Initial approach Report: Onion Module

Marko Dorfhuber (03658730) ✉marko.dorfhuber@tum.de

Christoph Rudolf (03662116) ✉christoph.rudolf@tum.de

1. Introduction

Our team, team 17, consists of the two students *Marko Dorfhuber* and *Christoph Rudolf*. As part of the anonymous and unobservable VoIP application for the P2PSEC lecture in 2017, we are going to develop the module concerned with the establishment of *onion tunnels*.

2. Choice of programming language

Based on the previous experience of all team members, we chose Java as our programming language. Due to the platform independent nature of Java, any operating system is supported. However, as the development will happen on Linux-based machines, this is the operating system we officially support.

Taking the experience of both team members into account, Python would have been a possible alternative. Java is preferred as it provides more sophisticated build systems for larger software applications, as well as a wide variety of available frameworks for testing, mocking and techniques like dependency injection. In regard to their capabilities with network programming, we see both high-level languages as equal, with Java having the edge in terms of speed.

3. Build system

As build system we chose *Maven*. The advantages of Maven are the existing default integration into the popular IDEs *Eclipse*, *NetBeans* and *IntelliJ IDEA*. In contrast to alternatives like *Ant* and *Gradle*, both team members have some experience in using Maven. It is also worth noting that after the first brainstorming on the concept laid out in the specification, we do not except a large amount of external libraries to be needed for the Onion module. *Maven*'s contribution will largely be the integration of dependencies relevant for testing like *JUnit*, *EasyMock* and possibly a dependency injection framework like *Guice*.

4. Quality Assurance

In order to insure the functionality and quality of our code to meet the desired standards, we intend to use unit tests with *JUnit*. As the Onion module relies on two adjacent modules that have to be mocked, we are either using *EasyMock* to mock these modules for testing or, is provided, rely on a reference implementation from a previous year.

Additionally, by using *IntelliJ IDEA*, we can ensure quality in terms of code by enforcing *Javadoc* comments for all public class members and further static code analysis.

5. Available libraries

Java supports a wide variety of network communication mechanisms out-of-the-box. In order to be scalable to a possibly large number of users, we aim to include the *Netty* framework to handle networking.

Additionally libraries for testing purposes like *JUnit* for unit tests, *EasyMock* for mock objects and *Google Guice* for dependency injection have already been mentioned. In order to provide logging which also helps to find bugs and maintain the maintainability of the system, we include a logging mechanism with *Log4J*.

We currently assume that the handling of RSA keys is completely encapsulated in the module regarding Onion Authentication. If it turns out that the Onion Forwarding module has to read or convert different key formats, we will rely on the *Bouncy Castle* API.

6. License

In order to decide for the license to use we first have to confirm whether or not we are limited by the libraries we intend to use. An overview over the libraries we've chosen so far is listed Table 1

Table 1: Overview over all libraries' licenses

Library	**License**
Maven	Apache License 2.0
JUnit	Eclipse Public License 1.0
EasyMock	Apache License 2.0
Google Guice	Apache License 2.0
Netty	Apache License 2.0
Log4J	Apache License 2.0
Bouncy Castle	MIT X11 License

As both, the *Apache License 2.0* and the *MIT X11 License* are compatible with the *GNU General Public License*[1], we are free to chose any of these three licenses. *JUnit* being licensed under the *Eclipse Public License 1.0* is not an issue as the library is not required for running the final application and we do not ship the source code of it. Taking all these licenses into consideration, we decide for our code to be released under the ***Apache License 2.0***, as it is very permissive and allows open and commercial use.

7. Previous experience of the team members

Marko (6th semester B.Sc. Informatics): Java was used as programming language for assignments for different courses. In addition, he used Java as the programming language for the practical project for the "Rechnerarchitektur" course in the second semester. Based on the assignments of GRNVS and the rootkit programming practical course visited last semester, he has experience in network and low-level programming using C.

[1] https://www.gnu.org/licenses/license-list.en.html#GPLCompatibleLicenses

Christoph (6th semester B.Sc. Informatics): His programming experience is mostly based on working with technologies provided by Microsoft's .NET framework. Here he has mainly worked with C# for desktop applications and web applications in ASP.NET. Java experience stems from using it repeatedly for course assignments since entering university. Network related programming with Java and C was done for the assignments in the introduction lecture on networks (GRNVS). Some additional knowledge in Python is present due to working with it for the bachelor's thesis.

8. Workload distribution

Due to the large amount of API calls from either the CM/UI or the Onion Authentication module and additional packets from the P2P protocol, a large amount of code for the Onion Forwarding module will be the parsing of traffic and ensuring its validity. This can be done separately from a second layer/submodule that relies on valid packets and processes them accordingly to the functionality provided by the Onion Forwarding module. These two submodules could serve as a separation of work. If we notice that the effort is not evenly distributed, rebalancing work between the team partners can be done accordingly.

9. Issues and complains

- The specification introduces the notion of a *round*. We currently struggle to understand which module ensures the synchronization of rounds over modules and even peers, or if this isn't needed at all.
- Are we correct in understanding that for each round one new tunnel is created over at least two hops. This tunnel is either for cover traffic or a legitimate call. As the caller choses all intermediate hops, he will know about the complete chain of hops to the destination.
- The specification says that "*Tunnels created in one period should be torn down and rebuilt for the next period.*". Are these teardowns issued by the CM/UI module? If not, how is the Onion module notified of a new round (alluding back to the first question)?