Syspeace Detector Providers for Developers

Prerequisites

The document **Syspeace Detector Providers for System Administrators** is a prerequisite for this document and explains many of the basic concept needed. Additionally, programming experience is required to create a Syspeace detector provider.

Required tools

To start programming a detector, you will need to create a .NET assembly targeting the .NET Framework version 4.0. You can use whichever tools needed, such as Visual Studio Express or higher, MonoDevelop, Xamarin Studio or SharpDevelop, and the .NET language of your choice, including C#, Visual Basic.NET, F# or C++/CLI.

A high-level overview of the basic steps

For more information, refer to more in-depth information later in this document.

Begin by creating a class library and adding the file Syspeace.ProviderInterface.dll as a reference.

Define a public subclass of BaseDetectorRule to represent the rules your detector will have for matching observation patterns. This rule class must have a public constructor taking no arguments.

Define a public subclass of BaseDetectorObservation to represent the observations your detector will collect. This observation class must have a public constructor taking no arguments. Make this class serializable by using the Serializable attribute.

Define a public class implementing the IDetector<TDetectorRule, TDetectorObservation> interface, substituting the rule and observation classes. Apply the DetectorAttribute attribute to this class.

Implement all methods required by the interface.

Implement the **listener code** – the code needed for your detector to collect or subscribe to the information about login attempts.

Store away the listener received in the StartObserver method, pass it to the listener code and use it to provide Syspeace with observations.

After building, package up the detector by making a folder with the same name as the class library assembly and a .provider extension – *AssemblyName*.provider – and putting all prerequisites and dependencies except for the Syspeace.Interfaces.dll assembly in this folder. Install this provider according to the instructions in **Syspeace Detector Providers for System Administrators**.

Extra information for the observation and rule

The observation and rule may keep extra information beyond the default information. This information will be saved by Syspeace and available to you just like the default information.

The default information kept by every observation is:

- UTCTimestamp: The point in time when the login attempt happened, in the UTC time zone.
- IPAddress: The IP address of the person who attempted to login.
- User: The username of the person who attempted to login.
- Success: Whether the person was logged in or not.

The default information kept by every rule is:

- Name: The name of the rule, for management purposes.
- Priority: The priority of the rule. The rule with the highest priority/lowest number is processed first.
- IsEnabled: Whether the rule is enabled or not.
- TriggerWindow and OccurrencesCount: During which length of time which number of failed logins are needed to trigger a block.
- LockoutDuration: For how long the block should last.

To keep extra information, use properties on the rule and observation classes as follows:

- Use public properties.
- Use the property types string, bool?, long?, DateTime?, string[] or long[]. (As always in .NET, the ? is shorthand for a nullable value type.) No other type is supported.
- Mark the properties with the attribute ExtraInfo. The constructor requires two parameters: the human-readable/friendly name and the store name.
  The store name is a terse name used to store the value of the property in the database and may only contain the letters a-z and numbers 0-9, start with a letter and be up to 50 characters long.
- If the value recorded is not readable, you can supply a converter method to format the value into a readable string. The ReadableTextConverterMethod property on the ExtraInfo attribute may be set to the name of a public static method in the same class, taking one parameter of the same type as the property and returning a string.

You may use properties that do not follow these rules, but they will not be kept as extra information.

Implementing the IDetector<,> interface

Apply the DetectorAttribute attribute to the implementing class

```
[Detector(99)] public class AcmeDetector...
```

Apply the DetectorAttribute attribute to the class. The parameter to the attribute is the order in which the detector should appear in the Settings window.

The FriendlyDetectorName property

```
string FriendlyDetectorName { get; }
```

A human-readable, short description. The settings pane used to configure the detector's rules will be named "*FriendlyDetectorName* rules".

For example, the Windows login detector provides "Windows login".

The DetectorType property

```
string DetectorType { get; }
```

A string in “reverse DNS” format to identify the detector and the observation.
For example, a test detector from Syspeace would provide the DetectorType “com.syspeace.TestDetector” since Syspeace is associated with the syspeace.com domain.

The GetDefaultRules method and UsageWarning property

```
string UsageWarning { get; }
TDetectorRule[] GetDefaultRules();
```

GetDefaultRules returns an array of rules to insert as the default rules.

If the detector could cause unwanted blocks in common environments, the default rules should not contain any enabled rules and UsageWarning should describe the risk. Otherwise, UsageWarning should be null.

The CanRunInEnvironment property

```
ProviderAvailabilityResult CanRunInEnvironment { get; }
```

Diagnoses whether the detector could be run and returns a ProviderAvailabilityResult. If a requirement is missing and unlikely to be resolved before Syspeace is stopped, ie the computer runs the wrong Windows version or a program is not installed, use the ProviderAvailabilityResult.IsUnavailableUnspecifiedReason value or the ProviderAvailabilityResult.IsUnavailableBecause method. If the detector can run, use the ProviderAvailabilityResult.IsAvailable value.

If the detector requires something that can be added when Syspeace has been started, for example if a file is being watched that may not exist from the beginning, the detector should say that it is available and keep looking for the file.

The UseLocation method

```
void UseLocation(string pathToDetectorFolder);
```

This method is called early on by Syspeace with the path to the detector’s folder, where it may keep configuration and other files.

The StartObserver method and the IDetectorObservationListener<> interface

```
void StartObserver(IDetectorObservationListener<TDetectorObservation> observationListener);
```

This method is called by Syspeace to provide a listener object. The detector should keep this object and use it to call one of the two RecordObservation methods.

```
interface IDetectorObservationListener<TDetectorObservation> {
        void RecordObservation(TDetectorObservation observation);
        void RecordObservation(TDetectorObservation observation, string reportingToken);
        CancellationToken CancellationToken { get; }
        void Log(string log, bool important = false);
}
```

If the detector watches the primary source of the observation information, it may use the RecordObservation method that does not take a reporting token to unconditionally record these observations.

If the detector gets reports with observation information sent to it by other applications, it should use the RecordObservation method that does take a reporting token to require the other applications to provide an accurate reporting token. Any observations that are attempted to be recorded without an accurate reporting token will be skipped.

The detector may also use the CancellationToken property of the listener object to determine when Syspeace is shutting down. For more information, see the MSDN documentation about .NET cancellation.

In addition, the detector may use the Log method to log information to the Detector settings pane.

The RuleMatchesObservation method

```
bool RuleMatchesObservation(TDetectorRule rule, TDetectorObservation observation);
```

Determines whether a rule matches an observation. Used when the observation provides extra information and the rule allows you to match that extra information.

The IsIgnorableDuplicate and IsIgnorableOtherReason methods

```
bool IsIgnorableDuplicate(TDetectorObservation earlierObservation, TDetectorObservation
laterObservation);
bool IsIgnorableOtherReason(TDetectorObservation observation);
```

Determines whether two observations that come after each other are duplicates and one should be ignored, and whether an observation should be ignored for some other reason.

If possible, simply do not record the ignorable or duplicate observation in the first place.

The FormatObservationInfo and FormatRuleInfo methods

```
KeyValuePair<string, string>[] FormatObservationInfo(TDetectorObservation observation);
KeyValuePair<string, string>[] FormatRuleInfo(TDetectorRule rule);
```

Formats the extra information given in an observation or a rule, as a series of human-readable labels and values in the intended order. Do not provide information about information that exists in every observation or rule, like the observation username or rule name. If no extra information is available, return an empty array.

The UserOrOtherOriginDescription and FormatCompositeAccountName method

```
string UserOrOtherOriginDescription(TDetectorObservation observation);
string FormatCompositeAccountName(TDetectorObservation observation);
```

The FormatCompositeAccountName method should return a formatted username, including other information, like including a Windows domain in front as “DOMAIN\user”, or just the username if no other information is necessary or available.

The UserOrOtherOriginDescription method should return the formatted username as provided by FormatCompositeAccountName, or in the case of user information being missing but replaceable with some other extra information from the observation, that information. For example, the SMTP Exchange detector provided by Syspeace does not record the user, but the name of the SMTP connector is used instead.

The GetAccessReportFormattedExtra method

```
string GetAccessReportFormattedExtra(TDetectorObservation observation);
```

Formats the most important extra information in the observation for inclusion in the Access Report result, or null if no extra information is kept.

Additional notes

IPv6 support

Syspeace does not currently support IPv6. The observation object will accept IPv6 and IPv4 addresses, but IPv6 observations will not be processed until IPv6 is supported.

IP and IPAddress properties in the observation class

The BaseObservation class provides an IP property, where the IP address is given as a string, and an IPAddress property, where the IP address is given as a System.Net.IPAddress object. If any of these are attempted to be set to something that is not a valid IPv4 or IPv6 address, they will instead be set to null. Both properties are available for convenience and are just two ways of looking at the same value.

Object reuse

Object reuse of observation and rule is not supported. A new observation must be created when a new observation is to be recorded and the default information properties cannot be changed after it has been recorded. Syspeace will detect object reuse and warn.