Beatrix: A Malicious Code Analysis Framework

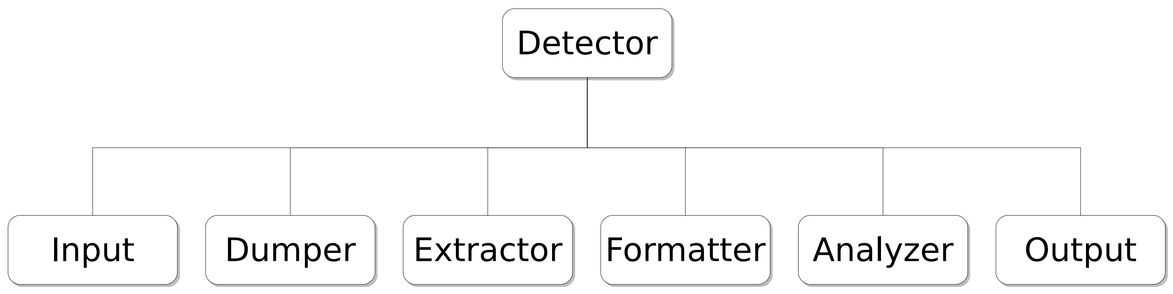
Christian Wressnegger (christian@wressnegger.info) **http://beatrix.sf.net**

Introduction

Especially in recent years malicious codes (malware) and the techniques used to bypass modern detection systems evolved at a tearing speed. We believe that in research a more advanced way of cooperation on malware detection is needed to bring forward security in general. We are working on bridging this gap and present **Beatrix** as an analysis framework. It allows researchers to focus on the development of new techniques instead of bothering about visualization, testing environments or the publication and distribution of the final prototypes. Especially simplifying the latter results in increased availability of state-of-the-art developments others may build on top of. For this purpose, we provide a software framework which introduces a plug-in infrastructure, that breaks down the detection process into disjoint sub-tasks. Hence, it is not only possible to utilize other prototypes but even single components of it without forcing a special license or demand open source implementations. The benefit for each individual developer arises from the reduced implementation effort for a complete prototype, since the framework already takes over significant parts for binary analyses.

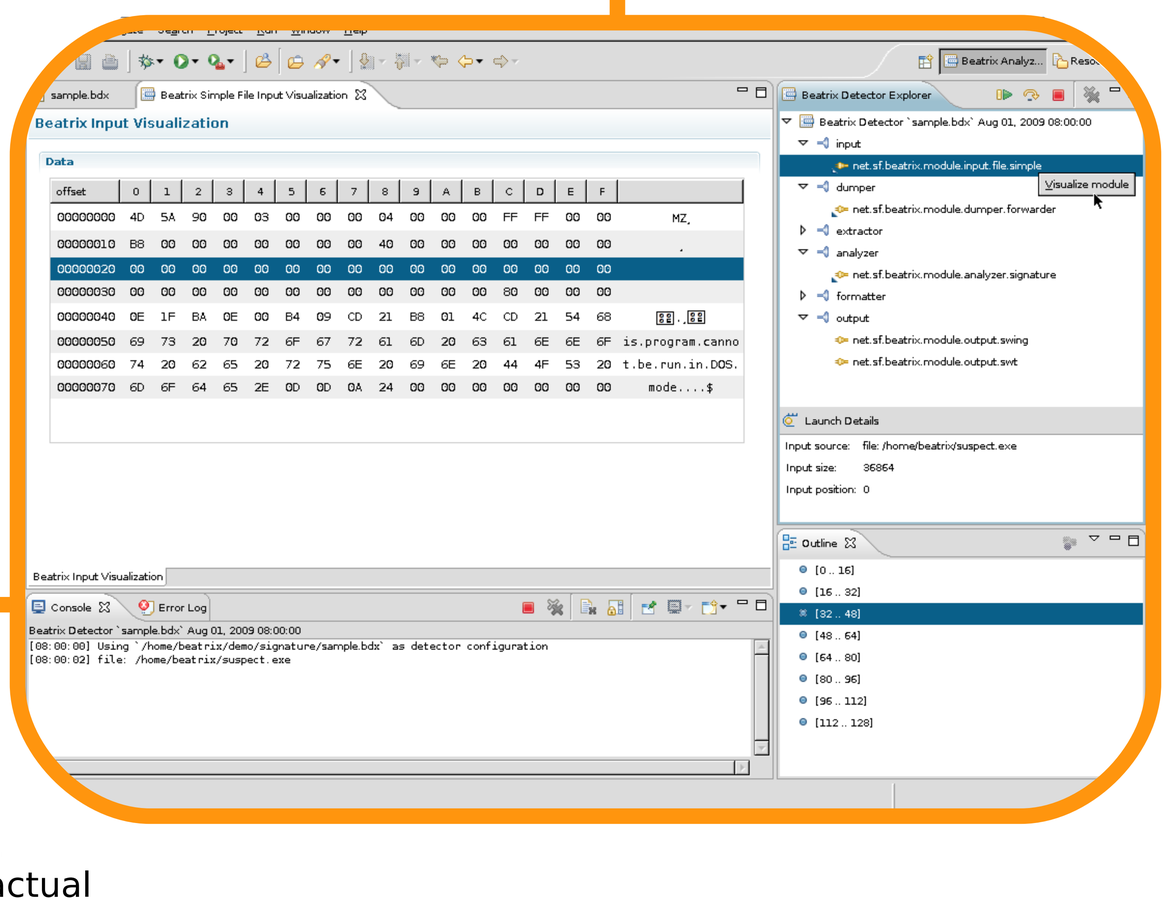
Structure

The Beatrix Framework splits up the task of malicious code detection into the six categories shown in the figure below. The internal processing scheme is as follows: we iterate over the set of Inputs and pass each piece of input one by one to the Dumper. The results of the Dumper are post-processed by a number of Extractors or left as they are if no Extractor is specified. Based on these results one or more Analyzers perform the actual classification, which again is optionally post-processed by a set of Formatters. The final results are handled by at least one Output module.




```
<detector id="net.sf.beatrix.detector.sample" name="Signature Sample">
  <modules>
    <item type="input" id="net.sf.beatrix.module.input.file.simple"/>
    <item type="dumper" id="net.sf.beatrix.module.dumper.forwarder"/>
    <item type="analyzer" id="net.sf.beatrix.module.analyzer.signature"/>
    <item type="output" id="net.sf.beatrix.module.output.swt"/>
  </modules>
  <preferences>
    <item name="input">
      <value>/home/beatrix/suspect.exe</value>
    </item>
    <item name="chunkSize">
      <value>128</value>
    </item>
    <item name="userDB">
      <value>/home/beatrix/PEiD/DB.txt</value>
    </item>
  </preferences>
</detector>
```

The configuration of the final detector is an XML file that declares the modules to use and their parameters. Since manipulating such configurations by hand is error-prone and especially memorizing single options and parameters is tedious support is provided in form of integrated graphical editors based on forms.



Tooling

When talking about **Beatrix** (Project) one distinguishes between two major components: the *Beatrix Detector* and the *Beatrix IDE*. Both are based on the *Beatrix Framework* which in turn makes use of *Beatrix Modules*.

The **Beatrix Detector** is a stand-alone application which performs the actual detection process based on a configuration file and its program arguments. This configuration file specifies the individual plug-ins (so-called Beatrix Modules) to use and contains their required parameters. Therefore, the Beatrix Detector represents the “operational implementation'' of the Beatrix Framework. The set of tools for using this infrastructure is collectively called the **Beatrix IDE**. It is intended to assist the developer in composing an individual detector (the mention configuration) from scratch. Furthermore, the IDE is able to start these detector configurations in a debugging mode (independent of the Beatrix Detector but sharing the same code base: the Beatrix Framework). This debugger allows one to inspect the data which are currently processed within each plug-in used (no matter if third-party or own developments) and navigate through their execution.