WeSee: dynamic visualization of Web Service use

Sergey Podanev
Advisor: Miguel L. Pardal
Seminar Future Internet WS2017/2018
Chair of Network Architectures and Services
Departments of Informatics, Technical University of Munich
Email: sergey.podanev@tum.de

ABSTRACT

The software applications that we use in our daily lives are typically not isolated when running in our phone or computer. In fact, they may connect to remote resources through REST or SOAP Web Services. Many times, programmers are not aware of this network activity, as the activity is done by libraries used by the application. The end-users are even more unaware of this reality. Usually these remote invocations are benign, but there is still a problem of transparency. To address this concern, we present *WeSee*, a tool designed to intercept network connections, display them in a graphical manner and, through this data visualization, make visible what is hidden. The goal is to raise the awareness of developers and end-users about the network interactions of the applications that they use.

This paper describes the technical details of the *WeSee* tool implementation, available as open-source [1]. The prototype is functional and can be used to capture and display network traffic in a single device. The architecture is extensible and, in the future, the system can be used for intercepting different types of network payloads and for monitoring multiple devices at the same time.

Keywords

Data visualization, Network Monitoring, Packet Interception, Privacy-awareness.

1. INTRODUCTION

Today, we cannot imagine our life without using modern information technologies. In the origins of computer science, programs ran on a single device and were relatively simple, as the machines did not have much computing resources and the code base was moderate in size. All variables were traceable and the program behavior could be verified by the programmer.

Since those early days, software has grown significantly in size, and in complexity [2] [3]. Modern programs use many layers of code, and programmers are divided into special fields. The full program algorithm is no longer transparent, even for experienced programmers. Debuggers and other tools can be used to follow the flow of execution, but this task is too complex and time-consuming.

Programs are composed of many libraries that are combined to produce the desired functionality. The use of third-party libraries is essential to save time and benefit from external expertise. But such use poses risks [4].

The issue is not just the use of libraries. These libraries can access the Internet and communicate with remote servers. Programmers are potentially unaware of the data being transmitted. Consider the following example: to display a map in the application, the developer is using a well-known maps library. To retrieve a map, location coordinates are shared with the service. Additionally, other data can be sent to other remote servers. For the developer, conceptually, the program is "just" showing a map, but, in reality, it is also sending usage data to a third party. This is a significant privacy concern.

1.1 Application monitoring

The first approach to monitor applications is to monitor network connections and select suspicious ones. There are "sniffing" tools such as *Network Inventory* and *TcpDump* in the operating system. *Network inventory* is a tool, with a GUI interface used to collect all network data from connected devices and get all operating system and device statistics [10]. *TcpDump* is a TCP/IP packet analyzer that runs from the command line [11]. *Wireshark* can be a good alternative for monitoring connections. It shows standardized values of packet fields with particular values (given in text and byte form). However, *Wireshark* cannot be adapted for other graphical representations (e.g. graphs) and cannot be directly used for "sniffing" particular parts of the code [9].

The other possible solution of handling suspicious code is to "jail" applications. There are several projects which isolate applications, for example *Firejail* [6] or *Linux containers* [7]. In addition, firewalls [8] can also be used to contain network traffic. Unfortunately, people cannot manually supervise all of connections, therefore it is difficult to manually set all necessary filters into the firewall without proper analysis. All applications can be "jailed". But, in this case, the restriction policy for each of the applications should be applied accurately. Sometimes, it is required to create connections with new destinations, but these new destinations might have been banned by a user previously. The content of the connections can be encrypted and not properly described by the general solution. The observed tools have a general approach for each technology and do not allow centralized monitoring of a network at different OSI model[1] levels from different devices.

[1]Open system Interconnection

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
45
doi: 10.2313/NET-2018-03-1_07

Table 1: Requirements list

#	Requirement
R1	Capture network payloads
R2	Store the payloads persistently
R3	Display the payloads in a graph inspired notation, where nodes are devices, and edges are messages between devices
R4	Allow the filtering and selection of data to display
R5	Allow inspection of captured message details
R6	Allow diverse types of payloads to be captured, at different levels in the OSI model (link, network, transport, application)
R7	Provide a user interface aimed primarily at software developers, but that can also be used by end-users.

If any of these tools lead to suspicion that something is wrong, then it is time to reconfigure and use firewalls and other security tools, such as an anti-virus, malware detectors, intrusion detection systems, etc.

1.2 Network visualization

The main idea of our approach is to use *visualization* of data from passive monitoring of selected parts of the network, to capture messages, and to present these messages in a way that can be understood by humans. The goal is to show network activity and, through this means, to raise the awareness of the users about the communication dependencies of the applications.

In this paper we present *WeSee*, a tool designed to meet the requirements stated in Table 1. The system designed is presented in detail in the next section.

2. SOLUTION

Intercepting communications with a universal interface allows collecting useful data from different sources with central node. Based on this data, statistics of network nodes and connections can be calculated. And finally, a graph is used to illustrate processed information. The listed functionality was reached by designing the next general deployment structure.

The *WeSee* tool performs the following next tasks:

- Intercept communications with a universal interface;
- Gather and calculate overall simple statistics;
- Display connections with a coherent graph.

2.1 Design

The design of *WeSee* is shown in Figure 1.

The `intercepted device` is concerned with the data interception and capture near the target application. To intercept new connections, an `interceptor` module is integrated inside the application to log it.

The role of the `Visualization server` is to collect and process data, collected from interceptors. It is implemented as

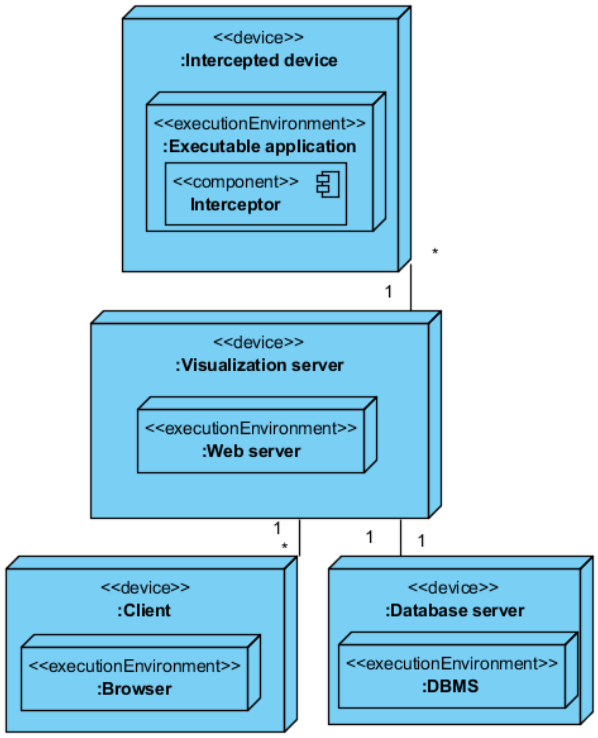

Figure 1: WeSee deployment diagram

a web server. A web server is most suitable for that purpose as it can serve multiple clients and provide a universal graphical interface as a web page, which is accessible from most modern devices.

The `database server` helps the web server to collect and process data with concurrent access.

A `browser` can be used to visualize the data, as it is supported by almost all computer platforms. Thereby, a user can monitor remotely and from multiple devices. When a user loads a web page, the browser downloads all requested data from the web and then draws a suitable graph.

The general algorithm is illustrated in Figure 2. Data is stored into a database and can be later collected, filtered and converted for presentation with statistics.

2.2 Implementation

The implementation of the *WeSee* tool consisted of several steps. At first, it was necessary to select suitable technologies to implement it, define program functional possibilities and therefore to narrow possible architecture solutions. As the tool consists of the several deployment parts, on the second step it is important to define the general architecture to connect these parts into the tool. Lastly, every program module can be implemented separately according to the general architecture.

2.2.1 Selected technologies

WeSee was implemented in Java, a high level programming language, which speeds up development, is platform inde-

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
46
doi: 10.2313/NET-2018-03-1_07

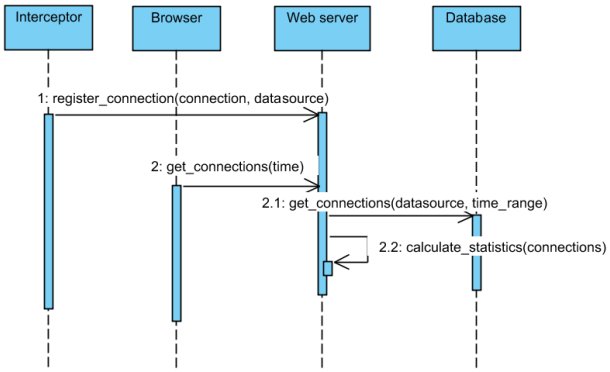

Figure 2: WeSee general sequence diagram

pendent, reliable, sufficiently optimized for the project task and maintainable due to its high popularity and longtime presence [12].

The connection interception is done by each monitored application individually. For instance, it can be implemented directly into the code of the target application.

To send the data, REST services are used. REST services use uniform stateless interfaces to register new connections [13]. Also, the REST service is used for requesting all occurrences for the selected connection in a given time range.

The alternative is SOAP services, however, currently SOAP has a downward popularity trend, because it is more complex to configurate [14].

To develop the web server, the *Spring Boot* framework was selected. It is a framework that provides much functionality, including REST services, and database connections support [15]. To connect to the database, *Hibernate* technology was used. *Hibernate* maps database tables to Java objects automatically, simplifing the database usage [16]. The underlying database was *MySQL*. It is a database with a long history, which obtained a rich support of connection technologies, including *Hibernate*. It provides a multiple thread access mode, has sufficient performance and is supplied with a community edition. As already mentioned the database can be replaced with any other as the result of using the *Hibernate* technology [17].

To represent the data in a visual way, the *D3* library was used [18]. *D3* (data-driven documents) has a rich interface for creating graphs on the SVG HTML element. This library provides powerful components to create dynamic graphs with a force simulation, which makes the visualization interactive. To draw the network graph, a force-directed graph is included in the *D3* library. A force-directed graph is the most common way to draw a network topology. It is a weighted graph with sets of vertices connected by edges. The edges as well as vertices can have their own number(weight). These weights are used to illustrate the number of messages that were transmitted over a connection or a network node.

This idea to use this graph was inspired by the Gource project, where an animation of contributions to a Git repository can be generated [20]. The other good example is vizceral project [21]. It builds an orientated graph of network traffic volume.

2.2.2 *Persistence*

The data layer is shown in Figure 3.

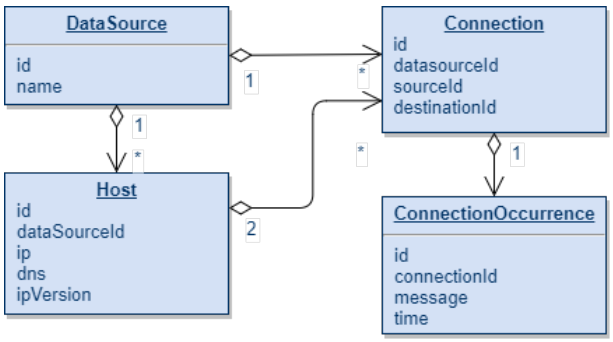

Figure 3: Database schema (relational model)

The database stores the general information about occurred connections. But connections can be received from different interception sources and at the same time they are described for the same network hosts. To represent connections on the graph, the collected information should be merged from the selection of the multiple data sources.

2.2.3 *Interception*

In order to intercept the data, a universal interface should be provided for each intercepted application. This interface sends the data on a REST service, which can then store the data on the server. This interface is represented in Figure 4. It contains an abstract interface with its concrete implementation and a set of data definitions to describe intercepted connections.

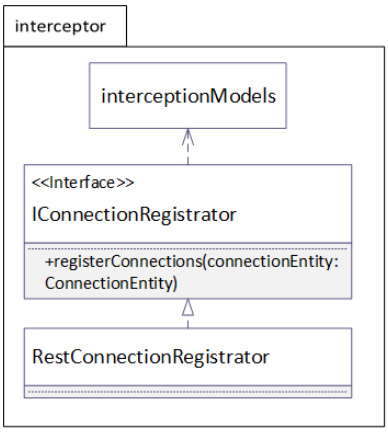

Figure 4: Interception module structure

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
47
doi: 10.2313/NET-2018-03-1_07

2.2.4 *Pcap interception*

In order to have a rich functional test and present a tool use case, a `pcapInterceptor` was developed as implementation of the `interception` module.

To intercept a connection, it uses the *pcap* library. *Pcap* is distributed for multiple operation systems platforms including Windows and Linux under different names [19]. The library "sniffs" all selected interfaces and finds network packets with a configured filter.

We had to decide at which level to capture network activity. Different levels have different advantages and disadvantages. The lower network levels provide more information, but usually there is more encoded data, which is not easily understandable. The higher network levels have more concise data but do not have identification of hosts and other details. The network layer contains necessary information for creating a network graph, which includes especially IP addresses, and even information from higher levels, for example, of the transport or application layer. Therefore, network layer is the most suitable for recording connection metadata. If the higher network packets can be extracted by the *pcap* library, then `pcapInterceptor` the packet payload from these levels is recorded as a sent message. As a result, the pcap library provides a rich data set for visualization even for one computer. The structure of the `pcapInterceptor` application is illustrated in Figure 5.

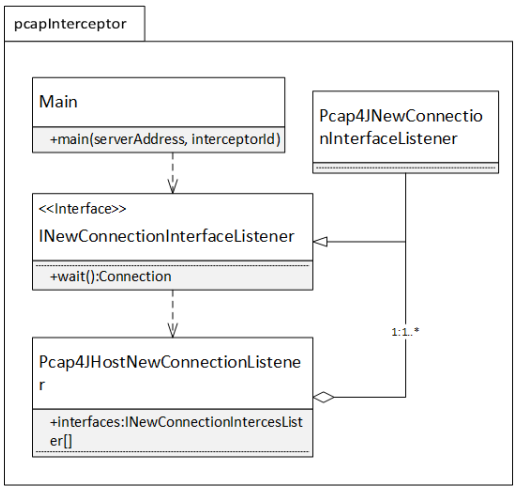

Figure 5: PcapInterceptor application class diagram

The main function is used to start the application, which repeatedly catches packets from sniffed interfaces in different threads. The `Pcap4JNewConnectionInterfaceListener` interface allows to refer one to a computer listener as a united object of combination of individual interface listeners. In case the volume of the packets is too large to report to the server, they are skipped. As the application works in an infinite loop, it refreshes interfaces periodically as shown in Figure 6.

2.2.5 *Visualization server*

The `visualization server` is the central node of the tool as it implements the functionality of gathering, analyzing and representing the data. Similarly to the logical division,

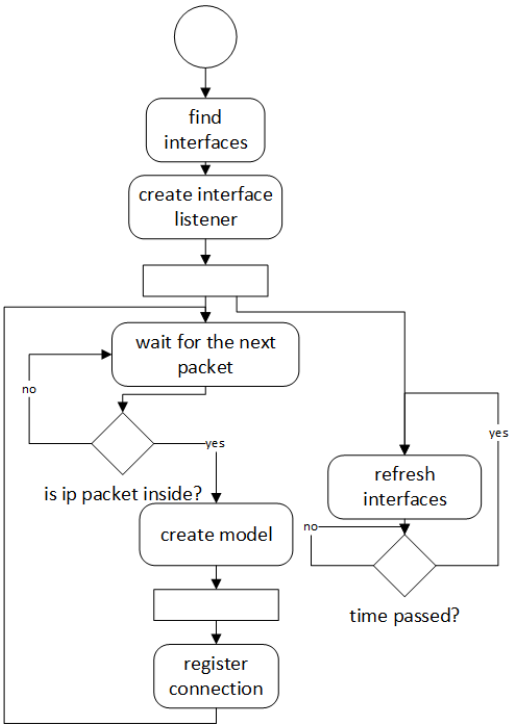

Figure 6: PcapInterceptor behaviour

WeSee data definitions were separated into a layered architecture according to Figure 7).

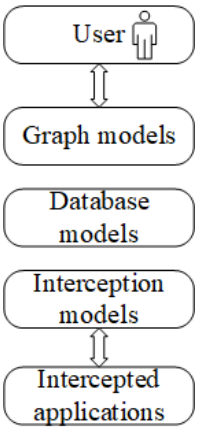

Figure 7: Visualization server layering

Intercepted applications send the data in a JSON format, which is different from the format stored in the database. Both data structures contain information about the hosts participating in the communication and the message payload. Applications which send the JSON data do not excessively put with information about themselves and send plain information about connections. In the database all received data is stored with data sources (application identifications) and indexed.

Graph layer data structures are a representation of the statistics, since the information about nodes from different data

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
48
doi: 10.2313/NET-2018-03-1_07

sources should be merged to present the composite picture of the network. The server back-end consists of 3 modules: `srv.rest`, `graph` and `repository`, depicted in Figure 8.

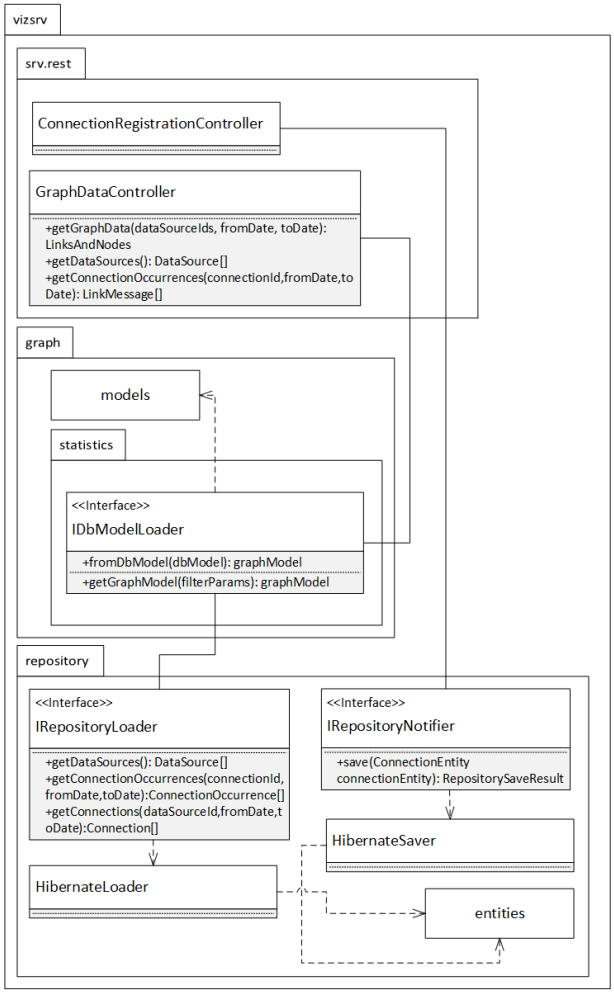

Figure 8: Visualization server class diagram

The `srv.rest` module consists of Spring REST service implementations. They interact with a browser to provide the data with asynchronous requests (for example, in case of filtering by the date/time and data sources) and collect the data from intercepted applications.

`Graph` and `repository` modules work with the graph and database data definitions respectively and convert the data to the proper form for controllers. The `database` module is based on the *Hibernate* technology. The application database connector enables to use *Hibernate* and contains configuration to connect the database. In addition to the connector, the database with a user account was created. That database user is delegated to the `visualization server` application. The information about database is placed in the *Hibernate* configuration file of the `visualization server`.

2.2.6 *Presentation*

To present the data, HTML was used. When the page is loaded the client makes requests to the server. Generally, the web page plays the role of a single-page application due to the narrow specialization use.

To configure a node spacing on the graph several forces were created to push unconnected nodes or magnify nodes keeping the link distance. The width of the links and the radius of the nodes scales with the power function. This limits the maximum size, but shows an object which can be scaled, depending on the object activity. The client-side application is written in JS (JavaScript) and consist of several parts represented in Figure 9.

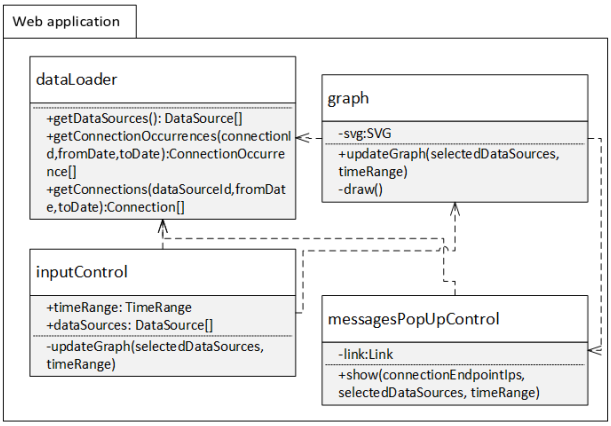

Figure 9: Web application structure main objects

The `dataLoader` is a JS object and is responsible for downloading data from the web server with requests to the server. Downloaded data can then be filtered with local search parameters and reorganized to fulfill the D3 library data structure requirement. The D3 library defines the special format of nodes and links for the force-directed graph. Graph control uses the SVG HTML element to draw vector graphic elements. The `InputControl` object generates possible filtering ranges and acts on entering new filter parameters for the data selection. It controls the limits of the possible data range for selected data sources and refreshes the graph on update. `MessagesPopUpControl` shows a pop up window with detailed connection occurrences. The user can list particular messages transmitted over a given time from selected data sources. To provide an opportunity of filtering and ordering data, the datatable JS library was used [22]. It allows a user to search by a plain string search or reorder rows by the columns: time, data source or message text.

3. WESEE IN ACTION

This section presents an example of the *WeSee* tool using the `pcapInterceptor`.

Figure 10 shows the start page. A user is able to examine the start page with the full selected data range by default. Then, it is possible to narrow the search results by selecting the names of data sources or pick the time range. The graph consist of nodes that represent network interfaces of the devices; and links show connections between them. On hovering, a pop-over message shows brief information about the host: IP, usage statistics, DNS and link information such as the number of messages host sent and received, last message, end-point addresses. The next steps show one way of

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
49
doi: 10.2313/NET-2018-03-1_07

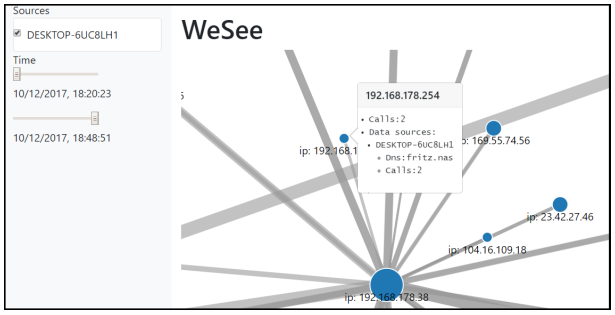

Figure 10: WeSee network graph screenshot

using the tool and demonstrates the functionality, stated in the requirements.

With the selected time range from the source "DESKTOP-6UC8LH1" it was discovered that a computer sent two messages to the DNS "fritz.nas", which actually is the name of a router (Figure 10). This information can be used for node identification and for exploring its activity.

On double clicking the link, a pop up window appears where all intercepted messages are listed with information of the date/time occurrence, name of interceptor, brief and full text of the sent message, as shown in Figure 11. It was

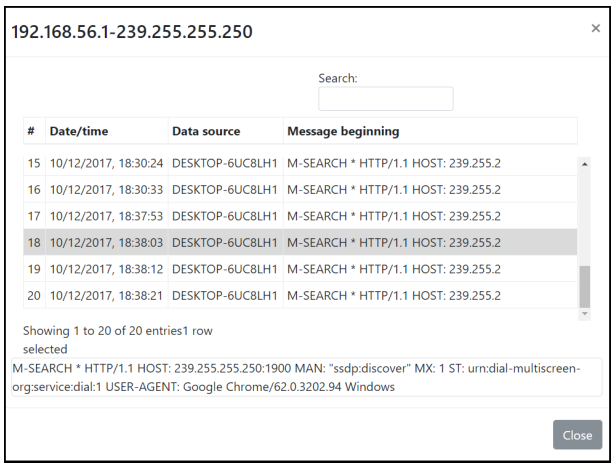

Figure 11: Detailed information about connection

discovered, that there is a link to the multicast IP address 239.255.255.250 and it primarily contains SSDP (Simple Service Discovery Protocol) messages. SSDP is used for searching UPnP devices in the home network [23]. The Google Chrome browser performed this action automatically after opening. Although this connection was initiated by an authorized application with a standardized protocol, the *WeSee* tool also can detect other hidden connections and depict it on the graph. Observing this graph, the awareness is raised, as it shows the entire network with elementary statistics, to which the computer is connected and sent messages within the given period of time.

Table 2: Requirements assessment.

#	✓/x	Comments
R1	✓	Network payloads are captured in `pcapInterceptor` for network level and higher.
R2	✓	Information is stored in a database
R3	✓	Implemented orientated graph with D3 library with nodes as hosts and edges as connections.
R4	✓	Filtering by data sources and time range. For selected connection: reordering messages, search by plain text.
R5	✓	Provided message details: occurrence time, data source, short and full message text.
R6	✓	`pcapInterceptor` captures payloads of network level and higher. The API allows to collect information of any layer, but should be converted to the defined data structures.
R7	✓	Provides user interface as a web page.

4. CONCLUSION

After creating a working prototype of the *WeSee* tool, it is necessary to check fulfilled requirements from the Table 1. As it can be seen from the Table 2, all requirements were reached as discussed in the comments column.

The *WeSee* example demonstrated, that even a single device can participate in a large number of network connections. As more interceptor implementations become available, for different network levels and protocols, and for different devices, the tool has potential to make visible even more interesting pictures of application activity.

The project is open-source and can be used by any programmer, who is concerned about network security and wants to reveal the "secret life" of his/her own devices [1].

In the author's opinion, with the trend of increasing heterogeneity in programming environment, *WeSee* will only become more relevant.

4.1 Future deployments

The following scenarios show how *WeSee* can be used in different deployments, to provide visibility of network activity.

4.1.1 SOAP web service

This is a specific example of a different kind of payload, in this case, at the application level. SOAP messages can be used for web services [14]. However, it is also necessary to track SOAP communication, as the information can be sent to the unknown web server or content of the message can be suspicious. The SOAP messages are more high-level then the packets, and, as a result, are easier to analyze. To integrate, a SOAP message listener can connect to the "interceptor" module, fill "interceptor" data structures and send collected data with this interface. It is also possible to create a specialized interceptor, that can be connected by adding a few lines to the standardized server configuration file.

In this deployment also, multiple service invocations can be captured and sent to the same visualization server, to pro-

Seminars FI / IITM WS 17/18,
Network Architectures and Services, March 2018
50
doi: 10.2313/NET-2018-03-1_07

vide an overview of the interactions between a set of user services.

4.1.2 IoT gateway

The lack of visibility of network activity can be further aggravated in the Internet of Things (IoT). These devices show a great potential with their ability to exchange information about environment and use it for interaction [26].

In the IoT, the programs are running on devices not usually considered as computers, like light bulbs or door bells. There is also no visibility on the network activities of all these devices. They can share their information not only with an authorized server, but with someone else too. Although, mostly these information is not security critical (for example, the temperature in home), but can indirectly harm too.

IoT software can be modified, but not if it is embedded in devices. The approach is redirecting the traffic from the used gateway. Typically, private networks, which usually used for IoT, have gateways to the external network. Modern routers support a traffic redirection feature. After that, a redirected traffic can be analyzed by an interceptor application on other computer and the result can be collected by a central *WeSee* server. However, this method can be problematically deployed for the network with many gateways. In this instance, it is required to configure multiple gateways and keep their configuration up-to-date.

4.2 Future improvements

The current state of the tool fulfills its main requirements. However, the project can be extended to improve the user experience and provide more functionality. Currently, the content of network packets is not decrypted, but unencrypted information is avaliable. It is done due to the wide specialization of the current interceptor in order to demonstrate tool possibilities. It can be fixed by implementing a decrypting module on the interceptor side or including build-in decryption into the server.

The other way is to use specialized interceptors, for example, which intercept REST service messages for the selected path.

Additionally, a node's geolocation identification can be added into the tool. The geolocation identification potentially raises the user's awareness about a host location as an object connected with "real" world places, not abstract IP addresses.

With the inclusion of extra search parameters, the web application can be built using one of the single-page frameworks, like AngularJs or OpenUI5 [24][25]. The *WebSee* web application already works as a single-page application. Currently, the amount of JS code is small, but on future releases the structure can be better organized with use of existing reliable frameworks. They can improve responsiveness and maintainability of the front-end side of the application.

We hope that, in the future, more use of *WeSee* is made, and that a *gallery* of interceptors and visualizations can be built.

WeSee also has potential as a learning tool, for example, as part of advanced and interactive teaching methods [27].

5. REFERENCES

[1] *Github repository: WeSee: dynamic visualization of Web Service use, last access: 21.01.2018*, `https://github.com/inesc-id/WeSee`

[2] Amit Deshpande, Dirk Riehle: *The Total Growth of Open Source*, Open Source Development, Communities and Quality. OSS 2008. IFIP – International Federation for Information Processing, vol 275. Springer, Boston, MA, 2008

[3] Barry Boehm: *A View of 20th and 21st Century Software Engineering*, Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, May 20-28, 2006

[4] Steven Raemaekers, Arie van Deursen, Joost Visser: *Exploring Risks in the Usage of Third-Party Libraries*, Software Improvement Group, Delft University of Technology, Amsterdam, The Netherlands, 2015

[5] *Global Internet Report 2016, Internet society, pages 31-34, 2016, last access: 21.01.2018*, `https://www.internetsociety.org/globalinternetreport/2016/wp-content/uploads/2016/11/ISOC_GIR_2016-v1.pdf`

[6] *Firejail security sandbox, last access: 21.01.2018*, `https://firejail.wordpress.com`

[7] *Linux containers, last access: 21.01.2018*, `https://linuxcontainers.org`

[8] *Internet Firewalls: Frequently Asked Questions, last access: 21.01.2018*, `http://www.faqs.org/faqs/firewalls-faq/`

[9] *Wireshark, last access: 21.01.2018*, `https://www.wireshark.org`

[10] *Network inventory advisor, last access: 21.01.2018*, `https://www.network-inventory-advisor.com`

[11] *TCP dump, last access: 21.01.2018*, `https://www.tcpdump.org/`

[12] *Java compared with other languages, last access: 21.01.2018*, `https://www.safaribooksonline.com/library/view/learning-java-4th/9781449372477/ch01s03.html`

[13] *REST (representational state transfer), last access: 21.01.2018*, `http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer`

[14] Pavan Kumar: *On the Design of Web Services: SOAP vs REST*. UNF Theses and Dissertations.138, pages 58-61, University of North Florida, USA, 2011

[15] *Spring framework, last access: 21.01.2018*, `https://spring.io`

[16] *Hibernate, last access: 21.01.2018*, `http://hibernate.org`

[17] *MySql open source database, last access: 21.01.2018*, `https://www.mysql.com`

[18] *Data-driven documents library, last access: 21.01.2018*, `https://d3js.org`

[19] *Pcap - Java library for capturing, crafting, and sending packets, last access: 21.01.2018*, `https://www.pcap4j.org`

[20] *Gource, software version control visualization, last access: 21.01.2018*, `http://gource.io`

doi: 10.2313/NET-2018-03-1_07

[21] *Vizceral, WebGL visualization for displaying animated traffic graphs, last access: 2.02.2018*, `https://github.com/Netflix/vizceral`

[22] *DataTables, Table plug-in for jQuery, last access: 2.02.2018*, `https://datatables.net/`

[23] *Simple Service Discovery Protocol/1.0, last access: 21.01.2018*, `https://tools.ietf.org/html/draft-cai-ssdp-v1-03`

[24] *Angular js framework, last access: 21.01.2018*, `https://angular.io`

[25] *OpenUI5 - open source js UI library, last access: 21.01.2018*, `http://openui5.org/`

[26] Dieter Uckelmann, Mark Harrison, Florian Michahelles Florian: *An architectural approach towards the future Internet of Things*, Architecting the Internet of Things, pages 1-24, Springer, 2011|

[27] Marc-Oliver Pahl: *The iLab Concept: Making Teaching Better, at Scale*, IEEE Communications Magazine, vol. 55, no. 11, pp. 178-185, November 2017

doi: 10.2313/NET-2018-03-1_07