Prelude Developer Manual

1. [[PreludeForeword|Foreword]]
2. [[RepositoryAccess|Repository Access]]
3. **Developing On Prelude Components**
   1. Libprelude
      1. [[DevelLibpreludeFetchingEvents|Developing a "Fetching Events" Feature]]
   2. LibpreludeDB
      1. [[DevelLibpreludedbSelect|Querying the Prelude database]]
      2. [[DevelLibpreludedbInsert|Inserting into the Prelude database]]
   3. Prelude-Correlator
      1. [[DevelCorrelatorRule|Creating a correlation rule]]
   4. Prewikka
      1. [[How to create a plugin - Part 1 HelloWorld|Creating a basic plugin]]
      2. [[How to create a plugin - Part 2 Inventory|Creating a more advanced plugin]]
4. **Developing On Prelude Agents**
   1. General
      1. [[DevelAgentClassification|Agent Classification]]
   2. 3rd Party Agents
      1. [[DevelAgentBuilding|Building a Sensor]]
      2. [[DevelAgentDebugging|Debugging your Sensor]]
      3. [[DevelAgentQuickly|Developping a Sensor Quickly]]

Wiki Documentation Contributer's Manual

Introduction

Contributions to the Prelude documentation are greatly appreciated!

- **If english is not your first language** that is okay. We don't mind fixing small grammar issues.
- **Prior to adding a new page** on a topic look through the various other sections to make sure there is no information already being covered in that area of focus.
- **When you find a page you wish to edit** refer to the page "History" to see if the author is still working on it by his comments.

Page Layout

Here are some elements of the formatting syntax you should use for a quick start and efficient work:

Page title and Tables of Contents

- Page title

  h1. Page title

- Adding the Manual Table of Contents in your page:

  [[TracNav(TOCManualUser)]] or [[TracNav(TOCManualDevel)]]

- Adding the Page Table of Contents if your page is quite long:

  Table of Contents
  {{toc}} => left aligned toc
  or:
  {{>toc}} => right aligned toc

Chapters, Subchapters and font style

- Chapter title

h2. Chapter title

- Subchapter title

h3. Subchapter title

- **Important information**

Important information

- *Comments, Details*

Comments, Details

Lists and Preformatted Text

- A list of elements
  - element A
  - element B
  - element C

* A list of elements
** element A
** element B
** element C

- Source code snippets or configuration scripts. Ex:

./configure

- Use:

<pre>
./configure
</pre>

Links and Images

- **Hyperlinks are automatically created** for [[WikiPageNames]] and URLs. Wiki page links can be disabled by prepending an exclamation mark "!" character, such as WikiPageNames.
- **Best practice Prelude wiki page links:**
  - PreludeIDS official website
  - [[WikiPageNames|The wiki page names page]]

"PreludeIDS official website":http://www.prelude-ids.com/
[[WikiPageNames|The wiki page names page]]

- The simplest way to include an **image** is to upload it as attachment to the current page, and put the filename in a macro call like:

!picture.png!

Help and Validation

- **A good example of our standardized layout** is the page covering the [[PreludeManager|Prelude Manager]] configuration.
- Also take advantage of the **WikiFormatting** page to understand the various ways to format in this wiki.

- **Before saving the page**, fill in at least your name into the "Your email or username:" field.
- **Contact the documentation maintainer** if you have any question at <documentation(at)prelude-ids.com>

Building a Sensor

The example listed on this page make use of new feature provided since libprelude *0.9.19*. If you use an earlier version, please look at the [[DevelAgentBuildingOld]] page.

Using C

Building a sensor in C to communicate with the prelude manager is done in three building steps:
- Creating the Prelude client
- Using the low level IDMEF API
- Using the high level IDMEF API

We will describe each step and the functions you can use from the API accordingly.

Initializing the Prelude library

Before any operation are done using the prelude library, the user first need to initialize it using the *prelude_init()* function.

```
#include "prelude.h"

int ret;

ret = prelude_init(&argc, argv);
if ( ret < 0 ) {
        prelude_perror(ret, "unable to initialize the prelude library");
        return -1;
}
```

Creating the Prelude client

The first thing to do before using the Prelude library function is to create a prelude_client_t object. This object will be necessary for most of the work you are going to do with prelude. The creation of a *prelude_client_t* object involve several steps, including, but not limited to:
- Parsing specific Prelude command line / configuration file options
- Checking that the profile has been registered (the "my-analyzer" argument ahead).

The *prelude_client_new()* function should be used for this task.

```
int ret;
prelude_client_t *client;

ret = prelude_client_new(&client, "my-analyzer");
if ( ! client ) {
        prelude_perror(ret, "Unable to create a prelude client object");
        return -1;
}
```

Once the client is created, an you have everything setup. You will need to start your client. The *prelude_client_start()* function is responsible for this, and will trigger the connection to the configured manager, and send the initial client heartbeat.

```
ret = prelude_client_start(client);
if ( ret < 0 ) {
        prelude_perror(ret, "Unable to start prelude client");
        return -1;
}
```

The prelude library will also register an internal timer in order to send heartbeat message at the defined interval. Timer registered by the library itself or by the program will either be called automatically if the *PRELUDE_CLIENT_FLAGS_ASYNC_TIMER* is set, otherwise, the program is responssible for calling the *prelude_timer_wakeup()* function every second from it's main loop, in order to check the registered timer.

- PRELUDE_CLIENT_FLAGS_HEARTBEAT - Used for client to send heartbeat (this is the default).
- PRELUDE_CLIENT_FLAGS_ASYNC_SEND - Used if you want message to be sent asynchronously.
- PRELUDE_CLIENT_FLAGS_ASYNC_TIMER - Used if you want timer to be automatically called from the asynchronous thread.

```
ret = prelude_client_set_flags(client,
PRELUDE_CLIENT_FLAGS_ASYNC_SEND|PRELUDE_CLIENT_FLAGS_ASYNC_TIMER);
if ( ret < 0 ) {
        fprintf(stderr, "Unable to set asynchronous send and timer.\n");
        return -1;
}
```

Which IDMEF API to use, low or high level?

It all depend on your sensor requirement. Using the low level API is more error prone, and the resulting code will be harder to maintain, but the result will be faster and more optimized. On the other end, the higher level API is slower, but maintaining the code will be much easier.

Using the low level IDMEF API

Here is an example of creating different IDMEF object. The top level IDMEF object
is always *idmef_message_t*. You should refer to the IDMEF draft, or to the Prelude API documentation in order to get a complete listing of theses objet, or a description of what information an object carry.

```
idmef_message_t *idmef;

ret = idmef_message_new(&idmef);
if ( ret < 0 ) {
        prelude_perror(ret, "unable to create IDMEF message");
        return;
}

ret = idmef_message_new_alert(idmef, &alert);
if ( ret < 0 ) {
        prelude_perror(ret, "unable to create IDMEF alert");
        idmef_message_destroy(idmef);
        return;
}

ret = idmef_alert_new_classification(alert, &class);
if ( ret < 0 ) {
        prelude_perror(ret, "unable to create IDMEF classification");
        idmef_message_destroy(idmef);
        return;
}

ret = idmef_classification_new_text(class, &str);
if ( ret < 0 ) {
        prelude_perror(ret, "unable to create classification text");
        idmef_message_destroy(idmef);
        return;
}

prelude_string_set_constant(str, "My classification");
```

Using the high level IDMEF API

Here is an example of using the high level IDMEF API:

```
idmef_message_t *idmef;

ret = idmef_message_new(&idmef);
if ( ret < 0 )
        return -1;

idmef_message_set_string(idmef, "alert.classification.text", "My classification text");
idmef_message_set_string(idmef, "alert.classification.reference(0).name", "OSVDB-XXXX");
idmef_message_set_string(idmef, "alert.classification.reference(0).origin", "osvdb");
```

```
idmef_message_set_string(idmef, "alert.classification.reference(0).url", "http://my.url/");
```

Sending the IDMEF message

This function will trigger the sending of the *idmef_message_t* you created above. Once this function is called, you might safely destroy the message (even if you are using the asynchronous sending mode).

```
prelude_client_send_idmef(client, idmef);
idmef_message_destroy(idmef);
```

Destroying the client

In case the analyzer you are developing is not a persistant analyzer (meaning an analyzer that is not supposed to exit), it is important that you call the *prelude_client_destroy()* function prior to exiting. This function have the side effect of sending an heartbeat to the remote manager, as well as an information regarding the analyzer state.

This state information is important since an analyzer not reporting a successful exit status, or an analyzer which stop sending heartbeat at all will be reported as having a problem.

- PRELUDE_CLIENT_EXIT_STATUS_SUCCESS - Exiting the sensor is the expected behavior.
- PRELUDE_CLIENT_EXIT_STATUS_FAILURE - There is something wrong going on, notice the security analyst.

```
prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_SUCCESS);
```

As a side note, please remember that a persistant sensor should never use this function (except maybe if it is working in batch mode), unless it want to report the *PRELUDE_CLIENT_EXIT_STATUS_FAILURE* exit status. This is also the case if your persistant sensor is interrupted by a signal.

Using C++

```
#include <libprelude/prelude.hxx>

using namespace Prelude;

int main(int argc, char **argv)
{
        prelude_init(&argc, argv);

        ClientEasy client = ClientEasy("prelude-correlator", Client::IDMEF_WRITE);
        client.start();

        IDMEF idmef;

        // Classification
        idmef.set("alert.classification.text", "C++ Example");

        // Source
        idmef.set("alert.source(0).node.address(0).address", "10.0.0.1");

        // Target
        idmef.set("alert.target(0).node.address(0).address", "10.0.0.2");
        idmef.set("alert.target(1).node.address(0).address", "10.0.0.3");

        // Assessment
        idmef.set("alert.assessment.impact.severity", "low");
        idmef.set("alert.assessment.impact.completion", "failed");
        idmef.set("alert.assessment.impact.type", "recon");

        // Additional Data
        idmef.set("alert.additional_data(0).data", "something");

        client.sendIDMEF(idmef);
        prelude_deinit();
}
```

Using Perl

```
use strict;
use Prelude;

# Create a new Prelude client.
my $client = new Prelude::ClientEasy("MySensor");
$client->start();

# Create an IDMEF message
my $idmef = new Prelude::IDMEF();

# Classification
$idmef->set("alert.classification.text", "Perl Example");
$idmef->set("alert.source(0).node.address(0).address", "10.0.0.1");
$idmef->set("alert.target(0).node.address(0).address", "10.0.0.2");
$idmef->set("alert.target(1).node.address(0).address", "10.0.0.3");

# Assessment
$idmef->set("alert.assessment.impact.severity", "low");
$idmef->set("alert.assessment.impact.completion", "failed");
$idmef->set("alert.assessment.impact.type", "recon");

# Additional Data
$idmef->set("alert.additional_data(0).data", "something");

# Send the generated message
$client->sendIDMEF($idmef);
```

Using Python

```
import prelude

# Create a new Prelude client.
client = prelude.ClientEasy("MySensor")
client.start()

# Create the IDMEF message
idmef = prelude.IDMEF()

# Classification
idmef.set( "alert.classification.text", "Python Example")

# Source
idmef.set("alert.source(0).node.address(0).address", "10.0.0.1")

# Target
idmef.set("alert.target(0).node.address(0).address", "10.0.0.2")
idmef.set("alert.target(1).node.address(0).address", "10.0.0.3")

# Assessment
idmef.set("alert.assessment.impact.severity", "low")
idmef.set("alert.assessment.impact.completion", "failed")
idmef.set("alert.assessment.impact.type", "recon")

# Additional Data
idmef.set("alert.additional_data(0).data", "something")

client.sendIDMEF(idmef)
```

Using Ruby

```
require("Prelude")

# Create a new Prelude client.
client = Prelude::ClientEasy.new("MySensor")
```

```
client.start()

# Create the IDMEF message
idmef = Prelude::IDMEF.new()

# Classification
idmef.set( "alert.classification.text", "Ruby Example")

# Source
idmef.set("alert.source(0).node.address(0).address", "10.0.0.1")

# Target
idmef.set("alert.target(0).node.address(0).address", "10.0.0.2")
idmef.set("alert.target(1).node.address(0).address", "10.0.0.3")

# Assessment
idmef.set("alert.assessment.impact.severity", "low")
idmef.set("alert.assessment.impact.completion", "failed")
idmef.set("alert.assessment.impact.type", "recon")

# Additional Data
idmef.set("alert.additional_data(0).data", "something")

client.sendIDMEF(idmef)
```

Using Lua

```
require("prelude")

-- Create a new Prelude client.
client = prelude.ClientEasy("MySensor")
client:start()

-- Create the IDMEF message
idmef = prelude.IDMEF()

-- Classification
idmef:set( "alert.classification.text", "Lua Example")

-- Source
idmef:set("alert.source(0).node.address(0).address", "10.0.0.1")

-- Target
idmef:set("alert.target(0).node.address(0).address", "10.0.0.2")
idmef:set("alert.target(1).node.address(0).address", "10.0.0.3")

-- Assessment
idmef:set("alert.assessment.impact.severity", "low")
idmef:set("alert.assessment.impact.completion", "failed")
idmef:set("alert.assessment.impact.type", "recon")

-- Additional Data
idmef:set("alert.additional_data(0).data", "something")

client:sendIDMEF(idmef)
```

Documentation

- [[PreludeStandards|PreludeIDMEF]]

Agent Classification

[[TracNav(TOCManualDevel)]]

When it comes to analyzer development, you usually define your class like this:

```
#define ANALYZER_CLASS "Host and Network IDS"
...
ret = idmef_analyzer_new_class(analyzer, &string);
if ( ret < 0 )
        goto err;
prelude_string_set_constant(string, ANALYZER_CLASS)
...
```

Classify your Analyzer as "Host and Network IDS" is **evil**, and this is the subject of this page.

Why do we need to classify our Analyzer?

According to the IDMEF standard, the Analyzer class identifies the analyzer from which the Alert or Heartbeat message originates. In this analyzer class, we focus on the children "class", which is the class of analyzer software and/or hardware. A class is a classification in order to let the manager know the analyzer capabilities.

Capabilities vary from being a Host IDS, which can include an anti-virus, a log analyzer, an integrity checker; a Network IDS, which can be behavior based, signature based etc..

The idea behind setting the analyzer class type is useful for the human operator and the correlator. And that is for the second that we need to be accurate in the way we define the analyzer class.

How to classify?

The following strings should be used as a comma separated list:

	Long name		Short name		Description		Analyzer Examples	
	Network IDS		NIDS		Network Intrusion Detection System		Snort	
	Signatures based Network IDS		SNIDS		When the NIDS perform the analysis using signatures		Snort	
	Host IDS		HIDS		Host Intrusion Detection System		Samhain, Ossec, Prelude-lml	
	Intrusion Prevention System		IPS		System that block any intrusion pattern at the firewall level		Snort-inline	
	File Integrity Checker		FICHIDS		When the HIDS does files integrity checking		Samhain, Ossec	
	Integrity Checker		ICHIDS		Usually performs integrity checking on files		Samhain, Ossec	
	Log Analyzer		LHIDS		Analyze logs to extract alerts		Prelude LML, Ossec	
	Network Anti-Virus		NAV		The anti-virus is network based		Snort+Clamav	

	Host Anti-Virus		HAV		The anti-virus is host based		Clamav	
	Correlator		COR		Alerts correlator		Prelude correlator	
	Firewall		FW		Firewall		[[NuFW]]	
	Honeypot		HNP		Honeypot		Nepenthes	
	Software Monitoring		SMNT		Software Monitoring program		Nagios	
	Harware Monitoring		HMNT		Harware Monitoring program		Nagios, SNMP	
	Active Vulnerability Scanner		AVS		Active Vulnerability Scanner		Nessus	
	Passive Vulnerability Scanner		PVS		Passive Vulnerability Scanner		PVS	
	Alarm harware		ALHW		Alarm system for physical intrusions			
	Private Branch Exchange		PBX		Private Branch Exchange		Asterisk	

Few rules:

- The short name is not recommended. Use it only if you have > 3 capabilities.
- Host IDS is not mandatory when the analyzer class is obviously what a HIDS is supposed to do, such as "File Integrity Checker"
- If the table has information missing, please complete it

Example, for Ossec:

```
#define ANALYZER_CLASS "File Integrity Checker, Logs Analyzer"
```

List of sensors with their class

	Snort		NIDS	
	Prelude LML		Log Analyzer	
	Prelude Correlator		Correlator	
	[[NuFW]]		Firewall	
	Sancp		NIDS	
	Samhain		Integrity Checker	
	Nepenthes		Honeypot	
	Ossec		Host IDS, File Integrity Checker, Log Analyzer	

Debugging your sensor

This section provide some tips you might find useful for sensor debugging.

Making your program abort when an error is displayed

When your program use libprelude in a way that doesn't conform to libprelude API usage, libprelude (0.9.15 or higher) will raise a warning:

```
18 Aug 23:53:18 (process:6744) CRITICAL: assertion 'ptr' failed (idmef-tree-wrap.c:13974 idmef_analyzer_get_name)
```

This usually mean that you did something wrong, in the above case: calling *idmef_analyzer_get_name()* with a NULL analyzer argument.

You can use the **LIBPRELUDE_ABORT** environment variable to force your application to abort if a critical assertion trigger.
Before starting your program, use:

```
export LIBPRELUDE_ABORT=CRIT
```

Then if a critical assertion is raised, your program will abort:

```
$ ./your_program
18 Aug 23:53:18 (process:6744) CRITICAL: assertion 'xxx' failed (filename:line function)
Aborted (core dumped)
```

You can set the LIBPRELUDE_ABORT environment variable to any available logging priority:
- CRIT
- ERROR
- WARN
- INFO
- DEBUG

The CRIT priority will be used in case LIBPRELUDE_ABORT is set to an empty value.

Developing a Sensor Quickly

Table of Contents

This page explains how to develop a sensor from scratch in C in 10 minutes of time.

If you want more documentation on sensor development, please refer to the [[DevelAgentBuilding|Building a Sensor Page]].

Source code

Note: this example use an improved API available from libprelude *0.9.19*. If you are using an earlier libprelude version, please have a look at the [[DevelAgentQuicklyOld]] page.

```
#include <libprelude/prelude.h>

#define ANALYZER_NAME "simple-analyzer"

int main(int argc, char **argv)
{
    int ret;

    prelude_client_t *client;
    idmef_message_t *idmef;

    /* Prelude init */
    ret = prelude_init(&argc, argv);
    if ( ret < 0 ) {
        prelude_perror(ret, "unable to initialize the prelude library");
        return -1;
    }

    ret = prelude_client_new(&client, ANALYZER_NAME);
    if ( ! client ) {
        prelude_perror(ret, "Unable to create a prelude client object");
        return -1;
    }

    ret = prelude_client_start(client);
    if ( ret < 0 ) {
        prelude_perror(ret, "Unable to start prelude client");
        prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_FAILURE);
        return -1;
    }

    /* Idmef init */
    ret = idmef_message_new(&idmef);
```

```
    if ( ret < 0 ) {
        prelude_perror(ret, "Unable to create the IDMEF message");
        prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_FAILURE);
        return -1;
    }

    /*
         * Fill IDMEF message(note that error checking should ideally be performed on production code.
         */
        idmef_message_set_string(idmef, "alert.assessment.impact.description", "As you can see, this description is useless,
because it is describing an event that isn't one!");
        idmef_message_set_string(idmef, "alert.assessment.impact.severity", "info");
        idmef_message_set_string(idmef, "alert.assessment.impact.completion", "succeeded");
        idmef_message_set_string(idmef, "alert.classification.text", "This alert was sent from the simplest analyzer ever");

        idmef_message_set_string(idmef, "alert.source(0).user(1)", "L'homme araignee");

        idmef_message_set_string(idmef, "alert.additional_data(0).type", "string");
        idmef_message_set_string(idmef, "alert.additional_data(0).meaning", "Signature ID");
        idmef_message_set_string(idmef, "alert.additional_data(0).data", "1");

    prelude_client_send_idmef(client, idmef);
    idmef_message_destroy(idmef);

    prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_SUCCESS);

    return 0;
}
```

Makefile

```
CC=gcc
CFLAGS=$(shell libprelude-config --cflags)
LDFLAGS=$(shell libprelude-config --libs)

all: prelude-simplest-sensor.c
        $(CC) prelude-simplest-sensor.c -o prelude-simplest-sensor $(CFLAGS) $(LDFLAGS)
```

Registration

If prelude manager is on localhost, run:

```
prelude-admin register simple-analyzer "idmef:w" localhost --uid 1000 --gid 1000
```

and in an other terminal:

```
prelude-admin registration-server prelude-manager
```

Follow instructions, and your sensor is will be registered.

See the resulf of your alert

- When you fire up prewikka, you see a line containing what we specified as classification.text:
  alert-prewikka.png
- When clicking on this alert, you can see a detailed view of your alert:
  alert-detail.png

Enjoy!

Developing a Sensor Quickly

[[TracNav(TOCManualDevel)]]

Table of Contents

This page explains how to develop a sensor from scratch in C in 10 minutes of time.

If you want more documentation on sensor development, please refer to the [[DevelAgentBuilding|Building a Sensor Page]].

Source code

```
#include <libprelude/prelude.h>

#define ANALYZER_NAME "simple-analyzer"

static int
add_idmef_object(idmef_message_t *message, const char *object, const char *value)
{
        int ret;
        idmef_value_t *val;
        idmef_path_t *path;

        ret = idmef_path_new_fast(&path, object);
        if ( ret < 0 )
                return -1;

        ret = idmef_value_new_from_path(&val, path, value);
        if ( ret < 0 ) {
        printf("path = %s", object);
        prelude_perror(ret, "Unable to create the IDMEF value from path");
                idmef_path_destroy(path);
                return -1;
        }

        ret = idmef_path_set(path, message, val);
        if ( ret < 0 ) {
        prelude_perror(ret, "Unable to create to set the IDMEF path");
        idmef_value_destroy(val);
                idmef_path_destroy(path);
                return -1;
        }

        idmef_value_destroy(val);
        idmef_path_destroy(path);

        return ret;
}

int main(int argc, char **argv)
{
    int ret;

    prelude_client_t *client;
    idmef_message_t *idmef;

    /* Prelude init */
    ret = prelude_init(&argc, argv);
    if ( ret < 0 ) {
        prelude_perror(ret, "unable to initialize the prelude library");
        return -1;
    }

    ret = prelude_client_new(&client, ANALYZER_NAME);
    if ( ! client ) {
```

```
        prelude_perror(ret, "Unable to create a prelude client object");
        return -1;
    }

    ret = prelude_client_start(client);
    if ( ret < 0 ) {
        prelude_perror(ret, "Unable to start prelude client");
        prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_FAILURE);
        return -1;
    }

    /* Idmef init */
    ret = idmef_message_new(&idmef);
    if ( ret < 0 ) {
        prelude_perror(ret, "Unable to create the IDMEF message");
        prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_FAILURE);
        return -1;
    }

    /* Idmef stuff */
    /* We do not check return values, this is evil but makes this example clearer */
    /* In your code, please check and find a way to handle the return value */
        add_idmef_object(idmef, "alert.assessment.impact.description", "As you can see, this description is useless, because it is
describing an event that isn't one!");
        add_idmef_object(idmef, "alert.assessment.impact.severity", "info");
        add_idmef_object(idmef, "alert.assessment.impact.completion", "succeeded");
        add_idmef_object(idmef, "alert.classification.text", "This alert was sent from the simplest analyzer ever");

        add_idmef_object(idmef, "alert.source(0).user(1)", "L'homme araignee");

        add_idmef_object(idmef, "alert.additional_data(0).type", "string");
        add_idmef_object(idmef, "alert.additional_data(0).meaning", "Signature ID");
        add_idmef_object(idmef, "alert.additional_data(0).data", "1");

    prelude_client_send_idmef(client, idmef);
    idmef_message_destroy(idmef);

    prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_SUCCESS);

    return 0;
}
```

Makefile

```
CC=gcc
CFLAGS=@libprelude-config --cflags@
LDFLAGS=@libprelude-config --libs@

all: prelude-simplest-sensor.c
        $(CC) prelude-simplest-sensor.c -o prelude-simplest-sensor $(CFLAGS) $(LDFLAGS)
```

Registration

If prelude manager is on localhost, run:

```
prelude-admin register simple-analyzer "idmef:w" localhost --uid 1000 --gid 1000
```

and in an other terminal:

```
prelude-admin registration-server prelude-manager
```

Follow instructions, and your sensor is will be registered.

See the resulf of your alert

- When you fire up prewikka, you see a line containing what we specified as classification.text:
  alert-prewikka.png

- When clicking on this alert, you can see a detailed view of your alert:
  alert-detail.png

Enjoy!

Developing an Application Using Prelude Fetching Events Feature

The goal of this page is to help you to develop and understand the libprelude fetching events feature. First, raw code is shown, compilable and usable. Each function is explained latter.

Code

```
#include <libprelude/prelude.h>
#include <libprelude/prelude-message-id.h>
#include <libprelude/idmef-message-print.h>

#define CLIENT_PROFILE "PoolingTest"
#define MANAGER_ADDRESS "127.0.0.1"
#define POOL_TIMEOUT 2        /* set the alert pooling check at two seconds */

static prelude_client_t *client;

static int
initialize_prelude(void)
{
        int ret;
        char *buffer;

        ret = prelude_init(NULL, NULL);
        if ( ret < 0 ) {
                prelude_perror(ret, "error initializing the prelude library");
                return ret;
        }

        ret = prelude_client_new(&client, CLIENT_PROFILE);
        if ( ret < 0 ) {
                prelude_perror(ret, "error creating the prelude client");
                return ret;
        }

        prelude_client_set_required_permission(client, PRELUDE_CONNECTION_PERMISSION_IDMEF_READ);

    ret = prelude_client_init(client);
        if ( ret < 0 ) {
                prelude_perror(ret, "error initializing the prelude client");
                return ret;
        }

        return 0;
}

static const char *
pooling_get_classification(idmef_alert_t *alert)
{
        int ret;

        idmef_classification_t *classification;
        prelude_string_t *pstr;
```

```
        const char *buffer;

        classification = idmef_alert_get_classification(alert);
        if ( ! classification ) {
                printf("Cannot get classification from alert\n");
                return NULL;
        }
        pstr = idmef_classification_get_text(classification);
        if ( ! pstr ) {
                printf("Cannot get text from classification\n");
                return NULL;
        }

        buffer = prelude_string_get_string(pstr);

        return buffer;

}

static idmef_impact_t *
pooling_get_impact(idmef_alert_t *alert)
{

        int ret;

        idmef_assessment_t *assessment;
        idmef_impact_t *impact;

        assessment = idmef_alert_get_assessment(alert);
        if ( ! assessment ) {
                printf("Cannot get assessment from alert\n");
                return NULL;
        }
        impact = idmef_assessment_get_impact (assessment);
        if ( ! impact ) {
                printf("Cannot get impact from assessment\n");
                return NULL;
        }

        return impact;

}

static const char *
pooling_get_impact_description(idmef_impact_t *impact)
{

        prelude_string_t *pstr;
        const char *buffer;

        pstr = idmef_impact_get_description(impact);
        if ( ! impact ) {
                printf("Cannot get description from impact\n");
                return NULL;
        }

        buffer = prelude_string_get_string(pstr);

        return buffer;

}

static void
pooling_get_target_infos(idmef_alert_t *alert, const char **node_name, const char **node_addr)
{
        idmef_target_t *target = NULL;
        idmef_address_t *address = NULL;
```

```
        idmef_node_t   *node;

        prelude_string_t *pstr;
        const char *buffer;

        while ( (target = idmef_alert_get_next_target(alert, target)) ) {
                node = idmef_target_get_node(target);

                pstr = idmef_node_get_name(node);
                buffer = prelude_string_get_string(pstr);
                *node_name = buffer;

                while ( (address = idmef_node_get_next_address(node, address)) ) {

                        pstr = idmef_address_get_address(address);
                        buffer = prelude_string_get_string(pstr);
                        *node_addr = buffer;
                }
        }
}

static int
process_idmef_message(prelude_msg_t *msg)
{

        int ret;

        idmef_message_t         *idmef;
        idmef_message_type_t    msg_type;
        idmef_alert_t           *alert;
        idmef_impact_t          *impact;
        idmef_impact_severity_t *severity;
        idmef_target_t          *target;

        const char *classbuf;
        const char *descrbuf;
        const char *target_name;
        const char *target_addr;

        ret = idmef_message_new(&idmef);
        if ( ret < 0 )
                return ret;

        ret = idmef_message_read(idmef, msg);
        if ( ret < 0 ) {
                prelude_perror(ret, "error reading IDMEF message");
                return ret;
        }

        /* We only process the Alert class */
        msg_type = idmef_message_get_type(idmef);
        if ( msg_type != IDMEF_MESSAGE_TYPE_ALERT )
        return 0;

        if ( idmef ) {
                alert = idmef_message_get_alert(idmef);
                if ( ! alert ) {
                        printf("Cannot get alert from message\n");
                        return -1;
                }
        } else {
                return -1;
        }

        pooling_get_target_infos(alert, &target_name, &target_addr);

        classbuf = pooling_get_classification(alert);
```

```
        if ( ! classbuf ) classbuf="";

        impact   = pooling_get_impact(alert);
        if ( impact ) {
                descrbuf = pooling_get_impact_description(impact);
                if ( ! descrbuf ) descrbuf="";
                severity = idmef_impact_get_severity(impact);
                if ( ! severity ) return -1;

        printf("Classbuf=[%s],descrbuf=[%s], target_name=[%s], targer_addr=[%s]\n",
              classbuf, descrbuf, target_name, target_addr);
        } else {
                printf("Cannot get impact!\n");
        }

        idmef_message_destroy(idmef);

        return 0;

}

extern int event_cb(prelude_connection_pool_t *pool, prelude_connection_pool_event_t event,
                    prelude_connection_t *conn, void *extra)
{
        int ret;
        prelude_msg_t *msg = NULL;

        ret = prelude_connection_recv(conn, &msg);
        if ( ret < 0 ) {
                prelude_perror(ret, "couldn't read message");
                return ret;
        }

        if ( prelude_msg_get_tag(msg) != PRELUDE_MSG_IDMEF )
                return 0;

        return process_idmef_message(msg);
}

extern int
check_for_event(prelude_connection_pool_t *pool)
{
        int ret;

        if ( ! pool ) {
                printf("Cannot get data from the pool");
                return -1;
        }

        ret = prelude_connection_pool_check_event(pool, 0, event_cb, NULL);
        if ( ret < 0 ) {
                prelude_perror(ret, "error setting callback");
                return -1;
        }

}

extern int
connect_prelude(const char *manager_addr)
{
        int ret;
        prelude_connection_pool_t *pool = NULL;

        prelude_connection_pool_new(&pool,
                                    prelude_client_get_profile(client),
                                    prelude_client_get_required_permission(client));
```

```
    ret = prelude_connection_pool_set_connection_string(pool, manager_addr);
    if ( ret < 0 ) {
            prelude_perror(ret, "error setting new prelude connection string");
            return -1;
        }

    prelude_client_set_connection_pool(client, pool);

        ret = prelude_connection_pool_init(pool);
        if ( ret < 0 ) {
            prelude_perror(ret, "error initializing connection pool");
            return -1;
        }

        while (1) {
            sleep(POOL_TIMEOUT);
            check_for_event(pool);
        }

        prelude_client_destroy(client, PRELUDE_CLIENT_EXIT_STATUS_SUCCESS);

        return 0;
}

int main(void)
{
    int ret;

    ret = initialize_prelude();
        if ( ret < 0 ) {
            printf("Cannot initialize prelude\n");
        }

    ret = connect_prelude(MANAGER_ADDRESS);
        if ( ret < 0 ) {
            printf("Cannot connecting prelude\n");
        }

    return 0;
}
```

Compile

```
gcc pooling.c -o pooling `libprelude-config --pthread-cflags --libs`
```

Register

If the pooling user as UID 1000 and GID 1000:

```
sudo prelude-admin register [[PoolingTest]] "idmef:r" localhost --uid 1000 --gid 1000
```

Use

On one side we run prelude-lml, on the other, the manager. Everything is localhost.

Now we can connect the pooling to the manager:

```
$ ./pooling
Classbuf=[SUDO Command Executed],descrbuf=[User toady successfully executed the command '/bin/ls /' as root.],
target_name=[myhost], targer_addr=[127.0.1.1]
```

Repository Access

GIT Access

You can obtain the latest Prelude source tree using the open source version control system GIT. Be warned that it contains development code which might be unstable, and may not even compile properly. The advantage of using GIT, though, is that you can get the latest fixes and updates, without having to wait for the official releases.

The module you wish to check out must be specified as MODULENAME.

libpreludedb
libprelude
prelude-correlator
prelude-handbook
prelude-lml
prelude-manager
prelude-notify
prelude-pflogger
prewikka

- **Checking out the development branch**

If you are behing a proxy, it may be necessary to configure your proxy parameters:

- By an environment variable
  $ export http_proxy="http://<login>:<passwd>@<proxy>"

- Or directly in the GIT configuration :
  $ git config --global http.proxy http://<login>:<passwd>@<proxy>

If you don't recognize COMODO certificate utilisé par , you can disable SSL verification by export next environment variable :
export GIT_SSL_NO_VERIFY=true

After, you can clone the repositories.
$ git clone https://www.prelude-siem.org/git/<MODULENAME>.git/

Contributing Code

Contributions to the Prelude code source are greatly appreciated and we are thankful for any patches and new code made for the project.

All change checked in the Prelude repository have to be reviewed. This is done for maintainability reason and to keep the code base clean.

Please open a ticket for the concerned module and attach an unified diff to it.

Copyright

When contributing, please keep in mind that we ask you for the copyright of the code. The reason lies in the dual licensed nature of Prelude-SIEM.

This means that by contributing code for the Prelude-SIEM project, this code will automatically be copyrighted to CS-SI.

How to create a Prewikka plugin - Part 1 HelloWorld

This tutorial is based on **Prewikka 1.2.6** and **Python 2.X** (where X >= 6).
It will teach you how to create a simple plugin that adds a HelloWorld page to the Prewikka interface.
For a more complete example of a Prewikka plugin, you can visit [[How_to_create_a_plugin_-_Part_2_Inventory|this page]].

Introduction

You will need only two Python files for this plugin:

- the main file containing the plugin interesting content;
- the setup script for distributing and installing the plugin.

File hierarchy

```
|-- helloworld
|   `-- __init__.py
`-- setup.py
```

Creating a new plugin

Let's begin by creating a directory named **helloworld** and editing our main file inside it named **__init__.py**:

```
"""A HelloWorld plugin"""
from prewikka import view

class HelloWorld(view.View):
    """The view that displays Hello World"""
    plugin_name = "Hello World"
    plugin_description = "A plugin that says hello world!"
    plugin_version = "1.0.0"
    view_name = "Hello"
    view_section = "Hello"
```

Here we defined our plugin **HelloWorld** as a sub-class of **view.View** with the following attributes:
- *plugin_name*, *plugin_description* and *plugin_version*: this information will be displayed on the Plugins page.
- *view_name*: this is the text of the tab that will contain our HelloWorld view.
- *view_section* : this is the menu section in which the tab will appear. Since there is no Hello section defined yet, this section will be automatically added to the menu.

Adding content

Now that the plugin is created, some content must be added in order to greet the user with our message.
This is done through the *render()* function, inside our HelloWorld class:

```
    def render(self):
        return "<div>Hello World!</div>"
```

And that's it.

Writing the setup script

The last thing to do is writing the **setup.py** script at the root of the project. The file should look like this:

```
from setuptools import setup

setup(name="prewikka-helloworld",
      version="1.0.0",
      author="John Doe",
      author_email="john.doe@acme.com",
      url="https://prelude-siem.org",
      packages=["helloworld"],
      install_requires=["prewikka"],
      entry_points={
          "prewikka.views": [
              "HelloWorld = helloworld:HelloWorld",
          ],
      })
```

Not much to say here, but please note the important part: defining the entry point for our plugin.
As we want Prewikka to take our plugin into account, we should register it in the **prewikka.views** entry point.

Testing the plugin

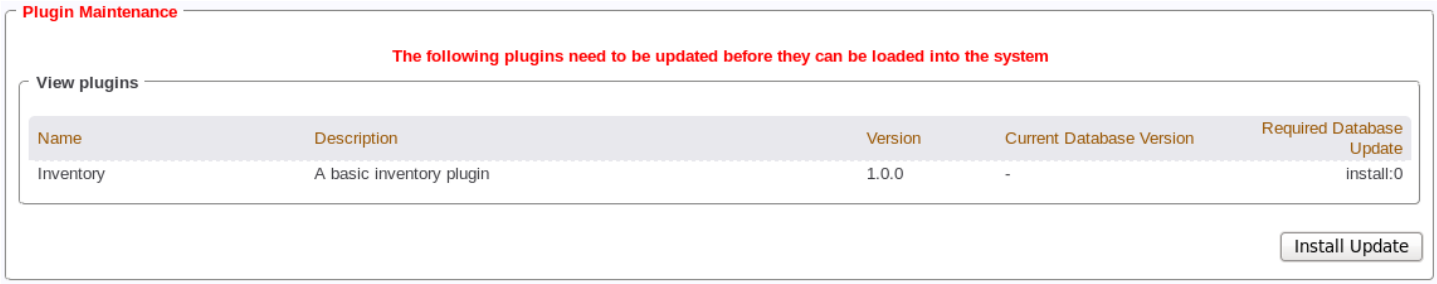
Installation

Let's install the plugin:

```
python setup.py install
```

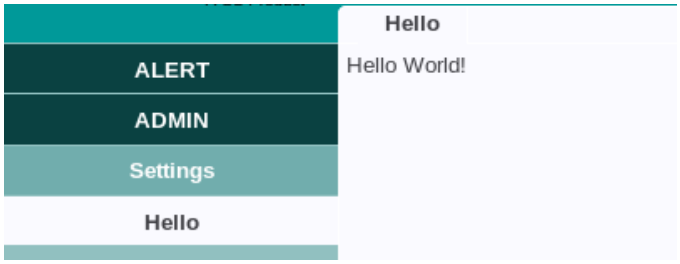
Plugins page

After a Prewikka restart, you can check that it has correctly been loaded by visiting the Plugins page:



HelloWorld page

And here is our HelloWorld page:



For a more complete example of a Prewikka plugin, please continue this tutorial [[How_to_create_a_plugin_-_Part_2_Inventory|here]].

How to create a plugin - Part 2 Inventory

This tutorial is based on **Prewikka 1.2.6** and **Python 2.X** (where X >= 6).
It will teach you how to create a plugin that adds a basic inventory view to the Prewikka interface. This can be useful to keep information about the computers and network equipment we want to supervise.
For a simpler example of a Prewikka plugin, you can visit [[How_to_create_a_plugin_-_Part_1_HelloWorld|this page]].

Introduction

In this tutorial we will learn how to:

- use a template file for displaying our page, including CSS and Javascript;
- define and use database tables for storing our inventory;
- create and handle a new configuration file;
- use the hook mechanism for more interactivity between different views.

File hierarchy

```
|-- inventory
|   |-- htdocs
|   |   |-- css
|   |   |   `-- inventory.css
|   |   `-- js
|   |       `-- inventory.js
|   |-- __init__.py
|   |-- sql
|   |   `-- install.py
|   `-- templates
|       |-- __init__.py
|       |-- inventory.py
```

```
|       `-- inventory.tmpl
`-- setup.py
```

Creating a new plugin

Like previously, let's begin by creating a directory named **inventory** and editing our main file inside it named **__init__.py**:

```
"""A basic inventory plugin"""
from prewikka import view

class Inventory(view.View):
    """The main Inventory view"""
    plugin_name = "Inventory"
    plugin_description = "A basic inventory plugin"
    plugin_version = "1.0.0"
    view_name = "Inventory"
    view_section = "Inventory"
```

And the **setup.py** script associated to it:

```
from setuptools import setup

setup(name="prewikka-inventory",
      version="1.0.0",
      author="Jane Doe",
      author_email="jane.doe@acme.com",
      url="https://prelude-siem.org",
      packages=["inventory"],
      install_requires=["prewikka"],
      entry_points={
          "prewikka.views": [
              "Inventory = inventory:Inventory",
          ],
      },
      package_data={
          "inventory": ["htdocs/css/*.css", "htdocs/js/*.js",
                        "sql/*.py", "templates/*.py"]
      })
```

Here you can see we added a *package_data* parameter to the setup, that will include all additional files necessary to our plugin.
From now on, if you want to see the plugin advancement before it is finished, you can directly jump to the [[How_to_create_a_plugin_-_Part_2_Inventory#Testing the plugin|last paragraph]].

Adding the database support

As we want to store data about our IT equipment, we will use the Prewikka database and create our own tables. To this purpose, we need to:

- define the SQL schema for the tables ;
- implement the Python functions to query the tables.

Defining the schema

The first task is done in a new file, **inventory/sql/install.py**:

```
from prewikka.database import SQLScript

class SQLUpdate(SQLScript):
    type = "install"
    version = "0"

    def run(self):
        self.query("""
DROP TABLE IF EXISTS Prewikka_Inventory;

CREATE TABLE Prewikka_Inventory (
    hostname TEXT NOT NULL,
    address TEXT NULL,
```

```
    os TEXT NULL
);
""")
```

Basically, we define the operations needed to create the inventory schema in version "0".
In this example, a single table named **Prewikka_Inventory** will be enough, that will store the name, the IP address and the operating system of the host.

Defining the API

The second task is done in the same **__init__.py** than previously but in a new class, named **InventoryDatabase**.
For a start, let's define a function to retrieve hosts from the inventory, and another function to add a new host into the inventory:

```
class InventoryDatabase(database.DatabaseHelper):
    """Handle database queries related to the inventory"""

    def get_hosts(self, keyword=None):
        """Return all hosts in the inventory database matching the keyword"""
        query = "SELECT hostname, address, os FROM Prewikka_Inventory"
        if keyword:
            query += (" WHERE hostname = %(keyword)s"
                      " OR address = %(keyword)s"
                      " OR os = %(keyword)s" %
                      {"keyword": self.escape(keyword)})
        return self.query(query)

    def add_host(self, hostname, address, os):
        """Add a host to the inventory database"""
        self.query("INSERT INTO Prewikka_Inventory (hostname, address, os) "
                   "VALUES (%s, %s, %s)" % (self.escape(hostname),
                                            self.escape(address),
                                            self.escape(os)))
```

Initializing the Inventory class

Make sure to import the **database** module in order to sub-class DatabaseHelper:

```
from prewikka import database, view

class Inventory(view.View):
    [...]
    plugin_database_version = "0"

    def __init__(self):
        view.View.__init__(self)
        self._db = InventoryDatabase()
```

Adding the template file

Creating the template

Now we have to create a **inventory/templates/inventory.tmpl** file based on Cheetah.
This is the template file for our inventory view, where we describe a table with three columns (hostname, address and os) and as many lines as returned by the Python code through the $inventory variable:

```
#filter CleanOutput
<fieldset>
 <legend>Inventory</legend>
 <table class="inventory">
  <tr>
   <th>Hostname</th>
   <th>Address</th>
   <th>OS</th>
  </tr>
  #set $classes = ["even", "odd"]
  #set $count = 0
```

```
    #for $host, $address, $os in $inventory
    <tr class="$classes[$count % 2]">
      <td>$host</td>
      <td>$address</td>
      <td>$os</td>
    </tr>
    #set $count += 1
    #end for
  </table>
</fieldset>
#end filter
```

A **inventory/templates/__init__.py** file should also be created with the following content:

```
from inventory import inventory
```

Defining the accepted parameters

By sub-classing view.Parameters, we are able to define which HTTP parameters are allowed for our inventory view, and which are mandatory.
The search parameter will be useful for querying the inventory, and the others (hostname, address and os) for adding a new host:

```
class InventoryParameters(view.Parameters):
    """A class that handles HTTP parameters for the Inventory view"""

    def register(self):
        self.optional("search", str)
        self.optional("hostname", str)
        self.optional("address", str)
        self.optional("os", str)
```

And the corresponding code in the **Inventory** class:

```
from . import templates

class Inventory(view.View):
    [...]
    view_template = templates.inventory
    view_parameters = InventoryParameters
    [...]

    def render(self):
        params = self.parameters
        if "hostname" in params:
            self._db.add_host(params.get("hostname"),
                              params.get("address"),
                              params.get("os"))
        self.dataset["inventory"] = self._db.get_hosts(params.get("search"))
```

You can see here that we initialize self.dataset["inventory"] to the host list. This variable can be used in the template as $inventory, as we previously did.

Including CSS and Javascript files

Defining the plugin_htdocs attribute

It is mandatory to define the **plugin_htdocs** attribute in Inventory class for the web server to search the CSS/Javascript files in the right place.

```
from pkg_resources import resource_filename

class Inventory(view.View):
    [...]
    plugin_htdocs = (("inventory", resource_filename(__name__, "htdocs")),)
    [...]
```

Creating the CSS

The **inventory/htdocs/css/inventory.css** file is fairly simple:

```
table.inventory {
    width: 100%
}
```

Creating the Javascript

Here is the **inventory/htdocs/js/inventory.js** file:

```
$(document).on("click", "button.create", function() {
    $("div.dialog").dialog({
        "buttons": {
            Ok: function() {
                $(this).find("form").submit();
            }
        }
    });
});
```

This jQuery snippet aims to open a dialog box when the user clicks on the creation button.
The button and form are yet to be added to the template for making that possible.

Updating the template

The template must be updated in order to include the CSS and Javascript files.
The second inclusion is done with LABjs.
Edit the **inventory/templates/inventory.tmpl** file to add these lines at the beginning:

```
<link rel="stylesheet" type="text/css" href="inventory/css/inventory.css">

<script type="text/javascript">
  \$LAB.script("inventory/js/inventory.js");
</script>
```

And these lines at the end:

```
<button class="create">Create</button>

<div class="dialog" title="Inventory" style="display: none;">
  <form>
    <table>
      <tr>
        <td>Hostname:</td>
        <td><input type="text" name="hostname"/></td>
      </tr>
      <tr>
        <td>Address:</td>
        <td><input type="text" name="address"/></td>
      </tr>
      <tr>
        <td>OS:</td>
        <td><input type="text" name="os"/></td>
      </tr>
    </table>
  </form>
</div>
```

Handling a custom configuration file

The main Prewikka configuration file is typically located in **/etc/prewikka/prewikka.conf**.
However, all .conf files located in **/etc/prewikka/conf.d/** are also imported by default, as we can see in the [include] section of the main config file:

```
[include]
conf.d/*.conf
```

We can therefore create our own **/etc/prewikka/conf.d/inventory.conf** which will contain the following lines:

```
[inventory]
title: My Inventory
```

The goal here is to allow the user to customize the title displayed on the inventory page.
So just add these lines to the Inventory class:

```
class Inventory(view.View):
    [...]

    def __init__(self):
        [...]
        self._config = getattr(env.config, "inventory", {})
        [...]

    def render(self):
        [...]
        self.dataset["title"] = self._config.get("title", "Inventory")
```

And replace in the template file:

```
<legend>Inventory</legend>
```

by:

```
<legend>$title</legend>
```

Implementing a hook

We're almost there. But because we want our plugin to be more practical, it would be a good idea to be able to search through the inventory from other Prewikka pages.
For example, how can we add a link to the inventory when clicking on a host in the AlertListing view?

This is where the hook mechanism comes into play.
Fortunately, a hook called **HOOK_LINK** has been created in AlertListing. This means that each plugin that registers to this hook will see its custom content inserted into the drop down menu appearing in the AlertListing view.

Let's do this by editing the Inventory class one last time:

```
class Inventory(view.View):
    def __init__(self):
        [...]
        env.hookmgr.declare_once("HOOK_LINK")
        env.hookmgr.register("HOOK_LINK", self._get_inventory_link)

    def _get_inventory_link(self, value):
        """Create a link to the inventory to be displayed in the alert view"""
        return ("host",
                "Search in inventory",
                utils.create_link(self.view_path, {"search": value}),
                False)
```

The **_get_inventory_link** function will be called from AlertListing with the clicked host as parameter, and returns a tuple (typ, linkname, link, widget) where:

- typ is the link type (only "host" is supported at the moment);
- linkname is the text of the link that will be displayed;
- link is the link to our inventory page;
- widget is a boolean controlling whether the link will be opened in a widget or not.

And let's not forget to import the necessary modules:

```
from prewikka import database, env, utils, view
```

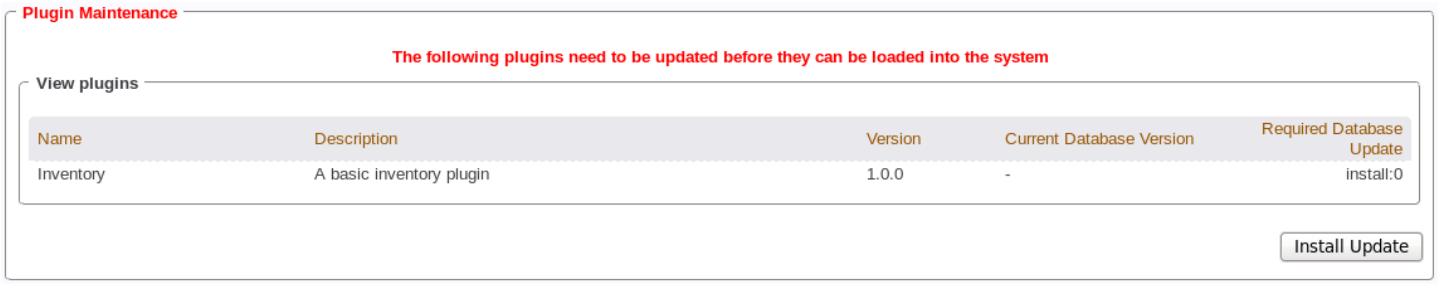
Testing the plugin

Installation

```
cheetah-compile inventory/templates/inventory.tmpl
python setup.py install
```

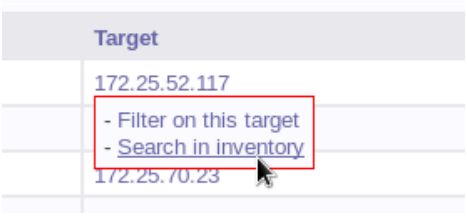
Plugins page

After a Prewikka restart, you must install your database schema by visiting the Plugins page:



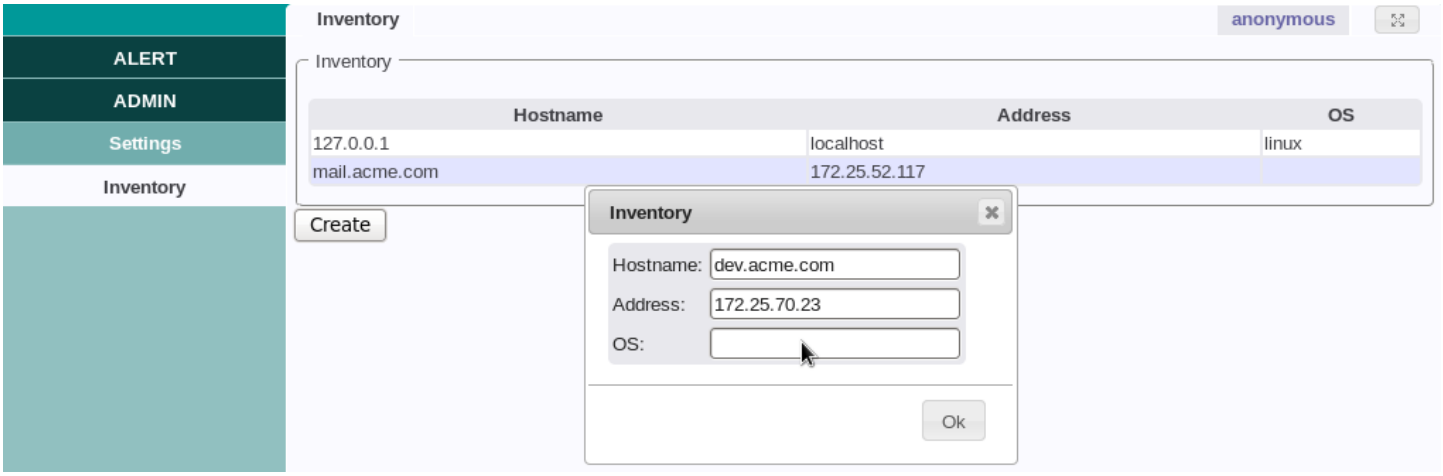
AlertListing page

The link to the inventory has appeared:



Inventory page

And here is our Inventory page:



Prewikka Apps

Create a plugin

- [[How to create a plugin - Part 1 HelloWorld]]

- [[How to create a plugin - Part 2 Inventory]]

Prelude Agent Contribution program

Object

The Prelude team is currently launching a campaign for the development of new security agents that "speak" IDMEF, on the occasion of the
adoption of IDMEF et IODEF by the French "General Interoperability Framework" (RGI) :

- IDMEF and IODEF in General Interoperability Framework v2 (French)
- General Interoperability Framework (French)

This campaign aims to encourage new software to be "IDMEF compatible" through Prelude. This page explains how to participate to this campaign.
Quick getting started :
- BASIC Agent
- EXPERT Agent

What is a Prelude agent ?

Any program/device logging potential security events can be a Prelude agent.

We distinguish between two types of Prelude agents.

- BASIC AGENT : The agent is producing log files and Prelude LML selects and normalize some of these logs into IDMEF Alerts to send them to the Prelude manager
- EXPERT AGENT : The agent is linked against the LibPrelude library with one of it's bindings and is generating and sending alerts to Prelude directly

Nota: A "security event" can be as simple as "Someone authenticate successfully on my software" unless your software as absolutely no security role, no sensible data, etc.

Example of existing Prelude agents :

- BASIC : OpenLDAP, Postgres, Apache, Cisco Router, selinux, ssh, etc.
- EXPERT : NIDS (Snort, Suricata), HIDS (Samhain, OSSEC), Auditd, Pam, etc.

Should I have the copyright on the agent ?

No if the software is open-source. This is the magic of open-source.

For BASIC agent you just have to have access to the log format, for EXPERT agent you can develop the connexion and propose it to the original software community (this is the way the Prelude Team is working)

For proprietary software, the BASIC agent is the same as open-source, for EXPERT agent you need to have access to the software code and contact Prelude to have access to a proprietary version of LibPrelude.

Where can I find informations about IDMEF ?

Please visit :

- The SECEF (SECurity Exchange Format) website : http://www.secef.net/
- The SECEF Redmine, with few tutorials : http://redmine.secef.net/
- The SECEF IDMEF RFC Browser : http://www.secef.net/idmef_parser/IDMEF/

What are the constraints ?

No constraints for basic agent.

To be an expert agent, your program's licence must be compatible with the GPLv2 licence

Nota : For proprietary softwares, please contact the prelude team : contact.prelude@c-s.fr

What is the difference between Basic and Expert Agent ?

Basic : Prelude Log Parser reads your log files, selects the "important" event, transform them in IDMEF and send them to Prelude manager

Advantages :
- No real coding
- No linking with LibPrelude
- Writting a parser file is quite easy (specially with Prelude Team help)

Inconvenients :
- IDMEF can contain only information from the log
- Heartbeats can't be sent so the agent can't be monitored in the manager

Expert : The software is linked to LibPrelude and can use the IDMEF API to connect to the manager and send IDMEF object

Advantages :
- The agent is fully connected to the manager through a secured channel (SSL),
- Heartbeats can be sent through this channel and the agent can be monitored in the manager GUI
- It is sometime possible to send more information than what is contained in the log
- No need for Prelude LML in the middle

Inconvenients
- LibPrelude must be linked to the software
- The software must be GPL V2 compliant
- There is some coding to be done

Nota: You can start with basic level and move to expert later

Shall I send all my logs through IDMEF ?

No !

That's the tricky part.

Only security relevant logs should be transform as IDMEF. The difference between "normal logs" and "IDMEF logs" is not allways easy to fix. It depend's on many factors starting by the nature of your program. The best thing to do about that is to ask in the Prelude Dev Forum and we will help you decide what should be send as IDMEF alerts.

Where should I start ?

If you want to develop an agent and your program is an open-source software, the best first step is to contact the prelude team through our Prelude Dev Forum so we can advice you on the relevancy of your choice, what best level would be (basic or expert) and how we can help you.

You can also install Prelude OSS and :

- Look how we work with famous software as BASIC agents like OpenLDAP, Apache, etc.
- Try to connect an EXPERT agent like Suricata to your Prelude installation
- You can also take a look at IDMEF implementation in Prelude compliants software like Suricata for example (the suricata agent is coded in C but other API are availables)

Some questions you should think about

What does your software do ?

- Is it a security software ?
- Is is "not" a security software but still producing few security informations ? Most of the software produce some security information like authentification logs for example.

What kind of security information is produced (in your logs)

Examples :
- authentication failed
- Authentication successfull
- Right granting
- Attempt to access to a forbiden ressource
- etc.

What is the format of those informations ?

Examples :
- syslog logs
- xml logs
- IDMEF files (some do !)
- specific log format (do you have documentation on this format ?)

Example of logs

Try to produce few examples of logs with what you think should be considered as security events and post it in the forum.

Should I go for BASIC or EXPERT ?

BASIC is good for nearly all sofware.

EXPERT should be more dedicated to software "specialised" in security.

A simple rule of the thumb is : The more your software generates "security related events" the more it should be an EXPERT agent.

Nota: You can start with BASIC and move to EXPERT later. BASIC will help you to understand and experiment what kind of information you should send as IDMEF without heavy coding. When you will have a good idea it will be easier to code the agent.

How to get help ?

Technical informations :
- BASIC AGENT
- EXPERT AGENT

For both level of implementation, don't hesitate to use the Prelude Dev Forum

List of existing agents

- BASIC AGENT : See Prelude LML rule files in the rules repository : Prelude LML Ruleset
- EXPERT AGENT : See this Third party agents

What about Copyright ?

A BASIC agent is a ruleset file included in Prelude LML ruleset package. So even if you are the one scripting it we will ask you (like any other Prelude contribution) to leave us the copyright so we can include it in the commercial edition of Prelude.

An EXPERT agent is code in the software agent. We won't need to keep the copyright on it, specially if you are ready to support the agent in time.

Any other question ?

Please use the Prelude Dev Forum

Source organization

Each Prelude projet have his own git repository :
- LibPrelude
  - With Web Browser
  - Tarball
  - Direct GIT access
- LibPreludeDB
  - With Web Browser
  - Tarball
  - Direct GIT access
- Prelude Manager
  - With Web Browser
  - Tarball
  - Direct GIT access
- Prelude LML
  - With Web Browser
  - Tarball
  - Direct GIT access

- Prelude LML-Rules
  - With Web Browser
  - Tarball
  - Direct GIT access
- Prelude Correlator
  - With Web Browser
  - Tarball
  - Direct GIT access
- Prewikka
  - With Web Browser
  - Tarball
  - Direct GIT access

Each GIT repository have :
- master branch : active development
- tags : for versioning and bug-fix

Development guidelines

Wiki Contribution

[[ContributeGuidelines|See this page]]

Development contributions

Source code must follow these guidelines :
- C modules (libprelude, libpreludedb, prelude-lml, prelude-manager)
  - Linux kernel guidelines
- Python modules (prewikka, prelude-correlator)
  - PEP 8

Libprelude API

{{iframe("https://www.prelude-siem.org/libprelude/")}}

Libprelude DB API

{{iframe("https://www.prelude-siem.org/libpreludedb/")}}

Querying the Prelude database

Note: The following examples are in Python, featuring the simplicity of the high-level libpreludeDB API.
For a complete overview of the low-level libpreludeDB API, go to [[LibpreludedbAPI]].

Initializing the database

```
import preludedb

sql = preludedb.SQL("type=mysql host=localhost name=prelude user=prelude pass=prelude")
db = preludedb.DB(sql)
```

Querying the database

The database can be queried directly through the sql object:

```
result = sql.query("SELECT messageid FROM Prelude_Alert LIMIT 10")
```

However, it is preferable to use the db object, for both simplicity and compatibility reasons:

```
result = db.getValues(["alert.messageid"], limit=10)
```

The getValues method takes the following arguments, only the first one being mandatory:
- a list of requested [[IDMEFPath|IDMEF paths]] with the following optional aggregation functions:
  - group_by
  - order_asc

    - order_desc
  - an [[IDMEFCriteria|IDMEF criterion]]
  - a boolean distinct option
  - an integer limit
  - an integer offset

For example:
```
result = db.getValues(["alert.classification.text/group_by"], "alert.assessment.impact.severity == 'high'", limit=10)
```

Iterating the results

The getValues() function returns a result representing the list of matched rows in the database. This list can be iterated, and each row inside it too. For example:
```
for row in result:
    for value in row:
        print value
```

Continuous Integration

{{iframe("https://www.prelude-siem.org/ci/builders")}}

Files

alert-detail.png	24.1 KB	04/29/2009	Yoann VANDOORSELAERE
alert-prewikka.png	4.99 KB	04/29/2009	Yoann VANDOORSELAERE
prelude-simplest-sensor.c	2.83 KB	08/22/2007	Sebastien Tricaud
alert-detail.png	24.1 KB	08/22/2007	Sebastien Tricaud
alert-prewikka.png	4.99 KB	08/22/2007	Sebastien Tricaud
plugins.png	10.9 KB	09/08/2015	Thomas ANDREJAK
helloworld.png	5.22 KB	09/08/2015	Thomas ANDREJAK
inventory.png	24.9 KB	09/08/2015	Thomas ANDREJAK
alertlisting.png	5.76 KB	09/08/2015	Thomas ANDREJAK
plugins.png	16.2 KB	09/09/2015	Antoine LUONG