Краткий обзор возможностей и принципов работы Sentinel EMS Web Service API

Оглавление

Общие принципы работы EMS Web Service API

Sentinel EMS глобально делится на три части:

1. Sentinel EMS Vendor Portal - основная часть Sentinel EMS, доступ к которой есть у разработчика ПО (вход по логину с паролем). Здесь разработчик может создавать Feature ID \ Product's \ Entitlement's \ активировать новые SL ключи \ разблокировать ранее заблокированные лицензии \ создавать и выпускать обновления и т.д.;
2. Sentinel EMS Customer Portal - клиентская часть Sentinel EMS, доступ к которой предоставляется конечным пользователям (вход с помощью ключа активации). Здесь клиент может самостоятельно активировать себе лицензию по предоставленному ему ключу активации (Product Key);
3. Sentinel EMS Channel Partner Portal - партнёрская часть Sentinel EMS, доступ к которой предоставляется партнёрам разработчика, через которых разработчик хочет вести продажи своего ПО (вход по логину с паролем от учётных записей, создаваемых разработчиком для своих партнёров в самом Sentinel EMS).

Основные моменты касающиеся EMS Web Service API:

1. EMS Web Service API - представляет из себя RESTFul API для взаимодействия с Sentinel EMS из любого приложения/интерфейса.
2. Всё что можно сделать через Sentinel EMS UI, всё это же можно выполнить и через EMS Web Service API.
3. EMS Web Service API, в силу своей специфики, не привязан к какому-либо конкретному языку программирования и является универсальным инструментом, работать с котором можно из любого приложения/интерфейса, способного выполнять HTTP запросы.

Возможности методов EMS Web Service API хорошо иллюстрирует таблица* из документации:

Resource	Methods			
	GET	PUT (create)	POST (update)	DELETE
Vendor	✓			
Namespace (deprecated)	✓			
Feature	✓	✓	✓	✓
License Model	✓			
Product	✓	✓	✓	✓
Customer	✓	✓	✓	✓
Contact	✓	✓	✓	✓
Entitlement	✓	✓	✓	✓
Target (or Protection key)	✓		✓	✓
Activation	✓		✓	

* - *https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/Introduction.htm*

Базовый принцип работы с EMS Web Services API заключается в составлении правильного URL запроса и отправки его на сервер Sentinel EMS.

Пример базовой части любого запроса к Sentinel EMS Web Services API:

http(s)://<Hostname>:<port>/ems/v710/ws/

Где:

- ➔ Hostname — доменное имя или IP адрес ПК с Sentinel EMS Server.
- ➔ port — порт на котором работает Sentinel EMS Server (по умолчанию это либо 8080, либо 80).
- ➔ ems — базовая директория где расположен Sentinel EMS Server.
- ➔ v710 — версия Sentinel EMS Web Services API.
- ➔ ws — константа, говорящая о том, что это запрос к Sentinel EMS Web Services API.

Для выполнения большей части запросов **(!НО НЕ ВСЕХ!)** к Sentinel EMS Web Services API, в начале потребуется выполнить аутентификацию на сервер Sentinel EMS.

Аутентификация позволяет разграничить уровни доступа к операциям в Sentinel EMS. Поддерживается аутентификация:

- Аутентификация по логину с паролем - для входа в: Sentinel EMS Vendor Portal \ Sentinel EMS Chanel Partner Portal;
- Аутентификация по ключу активации (Product Key) - для входа в: Sentinel EMS Customer Portal.

Аутентификация

Аутентификация по логину с паролем

При аутентификации в Sentinel EMS посредством логина с паролем, создаётся сессия, в рамках которой пользователю предоставляет доступ к Sentinel EMS Vendor Portal \ Sentinel EMS Channel Partner Portal, с соответствующим уровнем прав на дальнейшие операции. Уровень прав зависит от роли / ролей ассоциированных с используемой для входа учётной записью (заданных при её создании):

Resource	Roles and Associated Permissions			
	Product Manager	Entitlement Manager	Production	Customer Services
Product Group	GET, PUT, POST, DEL	GET		
Feature	GET, PUT, POST, DEL	GET		
License Model	GET	GET		
Product	GET, PUT, POST, DEL	GET		
Customer		GET, PUT, POST, DEL	GET	GET, PUT, POST, DEL
Contact		GET, PUT, POST, DEL	GET	GET, PUT, POST, DEL
Entitlement		GET, PUT, POST, DELETE	GET, POST	
Product Key		GET, POST	GET	GET, POST
Activation			GET, POST	GET, POST

> **NOTE** The following roles are not listed in this table as they have either all or no permission for all of the resources:
> > Super User—All permissions (GET, PUT, POST, DELETE) for all resources in all available Batch Codes.
> > Batch Code Admin—All permissions (GET, PUT, POST, DELETE) for all resources in the assigned Batch Codes.
> > Development—No permissions are available for any resource for this role.

* - *https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/UserRoles.htm*

Для выполнения аутентификации по логину с паролем, необходимо выполнить POST запрос **login.ws** из EMS Web Service API, и в качестве параметра в запросе, передать XML структуру вида (со своими логином и паролем):
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/Login.htm#Publisher_Login

```
//-----------
<?xml version="1.0" encoding="UTF-8"?>
<authenticationDetail>
    <userName>*Логин*</userName>
    <password>*Пароль*</password>
</authenticationDetail>
//-----------
```

При успешной аутентификации в ответ будет возвращён HTTP код 200 и другая XML структура вида:

```
//-----------
<?xml version="1.0" encoding="UTF-8"?>
<authenticationDetail>
  <userName>*Логин*</userName>
  <version>x.x.x</version>
</authenticationDetail>
//-----------
```

В случае неудачи, будет возвращён какой-то иной код ошибки, варианты возможных кодов описаны тут:
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/HTTPStatusCodes.htm

Аутентификация по ключу активации (Product Key)

При аутентификации в Sentinel EMS посредством ключа активации, создаётся сессия, в рамках которой пользователю предоставляется доступ к Sentinel EMS Customer Portal, где он сможет выполнить онлайн / оффлайн активацию своего ключа и / или загрузить лицензии (*.V2C файлы), ранее уже активированные по этому ключу активации, а также он сможет загрузить из Sentinel EMS утилиту RUS.exe (Remote Update System - утилита для удалённого обновления лицензий в ключах).

Для выполнения аутентификации по ключу активации, необходимо выполнить POST запрос **loginByProductKey.ws** из EMS Web Service API, и в качестве параметра в запросе, передать сам ключ активации:
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/loginByProductKey.htm

Пример запроса аутентификации по ключу активации:
https://localhost:8443/ems/v710/ws/loginByProductKey.ws

Request Body:
productKey=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

При успешной аутентификации будет возвращён ответ, содержащий XML структуру вида:

```
//-----------
<?xml version="1.0" encoding="UTF-8"?>
<EMSResponse>
  <regRequired>1</regRequired>
  <sessionId>xxxxxxxxxxxxxxxxxxxxxx</sessionId>
  <stat>ok</stat>
</EMSResponse>
//-----------
```

После успешной аутентификации можно выполнять остальные запросы к Sentinel EMS.

Также, важно помнить, что некоторые запросы EMS Web Service API не требуют предварительной аутентификации, например запрос Fetch Pending Updates (V2C) Using C2V:
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/FetchV2C_C2V.htm#Fetch_Pending_Updates_(V2C)_Using_C2V, - используемый для запроса и получения доступных обновлений для уже имеющегося у клиента ключа защиты.

Для простоты работы с EMS Web Service API и упрощения процесса его интеграции в своё решение, существует специальный пример-сервис по работе с EMS Web Service API, называется он **emsWSDemo**:
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/AbtEmsWSDemo.htm

emsWSDemo предназначен в первую очередь для наглядной демонстрации возможностей EMS Web Service API, а также для быстрого и простого тестирования запросов EMS Web Service API. Кроме того, emsWSDemo способен генерировать примеры кода вызова для запросов EMS Web Service API на Java.
С помощью emsWSDemo можно “на ходу” выполнить любой запрос EMS Web Service API и посмотреть что он возвращает в ответ.

Для того чтобы использовать emsWSDemo необходимо его развернуть на ПК с установленным Sentinel EMS. Выполняется это следующим образом:

1. Необходимо на ПК с установленным Sentinel EMS скопировать файл emsWSDemo.war из директории: %EMS_HOME%\samples\EmsWSDemo\bin в директорию: %EMS_HOME%\EMSServer\webapps

2. Подождать от нескольких секунд до ~2 минут, пока Tomcat развернёт сервис emsWSDemo из *.war файла

3. При необходимости отредактируйте настройки сервиса emsWSDemo в файле: %EMS_HOME%\EMSServer\webapps\emsWSDemo\WEB-INF\classes\emsWSDemo.properties
   Возможна настройка следующих параметров сервиса:

   - useSSL - если Sentinel EMS настроен на работу через HTTPS (SSL), установите значение = true (по умолчанию используется значение: false);

   - Server - укажите настройки адреса и порта, на котором работает Sentinel EMS (по умолчанию используется значение: localhost:80, - требует настройки в случае, когда сервис emsWSDemo разворачивается на ПК, отличном от того, где установлен и настроен сам Sentinel EMS);

   - trustStore - указывается путь до файла-хранилища сертификатов (certificate keystore file). Настраивается в случае работы через HTTPS (SSL);

   - trustStorePass - указывается пароль для файла-хранилища сертификатов (trust store password). Значение по умолчанию: changeit. Настраивается в случае работы через HTTPS (SSL).

4. Перезапустите службу Sentinel EMS service (или перезагрузите ПК с установленным Sentinel EMS).

После того как сервис emsWSDemo развёрнут и настроен, доступ к нему можно получить через браузер, открыв страницу с адресом: http://host:port/emsWSDemo, - с любого ПК, расположенного в той же сети что и ПК с emsWSDemo.

Где:

- host - IP адрес или имя ПК, где развёрнут сервис emsWSDemo;
- port - порт на котором работает Sentinel EMS Server.

Пример: http://localhost:8080/emsWSDemo - если emsWSDemo развёрнут и запускается на ПК с установленным Sentinel EMS.

Пример задач, решаемых с использованием EMS Web Service API (некоторые из примеров будут рассмотрены подробнее)

1. **Интеграция Sentinel EMS с backoffice (CRM/ERP системами)**

   Чаще всего разработчику удобнее все свои внутренние процессы в организации выполнять из единой системы. С помощью EMS Web Service API можно организовать работу с системой лицензирования Sentinel EMS изнутри своей единой системы (CRM/ERP). Так, например, можно интегрировать Sentinel EMS с 1С, и тогда управлять лицензиями можно будет напрямую из 1С, никак не контактируя с UI Sentinel EMS. При этом сам Sentinel EMS выступает в роли “ядра” системы лицензирования и отвечает за генерацию лицензий/обновлений, сервер активации, хранение истории выпущенных лицензий, и логирование процессов лицензирования.

   Довольно сложная, объёмная задача, решение которой во многом зависит как от используемой CRM системы, так и от требуемого уровня интеграции. По этим причинам пример реализации этой задачи рассматривать мы не будем.

2. Автоматизация процесса доставки новых лицензий

Благодаря EMS Web Service API можно встроить механизм активации новых лицензий напрямую в свой программный продукт. Благодаря чему процесс доставки лицензии может стать полностью автоматизированным. Разумеется это возможно при соблюдении двух условий:

a. Сервер Sentinel EMS настроен на работу через интернет и он доступен по сети;
b. Защищенное приложение, из которого выполняется запрос на активацию, имеет доступ в интернет.

Пример реализации:

В защищаемую программу добавляем окно для ввода ключа активации и кнопку “Активировать лицензию”.
В обработчик нажатия на кнопку “Активировать лицензию” добавляем код, логика которого подразумевает выполнение следующих действий:

1) Выполнение сбора данных (C2V с Fingerprint) с устройства клиента, требуемых для генерации клиенту программного Sentinel SL ключа (функция **Hasp.GetInfo** из Sentinel Licensing API): https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_get_info.htm

   Пример кода вызова функции:

```
//-----------
string scope =
"<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
"<haspscope>" +
" <license_manager hostname=\"localhost\" />" +
"</haspscope>";

string format =
"<haspformat format=\"host_fingerprint\"/>";

string vendorCode = “Ваш_Vendor_Code”;
string info = null;
HaspStatus status = Hasp.GetInfo(scope, format, vendorCode, ref info);
if (HaspStatus.StatusOk != status)
{
        //handle error
}
//-----------
```

2) Выполнение логина по ключу активации на сервер Sentinel EMS (POST запрос **loginByProductKey.ws** из EMS Web Service API): https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/loginByProductKey.htm

   В качестве параметра в запрос необходимо передать Product Key (ключ активации).

3) !ОПЦИОНАЛЬНО!: Если при генерации ключа активации была включена обязательная или опциональная регистрация клиента по ключу активации (MANDATORY или DESIRED), то может потребоваться реализация такой регистрации. Для этого необходимо предусмотреть дополнительное окно регистрации с требующимися полями: ФИО клиента, его e-mail адрес, физический адрес и т.д..
   Заполненные клиентом данные необходимо будет отправить на сервер Sentinel EMS посредством PUT запроса **customer.ws** из EMS Web Service API:
   https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/addCustomer.htm

   Разумеется, если клиент уже регистрировался ранее, ему должна быть доступна опция “Входа для уже зарегистрированного пользователя”, при этом задача сводится к двум действиям:
   -> поиску клиента в базе Sentinel EMS по адресу его почты, посредством GET запроса **contact.ws?contactEmailId={contactEmailId}** из EMS Web Service API:
   https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/getContbyEmail.htm
   -> получив данные о клиенте по его почте, убеждаемся тем самым что клиент действительно зарегистрирован в Sentinel EMS, после чего можно добавить (ассоциировать) клиента с его Product Key посредством добавления адреса почты клиента в его Product Key, делается это с помощью POST запроса **productKey/{productKeyId}.ws** из EMS Web Service API:
   https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/updatePK.htm

4) !ОПЦИОНАЛЬНО!: После регистрации/входа по ключу активации, можно вывести клиенту информацию по данному ключу: количество доступных активаций, список продуктов, активируемых по ключу активации и т.д..
   Делается это с помощью GET запроса **productKey/{productKeyId}.ws** из EMS Web Service API:
   https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/getEntByPK.htm#Get_Product_Key_Details_(XML)

5) Выполнение запроса на активацию, для чего потребуется выполнить POST запрос **productKey/{productKeyId}/activation.ws** из EMS Web Service API: https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/generateLic.htm#Generate_License

   В качестве параметра в запрос необходимо передать Product Key (ключ активации) и C2V с Fingerprint, полученный в результате выполнения функции **Hasp.GetInfo** из Sentinel Licensing API.

6) Полученный в результате ответ будет содержать XML структуру с V2C массивом данных (по сути файл лицензии), который необходимо будет из ответа извлечь и применить к устройству посредством функции **Hasp.Update** из Sentinel Licensing API: https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_update.htm

   Пример кода вызова функции:

```
//-----------
string update_data = “V2C_массив_данных”;
string ack_data = null;

HaspStatus status = Hasp.Update(update_data, ref ack_data);

if (HaspStatus.StatusOk != status)
{
//handle error
}
//-----------
```

3. Автоматизация процесса доставки обновлений для уже существующих лицензий

С помощью всего одного запроса из EMS Web Service API (Fetch Pending Updates (V2C) Using C2V) можно напрямую из защищенного продукта выполнить проверку наличия обновлений для существующего ключа на сервере Sentinel EMS, и, при наличие каких-либо обновлений, скачать их и применить к ключу. По требованиям для работы здесь ситуация полностью аналогична процессу доставки новых лицензий, а именно, требуется чтобы Sentinel EMS был настроен на работу через интернет и доступен по сети + защищенное приложение, из которого выполняется запрос на активацию, должно иметь доступ в интернет.

Пример реализации:

В защищаемую программу добавляем функцию, которая будет выполняться периодически при запущенном ПО и наличии на ПК подключения к интернету. Функция должна реализовывать следующую логику:

1) Выполнение сбора данных (C2V файла) с HL/SL ключа клиента, (функция **Hasp.GetInfo** из Sentinel Licensing API): https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_get_info.htm

   Пример кода вызова функции:

```
//-----------
string scope =
"<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
"<haspscope>" +
"    <hasp id=\\" + KeyID + "\" />" + // KeyID нужно заменить на Key ID того ключа, для которого необходимо проверить обновления
"</haspscope>" +
"";

string format =
"<haspformat format=\"updateinfo\"/>";

string vendorCode = “Ваш_Vendor_Code”;
string info = null;
HaspStatus status = Hasp.GetInfo(scope, format, vendorCode, ref info);

if (HaspStatus.StatusOk != status)
{
//handle error
}
//-----------
```

2) Получив C2V с ключа, необходимо выполнить один POST запрос **activation/target.ws** из Sentinel EMS Web Service API: https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/FetchV2C_C2V.htm#Fetch_Pending_Updates_(V2C)_Using_C2V

   В качестве параметра в запрос необходимо передать C2V, полученный в результате выполнения функции **Hasp.GetInfo** из Sentinel Licensing API.

3) В случае если для ключа есть обновление, сервер Sentinel EMS вернёт это обновление в виде V2CP массива, в ответ на наш запрос. Из ответа

потребуется извлечь V2CP массив с обновлением и затем применить его к ключу, посредством функции **Hasp.Update** из Sentinel Licensing API: https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_update.htm

Пример кода вызова функции:

```
//-----------
string update_data = “V2C/V2CP_массив_данных”;
string ack_data = null;

HaspStatus status = Hasp.Update(update_data, ref ack_data);

if (HaspStatus.StatusOk != status)
{
//handle error
}
//-----------
```

4. Создание проксирующего сервера для кастомизации логики генерации/получения лицензий/обновлений

В ряде случаев может оказаться необходимым руководствоваться своей (уникальной) логикой для генерации тех или иных обновлений лицензий клиентов. В таких ситуациях, на базе EMS Web Service API можно создать свой прокси сервер, между клиентами и своим сервером Sentinel EMS, и уже логика работы этого прокси сервера может быть абсолютно любой. Запросы на генерацию/получение лицензий/обновлений будут приходить на проксирующий сервер, а он, в свою очередь, руководствуясь своей логикой, будет выполнять обработку запроса и общение с сервером Sentinel EMS: генерировать новые лицензии/обновления, кастомизировать текущие Entitlement’ы, регистрировать в системе новых клиентов/контакты, автоматизировать процесс разблокировки ключей клиентов, в ситуациях когда по тем или иным причинам у клиента заблокировался ключ (например из-за замены оборудования на ПК, или из-за случайного перевода системных часов) и т.д..

5. Дополнительная валидация Entitlement’а

Например для контроля состояния текущих лицензий и срока их жизни на уровне Entitlement’а в Sentinel EMS. Можно узнать статус Entitlement’а и доступных по нему лицензий по ключу активации от этого Entitlement’а. По требованиям для работы

здесь ситуация полностью аналогична процессу доставки новых лицензий, а именно, требуется чтобы Sentinel EMS был настроен на работу через интернет и доступен по сети + защищенное приложение, из которого выполняется запрос на активацию, должно иметь доступ в интернет.

Пример реализации:

Необходимо реализовать функцию, которая будет выполняться периодически при запущенном ПО и наличии на ПК подключения к интернету.
Функция должна иметь примерно следующую логику:

Получение данных по ключу защиты, с использованием его Key ID (GET запрос **target/{protectionKeyId}/entitlement.ws** из EMS Web Service API):
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/GetKeyEnt.htm

В качестве параметра в запрос необходимо передать Key ID ключа, в параметре **protectionKeyId**.
В результате выполнения такого запроса будет возвращён список Entitlements с их статусами, ассоциированных с этим ключом защиты.

6. Контроль доставки лицензий до ключа клиента

Для того чтобы разработчик был уверен что все сгенерированные клиенту обновления были применены клиентом к ключу, можно к Entitlement'ам, на этапе их генерации добавлять атрибут ACKNOWLEDGEMENT_REQUEST
Данный атрибут отвечает за то, что статус Entitlement'а в Sentinel EMS UI, будет значится как Acknowledged(подтверждённый) только после того, как в Sentinel EMS будет добавлена информация о состоянии ключа из *.c2v файла, полученного с ключа на момент **после** применения обновления. Благодаря чему можно быть уверенным что состояние лицензий в ключах клиентов всегда в актуальном состоянии. С помощью EMS Web Service API можно выполнять доставку и чтение *.c2v файлов, подтверждающих факт применения обновлений, до сервера Sentinel EMS, для этого используется запрос: Update Protection Key.
По требованиям для работы здесь ситуация полностью аналогична процессу доставки новых лицензий, а именно, требуется чтобы Sentinel EMS был настроен на работу через интернет и доступен по сети + защищенное приложение, из которого выполняется запрос на активацию, должно иметь доступ в интернет.

Пример реализации:

В защищаемую программу добавляем функцию, которая будет выполняться периодически при запущенном ПО и наличии на ПК подключения к интернету.

Функция должна реализовывать следующую логику:

1) Выполнение сбора данных (C2V файла) с HL/SL ключа клиента, (функция **Hasp.GetInfo** из Sentinel Licensing API): https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_get_info.htm

   Пример кода вызова функции:

```
//----------
string scope =
"<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
"<haspscope>" +
"    <hasp id=\\" + KeyID + "\" />" + // KeyID нужно заменить на Key ID того ключа, для которого необходимо проверить обновления
"</haspscope>" +
"";

string format =
"<haspformat format=\"updateinfo\"/>";

string vendorCode = “Ваш_Vendor_Code”;
string info = null;
HaspStatus status = Hasp.GetInfo(scope, format, vendorCode, ref info);

if (HaspStatus.StatusOk != status)
{
//handle error
}
//----------
```

2) Получив C2V с ключа, необходимо выполнить один POST запрос **target.ws** из Sentinel EMS Web Service API: https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/UpdateKey.htm

   В качестве параметра в запрос необходимо передать в параметре C2V, данные, полученный в результате выполнения функции **Hasp.GetInfo** из Sentinel Licensing API, а в параметре **action** требуется передать значение “**Checkin**”.

7. Автоматизация процесса определения причины блокировки ключа клиента

В ситуации когда у клиента по той или иной причине заблокировался ключ (сменилось железо на ПК и т.д.), можно посредством запроса Decode C2V из EMS Web Service API получить от сервера Sentinel EMS расшифрованный *.c2v файл и показать клиенту причину блокировки ключа (указать какое именно оборудование сменилось и/или к какому оборудованию “привязан” ключ).

Пример реализации:

В защищаемую программу добавляем функцию, которая будет выполняться в случае если у клиент, при попытке открытия сессии с его ключом, была возвращена ошибка вида:
-> Error code: 5 - AccessDenied
-> Error code: 24 - InvalidTime
-> Error code: 45 - TimeError
-> Error code: 47 - CorruptStorage
-> Error code: 52 - HardwareModified
-> Error code: 64 - CloneDetected
-> Error code: 66 - HaspInactive
-> Error code: 72 - CannotReadFile
-> Error code: 78 - SecureStoreIdMismatch
-> Error code: 83 - HaspDisabled

Разумеется на ПК требуется наличие подключения к интернету.

Функция должна реализовывать следующую логику:

1) Выполнение сбора данных (C2V файла) с заблокированного HL/SL ключа клиента, (функция **Hasp.GetInfo** из Sentinel Licensing API): https://docs.sentinel.gemalto.com/ldk/LDKdocs/API-licensing-net/Licensing_API/hasp_get_info.htm

   Пример кода вызова функции:

```
//-----------
string scope =
"<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
"<haspscope>" +
"    <hasp id=\\" + KeyID + "\" />" + // KeyID нужно заменить на Key ID того ключа, для которого необходимо проверить обновления
"</haspscope>" +
```

```
"";

string format =
"<haspformat format=\"updateinfo\"/>";

string vendorCode = “Ваш_Vendor_Code”;
string info = null;
HaspStatus status = Hasp.GetInfo(scope, format, vendorCode, ref info);

if (HaspStatus.StatusOk != status)
{
//handle error
}
//----------
```

2) Получив C2V с ключа, необходимо выполнить один POST запрос **target.ws** из Sentinel EMS Web Service API:
https://docs.sentinel.gemalto.com/ldk/LDKdocs/WebHelpWS/UpdateKey.htm

   В качестве параметра в запрос необходимо передать в параметре C2V, данные, полученный в результате выполнения функции **Hasp.GetInfo** из Sentinel Licensing API, а в параметре **action** требуется передать значение “**Read**”.

3) Полученный в ответ на запрос массив данных будет содержать XML или JSON структуру с расшифрованным C2V, пример:

```
//----------
<?xml version="1.0" encoding="UTF-8"?>
<ProtectionKey>
 <ProtectionKeyOutput>
  <C2V>
   <sentinel_ldk_info>
    <key>
     <id>...</id>
     <type>...</type>
     <update_counter>...</update_counter>
     <vendor>
      <id>...</id>
      <name>...</name>
     </vendor>
     <configuration_info>
      <system_fingerprint>
       <raw_data>...</raw_data>
```

```
    <fingerprint_info>
      <criteria>
        <name>cpu</name>
        <value>2346217999</value>
      </criteria>
      <criteria>
        <name>mainboard</name>
        <value>2308647388</value>
      </criteria>
      …
    </fingerprint_info>
    ...
  </system_fingerprint>
  <reference_fingerprint>
    <fingerprint_control_type>ISV Managed</fingerprint_control_type>
    <raw_data>...</raw_data>
    <fingerprint_info>
      <criteria>
        <name>cpu</name>
        <value>2346217968</value>
      </criteria>
      <criteria>
        <name>mainboard</name>
        <value>2308647382</value>
      </criteria>
      …
    </fingerprint_info>
  </reference_fingerprint>
  <fridge_version>...</fridge_version>
  <vlib_version>...</vlib_version>
</configuration_info>
<clone_detected>Yes\No</clone_detected>
...
<product>
  <id>...</id>
  <name>...</name>
  <feature>
    <id>...</id>
    <name>...</name>
    <license_properties>
      <perpetual/>
      <concurrency>
        <count>Unlimited</count>
```

```
            <count_criteria>Per Station</count_criteria>
            <network_access>No</network_access>
            <detachable>No</detachable>
          </concurrency>
          <remote_desktop_access>No</remote_desktop_access>
          <virtual_machine_access>Yes</virtual_machine_access>
        </license_properties>
        <detach_license_count>0</detach_license_count>
        <locked>Yes</locked>
      </feature>
      ...
    </product>
    <need_force>false</need_force>
  </key>
  ...
 </sentinel_ldk_info>
</C2V>
</ProtectionKeyOutput>
</ProtectionKey>
//----------
```

В этой структуре будет содержаться два тега с данными:
-> **system_fingerprint** - текущий набор аппаратных характеристик ПК;
-> **reference_fingerprint** - набор аппаратных характеристик ПК, к которым был “привязан” SL ключ при первоначальной активации.
Далее можно обработать полученные данные и вывести их клиенту, в удобном разработчику виде.

!ВАЖНО!: следует учитывать, что схема привязки указывается не для всего ключа в целом, а может быть своя для каждого продукта в ключе (определяется на этапе создания продукта в Sentinel EMS). То есть возможна ситуация, когда в ключе клиента есть например два продукта:

- Product #1 - имеет схему привязки PMType1 (привязка к HDD и Motherboard);
- Product #2 - имеет схему привязки FQDN (привязка к полному доменному имени ПК).

Если при этом на ПК сменить железо (например HDD и Motherboard), то Product #1 заблокируется с пометкой **Cloned** (при попытке выполнить логин на Feature ID в этом продукте будет возвращаться ошибка с кодом: 64 - CloneDetected).
А вот Product #2 НЕ заблокируется и продолжит работать, так как его параметр привязки (полное доменное имя ПК) не изменился.

8. Интеграция с биллинговыми системами

В ситуации когда разработчик хочет вести свои продажи через ту или иную онлайн площадку для продаж ПО, он может автоматизировать процесс покупки клиентом его программного обеспечения с последующим автоматическим выпуском и доставкой клиенту приобретенной лицензии. Для подобной (полной) интеграции нужно чтобы выбранная биллинговая платформа поддерживала работу со сторонними системами посредствам RESTFul API. Если она это поддерживает то можно организовать примерно следующую схему работы:
Клиент заходит в свой личный кабинет на сайте онлайн магазина программного обеспечения -> выбирает там ПО которое хочет приобрести, оформляет покупку -> оплачивает покупку -> биллинговая платформа получает подтверждение получения оплаты и отсылает на сервер Sentinel EMS разработчика запрос на генерацию требуемой лицензии (Entitlement'a) -> затем либо автоматически, либо после ручной модерации (тут уже по выбору разработчика) клиенту генерируется ключ активации и высылается на почту и/или отображается в его личном кабинете на сайте онлайн магазина программного обеспечения -> после чего клиент может перейти к процессу активации своего приобретённого программного продукта.

Для более простой интеграции с биллинговыми системами (не требующей какой-либо доработки) разработчику достаточно сгенерировать N-е количество ключей активации для его продуктов и загрузить их в личный кабинет в самой биллинговой платформе. В результате оплаты клиентом того или иного продукта с лицензиями, он будет получать в своём личном кабинете в биллинговой платформе очередной ключ активации из списка ранее загруженных разработчиком.

Полную англоязычную документацию по EMS Web Service API можно найти тут.