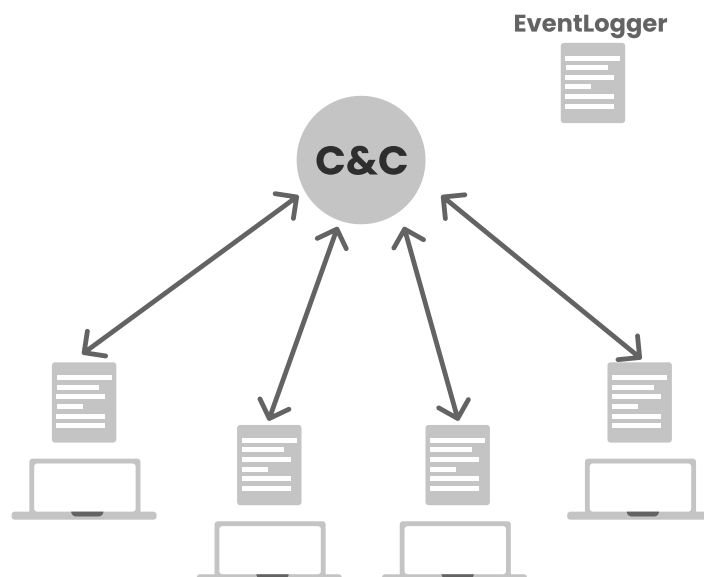


EventLogger for Windows

The project is composed of two parts:

- **EventLogger**: a [C#](#) script that logs mouse, keyboard and process events and sends them to the command and control server.
- **C&C Server**: the command and control web-server written [Python 3](#).



EventLogger

The **EventLogger** use the [Windows API](#) to log events. It logs the following events:

- *Mouse events*: MouseClick and MouseDoubleClick only
- *Keyboard events*: using the Windows key enumeration format (see [here](#))
- *Process events*: using the process name and the timestamp.

The logger has three main components:

- **Constants**: a class with the constants used by the logger.
- **Logger**: a class that implements the logging functionality.
- **EventLogger**: a class that implements the main functionality of the logger.

The **constants** class isn't very interesting, it contains just the constants used by the logger including the dynamic settings that can be changed in the **C&C Server**.

The **Logger** implements the logging functionality, including the creation of the log file. The logger stores all the events in a string variable called **_line** and periodically clears it to avoid memory overflow. There are two conditions to clear the **_line** variable:

- reaching the maximum size of events: in this case the **Logger** class saves the events in a file.
- use the **Server** api to send the events to the server.

The **EventLogger** class that is the main class. After setting up the methods that catch the events, it starts a thread that every **SECONDS_API_INVOKE** (a dynamic setting) calls the **Server** api to send the events. There are four dynamic settings:

```
public int SECONDS_API_INVOKE = 10;
public bool LOG_PROCESS_ON_DOUBLE_CLICK = true;
public bool LOG_MOUSE_EVENTS = true;
public bool LOG_KEYBOARD_EVENTS = true;
```

Thread that periodically calls the C&C server api

```
try{
    while (true){
        Thread.Sleep(config.SECONDS_API_INVOKE * 1000);
        logger.SendLog();
    }
}
catch (Exception ex){
    Console.WriteLine(ex);
}
```

- `LOG_PROCESS_ON_DOUBLE_CLICK`: if true, the `EventLogger` logs the process name and the timestamp after a double click event.
- `LOG_MOUSE_EVENTS`: if true, the `EventLogger` logs the mouse events.
- `LOG_KEYBOARD_EVENTS`: if true, the `EventLogger` logs the keyboard events.
- `SECONDS_API_INVOKE`: the time in seconds between two invocations of the api.

C&C Server

The `C&C Server` is a web-server written in Python 3 using the [Django](#) framework, using SQLite as the database. There are two apps, one for the `UI interface` and the other for the `EventLogger` api. In the `EventLogger` section is the api used to get the events from the `EventLogger` and convert every single event to a format compatible for the UI interface.

The `UI interface` helps to manage many different `EventLogger` instances, each one with its own settings. The interface basically allow the attacker to:

- Create new `EventLogger` instances
- View the events captured by the instance
- View some graphs of the events
- Change the dynamic settings a specific `EventLogger` instance

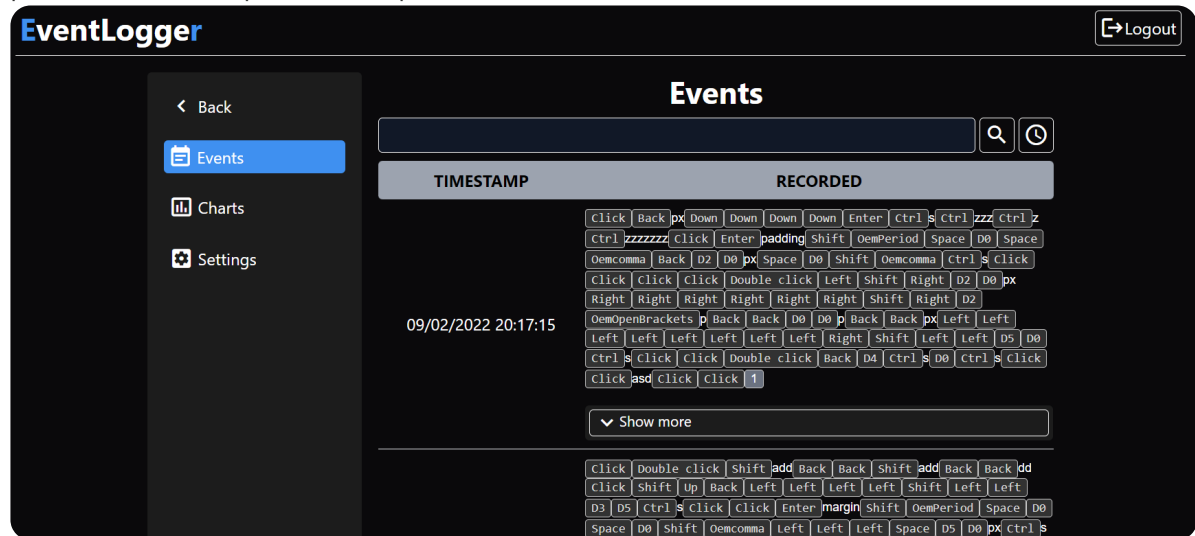
Homepage

Here are all the `EventLogger` instances, for each one there is a link to the app where it is possible to view the events, the graphs and change the dynamic settings.

EventLogger Logout			
<input type="text" value="Search by name..."/> <input type="button" value="Q"/> <input type="button" value="User +"/>			
LOGGER NAME	API KEY	LAST UPDATE	ACTIONS
LOGGER_1	AA91BVSDF	09/02/2022 20:17:15	<input type="button" value="Open"/> <input type="button" value="Delete"/>
LOGGER_2	VR4YJ5LR6K	10/02/2022 16:38:48	<input type="button" value="Open"/> <input type="button" value="Delete"/>

Events view

These are all the events of the `EventLogger`, the events are sorted by timestamp. It is also possible to see the processes captured in a record.

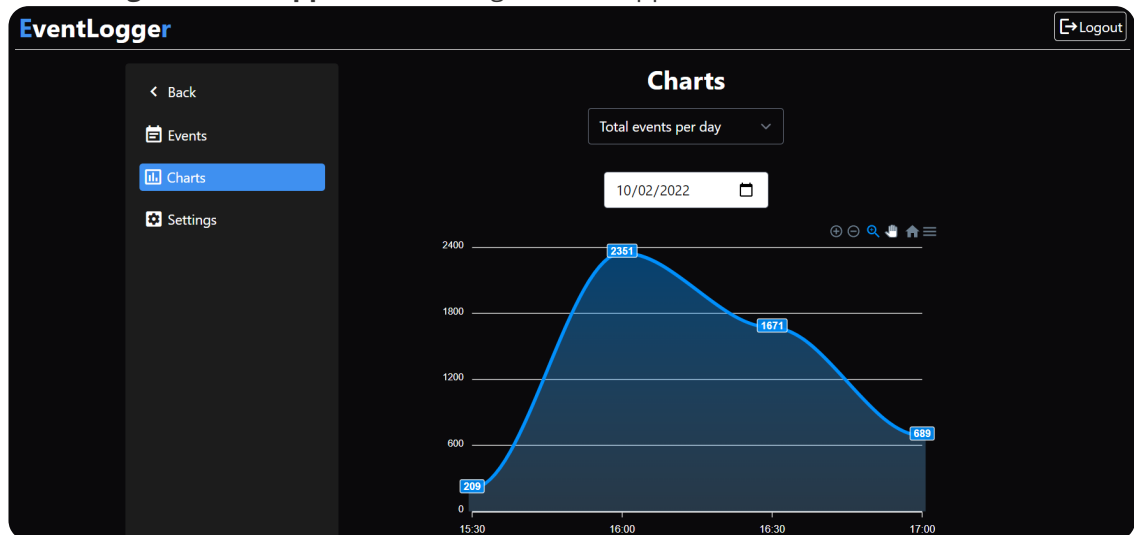


The screenshot shows the 'EventLogger' application interface. On the left is a sidebar with 'Back', 'Events' (selected), 'Charts', and 'Settings'. The main area is titled 'Events' and contains a search bar and a 'Logout' button. Below these is a table with two columns: 'TIMESTAMP' and 'RECORDED'. The 'TIMESTAMP' column shows the date and time '09/02/2022 20:17:15'. The 'RECORDED' column displays a list of keyboard events, such as 'Click', 'Back', 'px', 'Down', 'Enter', 'Ctrl', 's', 'Ctrl', 'z', 'Ctrl', 'z', 'Click', 'Enter', 'padding', 'Shift', 'OemPeriod', 'Space', 'D0', 'Space', 'Oemcomma', 'Back', 'D2', 'D0', 'px', 'Space', 'D0', 'Shift', 'Oemcomma', 'Ctrl', 's', 'Click', 'Click', 'Click', 'Double click', 'Left', 'Shift', 'Right', 'D2', 'D0', 'px', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Shift', 'Right', 'D2', 'OemOpenBrackets', 'p', 'Back', 'Back', 'D0', 'D0', 'p', 'Back', 'Back', 'px', 'Left', 'Left', 'Left', 'Left', 'Left', 'Left', 'Right', 'Shift', 'Left', 'Left', 'D5', 'D0', 'Ctrl', 's', 'Click', 'Click', 'Double click', 'Back', 'D4', 'Ctrl', 's', 'D0', 'Ctrl', 's', 'Click', 'Click', 'asd', 'Click', 'Click', '1'. A 'Show more' button is located below the list.

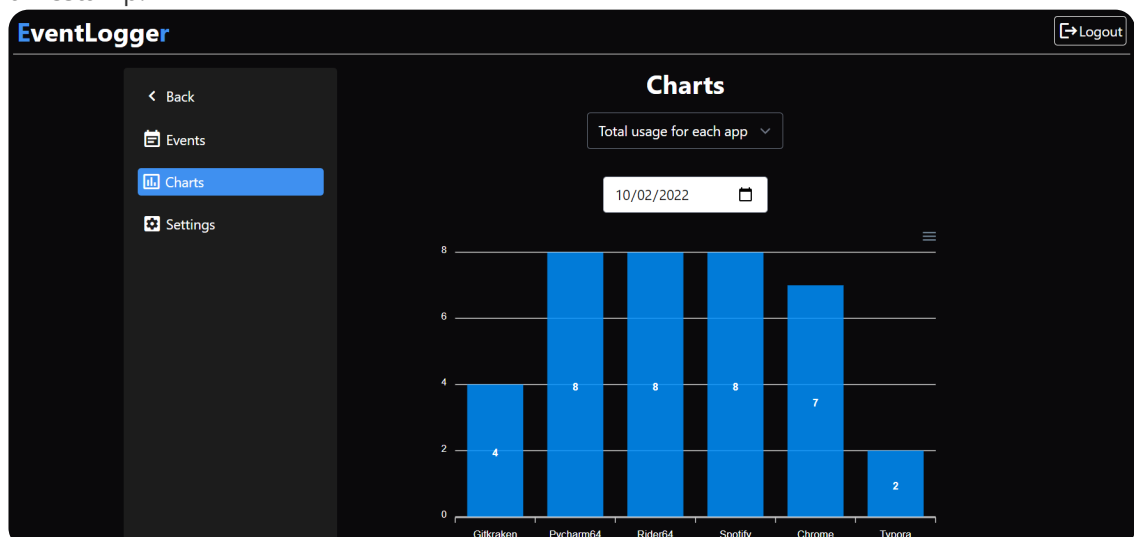
Chart view

All charts are useful to observe the victims behavior. The charts are:

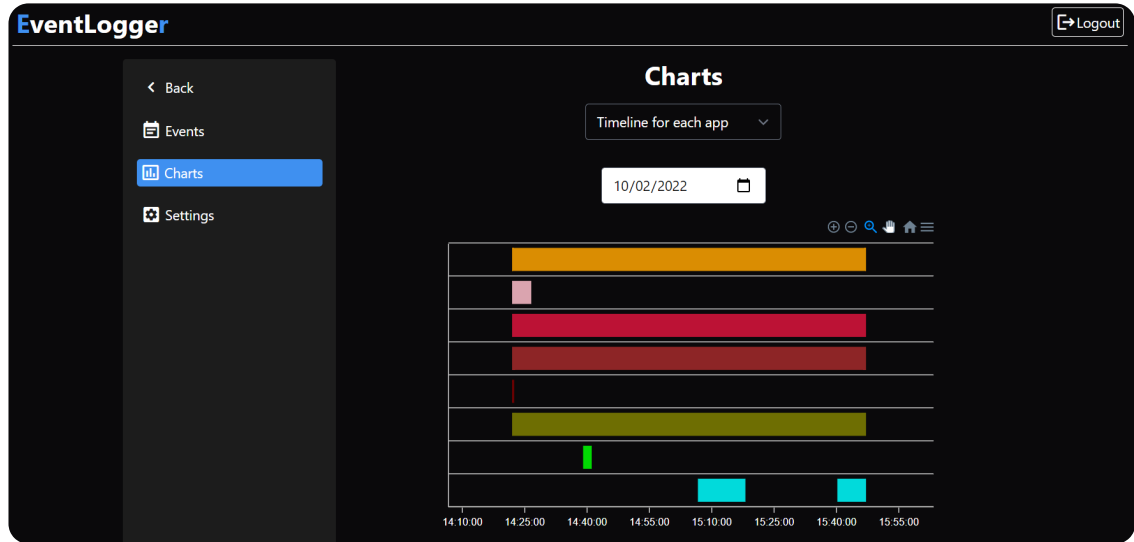
- **Total usage for each app:** the total usage of each app.



- **Total events per day:** all captured events in a day, grouped by app and sorted by timestamp.



- **Timeline for each app:** the graph shows the apps' usage timeline.



Dynamic settings

Here there are all the dynamic settings of the **EventLogger**. Already described in the previous section.

The screenshot shows the 'Settings' page in the EventLogger application. The left sidebar contains a navigation menu with 'Back', 'Events', 'Charts', and 'Settings' (selected). The main area is titled 'Settings' and displays four dynamic settings:

Setting Name	Value
LOG_PROCESS_ON_DOUBLE_CLICK	True
LOG_KEYBOARD_EVENTS	True
LOG_MOUSE_EVENTS	True
SECONDS_API_INVOKE	60

At the bottom of the settings list are two buttons: 'Annulla' and 'Salva'.