

---

# Rematch Documentation

**Nir Izraeli**

**Nov 28, 2018**



---

## Contents:

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Installing Rematch Server . . . . .	3
1.2	Installing Rematch IDA Plugin . . . . .	4
<b>2</b>	<b>Usage</b>	<b>7</b>
2.1	Server . . . . .	7
2.2	IDA Plugin . . . . .	7
<b>3</b>	<b>Architecture</b>	<b>11</b>
3.1	Data Model . . . . .	11
3.2	Matching Process . . . . .	12
3.3	Vectors . . . . .	12
3.4	Matcher / Matching Engines . . . . .	12
3.5	Strategies . . . . .	12
<b>4</b>	<b>Frequently Asked Questions</b>	<b>13</b>
<b>5</b>	<b>Glossary</b>	<b>15</b>
<b>6</b>	<b>Goal of Rematch</b>	<b>17</b>
<b>7</b>	<b>Contribute and Get Support</b>	<b>19</b>
<b>8</b>	<b>License</b>	<b>21</b>



Rematch, a simple binary diffing framework that just works.

---

**Note:** At least, we hope it will be. Rematch is still a work in progress and is not fully functional at the moment. We're currently working on bringing up basic functionality. Check us out again soon or watch for updates!

---

Rematch is intended to be used by reverse engineers for revealing and identifying previously reverse engineered similar functions, and then migrating documentation and annotations to current IDB. Rematch does that by locally collecting and uploading data about functions in your IDB. Rematch uploads information to a web service (which you're supposed to set up as well), that upon request, is able to match your functions against all (or part) of existing database of previously uploaded functions and provide matches.

A secondary goal of rematch (which is not currently pursued) is to allow synchronization between multiple reverse engineers working on the same file.



The rematch project is composed of two parts: a server and an IDA plugin client.

While installing the plugin is extremely easy, installing the server tends to be a little more difficult. Luckily, it's only done once per organisation.

## 1.1 Installing Rematch Server

Installing a rematch server is only required once for a group of rematch users. Once an admin user is created, additional users can be managed through the admin console.

**Warning:** Since permissions are not currently enforced, it is advised that confidential data will be kept on servers only accessible to those with permission to access said data. See Privacy section for more details.

---

**Tip:** Windows based server installations are possible but not recommended. Some packages (scikit-learn, numpy, scipy) are required by the server but are more difficult to install on windows. Windows Subsystem for Linux may ease the installation process. Using Anaconda for python package management may also be helpful.

---

### 1.1.1 Building and running Rematch server docker container

We provide a docker container with Rematch server installed and configured with nginx, mysql, rabbitmq and celery micro components. This makes server deployment a lot easier, however a docker installation and roughly 600 MB of free space are required.

**Warning:** The docker setup uses the file `./server/.env` to set passwords. We highly recommend changing those before building your docker.

If you wish to build your own docker image from source, the docker-compose command can be used to build and fire up a docker, when run inside the root directory of the rematch repository:

```
$ service docker start ;
$ docker-compose -f ./server/docker-compose.yml build ;
$ docker-compose -f ./server/docker-compose.yml up -d ;
```

Then execute the following command to set up the rematch server administrator account:

```
$ docker-compose -f ./server/docker-compose.yml exec web ./server/manage.py l
→createsuperuser
```

Finally, point your browser to [http://SERVER\\_IP:8000/admin/](http://SERVER_IP:8000/admin/) to manage the service and add more users.

## 1.2 Installing Rematch IDA Plugin

Installing IDA plugins is done by placing the plugin source inside IDA's plugins directory (location is based on operating system). To make plugin installation as simple as possible, the rematch plugin has no dependencies.

Once installed the plugin automatically updates itself (as long as it's configured to), so installing the plugin is a one-time process.

### 1.2.1 Installing plugin using pip

If pip is installed for IDA's version of python, using it is the simplest installation method.

---

**Note:** By default, pip is not installed for Windows installations of IDA, but is more commonly found in Mac and Linux installations.

---

To install using IDA's pip, simply run the following pip command:

```
$ pip install rematch-idaplugin
```

**Warning:** Make sure you're installing the plugin using a version of pip inside IDA's copy of python.

If pip is not installed for IDA's version of python, it is still possible to install the plugin with another copy of pip using pip's `--target` flag. To do this run the following pip command line with any instance of pip:

```
$ pip install rematch-idaplugin --target="<Path to IDA's plugins directory>"
```

**Warning:** Using the pip `--target` flag with a pip version installed by Homebrew does not work because of a [known issue](#) with Homebrew. Homebrew OSX users will have to use a different installation method.

---

**Note:** IDA's plugins directory is located inside IDA's installation directory. For example if IDA is installed at:

C:\Program Files (x86)\IDA 6.9

Then the plugins directory will be:

C:\Program Files (x86)\IDA 6.9\plugins

and the executed command line should be:

```
$ pip install rematch-idaplugin
  --target="C:\Program Files (x86)\IDA 6.9\plugins"
```

### 1.2.2 Installing plugin manually

If you don't have pip, or prefer not to use it, you can still manually install the plugin by simply extracting the contents of the [idaplugin directory](#) in the repository's root, to IDA's plugins directory.

Simply download the package from [PyPI](#) or [Github](#) and extract the idaplugin directory contents into IDA's plugins directory, so that the file idaplugin/rematch\_plugin.py is located in the plugins sub-directory in IDA's installation directory.



In this page will go over basic rematch usage and functionality. We'll start with the server (which is currently limited in it's direct usability) and move on to the IDA plugin, where users will spend most of thier time.

## 2.1 Server

The rematch server is built on top of the [Django Framework](#), which has it's own built-in administration panel. The administration panel functionality Django exposes makes it trivial to manage database objects through the admin panel, granting admin users full control over the server. While Remach doesn't have its own web interface, it's common to use the admin panel to manage users and perform other tasks not currently available throught the rematch project.

Django's admin panel can be used for fine-grained control over most database objects (Such as [Vectors](#) and [Annotations](#)) through the rematch server, but it's main functionality is managing users, projects and files.

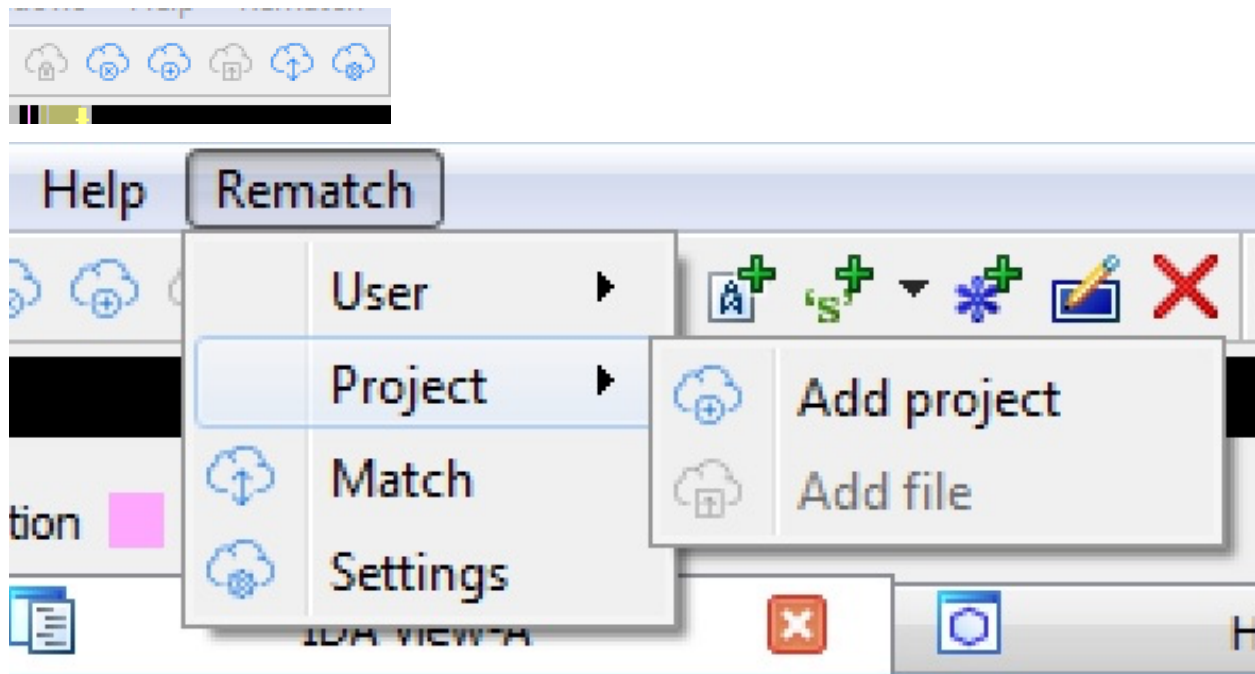
the admin panel is available at [https://SERVER\\_URL/admin/](https://SERVER_URL/admin/). Once logged in, you'll see the lists of database objects divided to categories. Selecting the "Users" object will show you a list of all registered users and a set of filters to filter by. You could edit, delete and create users which will then be able to login using the IDA plugin. Similarly, you can manage [Projects](#), [Files](#) and any other object stored on the server.

## 2.2 IDA Plugin

The IDA plugin is the interface to the rematch server. Most of the interaction a user has with rematch will actually be through the clients, dispatching work and uploading data to the server.

### 2.2.1 Toolbar and Menu

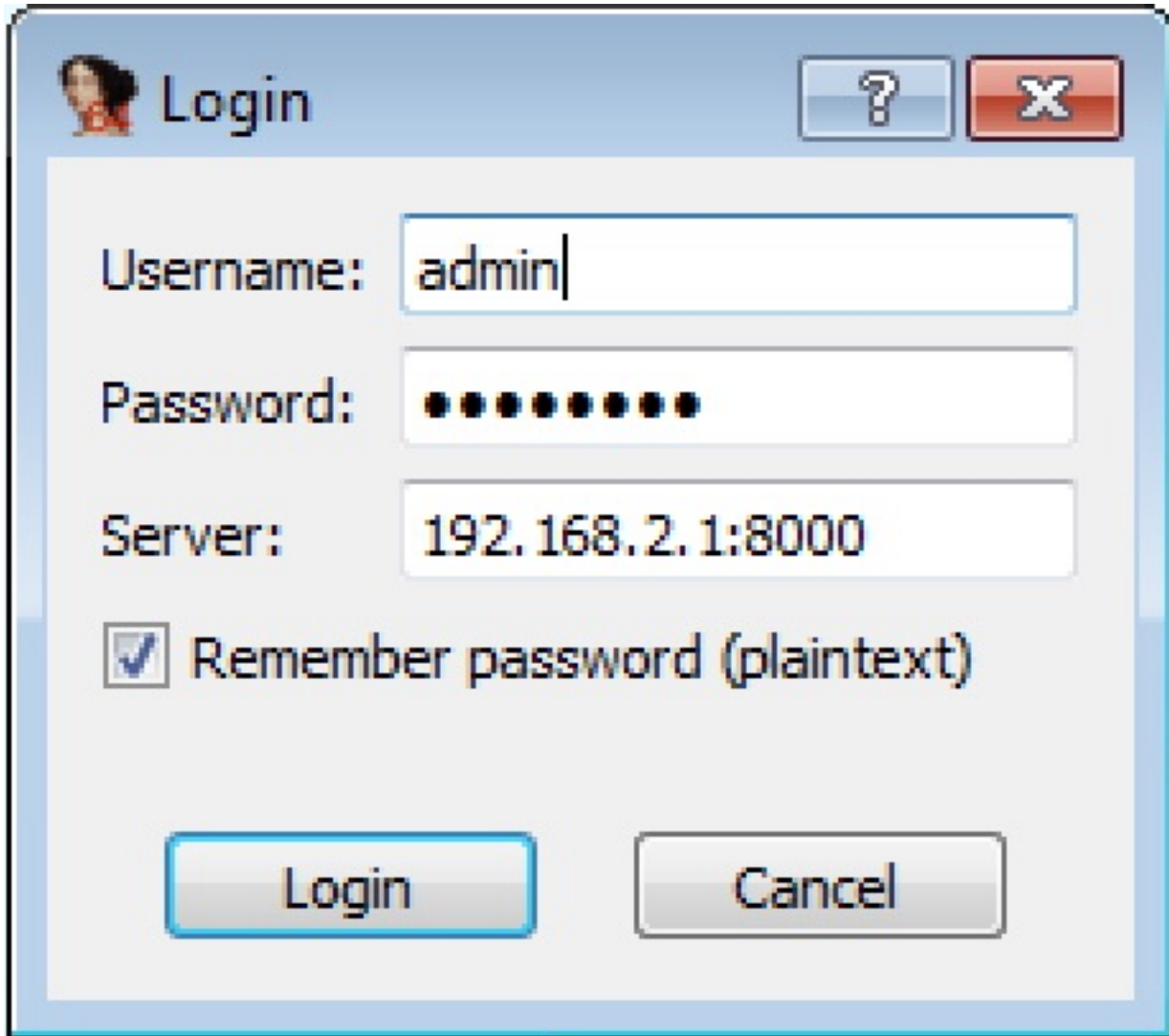
All rematch IDA plugin functionality is exposed through a set of toolbar and menu. Both have exactly the same functions and could be used interchangeably.



### 2.2.2 Login

Before uploading a file, starting a matching task or creating a project, a user must log in. If you do not have a user account on a rematch server, you'll need to contact the nearest rematch server admin, or set up your own rematch server.

By clicking the “Login” command in the rematch toolbar or menu, the following dialog box will appear:

A screenshot of a 'Login' dialog box. The dialog has a title bar with a small icon of a person, the text 'Login', and two buttons: a question mark '?' and a red 'X'. Inside the dialog, there are three text input fields. The first is labeled 'Username:' and contains the text 'admin'. The second is labeled 'Password:' and contains ten black dots. The third is labeled 'Server:' and contains the text '192.168.2.1:8000'. Below these fields is a checkbox that is checked, with the text 'Remember password (plaintext)' next to it. At the bottom of the dialog are two buttons: 'Login' and 'Cancel'.

Into which you'll need to specify the server address, your username and its password. You can optionally mark the "remember this password" checkbox to store the password for future logins. To login click the "Login" button and the login dialog will be closed.

**Warning:** Passwords are stored in plaintext, so make sure to only mark this checkbox on machines you trust. Rematch also supports token based login, and after a successful login a token will be stored automatically (unless disabled from the configuration dialog). For that reason, it is not required to store login details in most circumstances.

Upon a successful login you'll be able to create projects, add files, request matches, etc.

### 2.2.3 Projects, Files and Binding

---

**Todo:** Fill up the rest of idaplugin functionality

---

As discussed in detail on the [Architecture](#) page, *Files* can be created under *Projects*.

## **2.2.4 Matching and Data Upload**

## **2.2.5 Match Results and Filter Scripts**

The rematch solution is divided into two main parts: a client and a server. The server is in-charge of most of the heavy lifting, matching, and data storage. The client is collecting *Annotations* and *Vectors*, applying annotations after matches are displayed to the user and overall user interface.

Clients are designed to be replacable, however we only have an IDA client at the moment.

## 3.1 Data Model

### 3.1.1 Project Layout

A *Files* object is a database representation of a single binary instance being reverse engineered. While working on two distinctive versions of the same application, each of those versions should have a different File object for it's executable binary. If the application also has a single DLL, you should have 4 File objects. A good rule of thumb is that each File object should have an IDA IDB file.

In rematch, files are grouped together into *Projects*. The purpose of projects is completely left to the user. For example a project could be holding all versions of a single executable or all executables of a single application.

The obvious purpose of dividing files to projects is logical separation and ease of use, but another notable advantage is matching granularity. When starting a *Match Task*, the user is able to choose to match the current file against either a single other file, all files in it's project, all files in another project or against the entire database. This is so a user could create a single project for all files of a specific application version and another project for a different version. Then, to only get matches from the previous version, a single remote project match is performed. Alternatively, a single project can hold multiple versions of a single library (or all libraries with a similar functionality) and then requesting matches between an malware executable and all SSL libraries.

### 3.1.2 File Binding

Binding a file means an IDB will be associated with a specific *File* object in the remote server. This lets rematch automatically identify the database object describing the current IDB. This is how matches are made and are linked to a specific IDB, this is how caching of uploaded *Vectors* and *Annotations* is done.

File bindings are embedded inside IDB files, which means multiple copies of the same IDB file will share their File database objects.

### 3.1.3 File Version

---

**Todo:** Document the file version concept

---

## 3.2 Matching Process

Matches are made using three entity types defined throughout the rematch project:

- *Vectors*
- *Matchers*
- *Strategies*

## 3.3 Vectors

---

**Todo:** explain what vectors are

---

---

**Todo:** document existing vectors

---

## 3.4 Matcher / Matching Engines

---

**Todo:** explain what matchers are

---

---

**Todo:** document existing engines

---

## 3.5 Strategies

Strategies control the way multiple *Matchers* are used together, which *Instances* are matched against which and other similar logical decisions that may have significant implications on the overall outcome of the matching process.

For example, one could wish to match all instances against all other instances, in an “All VS All” kind of way. This is the “All” Strategy. However when comparing big databases, one may point out matching 5-byte and 1000-byte long functions to each-other is redundant, as those are highly unlikely to match. Therefore, “Binning” functions and only matching the bins might speed up the matching process without causing a decrease in match accuracy, as it may reduce a lot of unnecessary matches. This is called the “Binning strategy”.

## CHAPTER 4

---

### Frequently Asked Questions

---



---

Sorted alphabetically

**Annotation** A piece of information describing an *Instance* in more detail, often created by the user while reverse engineering as part of the reverse engineering process. Annotations help the reverse engineer and therefore there's an advantage in applying annotations to matched *Instances*.

### Engine

---

**Todo:** TODO

---

### File

---

**Todo:** TODO

---

**Instance** When used throughout these docs, an Instance generally means a matchable object inside a binary file, or its representation in any rematch component.

The following are currently entities:

1. A function defined within a binary executable.
2. A function imported into a binary file from another binary.
3. A stream of initialised data or structure.
4. A stream of uninitialised data or structure.

### Project

---

**Todo:** TODO

---

**Vector** Raw data used to describe an *Instance* in a way that facilitates and enables matching. Those are also occasionally called features in data-science and machine learning circles.

**Matcher** Matchers implement the logic of matching *Instances* together using thier *Vectors*.

### Match Task

---

**Todo:** TODO

---

**Strategy** Strategies control the way multiple *Matchers* are used together, which *Instances* are matched against which and other similar logical decisions that may have significant implications on the overall outcome of the matching process.

## CHAPTER 6

---

### Goal of Rematch

---

The goal of Rematch is to act as a maintained, extendable, open source tool for advanced assembly function-level binary comparison and matching. Rematch will be a completely open source and free (as in speech) community-driven tool. We support bottom-up organizational methods and desire Rematch to be heavily influenced by it's users (both in decision making and development).

We've noticed that although there are more than a handful of existing binary matching tools, there's no one tool that provides all of the following:

1. Open source and community driven.
2. Supports advanced matching algorithms (ML included <sup>TM</sup>).
3. Fully integrated into IDA.
4. Allows managing multiple projects in a single location.
5. Enables out of the box one vs. many matches.
6. Actively maintained.



---

### Contribute and Get Support

---

Here are a few useful links to help you get in contact, find help and participate:

- Source Code: <https://github.com/nirizr/rematch>
- Issue Tracker: <https://github.com/nirizr/rematch/issues>
- IRC channel: <ircs://freenode/#rematch> (or insecure)
- Telegram Chat: <https://t.me/joinchat/ASYl0wzNqvkJjjqbOOJnhA>

In particular, we're having trouble with implementing a stylish web interface for the rematch server, enriching this documentation and the testing framework. Engine suggestions (and obviously PRs) are extremely welcome, as well as feature requests. We tend to be permissive about approving PRs, but you can always raise an issue or discussion about a feature before implementing it.



## CHAPTER 8

---

### License

---

The project is licensed under the GNU-GPL v3 license.



## A

Annotation, [15](#)

## E

Engine, [15](#)

## F

File, [15](#)

## I

Instance, [15](#)

## M

Match Task, [16](#)

Matcher, [16](#)

## P

Project, [15](#)

## S

Strategy, [16](#)

## V

Vector, [15](#)