Design, Implementation & Evaluation of a Modular Network-Proxy Framework

Frederik Hauser
Hochschule der Medien, Stuttgart
SySS GmbH, Tübingen

Objective

Proxies are irreplaceable tools in performing security-audits of distributed applications: The line of communication needs to be interrupted to insert, delete and change network packets. Existing proxy-systems are mostly limited to a specific context of operation and don't provide the needed flexibility.

The modular proxy-framework should resolve those limitations in providing a flexible system for doing network packet manipulation in every layer of the TCP/IP-stack. To gain a full intervention, exchanged network packets must be handled by an User-Mode TCP/IP-stack instead of the operating-system's internal stack.

A proxy consists of one or more modules which encapsulate the functions and are implemented in an adequate programming language. The architecture should be engineered in an advanced designing- and modelling-process. A modular system based on XML-configurations should offer the possibility to compile proxy-structures, which are superior tools to analyze specific aspects of communicating systems. The modularity should provide a high reutilisation for existing modules.

The implementation should be evaluated through a concrete Man-in-the-middle attack on a virtualized Mail server-Client-scenario.

Features

- **Flexibility**
  The proxy-framework is able to operate on various layers in the *TCP/IP*-stack.
- **Reusability**
  Functions can be encapsulated in modules which are stored in a central repository.
- **Extensibility**
  New modules can be implemented easily in *Python* - all popular and powerful libraries can be used!
- **Compatibility**
  The proxy-framework can be executed on all *Python*-capable host operating systems.
- **Documentation**

Release

The thesis as well as the proxy-framework will be released under the *GNU Public License* until end of february. Please visit `www.fhauser.de` to download the sources.

Modules

To achieve a maximum way of flexibility in imlementing and useing functions, two types of modules provide powerful elements for creating individual proxy-configurations:

- **Simple Modules** represent actual functions which can be implemented in *Python* or executed within a *Python* wrapper

SimpleModules are described by a *XML* file ...

```
<module type="simple">
  <name>HttpProxy</name>
  <author>Frederik Hauser</author>
  <src>httpProxy.py</src>
  <shortDesc>Provides a HTTP-Proxy</shortDesc>
  <longDesc>httpProxy.md</longDesc>

  <requirements>
    <requirement name="socket" version="python2.7" />
    <requirement name="asyncore" version="python2.7" />
  </requirements>

  <ports>
    <output id="http_proxy_out" desc="Output...." />
  </ports>

  <config>
    <param name="port" req="True"/>
    <param name="host" req="False" default="0.0.0.0" />
  </config>
</module>
```

... and implemented in *Python*

```
from proxyframework.core import SimpleModule

class HttpProxy(SimpleModule):
    ...

    def http_request(self, request):
        req_parsed = http.HTTPRequest.build(request)
        self.send("http_proxy_out", req_parsed)
        ...
```

- **Compound Modules** encapsulate different simple modules, providing a construct to perform a particular task

Include *Simple Modules* or other *Compound Modules*...

```
...
<module type="simple" repo="gather/http/httpProxy">
  <config>
    <param name="port" value="8080"/>
  </config>
 </module>
...
```

- A **Proxy** module is an extended *Compound Module* built out of a *XML* configuration

The proxy can be started by calling a console-script:

```
root@pentest: pf-starter --config medianight-demo.xml
```