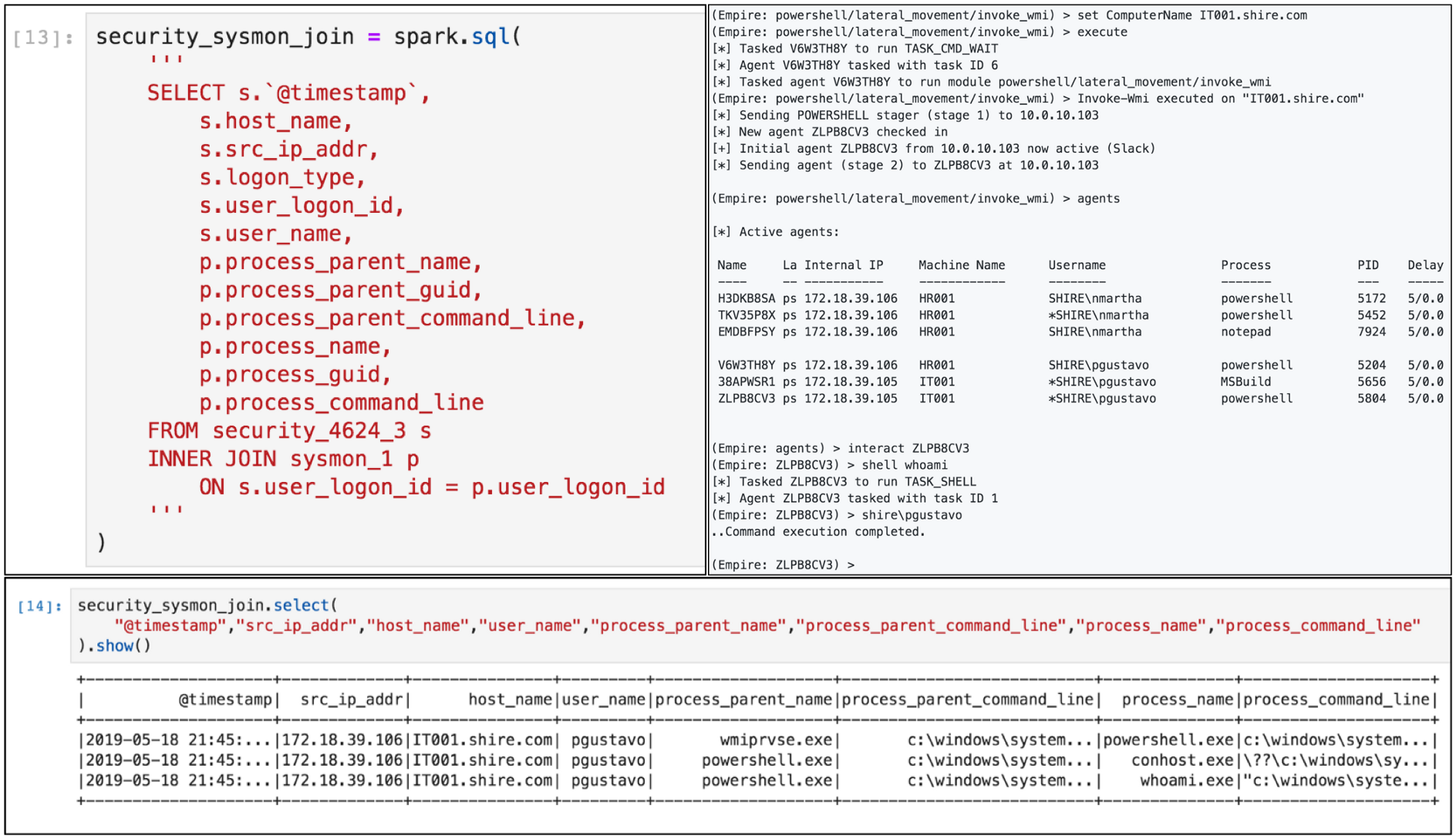
**Поиск угроз с помощью Jupyter Notebook.**

**Часть 4. SQL JOIN через Apache SparkSQL**

В предыдущей статье мы рассмотрели основы Apache Spark и несколько базовых классов PySpark SQL для чтения и загрузки данных из баз данных Elasticsearch.

В этой статье я покажу вам, как выполнять реляционные запросы через модуль Apache SparkSQL и API-интерфейсы Spark Python для объединения нескольких событий безопасности, которые могут предоставить интересный дополнительный контекст.

Эта статья является частью из пяти статей. Остальные четыре статьи можно найти по следующим ссылкам:

* Поиск угроз с помощью Jupyter Notebook. Часть 1. Ваш первый Notebook.
* Поиск угроз с помощью Jupyter Notebook. Часть 2. Базовый анализ данных с помощью Pandas.
* Поиск угроз с помощью Jupyter Notebook. Часть 3. Запросы Elasticsearch через Apache Spark.
* Поиск угроз с помощью Jupyter Notebook. Часть 5. Документирование, совместное использование и запуск книг-охотников за угрозами!

**Требования**

Предполагается, что вы прочитали предыдущую статью, и у вас есть сервер HELK, работающий с набором данных empire\_invoke\_wmi Mordor, которые хранятся в Elasticsearch (если вы этого не делали, вернитесь к предыдущей статье).

**Во-первых, данные Wiki**

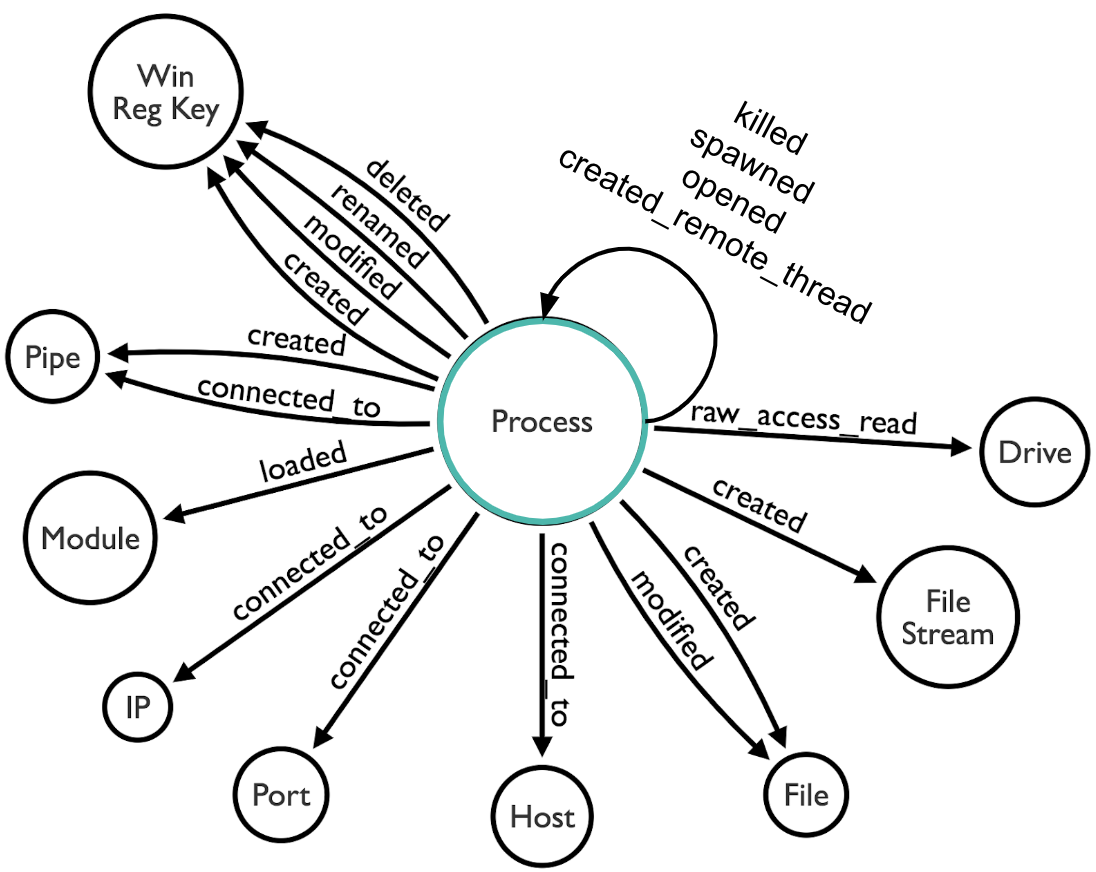
* Прежде чем мы начнём говорить о реляционных запросах и соединениях между конкретными событиями безопасности, я считаю, что очень важно для начала понять и задокументировать данные, которые вы собираете, и то, что они предоставляют, вплоть до полевого уровня. Делать всё сразу не нужно, но это инициатива, которая помогает аналитикам по безопасности по-настоящему понять свои пробелы и изучить их с точки зрения данных.
* Я считаю документирование журналов событий лучшим первым шагом в любой методологии, применяемой для разработки анализа данных и подготовки к поиску.
* Также с помощью этого шага вы сможете определять структуры данных и связи между источниками данных, о которых вы, возможно, не знали раньше. Это основы моделирования данных.

**Что такое модель данных?**

* Модель данных определяет структуру объектов данных, присутствующих в наборе данных, и взаимосвязи, идентифицируемые между собой.
* С точки зрения событий безопасности объекты данных могут быть определены как объекты, такие как «Пользователь», «Процесс», «Файл», «IP-адрес» или даже «Ключ реестра».
* Кроме того, они могут обладать такими свойствами, как «user\_name», «user\_domain», «process\_path» или «process\_name». В зависимости от структуры источников данных они могут иметь уникальные идентификаторы через такие свойства, как «process\_id» или «user\_logon\_id », которые помогут идентифицировать и сопоставить несколько событий, сопоставленных с одним и тем же объектом данных.

**Модель данных Sysmon**

* Часть моего первоначального исследования в рамках проекта OSSEM заключалась в документировании структуры данных журналов событий Windows Sysmon, а также существующих связей между примерно 25 событиями, которые он предоставляет.
* Sysmon предоставляет информацию о нескольких объектах данных, таких как «Процесс», «IP», «Файл», «Ключ реестра» и даже «Именованный канал».
* Кроме того, большинство их объектов данных имеют общее свойство с именем ProcessGUID, которое определяет прямые отношения между несколькими событиями Sysmon. По словам моих товарищей по команде Мэтта Гребера (Matt Graeber) и Ли Кристенсена (Lee Christensen), в их недавнем техническом документе «Subverting Sysmon», ProcessGUID — это уникальное значение, полученное из GUID машины, времени запуска процесса и идентификатора токена процесса, которые можно использовать для корреляции других связанных событий.
* После документирования отношений между объектами данных Sysmon на основе их свойства ProcessGUID возможно следующее:



**А как насчет аудита безопасности Windows?**

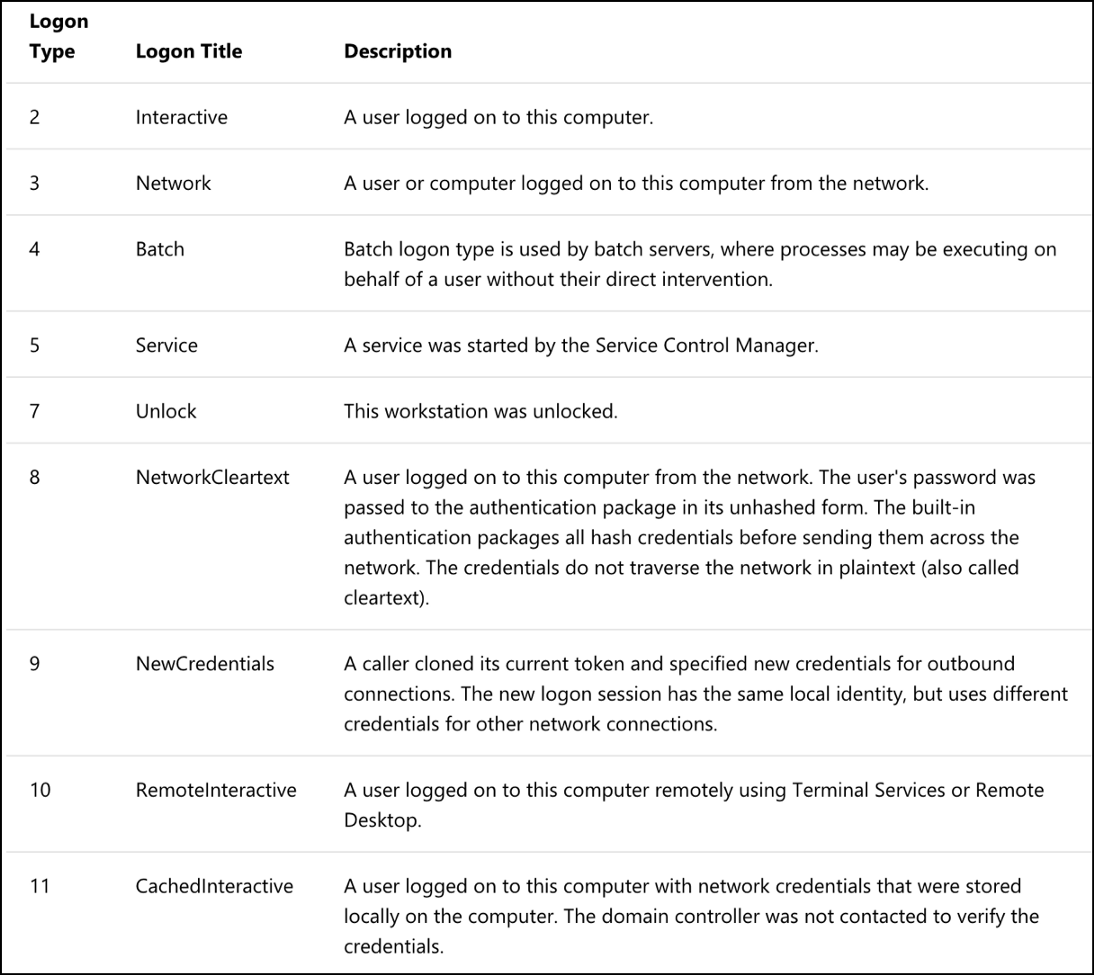
* Для событий аудита безопасности Windows не так просто придумать модель данных или изображение, как показано выше, из-за количества доступных событий, категорий аудита и подкатегорий. Это немного сложнее.
* Однако, как и в модели данных Sysmon, для большинства событий безопасности Windows существует общее свойство, известное как logon\_id, которое позволяет сопоставлять события, происходящие в одном сеансе входа.

**Что такое сеанс входа в систему?**

* Согласно документации Microsoft, сеанс входа в систему — это компьютерный сеанс, который начинается, когда аутентификация пользователя прошла успешно, и заканчивается, когда пользователь выходит из системы.
* При успешном опознании пользователя пакет аутентификации создаёт сеанс входа в систему и возвращает информацию в локальный орган безопасности (LSA), который используется для создания токена для нового пользователя. Этот токен включает, среди прочего, локально уникальный идентификатор (LUID) для сеанса входа в систему, называемый идентификатором входа.
* С точки зрения события безопасности, logon\_id видно сразу, как только происходит успешная аутентификация.
* На основе документации, предоставленной Microsoft и OSSEM, события из подкатегории Audit Logon связаны с созданием сеансов входа в систему и происходят на компьютере, к которому был получен доступ. Для интерактивного входа в систему события генерируются на компьютере, на котором выполнен вход. Для входа в сеть, например доступа к общему ресурсу, события генерируются на компьютере, на котором размещен ресурс, к которому был получен доступ.

**Событие 4624: учётная запись была успешно зарегистрирована**

* Это конкретное событие из подкатегории Audit Logon генерируется при создании сеанса входа (на целевом компьютере).
* После документирования конкретных полей и допустимых значений для события 4624 я нашел поле logon\_type очень интересным. Это потому, что событие 4624 является единственным доступным событием, которое может сообщить нам конкретный тип входа в систему, запрошенного во время процесса входа в систему.

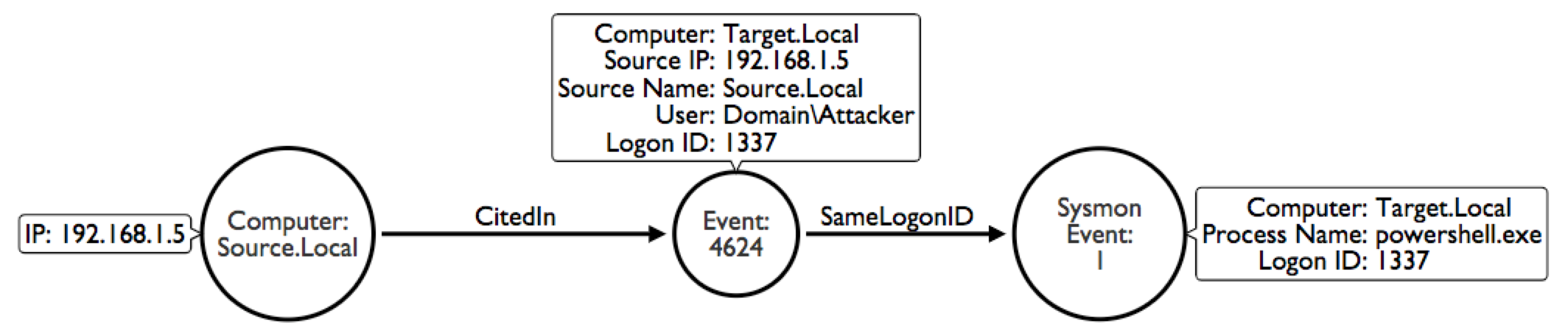


**Почему это актуально?**

* Я считаю, что это очень интересно, потому что я мог бы использовать значение logon\_id из события 4624 для сопоставления других событий безопасности Windows, которые происходят в том же сеансе входа в систему. Кроме того, из-за информации logon\_type, доступной в событии 4624, я смог классифицировать результаты по конкретному типу входа в систему, запрошенному после успешного прохождения аутентификации.

**Я уже это слышал!**

* Помню, в далеком 2017 году мой коллега Ли Кристенсен упомянул, что было бы здорово объединить события входа в систему типа 9 (NewCredentials) с другими событиями, происходящими в том же сеансе входа. Однако в то время мы не знали простого способа достичь этого в масштабе.
* Год спустя, примерно в июне 2018 года, я помню, как разговаривал с другим коллегой Энди Роббинсом о нескольких случаях использования обнаружения с Bloodhound. Я помню, как рассказывал ему о возможном обнаружении бокового движения, сопоставляя событие безопасности 4624 (тип входа 3) и другие события с тем же logon\_id на целевой машине. После нескольких минут разговора об этом он собрал нечто похожее в Arrows, и идея объединения событий по их logon\_id стала более понятной:



MATCH p = (n:Computer)-[:CitedIn|SameLogonID\*1..]->(m:SysmonEvent)  
RETURN p

* Позже в том же году я выпустил Real-Time Sysmon Processing через серии KSQL и HELK. Мне было приятно узнать, что соединения SQL JOIN возможны на уровне конвейера (при прохождении данных). В этой статье я присоединился к событиям Sysmon ProcessCreate и NetworkConnect, используя Confluent KSQL. Вскоре я поделюсь, как присоединить событие безопасности 4624 к другим событиям, которые используют тот же logon\_id с помощью KSQL.

**Что такое JOIN?**

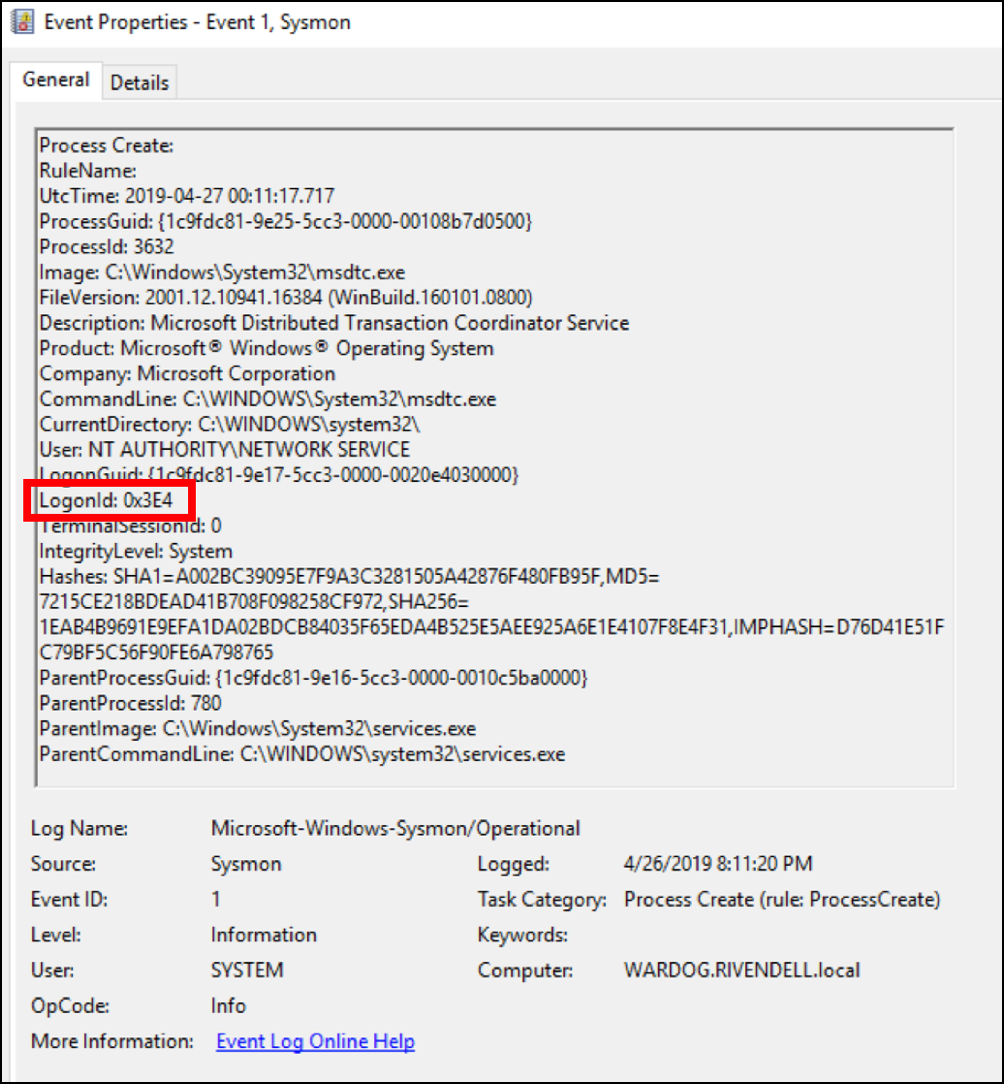
* Мы можем объединить событие 4624 с другими событиями по их значениям logon\_id и добавить контекст logon\_type ко всем событиям, соответствующим логике.

Существует несколько вариантов использования в зависимости от типа запрашиваемого входа:

* Тип входа 3: потенциальное боковое движение
* Тип входа 9: потенциальное олицетворение

**Читай, документируй, моделируй, повторяй!**

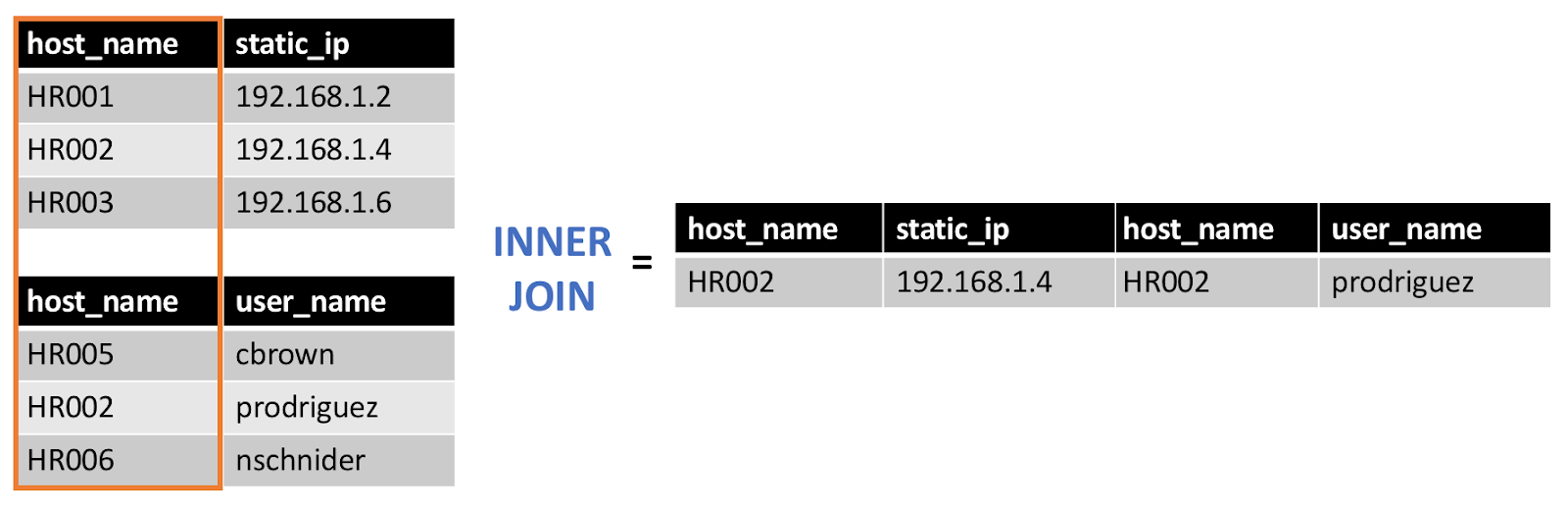
* Единственное, что я хочу подчеркнуть, это потенциальные базовые варианты использования, которые были определены простым чтением, документированием и моделированием журналов событий безопасности. Вот почему так важно делать это как часть вашей стратегии обнаружения, чтобы охватить все возможные варианты использования.
* Например, я не знаю, были ли вы в курсе, но идентификатор события Sysmon 1 (ProcessCreate) – единственное событие в Sysmon, которое предоставляет информацию logon\_id. Поэтому я мог бы легко добавить информацию logon\_type к информации ProcessCreate, предоставленной Sysmon.



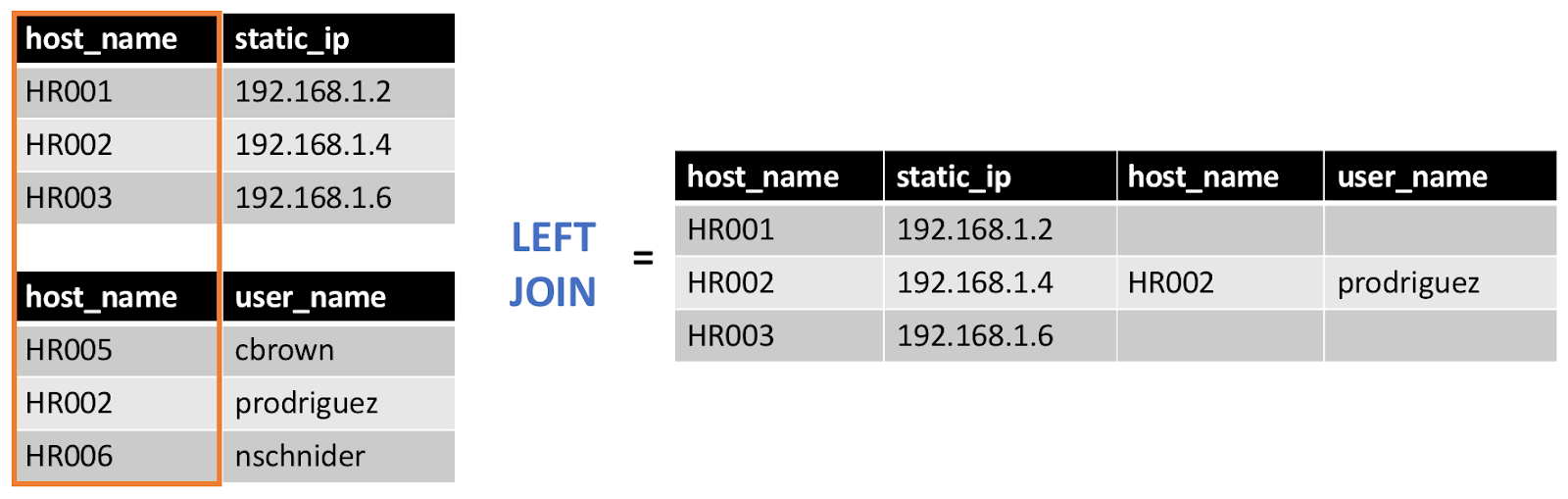
**Как помогает Apache SparkSQL?**

* Мы можем легко использовать возможности реляционной обработки данных Apache SparkSQL и выполнять отчёты SQL JOIN над данными из Elasticsearch в масштабе.
* Прежде чем продолжить, давайте рассмотрим основные концепции каждого типа JOIN.

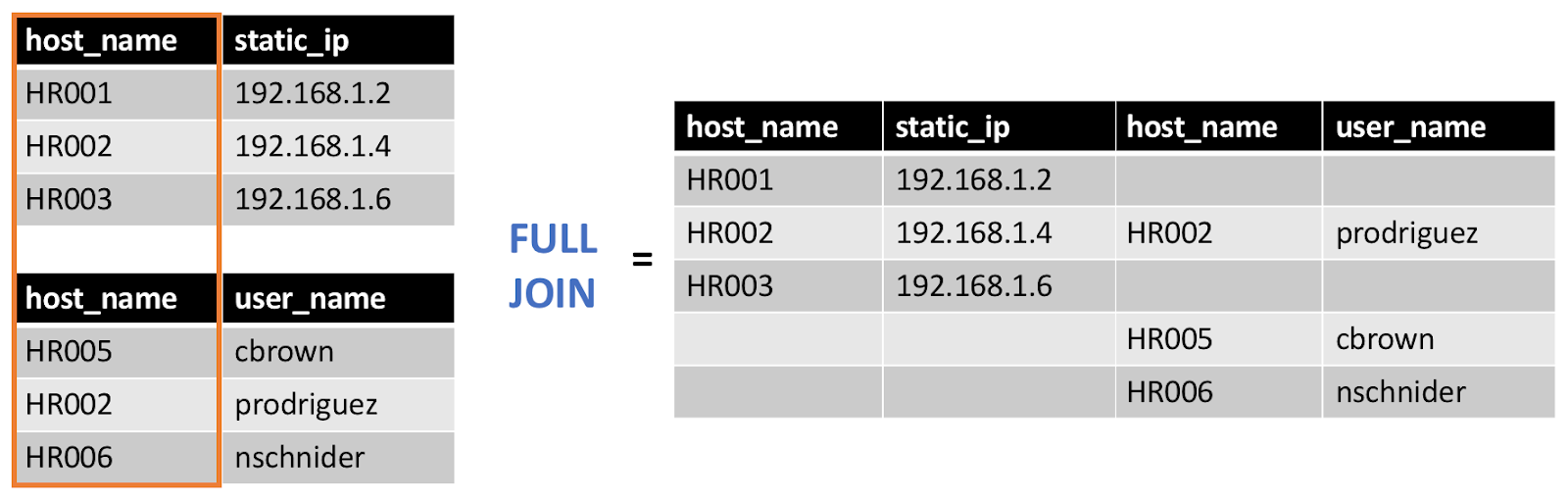
**Inner Join**: возвращает записи данных, которые имеют совпадающие значения в обоих источниках.



Left Outer Join: возвращает записи данных из левого источника и сопоставленные записи данных из правого источника.



**Full Outer Join**: возвращает записи данных при совпадение в левом или правом источнике.



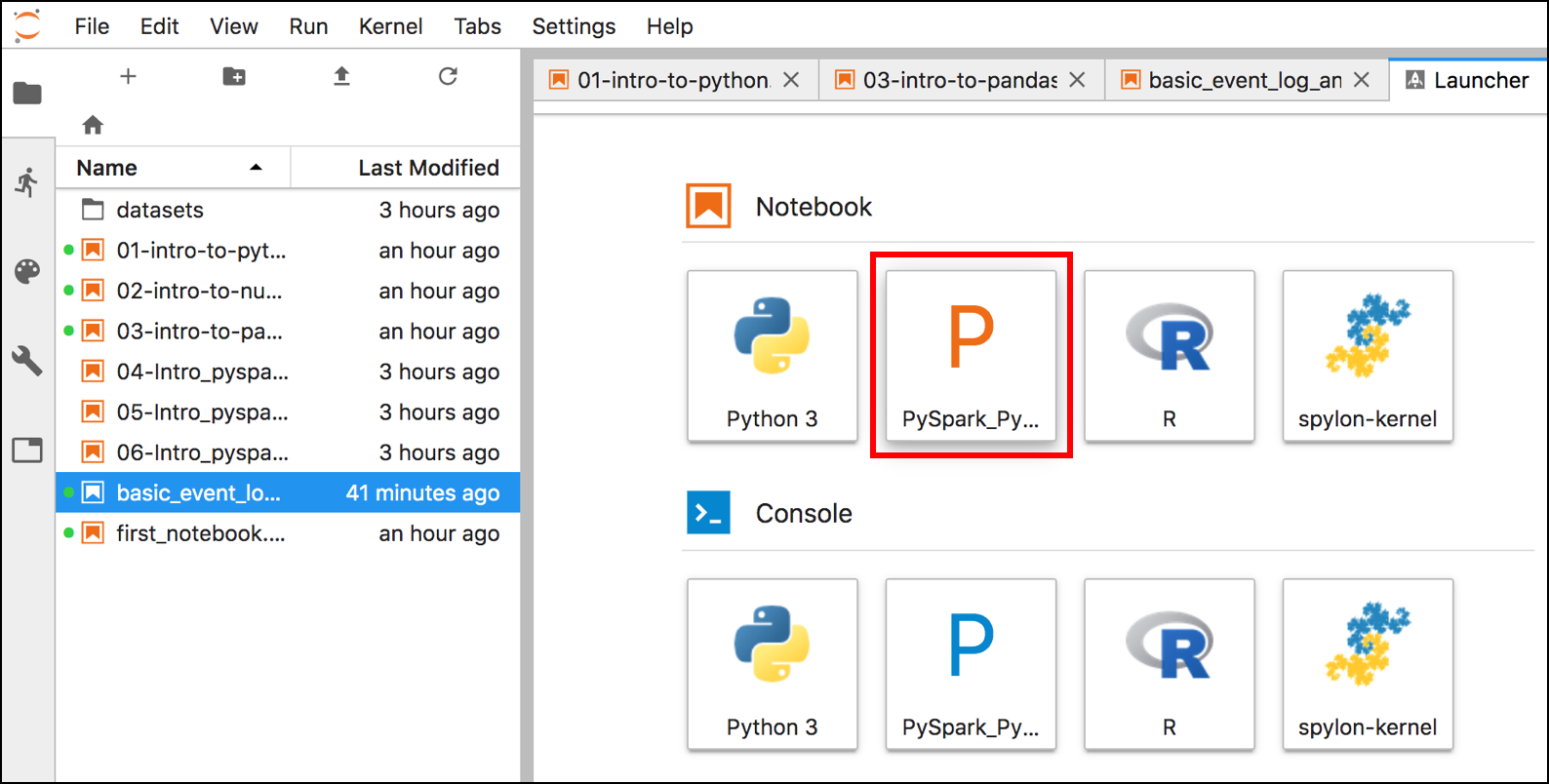
**Набор команд Apache SparkSQL JOIN**

* Чтение из HELK Elasticsearch индекс безопасности Windows
* Создание DataFrame для событий, совпадающих с идентификатором 4624 (успешная аутентификация)
* Фильтрация DataFrame по logon\_type (то есть Network (3) или NewCredentials (9))
* Чтениеот HELK Elasticsearch Sysmon Index
* Создание DataFrame для событий, соответствующих идентификатору события 1 (ProcessCreate)
* Выполнение INNER JOIN между обоими Фреймами данных на logon\_id

**Обнаружение потенциального бокового смещения**

**Создайте новый блокнот**

* Убедитесь, что вы выбрали ядро ​​PySpark.



## Создайте сессию Spark

## Чтобы создать SparkSession, мы используем класс builder

from pyspark.sql import SparkSession

spark = SparkSession.builder \  
 .appName("HELK JOIN") \  
 .master("spark://helk-spark-master:7077") \  
 .enableHiveSupport() \  
 .getOrCreate()

## 

## Запустите Reader & Read

## Используйте класс DataFrameReader и метод read для чтения данных в качестве DataFrame из Elasticsearch.

## Используйте метод load для загрузки данных через считыватель DataFrame и возврата DataFrame. Для начала прочитаем из индекса безопасности.

es\_reader = (spark.read  
 .format("org.elasticsearch.spark.sql")  
 .option("inferSchema", "true")  
 .option("es.read.field.as.array.include", "tags")  
 .option("es.nodes","helk-elasticsearch:9200")  
 .option("es.net.http.auth.user","elastic")  
)security\_df = es\_reader.load("logs-endpoint-winevent-security-\*/")

## 

## Зарегистрируйте Security DataFrame

## С помощью метода createOrReplaceTempView создайте локальное временное представление Security DataFrame.

## Время существования этой временной таблицы связано с SparkSession, который использовался для создания этого DataFrame.

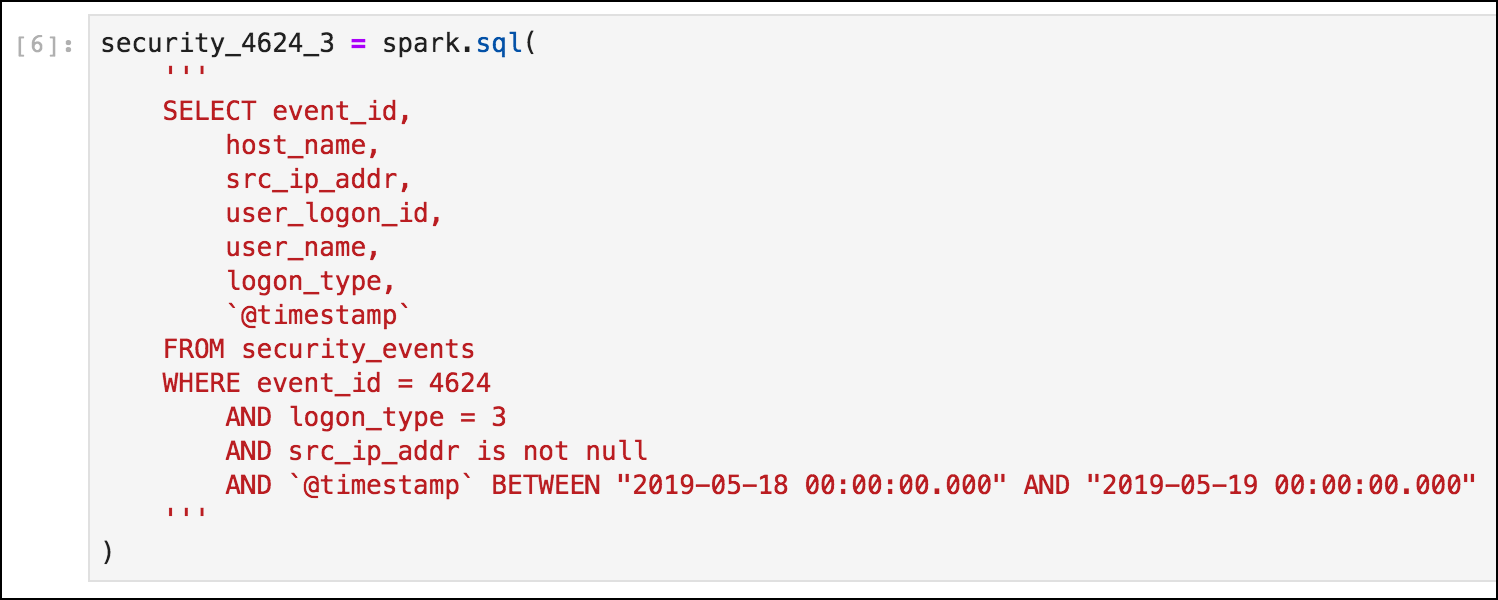
## security\_df = es\_reader.load("logs-endpoint-winevent-security-\*/")



**Создайте события безопасности 4624: тип входа 3 DataFrame**

* Мне нравится гибкость Apache SparkSQL для анализа данных.
* Отфильтруйте данные по event\_id 4624 и logon\_type 3 (вариант использования потенциального бокового смещения). Помните, что мы работаем с набором данных Mordor empire\_invoke\_wmi. Поэтому я жду интересных результатов.
* Отфильтруйте данные для отображения только событий со значениями IP-адресов.
* Отфильтруйте данные за определенное время. Помните, что мы работаем с набором данных Mordor, который был записан около 18 мая 2019 года.

security\_4624\_3 = spark.sql(  
'''  
SELECT event\_id,  
host\_name,  
src\_ip\_addr,  
user\_logon\_id,  
user\_name,  
logon\_type,  
`[@timestamp](http://twitter.com/timestamp" \t "_blank)`  
FROM security\_events  
WHERE event\_id = 4624  
AND logon\_type = 3  
AND src\_ip\_addr is not null  
AND `[@timestamp](http://twitter.com/timestamp" \t "_blank)` BETWEEN "2019-05-18 00:00:00.000" AND "2019-05-19 00:00:00.000"  
'''  
)



## Зарегистрируйте новый Security DataFrame

security\_4624\_3.createOrReplaceTempView("security\_4624\_3")



**Подготовьте Sysmon DataFrame**

* Чтение из индекса Sysmon и регистрация DataFrame в качестве Temp View
* Фильтрация DataFrame по event\_id 1.
* Фильтрация данных в определённом временном окне. Помните, что мы работаем с набором данных Mordor, который был записан около 18 мая 2011 г.

sysmon\_df = es\_reader.load("logs-endpoint-winevent-sysmon-\*/")sysmon\_df.createOrReplaceTempView("sysmon\_events")sysmon\_processcreate = spark.sql(  
 '''  
 SELECT event\_id,  
 host\_name,  
 process\_parent\_name,  
 process\_parent\_guid,  
 process\_parent\_command\_line,  
 process\_name,  
 process\_guid,  
 process\_command\_line,  
 user\_logon\_id,  
 `[@timestamp](http://twitter.com/timestamp" \t "_blank)`  
 FROM sysmon\_events  
 WHERE event\_id = 1  
 AND `[@timestamp](http://twitter.com/timestamp" \t "_blank)` BETWEEN "2019-05-18 00:00:00.000" AND "2019-05-19 00:00:00.000"  
 '''  
)



## Зарегистрируйте новый Sysmon DataFrame

## sysmon\_processcreate.createOrReplaceTempView("sysmon\_1")



## Выполните SQL INNER JOIN

* Создав новые временные представления, мы сможем выполнять внутреннее sql-соединение значений logon\_id.

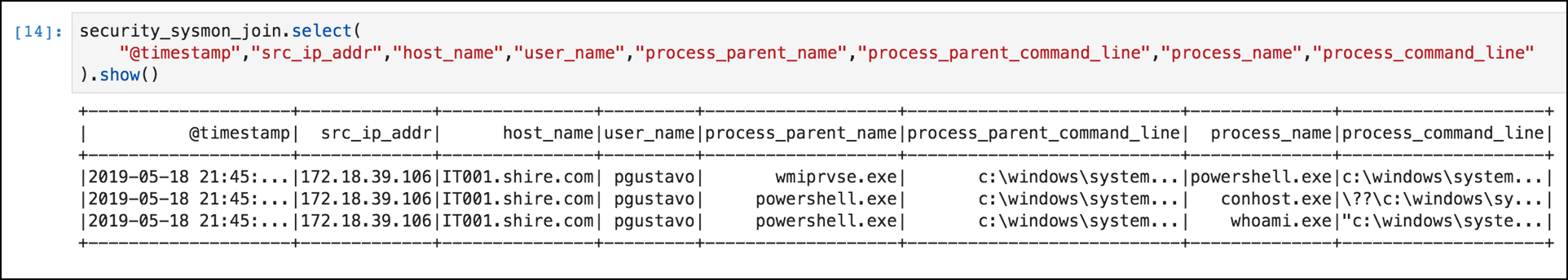
security\_sysmon\_join = spark.sql(  
'''  
SELECT s.`[@timestamp](http://twitter.com/timestamp)`,  
s.host\_name,  
s.src\_ip\_addr,  
s.logon\_type,  
s.user\_logon\_id,  
s.user\_name,  
p.process\_parent\_name,  
p.process\_parent\_guid,  
p.process\_parent\_command\_line,  
p.process\_name,  
p.process\_guid,  
p.process\_command\_line  
FROM security\_4624\_3 s  
INNER JOIN sysmon\_1 p  
ON s.user\_logon\_id = p.user\_logon\_id  
'''  
)



**Посмотрите результатов**

* как вы можете видеть ниже наша логика JOIN смогла сопоставить потенциальное боковое смещение через поведение WMI.

security\_sysmon\_join.select(  
"[@timestamp](http://twitter.com/timestamp)","src\_ip\_addr","host\_name","user\_name","process\_parent\_name","process\_parent\_command\_line","process\_name","process\_command\_line"  
).show()

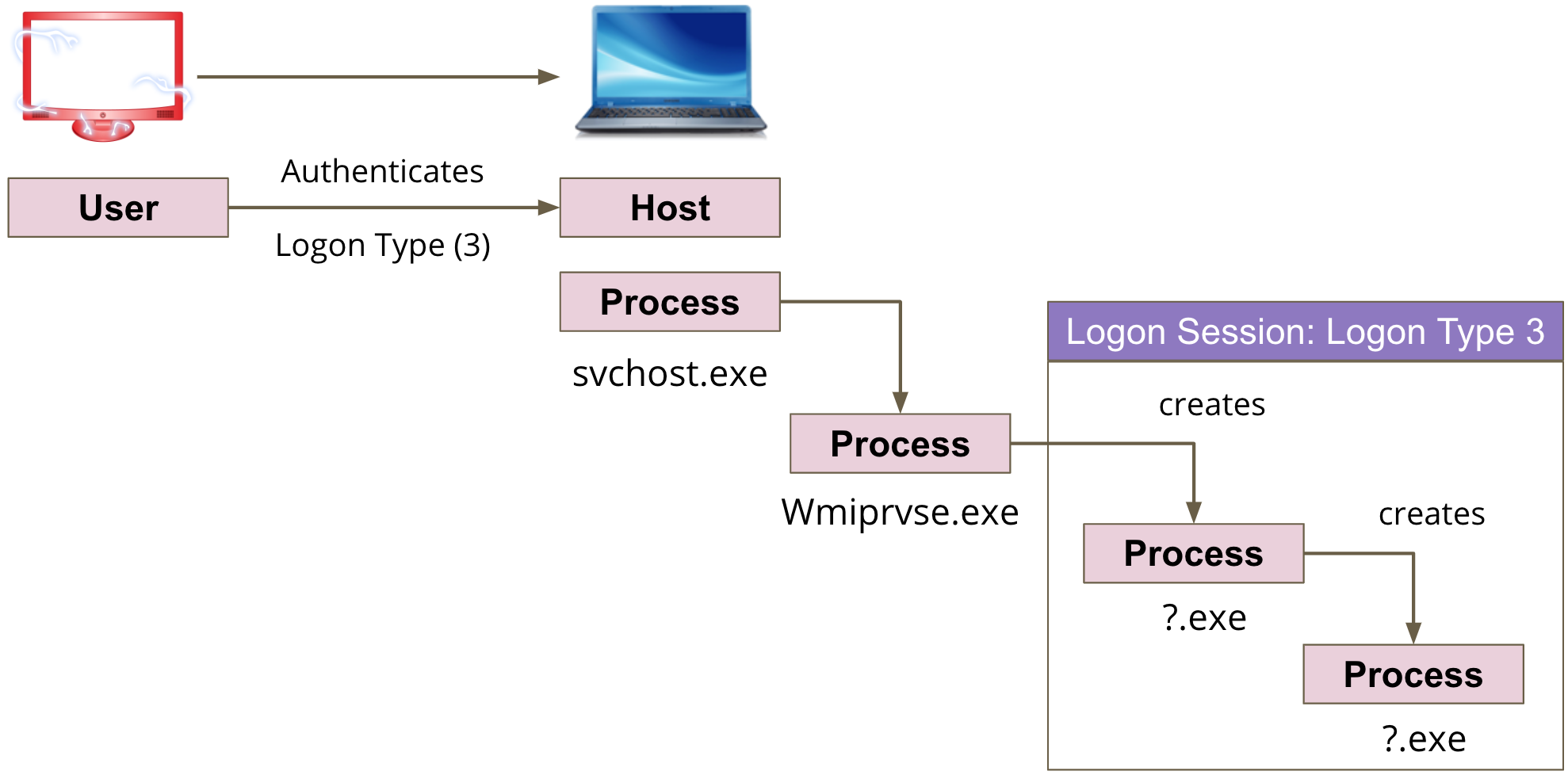


**Боковое смещение через WMI поведение?**

* Один хорошо известный метод бокового смещения выполняется через объект WMI — класс Win32\_Process и его метод Create. Это связано с тем, что метод Create позволяет пользователю создавать процесс локально или удалённо.
* Следует отметить, что, когда метод Create используется в удалённой системе, он запускается в хост-процессе с именем «Wmiprvse.exe».
* Процесс WmiprvSE.exe — это то, что порождает процесс, определённый в параметре CommandLine метода Create. Следовательно, Wmiprvse.exe будет выполнять роль родителя для нового процесса, созданного удалённо.
* WmiprvSE.exe -— это сервер DCOM, который создаётся под хостом службы DCOM svchost.exe со следующими параметрами C: \ WINDOWS \ system32 \ svchost.exe -k DcomLaunch -p.

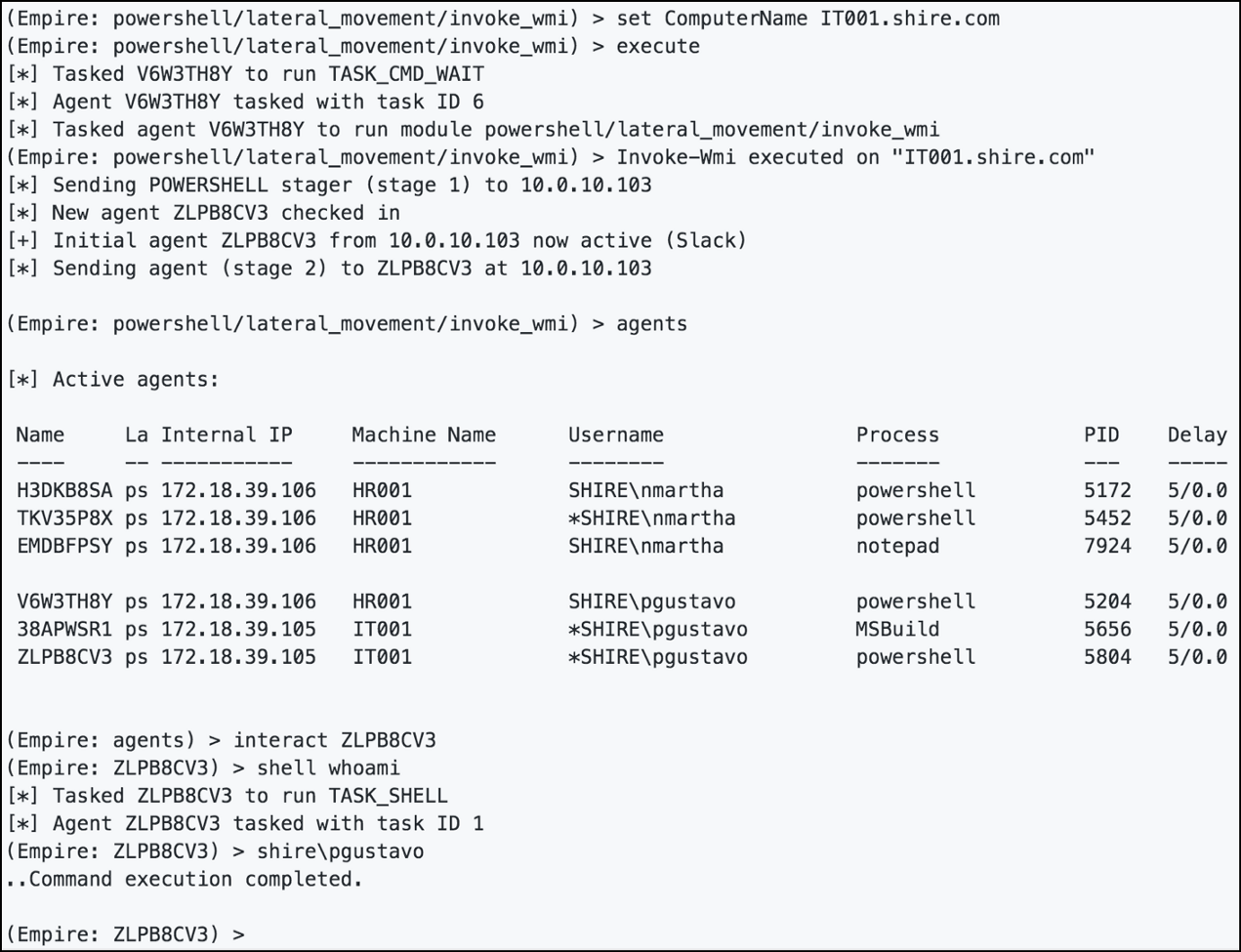
**WMI и сеанс входа в систему типа 3**

WmiprvSE.exe порождается в другом сеансе входа хостом службы DCOM. Однако всё, что выполняет WmiprvSE.exe, происходит в новом сеансе входа в систему, созданном пользователем, прошедшим проверку подлинности из сети.



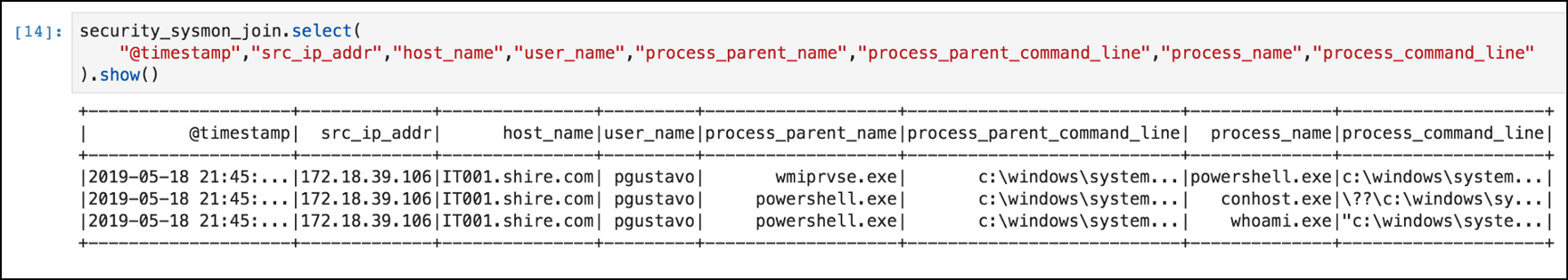
Согласно документации набора данных Mordor:

* Злоумышленник использовал модуль empire\_invoke\_wmi. для создания нового агента на компьютере IT001.shire.com (172.18.39.105) из HR001.shire.com (172.18.39.106)
* Злоумышленник использовал учётные данные SHIRE \ pgustavo
* Злоумышленник взаимодействовал с новым агентом и выполнил whoami



Если снова посмотреть на результаты из нашего SparkSQL JOIN:

* Исполнение постановщика пришло с 172.18.39.106.
* Аккаунт pgustavo использовался для выполнения сетевой аутентификации.
* Выполнение PowerShell было проведено WmiprvSE.
* PowerShell был создан в контексте нового сеанса входа в сеть.
* PowerShell выполнил Whoami
* Whoami был создан в контексте нового сеанса входа в сеть.



Надеюсь, вам понравилась статья. Я попытался использовать все основополагающие концепции, которые мы изучили в первых трех статьях, и применил их к исследовательским идеям, появившимся в результате документирования и моделирования события безопасности.

В следующей статье я покажу вам, как использовать Jupyter Notebook для документирования и обмена сценариями по поиску угроз в вашей организации и с сообществом.

**Ссылки**

<http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf>

<https://posts.specterops.io/real-time-sysmon-processing-via-ksql-and-helk-part-1-initial-integration-88c2b6eac839>

<https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4624>

<https://docs.microsoft.com/en-us/windows/desktop/secauthn/lsa-logon-sessions>

<https://github.com/Cyb3rWard0g/mordor/blob/master/small_datasets/windows/execution/windows_management_instrumentation_T1047/empire_invoke_wmi.md>