at. Об этом стоит помнить, при создании правил на базе выборочных системных вызовов.
6. Если вы написали правило и видите множество ложных срабатываний, то, возможно, вместо написания множества фильтров, следует выбрать другой подход к определению события логирования.

Как тестировать правила

Прежде чем перейдем к конкретным примерам, стоит рассмотреть порядок тестирования будущих правил. В случае с инфраструктурой состоящей из нескольких серверов правила можно практически сразу применять, чего не скажешь о случаях, когда серверов десятки, сотни, или даже больше. Применение многих правил без предварительного тестирования способно вызвать перегрузку в разных местах: на самом сервере, на сети или в системе обработки событий. Чтобы избежать этого можно проводить тестирование следующим образом:

1. Проверка тестируемого правила на одном сервере. Чем ближе по составу установленного ПО сервер приближен к "боевой" среде, тем лучше. На этом этапе необходимо во-первых смоделировать условия, для которых написано правило, а во-вторых протестировать все возможные варианты поведения сервиса/приложения/пользователя, которые могут вызвать ложные срабатывания.
2. Проверка тестируемого правила в составе имеющихся других правил. На этом этапе проводим тесты и убеждаемся в том, что всё по-прежнему работает как задумано: другие правила могут перекрывать новое и наоборот.
3. Проверка нового набора правил на группе серверов. Снова, чем ближе к будущей инфраструктуре будут сервера по составу ПО, тем лучше. При необходимости этот этап можно повторять, увеличивая количество серверов.
4. Применение правил на всей инфраструктуре. Если по предыдущим пунктам вы убедились в "безопасности" нового набора, то можно применять его на всей инфраструктуре.

Если у вас большое количество серверов, то всегда стоит помнить о том, что какой-то участок вашей инфраструктуры может сработать не так как ожидалось и породить большое количество событий, поэтому пути решения возможных проблем лучше обдумать сразу.

Алгоритм тестирования правил

Для каждого правила в частности будем использовать следующий алгоритм:

1. Пробуем определить события для отслеживания по **формуле**:

AA - SA = IE

где AA - attacker's actions - набор действий атакующего, SA - service actions - набор действий сервиса в ходе его обычной деятельности, IE - incident events - события инцидента. Положительный результат (события инцидента) в формуле и будет основной целью для нас. Каждый набор действий состоит из системных вызовов и обращений к файлам. Эта формула может быть особенно полезна в случае с обнаружением инцидентов. Если явно выявить события инцидента не получается, то можно определить события, которые помогут при расследовании инцидента: обращения к важным файлам, выполнение команд и пр. На мой взгляд - хорошее правило то, для которого не нужно писать много фильтров и порождает события, по которым с высокой вероятностью можно сказать, что произошёл инцидент.

2. Пишем правило(а) для набора действий определенных в пункте 1.

3. Применяем правила: помимо событий которые мы хотим отслеживать, могут попадаться и другие события, о которых мы не знаем. Поэтому здесь необходимо создать как можно больше возможных вариантов легитимного поведения сервиса или событий, чтобы выявить действия сервиса, которые могут вызывать события по нашему правилу.
4. Пишем фильтры: если мы нашли такие события (п.3) то стоит подумать - можем ли мы от них избавиться. Писать фильтры желательно максимально точно, чтобы не "зацепить" лишних событий. В статье будет предложен вариант с написанием фильтров на хостах-источниках событий.
5. Моделируем ситуацию: создаём ситуацию, для которой написали правило и убеждаемся, что создаются необходимые события.

Модель угроз

Модель угроз для определения отслеживаемых событий, на мой взгляд, стоит рассматривать аналогично классическому: для обнаружения инцидента определяем все возможные способы взлома (вся информация, которая попадает на сервер), для расследования - вся информация, за которой мы хотим наблюдать. На этом этапе будет полезно устроить мозговой штурм среди коллег с целью составления способов взлома и/или проникновения на сервер в случае с обнаружением инцидента, а также наблюдаемых ресурсов для расследования. Затем можно в каждой из подгрупп провести ранжирование по важности отслеживаемых событий, после чего приступать к написанию правил. Подробную моделирование угроз выполнять не будем, т.к. это отдельная большая тема. Рассмотрим подгруппы подробнее.

Обнаружение инцидентов

С точки зрения логирования будем считать инцидентом ситуацию, при которой у атакующего появилась возможность выполнения команд и/или чтения файлов операционной системы. Для написания правил по обнаружению инцидентов, необходимо понять, каким образом он может вообще возникнуть. Поскольку для взлома сервера на него должны каким-то образом попасть данные от атакующего по сети, то в первую очередь необходимо определить пути попадания такой информации. Это любые сервисы, которые устанавливают соединения наружу и/или принимают входящие соединения. Причем чем раньше мы сможем установить факт компрометации сервера, тем, очевидно, лучше. Основные способы выполнения команд на сервере:

- выполнение уже имеющихся исполняемых файлов (файлы директорий /bin/, /usr/bin/ и т.д.) — самый популярный способ. Например, атакующий получил доступ к командной оболочке напрямую или через имитацию шелла;
- выполнение команд через запись в файлы. Пример — планировщик задач cron. Если вы используете файлы, содержимое которых может быть выполнено как команды системой, то за ними необходим контроль;
- выполнение команд напрямую через системные вызовы — такой способ может применяться, например, в ходе "бинарных" атак: переполнения стека, кучи и т.д.

Что касается файлов, то наша задача определить те, что используется сервисом в обычной деятельности точно исключить такие события, поскольку в случае взлома сервиса, определить аномалию по отношению к таким файлам очень сложно.

Помимо вышеперечисленного, будем опираться на этапы проведения атаки:

1. Сканирование сервера.
   Злоумышленник производит обнаружение открытых портов на сервере извне.
2. Атака сервера, эксплуатация.
   На этом этапе злоумышленник каким-либо образом получает возможность чтения файлов, выполнению команд или системных вызовов на сервере. В этот условный этап будем относить все действия атакующего вплоть до момента выполнения любой первой команды и/или чтения первого файла, когда ему необходимо понять, что он смог добиться успеха в проведении атаки.
3. Взаимодействие с сервером, постэксплуатация.
   Вслед за взломом злоумышленник может попытаться закрепиться на сервере (оставить файл с шеллом, получить ssh-ключ и т.д.), найти информацию для продвижения вглубь инфраструктуры (поиск паролей, токенов в файлах, базах данных и т.д.).

Вы можете воспользоваться как уже имеющимися вариантами, например, mitre, так и использовать свой подход.

Расследование инцидентов

В этом случае нам необходима информация, которая поможет разобраться какие действия злоумышленник совершал после взлома: какие действия он выполнял, какие файлы читал и т.д. Принципы написания правил здесь будут отличаться от принципов при обнаружении инцидентов. В общем случае здесь необходимо устанавливать мониторинг за выполненными командами и важными с точки зрения безопасности файлами. Правила можно поделить на общие и частные. Общие - это те правила, которые подходят к лю-

бому серверу, вне зависимости от его назначения. Сюда можно отнести файлы конфигурации системы, пользователей и т.д. Частные - файлы и команды, характерные для сервера с конкретным назначением - веб-сервер, гипервизор и пр. Подробнее об этом рассмотрим в примере далее.

Основное отличие при работе по обнаружению инцидентов от расследования - стремление к созданию таких правил, события которых будут означать возникновение инцидента. Кратко: если возникло событие по правилу инцидента, то возник и сам инцидент.
В качестве примера возьмём сервер на котором есть http-сервис и ssh. Для каждого сервиса каждый этап взлома проведем через наш алгоритм. Проведём каждый сервис по этапам атаки через алгоритм, для каждого этапа будем применять формулу. Правила для расследования инцидентов рассмотрим отдельно позже, поскольку они подходят практически для любых видов сервисов. Все примеры ниже приведены на виртуальной ОС ubuntu 18.04, веб-сервер apache.
Будет полезен скрипт для автоматизации наших действий:

```
#!/bin/bash

killall -9 /sbin/auditd 2>/dev/null; \
systemctl stop auditd; \
systemctl stop apache2; \
sleep 2; \
rm /var/log/audit/audit.log*; \
systemctl start auditd; \
systemctl start apache2; \
auditctl -R /home/ubuntu/auditd/apache.rules > /dev/null 2>&1; \
auditctl -l; \
wc -l /var/log/audit/audit.log
```

Для быстрого завершения сначала убиваем процессы auditd, останавливаем сервисы auditd и apache2 (иногда без перезапуска apache2 правила по какой-то причине не применяются), удаляем все журналы логов, чтобы в поиске не выдавались старые результаты, запускаем сервисы, читаем правила из файла /home/ubuntu/auditd/apache.rules, проверяем список примененных правил, чтобы убедиться в что они работают и проверяем количество записей в файле лога аудита.
Файл /home/ubuntu/auditd/apache.rules:

```
-D
<тестируемые правила>
```

В первой строке **-D** используется для очистки существующих правил, которые мы могли забыть, чтобы избежать неожиданного поведения.
HTTP-Сервис

HTTP. Сканирование сервиса

1. На уровне системных вызовов мы можем определить лишь то сканирование, которое используют вызов connect, потому что тогда наш сервер использует accept во время установления соединения. В большинстве случаев мы не можем отличить соединение злоумышленника, поскольку такое соединение ни чем не отличается от обычных пользователей. Однако, если у вас такой веб-сервер находится внутри инфраструктуры и есть ограниченное количество сетевых устройств, которые могут к нему подключаться, то такое правило может иметь смысл в случае определения белого списка сетевых устройств устанавливающих соединение.
2. Правило для auditd будет выглядеть следующим образом:

-a **exit**,always -S accept -S accept4 -F exe=/usr/sbin/apache2 -k accept
Читается как: всегда на выходе из системных вызовов accept или accept4 для исполняемого файла /usr/sbin/apache2 логировать события и добавлять к ним метку accept.

3. Понятно, что любой соединение от клиента будет вызывать появление таких событий, смоделируем сканирование:

nc -z localhost 80
Результат:

```
$ ausearch -k apache_accept -i
----
type=PROCTITLE msg=audit(12.08.2021 16:05:29.654:395) : proctitle=/usr/sbin/apache2 -k start
type=SOCKADDR msg=audit(12.08.2021 16:05:29.654:395) : saddr={ fam=inet6 laddr=::ffff:127.0.0.1
lport=37112 }
type=SYSCALL msg=audit(12.08.2021 16:05:29.654:395) : arch=x86_64 syscall=accept4 success=yes exit=11
```

a0=0x4 a1=0x7ffd4ca5c800 a2=0x7ffd4ca5c7e0 a3=0x80000 items=0 ppid=5041 pid=5050 auid=**unset** uid=www-data gid=www-data euid=www-data suid=www-data fsuid=www-data egid=www-data sgid=www-data fsgid=www-data tty=(none) ses=**unset** comm=apache2 exe=/usr/sbin/apache2 key=apache_accept
Видим, что поле по которому мы хоть как-то могли бы фильтровать события это saddr, однако такого поля в опциях у auditd нет и такое возможно вспомогательными средствами: go-audit, elk;

4. Поскольку фильтры, которые были бы нам полезны, написать не можем, пропускаем этот шаг.

Результат: в широком смысле мы можем отслеживать все входящие сетевые соединения, а логику для написания алертов необходимо дополнительно настраивать. В целом - это не лучшее правило для создания событий с целью обнаружения инцидентов, однако в определенных случаях такие правила могут быть полезны.

HTTP. Атака сервиса

1. На данном этапе злоумышленник каким-либо образом получил возможность читать файлы ОС или выполнять команды. Можем предположить, что действия атакующего будут затрагивать файлы в директориях /bin/, /usr/bin/, часто пытаются прочитать файл /etc/passwd как в ходе автоматизированной так и при ручной атаках. Действия веб-сервера в большинстве случаев находится в директории /var/www/. Таким образом, возникает идея логировать действия для пользователя www-data везде, кроме директории /var/www.
2. Далее пишем правила для наших предположений:

-a never,exit -F dir=/var/www/ -F uid=www-data
-w / -F uid=www-data -k apache_alert

3. Затем переходим к тестированию. После выполнения скрипта check.sh, обратимся к одной из страниц веб-сервера. Сделаем файл /var/www/html/test.php с содержимым:

<**?php** phpinfo(); >?
После этого загружаем через браузер этот файл, останавливаем и запускаем службу:
Логи
Как видим, в логах присутствуют записи, которые мы не ожидали увидеть.

4. Пишем фильтры. Из получившихся событий мы видим, что пользователь www-data обращается к файлам директории /usr/share/zoneinfo/. Поскольку никакой ценной информации с точки зрения безопасности в этой директории не содержится, мы можем написать фильтр, чтобы избавиться от этих событий:

-a never,exit -F dir=/usr/share/zoneinfo/ -F uid=www-data
Тестируем снова, убеждаемся, что эти события пропали из журналов и других "лишних" событий не присутствует. Можем переходить к моделированию поведения атакующего.

5. Смоделируем условия, при которых атакующий смог воспользоваться уязвимостью LFI или загрузить веб-шелл и воспользоваться какой-то командой.

В директории /var/www/html/ разместим файл read_passwd.php с содержимым:

<**?php**
$myfile = fopen("/etc/passwd", "r") or die("Unable to open file!");
echo fread($myfile,filesize("/etc/passwd"));
fclose($myfile);
?>
И еще один shell.php который имитирует простейший веб-шелл:
<**?php if**(isset($_REQUEST['cmd'])){ **echo** "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); **echo** "</pre>"; **die**; }**?>**
Получим через браузер сначала read_file.php и выполним поиск по логам:
$ ausearch -k apache_alert -i
---
type=PROCTITLE msg=audit(12.08.2021 23:12:08.373:688) : proctitle=/usr/sbin/apache2 -k start
type=PATH msg=audit(12.08.2021 23:12:08.373:688) : item=0 name=/etc/passwd inode=264156 dev=08:01 mode=file,644 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(12.08.2021 23:12:08.373:688) : cwd=/var/www/html
type=SYSCALL msg=audit(12.08.2021 23:12:08.373:688) : arch=x86_64 syscall=openat success=yes exit=12 a0=0xffffff9c a1=0x7ffea2903630 a2=O_RDONLY a3=0x0 items=1 ppid=3410 pid=3419 auid=**unset** uid=www-data gid=www-data euid=www-data suid=www-data fsuid=www-data egid=www-data sgid=www-data fsgid=www-data tty=(none) ses=**unset** comm=apache2 exe=/usr/sbin/apache2 key=apache_alert
После этого в браузере выполним запрос:
http://192.168.0.101/shell.php?cmd=ls

И выполнив поиск по логам, увидим, что добавились события:
---
type=PROCTITLE msg=audit(12.08.2021 23:14:57.566:707) : proctitle=sh -c ls
type=PATH msg=audit(12.08.2021 23:14:57.566:707) : item=0 name=/etc/ld.so.cache inode=265858 dev=08:01 mode=file,644 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(12.08.2021 23:14:57.566:707) : cwd=/var/www/html
type=SYSCALL msg=audit(12.08.2021 23:14:57.566:707) : arch=x86_64 syscall=openat success=yes exit=3 a0=0xffffff9c a1=0x7feec9ccbea8 a2=O_RDONLY|O_CLOEXEC a3=0x0 items=1 ppid=3422 pid=3433 auid=**unset** uid=www-data gid=www-data euid=www-data suid=www-data fsuid=www-data egid=www-data sgid=www-data fsgid=www-data tty=(none) ses=**unset** comm=sh exe=/bin/dash key=apache_alert
---
type=PROCTITLE msg=audit(12.08.2021 23:14:57.562:706) : proctitle=sh -c ls
type=PATH msg=audit(12.08.2021 23:14:57.562:706) : item=1 name=/lib64/ld-linux-x86-64.so.2 inode=3156086 dev=08:01 mode=file,755 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=PATH msg=audit(12.08.2021 23:14:57.562:706) : item=0 name=/bin/sh inode=1966114 dev=08:01 mode=file,755 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(12.08.2021 23:14:57.562:706) : cwd=/var/www/html
type=EXECVE msg=audit(12.08.2021 23:14:57.562:706) : argc=3 a0=sh a1=-c a2=ls
type=SYSCALL msg=audit(12.08.2021 23:14:57.562:706) : arch=x86_64 syscall=execve success=yes exit=0 a0=0x7f5978e1de1a a1=0x7ffea29045e0 a2=0x7ffea29073d8 a3=0x1 items=2 ppid=3422 pid=3433 auid=**unset** uid=www-data gid=www-data euid=www-data suid=www-data fsuid=www-data egid=www-data sgid=www-data fsgid=www-data tty=(none) ses=**unset** comm=sh exe=/bin/dash key=apache_alert
....
Всё работает как было задумано.
Результат: такими правилами мы обнаружим любое обращение к файлам, которые не используются веб-сервером в рамках привычной своей деятельности и могут являться целью злоумышленника. В приведенном примере использовался веб-сервер без каких-либо приложений, фильтров может быть больше, однако, если вам удастся подобным образом написать правила, то многие наиболее критичные виды веб-уязвимостей (RCE, LFI, SSTI, XXE и т.д.) при выполнении команд в системе или чтении файлов могут быть обнаружены в момент их реализации.
В случае, если исключений слишком много. можно пойти другим путем и написать правила для наблюдения только за наиболее важными директориями в системе. Например:
dirs from PATH var
-w /bin -F uid=www-data -k apache_alert
-w /usr/local/sbin -F uid=www-data -k apache_alert
-w /usr/local/bin -F uid=www-data -k apache_alert
-w /usr/sbin -F uid=www-data -k apache_alert
-w /usr/bin -F uid=www-data -k apache_alert
-w /sbin -F uid=www-data -k apache_alert

-w /etc -F uid=www-data -k apache_alert
Таким образом при обращении пользователя веб-сервера к исполняемым файлам системы будут созданы соответствующие события.
HTTP. Постэксплуатация
Поскольку на предыдущем шаге мы установили наблюдение за почти всеми возможными ... непокрытое поле для злоумышленника остаётся только в директории /var/www/. Можем предположить, что он может попытаться записать новый файл с функционалом для веб-приложения или изменить существующий для реализации новой логики. Если у вас приложение не создаёт динамических страниц, то для обнаружения изменений в существующих файлах веб-приложения можно написать следующие правила:
-w /var/www/ -p wa -F uid=www-data -k apache_file_change
Тогда для всех случаев когда файлы открываются для записи или изменения прав веб-сервером будут записаны события. Это правило необходимо разместить над фильтрами. Итоговый набор правил может выглядеть так:
-D
-w /var/www/ -p wa -F uid=www-data -k apache_file_change

-a never,exit -F dir=/usr/share/zoneinfo/ -F uid=www-data
-a never,exit -F dir=/var/www/ -F uid=www-data
-w / -F uid=www-data -k apache_alert
SSH сервис
Немного забегая вперёд, стоит отметить, что правила по обнаружению инцидентов возникающих через ssh-сервис очень схожи с правилами по расследованию инцидентов, поэтому многие правила из этого раздела можно использовать и при расследовании инцидентов. Это связано со спецификой самого ssh-сервиса.
SSH. Сканирование сервиса
Данный этап идентичен с этапом для HTTP-сервиса, поэтому его пропускаем.
SSH. Атака сервиса
Учитывая назначение сервиса, объеденим атаку и постэксплуатацию - поскольку взлом самого сервиса маловероятен, сфокусируемся на случае, когда злоумышленник получил доступ к сервису "популярным" способом - узнал пароль или ключ.

1. Определим действия. Рассматривать атакую перебор учетных данных сервиса не будем, т.к. есть более удобные инструменты для отслеживания таких атак, однако стоит отметить, что по умолчанию в журналы аудиты попадают все события связанные с попытками логина и аутентификации. Пример неудачной попытки ввода учетных данных:

time->Fri Aug 13 14:45:27 2021
type=USER_LOGIN msg=audit(1628855127.918:860): pid=8622 uid=0 auid=4294967295 ses=4294967295 msg='op=login acct=28756E6B6E6F776E207573657229 exe="/usr/sbin/sshd" hostname=? addr=192.168.0.104 terminal=sshd res=failed'
---
time->Fri Aug 13 14:45:27 2021
type=USER_LOGIN msg=audit(1628855127.918:861): pid=8622 uid=0 auid=4294967295 ses=4294967295 msg='op=login acct=28696E76616C6964207573657229 exe="/usr/sbin/sshd" hostname=? addr=192.168.0.104 terminal=sshd res=failed'
---
time->Fri Aug 13 14:45:30 2021
type=USER_AUTH msg=audit(1628855130.974:862): pid=8622 uid=0 auid=4294967295 ses=4294967295 msg='op=PAM:authentication acct="testy" exe="/usr/sbin/sshd" hostname=192.168.0.104 addr=192.168.0.104 terminal=ssh res=failed'
---
time->Fri Aug 13 14:45:30 2021
type=USER_LOGIN msg=audit(1628855130.974:863): pid=8622 uid=0 auid=4294967295 ses=4294967295 msg='op=login acct=28696E76616C6964207573657229 exe="/usr/sbin/sshd" hostname=? addr=192.168.0.104 terminal=sshd res=failed'

SSH изначально предполагает подключение пользователей и предоставление им возможности выполнять команды, поэтому обнаружение подозрительной активности становится весьма нетривиальной задачей. В зависимости от предназначения сервера, наборы подозрительных действий могут отличаться. Так если сервер предназначен, например, для веб-разработки, то маловероятно, что пользователям нужны утилиты связанные с дампом трафика. Можно предположить, что злоумышленника будет интересовать сбор сведений о системе, поиск ценных данных о проекте (компании), повышение привилегий в системе. Для начала можно пройтись по спискам Linux Privilege Escalation, ознакомиться с основными способами сбора информации там и настроить правила для них. Затем, исходя из специфики сервера, необходимо добавить дополнительные правила.

2. Поскольку правил может быть очень много, рассмотрим основные. Много полезных примеров для этого раздела я нашёл здесь. Интересно, что auditd с собой несёт тоже несколько примеров наборов правил, есть даже для pci-dss-v31:

$ ls /usr/share/doc/auditd/examples/rules
10-base-config.rules 12-cont-fail.rules 21-no32bit.rules 30-nispom.rules.gz 31-privileged.rules 41-containers.rules 70-einval.rules README-rules
10-no-audit.rules 12-ignore-error.rules 22-ignore-chrony.rules 30-pci-dss-v31.rules.gz 32-power-abuse.rules 42-injection.rules 71-networking.rules
11-loginuid.rules 20-dont-audit.rules 23-ignore-filesystems.rules 30-stig.rules.gz 40-local.rules 43-module-load.rules 99-finalize.rules

Для каждого события будем добавлять строку "susp_", по этому ключу можно будет найти любые подозрительные действия в системе, а также удобно настраивать поиск в подсистеме анализа.
Начнём с логирования действий над самим инструментом:
-w /etc/audit/ -p wa -k susp_auditconfig
-w /etc/audisp/ -p wa -k susp_audispconfig
-w /sbin/auditctl -p x -k susp_audittools
-w /sbin/auditd -p x -k susp_audittools
-w /usr/sbin/augenrules -p x -k susp_audittools
-w /var/log/audit/ -k susp_auditlog
Запись или изменение прав файлов планировщика задач:
-w /etc/cron -p wa -k cron_change
-w /etc/crontab -p wa -k cron_change
-w /etc/cron.allow -p wa -k cron_change
-w /etc/cron.d -p wa -k cron_change
-w /etc/cron.deny -p wa -k cron_change
-w /etc/cron.daily -p wa -k cron_change
-w /etc/cron.hourly -p wa -k cron_change
-w /etc/cron.monthly -p wa -k cron_change
-w /etc/cron.weekly -p wa -k cron_change
-w /etc/anacrontab -p wa -k cron_change
-w /var/spool/cron -p wa -k cron_change
-w /var/spool/cron/crontabs/root -p wa -k cron_change
Интересным кажется вариант с логированием выполнения команды sudo всеми пользователями, кроме того (тех), кому такие права не выданы:
-w /usr/bin/sudo -F auid!=<имя пользователя> -k susp_sudo
Важно наблюдать за работой сетевых утилит:
-w /sbin/iptables -p x -k susp_netutil
-w /sbin/ip6tables -p x -k susp_netutil
-w /sbin/ifconfig -p x -k susp_netutil
-w /usr/sbin/arptables -p x -k susp_netutil
-w /usr/sbin/ebtables -p x -k susp_netutil
-w /sbin/xtables-nft-multi -p x -k susp_netutil
-w /usr/sbin/nft -p x -k susp_netutil
Особый интерес представляют события, связанные с различными нарушениями доступа пользователей: при чтении, записи, изменении файлов:
File Access
Unauthorized Access (unsuccessful)
-a always,exit -F arch=b32 -S creat -S open -S openat -S open_by_handle_at -S truncate -S ftruncate -F exit=-EACCES -F auid>=1000 -F auid!=-1 -k file_access
-a always,exit -F arch=b32 -S creat -S open -S openat -S open_by_handle_at -S truncate -S ftruncate -F exit=-EPERM -F auid>=1000 -F auid!=-1 -k file_access
-a always,exit -F arch=b64 -S creat -S open -S openat -S open_by_handle_at -S truncate -S ftruncate -F exit=-EACCES -F auid>=1000 -F auid!=-1 -k file_access
-a always,exit -F arch=b64 -S creat -S open -S openat -S open_by_handle_at -S truncate -S ftruncate -F exit=-EPERM -F auid>=1000 -F auid!=-1 -k file_access

Unsuccessful Creation
-a always,exit -F arch=b32 -S creat,link,mknod,mkdir,symlink,mknodat,linkat,symlinkat -F exit=-EACCES -k file_creation
-a always,exit -F arch=b64 -S mkdir,creat,link,symlink,mknod,mknodat,linkat,symlinkat -F exit=-EACCES -k file_creation
-a always,exit -F arch=b32 -S link,mkdir,symlink,mkdirat -F exit=-EPERM -k file_creation
-a always,exit -F arch=b64 -S mkdir,link,symlink,mkdirat -F exit=-EPERM -k file_creation

Unsuccessful Modification
-a always,exit -F arch=b32 -S rename -S renameat -S truncate -S chmod -S setxattr -S lsetxattr -S removexattr -S lremovexattr -F exit=-EACCES -k file_modification
-a always,exit -F arch=b64 -S rename -S renameat -S truncate -S chmod -S setxattr -S lsetxattr -S removexattr -S

lremovexattr -F exit=-EACCES -k file_modification
-a always,exit -F arch=b32 -S rename -S renameat -S truncate -S chmod -S setxattr -S lsetxattr -S removexattr -S lremovexattr -F exit=-EPERM -k file_modification
-a always,exit -F arch=b64 -S rename -S renameat -S truncate -S chmod -S setxattr -S lsetxattr -S removexattr -S lremovexattr -F exit=-EPERM -k file_modification
Обратите внимание, что правила написаны для системных вызовов разных архитектур отдельно.
Как показала практика, у таких правил часто возникают ложные срабатывания, в таким случае их можно изменить, добавив срабатывание только на пользовательские действия. После опции -S <системный вызов> необходимо добавить:
-F auid>=1000 -F auid!=-1
Результат: SSH-сервис, на мой взгляд, один из самых сложных сервисов для обнаружения инцидента при помощи auditd, поэтому для него должны быть использованы надежные средства защиты, которые не позволят злоумышленнику получить доступ к серверу, . Созданные на этом этапе правила во многом могут быть использованы для расследования инцидентов о чем будет сказано далее.

Практическая работа № 13 Проведение расследования инцидента

Задание:

Изучите выданные дампы памяти и снимки состояния ОС. Обнаружьте инцидент и дайте полное описание с указанием времени и используемых нарушителем средств.

Оформите отчет об инциденте в правильном формате.

Практическая работа № 14 Анализ сетевого трафика

Задание:

Изучите выданные дампы сетевой памяти. Проведите анализ сетевого трафика и выявите флаги в каждом. Заполните таблицу.

Название файла	Флаг

Практическая работа № 15 Обнаружение действий нарушителя

Задание:

Изучите выданные дампы сетевой памяти. Опишите полную последовательность действий нарушителя, какие протоколы были использованы, изучите последствия и дайте рекомендации по устранению.

Практическая работа № 16 Поиск подключений по SSH

Задание:

Изучите выданные дампы сетевой памяти. Выявите подключение злоумышленника используя фазы установления сессии.
установление SSH соединения проходит через несколько этапов:

Version negotiation -> algorithm negotiation -> key exchange -> user authentication -> and session exchange

Это очень красивые слова и понятия, да. Но что мы будем с этим делать, если сессия по какой-то причине не будет устанавливаться, как в этом случае эти знания помогут нам найти причину сбоя? Для этого мы должны уметь траблшутить и понимать эти понятия изнутри, как они работают и выглядят через призму разных инструментов для поиска неисправностей. В этом примере я буду искать ответы на эти вопросы через Wireshark и debugging на CloudEngine.

Чтобы собрать данные на CE включил дебагинг ssh пакетов таким образом:
debugging ssh server packet
terminal monitor
terminal debugging

И из linux терминала ввел команду, чтобы подключиться к CE с адресом 7.7.7.4. Сразу скажу, что у меня на CE настроена аутентификацию по паролю, а не по приватному ключу - **ssh user client authentication-type password**.
"

vasyotube@DESKTOP-1V1AP4E:~$ ssh ClimbeR2022@7.7.7.4
User Authentication
(ClimbeR2022@7.7.7.4) Enter password:
"
Пароль я пока вводить не стал - посмотрим, что уже произошло на этом этапе...

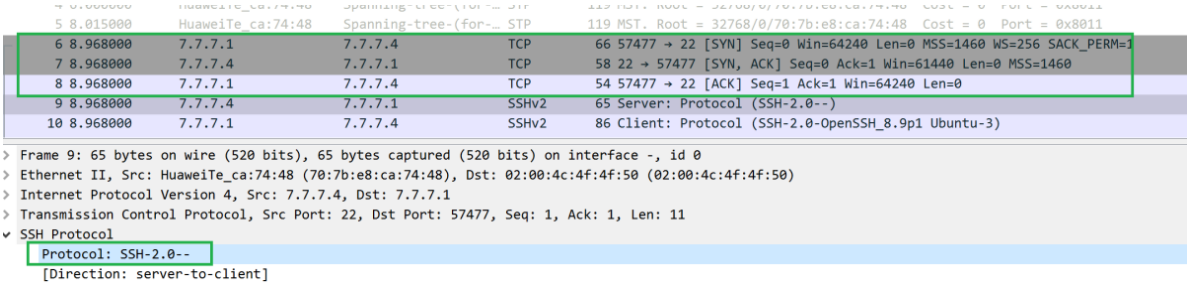
1 Фаза. Согласование версии протокола SSH.
В логах на CE первыми мы увидим два сообщения:

<Gokyo>
Jan 23 2023 15:17:19.435 Gokyo %SSHS/7/SSHS_DBG_**SSH_VERSION_SENT**(d):CID=0x809
3271e;SSH protocol version string sent. (Channel Id = 135169, Version String = S
SH_VERSION).

Jan 23 2023 15:17:19.438 Gokyo %SSHS/7/SSHS_DBG_**SSH_VERSION_RECEIVED**(d):CID=0
x8093271e;SSH protocol version string received. (Channel Id = 135169, Version St
ring = **SSH-2.0-OpenSSH_8.9p1 Ubuntu-3**
).

Первое из них сообщит о том, что сервер (наш CE коммутатор) отправил свою версию SSH (при этом самой версии не указано), а во втором, что SSH клиент отправил (а наш коммутатор принял) версию SSH, которую поддерживает клиент - **SSH-2.0-OpenSSH_8.9p1 Ubuntu-3 (+операционная система).**

A Wireshark?



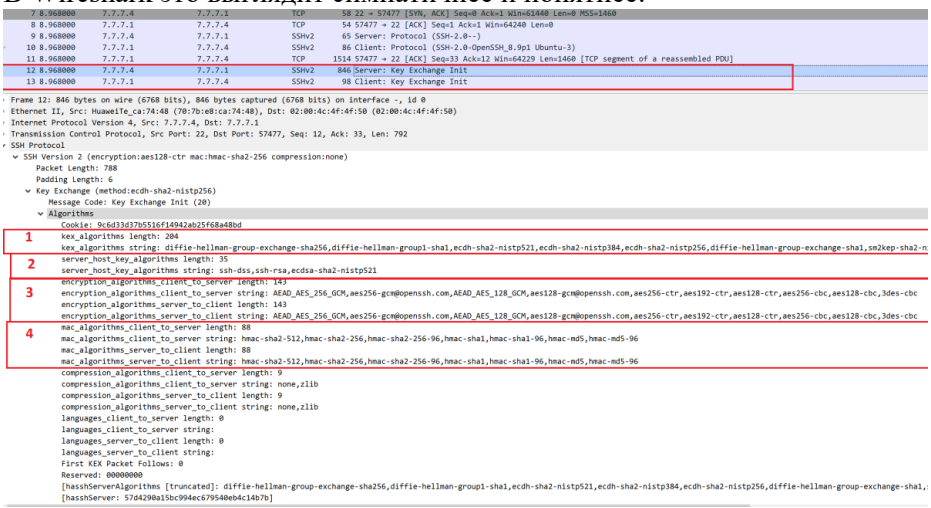
Здесь у нас первые три сообщения - это установление TCP сессии, а затем следующие два - это именно те самые Version negotiation сообщения. И здесь мы видим, что сервер (наш коммутатор) указывает, что версия у него SSHv2.

2 Фаза. Согласование алгоритма протокола SSH.
Здесь у нас вступает в игру тип пакета SSH2_MSG_KEXINIT, что значит KEY EXCHANGE INIT (инициация обмена ключа). Сервер и клиент сообщают друг другу какие алгоритмы шифрования поддерживают.
Четыре алгоритма должно быть представлено:
1.Алгоритм обмена ключами.
2.Публичный алгоритм ключа.
3.Симметричный алгоритм шифрования
4.Алгоритм аутентификации сообщения

Jan 23 2023 15:17:19.438 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093
271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = **SSH2_MSG_KEXI
NIT)**.
1:
Jan 23 2023 15:17:19.438 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093
271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = SSH2_MSG_KEXI
NIT: **key_ex(diffie-hellman-group-exchange-sha256,diffie-hellman-group1-sha1,ecdh
-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchan
ge-sha1,sm2kep-sha2-nistp256,diffie-hellman-group14-sha1)).**
2:

Jan 23 2023 15:17:19.438 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = **SSH2_MSG_KEXINIT: ser_host_key(ssh-dss,ssh-rsa,ecdsa-sha2-nistp521),**
3:
ciph_ctos(AEAD_AES_256_GCM,aes256-gcm@openssh.com,AEAD_AES_128_GCM,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes128-cbc,3des-cbc),
ciph_stoc(AEAD_AES_256_GCM,aes256-gcm@openssh.com,AEAD_AES_128_GCM,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes128-cbc,3des-cbc)).
4:
Jan 23 2023 15:17:19.438 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = **SSH2_MSG_KEXINIT: hmac_ctos(hmac-sha2-512,hmac-sha2-256,hmac-sha2-256-96,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-md5-96), hmac_stoc(hmac-sha2-512,hmac-sha2-256,hmac-sha2-256-96,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-md5-96), compress_ctos(none,zlib), compress_stoc(none,zlib)).**

В Wireshark это выглядит симпатичнее и понятнее:



3 Фаза. Обмен ключами.

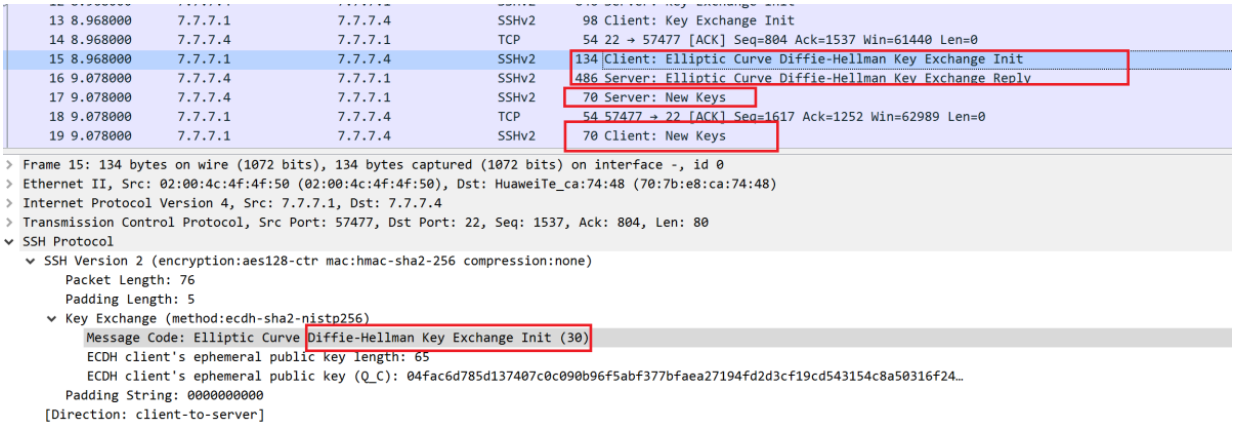
Для обмена ключами используется алгоритм обмена ключами Диффи-Хеллмана, который основан на математическом дискретном логарифме и не описан в этом курсе. Во время обмена ключами закрытые ключи не передаются, и из-за сложности вычисления дискретных логарифмов они не могут быть расшифрованы другими пользователями. Это обеспечивает конфиденциальность сеансовых ключей. При этом открытый и закрытый ключи, сгенерированные на этом этапе, используются только для генерации сеансовых ключей и не имеют отношения к последующей аутентификации пользователя. После завершения фазы обмена ключами все последующие пакеты шифруются на основе сеансовых ключей.

Jan 23 2023 15:17:19.441 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x8093271e;SSH protocol packet received. (Channel Id = 135169, Packet Type = **SSH2_MSG_KEX_DH_GEX_INIT**).
Jan 23 2023 15:17:19.537 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x8093271e;SSH protocol packet received. (Channel Id = 135169, Packet Type = SSH2_MSG_KEX_DH_GEX_INIT_THREAD).

Jan 23 2023 15:17:19.537 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = **SSH2_MSG_KEXDH_REPLY**).

Jan 23 2023 15:17:19.537 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093271e;SSH protocol packet **sent**. (Channel Id = 135169, Packet Type = **SSH2_MSG_NEWKEYS**).

Jan 23 2023 15:17:19.541 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x8093271e;SSH protocol packet **received**. (Channel Id = 135169, Packet Type = **SSH2_MSG_NEWKEYS**)



4 Фаза. Аутентификация пользователя.

Интересная фаза. Здесь мы, наконец, подходим к моменту, когда нужно ввести пароль для нашего пользователя.

Клиент отправляет сообщение MSG_USERAUTH_REQUEST, в котором указано имя пользователя.

Jan 23 2023 16:56:52.932 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x8093271e;SSH protocol packet received. (Channel Id = 135168, Packet Type = SSH2_MSG_USERAUTH_REQUEST:(UserName: **ClimbeR2022**, Service: ssh-connection, Method: none)).

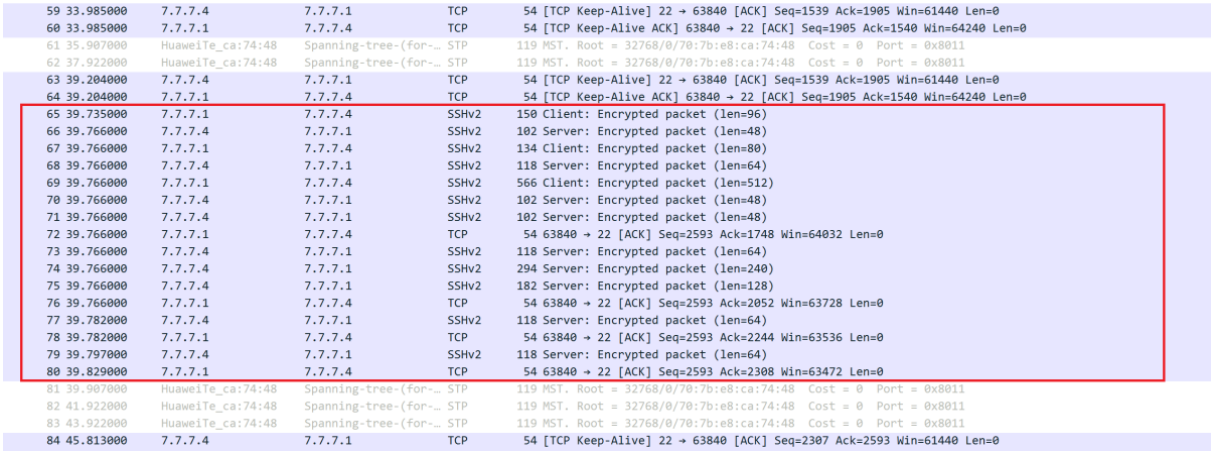
А вот пароль передается в отдельном сообщении после того, как оно будет введно нами с клавиатуры:

Jan 23 2023 16:57:27.511 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x8093271e;SSH protocol packet received. (Channel Id = 135168, Packet Type = **KEYBOARD_INTERACTIVE_INFO_RESPONSE**).

И если пароль совпадает с тем, что указано в конфигурации CE, то сервер генерируется сообщение об успешной аутентификации:

Jan 23 2023 16:57:27.538 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093271e;SSH protocol packet sent. (Channel Id = 135168, Packet Type = **SSH2_MSG_USERAUTH_SUCCESS**).

Ничего этого в Wireshark мы не увидим, потому что вся информаци зашифрована:



5 Фаза. Установление сессии.

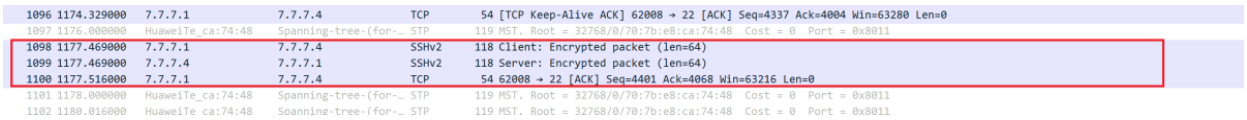
Наконец, мы можем установить сессию: два сообщения подтверждают это - OPEN и CONFIRMATION.

Jan 23 2023 16:57:27.540 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x 8093271e;SSH protocol packet received. (Channel Id = 135168, Packet Type = **SSH2_ MSG_CHANNEL_OPEN**).

Jan 23 2023 16:57:27.540 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093 271e;SSH protocol packet sent. (Channel Id = 135168, Packet Type = **SSH2_MSG_CHAN NEL_OPEN_CONFIRMATION**).

В Wireshark мы этих сообщений тоже не найдем. Зато теперь всякий раз, когда мы даже если будем нажить Enter в CLI, находясь в SSH сессии, у нас будет происходить обмен данных между клиентом и сервером:

<Gokyo> (Enter)
<Gokyo> (Enter)
<Gokyo> (Enter)
<Gokyo> (Enter)
<Gokyo> (Enter)
<Gokyo> (Enter)



Jan 23 2023 17:16:16.711 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x 8093271e;SSH protocol packet received. (Channel Id = 135169, Packet Type = SSH2_ MSG_CHANNEL_DATA).

Jan 23 2023 17:16:16.711 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093 271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = 94).

Jan 23 2023 17:16:16.711 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093 271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = SSH2_MSG_CHAN NEL_DATA).

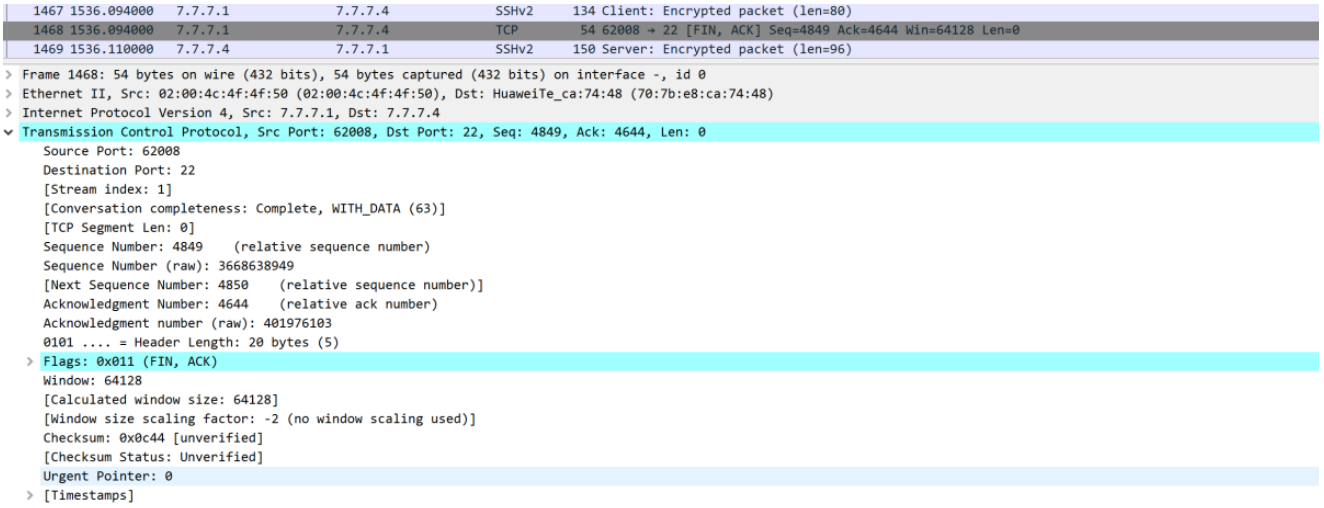
Jan 23 2023 17:16:25.246 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x 8093271e;SSH protocol packet received. (Channel Id = 135169, Packet Type = 94).

Jan 23 2023 17:16:25.246 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_RECEIVED(d):CID=0x 8093271e;SSH protocol packet received. (Channel Id = 135169, Packet Type = SSH2_ MSG_CHANNEL_DATA).

Jan 23 2023 17:16:25.247 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093 271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = 94).

Jan 23 2023 17:16:25.247 Gokyo %SSHS/7/SSHS_DBG_SSH_PACKET_SENT(d):CID=0x8093 271e;SSH protocol packet sent. (Channel Id = 135169, Packet Type = SSH2_MSG_CHAN NEL_DATA).

Завершение TCP сессии после вводы команды quit:

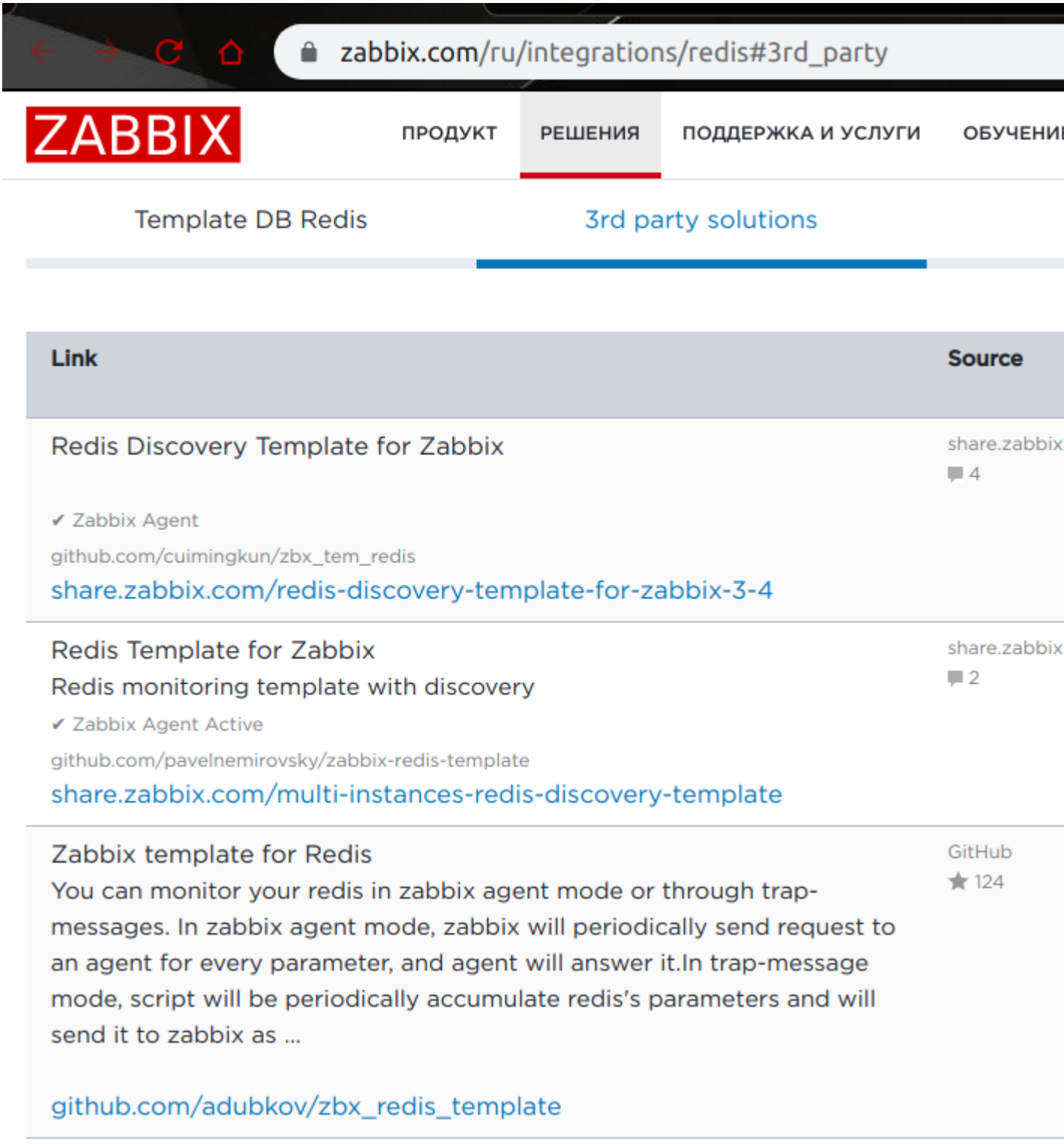


Отключаем дебаг:

undo debugging all
undo t d
undo t m

Практическая работа № 17 Настройка Zabbix для сбора данных

Задание:
Настройте сервер мониторинга для сбора данных с redis.
Сам Zabbix предлагает свой собственный плагин для мониторинга состояния Redis'a, но на моей версии сервера (4.2.8) мне не удалось его задействовать (плагин для версии 4.4 и выше). Также предлагаются решения от третьих лиц (около десятка вариантов под различные версии Zabbix'a, на картинке только первых три):



Каждый из них обладал своими плюсами-минусами, пришлось заглянуть внутрь, чтобы выбрать. Лучшим, на мой взгляд, оказался плагин Shakeeljaveed/zabbix-redis-userparamaters, состоявший из двух файлов:

- README.md
- redis-userparameter.conf

Немножко пришлось поработать "ручками", но зато на его примере стало чуть понятнее, как данные от агента попадают на сервер. По предложению автора Javeed Shakeel состояние Redis'а каждые 2 минуты сбрасывалось кроном в файл /tmp/redismetric:

```
*/2 * * * * /usr/bin/redis-cli info > /tmp/redismetric
```

А затем каждый параметр мониторинга извлекался агентом из файла /tmp/redismetric при помощи средств самой операционной системы. Инструкции для этого размещались в конфигурации Zabbix-агента /etc/zabbix/zabbix_agent.conf.d/userparameter_redis.conf. Например, вот так выглядят инструкция для извлечения параметра used_memory (использование памяти Redis-сервером):

```
UserParameter=used_memory,grep -w 'used_memory' /tmp/redismetric | cut -d: -f2
```

То есть, в файле /tmp/redismetric с выводом redis-cli INFO по ключу used_memory ищется строка (grep -w ...)

```
used_memory:7153216
```

которая затем разбивается на столбцы по разделителю ":" (cut -d: -f2). На выходе агент получает число 7153216 и присваивает его параметру used_memory.

Остаётся через web-интерфейс настроить сервер, чтобы он периодически отправлял запросы агенту на получение данных по параметру used_memory, после чего данные начинают литься на сервер, сохраняться в базе, по ним можно строить графики и создавать триггера, реагирующие на изменения этого параметра.

Цель

Задачей мониторинга состояния любой системы является не только сбор статистики, но и предупреждение о возникновении ситуаций, требующих вмешательства человека. Так как с Redis'ом я работаю на уровне очень начинающего пользователя, то пришлось поискать информацию, на какие параметры "здоровья" обращать внимание и что они значат. Наиболее достойной показалась статья "6 Crucial Redis Monitoring Metrics You Need To Watch". Проанализировав её, я пришёл к выводу, что "для полного счастья" мне нужно собирать данные для обнаружения следующих событий:

- Memory fragmentation: used_memory_rss / used_memory > 1.5
- Low cache hit ratio: (keyspace_hits)/ (keyspace_hits + keyspace_misses) < 0.8
- Rejected connections: rejected_connections > 0
- Evicted keys: evicted_keys > 0

Также я хотел собирать статистику по дополнительным параметрам (версия Redis'а, uptime и т.п.). В общем, имея общее представление о том, каким образом данные собираются агентом и передаются на сервер, "хотелки" можно сильно не ограничивать. В итоге получился список параметров для мониторинга из 12 позиций.

Создание собственного плагина

Параметры мониторинга

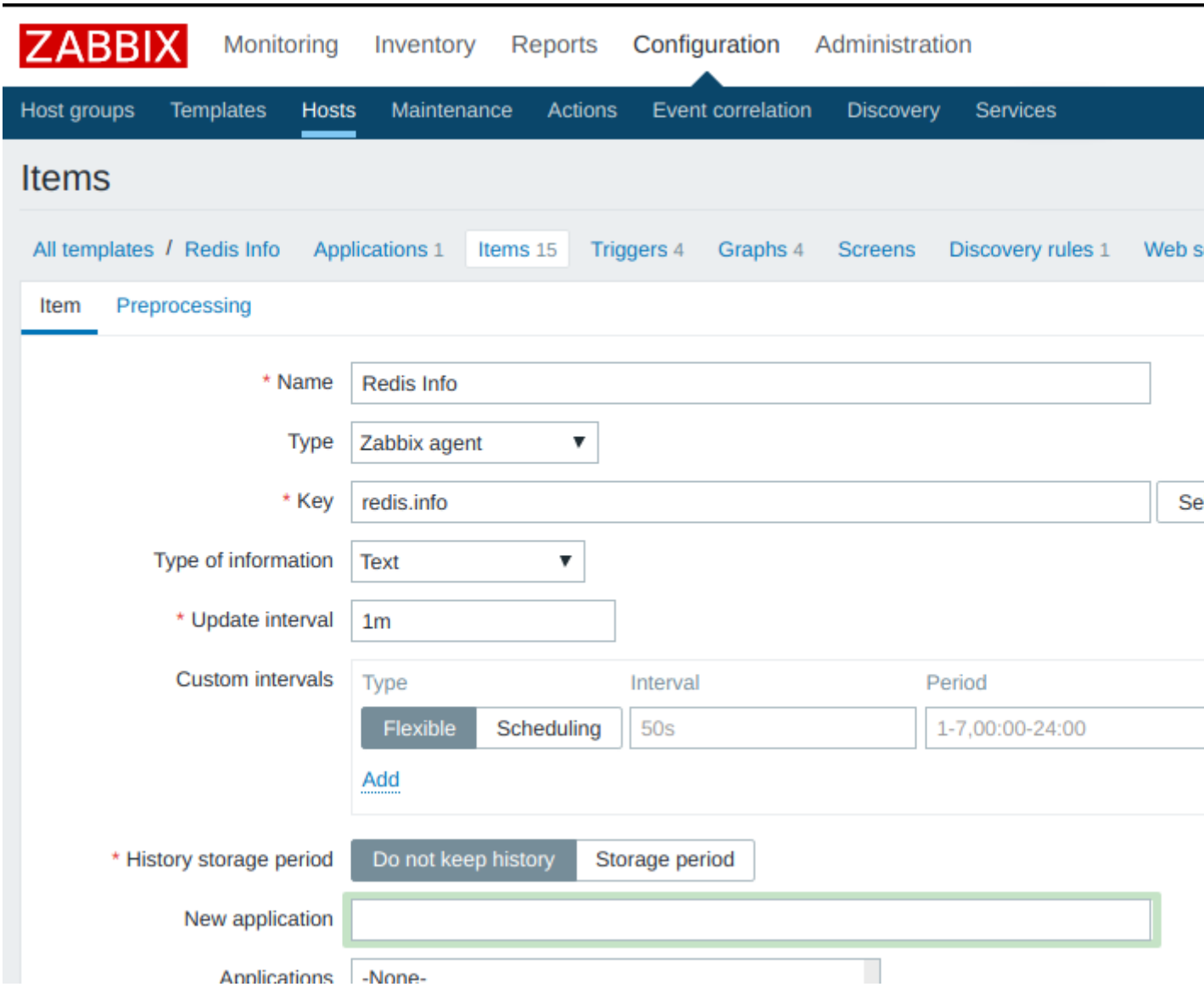
Плагин, который я анализировал, предполагал выполнение отдельной команды для получения отдельного параметра (элемента данных, item'а):

```
grep -w 'used_memory' /tmp/redismetric | cut -d: -f2
```

Т.е., для получения данных по 12 параметрам агент должен будет 12 раз выполнить различные наборы команд. А если мне нужно мониторить параметры, которые сложно извлечь цепочкой команд и нужно будет писать отдельный shell-скрипт или полноценную программу? Для таких "хотелок" Zabbix предлагает вариант с зависимыми элементами данных. Суть его в том, что на стороне агента скриптом формируется набор данных (например, в формате JSON), который передаётся на сервер в виде строкового параметра. Затем на стороне сервера происходит разбор полученных данных и вычленение из них отдельных элементарных параметров.

Основной элемент данных

Я описал основной элемент данных redis.info строкового типа с периодом обновления в 1 мин., без сохранения истории изменений:



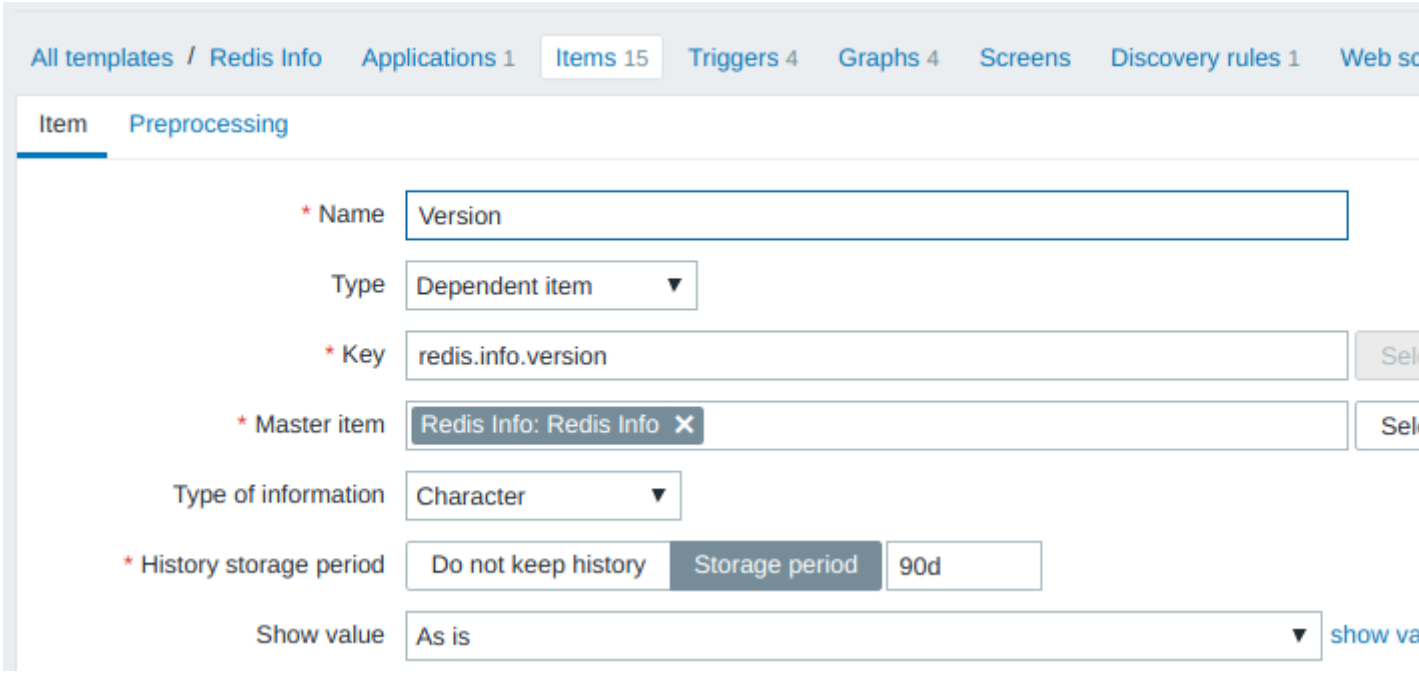
Предположительно, на стороне агента должен генерироваться такой JSON:


```
{
 "version": "4.0.9",
 "uptime": 1897336,
 "used_memory": 1054943416,
 "used_memory_rss": 1138159616,
 "keyspace_hits": 75810274,
 "keyspace_misses": 13545949,
 "connected_clients": 15,
 "rdb_last_save_time": 1580126954,
 "total_connections_received": 1258614,
 "rejected_connections": 0,
 "expired_keys": 60270,
 "evicted_keys": 0
}
```

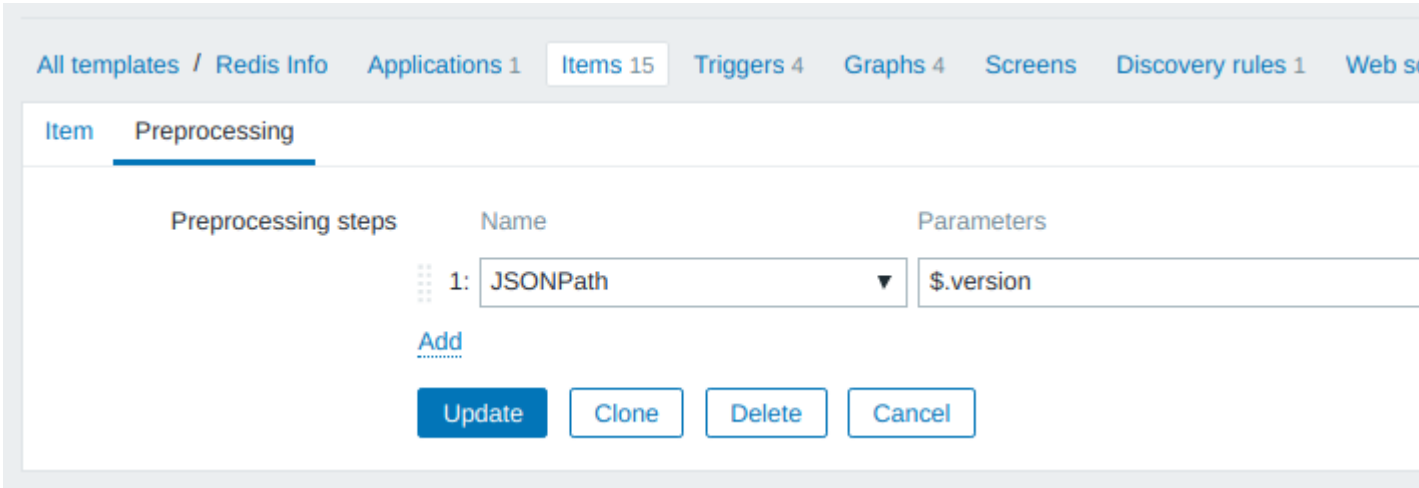

после чего этот текст должен попадать на сервер в виде элемента данных redis.info, но не сохраняться, а служить базой для других элементов данных (параметров мониторинга).

Зависимый элемент данных

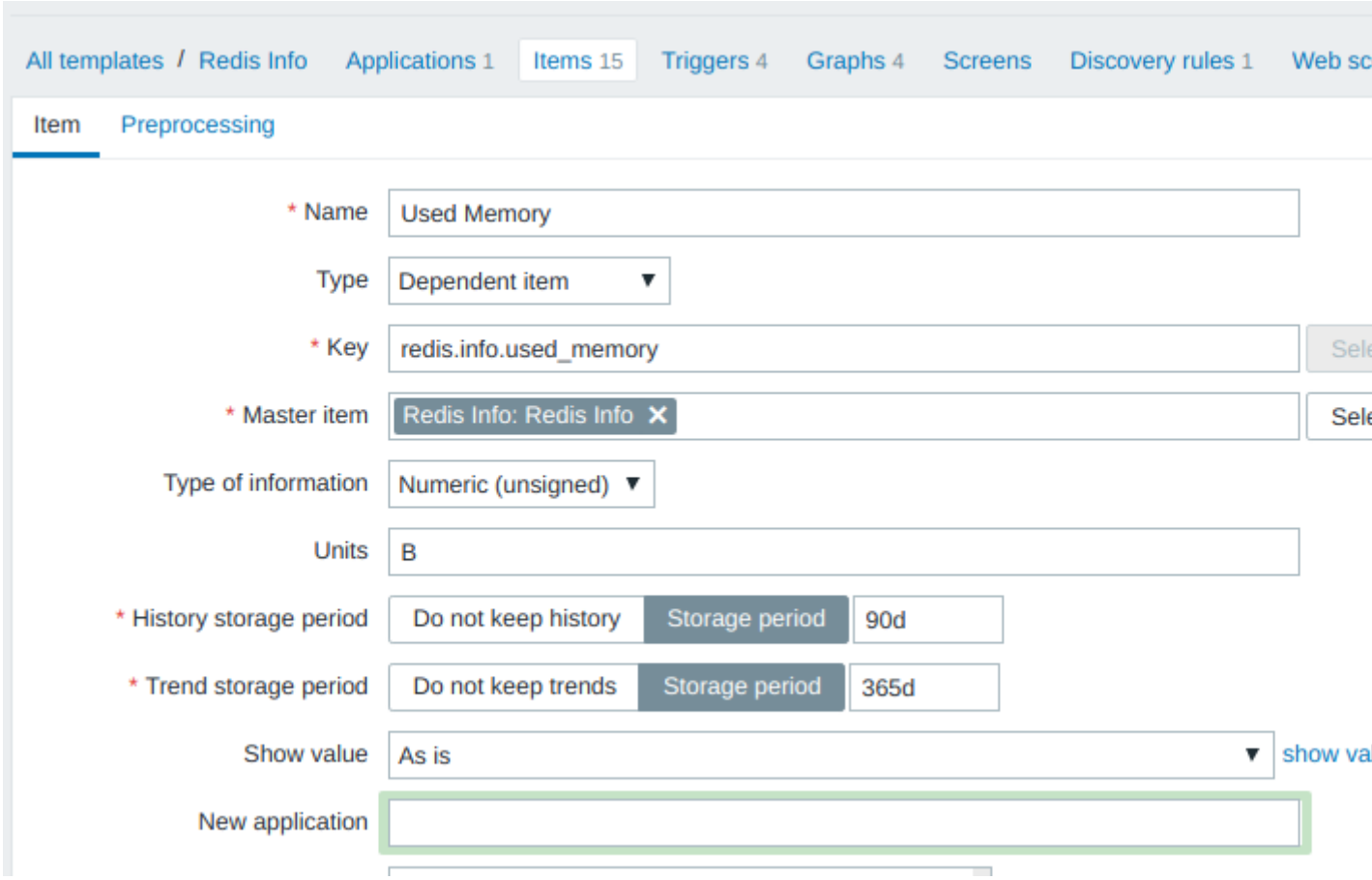
Тестовый параметр redis.info.version зависит от redis.info и сохраняет свои значения в базе в течение 90 дней. Периодичность мониторинга параметра зависит от базового элемента (redis.info):



Значение параметра redis.info.version извлекается из значения redis.info при помощи инструкций JSONPath:



По аналогичной схеме описываются остальные зависимые элементы данных (параметры мониторинга), которые передаются в виде JSON'a. Вот пример описания числового параметра redis.info.used_memory:

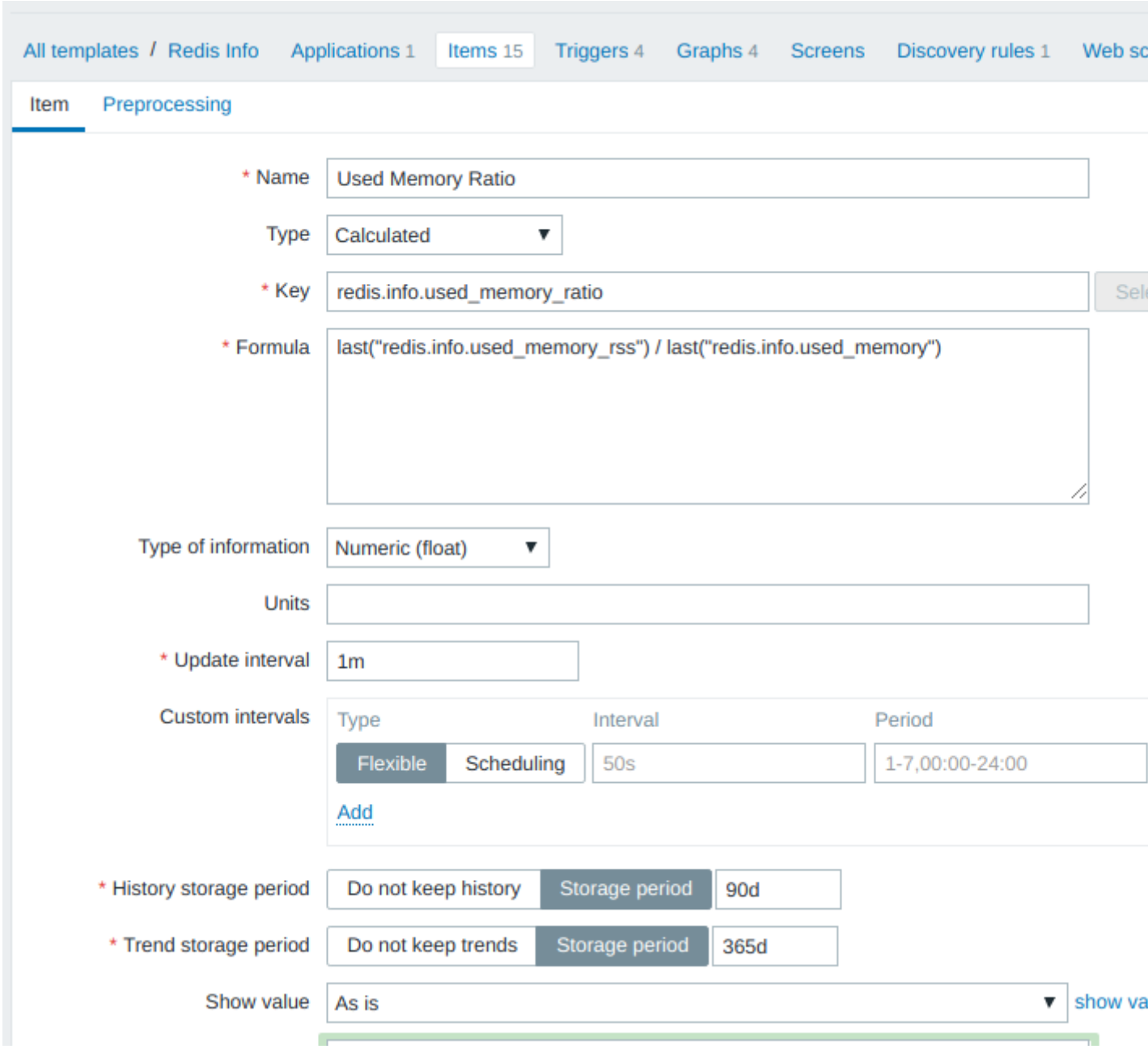


Всё достаточно прозрачно, за исключением Units и Trend storage period. Со вторым пунктом я не разбирался, оставил по-умолчанию, а единицы измерения объяснены в документации. В данном случае значение redis.info.used_memory измеряется в байтах и в web-интерфейсе сворачивается до кило/мега/гига/...-байт.

Формула для извлечения значения из JSON'a: JSONPath = $.used_memory

Вычисляемый элемент данных

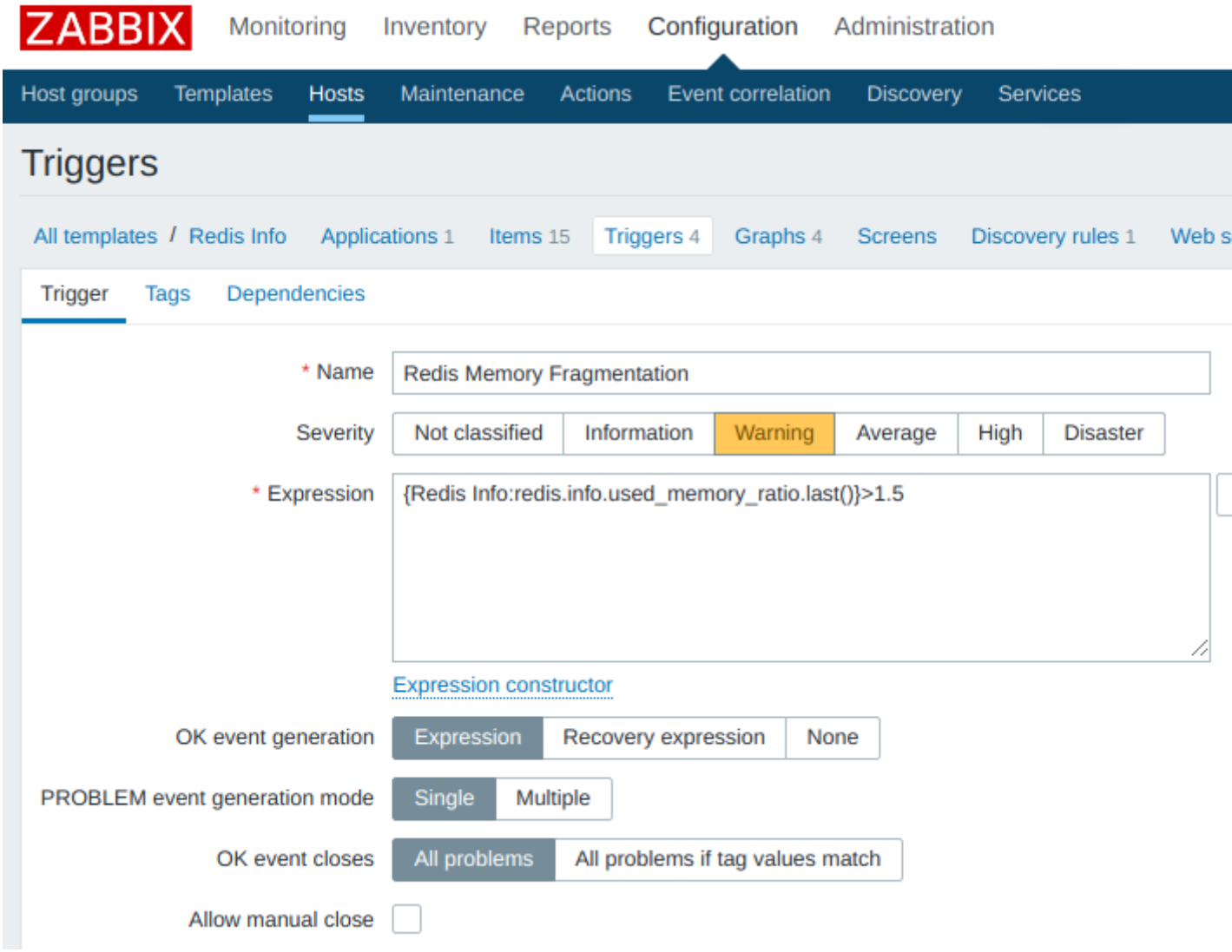
Для вычисления фрагментации памяти используется отношение used_memory_rss / used_memory и на его базе определяется триггер, срабатывающий при превышении отношением значения 1.5. В Zabbix'e есть вычисляемый тип элементов данных:



Значение для параметра redis.info.used_memory_ratio вычисляется каждую минуту на основании последних значений двух других параметров (redis.info.used_memory_rss и redis.info.used_memory), сохраняется в базе в течение 90 дней и т.д.

Триггеры

Вот пример триггера, срабатывающего при излишней фрагментации памяти:



Ничего необычного, за исключением формата выражений, используемого в формуле изменения состояния триггера. В Zabbix'e есть конструктор форм, можно воспользоваться им или обратиться к документации/примерам (список триггеров доступен через web-интерфейс по адресу "*Configuration / Templates / ${TemplateName} / Triggers*").

Триггер может базироваться на любых элементах данных (item'ax) вне зависимости от их типа (основной, зависимый, вычисляемый).

Настройка агента

Генерация JSON'a

Для получения значений параметров мониторинга и формирования JSON'a я использую вот такой shell-скрипт:

```
#!/bin/bash
## ============================================================================
#   Script to generate Redis monitoring data in JSON format
#   for 'zabbix-agent'.
```

```
## ============================================================================
# collect working variables/data
BIN_REDIS=$(whereis -b redis-cli | cut -d" " -f2) # /usr/bin/redis-cli
DATA_INFO=$(${BIN_REDIS} INFO | tr -d '\r')       # get info and remove trailing '\r' from output

##
#   Extract stats and save it into env. vars.
##
# find lines with 'grep', cut second field after ":" and put it into env. variable:
ITEM_VERSION=$(grep "^redis_version:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_UPTIME=$(grep "^uptime_in_seconds:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_USED_MEMORY=$(grep "^used_memory:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_USED_MEMORY_RSS=$(grep "^used_memory_rss:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_KEYSPACE_HITS=$(grep "^keyspace_hits:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_KEYSPACE_MISSES=$(grep "^keyspace_misses:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_CONNECTED_CLIENTS=$(grep "^connected_clients:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_RDB_LAST_SAVE_TIME=$(grep "^rdb_last_save_time:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_TOTAL_CONNECTIONS_RECEIVED=$(grep "^total_connections_received:" <<<"${DATA_INFO}"
| cut -d: -f2)
ITEM_REJECTED_CONNECTIONS=$(grep "^rejected_connections:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_EXPIRED_KEYS=$(grep "^expired_keys:" <<<"${DATA_INFO}" | cut -d: -f2)
ITEM_EVICTED_KEYS=$(grep "^evicted_keys:" <<<"${DATA_INFO}" | cut -d: -f2)

# compose output JSON for Zabbix server:
echo -n "{"
echo -n "\"version\": \"${ITEM_VERSION}\","
echo -n "\"uptime\": ${ITEM_UPTIME},"
echo -n "\"used_memory\": ${ITEM_USED_MEMORY},"
echo -n "\"used_memory_rss\": ${ITEM_USED_MEMORY_RSS},"
echo -n "\"keyspace_hits\": ${ITEM_KEYSPACE_HITS},"
echo -n "\"keyspace_misses\": ${ITEM_KEYSPACE_MISSES},"
echo -n "\"connected_clients\": ${ITEM_CONNECTED_CLIENTS},"
echo -n "\"rdb_last_save_time\": ${ITEM_RDB_LAST_SAVE_TIME},"
echo -n "\"total_connections_received\": ${ITEM_TOTAL_CONNECTIONS_RECEIVED},"
echo -n "\"rejected_connections\": ${ITEM_REJECTED_CONNECTIONS},"
echo -n "\"expired_keys\": ${ITEM_EXPIRED_KEYS},"
echo -n "\"evicted_keys\": ${ITEM_EVICTED_KEYS}"
echo -n "}"
```

Этот скрипт я поместил в файл /var/lib/zabbix/user_parameter/redis/get_info.sh на сервере с Redis'ом, на котором уже установлен агент Zabbix'а. Пользователь, под которым запускается Zabbix-агент (обычно zabbix) должен иметь права на выполнение файла get_info.sh.

Файл userparameter_XXX.conf

На стороне агента дополнительные параметры мониторинга прописываются в файлах userparameter_*.conf в каталоге /etc/zabbix/zabbix_agentd.d. Поэтому для того, чтобы агент узнал о том, каким образом ему нужно собирать данные по параметру redis.info, я создал файл /etc/zabbix/zabbix_agentd.d/userparameter_redis.conf с таким содержимым:

```
UserParameter=redis.info,/var/lib/zabbix/user_parameter/redis/get_info.sh
```

Т.е., для получения данных по параметру redis.info агент должен запустить скрипт /var/lib/zabbix/user_parameter/redis/get_info.sh и передать на сервер результат выполнения. После рестарта Zabbix-агента (sudo service zabbix-agent restart) у него появляется возможность собирать данные для параметра redis.info и отправлять их на сервер.

UPDATE: коллега banzayats обратил внимание, что текстовые данные с хоста можно получить без создания промежуточного скрипта userparameter_*.conf — при помощи параметра "system.run" и проводить постпроцессинг уже на стороне zabbix-сервера.

Практическая работа № 18 Проведение расследования инцидента

Задание:

Изучите выданные дампы сетевой памяти. Обнаружьте инцидент и дайте полное описание с указанием времени и используемых нарушителем средств.

Оформите отчет об инциденте в правильном формате.