|  |  |  |
| --- | --- | --- |
|  | |  |
|  | | Engineering Design Document |
|  | |  |
|  | **Version: 1.3**  **Last Revised:**  **Author: Anatoliy Lokshin** | |

Table of Contents

[1 System Overview 4](#_Toc277943303)

[2 Assumptions and Dependencies 4](#_Toc277943304)

[3 Functional Design 4](#_Toc277943305)

[4 Technical Design 4](#_Toc277943306)

[4.1 Class diagram and description 4](#_Toc277943307)

[4.2 Interaction of components 14](#_Toc277943308)

[4.3 High-Level Design / Tasks 17](#_Toc277943309)

[5 Error list 18](#_Toc277943310)

[6 Outstanding Issues 19](#_Toc277943311)

Document Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Revision No.** | **Date** | **Description of Change** |
| **Anatoliy Lokshin** | 1.0 | 09/13/2010 | First revision |
| **Anatoliy Lokshin** | 1.1 | 10/05/2010 | Fixed design issues |
| **Anatoliy Lokshin** | 1.2 | 10/27/2010 | Minor fixes and extension |
| **Anatoliy Lokshin** | 1.3 | 11/19/2010 | Specified additional policy rules. Added API method to report statistics events. |

Functional Design Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Kyle Quest** | Architect |  |  |
| **Anatoliy Lokshin** | Development lead |  |  |
| **Yulia Kuchmai** | QA Representative | 11/18/2010 | 11/18/2010 |
|  |  |  |  |

Technical Design Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Kyle Quest** | Architect |  |  |
|  |  |  |  |
|  |  |  |  |

Engineering Task Worklist Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Anatoliy Lokshin** | Development lead |  |  |
|  |  |  |  |
|  |  |  |  |

References

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Author** | **Location** |
| MetraTech Security Framework Specification | Kyle Quest | http://seceng.metratech.com/gf/download/docmanfileversion/20/78/MtSecurityFrameworkSpec.doc |
|  |  |  |

Version Configuration

|  |  |  |
| --- | --- | --- |
| **Name** | **Version** | **Additional Comments** |
| MetraNet | 6.05 |  |
|  |  |  |

# System Overview

Short description of the existing requirement. Reference to the requirement document.

# Assumptions and Dependencies

Security Monitor depends on the configuration loader and custom exceptions. It is independent from other subsystems and does not use common interfaces and terms.

# Functional Design

N/A

# Technical Design

## Class diagram and description

A high-level class diagram for the Security Monitor subsystem is shown on the Figure 1.



Figure 1. Security Monitor class diagram

**SecurityEventType** enumeration enlists security event categories.

|  |  |
| --- | --- |
| **Member** | **Description** |
| Unknown | Unknown security event category. |
| WebRequestEventType | HTTP Request Events (Unexpected HTTP Commands; Attempts to Invoke Unsupported HTTP Methods; GET When Expecting POST; POST When Expecting GET; duplicate request parameters; unexpected number of request parameters; bad request parameter names; malformed request parameter values). |
| WebResponseEventType | HTTP Response Events (Unexpected response headers; unexpected response data). |
| AuthenticationEventType | Authentication Events (Use of multiple usernames; Multiple failed passwords; High rate of login attempts; Unexpected quantity of characters in username; Unexpected quantity of characters in password; Unexpected types of characters in username; Unexpected types of characters in password; Providing only the username; Providing only the password). |
| AccessControlEventType | Access Control Events (Modifying URL arguments within a GET for Direct Object Access attempts; Modifying parameters within a POST for Direct Object Access attempts; Force browsing attempts; Evading presentation access control through custom posts). |
| SessionEventType | Session Events (User login; User logout; User operation executed; Modifying existing cookies; Adding new cookies; Deleting existing cookies; Substituting another user's valid session ID or cookie; Source IP address changes during session). |
| InputDataProcessingEventType | Ingress Data Processing Events (Cross Site Scripting Attempt; Violations of implemented White Lists; Double encoded characters; Unexpected encoding used; Blacklist inspection for common SQL injection values; NULL byte character in file request; Carriage Return or Line Feed character in file request). |
| OutputDataProcessingEventType | Egress Data Processing Events (Detect abnormal quantity of returned records; Detect system shell command result information). |
| FileIoEventType | File I/O Events (Detect large individual files; Detect large number of file uploads). |
| UserActivityTrendEventType | User Trend Events (irregular use of application for different users and a common baseline across all users, speed of application use for different users and a common baseline across all users, frequency of site use for different users and a common baseline across all users, frequency of feature use for different users and a common baseline across all users, number of data processing security events for different users and a common baseline). |
| AppActivityTrendEventType | Application Trend Events (number of logouts, number of logins, number of logins vs. the number of explicit logouts, average transaction/operation count, normal deviation from a baseline logout counts, normal deviation from a baseline transaction/operation counts). |

**IPAddress** class represents an IPv4 or IPv6 IP address and is used for validating (parsing) of the string representations of addresses and comparisions of them.

**SecurityEvent** class represents an event data to be processed by the Security Monitor. An instance of the class can be created directly by the MetraTech application or it can be created within the Exception.Report extension method when the Exception is of BadInputDataException type. The class contains the following properties and methods:

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| EventType | Security event category. |
| SubsystemName | Name of the event source subsystem. |
| SubsystemCategoryName | Name of the event source subsystem’s category. |
| InputData | The data caused the security event. |
| Reason | The security event origination reason. |
| Timestamp | Date and time when the event happened. |
| Path | Specifies the operation’s path. It is a WEB page URL for WEB app. |
| HostName | Specifies a name of the application server where the app is hosted. |
| Message | Text message describing the event. |
| ClientAddress | A user IP address. |
| SessionId | App specific user’s session ID. |
| ClientInfo | Describes a user client (WEB browser for WEB apps). |
| StackTrace | A call stack for the code caused the event. |
| InputDataSize | Specifies a size of large data (sich as files) that is not stored to save a space. |
| Record(writer: SecurityEventWriter) | Calls the specified writer to record the event to storage. |

***SecurityEventWriter*** class – it’s an abstract base that provides an interface for events writers. It supplies a thread safety. It will implement IDisposable interface.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| CanWriteActions | Indicats the writer can log information about responding actions recommended by the monitor. |
| BeginRecord() | Accurate an exclusive lock. Starts writing of the event data. |
| EndRecord() | Finishes writing of the event data. Releases an exclusive lock. |
| Write(fieldName: string, value: string) | Writes a string value to the field with the specified name. |
| Write(fieldName: string, value int) | Writes an integer value to the field with the specified name. |
| Write(fieldName: string, value: bool) | Writes a Boolean value to the field with the specified name. |
| Write(fieldName: string, value: DateTime) | Writes a date&time value to the field with the specified name. |
| Write(policyAction: PolicyAction) | Writes an information about the security policy action recommended by the Security Monitor. Multiple actions can be linked to a single event. |

An idea with the writer is that the security event “knows” who to record itself using the writer’s public interface.

The following implementation of the class will be ***SecurityEventWriter*** created:

1. **CsvFileEventWriter** – writes the event data to a text file in CSV format. The file path is specified in the configuration. Optionally it can create rolls the output file when especial time is exceeded, i.e. it creates one file per day.
2. **SQLiteWriter** – writes the event data into SQLite database The database file path is specified in the configuration.
3. **EmailEventWriter** – (optional) creates a email message where the event data is written to the message body and sends it.

**EventFilter** class – allows filtering security events before recording. The filtering is based on the event category and/or event source subsystem and its category. Also a custom filter can be set as a delegate of the CustomFilterEventHandler type.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| EventType | Security event category. |
| SubsystemName | Name of the event source subsystem. |
| SubsystemCategoryName | Name of the event source subsystem’s category. |
| OnCustomFilter | Sets a custom filtering event handler. |
| IsMatched(securityEvent: SecurityEvent) | Determines if the specified event matched with the filter’s condition. |

**EventRecorderDefinition** class specifies how events have to be recorded. It represents a combination of the recorder (subclass of the **SecurityEventWriter**) and a set of filters combined with OR operation (i.e., when an event matches any of the filters it will be recorded). It will implement IDisposable interface.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| Initialize() | Reads the definition from the configuration. |
| Write(securityEvent: SecurityEvent) | Verifies whether the specified event matches with one of the filters and records it if matches. |
| AddFilter(filter: EventFilter) | Adds a specified filter to the event recorder definition. |
| RemoveFilters() | Remove all filters from the event recorder. |
| Id | A unique identifier of the log definition. |

**SecurityMonitorApi** class is responsible for processing all security events. Also it provides an API to setup custom filters and security action handlers. It has the following public methods:

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| ReportEvent(securityEvent: SecurityEvent) | Processes the specified security event. |
| AddPolicyActionHandler(idHandler: string, actionType: PolicyActionType, handler: ISecurityPolicyActionHandler) | Sets a handler with the specified unique ID to be called by the Security Monitor for actions with the specified PolicyActionType. It is possible to set one handler for multiple policy action types. |
| RemovePolicyActionHandler(idHandler: string) | Removes a handler with the specified ID. |
| GetRecorder(id: string): EventRecorderDefinition | Gets an event recorder definition with the specified ID. When the definition with such ID is not found, throws a **SubsystemInputParamException** exception. |
| ReportLogin() | Processes a successful user login security event. Must be called after the user authorization. |
| ReportLogin(userName: string, success: bool, reason: string) | Processes the user login security event. Can be used to report fallen logins. |
| ReportLogout() | Processes an explicit user logout security event. Must be called before user is logged out. |
| ReportFeatureUsage(featurePath: string) | Processes a feature usage security events. |
| ReportTransactionUsage(featurePath: string) | Processes a transaction usage security event. |
| ReportIrregularUsage(featurePath: string) | Processes an irregular feature usage security event. |
| ReportFileUpload(fileSize: long) | Processes a file upload security event. |

**SecurityPolicyEngine** class provides an events processing mechanism. It performs the processing using security policies specified via the configuration. **SecurityPolicyEngine** evaluates whether the event is a subject of any configured security policy and calls event handlers if it is.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| ProcessEvent(securityEvent: SecurityEvent, actions: PolicyAction[]): bool | Processes the specified security event. Returns true if the event matches any of configured security policies and false otherwise.  **When multiple matched policies require for the action of the same type, only first action is returned by the method!** |

**SecurityPolicy** class contains a policy rule set and definitions of actions those have to be performed if the processed event matches the rules. Unlike the event filters the policy rules are combined with AND operation (i.e., an event has to match all rules to be a subject of the policy).

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| Evaluate(securityEvent: SecurityEvent, actions: PolicyAction[]): bool | Evaluates if the event matches the policy rules and returns a list of actions to be performed in the response to the event if it does. |

***PolicyRule*** is an abstract class determining an interface that a particular rule class must implement.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| Evaluate(securityEvent: SecurityEvent): bool | Evaluates if the event matches the policy rule. |

The following implementations of the ***PolicyRule*** class will be created:

1. **EventSourceRule**: checks if the event belongs to the specified type (source subsystem and subsystem’s category).
2. **EventTypeRule**: checks if the event belongs to the **SecurityEventType**.
3. **EventRepeatThresholdRule**: checks if the event of the same type happened some number of times in the current user session.
4. **EventTimespanThresholdRule**: checks if the event of the same type happened in the specified time window either for entire app or for the current user only.
5. **IPChangeRule**: checks if the client’s IP address changed in the current session.
6. **ActionFrequencyRule**: checks how much time passed since an action of the especial type was recommended. Allows to set a maximum frequency of an especial action recommendations.
7. **EventRatioThresholdRule**: checks a ratio between a number of events from one specific source to a number of actions from another source in the specified time window for entire app or for the current user only.
8. **InputDataSizeRule**: checks an input data size.

**PolicyActionType** enumeration enlists possible action types.

|  |  |
| --- | --- |
| **Member** | **Description** |
| None | No action. |
| BlockOperation | Block operation – to block an individual user operation. |
| BlockUser | Block user – to block a specific user. |
| BlockAddress | Block address – to block a specific IP address. |
| Log | Log – to log an application security event. |
| RedirectOperation | Redirect Operation – to redirect an operation to an alternative location. |
| RedirectUser | Redirect User – to redirect all operations from a specific user to an alternative location. |
| LogoutUser | Logout user – to logout the current session user. |
| SendSecurityWarningToUser | User security warning – to send back a visual security warning to the current session user. |
| SendAdminNotification | Application administrator notification – to send an admin notification. |
| ChangeSessionParameter | Session parameter change – to change one of the current session parameters (e.g., change caching settings, change timeout settings, change security thresholds, etc). |

***PolicyAction*** is an abstract base class for all policy action definitions those are specific for each **PolicyActionType**.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| ActionType | Gets a **PolicyActionType** the action is intended for. |

There will be the following implementation of the class ***PolicyAction***:

1. **BlockOperationAction**
2. **BlockUserAction** – specifies a time the user is blocked for.
3. **BlockAddressAction**
4. **LogAction**
5. **RedirectOperationAction** – specifies a path of the operation to redirect the user to.
6. **RedirectUserAction** – specifies a path of the destination to redirect the user to.
7. **LogoutUserAction**
8. **SendSecurityWarningToUserAction** – specifies a message text to be sent to the user.
9. **SendAdminNotificationAction** – specifies a message text to be sent to the admin.
10. **ChangeSessionParameterAction** – specifies a name of the parameter to be changed.

**ILogAnalizerRepository** is an interface that determines methods to read and analyze recorded security events. It is used by policy rules and can be used by a prospective monitoring tool with UI. An implementation is used for analyses must be coupled with one of the configured event recorders.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| GetEventsNumberInSession (sessionId: string, subsystemName: string, categoryName: string): int | Determines a number of the events of the specified kind happened within the session with the specified ID. |
| GetEventsNumberInTimespan (time: TimeSpan, subsystemName: string, categoryName: string): int | Determines a number of the event of the specified kind happened within the specified time span. |

**LogAnalizerFactory** is factory class that creates and keeps an instance of the **ILogAnalizerRepository** implementation. The type of the implementation is specified via the configuration.

|  |  |
| --- | --- |
| **Properties/Methods** | **Description** |
| Analyzer | Gets an instance of the class that implements **ILogAnalizerRepository** interface. |

## Interaction of components

The high-level interaction flow is shown on Figure 2.

An external code creates an event instance and passes it into the monitor API. The event can be created either directly or with Exception.Report extension method.

**SecurityMonitorApi** class calls the policy engine to evaluate the event against security policies.

Each policy calls its rules and if the event matches all of them policy returns a list of actions associated with it.

The policy engine combines all actions from matched policies and calls action handlers associated with these actions by their type.

After all actions will be executed the event will be recorded to a storage.



Figure 2. High-level interaction flow.

Figure 3 shows how the security event is analyzed and Figure 4 shows how the event is recorded.



Figure 3. Security event analyses flow.



Figure 4. Security event recording flow.

## High-Level Design / Tasks

Strikethrough tasks are already completed.

1. ~~Data object classes (SecurityEventType, SecurityEvent, IPAddress) development.~~
2. ~~Security monitor logging infrastructure (SecurityEventWriter, EventRecorderDefinition, SecurityMonitorApi) development.~~
3. ~~Event recorder configuration development.~~
4. ~~CSV file recorder development.~~
5. ~~Database recorder development.~~
6. ~~Security policy engine infrastructure development (SecurityPolicyEngine, SecurityPolicy, PolicyRule, PolicyAction and its descendants).~~
7. ~~Security policy engine configuration development.~~
8. ~~Security policy rule classes development (EventSourceRule, EventTypeRule, EventRepeatThresholdRule, EventTimespanThresholdRule).~~
9. ~~Configuration for security rules development.~~
10. ~~Unit tests development.~~
11. Implement recommended actions recording to the SQLite database.
12. Implement storing info about the input data size.
13. Add insufficient event sources to the SQLite database dictionaries.
14. Implement additional security policy rules (IPChangeRule, ActionFrequencyRule, EventRatioThresholdRule, InputDataSizeRule).
15. Implement additional API methods.
16. Email recorder development.
17. Update user’s guide.

# Error list

The following table shows a listing of errors that can occur.

|  |  |  |  |
| --- | --- | --- | --- |
| **Error Code** | **Error Message** | **Description** | **Area** |
|  |  |  |  |
|  |  |  |  |

# Outstanding Issues

List all open issues regarding this document.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Date raised** | **Description and Resolution** | **Page/ Section** | **Raised by** | **Allocated to** | **Status** |
|  | 10/05/2010 | There is a possibility for some stage of processing to fail (throw an exception). What should we do if an exception is thrown when a security event was not written to the log for analyses?  Simply log that there was an exception during security event processing and discard the event. |  | Anatoliy Lokshin |  | Solved |
|  |  |  |  |  |  |  |